

# FUJITSU SEMICONDUCTOR HARDWARE MANUAL

## **MB87J2120 & MB87P2020 -A Lavender & Jasmine**

---

### **Colour LCD/CRT/TV Controller Specification**

**FUJITSU**

Fujitsu Microelectronics Europe GmbH  
European MCU Design Centre (EMDC)  
Am Siebenstein 6-10  
D-63303 Dreieich-Buchschlag  
Germany

Version: 1.7

File: MB87P2020.fm



# Revision History

Version	Date	Remark
0.8	05. Apr. 2001	First Release
0.9	27. Apr. 2001	Preliminary Release
1.0	29. Jun. 2001	Overview Section and Register List reviewed SDC, PP, AAF, DIPA and ULB descriptions reviewed
1.1	20. Jul. 2001	Register List improved, Lavender pinning added, overall review
1.2	02. Aug. 2001	APLL spec included (CU) Review: overview, functional descriptions, register/command lists Preliminary AC Spec for Jasmine
1.3	05. Oct. 2001	AC Spec for both devices, Lavender added/Jasmine reviewed Two pinning lists - sorted by name/pin number ULB DMA limit description (DMA FIFO limits vs. IPA block size) SDC Register description reviewed
1.4	11. Oct 2001	Clarified AC Spec output characteristics (20/50pF conditions)
1.5	27. Mar 2002	Pinning and additional registers for MB87P2020-A added Design description for changes in MB87P2020-A added AC Spec updated for MB87P2020-A
1.6	05. Apr 2002	AC Spec for MB87J2120 updated
1.7	22. Jul 2002	Description changed to MB87P2020-A

File: */usr/home/msed/gdc\_dram/doc/manual/MB87P2020.fm*

**Copyright** © 2001 by Fujitsu Microelectronics Europe GmbH  
European MCU Design Centre (EMDC)  
Am Siebenstein 6-10  
D-63303 Dreieich-Buchschlag  
Germany

This document contains information considered proprietary by the publisher. No part of this document may be copied, or reproduced in any form or by any means, or transferred to any third party without the prior written consent of the publisher. The document is subject to change without prior notice.

# Table of Contents

## *PART A - Lavender and Jasmine Overview*

<b>1 Overview</b>	<b>15</b>
1.1 Application overview	15
1.2 Jasmine/Lavender Block Diagram	15
<b>2 Features and Functions</b>	<b>19</b>
<b>3 Clock supply and generation</b>	<b>21</b>
<b>4 Register and Command Overview</b>	<b>23</b>
4.1 Register Overview	23
4.2 Command Overview	23

## *PART B - Functional Descriptions*

<b>B-1 Clock Unit (CU)</b>	<b>27</b>
<b>1 Functional Description</b>	<b>29</b>
1.1 Overview	29
1.2 Reset Generation	30
1.3 Register Set	30
<b>2 APLL Specification</b>	<b>33</b>
2.1 Definitions	34
2.1.1 Phase Skew	34
2.1.2 Duty	34
2.1.3 Lock Up Time	34
2.1.4 Jitter	34
2.1.5 Variation in Output Cycle	34
2.1.6 Maximum Power Consumption	35
2.2 Usage Instructions	35
<b>3 Clock Setup and Configuration</b>	<b>36</b>
3.1 Configurable Circuitry	36
3.2 Clock Unit Programming Sequence	37
3.3 Application Notes	38
<b>B-2 User Logic Bus Controller (ULB)</b>	<b>39</b>

---

<b>1 Functional description</b>	<b>41</b>
1.1 ULB functions	41
1.2 ULB overview	42
1.3 Signal synchronisation between MCU and display controller	44
1.3.1 Write synchronization for Lavender and Jasmine	44
1.3.2 Read synchronization	45
1.3.3 DMA and interrupt signal synchronization	45
1.3.4 Output signal configuration	46
1.4 Address decoding	46
1.4.1 Overview	46
1.4.2 Register space	47
1.4.3 SDRAM space	49
1.4.4 Display controller bus access types (word, halfword, byte)	51
1.4.5 Display controller data modes (32/16 Bit interface)	52
1.5 Command decoding and execution	53
1.5.1 Command and data interface to MCU	53
1.5.2 Command execution and programming	55
1.5.3 Structure of command controller	58
1.5.4 Display controller commands	59
1.5.5 Registers and flags regarding command execution	60
1.6 Flag and interrupt handling	61
1.6.1 Flag and interrupt registers	61
1.6.2 Interrupt controller configuration	61
1.6.3 Interrupt generation	62
1.6.4 Interrupt configuration example	63
1.6.5 Display controller flags	64
1.7 DMA handling	64
1.7.1 DMA interface	64
1.7.2 DMA modes	64
1.7.2.1 Level triggered DMA (demand mode)	65
1.7.2.2 Edge triggered DMA (block-, step-, burstmode)	65
1.7.3 DMA settings	66
1.7.4 DMA programming examples	68
<b>2 ULB register set</b>	<b>71</b>
2.1 Description	71
2.2 ULB initialization	74
<b>B-3 SDRAM Controller (SDC)</b>	<b>75</b>

---

<b>1 Function Description</b>	<b>77</b>
1.1 Overview	77
1.2 Arbitration	78
1.3 SDRAM Timing	78
1.4 Sequencer for Refresh and Power Down	79
1.5 Address Mapping	81
1.5.1 Elucidations regarding Address Mapping	83
1.5.1.1 Block Structure of Pixel Data	83
1.5.1.2 Access Methods and Devices	84
1.5.1.3 Program Example	86
1.5.1.4 Address Calculation - Example with concrete {X,Y} Pixel	90
<b>2 SDRAM Ports</b>	<b>92</b>
2.1 External SDRAM I-/O-Pads with configurable sampling Time (Lavender)	92
2.2 Integrated SDRAM Implementation (Jasmine)	93
<b>3 Configuration</b>	<b>94</b>
3.1 Register Summary	94
3.2 Core clock dependent Timing Configuration	95
3.2.1 General Setup	95
3.2.2 Refresh Configuration for integrated DRAM (Jasmine)	95
<b>B-4 Pixel Processor (PP)</b>	<b>97</b>
<b>1 Functional Description</b>	<b>99</b>
1.1 Overview	99
1.1.1 PP Structure	99
1.1.2 Function of submodules	100
1.2 Configuration Registers	101
1.3 Special Command Options	103
1.3.1 Bitmap Mirror	103
1.3.2 Bitmap Direction	103
<b>2 Format Definitions</b>	<b>104</b>
2.1 Legend of symbols	104
2.2 Data Formats at ULB Interface	105
2.3 Data Formats for Video RAM / SDC Interface	106
<b>B-5 Antialiasing Filter (AAF)</b>	<b>109</b>
<b>1 Functional Description</b>	<b>111</b>

---

1.1 AAF Overview .....	111
1.1.1 Top Level Structure .....	111
1.1.2 AAU Function .....	112
1.1.2.1 Super Sampling .....	112
1.1.2.2 Drawing Resolution .....	112
1.1.2.3 Data and Address Processing .....	112
1.2 Configuration Registers .....	113
1.3 Application Notes .....	114
1.3.1 Restrictions due to Usage of AAF .....	114
1.3.2 Supported Colour Formats .....	115
1.3.3 Related SDC Configuration .....	115
<b>B-6 Direct and Indirect Physical Memory Access Unit (DIPA) .....</b>	<b>117</b>
<b>1 Overview .....</b>	<b>119</b>
<b>2 Configuration Registers .....</b>	<b>121</b>
2.1 Register List .....	121
2.2 Recommended Settings .....	121
2.3 Related Settings and Informations .....	122
<b>B-7 Video Interface Controller (VIC) .....</b>	<b>123</b>
<b>1 Introduction .....</b>	<b>125</b>
1.1 Video Interface Controller functions and features .....	125
1.2 Video data handling .....	125
<b>2 VIC Description .....</b>	<b>127</b>
2.1 Data Input Formats .....	127
2.2 Data format in Video RAM (SDRAM) .....	129
2.3 Data Input Timing .....	130
2.3.1 Videoscaler-Mode .....	130
2.3.2 CCIR-Mode .....	131
2.3.3 External-Timing-mode .....	132
<b>3 VIC settings .....</b>	<b>135</b>
3.1 Register list .....	135
3.2 Register Description .....	139
<b>B-8 Graphic Processing Unit (GPU) .....</b>	<b>149</b>
<b>1 Functional Description .....</b>	<b>151</b>

---

1.1 GPU Features . . . . .	151
1.2 GPU Overview . . . . .	152
1.2.1 Top Level Structure . . . . .	152
1.2.2 DFU Function . . . . .	152
1.2.3 CCU Function . . . . .	153
1.2.4 LSA Function . . . . .	154
1.2.5 BSF Function . . . . .	154
<b>2 Color Space Concept . . . . .</b>	<b>156</b>
2.1 Background . . . . .	156
2.2 Data Flow for Color Space Conversion within GPU . . . . .	156
2.3 Mapping from Logical to Intermediate Color Space . . . . .	157
2.4 Mapping from Intermediate to Physical Color Space . . . . .	157
<b>3 GPU Control Information . . . . .</b>	<b>162</b>
3.1 Layer Description Record . . . . .	162
3.2 Merging Description Record . . . . .	164
3.3 Display Interface Record . . . . .	165
3.4 Supported Physical Color Space / Bit Stream Format Combinations . . . . .	167
3.5 Twin Display Mode . . . . .	167
3.6 Scan Modes . . . . .	168
3.7 YUV to RGB conversion . . . . .	171
3.7.1 YUV422 Demultiplexing and Chrominance Interpolation . . . . .	171
3.7.2 Matrix Multiplication . . . . .	172
3.7.3 Inverse Gamma Correction . . . . .	172
3.8 Duty Ratio Modulation . . . . .	173
3.8.1 Working Principle . . . . .	173
3.8.2 Usage . . . . .	174
3.9 Master Timing Information . . . . .	175
3.10 Generation of Sync Signals . . . . .	176
3.10.1 Overview . . . . .	176
3.10.2 Position Matching . . . . .	176
3.10.3 Sequence Matching . . . . .	177
3.10.4 Combining First-Stage Sync Signals . . . . .	178
3.10.5 Sync Signal Delay Adjustment . . . . .	180
3.11 Pixel Clock Gating . . . . .	180
3.12 Numerical Mnemonic Definitions . . . . .	181
3.12.1 Color Space Code . . . . .	181
3.12.2 Bit Stream Format Code . . . . .	182
3.12.3 Scan Mode Code . . . . .	182

---

3.13 Bit to Color Channel Assignment . . . . .	183
3.14 GPU Signal to GDC Pin Assignment . . . . .	184
3.14.1 Multi-Purpose Digital Signals . . . . .	184
3.14.2 Analog Pixel Data. . . . .	184
3.14.3 Dedicated Sync Signals . . . . .	184
3.14.4 Sync Mixer connections . . . . .	185
3.14.5 Color Key Output . . . . .	185
<b>4 GPU Register Set . . . . .</b>	<b>187</b>
4.1 Description . . . . .	187
4.2 Determination of Register Contents . . . . .	194
4.2.1 Values Derived from Display Specs. . . . .	194
4.2.2 Values Determined by Application . . . . .	196
4.2.3 User preferences . . . . .	197
4.3 GPU Initialization Sequence . . . . .	197
<b>5 Bandwidth Considerations . . . . .</b>	<b>198</b>
5.1 Processing Bandwidth. . . . .	198
5.1.1 Average Bandwidth . . . . .	198
5.1.2 Peak Bandwidth . . . . .	198
5.2 Memory Bandwidth . . . . .	200
5.2.1 Average Bandwidth . . . . .	201
5.2.2 Peak Bandwidth . . . . .	201
5.3 Recommendations . . . . .	202
<b>6 Functional Peculiarities. . . . .</b>	<b>204</b>
6.1 Configuration Constraints . . . . .	204
6.2 Bandwidth . . . . .	204
6.3 Image Processing . . . . .	205
6.4 Data Output . . . . .	206
6.5 Diagnostics . . . . .	206
<b>7 Supported Displays . . . . .</b>	<b>207</b>
7.1 Passive Matrix LCD . . . . .	208
7.2 Active Matrix (TFT) Displays . . . . .	209
7.3 Electroluminescent Displays . . . . .	210
7.4 Field Emission Displays . . . . .	211
7.5 Limitations for support . . . . .	212
7.5.1 No Support due to VCOM Inversion . . . . .	212
7.5.2 Problems due to 5V CMOS Interfaces . . . . .	212



---

**B-9 Cold Cathode Fluorescence Light Driver (CCFL) . . . . . 213****1 Introduction . . . . . 215****2 Signal Waveform . . . . . 217**

2.1 General Description . . . . . 217

2.2 Duration of the Phases . . . . . 217

2.3 Pulse shape of FET1 and FET2 . . . . . 218

**3 Register Description. . . . . 219**

3.1 Overview . . . . . 219

3.2 Control Bits . . . . . 219

**4 Application Notes . . . . . 221**

4.1 CCFL Setup Example . . . . . 221

4.2 CCFL Protection . . . . . 221

***PART C - Pinning and Electrical Specification*****1 Pinning and Buffer Types . . . . . 225**

1.1 Pinning for MB87P2020-A . . . . . 225

1.1.1 Pinning . . . . . 225

1.1.2 Buffer types . . . . . 238

1.2 Pinning for MB87P2020 . . . . . 238

1.2.1 Pinning . . . . . 238

1.2.2 Buffer types . . . . . 251

1.3 Pinning for MB87J2120. . . . . 252

1.3.1 Pinning . . . . . 252

1.3.2 Buffer Types . . . . . 268

**2 Electrical Specification . . . . . 270**

2.1 Maximum Ratings . . . . . 270

2.1.1 Power-on sequence . . . . . 270

2.1.2 External Signal Levels . . . . . 271

2.1.3 APLL Power Supply Level . . . . . 271

2.1.4 DAC supply . . . . . 271

2.1.5 SDRAM Supply . . . . . 271

2.2 Recommended Operating Conditions . . . . . 272

2.3 DC Characteristics . . . . . 272

2.4 Mounting / Soldering . . . . . 274

2.5 AC Characteristics . . . . . 275

2.5.1 Measurement Conditions .....	275
2.5.2 Definitions .....	275
2.5.3 Clock inputs .....	276
2.5.4 MCU User Logic Bus Interface .....	278
2.5.5 Interrupt .....	280
2.5.6 DMA Control Ports .....	281
2.5.7 Display Interface .....	282
2.5.8 Video Input .....	283
2.5.9 CCFL FET Driver .....	283
2.5.10 Serial Peripheral Bus .....	284
2.5.11 Special and Mode Pins .....	284
2.5.12 SDRAM Ports (Lavender) .....	284

## ***PART D - Appendix***

### **D-1 Jasmine Command and Register Description.....289**

#### **1 Register Description .....291**

#### **2 Flag Description.....319**

#### **3 Command Description .....323**

##### 3.1 Command List..... 323

##### 3.2 Command and I/O Control ..... 330

### **D-2 Hints and restrictions for Lavender and Jasmine .....331**

#### **1 Special hints .....333**

##### 1.1 IPA resistance against wrong settings..... 333

##### 1.2 ULB\_DREQ pin timing to host MCU ..... 333

##### 1.3 CLKPDR master reset..... 334

##### 1.4 MAU (Memory Access Unit) commands..... 334

##### 1.5 Pixel Processor (PP) double buffering ..... 335

##### 1.6 Robustness of ULB\_RDY signal ..... 336

##### 1.7 Robustness of command pipeline against software errors ..... 337

##### 1.8 DMA resistance against wrong settings ..... 337

#### **2 Restrictions.....339**

##### 2.1 ESD characteristics for I/O buffers..... 339

##### 2.2 Command FSM..... 341

##### 2.3 GPU mastertiming synchronization ..... 341

---

2.4 Read limitation for 16 Bit data interface to MCU . . . . .	342
2.5 SDC sequencer readback . . . . .	344
2.6 Direct SDRAM access with 16bit and 8bit data mode . . . . .	344
2.7 Input FIFO read in 16bit mode . . . . .	345
2.8 ULB_DSTP pin function . . . . .	345
2.9 Software Reset for command execution . . . . .	346
2.10 AAF settings double buffering . . . . .	347
2.11 Pixel Engine (PE) Commands . . . . .	348
2.12 Pixel read back commands (GetPixel, XChPixel) . . . . .	349
2.13 Display Interface Re-configuration . . . . .	351
<b>D-3 Abbreviations . . . . .</b>	<b>353</b>



---

## ***PART A - Lavender and Jasmine Overview***



# 1 Overview

## 1.1 Application overview

The MB87J2120 "Lavender" and MB87P2020-A "Jasmine" are colour LCD/CRT graphic display controllers (GDCs)<sup>1</sup> interfacing to MB91xxxx micro controller family and support a wide range of display devices. The architecture is designed to meet the low cost, low power requirements in embedded and especially in automotive<sup>2</sup> applications.

Lavender and Jasmine support almost all LCD panel types and CRTs or other progressive scanned<sup>3</sup> monitors/displays which can be connected via the digital or analog RGB output. Products requiring video/camera input can take advantage of the supported digital video interface. The graphic instruction set is optimized for minimal traffic at the MCU interface because it's the most important performance issue of co-processing graphic acceleration systems. Lavender uses external connected SDRAM, Jasmine is a compatible GDC version with integrated SDRAM (1MByte) and comes with additional features.

Lavender and Jasmine support a set of 2D drawing functions with built in Pixel Processor, a video scaler interface, units for physical and direct video memory access and a powerful video output stream formatter for the greatest variety of connectable displays.

Figure 1-1 displays an application block diagram in order to show the connection possibilities of Jasmine. For Lavender external SDRAM connection is required in addition.

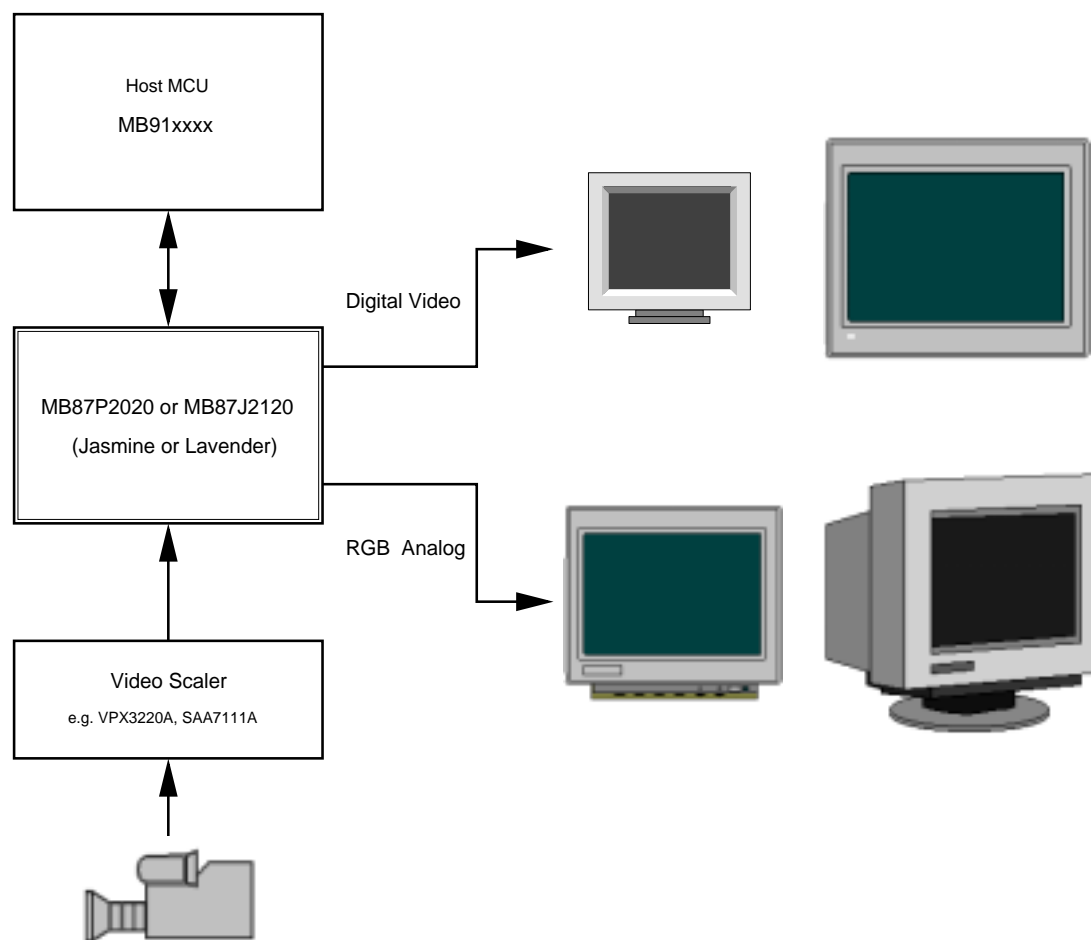
## 1.2 Jasmine/Lavender Block Diagram

Figure 1-2 shows all main components of Jasmine/Lavender graphic controllers. The User Logic Bus controller (ULB), Clock Unit (CU) and Serial Peripheral Bus (SPB) are connected to the User Logic Bus interface of 32 bit Fujitsu RISC microprocessors. 32 and 16 bit access modes are supported.

**Table 1-1:** GDC components

Shortcut	Meaning	Main Function
CCFL	Cold Cathode Fluorescence Lamp	Cold cathode driver for display backlight
CU	Clock Unit	Clock gearing and supply, Power save
DAC	Digital Analog Converter	Digital to analog conversion for analog display
DPA (part of DIPA)	Direct Physical memory Access	Memory mapped SDRAM access with address decoding
GPU	Graphics Processing Unit	Frame buffer reader which converts to video data format required by display
DFU (part of GPU)	Data Fetch Unit	Graphic/video data acquisition
CCU (part of GPU)	Colour Conversion Unit	Colour format conversion to common intermediate overlay format

1. The general term 'graphic display controller' or its abbreviation 'GDC' is used in this manual to identify both devices. Mainly this is used to emphasize its common features.
2. Both display controllers have an enhanced temperature range of -40 to 85 °C.
3. TV conform output (interlaced) is also possible with half the vertical resolution (line doubling).

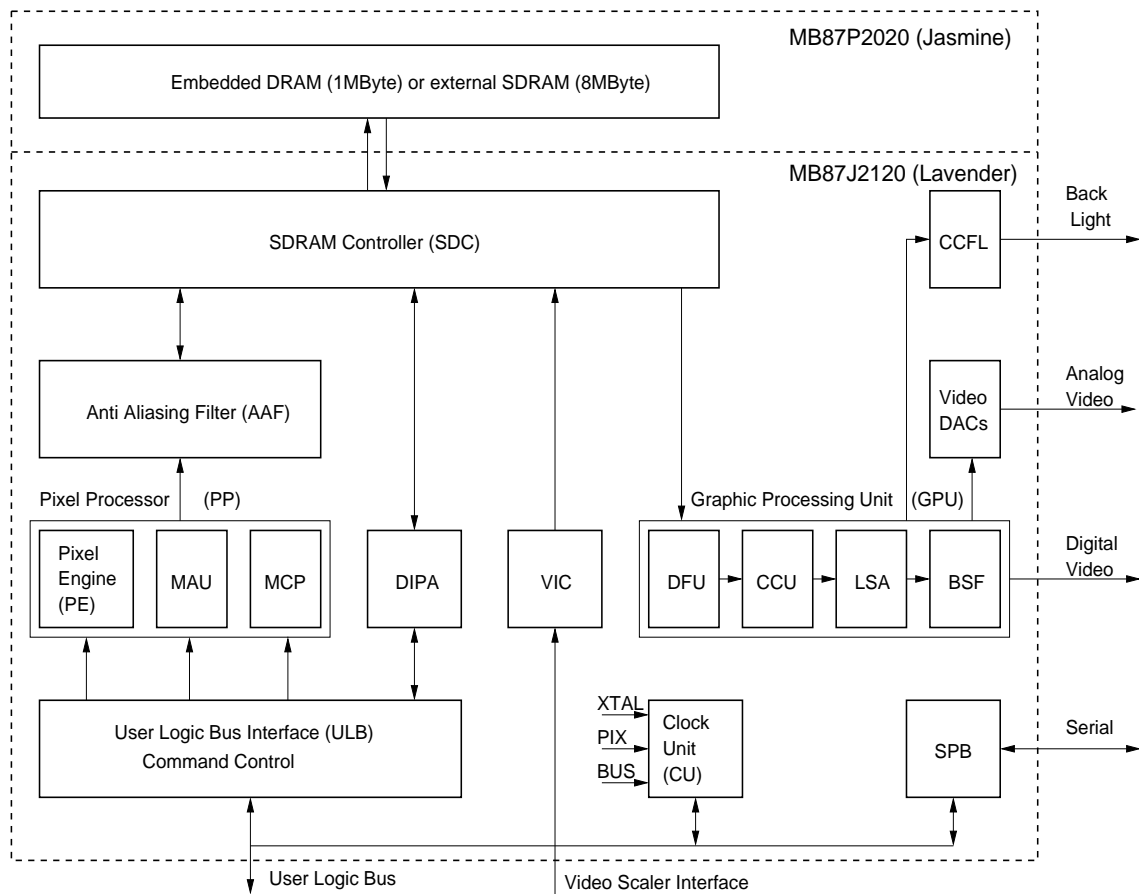


**Figure 1-1:** Application overview

**Table 1-1:** GDC components

Shortcut	Meaning	Main Function
LSA (part of GPU)	Line Segment Accumulator	Layer overlay
BSF (part of GPU)	BitStream Formatter	Intermediate format to physical display Format converter, Sync generation
IPA (part of DIPA)	Indirect Physical memory Access	SDRAM access with command register and FIFO
MAU (part of PP)	Memory Access Unit	Pixel access to video RAM
MCP (part of PP)	Memory CoPy	Memory to memory copying of rectan- gular areas
PE (part of PP)	Pixel Engine	Drawing of geometrical figures and bit- maps
PP	Pixel Processor	Graphic oriented functions
SDC	SDRAM Controller	SDRAM access and arbitration
SPB	Serial Peripheral Bus	Serial interface (master)
ULB	User Logic Bus (see MB91360 series specification)	Address decoding, command control, flag, interrupt and DMA handling





**Figure 1-2:** Component overview for Lavender and Jasmine graphic controllers

**Table 1-1:** GDC components

Shortcut	Meaning	Main Function
VIC	Video Interface Controller	YUV-/RGB-Interface to video grabber

The ULB provides an interface to host MCU (MB91360 series). The main functions are MCU (User Logic Bus) control inclusive wait state handling, address decoding and device controls, data buffering / synchronisation between clock domains and command decoding. Beside normal data and command read and write operation it supports DMA flow control for full automatic data transfer from MCU to GDC and vice versa. Also an interrupt controlled data flow is possible and various interrupt sources inside the graphics controller can be programmed.

The Clock Unit (CU) provides all necessary clocks to module blocks of GDC and a FR compliant (ULB) interface to host MCU. Main functions are clock source select (XTAL, ULB clock, display clock or special pin), programmable clock multiplier/divider with APLL, power management for all GDC devices and the generation of synchronous RESET signal.

For Fujitsu internal purposes one independent macro is build in the GDC ASIC, the Serial Peripheral Bus (SPB). It's a single line serial interface. There is no interaction with other GDC components.

All drawing functions are executed in Pixel Processor (PP). It consists of three main components Pixel Engine (PE), Memory Access Unit (MAU) and Memory Copy (MCP). All functions provided by these blocks are related to operations with pixel addresses {X, Y} possibly enhanced with layer information. GDC supports 16 layers by hardware, four of them can be visible at the same time. Each layer is capable of storing

any data type (graphic or video data with various colour depths) only restricted by the bandwidth limitation of video memory at a given operating frequency.

Drawing functions are executed in the PE by writing commands and their dedicated parameter sets. All commands can be taken from the command list in section 4.2. Writing of uncompressed and compressed bitmaps/textures, drawing of lines, poly-lines and rectangles are supported by the PE. There are many special modes such as duplicating data with a mirroring function.

Writing and reading of pixels in various modes is handled by MAU. Single transfers and block or burst transfers are possible. Also an exchange pixel function is supported.

With the MCP unit it is possible to transfer graphic blocks between layers of the same colour representation very fast. Only size, source and destination points have to be given to duplicate some picture data. So it offers an easy and fast way to program moving objects or graphic libraries.

All PP image manipulation functions can be fed through an Antialiasing Filter (AAF). This is as much faster than a software realisation. Due to the algorithm which shrinks the graphic size by two this has to be compensated by doubling the drawing parameters i.e. the co-ordinates of line endpoints.

DIPA stands for Direct/Indirect Physical Access. This unit handles rough video data memory access without pixel interpretation (frame buffer access). Depending on the colour depth (bpp, bit per pixel) one or more pixel are stored in one data word. DPA (Direct PA) is a memory-mapped method of physical access. It is possible in word (32 bit), half word (16 bit) or byte mode. The whole video memory or partial window (page) can be accessed in a user definable address area of GDC. IPA (Indirect PA) is controlled per ULB command interface and IPA access is buffered through the FIFOs to gain high access performance. It uses the command GetPA and PutPA, which are supporting burst accesses, possibly handled with interrupt and DMA control.

For displaying real-time video within the graphic environment both display controllers have a video interface for connection of video-scaler chips, e.g. Intermetall's IC VPX32xx series or Phillips SAA711x. Additionally the video input of Jasmine can handle CCIR standard conform digital video streams.

Several synchronisation modes are implemented in both controllers and work with frame buffering of one up to three pictures. With line doubling and frame repetition there exist a large amount of possibilities for frame rate synchronisation and interlaced to progressive conversion as well. Due to the strict timing of most graphic displays the input video rate has to be independent from the output format. So video data is stored as same principles as for graphic data using up to three of the sixteen layers.

The SDC is a memory controller, which arbitrates the internal modules and generates the required access timings for SDRAM devices. With a special address mapping and an optimized algorithm for generating control commands the controller can derive full benefit from internal SDRAM. This increases performance respective at random (non-linear) memory access.

The most complex part of GDC is its graphic data processing unit (GPU). It reads the graphic/video data from up to four layers from video memory and converts it to the required video output streams for a great variety of connectable display types. It consists of Data Fetch Unit (DFU), Colour Conversion Unit (CCU) which comes with 512 words by 24-bit colour look up table, Line Segment Accumulator (LSA) which does the layer overlay and finally the Bitstream Formatter (BSF). The GPU has such flexibility for generating the data streams, video timings and sync signals to be capable of driving the greatest variety of known display types.

Additional to the digital outputs video DACs provide the ability to connect analog video destinations. A driver for the displays Cold Cathode Fluorescence Lamp (CCFL) makes the back light dimmable. It can be synchronized with the vertical frequency of the video output to avoid visible artefacts during modulating the lamp.

## 2 Features and Functions

**Table 2-1:** Lavender and Jasmine features in comparison

MB87J2120 (Lavender)	MB87P2020-A (Jasmine)
<i>General features</i>	
<ul style="list-style-type: none"> <li>2M words x 32 Bit <b>external</b> SDRAM (64 Mbit)</li> <li>no <b>internal</b> SDRAM</li> </ul>	<ul style="list-style-type: none"> <li>256k words x 32 Bit <b>internal</b> SDRAM (8 Mbit)</li> </ul>
<ul style="list-style-type: none"> <li>Package: BGA-256P-M01</li> </ul>	<ul style="list-style-type: none"> <li>Package: FPT-208P-M06</li> </ul>
<ul style="list-style-type: none"> <li>Chip select sharing for up to four GDC devices</li> </ul>	
<ul style="list-style-type: none"> <li>synchronized reset (needs applied clock)</li> </ul>	<ul style="list-style-type: none"> <li>immediate asynchronous reset, synchronized reset release</li> </ul>
<i>Pixel manipulation functions</i>	
<ul style="list-style-type: none"> <li>2D Drawing and Bitmap Functions               <ul style="list-style-type: none"> <li>Lines and Polygons</li> <li>Rectangular Area</li> <li>Uncompressed Bitmap/1bit pixelmask</li> <li>Compressed Bitmap (TGA format)/1bit pixelmask</li> </ul> </li> </ul>	
<ul style="list-style-type: none"> <li>Pixel Memory Access Functions               <ul style="list-style-type: none"> <li>Put Pixel</li> <li>Put Pixel FC (fixed colour)</li> <li>Put Pixel Word (packed)</li> <li>Exchange Pixel</li> <li>Get Pixel</li> </ul> </li> </ul>	
<ul style="list-style-type: none"> <li>Layer Register for text and bitmap functions</li> </ul>	<ul style="list-style-type: none"> <li>Layer Register enhancement for drawing functions (simplifies pixel addressing)</li> </ul>
<ul style="list-style-type: none"> <li>Copy rectangular areas between layers</li> </ul>	
<ul style="list-style-type: none"> <li>Anti Aliasing Filter (AAF)               <ul style="list-style-type: none"> <li>resolution increase by factor 2 for each dimension (2x2 filter operator size)</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>Additional 4x4 AAF operator size</li> </ul>
<i>Display</i>	
<ul style="list-style-type: none"> <li>Free programmable Bitstream Formatter for a great variety of supported displays (single/dual/alternate scan):               <ul style="list-style-type: none"> <li>Passive Matrix LCD (single/dual scan)</li> <li>Active Matrix (TFT) Displays</li> <li>Electroluminescent Displays</li> <li>Field Emission Displays</li> <li>TV compatible output</li> <li>CRTs...</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>Additional Twin Display Mode feature (simultaneous digital and analog output without limitation of DIS_D[23:16] that carry special sync signals).</li> </ul>
<ul style="list-style-type: none"> <li>24 bit digital video output (RGB)</li> </ul>	

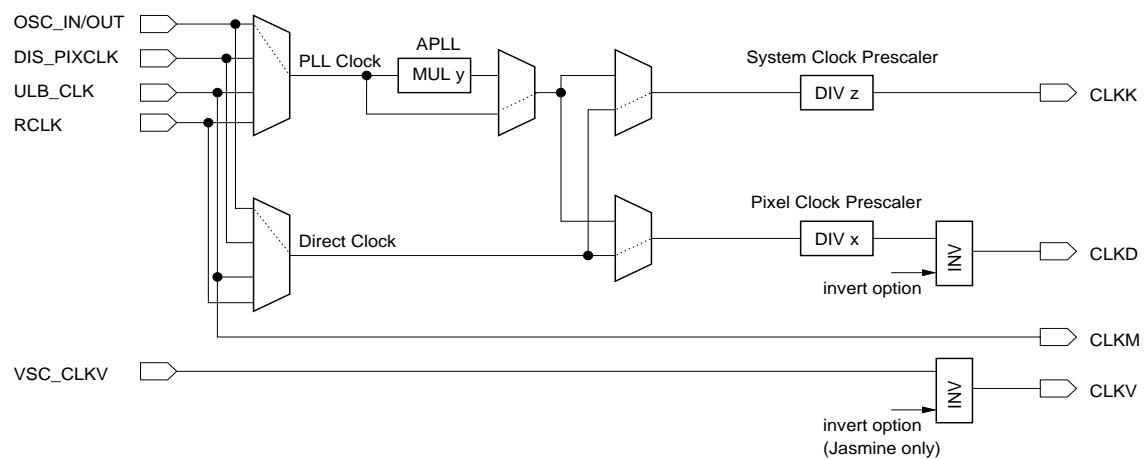
**Table 2-1:** Lavender and Jasmine features in comparison

MB87J2120 (Lavender)	MB87P2020-A (Jasmine)
<ul style="list-style-type: none"> <li>On-Chip Video DAC, 50M Samples/s (dot clock)</li> </ul>	
<ul style="list-style-type: none"> <li>Flexible three-stage sync signal programming (trigger position/sequence, combining and delay) for up to 8 signal outputs</li> </ul>	
<ul style="list-style-type: none"> <li>Colour keying between two limits</li> </ul>	
<ul style="list-style-type: none"> <li>Brightness modulation for displays with a Cold Cathode Fluorescence Lamp back-light</li> </ul>	
<ul style="list-style-type: none"> <li>Display resolution/drawing planes up to 16383 pixels for each dimension</li> </ul>	
<ul style="list-style-type: none"> <li>4 layer + background colour simultaneous display and graphic overlay, programmable Z-order</li> </ul>	
<ul style="list-style-type: none"> <li>Blinking, transparency and background attributes</li> </ul>	
<ul style="list-style-type: none"> <li>Free programmable display section of a layer</li> </ul>	
<ul style="list-style-type: none"> <li>Separable Colour LUT with 256 entries x 24 Bit</li> </ul>	<ul style="list-style-type: none"> <li>Colour LUT expansion to 512 entries</li> </ul>
<ul style="list-style-type: none"> <li>Duty Ratio Modulation (DRM) for pseudo hue/grey levels</li> </ul>	
<ul style="list-style-type: none"> <li>Hardware support for 16 layers, usable for graphic/video without restrictions</li> </ul>	
<ul style="list-style-type: none"> <li>Performance sharing with adjustable priorities and configurable block sizes for memory transfers enable maximal throughput for a wide range of applications</li> </ul>	
<ul style="list-style-type: none"> <li>Variable and display independent colour space concept: Layers with 1, 2, 4, 8, 16, 24 bit per pixel can be mixed and converted to one display specific format (logical-intermediate-physical format mapping)</li> </ul>	<ul style="list-style-type: none"> <li>Additional GPU a YUV to RGB converter in order to allow YUV coded layers</li> <li>Additional Gamma correction RAMs are included (3x256x8Bit)</li> </ul>
<i>Physical SDRAM access</i>	
<ul style="list-style-type: none"> <li>Memory mapped direct physical access for storage of non-graphics data or direct image access</li> </ul>	
<ul style="list-style-type: none"> <li>Indirect physical memory access for high bandwidth multipurpose data/video memory access</li> </ul>	
<i>MCU interface</i>	
<ul style="list-style-type: none"> <li>32/16 Bit MCU interface, designed for direct connection of MB91xxxx family (8/16/32Bit access)</li> </ul>	
<ul style="list-style-type: none"> <li>DMA support (all MB91xxxx modes)</li> </ul>	
<ul style="list-style-type: none"> <li>Interrupt support</li> </ul>	
<i>Video interface</i>	
<ul style="list-style-type: none"> <li>Video interface VPX32xx series by Micronas Intermetall, Phillips SAA711x and others</li> </ul>	<ul style="list-style-type: none"> <li>Additional CCIR conform input mode</li> </ul>
<ul style="list-style-type: none"> <li>Video synchronization with up to 3 frame buffers</li> </ul>	
<i>Clock generation</i>	
<ul style="list-style-type: none"> <li>Flexible clocking concept with on-chip PLL and up to 4 external clock sources: <ul style="list-style-type: none"> <li>XTAL</li> <li>ULB bus clock</li> <li>Pixel clock</li> <li>Additional external clock pin (MODE[3]/RCLK)</li> </ul> </li> </ul>	
<ul style="list-style-type: none"> <li>Separate power saving for each sub-module</li> </ul>	

### 3 Clock supply and generation

GDC has a flexible clocking concept where four input clocks (OSC\_IN/OUT, DIS\_PIXCLK, ULB\_CLK, RCLK) can be used as clock source for Core clock (CLKK) and Display clock (CLKD).

The user can choose by software whether to take the direct clock input or the output of an APLL independent for Core- and Display clock. Both output clocks have different dividers programmable by software (DIV x for CLKD and DIV z for CLKK). The clock gearing facilities offer the possibility to scale system performance and power consumption as needed.



**Figure 3-1:** Clock gearing and distribution

Beside these two configurable clocks (CLKK and CLKD) GDC needs two additional internal clocks: CLKM and CLKV (see also figure 3-1). CLKV is exclusively for video interface and is connected to input clock pin VSC\_CLKV. CLKM is used for User Logic Bus (ULB) interface and is connected to input clock ULB\_CLK. As already mentioned ULB\_CLK can also be used to build CLKK and/or CLKD.

Table 3-1 shows all clocks used by GDC with their requirements.

**Table 3-1:** Clock supply

Clock	Type	Symbol	Requirements			Unit
			Min	Typ	Max	
XTAL clock	input	OSC_IN, OSC_OUT	12 <sup>a</sup>	-	64	MHz
Reserve clock	input	RCLK	ULB_CLK <sup>b</sup>	-	64	MHz
ULB clock	input	ULB_CLK	-	-	64	MHz
Pixel clock	input	DIS_PIXCLK	-	-	54	MHz
Video clock	input	VSC_CLKV	-	-	54 <sup>c</sup>	MHz
Core clock	internal	CLKK	ULB_CLK	-	64	MHz
Display clock	internal	CLKD	-	-	54	MHz
Video clock	internal	CLKV	-	-	54 <sup>c</sup>	MHz
ULB clock	internal	CLKM	-	-	64	MHz

- a. If used as PLL input. APLL input frequency has to be at minimum 12 MHz, regardless which clock is routed to APLL.
- b. If used as direct clock source bypassing the APLL, the user should take care that resulting core clock frequency is above or equal to MCU bus interface clock. **Be aware of tolerances!**
- c. The video interface is designed to achieve 54 MHz but there is a side condition that video clock should be smaller than half of core clock.

## 4 Register and Command Overview

### 4.1 Register Overview

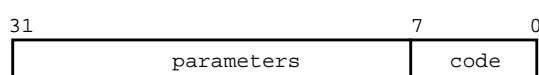
The GDC device is mainly configurable by registers. These configuration registers are mapped in a 64 kByte large address range from 0x0000 to 0xffff. It is possible to shift this register space in steps of 64 kByte by the Mode[1:0] pins in order to connect multiple GDC devices.

Above this 4\*64 kByte = 256 kByte address range the SDRAM video memory could be made visible for direct physical access.

At byte address 0x1f:ffff GDC memory map ends with a total size of 2 MByte.

### 4.2 Command Overview

The command register width is 32 Bit. It is divided into command code and parameters:



Partial writing (halfword and byte) of command register is supported. Command execution is triggered by writing byte 3 (code, bits [7:0]). Thus parameters should be written before command code.

Not all commands need parameters. In these cases parameter section is ignored.

In table 4-1 all commands are listed with mnemonic, command code and command parameters (if necessary). This is only a short command overview, a more detailed command list can be found in appendix.

**Table 4-1:** Command List

Mnemonic	Code	Function	Addressed device
Bitmap and Texture Functions			
PutBM	01H	Store bitmap into Video RAM	Pixel Processor
PutCP	02H	Store compressed bitmap into Video RAM	
PutTxtBM	05H	Draw uncompressed texture with fixed foreground and background colour	
PutTxtCP	06H	Draw compressed texture with fixed foreground and background colour	
Drawing Functions (2D)			
DwLine	03H	"Draw a line" - calculate pixel position and store LINECOL into Video RAM	Pixel Processor
DwPoly	0FH	"Draw a polygon" - draws multiple lines between defined points, see DwLine	
DwRect	04H	"Draw an rectangle" - calculate pixel addresses and store RECTCOL into Video RAM	
Pixel Operations			

**Table 4-1:** Command List

Mnemonic	Code	Function	Addressed device
PutPixel	07H	Store single pixel data into Video RAM	Pixel Processor
PutPxWd	08H	Store word of packed pixels into Video RAM	
PutPxFC	09H	Store fixed colour pixel data in Video RAM	
GetPixel	0AH	Load pixel data from Video RAM	
XChPixel	0BH	Load old pixel in Output FIFO and store pixel from Input FIFO into Video RAM	
Memory to Memory Operations			
MemCopy	0CH	Memory Copy of rectangular area. Transfer of bit-maps from one layer to another or within one layer.	Pixel Processor
Physical Framebuffer Access			
PutPA	0DH	Store data in physical format into Video RAM, with physical address auto-increment	DIPA
GetPA	<n>,0EH	Load data in physical format from Video RAM with address auto-increment, stop after n words	
System Control Commands			
SwReset	00H	Stop current command immediately, reset command controller and FIFOs	All drawing and access devices
NoOp	FFH	No drawing or otherwise operation, finish current command and flush buffers	Command Control (ULB)



---

## ***PART B - Functional Descriptions***



---

## ***B-1 Clock Unit (CU)***



# 1 Functional Description

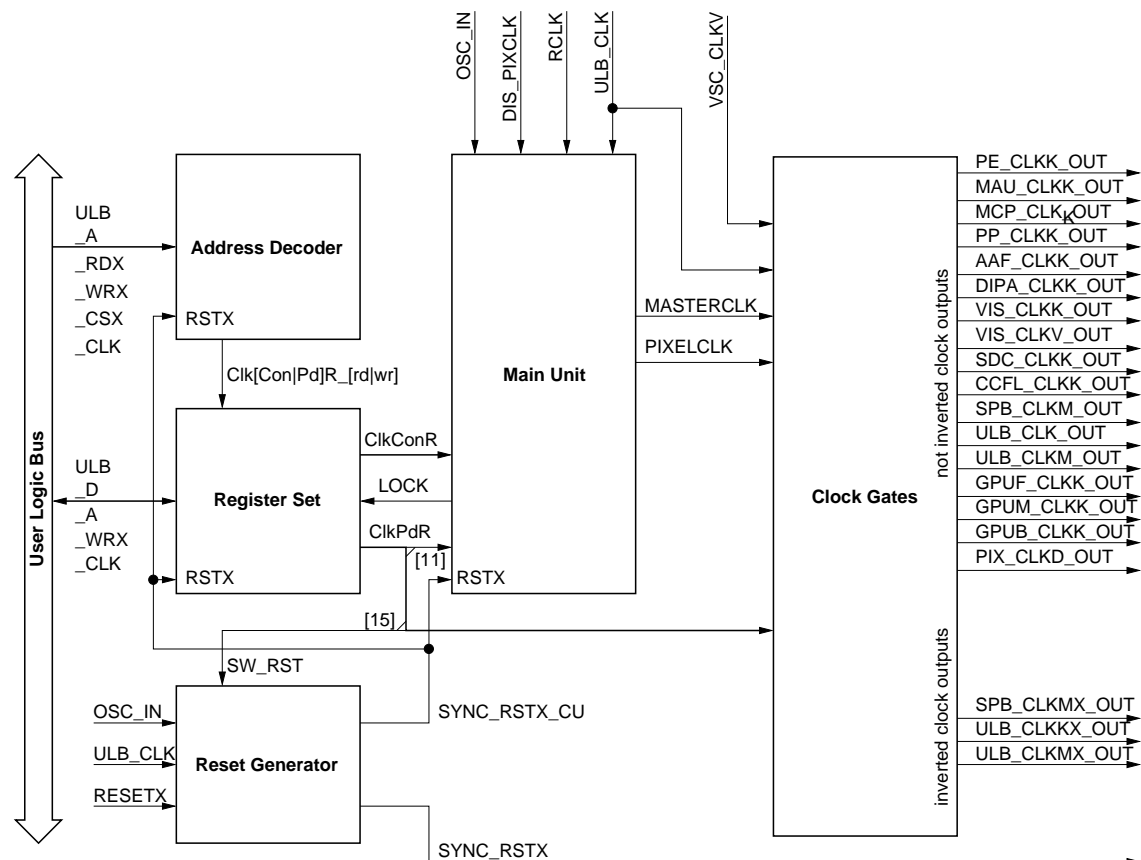
## 1.1 Overview

The clock unit (CU) provides all necessary clocks to GDC modules and an own interface to host MCU (MB91360 series) in order to have durable access even if ULB clocks switched off.

The main functions of CU are:

- Clock source select (Oscillator, MCU Bus clock, Display clock and a reserve clock input)
- Programmable clock multiplier with APLL
- Separate dividers for master (core) clock and pixel clock
- Power management for all GDC modules
- Generation of synchronized RESET signal
- MB91360 series compliant (ULB) Bus interface for clock setup

Figure 1-1 shows the overview of the Clock Unit. OSC\_IN, DIS\_PIXCLK, ULB\_CLK and RCLK<sup>1</sup> are possible to use as input sources. Both clock outputs of the main unit (MASTERCLK and PIXELCLK) and two directly used clock inputs (ULB\_CLK and VSC\_CLKV) driving the clock gates unit which distributes to all connected GDC sub-modules.



**Figure 1-1:** Block diagram of Clock Unit

1. MODE[3] pin is used for RCLK at Lavender

The GDC device has four different clock domains, that means clocks derived from four different sources. The largest part of the design runs at core clock which operates at the highest frequency driven by the MASTERCLK output. Thus normally the APLL is used to provide a higher internal operation frequency. The next domain is the display output interface which operates at pixel clock frequency. For most applications it is recommended that this is the clock from OSC\_IN pin, divided by two<sup>1</sup>. So the crystal oscillator has to be chosen to have a whole-numbered multiple of the display clock frequency. Preferred routing is the DIRECT clock source channel since some displays require a small clock jitter which is not able to provide by the APLL. The other clocks for MCU interface (ULB\_CLK) and video interface (VSC\_CLKV) are not derived by the clock routing and generating part and used directly from the appropriate input pin.

Finally the generated source clocks of the for domains go to the clock gating/distribution module. There are gated clock buffers and inverters for each GDC module implemented. Each module has it's own clock enable flag which can be programmed for modules needed by the application only. This method saves power of not used functional blocks of GDC (refere to table 3-1).

The configuration of CU is stored in two registers, ClkConR and ClkPdR, which are connected to User Logic Bus for writing and reading. The bus interface consists of an address decoder and circuitry for different access types (word, halfword and byte access over a 16 or 32 bit bus connection).

## 1.2 Reset Generation

GDC works with an internally synchronized, low active reset signal. The global chip reset can be triggered by an external asynchronous reset or internally by software reset (configuration bit in ClkPdR). The external triggered RESETX results in resetting all GDC components including the Clock Unit, however software reset has no influence on CU internal registers.

Lavender synchronizes its external reset (RSTX pin). Reset is delayed until 4 clock cycles of each ULB\_CLK and OSC\_IN are executed. This gives stability against spikes on the RSTX line but has the disadvantage of delayed reset response of Lavender.

For Jasmine internal reset is active immediately after tying RESETX low plus a small spike filter delay. Due to the synchronization of RESETX the internal reset state ends after 4 clock periods of OSC\_IN and 4 clock periods of ULB\_CLK after releasing RESETX pin. Reset output RSTX\_SYNC for all internal GDC register states are synchronized with OSC\_IN, however internal Clock Unit registers are synchronized with ULB\_CLK in addition. Thus a minimum recovery time of 4 clock cycles of OSC\_IN plus 4 cycles of ULB\_CLK is needed before writing to Clock Unit configuration registers is possible after RESETX becomes inactive.

The reset generator of Jasmine has a spike filter implemented, which suppresses short low pulses, typical smaller than 9 ns. Under best case operating conditions (-40 deg. C; 2.7V; fast) maximum suppressed spike width is specified to 5.5ns. This is the maximum reset pulse width which did not result in resetting the GDC device. Minimum pulse width for guaranteed reset is specified to 1 clock cycle of OSC\_IN (80 ns typical).

## 1.3 Register Set

Table 1-1 listst the clock setup registers. ClkConR (Clock Configuration Register) is mainly for generation of the base clocks and the routing/selection from one of the four input sources. It controls the clock dividers and the use of the APLL. The possibility to use a second clock path, called direct clock source, gives a high flexibility for using the APLL either for MASTERCLK or PIXELCLK generation or both. Also the pin function of DIS\_PIXCLK can be defined in this register. If DIS\_PIXCLK is selected as clock source the pin should be configured as an input.

Upper 8 bits of ClkConR are used as identification of the different GDC types. Lavender is identified with reading back a '0x00', Jasmine with a '0x01'.

Use of DIS\_PIXCLK as pixel clock output and selection of DIS\_PIXCLK for the clock source can result in unintentional feedbacks and has to be avoided.

---

1. Preferred is an even divider value to achive 50% clock duty

ClkPdR (Clock Power Down Register) is a set of enable bits for the clocks provided to the dedicated GDC modules. A bit set to '1' means the clock is enabled. If a module requires multiple clocks (inverted ones or different domains) the enable bit switches all these lines.

Additional ClkPdR controls the work of the PLL and gives status information about it's lock-state. Also a global GDC reset function can be executed by setting a configuration bit of this register.

**Table 1-1:** CU registers

Register	Bit	Function	Description	Reset Value
ClkConR	[31:30]	Direct Clock Source	00 Crystal oscillator (reset default) 01 Pixel clock 10 MCU Bus clock 11 reserved clock input (RCLK <sup>a</sup> )	"00"
	[29:24]	System clock prescaler (DIV z)	[5:0] system clock prescaler value	0
	[23:22]	PLL Clock Source	00 Crystal oscillator (reset default) 01 Pixel clock 10 MCU Bus clock 11 reserved clock input (RCLK)	"00"
	[21:16]	PLL Feedback divider (DIV y)	[5:0] pll multiplier value	0
	[15]	System Clock Select	0 Direct 1 PLL output	'0'
	[14]	Pixel Clock Select	0 Direct 1 PLL output	'0'
	[13]	Inverted Pixel Clock	0 not inverted 1 inverted	'0'
	[12]	Output disable DIS_PIXCLK	0 internal pixelclock (output) 1 external pixelclock (input), high-Z output	'1'
	[11]	reserved test operation <sup>b</sup>	0 normal operation 1 core clock output on pin SPB_TST	'0'
	[10:0]	Pixel clock prescaler (DIV x)	[10:0] pixelclock prescaler value	0

**Table 1-1:** CU registers

Register	Bit	Function	Description	Reset Value
ClkPdR	0	PP/PE Pixel engine clock enable	0 = disable, 1 = enable	'0'
	1	PP/MAU clock enable	0 = disable, 1 = enable	'0'
	2	PP/MCP clock enable	0 = disable, 1 = enable	'0'
	3	AAF clock enable	0 = disable, 1 = enable	'0'
	4	DIPA clock enable	0 = disable, 1 = enable	'0'
	5	VIS clock enable	0 = disable, 1 = enable	'0'
	6	SDC clock enable	0 = disable, 1 = enable	'0'
	7	CCFL clock enable	0 = disable, 1 = enable	'0'
	8	SPB clock enable	0 = disable, 1 = enable	'0'
	9	ULB clock enable	0 = disable, 1 = enable	'0'
	10	GPU clock enable	0 = disable, 1 = enable	'0'
	11	PLL enable	0 = power down , 1 = run mode	'0'
	12	VIC clock invert (Jasmine)	0 = not inverted, 1 = inverted	'0'
		PLL Lock (Lavender)	0 = unlocked, 1 = locked (read only) <sup>c</sup>	
	13	reserved	-	'0'
	14	PLL Lock (Jasmine)	0 = unlocked, 1 = locked (read only) <sup>b</sup>	'0'
	15	Global HW Reset	0 = run mode, 1 = reset (write only) <sup>d</sup>	'0'
	[31:24]	Chip Id	0 = Lavender, 1 = Jasmine (read only)	-

a.RCLK is mapped on MODE[3] at Lavender.

b.Only applicable for Jasmine

c.Normally all register bits are read-write. As the PLL lock bit is status information only, no write access is possible to it. The lock bit is for test operation only and should not be used.

d.Read access results always in a value of '0'. Writing '1' starts global HW Reset function, writing '0' releases reset.



## 2 APLL Specification

This information is for implemented APLL - U1PN741A. Output range is given for APLL output directly, not for the divider outputs.

The APLL macro has an operating supply voltage (VDD) of 2.5 V  $\pm$  0.25 V. The oscillation guaranteed frequency range, maximum output frequency range and operating junction temperature range of the APLL are shown in the table below.

**Table 2-1:** APLL Specifications

<i>Operating Junction Temperature (Tj)</i>	-40 to 125 deg. C
<i>Voltage supply (VDDI)</i>	2.5 V $\pm$ 0.25 V
<i>Oscillation guaranteed frequency range</i>	120 to 208.4 MHz
<i>Maximum output frequency range<sup>a</sup></i>	5.77 to 598.1 MHz
<i>Input Frequency range</i>	12 to 160 MHz

a. Range in which oscillation may be possible.

**Table 2-2:** APLL Characteristics for guaranteed design range

Input [MHz]	FB divider	Output [MHz]	Phase Skew [ps]	Duty [%]	Lock Up Time [us]	Jitter [ps]	Variations in output cycle [ps]	Power consumption [mW]
25	8	200	444 -448	100.5 87.9	70	176 -142	+130 -134	2.45
20 <sup>a</sup>	8	160	540 -520	102.6 94	100	232 -168	+64 -170	1.9
13.217 <sup>b</sup>	10	132.17	1200 -1360	104.4 99.33	500	420 -246	+234 -278	1.88
12 <sup>b</sup>	10	120	1334 -1466	111.1 100.7	25	760 -560	480 -560	1.793
40 <sup>b</sup>	4	160	640 -700	105.9 101.5	20	350 -190	238 -304	1.147
23.5 <sup>b</sup>	8	188	740 -940	110.6 97.2	65	208 -162	160 -182	2.387
33 <sup>b</sup>	6	198	560 -540	100.4 88	165	172 -140	148 -180	2.397
160 <sup>b</sup>	1	160	240 -150	101.2 98.9	12	190 -140	110 -140	1.147
50 <sup>b</sup>	4	200	340 -280	98.5 87.7	26	148 -140	96 -120	2.45
20 <sup>b</sup>	10	200	3600 -4200	109.5 87.5	88	560 -360	420 -480	2.45

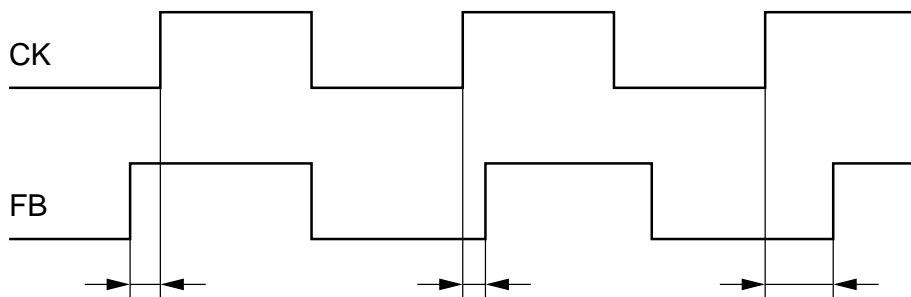
a. Operating temperature -20 ... 90 deg. C, operating voltage 2.5 V  $\pm$  0.15 V.

b. Operating temperature -40 ... 125 deg. C.

## 2.1 Definitions

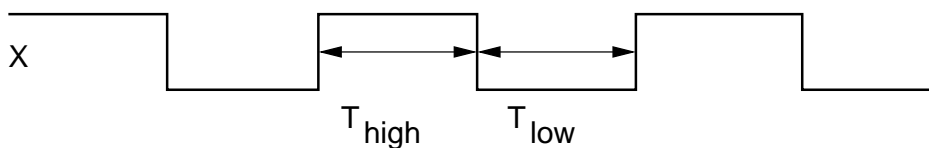
### 2.1.1 Phase Skew

Maximum phase differences between reference clock and feed back clock measured by the CK pin of the PLL and the feedback clock measured by the FB pin.



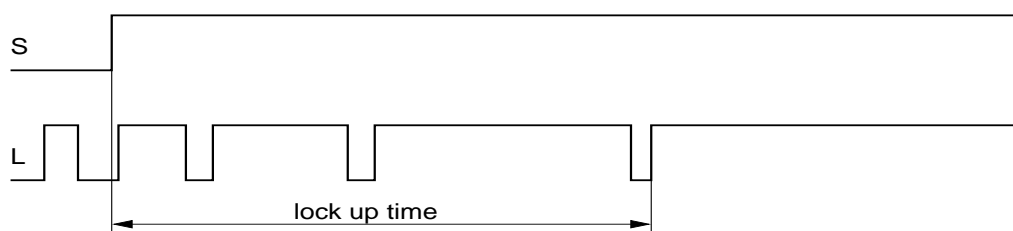
### 2.1.2 Duty

Duty is the maximum and minimum values indicated by the ratio of a high pulse width to a low pulse width ( $T_{low}$  to  $T_{high}$ ) in one cycle of the output clock measured by the X pin of the PLL.



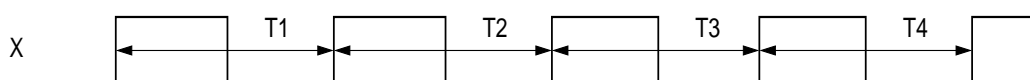
### 2.1.3 Lock Up Time

Lock up time is the time period that starts when the S pin of the PLL changes from 0 to 1 and ends when the PLL is locked.



### 2.1.4 Jitter

Jitter is the maximum and minimum values for cycle variations ( $T_2-T_1$ ,  $T_3-T_2$  and  $T_4-T_3$ ) in two continuous cycles measured by the X pin of the PLL.



### 2.1.5 Variation in Output Cycle

The max./min. time periods from the rising edge of the output clock to the next rising edge measured by the X pin of the PLL. Observation points: 1400. The max./min values in  $T_1$ ,  $T_2$ ,  $T_3$  and  $T_4$  in above figure.

### 2.1.6 Maximum Power Consumption

This is the maximum power consumption of the PLL when PLL is in locked state. The power dissipated by external dividers is not included into this amount.

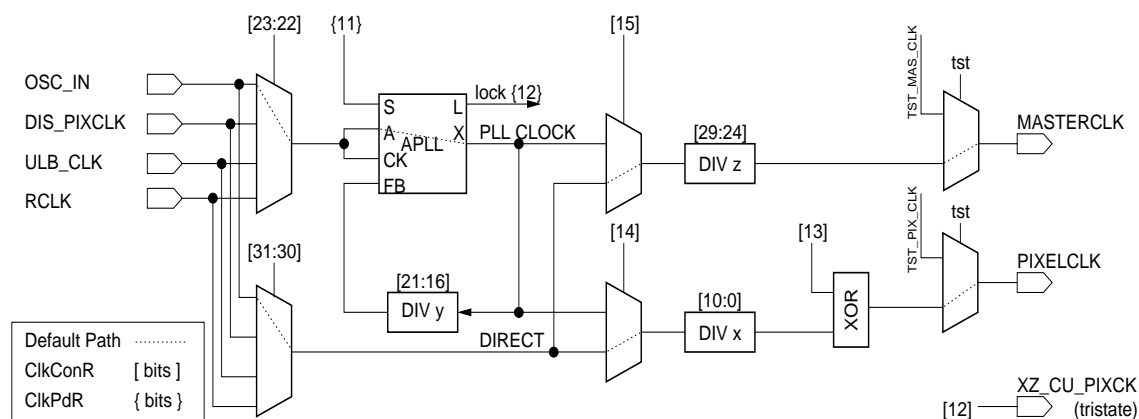
## 2.2 Usage Instructions

- Input the clock of crystal oscillator level into the CK pin of the PLL. Variations in the input clock directly affects the PLL output, leading to unconformity to the specifications.
- In addition to the normal power supply, the PLL has a power supply for VC0 (pin name: APLL\_AVDD, APLL\_AVSS). The VC0 power supply should be separate from other power supplies.

## 3 Clock Setup and Configuration

### 3.1 Configurable Circuitry

Clock configuration can be easily done by setting up both registers ClkConR (Clock Configuration Register) and ClkPdR (Clock Power Down Register). ClkConR mainly controls the setup of multiplexers and clock dividers in the main part of CU, which is shown in figure 3-1.



**Figure 3-1:** Clock routing and configuration bits

ClkPdR decides which clocks should be enabled and distributed to the appropriate modules, listed in table 2-1. During change of ClkConR all enable bits in ClkPdR[10:0] have to be turned off to attain spike protection.

**Table 3-1:** Mapping of clock sources, outputs and their enable bits

ClkPdR Control Bit	Clock Source	Clock Output
0 1 2 3	Master <sup>a</sup>	Pixel Processor (PP)
0	Master	PP: Pixel Engine
1	Master	PP: Memory Access Unit
2	Master	PP: Memory Copy
3	Master	Anti Aliasing Filter
4	Master	Direct/Indirect Physical Access
5	Master	Video Interface
5	Video Scaler (VSC_CLKV)	Video Interface
6	Master	SDRAM Controller
7	Master	Cold Cathode Fluorescence Light
8	MCU Bus (ULB_CLK)	Serial Peripheral Bus
9	Master	User Logic Bus Interface and Command Controller
9	MCU Bus (ULB_CLK)	User Logic Bus Interface

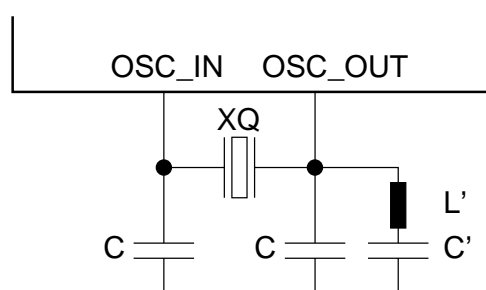
**Table 3-1:** Mapping of clock sources, outputs and their enable bits

ClkPdR Control Bit	Clock Source	Clock Output
10	Master	Graphic Processing Unit
10	Pixel Clock <sup>a</sup>	Graphic Processing Unit

a.Master and Pixel clock could be derived from one of four possible clock inputs (OSC\_IN, DIS\_PIXCLK, ULB\_CLK, RCLK/MODE[3]) with or without use of the PLL.

All clocks except VSC\_CLKV can be used as Master or Pixel clock source. VSC\_CLKV is for video interface dedicated use only.

There are no special requirements for the quartz crystal parameters. At the case of overtone oscillation additional external inductance  $L'=5-10\mu\text{H}$  and capacitor  $C'=10\text{pF}$  are needed. Two capacitors  $C=22\text{pF}$  have to be connected from the OSC pins to ground in any case. Figure 3-2 shows recommendet circuit.

**Figure 3-2:** Crystal connection between pins OSC\_IN and OSC\_OUT

Without a crystal oscillator connected (e.g. external oscillator) the clock has to fed in the OSC\_IN pin.

## 3.2 Clock Unit Programming Sequence

This section gives a recommendation for the sequence for GDC clock configuration. In general the Clock Unit registers should be configured before all other GDC setup information.

- Apply stable clock and do a hardware reset.
- Write ClkConR for the required mode. Clock gates are disabled per reset default.
- Switch on PLL and optionally apply software reset (Set bits [11] and [15] in ClkPdR).
- Clear software reset bit (Optional, only if set before).
- Wait until APLL has stabilized and locked (lock-up time)<sup>1</sup>  
Polling of lock bit is optional and *not* sufficient that PLL locking process has finished. This signal is for Fujitsu test purpose only. APLL lock-up state is reached if Lock bit becomes stable '1'. This is guaranteed after specified PLL lock-up time of 500us.
- Set required clock enable bits to open the clock gates.

1.The lock up time measured from PLL start (CLKPDR\_RUN='1') to lock state (CLKPDR\_LCK=fixed '1').

An example sequence for this procedure is listed below:

```
/* CU control information */
G0CLKPDR = 0x00008800; // SW reset and APLL enable
G0CLKPDR = 0x00000800; // release SW reset, clock gates are still closed
G0CLKCR  = 0x010E8001; // MASTER=XTAL*15/2, PIXCLK=XTAL/2 not inverted output
G0CLKPDR = 0x00000EF1; // enable GPU, ULB, CCFL, SDC, VIC, DIPA, PE
```

**Figure 3-3:** Clock configuration procedure with reset

### 3.3 Application Notes

The Clock Unit provides an internally synchronized reset signal to all GDC components. Therefore it's necessary to have a stable clock applied to the OSC\_IN pin during RESETX is low and/or at least after release of RESETX. Otherwise the internal circuitry is not initialized properly or clock unstability after reset release can cause malfunction.

With the direct clock source it's possible to use a external ULB\_CLK from the MCU or RCLK as clock source for almost all internal GDC components. The APLL is not able to handle jitter/variations in input clock.

If the GDC should operate in single clock mode over ULB\_CLK driven by the MCU, ULB\_CLK and OSC\_IN have to be bridged. In any case a clock has to feed in OSC\_IN pin, otherwise the reset state would not be left.

If system or pixel clock divider are initialized with an even value tis results in an odd divider value (value interpreted +1). In this situation the duty of the output clock is not even 1:1. Most important this is for low values. In case of not even duty the high duration is smaller than the low duration. Following table lists clock divider and duty relationship.

**Table 3-2:** Clock division and resulting duty

Setup Value	Divider	Duty
0	1	1/1
1	2	1/1
2	3	2/1
3	4	1/1
4	5	3/2
5	6	1/1
6	7	4/3
...		

---

## ***B-2 User Logic Bus Controller (ULB)***





# 1 Functional description

## 1.1 ULB functions

The “User Logic Bus Interface” (ULB) provides an interface to host MCU (MB91360 series). It is responsible for data exchange between MCU and the graphic display controllers (GDC) Lavender or Jasmine<sup>1</sup>. The communication between MCU and the display controller is register based and all display controller registers are mapped in the MCU address space.

The task of ULB is to organise write- or read accesses to different display controller components, including ULB itself, depending on a given address. For read accesses the ULB multiplexes data streams from other components and has to control the amount of needed bus wait states using MCU’s ULB\_RDY pin.

The ULB provides also a command- and data interface to so called ‘execution devices’ (Pixel Processor (PP) and Indirect Memory Access Unit (IPA)). These execution devices are responsible for drawing command execution or for the handling of SDRAM access commands. The data transfer to and from execution devices is always FIFO buffered. In order to ensure a rapid data transfer between MCU and display controller ULB contains one input and one output FIFO which are mapped to certain memory addresses within the display controllers memory space. ULB controls the MCU port of these FIFOs (write for input FIFO and read for output FIFO) while the ports to execution devices is controlled by the device itself.

The command interface has a two stage pipeline so command and register preparation is possible during command execution of previous command. Most commands can have an infinite amount of processing data. The FIFOs help to reduce the number of bus wait states.

Additionally to FIFO data exchange direct access to SDRAM and to initialisation registers is managed by ULB. This direct SDRAM access maps the SDRAM physical into MCUs address space. Drawing functions use a logical address mode for SDRAM access. Due to this direct (and also indirect via FIFOs) physical access to SDRAM it can also be used to store user data and not only layer data (bitmaps, drawing results). For direct SDRAM access (frame buffer or video RAM) the display controller internal SDRAM bus arbitration influences the MCU command execution time directly via ULB bus wait states via ULB\_RDY signal. Therefore longer access times should be calculated for this kind of memory access. ULB is able to handle memory or register access operations concurrently to normal command execution (FIFO based).

Beside normal data and command read and write operation ULB supports also DMA flow control for full automatic (without CPU activity) data transfer from MCU to display controller or vice versa. Only one direction at one time is supported because only one MCU-DMA channel is utilised. Also an interrupt controlled data flow based on programmable FIFO flags is possible. In both cases the ULB bus is used for data transfer.

ULB offers a set of some special registers controlling the display controller behaviour or show the state of the controller with respect to MCU:

- Flag-Register
- Flag-Behaviour-Register
- Interrupt-Mask-Register
- Interrupt-Level-Register
- DMA-Control-Register
- Command Register
- SDRAM access settings
- Debug Registers

ULB also provides an interrupt controller that can be programmed very flexible. Every flag can cause an interrupt (controllable by Interrupt-Mask-Register) and for Jasmine it is selectable whether the interrupt for a certain flag is level or edge triggered. Furthermore for every flag the programmer can determine whether the flag is allowed to be reset by hardware or not (static or dynamic flag behaviour).

---

1. The term ‘display controller’ is used as generic name for Lavender and Jasmine and covers both devices.

The ULB internal DMA controller is able to use all DMA modes supported by MB91360 series. It operates together with input or output FIFO and uses programmable limits. In demand mode the controller calculates the amount of data to transfer by its own. In other modes the programmer has to ensure that enough space is available in input FIFO so that no data loss can occur.

Because of different clock frequencies for ULB bus and display controller core clock an important ULB function is the data synchronisation between these two clock domains. A asynchronous interface is offered by ULB which allows independent clocks for ULB and core as long as ULB clock is equal or slower than core clock.

In order to adjust the display controller's operation mode so called 'mode pins' (MODE[3:0]) are used:

- The display controller can also act as an 16 Bit device from MCU's point of view. In this case ULB converts write data from 16 to 32 Bit and read data from 32 back to 16 Bit in order to hide interface parameters to internal display controller components. The data mode can be set by MODE[2].
- Up to four display controllers can join one chip select signal. So it is necessary to set the controller 'number' by MODE[1:0]. ULB takes care about correct address decoding depending on this 'number'.
- In order to allow flexible PCB layout some signals can be inverted and can be set to tristate. For Jasmine the inversion of ULB\_RDY is controlled by MODE[3]. For more details about signal settings see chapter 1.3.4.

In short terms the main functions of ULB are:

- MCU (User Logic Bus) bus control inclusive wait state handling for read access via ULB\_RDY pin
- Address decoding and control of other display controller components
- Data buffering and synchronisation between different clock domains
- Command decoding and execution control
- Flag handling
- Programmable interrupt handling
- Programmable DMA based input/output FIFO control
- 16/32 Bit conversion for writing and reading

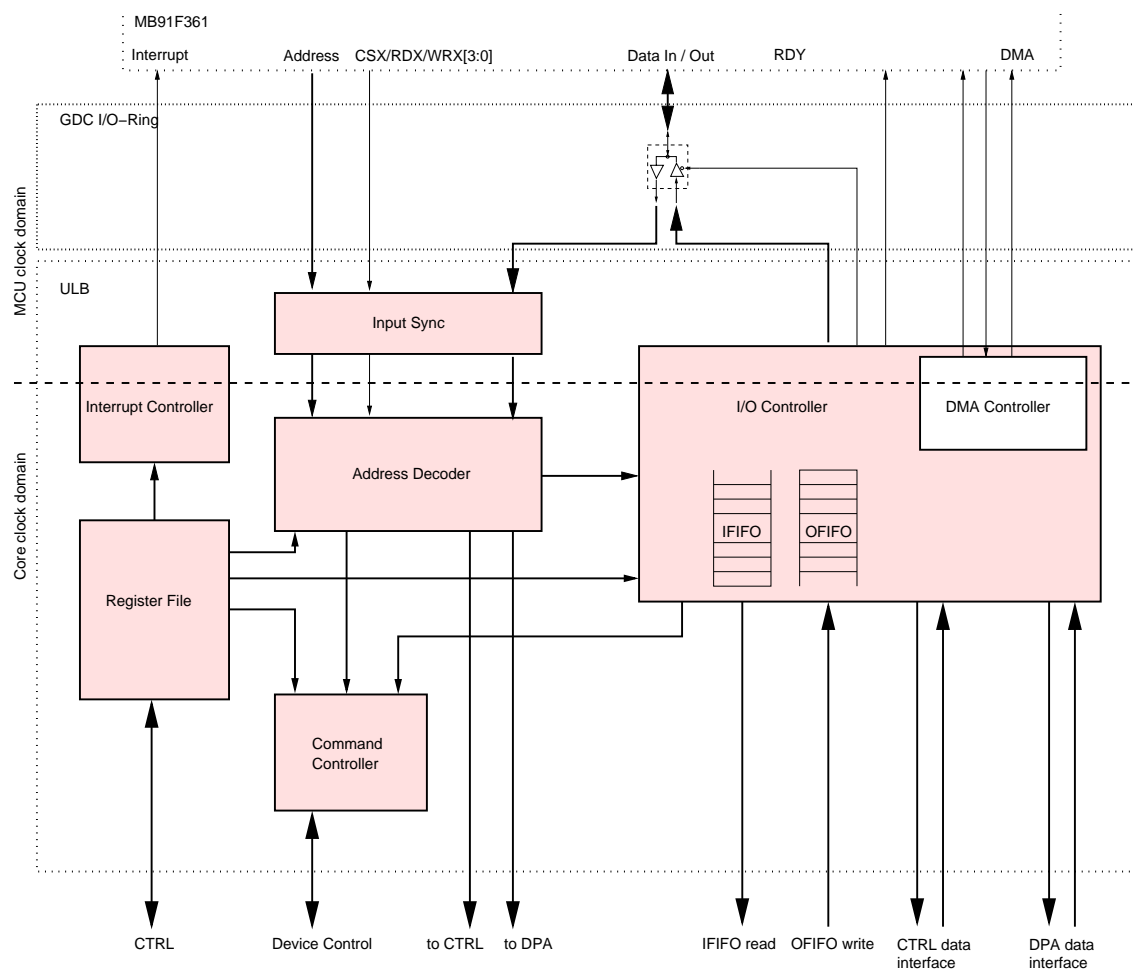
## 1.2 ULB overview

Figure 1-1 shows the block diagram of ULB top level design.

Table 1-1 gives an overview on main functions of ULB top level modules. Important modules will be explained in more detail in the following chapters.

**Table 1-1:** Top level modules of ULB

Name	Main functions
Input Sync	<ul style="list-style-type: none"> <li>• Synchronization for write signals (MCU -&gt; display controller)</li> </ul>
Interrupt Controller (IC)	<ul style="list-style-type: none"> <li>• Flag register management</li> <li>• Management of different interrupt sources</li> <li>• User definable interrupt source masking and trigger condition</li> <li>• Interrupt signal generation (Jasmine: programmable length for edge request)</li> </ul>
Address Decoder (AD)	<ul style="list-style-type: none"> <li>• Handling of other display controllers using the same chip select signal</li> <li>• Address decoding and calculation for direct SDRAM access</li> <li>• Address segmentation management</li> <li>• Control of MCU data I/O as result of address decoding</li> <li>• Activation of Command Control, Register Control Unit (CTRL) and Direct Physical Memory Access Units (DPA) as result of address decoding</li> </ul>



**Figure 1-1:** Block diagram for top level of ULB

**Table 1-1:** Top level modules of ULB

Name	Main functions
Command Controller (CC)	<ul style="list-style-type: none"> <li>Command decoding</li> <li>Management of command execution</li> <li>Condition decoding for control command execution</li> </ul>
I/O controller (IOC)	<ul style="list-style-type: none"> <li>Read/write control for data part of user logic bus</li> <li>Access control and status signal generation for data FIFOs and registers for MCU and display controller site</li> <li>Clock domain synchronization for read data bus (display controller-&gt;MCU)</li> <li>Bus multiplexing for display controller read data busses</li> </ul>
DMA Controller (DMAC)	<ul style="list-style-type: none"> <li>DMA flow control for MCU site of input and output FIFO</li> </ul>
Register File (RF)	<ul style="list-style-type: none"> <li>Storing of user definable parameters for ULB</li> </ul>

## 1.3 Signal synchronisation between MCU and display controller

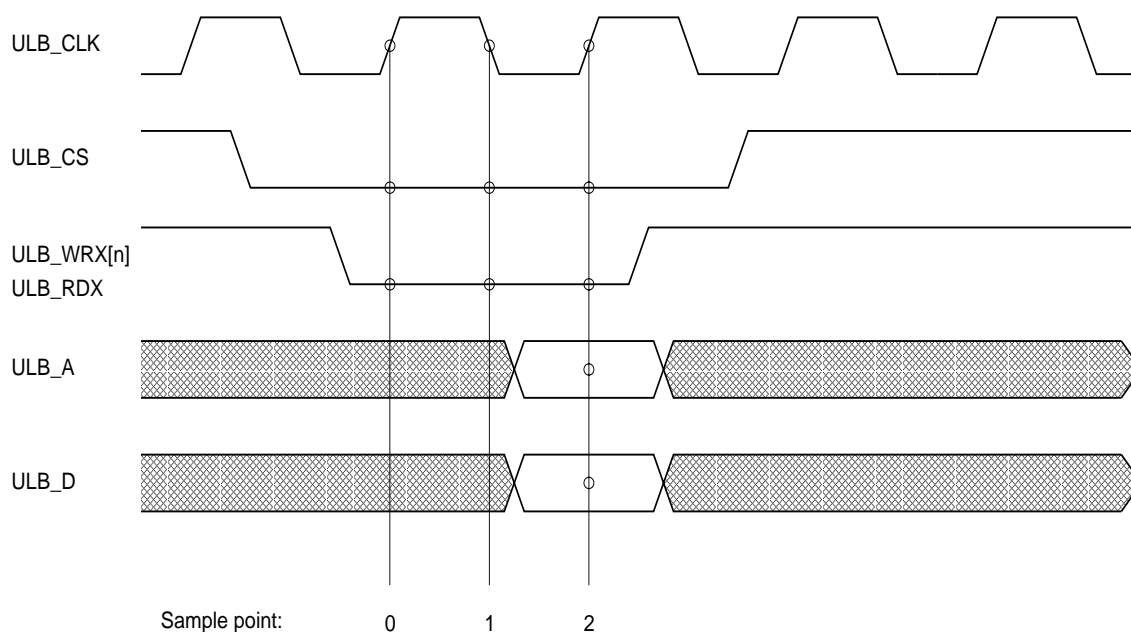
### 1.3.1 Write synchronization for Lavender and Jasmine

The data flow inside the ULB is divided in write- and read-direction. These data directions are completely independent. That's why there are two synchronisation points for ULB bus signals; one for write direction and one for read direction.

The first synchronisation point is located inside 'Input Sync' and is responsible for all ULB signals coming from MCU (ULB\_CSX, ULB\_WRX, ULB\_RDX, ULB\_A, D\_in). Other incoming MCU signals for DMA are used directly inside the DMA controller which is partly clocked by ULB clock.

*Note: For a correct operation of Input Synchronization ULB clock has to be equal or slower than display controller core clock. Note that all tolerances for clocks should be taken into consideration.*

In Jasmine a configurable sample behaviour for input signals has been introduced. In order to avoid interferences from external bus different sample modes can be set. It can be chosen between four different sample modes. Each mode combines one or more out of three sample points distributed over one bus cycle. Figure 1-2 shows these sample points. Note that the sample points are only valid for control signals



**Figure 1-2:** Bus cycle sample points for Jasmine

(ULB\_CSX, ULB\_WRX, ULB\_RDX); address and data bus are only sampled at point 2 (see figure 1-2). The sampling is equal for read and write accesses.

The sample mode can be set in register IFCTRL\_SMODE (see also table 2-1); table 1-2 shows all possible settings and sample modes.

**Table 1-2:** Control signal sample modes for Jasmine

Mode	IFCTRL_SMODE[1:0]	Comment
3 point mode [default]	00	all three sample points have to have the same value
2 of 3 point mode	01	two out of three sample points have to have the same value (majority decision)

**Table 1-2:** Control signal sample modes for Jasmine

Mode	IFCTRL_SMODE[1:0]	Comment
2 point mode	10	sample point 1 and 2 (see figure 1-2) have to have the same value
1 point mode	11	sample point 1 determines result value

Beside different sample modes Jasmine's input synchronisation circuit contains priority logic to distinguish between read or write access in the case that both control signals (ULB\_RDX and ULB\_WRX[ n ]) were detected. Because the I/O controller (ULB read path) may detect a read access the ULB\_RDX signal for read access has the highest priority.

In Lavender a different input synchronisation circuit is implemented where always one sample point is used<sup>1</sup>.

### 1.3.2 Read synchronization

For Lavender and Jasmine the ULB\_RDY signal is gated by ULB\_CSX. This means that the ULB\_RDY signal goes immediately high after ULB\_CSX= '1' has been detected. It can not be ensured that correct data are transferred to MCU in this case.

A protection against wrong tristate bus control signal switching is implemented in ULB. The bus driver is only valid if ULB\_CSX= '0' and ULB\_RDX= '0'. In all other cases ULB data bus is not driven.

Additionally to the described safety mechanisms which are implemented in both devices (Lavender and Jasmine) Jasmine has a programmable timeout for ULB\_RDY signal generation. Therefore a counter is implemented which is loaded with the value set in register RDYTO\_RDYTO[ 7 : 0 ] at the beginning of a bus read cycle<sup>2</sup>. If the read value does not arrive within the counter runtime<sup>3</sup> ULB\_RDY signal is forced to '1' and the MCU can finish the bus read cycle. Note that in this case data transferred to MCU are **not** valid. The occurrence of a ULB\_RDY timeout is reflected in the flag FLNOM\_ERDY (ULB\_RDY timeout error; see also flag description in appendix) which has been implemented in Jasmine. Additionally to the error flag the address where the timeout error occurred is stored in register RDYADDR\_ADDR[ 20 : 0 ] in order to allow an application running on MCU an error handling.

The ULB\_RDY timeout counter can be disabled by turning RDYTO\_RDTOEN off (set to '0').

In regular operation mode a ULB\_RDY timeout can only occur if no memory bandwidth can be allocated by the device handling the read request. Because the command execution is FIFO buffered (see also chapter 1.1) and a read access from FIFO always returns a value within short time no timeout error can occur in normal command execution. Only a direct mapped memory read access (DPA read access; see also chapter 1.4.3) in conjunction with very limited bandwidth may cause a ULB\_RDY timeout error in normal operation.

Beside this reason for timeout error in normal operation also disturbed bus transfers or bad signal integrity may cause a timeout error.

### 1.3.3 DMA and interrupt signal synchronization

The DMA input and output signals (ULB\_DREQ, ULB\_DACK, ULB\_DSTP) are used/generated in the DMA Controller which is partly operating at ULB clock. Because these signals are generated in this part no synchronisation is necessary. The signals influencing the DMA signal generation have to be synchronized from Lavender/Jasmine core to ULB domain. For more details regarding DMA see chapter 1.7.

Another signal which needs to be transferred from Lavender/Jasmine core to ULB clock domain is the interrupt request signal (ULB\_INTRQ). In chapter 1.6 a detailed description of interrupt signal generation is given. The synchronisation of the interrupt request signal is different between Lavender and Jasmine. Jas-

1. For Lavender sample point 1 is used to catch signals within ULB clock domain. Afterwards the caught signals will be sampled with core clock. As a result the real sample point depends on clock ration between ULB and core clock.
2. A ULB bus read cycle is detected when ULB\_CSX=0 and ULB\_RDX=0.
3. The runtime ends when the counter reached value '0'.

mine supports level and edge triggered interrupt requests with programmable edge length while Lavender is only supporting level triggered interrupt requests.

In level triggered interrupt mode the interrupt request is only reset if the flag which causes this interrupt is also reset. Because of software flag reset the request signal is stable for a very long time and no internal handshake mechanism is needed.

In difference to level triggered interrupt edge triggered interrupt reacts on a rising edge of a flag. This edge causes a pulse of programmable length on interrupt request signal.

### 1.3.4 Output signal configuration

In order to allow flexible system integration both display controllers allow configuration for some output signals to MCU. This includes an option to signal inversion and a tristate control in order to allow external pull up resistors.

If the tristate control is enabled the according pin drives tristate ('Z') instead of high level ('1') while low level ('0') is driven in any case (emulated open drain).

Table 1-3 contains the settings for all configurable signals (ULB\_DREQ, ULB\_DSTP and ULB\_INTRQ)

**Table 1-3:** Control for feedback signals

Signal	Setting	Lavender	Jasmine
ULB_DREQ	tristate	Register: DMAFLAG_TRI	Register: IFCTRL_DRTRI (1: tristate)
	invert	Register: DMAFLAG_INV	Register: IFCTRL_DRINV (1: invert)
ULB_DSTP	tristate	Register: DMAFLAG_TRI	Register: IFCTRL_DSTRI (1: tristate)
	invert	Register: DMAFLAG_INV	Register: IFCTRL_DSINV (1: invert)
ULB_INTRQ	tristate	Register: DMAFLAG_TRI	Register: IFCTRL_INTTRI (1: tristate)
	invert	Register: DMAFLAG_INV	Register: IFCTRL_INTINV (1: invert)
ULB_RDY	tristate	no control possible	Pin: RDY_TRIEN (1: tristate)
	invert	no control possible	Pin: MODE[3] (1: invert)

valid for both display controllers. In Jasmine additionally ULB\_RDY can be controlled by special pins<sup>1</sup>.

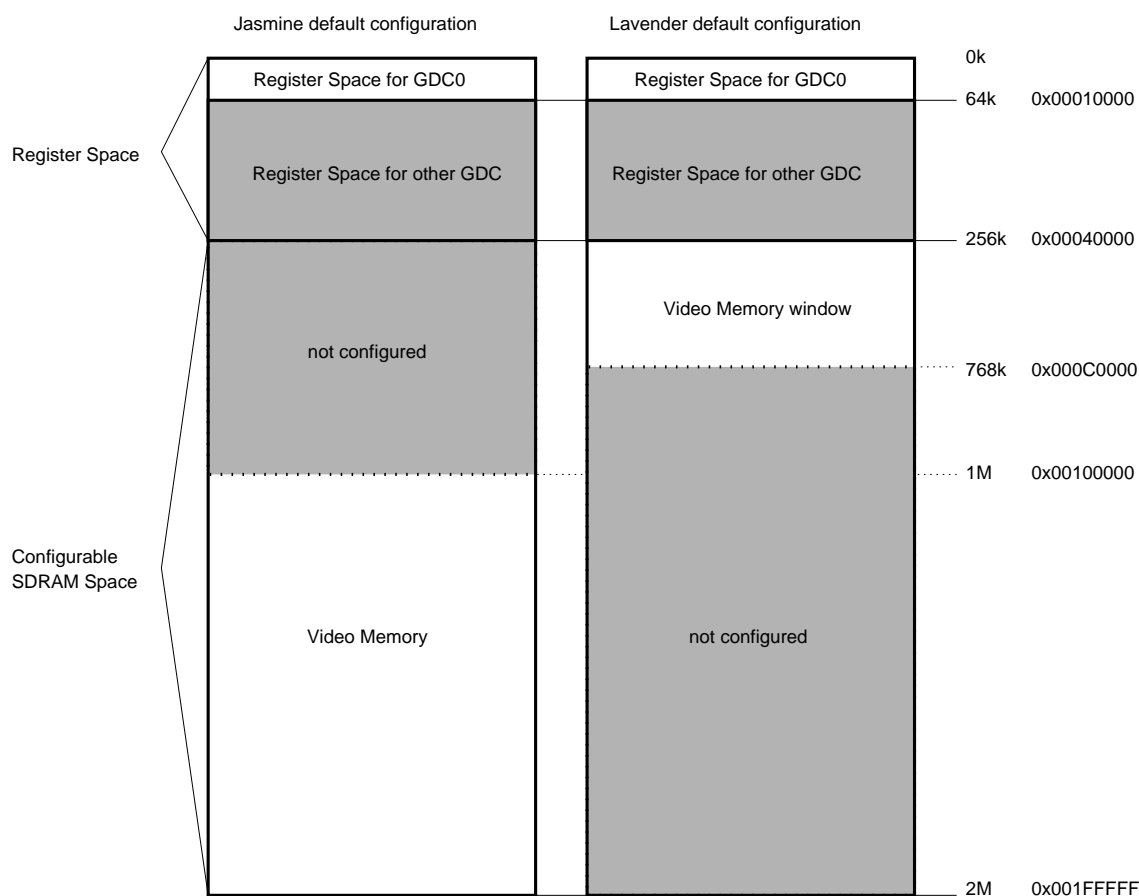
Note that there are differences in controlling the signal behaviour between Lavender and Jasmine. While in Lavender the signals ULB\_DREQ, ULB\_DSTP and ULB\_INTRQ are controlled together with two register Bits for tristate and inversion (DMAFLAG\_TRI and DMAFLAG\_INV) in Jasmine every signal has its own control (see table 1-3).

## 1.4 Address decoding

### 1.4.1 Overview

The useable address space for display controller chip select signal (ULB\_CSX) is  $2^{21}$  Byte (2 MByte) and the available space is divided into one configuration register space where all configuration registers are located and one SDRAM space where SDRAM windows can be established and accessed. This space is configurable. Figure 1-3 shows the address space with the default settings for SDRAM space of Lavender and Jasmine.

1. A register based control is not possible because read accesses would potentially not work in this case or the MCU may hang with a wrong signal.



**Figure 1-3:** Address space for Lavender and Jasmine with default configuration

Jasmine and Lavender support up to four devices for one chip select (ULB\_CSX) signal. Also a mixed environment with different display controllers is possible. Register and SDRAM space are used by all connected display controllers. Figure 1-4 shows a possible scenario for four display controllers which treats only as an example. Many other configurations for SDRAM space are possible while register space is fixed configured.

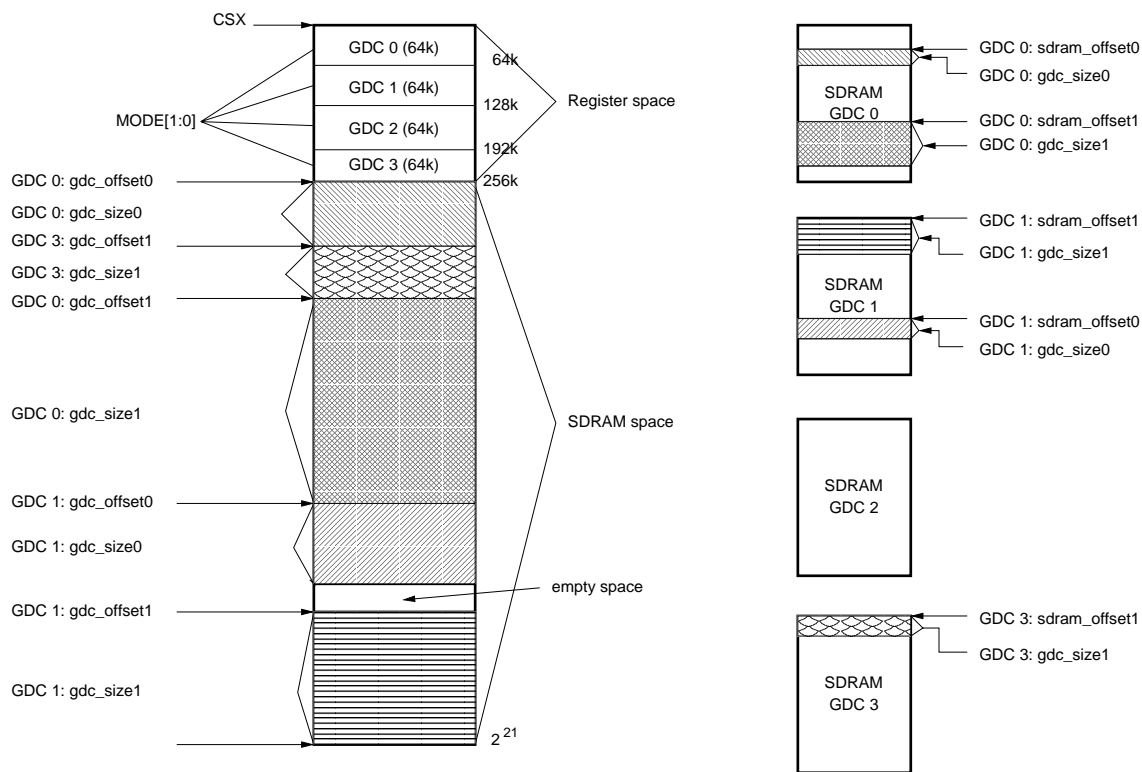
### 1.4.2 Register space

The size and location of configuration registers for every display controller is fixed. The size is set to 64 kByte for every display controller and the location is specified by Mode Pins (MODE[1:0]) as described in Table 1-4.

**Table 1-4:** Address ranges for register space of different display controllers

Controller number	MODE[1:0]	Address range
0	00	00000h...0FFFFh
1	01	10000h...1FFFFh
2	10	20000h...2FFFFh
3	11	30000h...3FFFFh

Table 1-5 shows the register space for one display controller decoded by ULB address decoder. This decoder has a built in priority for the case of overlapping address areas. One display controller reserves the register space for other controllers. Because the address decoder has a decoding priority it is **not** possible to overlay the register space for other controllers with SDRAM windows.



**Figure 1-4:** Address space example for four graphic display controller (GDC) devices

Within the upper part of register space (address range 0xFC00-0xFFFF) for one display controller a reserved area is located. This area is needed for internal and/or external devices with the same ULB\_CSX signal as the display controller which have their own address decoders. Internal devices using this area are currently the Clock Unit (CU) and the Serial Peripheral Bus driver (SPB) (see also table 1-5).

Note that Lavender register space is compatible to Jasmine register space. Only new registers for new functions were added or not needed registers were removed but the same function can be found on the same registers. There are only a few exceptions (see register list for more details).

**Table 1-5:** Register address space for GDC

Priority	Address range	ULB addressed component	Target component/Register
1	0x0000	ULB	Command register
	0x0004		Input FIFO
	0x0008		Output FIFO
	0x000C		Flag register (normal access) <sup>a</sup>
	0x0010		Flag register (reset access) <sup>a</sup>
	0x0014		Flag register (set access) <sup>a</sup>
	0x0018		Interrupt mask (normal access) <sup>a</sup>
	0x001C		Interrupt mask (reset access) <sup>a</sup>
	0x0020		Interrupt mask (set access) <sup>a</sup>



**Table 1-5:** Register address space for GDC

Priority	Address range	ULB addressed component	Target component/Register
2	0xFC00 - 0xFC07	Reserved area	CU
	0xFC08 - 0xFCFF		empty <sup>c</sup>
	0xFD00 - 0xFD0F		SPB
	0xFD10 - 0xFFFF		empty <sup>c</sup>
3	0x0024 - 0x009F	CTRL	ULB
	0x0100 - 0x0217		SDC
	0x1000 - 0x1243		GPU - LDR
	0x1300 - 0x1383		GPU - MDR
	0x1400 - 0x140F		GPU - MTX <sup>b</sup>
	Lavender: 0x2000 - 0x23FF Jasmine: 0x2000 - 0x27FF		GPU - CLUT
	0x2800 - 0x2BFF		GPU - GAMMA <sup>b</sup>
	0x3000 - 0x3263		GPU - DIR
	0x3270 - 0x3273		GPU - SDC
	0x4000 - 0x403B		VIC
	0x4100 - 0x4133		PP
	0x4200 - 0x420B		DIPA
	0x4400 - 0x4407		CCFL
	0x4500 - 0x450F		AAF
4	other display controllers (see table 1-4)	-	empty <sup>c</sup>
5	SDRAM window1 range	DPA	SDRAM
6	SDRAM window0 range	DPA	SDRAM
7	Empty area within SDRAM space	-	empty <sup>c</sup>

a. Refer to section 1.6 for an explanation of register access types.

b. Jasmine only

c. A special 'empty' signal is generated because the ULB is not allowed to drive the data bus for a read access inside an empty area while the I/O Controller detects a valid read access.

### 1.4.3 SDRAM space

For direct SDRAM access two independent windows can be set within ULB for one display controller. The term 'window' means that a part of the SDRAM memory can be blended into the MCU address space. Windows can be set up by defining window parameters within ULB's register space. Before an established window is useable the SDRAM space has to be enabled. The following parameters can be used to set up SDRAM windows (see also figure 1-4):

- WND0F0\_OFF, WND0F1\_OFF to define an offset in MCU address space
- WND0SZ0\_SIZE, WND0SZ1\_SIZE to define SDRAM window size for MCU and SDRAM address space
- WND0SD0\_OFF, WND0SD1\_OFF to define SDRAM offset

All these parameters are explained in detail in table 2-1.

As displayed in figure 1-3 2 MByte ( $2^{21}$  Byte) - 256 kByte (register space) = 1.75 MByte can be used for both windows within MCU address space. This address space is equal in Lavender and Jasmine while the available SDRAM memory differs according to table 1-6. For Jasmine it is possible to map the entire SDRAM memory into ULB's SDRAM space in order to have linear access to SDRAM. For Lavender only parts of SDRAM can be mapped to MCU address space at one given time but a dynamic reconfiguration is possible.

**Table 1-6: SDRAM memory for Lavender and Jasmine**

Display Controller	SDRAM type	SDRAM size
Lavender	external	8 MByte (64 MBit)
Jasmine	internal	1 MByte (8 MBit)

All to one chip select signal connected display controllers share the available SDRAM space (1.75 MByte). There is no additional restriction about the size or order of SDRAM windows of different controllers. Figure 1-4 gives just one example but many more configurations are possible.

Gaps between SDRAM windows for a particular display controller are handled by ULB as well as SDRAM windows of other controllers. Both possibilities can not be distinguished by the address decoder and they produce an empty space hit which means that no data are driven for read access and a write access is simply ignored.

Within one display controller overlapping SDRAM windows are allowed; this is controlled by address decoder priority according to table 1-5. This is **not** true for SDRAM windows from different controllers. Write access is possible; the value is written to all SDRAM windows mapped to this address. Read access is **not** possible and **can damage display controller or MCU** because no ULB bus driving control is available between different display controllers.

*Note: There is no control for overlapping windows of more than one display controller within SDRAM space. Reading from such an area can damage display controller or MCU.*

With help of SDRAM windows the SDRAM memory can be written or read with physical SDRAM addresses via a display controller component called 'DPA' (Direct Physical Memory Access). In difference to this addressing mode all drawing functions use a logical address (pixel coordinates with (layer, x, y)). The SDRAM controller (SDC) within the display controller maps the logical pixel coordinates to a physical SDRAM address. This mapping is block based<sup>1</sup> and differs between Lavender and Jasmine since it depends on SDRAM architecture. A picture previously generated by Pixel Processor (PP) with help of drawing/bit-map commands can not be read back in a linear manner because the logical to physical address mapping has to be performed in order to get the right physical address for a given logical address (layer, x, y). Also for writing graphics which should be displayed by the Graphic Processing Unit (GPU) of a display controller the logical to physical mapping has to be taken into account. The easiest and most portable way (between Lavender and Jasmine) to write or read to/from logical address space is to use PP commands. In this case the mapping is done automatically and controller independent in hardware.

The physical SDRAM windows can be used for any kind of data as long they do not present a picture which should be displayed by GPU. Within one SDRAM window a linear access to the data is possible.

Because the start address of each layer can be set in physical SDRAM addresses it is possible to divide the SDRAM memory between layer and user data. Note that there is no layer overrun control implemented in hardware within the display controller.

While the SDRAM memory windows are mapped into the MCU address space the access is basically organised as normal register access. But there are some important differences compared to normal register

1. See SDC specification for details.

access. In difference to a register which can be exclusively accessed via its address in MCU address space the SDRAM is a resource that has to be shared between different display controller components. For one special SDRAM address many different ways to access it<sup>1</sup> can be found. Therefore the SDRAM controller (SDC) arbitrates the accesses with different priority.

For the direct SDRAM access does this mean that an undefined access time depending on current system load occurs. For read accesses the ULB\_RDY signal is used for synchronisation between MCU and display controller while for write accesses a flag (RDPA) in flag register is used. The application has to poll this flag (wait as long as flag is zero<sup>2</sup>) in order to synchronize write accesses to SDRAM window. As a side effect **no** DMA access is possible for writing to SDRAM window. Since for reading the ULB\_RDY signal is used DMA access is possible.

While for Jasmine every kind of access (word (32 Bit), halfword (16 Bit) and byte (8 Bit)) is possible for reading and writing to/from SDRAM windows for Lavender the read access is limited to word access (32 Bit).

Table 1-7 sums up the access types and synchronization methods for both display controllers.

**Table 1-7:** Access type and synchronisation for Lavender and Jasmine

Access	Item	Lavender	Jasmine
<b>Read</b>	Synchronisation	ULB_RDY signal	ULB_RDY signal
	Access	word	word, halfword, byte
	DMA	possible	possible
<b>Write</b>	Synchronisation	Flag: FLNOM_RDPA	Flag: FLNOM_RDPA
	Access	word, halfword, byte	word, halfword, byte
	DMA	-	-

Because of the SDRAM access arbitration and write restrictions (flag polling) direct physical SDRAM access is not the best method to access SDRAM memory.

For logical pixel access it is recommended to use command based and FIFO buffered drawing and access functions (see chapter 1.5).

A better way to access the SDRAM memory in physical addressing mode is the indirect SDRAM memory access via IPA component of display controller. This access type has some advantages compared to direct access:

- linear access to SDRAM without window limits
- FIFO buffered transfer for effective SDRAM access
- address auto increment for reading and writing (burst SDRAM access)
- DMA is possible for reading and writing

#### 1.4.4 Display controller bus access types (word, halfword, byte)

The MB91xxxx MCUs support three different bus access types for writing data from MCU to display controller.

- Word access: write 32Bit
- Halfword access: write 16Bit
- Byte access: write 8Bit

For a more detailed description see MB91360 series hardware manual.

1. Beside the described direct physical access the Pixel Processor can access with logical addresses and also an indirect physical access (command based) via IPA is possible.  
2. Make sure this flag is set to dynamic behaviour.

Table 1-8 lists the supported bus access types for different address areas and data modes for Lavender and

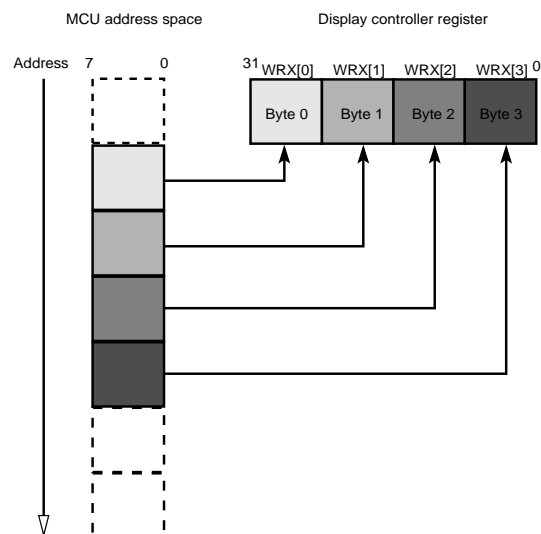
**Table 1-8: Bus access types for Lavender and Jasmine**

Address area	Lavender		Jasmine	
	32Bit data mode (MODE[2]=1)	16Bit data mode (MODE[2]=0)	32Bit data mode (MODE[2]=1)	16Bit data mode (MODE[2]=0)
Input FIFO	word	word	word	word
Output FIFO	word	word	word	word
Register space <sup>a</sup>	word, halfword, byte	word, halfword, byte	word, halfword, byte	word, halfword, byte
SDRAM space	word	word	word, halfword, byte	word, halfword, byte

a. without input and output FIFO

Jasmine. Note that for input and output FIFO only word access is allowed. Partial access can not be supported because every write/read access increments internal pointers.

Figure 1-5 shows the address to register mapping within display controller. Note that MB91xxxx MCUs use



**Figure 1-5: MCU address to display controller register mapping**

Big Endian byte order which means that byte0 is the MSB byte of a 32Bit word. Every byte has its own valid signal (ULB\_WRX[ 3 : 0 ]) and a combination of valid signals determines the access type for a write access.

For reading always the whole bus width depending on data mode (32- or 16Bit) is transferred from display controller to MCU. The MCU selects the right part for current read instruction internally.

#### 1.4.5 Display controller data modes (32/16 Bit interface)

MB91xxxx MCUs are able to communicate with devices with different bit sizes for data bus (32-, 16- and 8Bit). Normally Lavender and Jasmine were designed to act as 32Bit devices. In this mode optimal performance can be achieved and no internal data mapping is necessary.

For special purposes both display controller can act as 16Bit devices. With help of this mode parts of data bus can be made available as general purpose I/O pins or it may be possible to connect the display controller to a 16Bit MCU<sup>1</sup>.

Beside the chip select signal configuration the 16 Bit mode has to be selected for the address space of GDC(s) within MCU (see MB91360 series hardware manual) and with MODE[2] pin (1: 32Bit mode; 0: 16Bit mode) at the display controller. All data transfer is done at MSB side of data bus (Pins: ULB\_D[31:16]). For write accesses does this mean that the bus signals ULB\_WRX[2:3] are not used and should be set to '1' in 16Bit mode. Table 1-9 gives an overview on used parts of data bus and used control signals.

**Table 1-9:** Signal connection for data modes

Signal	Data mode	
	32 Bit data mode (MODE[2]=1)	16 Bit data mode (MODE[2]=0)
Data bus	ULB_D[31:0]	ULB_D[31:16]
Write control signals	ULB_WRX[3:0]	ULB_WRX[1:0] set ULB_WRX[3:2] to '1' (pull up)
Read control signals	ULB_RDY	ULB_RDY

In 16Bit mode display controller address space (register and SDRAM space) is accessible as in 32Bit mode. All bus access types described in chapter 1.4.4 can be used transparently.

For word write accesses a MB91xxx MCU splits the word access into two half word accesses with consecutive addresses (increment two). Since the display controller is able to handle half word and byte accesses (see chapter 1.4.4) 16Bit mode can be supported.

ULB contains a 16Bit to 32Bit converter that is responsible for adapting the 16Bit bus access to internal 32Bit bus structure. For the special case of FIFO write access, where only word access is allowed not only in 16Bit data mode, a special circuit was included which collects data for FIFO write access.

For read accesses ULB contains an additional converter which is responsible for converting the internal 32Bit bus structure to external 16Bit bus structure. For reading from output FIFO a special circuit is in charge of reading only once from FIFO and store the value for second read.

## 1.5 Command decoding and execution

### 1.5.1 Command and data interface to MCU

The display controller command interface to MCU consists of the following main parts:

- Command register where command instruction and command coded parameters can be written and read
- Input- and output FIFO for data exchange between MCU and display controller
- Command dependent flags set by command controller
- Debug register (read only) in order to watch command controller behaviour

Writing to command register has to be synchronised with command execution within display controller. In chapter 1.5.3 the structure and function of ULB's command decoder will be explained in detail. From programmers' point of view a flag (FLNOM\_CWEN) shows when command controller is able to receive a new command. An application should poll FLNOM\_CWEN flag before writing a new command or if interrupt controlled flow is implemented it should write commands only after FLNOM\_CWEN causes an interrupt.

If the display controller 'Application Programming Interface (API)' is used, flag polling is done automatically before writing a new command. Figure 1-6 shows the draw line command within API as an example.

```
word GDC_CMD_DwLn(dword line_col) {
    while (!G0FLNOM_CWEN); /* Wait until Command Write Enable flag is set */
}
```

1. This MCU should have the same bus interface as MB91xxx or it has to be adapted with help of glue logic.

```

    G0LINECOL = line_col;
    G0CMD = GDC_CMD_DWLINE;
    return (0);
};

```

**Figure 1-6:** Draw line function as an example for command synchronisation

The function in figure 1-6 writes only the command and command dependent register settings (in this case the line colour) to display controller, data transfer is done afterwards with help of FIFOs.

```

word GDC_FIFO_INP(dword *p_arr, word pcnt, byte dma_ena) {
    if (dma_ena == 0) {
        for (;pcnt>0;pcnt--) {
            while (G0FLNOM_IFF != 0) G0FLRST_IFF = 1;
            G0IFIFO = *p_arr++;
        }
    } else {
        while ((DMACA0 & 0x80000000) != 0);
        DMASAO = (IO_LWORD)p_arr; // source address
        DMACA0 = DMACA0 | 0x80000000 | 0x0E100000 | pcnt; // DMA operation enable
        G0DMAFLAG_EN = 1;
    }
    return (0);
};

```

**Figure 1-7:** Function to put data to display controllers input FIFO

To write data to input FIFO another API function (GDC\_FIFO\_INP) shown in figure 1-7 is used. GDC\_FIFO\_INP takes a pointer to an array with values<sup>1</sup> which should be written to FIFO together with data count. Note that data count is not limited to FIFO size, it can have any size. Before data can be written to FIFO (register: G0IFIFO) the API function polls the full flag of input FIFO in order to avoid FIFO overflow. The general flow for command execution and programming is discussed in chapter 1.5.2. Optional the transfer can be controlled by DMA (parameter dma\_ena). See chapter 1.7 for more details about DMA and its handling within an application.

Both display controllers (Lavender and Jasmine) contains one input and one output FIFO. The sizes of these FIFOs are different in display controllers and listed in table 1-10.

**Table 1-10:** FIFO sizes for Lavender and Jasmine

FIFO	Size for Lavender	Size for Jasmine
Input FIFO	128 words	64 words
Output FIFO	128 words	64 words

Every FIFO has a set of flags which allow a flow control by an application. A detailed description of FIFO flags can be found in flag description located in appendix. Beside full and empty flag also two programmable limits (one lower and one upper limit) is implemented in both display controllers. These limits can be used to perform an action within an application based on a certain FIFO load. This is often more efficient than polling the full or empty flag for every single data word. Note that every flag can cause an interrupt (for details see chapter 1.6) so that FIFO flags can be used for interrupt based flow control.

A general rule for input FIFO should be to check whether the free space in FIFO is large enough for the amount of data words which have to be written.

Before reading from output FIFO the application should check whether enough data are available in output FIFO. This can be done by polling the FIFO flags or by generating an interrupt.

As a replacement of application based flow control (polling or interrupt) DMA can be used to write data to input FIFO or read data from output FIFO. For DMA the hardware takes care about FIFO load but an ap-

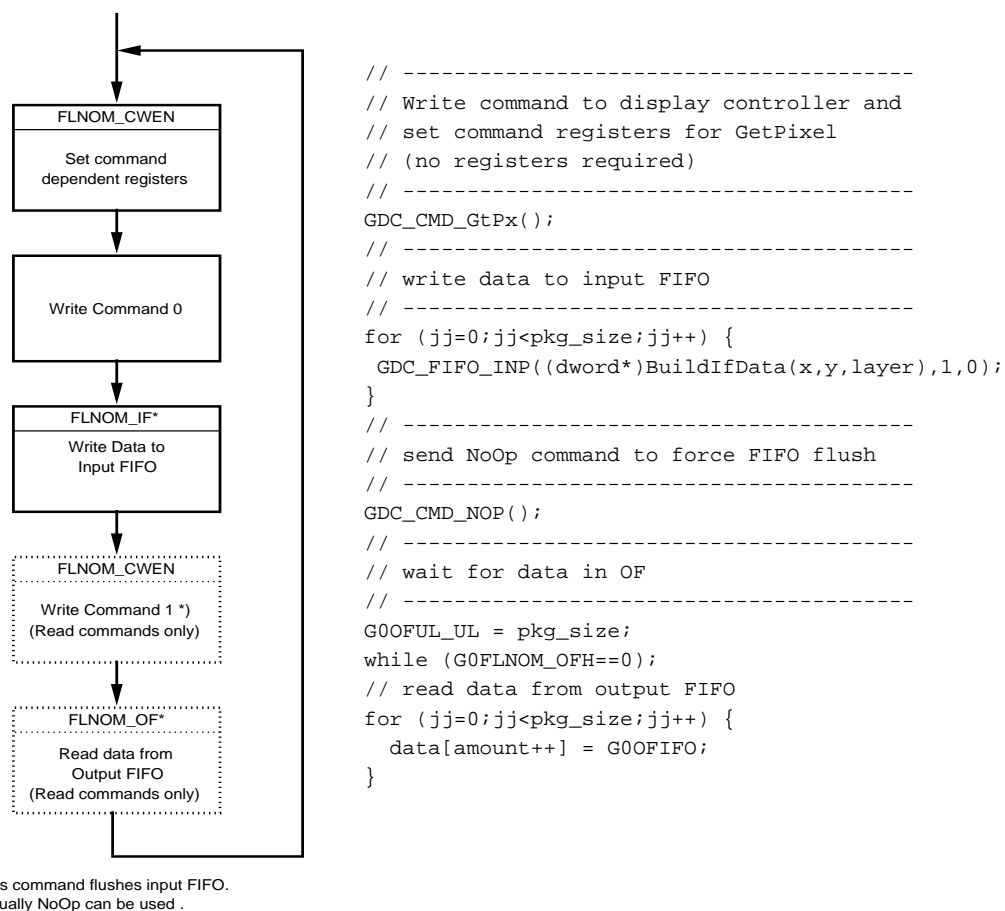
1. 'dword' means 32 Bit unsigned.

plication has to prepare the data first and can not generate them 'on the fly'. The ULB DMA controller is described in detail in chapter 1.7.

Command dependent flags and debug registers are listed and described in chapter 1.5.5.

### 1.5.2 Command execution and programming

Figure 1-8 shows a flow chart of display controller command execution and a C-example of a display controller command (GetPixel). For this example the C-API for Lavender and Jasmine is used. It is described in a separate manual.



**Figure 1-8:** Command flow for display controller commands with an example for GetPixel

Before a new command can be written to command register the flag FLNOM\_CWEN should be checked. Therefore the flag register can simply be polled or an interrupt can be generated as result from the rising edge of this flag. See chapter 1.6 for more details about flag and interrupt handling.

Afterwards command buffered registers can be written as well as the new command code itself. Note that at this time the previous command may be still running (see also chapter 1.5.3 for a detailed discussion) and also the previously buffered register contents is used.

Command buffered registers contain settings for a certain command (e.g. line colour for the DwLine command) which have to be synchronized to command change. The time of command change is determined by hardware so that it is necessary to store values for next command in a separate register (e.g. new line colour for next DwLine command right after the first one).

A list of command buffered registers can be found in the command description located in appendix. Also a detailed register description is placed there.

The next step within command execution is to send data to input FIFO. The application has to take care that no FIFO overrun occurs. Therefore it should watch the FIFO flags either by polling or via interrupt (see also chapter 1.5.1). An application can compute input data with its own speed, the display controller waits for new data if input FIFO runs empty.

Note that some commands use the input FIFO and internal buffers for collecting data before processing them. Therefore an application can not expect that all data are processed immediately. This can lead to an incomplete drawing of figures or to an incomplete memory transfer depending on running command. The input FIFO and all internal buffers will be forced to flush when the next command is sent to display controller.

The amount of collected data depends on executed command and can be programmed in separate registers. These registers are REQCNT for all Pixel Processor commands and DIPAI for physical memory access commands (PutPA and GetPA).

Write commands can be executed with any amount of data in this way. The next command can be written to command register after all data have been sent to input FIFO (see figure 1-8). This causes the termination of previous command inclusive input FIFO and internal buffer flushing and the start of the new command itself. If only a buffer flushing should be performed and no more commands have to be executed a NoOp command can be used for this purpose.

For read commands (GetPixel, XChPixel)<sup>1</sup> the scenario is a bit more complex. Due to the buffer usage of input FIFO and internal buffers the display controller does not process all data. As a result not all expected read data can be found in output FIFO. A read loop over the expected amount of data would hang because not all data are available in FIFO.

A possibility to force the display controller to fill the output FIFO with the expected amount of data is to send a new command to command register. This can be the next command that has to be executed (including command buffered registers as described above) or in order to keep the code simple and readable a NoOp command (see also figure 1-8).

Another possibility is to set the register REQCNT which controls the buffered data amount to '0'. This forces a single transfer for every written data word. Note that this setting decreases performance compared to larger values of REQCNT because no burst accesses to video memory are possible.

For data amounts larger than output FIFO size a division of data stream into packages is necessary. For each of these packages the command flow according to figure 1-8 should be applied.

If an application wants to transfer a **complete package at once** without checking FIFO load for every data word for instance via DMA or within an interrupt controlled application it is possible that not all data appear in output FIFO and the initialized limit is not reached. Even if the next command was sent after GetPixel or XChPixel which is normally suitable to flush input FIFO data flow blocking is not escaped.

Note that this behaviour does **not** occur if output FIFO is read with **flag polling for every data word** because the amount of words in output FIFO falls below the limit FIFOSIZE-REQCNT-1 at a certain time. GetPA is not affected because it uses other registers than REQCNT for block size calculation as already mentioned.

A possibility to utilize the full output FIFO size is to ensure that always REQCNT+1 words can be placed in output FIFO. This limits the maximal package size (number of words to transfer for one output FIFO fill) for a given REQCNT. The maximal package size can be calculated according to (1).

$$\text{pkg\_size} \leq \text{trunc}\left(\frac{\text{FIFOSIZE}}{\text{REQCNT} + 1}\right) \times (\text{REQCNT} + 1) \quad (1)$$

The function 'trunc' in (1) means that only the natural part of this fraction should be taken for calculation. The parameter 'FIFOSIZE' is the size of output FIFO according to table 1-10. Note that 'pkg\_size' is the maximal package size, sizes smaller than the calculated size can be used.

Figure 1-9 shows an example how to read back large data amounts from display controller. The shown function reads back a complete bitmap defined by a pointer to the structure 'S\_BM' ('bm').

The example for automatic calculation of pkg\_size is given to show whole FIFO and command usage mechanism and to point out differences between Jasmine and Lavender implementations.

In order to calculate required package size GetBM reads back the register REQCNT<sup>2</sup>, determines the correct output FIFO size with help of chip ID and calculates the required package size according to (1).

For every data package the command execution flow described above is used. It is nested into a double loop

---

1. GetPA is also a read command but it is a so called 'finite' command which gets the number of data to transfer within a special register. This command is not concerned by this discussion.  
2. G0REQCNT addresses the REQCNT register for GDC with number '0' (see chapter 1.4).



for every bitmap dimension. The reading of data from output FIFO is only done if the calculated package size (pkg\_size) has been reached or if the bitmap has been completely finished (last package). As flush command for input FIFO after writing a whole package the next GetPixel command is sent to display controller.

```
// Struct for bitmap data
struct S_BM {
    byte layer;          // layer to operate on
    word x,y,dx,dy;      // coordinates: offsets x,y; length dx,dy
    dword *data;         // bitmap data from x,y to (x+dx-1,y+dy-1)
                        // amount: dx*dy
};

/* Read back function with optimal package sizes */
dword GetBM(struct S_BM *bm) {
    dword amount;
    word  x, y;
    byte  pkg_cnt, pkg_size, reqcnt, of_size;

    // calculate optimal package size for given request count
    reqcnt  = G0REQCNT + 1;          // minimum data amount for block transfer
    of_size = G0CLKPDR_ID? 64: 128; // FIFO size for Jasmine : Lavender
    pkg_size = (of_size / reqcnt) * reqcnt;

    // initialize data counters
    amount  = 0; // bitmap pixels accumulative
    pkg_cnt = 0; // intra package count

    GDC_CMD_GtPx(); // GDC Mode: Get Pixel
    while (!G0FLNOM_IFE); /* IFIFO should be empty that packet fits into */
    // bitmap region, picture processing loop
    for (y = 0; y < bm->dy; y++) {
        for (x = 0; x < bm->dx; x++) {
            // write address to input FIFO (relative to BM start point)
            G0IFIFO = pix_address(bm->layer, x + bm->x, y + bm->y);
            pkg_cnt++;
            // get data if pkg_size completed (or even smaller last package)
            if (pkg_cnt == pkg_size || (y == bm->dy - 1 && x == bm->dx - 1)) {
                GDC_CMD_GtPx(); // flush by sending new command
                G0OFUL_UL = pkg_cnt; // initialize block size FIFO limit
                while (G0FLNOM_OFH == 0); // wait for all data avoids OF empty polling
                while (pkg_cnt) { // receive data from OFIFO
                    bm->data[amount++] = G0OFIFO; // Set pixel in bitmap array
                    pkg_cnt--;
                } // while pkg_cnt
            } // if pkg_cnt == pkg_size
        } // x-loop
    } // y-loop
    return amount; // return number of data words in array
}
```

**Figure 1-9:** C-example for reading large data amounts from display controller

Note that for writing data to input FIFO in example from figure 1-9 no flag polling is necessary because it is known that the amount of data is not larger than FIFO size (output FIFO and input FIFO have the same size) and the input FIFO is empty at package start. Normally flag polling is necessary before writing data to input FIFO. The API function 'GDC\_FIFO\_INP' which was already described in chapter 1.5.1 automatically takes care of this issue.

In the example in figure 1-9 the package size is calculated dynamically in order to experiment with different values for REQCNT. Normally it is possible to set up REQCNT according to application needs and calculate the maximal package size offline. If an application is not dependent on reading data from output FIFO at

once (e.g. per DMA or interrupt controlled) it may be easier to read data from FIFO as soon as they appear. Figure 1-10 shows a code example where this is demonstrated.

```
// data array
dword data[SIZE];
// loop over package size
for (k=0; k < SIZE;k++){
    // wait as long as OFIFO is empty
    // make sure G0FLNOM_OFE is to dynamic!!!
    while (G0FLNOM_OFE);
    // read data into array
    data[k] = G0OFIFO;
}
```

**Figure 1-10:** C-example for reading data continuously

### 1.5.3 Structure of command controller

The ULB contains a command controller which is responsible for controlling so called 'execution devices (ED)' within display controller. These EDs are responsible for command execution and data processing. In current implementation of Lavender and Jasmine four execution devices are handled by ULB command

**Table 1-11:** Execution devices within Lavender and Jasmine

Execution device		Function
Pixel Processor (PP)	Pixel Engine (PE)	<ul style="list-style-type: none"> <li>Drawing of graphical primitives</li> <li>Drawing and RLE decompression of bitmaps</li> </ul>
	Memory Access Unit (MAU)	<ul style="list-style-type: none"> <li>Writing and reading of pixel-addressed data</li> </ul>
	Memory Copy (MCP)	<ul style="list-style-type: none"> <li>Copying of pixel-addressed blocks within SDRAM memory</li> </ul>
Physical memory access unit (DIPA)	IPA	<ul style="list-style-type: none"> <li>Writing and reading of word-addressed data via input- and output FIFO</li> </ul>

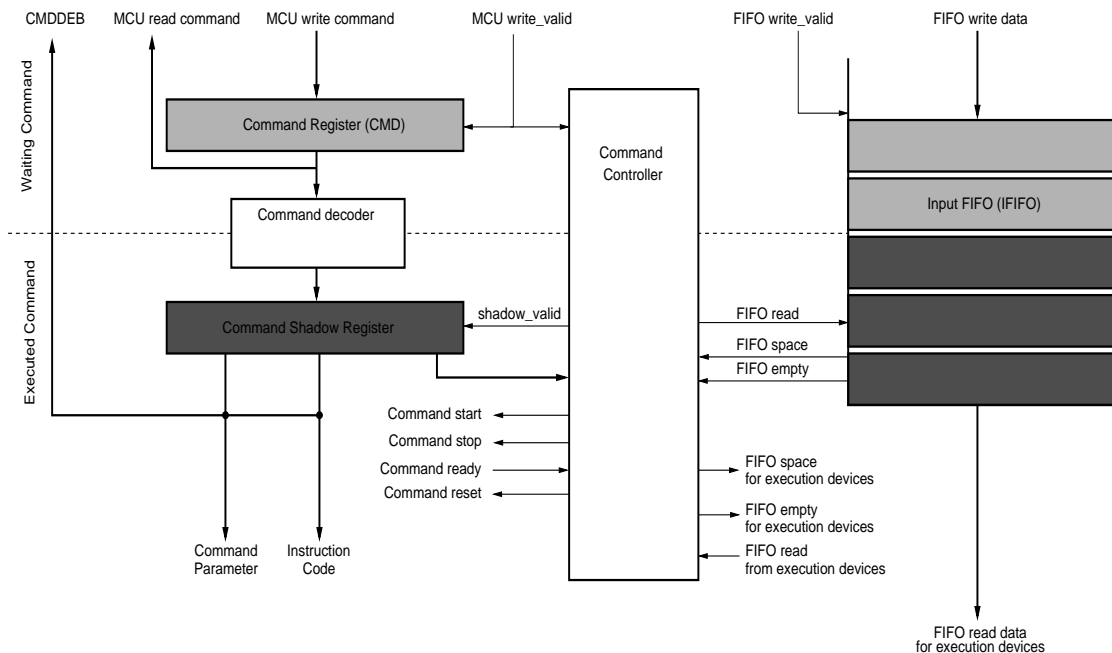
controller. An overview on execution devices and their functions is given in table 1-11. See specifications of these devices for a detailed description.

Most of display controller commands are so called 'infinite commands' which means that these commands have an unlimited number of processing data. The stop condition for infinite commands is writing a new command. Therefore a second register is needed which contains the currently executed command. This register is controlled by hardware and not writeable by MCU. Jasmine has the possibility to watch currently executed command with a read only debug register (CMDDEB; see chapter 1.5.5 for details).

Between the two command registers the Command Decoder is located. The command write time from command to shadow command register is determined by hardware because it depends on the execution state of previous command. The structure of Command execution unit within ULB is shown in figure 1-11.

In order to avoid command pipeline overflow and to implement a command flow control between display controller and MCU a flag 'command write enable' (FLNOM\_CWEN) is implemented. This flag signals that a new command can be written into command register (FLNOM\_CWEN=1). By writing the new command the old command is still executed and the pipeline is filled with two commands which can be watched from MCU by 'CMD\_WR\_EN=0'<sup>1</sup>.

1. This is only true when this flag is set to dynamic behaviour which is the reset value. See section 1.6 for a detailed explanation.



**Figure 1-11:** Command execution within ULB

Additionally the write event of a new command triggers a mechanism that is responsible for dividing data between different commands. This allows to write data for next command into input FIFO while the current command is still running and the selected execution device reads data from FIFO.

*Note: For the programmer it is important not to write data for currently executed command after writing a new command because these data would be interpreted as data for new command.*

After the currently running command has finished and all data in input FIFO have been processed the ULB command controller writes next command into shadow register and sets 'CMD\_WR\_EN=1'.

Lavender differs between read and write commands. For commands reading data from display controller via output FIFO for Lavender an additional condition has to be met before execution of a new command is started.

*Note: For a Lavender read command output FIFO has to be empty before a new command is started. In Jasmine this additional condition need not be respected and the output FIFO can collect data from different commands.*

Two different error cases regarding the ULB command controller can be observed by MCU via special error flags (FLNOM\_ECODE, FLNOM\_EDATA). A detailed description of flags regarding the command execution can be found in chapter 1.5.5.

### 1.5.4 Display controller commands

Display controller commands for Lavender and Jasmine are divided per execution type into infinite, finite and special commands. An additional subdivision can be made into write and read commands independent from execution type (finite or infinite). For a detailed command list see appendix.

Infinite command execution is processed as described in section 1.5.3. The stop condition for infinite commands is the writing of next command. Infinite commands are:

- PutPA, DwLine, DwRect, PutPixel, PutPxWd, PutPxFC, GetPixel, XChPixel, MemCP and DwPoly

In difference to infinite commands for finite commands the amount of data to be processed is fixed. It is defined by various register settings inside the execution devices or command coded parameters in case of GetPA (CMD\_PAR).

Finite commands are:

- PutBM, PutCP, PutTxtBM, PutTxtCP, GetPA

Special commands are control commands influencing the command execution itself or execution devices but they do not process data. There are two special commands:

- Software reset (SwReset) and No Operation (NoOp)

The NoOp command is included in the normal command execution pipeline as described in section 1.5.3 with the exception that no execution device is activated and no data are read from input FIFO or written to output FIFO. This command can be used to force the previous command to end data processing. Note that all data send during NoOp command is active are kept for next command.

The SwReset command treats as a synchronous reset for command execution. This command is **not** included in the normal command pipeline and is executed immediately. The Command Controller inside ULB and also all execution devices (complete PP, AAF and IPA inside DIPA) go in its initial state, the command pipeline will be emptied and the FIFOs will be reset so that all data will be lost.

*Note: During SwReset input- and output FIFO will be reset so that data loss will occur.*

Not affected by software reset are display controller parts not responsible for command execution (SDC, VIC, GPU, DPA inside DIPA, CU, CCFL and parts of ULB). These devices continue running and processing data. To reset these devices a hardware reset is necessary<sup>1</sup>.

Due to a not interruptible SDRAM access software reset is not completed in one clock and needs execution time depending on running SDRAM access for PP or IPA. Therefore the command flow control has also to be used by an application after software reset.

*Note: Also after SwReset the flag FLNOM\_CWEN has to be polled in order to ensure a save reset operation and execution of following commands.*

### 1.5.5 Registers and flags regarding command execution

In order to allow an application controlling and watching the command execution ULB contains some flags (within flag register) to provide the following functions:

- Flag: FLNOM\_CWEN  
Watching the command execution state as already described in section 1.5.3
- Flags: FLNOM\_RIPA, FLNOM\_RMCP, FLNOM\_RMAU, FLNOM\_RPE, FLNOM\_BIPA, FLNOM\_BMCP, FLNOM\_BMAU, FLNOM\_BPE  
Watching the state (busy or ready) of a specific execution device. Because all flags are high active both variants are offered by display controller to capture the needed event (busy or ready).
- Flag: FLNOM\_ECODE  
A wrong command code was sent to display controller. The wrong code is treated internally as a NoOp command which means that the command controller simply waits for a new command and no data processing is performed. All data sent to input FIFO are kept for next command as an exception for NoOp command behaviour.
- Flag: FLNOM\_EDATA  
This error flag is set when an execution device tries to read data from an empty input FIFO. This may indicate a malfunction of execution device but the interpretation of this flag heavily depends on execution device implementation.

All flags are handled as described in section 1.6 and all are able to cause an interrupt if required. A detailed flag description of all display controller flags can be found in flag description located in appendix.

---

1. A hardware reset can also be triggered by software by set register CLKPDR\_MRST to '1'.

Additionally to flags inside flag register debug registers are implemented in Lavender and Jasmine in order to watch command controller status. Table 1-12 lists these debug registers. Note that some registers are only implemented in Jasmine.

**Table 1-12:** ULB debug registers

Register		Bits	Name	Description	Device
Name	Address				
ULBDEB	0x0098	7/6 <sup>a</sup> :0	IF	Current input FIFO load (command independent)	all
		15/14 <sup>a</sup> :8	OF	Current output FIFO load	all
		23/22 <sup>a</sup> :16	IFLC	Command dependent input FIFO load	all
CMDDEB	0x009C	7:0	CMD	Currently executed command (see chapter 1.5.3)	Jasmine
		31:8	PAR	Command coded parameter for current command (GetPA only)	Jasmine

a. Lavender/Jasmine value due to different FIFO sizes

## 1.6 Flag and interrupt handling

### 1.6.1 Flag and interrupt registers

The Interrupt Controller inside ULB contains one 32 Bit Flagregister (FLNOM; address 0x000C) and one Interrupt-Mask-Register (INTNOM; address 0x0018) which allows a very flexible flag handling and interrupt generation control.

In order to avoid data inconsistencies during bit masking within flag- or interrupt-mask-register the mask process is implemented in hardware for these two registers. This helps to avoid flag changes by hardware between a read and a write access (read->mask->write back).

To distinguish between set-, reset- and direct write access different addresses are used:

- Address (FLNOM, INTNOM): normal write operation
- Address + 4 (FLRST, INTRST): reset operation (1: reset flag on specified position, 0: don't touch)
- Address + 8 (FLSET, INTSET): set operation (1: set flag on specified position; 0: don't touch)

All of these three addresses write physically to one register with three different methods. For reading all addresses return the value of the assigned register (FLNOM or INTNOM).

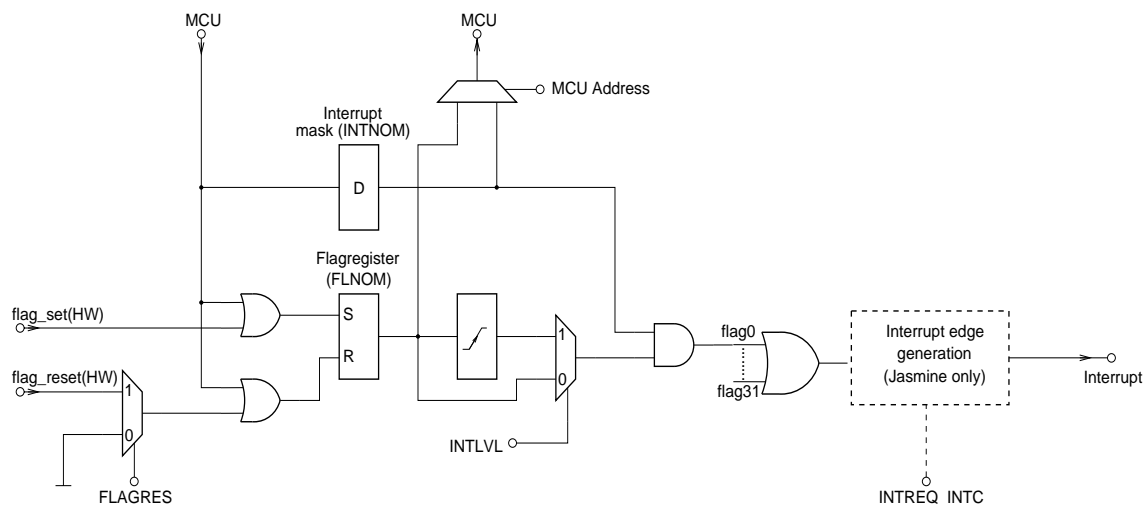
For writing all bus access types as described in chapter 1.4.4 are possible for each of these addresses.

### 1.6.2 Interrupt controller configuration

Figure 1-12 shows the basic structure of interrupt generation circuit for one flag.

The Flagregister itself is set and reset able by hard- or software. A set event by hard- or software sets the flag to '1' and a reset event sets the flag to '0'. Software flag access has a higher priority than hardware events but hardware events may be present some clock cycles around software access which is only one clock active after synchronisation.

*Note: Despite the higher priority of software access the hardware event may overwrite the software settings **after** MCU write access if the set- or reset condition for the desired flag is still true.*



**Figure 1-12:** Interrupt generation within interrupt controller for one flag

A flag set by hardware is always possible; the hardware reset can be switched off in order to avoid dynamic flag changing. This flag behaviour is referred to as 'static' flag behaviour. A forbidden hardware reset is important for a handshake implementation between display controller and MCU for instance in connection with interrupts.

If flag set and reset is allowed the flag behaviour is called 'dynamic' behaviour. In this case the desired flag simply follows the input signal. Note that flag changes may occur with core or display clock which may be higher than ULB bus clock<sup>1</sup>. Therefore it is not possible to trace some hardware events because you can not achieve a suitable sample rate. If the toggle rate for a real hardware flag is slow enough you can of course set the flag behaviour to dynamic.

Many flags represent a state which can be manipulated by software so that dynamic behaviour makes sense for these flags because they can be indirectly influenced by software. For instance the full flag for input FIFO can only change its value after writing data to input FIFO.

The flag behaviour can be set with register FLAGRES. Flag hardware reset can be turned on (1: dynamic flag behaviour) or off (0: static flag behaviour) for each flag separately.

### 1.6.3 Interrupt generation

The first operation for interrupt generation is the masking of Flagregister by Interrupt-Mask-Register. With this mechanism an application can determine which flags can cause an interrupt by simply set ('1') at the same bitposition as the flag in Interrupt-Mask-Register. Every flag can be source for an interrupt because an OR combination of flags is implemented in Interrupt Controller.

After the Flagregister a level detection circuit is implemented (see figure 1-12) which detects the rising edge of a flag. With the INTLVL register the programmer can choose for every flag whether to take the flag itself (level interrupt) or the edge detection signal with one core clock length (edge interrupt).

In level triggered interrupt mode an interrupt handshake should normally be used between MCU and Lavender/Jasmine. This means that the flag responsible for interrupt will be reset inside interrupt service routine (ISR). The ISR is only called when MCU detects an interrupt request. As a result the interrupt request is only taken back from Lavender/Jasmine after flag reset. So it is ensured that the interrupt signal is stable for many ULB clocks in level triggered mode. **Be careful with dynamic flags in this context.**

In edge triggered interrupt mode no handshake between MCU and display controller is necessary. The display controller signals the MCU with a pulse on interrupt request signal (pin ULB\_INTRQ) that a certain event occurred within display controller. The MCU can call its ISR and does not need to reset the flag which caused the interrupt if the flag behaviour is set to dynamic. If the flag behaviour is set to static and a reset access from MCU is missing no more interrupts can be generated because no more rising edges for the interesting flag occur.

1. The real sample rate is again lower since it is the time between two bus read cycles.

Jasmine contains an edge generation circuit which is responsible for a prolongation of an impulse in case of an edge interrupt. The impulse length can be set in INTREQ\_INTC within a range from 0 to 63 ULB clocks. For every edge impulse at the input of the edge generation circuit a pulse with the programmed length will be generated at output. If a level interrupt occurs the output signal follows the input signal synchronized to ULB clock domain.

Lavender does not contain an edge generation circuit. Therefore no edge interrupt is possible.

For Lavender the default value for INTLVL register is edge trigger for interrupt for all flags. Make sure to set the register INTLVL to 0x00000000 during Lavender initialisation.

For MCU interrupt programming 'H' level should be used for display controller interrupt.

### 1.6.4 Interrupt configuration example

In figure 1-13 an example configuration for display controller and MCU is given. In this example an interrupt should be activated when the input FIFO load is equal or lower than '1' (Register: G0IFUL). To activate the interrupt generation the Bit 3 of Interrupt-Mask-Register is set to '1' via the set address for this register. The interrupt trigger for display controller is set to level.

For MCU first all interrupts are turned off, global interrupt level and level for GDC-interrupt is set, the MCU interrupt trigger is also set to level to ensure a save detection. At the end the port for external interrupts is enabled, pending interrupt requests will be deleted and interrupt execution is turned on again in order to enable GDC interrupt execution. In MB91xxx hardware manual the interrupt initialisation is described in more detail.

```
// -----  
// Init GDC  
writereg G0FLAGRES, 0x3f401fff; set FIFO flags to dynamic  
writereg G0IFUL, 0x00000001; set input FIFO limits for interrupt  
writereg G0INTLVL, 0x0 ; level triggered  
writereg G0INTSET, 0x8 ; IF <= IF-low(=1)  
// -----  
// Init MCU  
andccr #0xef ; disable all interrupts  
// set interrupt level for ext. INT0  
ldi #0x14, r0; set interrupt level to 20  
ldi #ICR00, r1; load address for ext. INT0  
stb r0, @r1  
// set global interrupt level to 30  
stilm #0x1e  
// initialize external interrupt  
ldi #0x1, r0; enable only INT0  
ldi #ENIR, r1; load address for int. enable register  
stb r0, @r1  
// set interrupt request level  
ldi #0b01, r0; set 'H' level for INT0  
ldi #ELVR, r1; load address for external level register  
sth r0, @r1  
// enable interrupt ports  
ldi #0b00000001, r0; enable INT0  
ldi #PFRK, r1; port function register for interrupt 0  
stb r0, @r1  
// clear all interrupt requests  
ldi #0, r0  
ldi #EIRR, r1  
stb r0, @r1  
nop  
nop  
nop  
// enable interrupts  
orccr #0x10; set I-bit in CCR register
```

;; -----  
**Figure 1-13:** Interrupt display controller and MCU initialisation example

### 1.6.5 Display controller flags

All display controller flags are located in the Flagregister (FLNOM) inside ULB Interrupt Controller and handled as described in section 1.6.1.

All flags are explained in appendix. Note that some flags are only available for Jasmine.

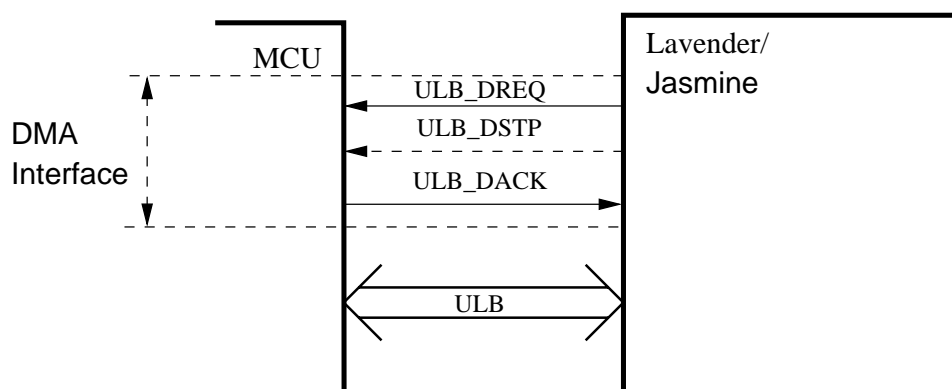
## 1.7 DMA handling

### 1.7.1 DMA interface

In order to improve data transfer speed and to automate FIFO load controlling during command execution Lavender and Jasmine contain a DMA controller which operates together with DMA-Controller (DMAC) integrated in MB91xxxx series MCUs. It is located inside ULB's I/O-Controller and handles the display controller DMA interface (GDC-DMAC).

This interface consists of additional control signals; data transfer is handled by I/O Controller as for normal MCU accesses. The DMA connection between display controller and MCU is shown in figure 1-14.

The GDC-DMAC requests a DMA transfer by setting ULB\_DREQ to '1'; the MCU acknowledges this request by set ULB\_DACK to '0' during a valid bus cycle. ULB\_DACK-pulses for other devices connected to MCU are ignored by GDC-DMAC because the ULB\_DACK signal is gated with ULB\_CSX for display controller.



**Figure 1-14:** DMA connection between display controller and MB91xxxx

In order to stop the MCU-DMAC externally by display controller the DMA stop signal (ULB\_DSTP) exist. This signal creates an error condition inside MCU-DMAC that can also cause an interrupt (see MB91xxxx manual for more details).

A better solution than using ULB\_DSTP signal for disabling MCU-DMAC is to disable DMA in MCU first by writing DMACAx\_PAUS=0 and DMACAx\_DENB=0 and turn off GDC-DMAC afterwards. The DSTP pin at MCU is not needed in this case and can be used as general purpose I/O. Note that the ULB\_DSTP pin at display controller may not be supported in future display controller releases.

### 1.7.2 DMA modes

The MCU DMA-Controller can deliver/get data in two different ways:

1. ULB\_DREQ Level triggered (Demand mode)
2. ULB\_DREQ Edge triggered (Block-, Step- and Burstmode)

For a detailed description of supported DMA modes see MB91360 series hardware manual.



### 1.7.2.1 Level triggered DMA (demand mode)

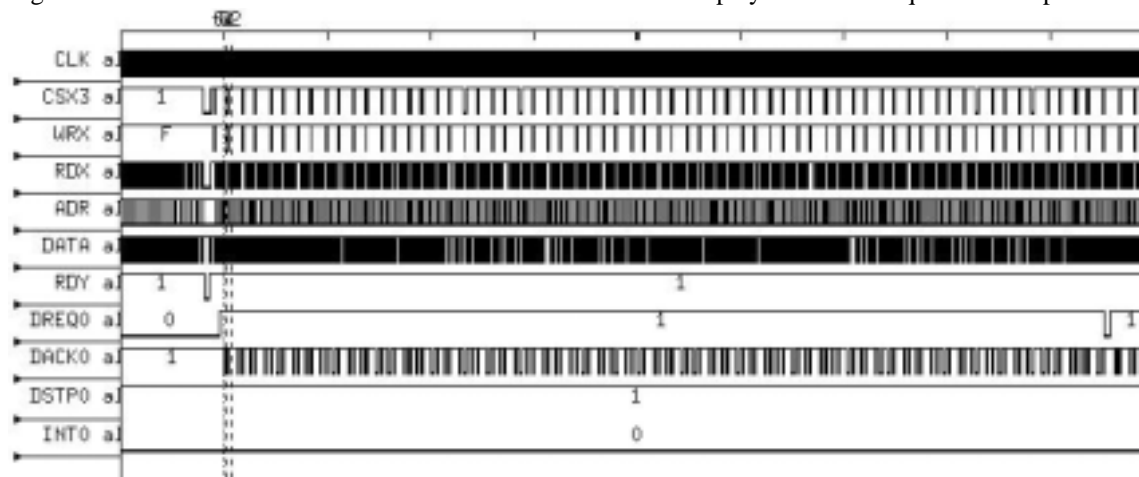
In case 1. the length of the DREQ signal defines the amount of data to be transferred.

From MCU point of view an external device has to control the length of DREQ impulses according to internal buffer sizes. It is responsible for the division of data stream while the MCU is only controlling the total amount of data to be transferred. In the special case of Lavender/Jasmine does this mean that the GDC-DMAC counts the amount of free words for input FIFO (write DMA) or the number of words in output FIFO (read DMA). Table 1-14 gives an overview on transfer sizes in different modes for display controller.

Before starting a demand transfer the GDC-DMAC tests for DMA start condition<sup>1</sup>, detects the number of words to be transferred, loads a counter with this value and counts this counter to zero. During counting ULB\_DREQ is set to active. This procedure is repeated until DMA within display controller is disabled.

The GDC-DMAC does **not** know the total amount of words to be transferred. It only tries to fill (write DMA) or to flush (read DMA) its FIFOs. At the end of a complete DMA transfer ULB\_DREQ could still be active because from display controller's point of view DMA is enabled and input FIFO needs data or output FIFO has to deliver data. After disabling DMA for display controller the ULB\_DREQ signal goes inactive.

Figure 1-15 shows the start of a write DMA demand transfer. Display controller requests one input FIFO



**Figure 1-15:** Write DMA in demand mode

fill cycle<sup>2</sup>. During this time a display controller command is active and reads data from input FIFO concurrently. After a short break the second fill cycle is requested.

### 1.7.2.2 Edge triggered DMA (block-, step-, burstmode)

In case 2. (edge triggered DMA transfer) only the rising edge of ULB\_DREQ signal is important. The amount of data to be transferred is set within MCU (see also table 1-14).

A MCU peripheral device has to ensure that the ULB\_DREQ impulse is long enough to be recognized by MCU. In case of GDC the ULB\_DREQ signal goes inactive after the MCU has acknowledged the DMA request<sup>3</sup>. Depending on MCU mode (block-, step- or burstmode) a MCU defined amount of data words is transferred to or from display controller FIFOs. The programmer has to ensure that no FIFO overflow can occur by setting up the appropriate value for input FIFO lower limit (IFDMA\_LL) or output FIFO upper limit (OFDMA\_UL) (see chapter 1.7.3 for a detailed description).

Figure 1-16 shows a write DMA transfer in block mode. The block size is set to 10 words. Despite of this Jasmine<sup>4</sup> toggles the ULB\_DREQ signal after every falling edge of ULB\_DACK signal because it does not know the MCU settings. It can not distinguish between block-, step- or burst mode.

1. For write DMA: IFDMA\_LL >= input FIFO load; for read DMA: OFDMA\_UL <= output FIFO load.
2. For Jasmine one complete fill cycle contains 64 words.
3. This is the first high to low edge of the ULB\_DACK signal combined with a valid chip select signal.
4. Lavender shows the same behaviour but this example was made with Jasmine.

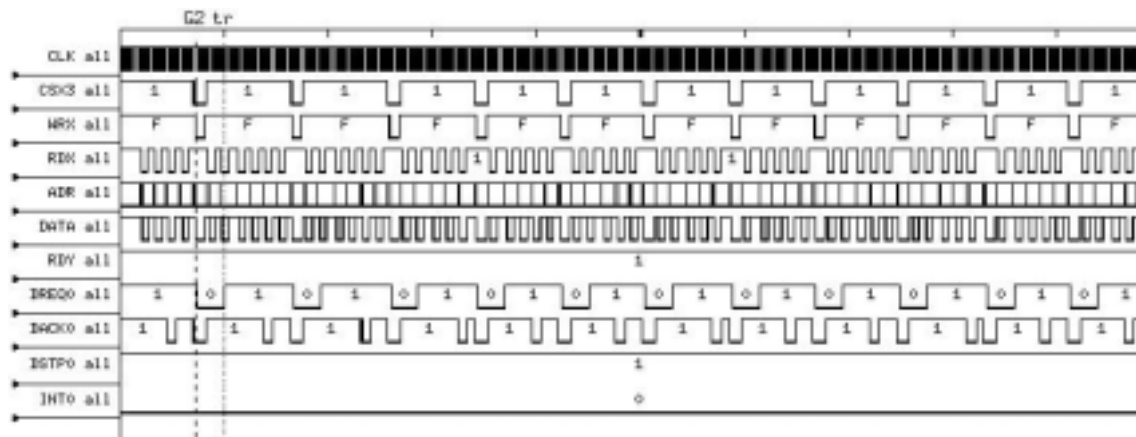


Figure 1-16: Write DMA in block mode

### 1.7.3 DMA settings

In order to use DMA feature for display controller it is necessary to set up DMA according to table 1-13.

DMAFLAG\_EN is the general DMA enable flag; if this bit is set to '0' all DMA operations are stopped. Additionally the falling edge of this flag during a running DMA transfer causes a reset of GDC-DMAC and MCU-DMAC via ULB\_DSTP signal in order to stop DMA transfer completely. This is important because the transferred data belong to a command and a running DMA transfer influences next command and its data stream which is not necessarily controlled by DMA.

Table 1-13: DMA register settings

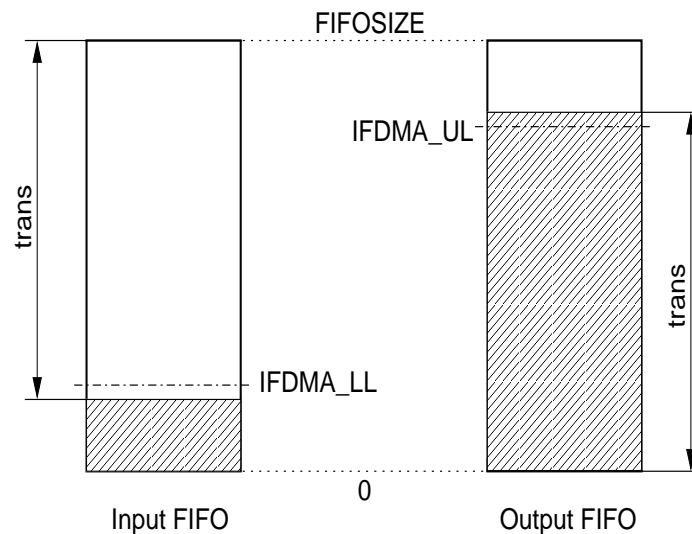
Register		Bits	Flag name	Description	Default value
Name	Address				
IFDMA	0x0088	7/6 <sup>a</sup> :0	LL	• Lower limit for DMA access to input FIFO	10
OFDMA	0x008C	23/22 <sup>a</sup> :16	UL	• Upper limit for DMA access from output FIFO	60
DMAFLAG	0x0090	12:8	DSTP	• Duration of ULB_DSTP signal in ULB clocks.	7
		2	MODE	• '1': DMA demand mode • '0': DMA block/step- or burst mode	0
		1	EN	• '1': enable DMA	0
		0	IO	• '1': use DMA for input FIFO • '0': use DMA for output FIFO	1

a. Lavender/Jasmine value due to different FIFO sizes

For DMA operation only one DMA channel is available between display controller and MCU. Therefore only one FIFO can be written or read per DMA at a given time. The programmer can select the FIFO that should be read or written with help of DMA by set DMAFLAG\_IO according to table 1-13. An additional gate with ULB\_WRX or ULB\_RDX ensures that only accesses for the selected mode are accepted.

The selected DMA mode can be selected with DMAFLAG\_MODE. See chapter 1.7.2 for more details about DMA modes.

The trigger condition for DMA start can be set separately for input (IFDMA\_LL) and output FIFO (OFDMA\_UL). It represents a FIFO load and is completely independent from flag settings according to flag



**Figure 1-17:** FIFO limits for DMA transfer

description table in appendix. Figure 1-17 shows the meaning of these limits for input and output FIFO while in table 1-14 the calculation for transfer sizes for FIFOs is listed.

In case of input FIFO the FIFO load has to be equal or smaller to meet trigger condition for DMA. For output FIFO the load has to be equal or greater to trigger DMA transfer (see also figure 1-17).

**Table 1-14:** Transfer count calculation for DMA

Mode	Input FIFO	Output FIFO
Demand mode	$\text{trans} = \text{FIFOSIZE}^a - \text{FIFO load}$	$\text{trans} = \text{FIFO load}$
Block/step- or burst mode	$\text{trans} = \text{<MCU defined>}$	$\text{trans} = \text{<MCU defined>}$

a. see table 1-10 for FIFO sizes for Lavender and Jasmine

Because the FIFOs can be accessed from GDC side (read from input FIFO and write to output FIFO) when trigger condition becomes true the FIFO load itself is taken for calculation. In figure 1-17 this situation is drawn.

$$\text{IFDMA\_LL} \leq \text{FIFOSIZE} - \text{BLOCKSIZE}^1 \quad (2)$$

$$\text{OFDMA\_UL} \geq \text{BLOCKSIZE} \quad (3)$$

For block/step- or burst mode condition (2) should be fulfilled for input FIFO in order to avoid FIFO overflow. For output FIFO at least one block should be available so that the condition (3) should be met. Otherwise wrong data may be delivered because BLOCKSIZE is transferred at once and not all data are available at transfer time.

For setup of DMA trigger levels the GDC internal packet sizes for data processing have to be considered. This are IPA block transfers and REQCNT for pixel data packetizing in Pixel Processor. It depends on command how many words are read from input FIFO or written to output FIFO at once. The amount of data words is determined by type of command data stream (address informations, colour data with different colour depths). See command description in appendix for a detailed command description.

In general the at once processed information must be available in IFIFO or has to fit into OFIFO. If this state is not reached the execution devices wait for data transfer by MCU until the requirements are fulfilled. If the DMA trigger levels are not set up accordingly deadlock situations can occur (data lack or jam).

1. BLOCKSIZE is the amount of words which is transferred at one DMA request (depends on MCU settings).

For DMA demand mode settings according to (4) (0x3f for Jasmine, 0x7f for Lavender) to keep the input FIFO full all the time and according to (5) to keep output FIFO flushed all the time avoid problems with internal packetizing. Data will be transferred immediately if possible.

$$\text{IFDMA\_LL} = \text{FIFOSIZE} - 1 \quad (4)$$

$$\text{OFDMA\_UL} = 1 \quad (5)$$

For DMA Block/Step/Burst mode other trigger levels are required due to additional restriction for MCU block transfer sizes (see equation (2) and (3) with their description).

Also at demand mode it is possible to set up other trigger levels in order to transfer more words at once. In this cases special care should be taken to avoid the described deadlock situations. Reserve for required data amount (IFIFO) or space (OFIFO) for packetized procession must be guaranteed all the time.

There are two possibilities to stop DMA transfer. The first is the falling edge of DMAFLAG\_EN as already mentioned. The second possibility is to interrupt the running command by a SWReset command. Because the DMA controlled stream is normally coupled to the currently executed command<sup>1</sup> interruption of this command should also cause DMA dropout. Because of FIFO reset during SWReset already transferred data will also be deleted.

### 1.7.4 DMA programming examples

In figure 1-18 an example for a MCU- and Lavender DMA initialization is given. The MCU DMA channel '0' is used for DMA connection.

In DMACB0 and DMACA0 MCU registers the parameters for MCU-DMAC are set; with set of Bit 31 in DMACR DMA operation is enabled inside MCU.

```
;;; DMA variables
blk_dmal: equ 1 ; block size
dte_dmal: equ 0 ; transfer count
ofhigh_dmal: equ 1 ; DMA limit
dmala: equ 0x8e100000 + (blk_dmal << 16) + dte_dmal ; build data word for DMACA0
dmalof: equ (ofhigh_dmal << 16)

;; -----
;; Init DMA for Data output
;; -----

writereg DMASA0,      G0OFIFO ; source address
writereg DMADA0,      0x001c0000 ; destination address
writereg DMACB0,      0x28000004 ; type=00,md=10(demand),ws=10(word),inc.destination
writereg DMACR,       0x80000000 ; enable MCU-DMAC
writereg DMACA0,      dmala
writereg G0OFDMA,     dmalof
writereg G0DMAFLAG,   0x00000206 ; OF, EN_DMA=1, demand mode, DSTP=2
; writereg G0DMAFLAG,  0x0000021E ; OF, EN_DMA=1, demand mode, DSTP=2, INV, TRI
; wait for DMA to finish
waitdma2:
    readreg DMACA0 ;nach r2
    ldi    #0x80000000, r3
    and    r3, r2
    bne    waitdma2
```

**Figure 1-18:** MCU- and GDC-DMA initialization example

---

1. For input FIFO it is also possible to deliver data for waiting command (see section 1.5). But the SWReset command flushes the command pipeline completely so that also the waiting command will be deleted. DMA transfer has to stop anyway.

---

The chosen DMA mode is demand mode which should be set inside MCU **and** display controller. Additionally for display controller the output FIFO is selected for DMA operation and signal inversion and tristate behaviour is not selected according to board implementation (see chapter 1.3.4). The DMA trigger limit for output FIFO is set to 1 (register: G0OFDMA) which means that every new data word in output FIFO causes a DMA transfer. In demand mode this ensures an empty FIFO after DMA operation but this causes also a lot of protocol overhead because a handshake between display controller and MCU is necessary for every data word. Therefore transfer performance may decrease slightly.

The loop labelled with 'waitdma2' at the end of figure 1-18 stops program execution until the end of DMA transfer. The following code can assume that data are transferred from display controller to MCU also as result of this low FIFO limit.

Figure 1-19 shows an example where a RLE compressed bitmap is transferred to display controller with help of DMA. This example uses C-API functions which are described in detail in a separate manual. See C-Comments for a short explanation.

```
/* ***** */
/* * Constant declarations */
/* ***** */
// Picture infos:
// - RLE compressed.
// - 16BPP.
// - 111 x 43 pixel, origin: 0, 0

// Picture dimensions
const hd_fujitsu_x = 111;
const hd_fujitsu_y = 43;

// Number of data words for 'hd_fujitsu' array
const hd_fujitsu_num = 800;

// Array definition
const dword hd_fujitsu_array[] = {
    0xeeffdfb8,
    0xffdf04fd,
    // ..data..
    0xdf000000};

void main(void)
{
    // use DMA (demand mode)
    // -----
    // Set up MCU- and GDC-DMAC
    // -----
    // ULB_DMA_HDG(dummy,dummy,mode,block,direction)
    // mode: 00: block/step, 01: burst, 10: demand
    // block: block size (1 in demand mode)
    // direction: 1: input FIFO; 0: output FIFO
    ULB_DMA_HDG(0, 0, 2, 1, 1);
    // -----
    // write parameter and command (see API description)
    // -----
    GDC_CMD_PtCP(0x0,hd_fujitsu_x-1,0x0,hd_fujitsu_y-1,0x0,0x0,0x0,0x0);
    // -----
    // activate DMA transfer
    // -----
    GDC_FIFO_INP((dword *)hd_fujitsu_array, (word)hd_fujitsu_num, 1);
    // -----
    // wait for end of transfer
    // -----
    while ((DMACA0 & 0x80000000) != 0);
    // -----
    // send NoOp in order to flush input FIFO
    // -----
}
```

---

```
GDC_CMD_NOP ( ) ;  
}
```

**Figure 1-19:** DMA programming example with help of C-API

The first step is to initialize MCU-DMAC as well as GDC-DMAC with help of API function 'ULB\_DMA\_HDG'. This function does not start the transfer; it sets only DMA parameters for the following transfer.

Afterwards the command for writing compressed bitmaps to display controller is sent to command register together with the dimensions of the bitmap. See chapter 1.5 for more details about command execution.

With help of API function 'GDC\_FIFO\_INP' DMA transfer is started. Note that the last function parameter is '1' which means that DMA transfer is enabled (see chapter 1.5.1 for a discussion about this function). In order to synchronize DMA data flow with program flow a wait loop for end of DMA transfer is included into the source code. This is not necessary in any case since the API functions 'ULB\_DMA\_HDG' and 'GDC\_FIFO\_INP' wait for the end of previous DMA transfer before they perform any action.

In the example in figure 1-19 the DMA synchronization is necessary in order to make sure that all data are transferred when a NoOp command is sent to display controller to force a input FIFO flush. Without this synchronization not sent data would be kept for the next command after NoOp.

## 2 ULB register set

### 2.1 Description

Some ULB registers are controlled by ULB itself and some are handled by CTRL in the same manner as for all other GDC components. For the programmer there is no difference accessing these registers. An overview on ULB- and CTRL controlled registers has already been given in table 1-5, section 1.4.

In table 2-1 an overview on all ULB registers is given. All addresses are relative to start of register space for given GDC; see section 1.4 for details. The address values are byte addresses and can be accessed in word (32 Bit), halfword (16 Bit) or byte (8 Bit) mode from MCU, except the FIFOs which can only be accessed in word mode.

Flag and interrupt mask register handling:

As already mentioned in section 1.6 the ULB contains one flag- and one interrupt mask register with special access modes; therefore in table 2-1 flag- and interrupt mask register have each three addresses.

**Table 2-1:** ULB register description

Register		Bits	Group Name	Description	Default value
Name	Address				
CMD	0x0000	31:8	PAR	Command parameter	0
		7:0	CODE	Command code	0xFF (NoOp)
IFIFO	0x0004	31:0	–	Input FIFO	–
OFIFO	0x0008	31:0	–	Output FIFO	–
FLNOM	0x000C	31:0	–	Flag register (normal write access) <sup>a</sup>	0x20400000
FLRST	0x0010	31:0	–	Flag register (reset write access) <sup>a</sup>	0x20400000
FLSET	0x0014	31:0	–	Flag register (set write access) <sup>a</sup>	0x20400000
INTNOM	0x0018	31:0	–	Interrupt mask register (normal write access) '1': use flag for interrupt	0
INTRST	0x001C	31:0	–	Interrupt mask register (reset write access) '1': use flag for interrupt	0
INTSET	0x0020	31:0	–	Interrupt mask register (set write access)	0
INTLVL	0x0024	31:0		Interrupt level/edge settings '1': positive edge of flag triggers interrupt <sup>b</sup> '0': high level of flag triggers interrupt	0xFFFFFFFF
WNOF0	0x0040	20:0	OFF	MCU offset for SDRAM window 0	0x10000
WNDSZ0	0x0044	20:0	SIZE	Size of SDRAM window 0	0x20000

Table 2-1: ULB register description

Register		Bits	Group Name	Description	Default value
Name	Address				
WNOF1	0x0048	20:0	OFF	MCU offset for SDRAM window 1	0x50000
WNDSZ1	0x004C	20:0	SIZE	Size of SDRAM window 1	0x00001
WNDS0	0x0050	23:0	OFF	SDRAM offset for SDRAM window 0	0x000000
WNDS1	0x0054	23:0	OFF	SDRAM offset for SDRAM window 1	0x100000
SDFLAG	0x0058	0	EN	'1': enable SDRAM space for GDC <sup>c</sup> '0': any access to SDRAM space is ignored by GDC <sup>c</sup>	0
IFUL	0x0080	23:16	UL	Input FIFO upper limit for flag- or interrupt controlled flow control Flag IFH=1 if IFLOAD <sup>d</sup> >= IFUL : UL	0x0C
		7:0	LL	Input FIFO lower limit for flag- or interrupt controlled flow control Flag IFL=1 if IFLOAD <sup>d</sup> <= IFUL : LL	0x03
OFUL	0x0084	23:16	UL	Output FIFO upper limit for flag- or interrupt controlled flow control Flag OFH=1 if OFLOAD <sup>e</sup> >= OFUL : UL	0x3C
		7:0	LL	Output FIFO lower limit for flag- or interrupt controlled flow control Flag OFL=1 if OFLOAD <sup>e</sup> <= OFUL : LL	0x0F
IFDMA	0x0088	7:0	LL	Lower limit for DMA access to input FIFO	0x0A
OFDMA	0x008C	23:16	UL	Upper limit for DMA access from output FIFO	0x3C



Table 2-1: ULB register description

Register		Bits	Group Name	Description	Default value
Name	Address				
DMAFLAG	0x0090	12:8	DSTP	Duration of ULB_DSTP signal. This value can be set in order to ensure a save MCU-DMAC reset. Normally the default value should work.	7
		4	TRI	'1': Set '1' to tristate ('Z') for ULB_DREQ, ULB_DSTP and INTRQ	0
		3	INV	'1': Invert ULB_DREQ, ULB_DSTP and INTRQ	0
		2	MODE	'1': DMA demand mode '0': DMA block/step- or burst mode	0
		1	EN	'1': enable DMA	0
		0	IO	'1': use DMA for input FIFO '0': use DMA for output FIFO	1
FLAGRES	0x0094	31:0	-	'1': set flag to dynamic behaviour <sup>f</sup> '0': set flag to static behaviour <sup>f</sup>	0x20400000
ULBDEB	0x0098	23:16	IFLC	Input FIFO load for current command <b>Attention:</b> This value changes with GDC core clock; correct sampling by MCU can't be ensured. Value is read-only; writing is ignored.	0x00
		15:8	OF	Output FIFO load <b>Attention:</b> This value changes with GDC core clock; correct sampling by MCU can't be ensured. Value is read-only; writing is ignored.	0x00
		7:0	IF	Input FIFO load independent from current command <b>Attention:</b> This value changes with GDC core clock; correct sampling by MCU can't be ensured. Value is read-only; writing is ignored.	0x00

- a. For meaning of flags and default value see section 1.6.
- b. **Attention:** This is only allowed when GDC core clock is equal to ULB bus clock (see section 1.6)
- c. See section 1.4.
- d. IFLOAD: Input FIFO load
- e. OFLOAD: Output FIFO load
- f. For a description of flag handling see section 1.6.

## 2.2 ULB initialization

ULB contains no lockable registers; so it is possible to write to every register at any time.

There is no initialization order for ULB but some general rules should be followed:

- For Lavender INTLVL register should normally be set to 0x00000000 in order to trigger interrupt on high level (see section 1.6 for more details).
- SDRAM space for direct memory access has to be initialized and enabled for use. Write valid values to WNDOFx, WNDSZx, WNDSx and 0x00000001 to SDFLAG. Be careful about overlapping windows when more than one GDC is connected to MB91xxxx (see section 1.4 for more details).
- Initialize IFUL or OFUL with valid limits before using FIFO limit flags (OFH, OFL, IFH, IFL) for interrupt or polling. Otherwise default values will be taken as valid limits.
- Initialize MCU, IFDMA, OFDMA and DMAFLAG in this sequence with valid values in order to use DMA for data transfer (see section 1.7 for details). Note that DMAFLAG\_EN should be written at last because it starts DMA transfer triggered by GDC.
- FLAGRES register should be initialized correctly before interrupt is enabled inside MCU or flags are polled within user program in order to meet applications need.
- Read-only ULBDEB register exists only for debugging purpose; in normal applications flags in connection with FIFO limits should be preferred.

---

## ***B-3 SDRAM Controller (SDC)***



# 1 Function Description

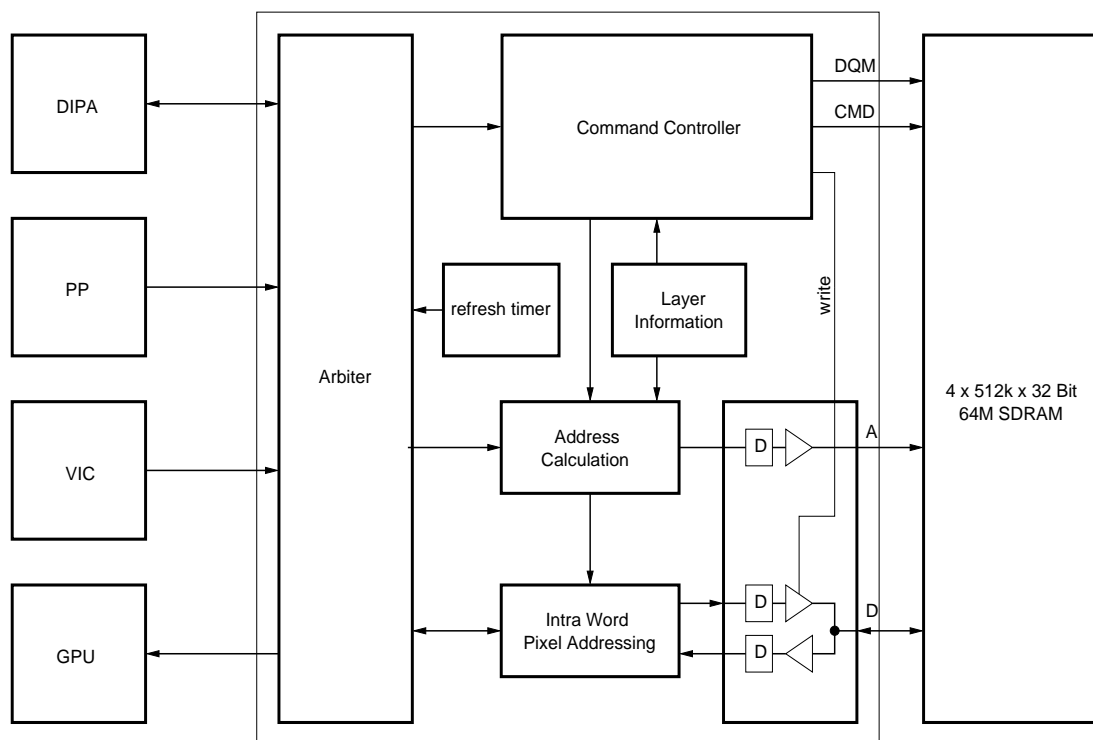
## 1.1 Overview

This module is part of a graphic display controller (GDC) especially for automotive applications. The GDC supports a set of 2D drawing functions (Pixel Processor) a video scaler interface, units for physical and direct video memory access and a powerful video output stream formatter for a great variety of connectable displays.

Inside the GDC there is a memory controller which arbitrates the internal modules and generates the required access timings for SDRAM devices. With a special address mapping and an algorithm for generating optimized control commands the controller can derive full benefit from 4-bank-interleaving supported by the SDRAMs. So the row activation time is hidden if switching to another memory bank in most cases. This increases performance respective at random (non-linear) memory access. Power down modes with Clock Suspend (CSUS) with and without SELF-Refresh are supported.<sup>1</sup>

The SDRAM Controller arbitrates GDC internal device requests for data transfers from/to the video memory. Important is also the address calculation which controls the mapping from a given logical address (in layer, x- and y-pixel format) to the physical bank, row and column address. Thus the other devices are independent from the physical implementation of the memory structure.

If the Application for GDC needs physical video memory (frame buffer) access, knowledge is needed how a logical address (layer, x, y) is converted to the physical address in video memory and how this address maps into physical host MCU address space. There is also a buffered access method without mapping into MCU address space possible, then only internal video memory address is of interest.



**Figure 1-1:** SDC Block Diagram embedded into GDC

<sup>1</sup>Jasmine implementation (GDC with integrated DRAM) makes use of an integrated single-bank SDRAM. Therefore special features as 4 bank interleaving and power suspend/self refresh are not supported by the device.

Main functionality is to provide an arbitrated video memory access for GDC components such as Pixel Processor (PP), Direct/Indirect Memory Access (DIPA), the Video Interface Controller (VIC) and finally the Graphic Processing Unit (GPU) which reads pixel data from memory and formats the output stream. Because of the different requirements of the various components there are to support various access types, such as burst and block modes with adjustable transfer sizes.

## 1.2 Arbitration

The arbitration of the four main GDC parts works priority based. The setup of priority values can be decided by the requester component itself and is signaled to the SDRAM controller. The benefit is that the setup can vary for different applications. There is also the possibility to change priority on the fly, e.g. if buffer state changes. The connected component can decide about the urgency of the transfer.

**Table 1-1:** GDC modules and its priority registers with recommended configuration

Device	Register		Comment
GPU	SDCP_LP = 3	SDCP_HP = 7	low and high priority, real time device with automatic priority scaling
VIC	SDRAM_LP = 2	SDRAM_HP = 6	low and high priority, real time device with automatic priority scaling
PP/AAF	SDCPRIO = 1		no automatic priority scaling supported
DIPA	DIPACTRL_PDPA = 5	DIPACTRL_PIPA = 0	priorities for DPA and IPA access

Video RAM arbitration is done in principle of cooperative multitasking. This did not waste bandwidth if a requester device uses only a part of its dedicated bandwidth as if time slicing would be used. All devices share the commonly available bandwidth resource. Main advantage is that system performance could be scaled and optimized for a wide range of different applications.

A decision about granting the next device is done priority based at the end of a currently processed device request. The currently processed device is excluded from priority based selection for the next one. This results in granting requests for the two devices with given highest priorities alternately, if requested. Only idling between these two devices could be used for the other ones. Therefore the devices with the two highest priorities could be considered as real-time (normally this should be GPU and VIC).

## 1.3 SDRAM Timing

Configurable options for the appropriate SDRAM timings listed in table 1-2. Defaults are listed for 100 MHz SDRAM types of MB811643242A for Lavender. Values for integrated DRAM version for Jasmine are given in a separate column. The configuration value is a number of wait states. That means additional

**Table 1-2:** SDRAM Command Timings

Parameter	Default	Jasmine	Description
tRP (RAS Precharge Time)	30 ns	22.5 ns	Time from same bank PRE to row ACTV command
tRRD (RAS to RAS Bank Active Delay Time)	20 ns	-	Time from ACTV to opposite bank ACTV command
tRAS (RAS Active Time)	60 ns	37.5 ns	Time from ACTV to same bank PRE command

**Table 1-2: SDRAM Command Timings**

Parameter	Default	Jasmine	Description
tRCD (RAS to CAS Delay Time)	30 ns	22.5 ns	Time from same row ACTV to READ or WRIT command
tRW (Read to Write Recovery Time) <sup>a</sup>	10 (8) T	7 (5) T	Pipeline recovery time from each READ to WRIT command

a. This setup is not regarding DRAM timing, but required to avoid bus collision on internal busses or external SDRAM tri-state busses due to pipelined operation. Values in parenthesis are possible if anti aliasing filter is switched off and then no read-modify-write access is required.

clocks of idling before the next SDRAM access command. So the configuration value is lower by one than the required timing from the SDRAM data sheet. If an absolute minimum time is given it's necessary to evaluate the corresponding number of clock periods for configuration. This depends on and should be optimized for the required core clock frequency.

Following procedure should be used:

1. divide given timing by the core clock period
2. round up to next integer
3. subtract one to have the right wait-state value

CAS Latency can be setup to values of 2 or 3 for Lavender. Jasmine is not programmable for different CL values.

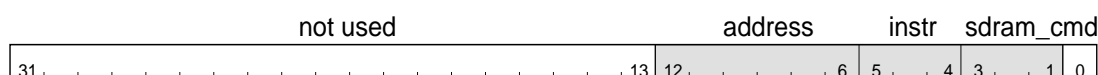
Additional configurable options are the refresh period (normally 16 us for one row) and the power on stabilization timer (200 us) before the first initialization sequence begins to run. The refresh counter is reset after each execution of a single row refresh job (not considering if it runs as time-out or idle task) and it causes an time-out if the counter value reaches zero. The configuration values are given in a number of system clocks.

## 1.4 Sequencer for Refresh and Power Down

Fixed command sequences such as SDRAM initialization, auto refresh, power down or wake-up and transfers of special data structures are easier to implement in a fixed and preprogrammed manner. These tasks are assigned to the sequencer unit of SDC.

To keep the amount of memory low and guarantee a defined device shutdown the power down sequences are not a part inside the standard micro program for refresh and initialization. However special power down sequences are loaded into memory when needed. If the SDC currently processes a transfer controlled by the address/command generator unit these sequence will be finished normally and then the new loaded routine inside the micro program storage is executed.

One word of micro program code consists of an address argument (bits [12:6] for Lavender, bits [11:6] for Jasmine<sup>1</sup>), flow control instruction and a container command (SDRAM command).<sup>2</sup> Figure 1-2 shows the

**Figure 1-2: Micro program entry**

format of one micro program entry. Bits [3:1] of SDRAM command coding the RAS, CAS and WE signal. The internal representation is inverted compared with the SDRAM ports. Bit [0] for controlling the auto precharge (AP) feature is not controllable by the sequencer and internally fixed to '0'. Table 1-3 lists the

1. Jasmine has reduces sequencer size of 32 words. Thus address argument is 5 bit only.

2. Logical address operations and data validation flag are not needed in this application without preprogrammed data structures.

possible entries. In this Application there is no preprogrammed data transfer implemented. So the commands `actv`, `writ`, `read` and `bst` should not be used. Sequence programming is done with special flow control

**Table 1-3:** SDRAM control commands

Mnemonic	Description	Representation {ras, cas, we}
mrs	Mode Register Set	111
ref	Auto/Self Refresh	110
pall	Precharge all Banks	101
actv	Activate Row	100
writ	Write	011
read	Read	010
bst	Burst Stop	001
nop	No Operation	000

instructions, sub program calls, loops, power down entry and exit are supported.<sup>1</sup> Table 1-4 lists possible flow control instructions and their coding.

**Table 1-4:** Flow control instructions

Mnemonic	Description	Representation
run	Run linear program flow	000
ret	Return from sub routine	001
call	Call subroutine on address argument (no nested calls possible)	010
loop	Repeat program at address argument if loop counter not reached	011
end	Alias for 'loop #0' when loop counter is '1'	011
pde	Power down entry	110
pdx	Power down exit	111

In general it's recommended to use the 'mkctrl' tool for GDC setup. It optimizes SDRAM timing based on core clock frequency, calls the `asmseq` sequencer program and generates valid code for the sequencer. An example of the micro code is provided with the assembler tools. There is also a compression tool which generates smaller programs with sub-routine calls from a linear coded micro program (`asmseq_delay`).

Size of sequencer memory is 64 entries for Lavender and 32 entries for Jasmine.

---

1. Read-write control, supported by the assembler (`srw`, `rrw`) is not implemented. There are no data structures programmable for special transfers.

---



## 1.5 Address Mapping

This section describes the relationship between logical and physical addresses and how to map from a given logical address to the physical memory position. At the begin we have to introduce the meaning of the used Layer Description Record (LDR) information. The Address Unit uses the parameters of the 16 LDR entries:

- PHA(i)                      Physical Address Offset
- DSZ\_X(i)                  Domain Size X
- CSPC\_CSC(i)            Color Space Code.

The address offset PHA, stored in the LDR, describes the start address of a layers position. This is where the most upper left pixel of a picture is located. Due to the block structure of the picture data, only the part of the row address is valid for the physical start address offset entry (bits [22:12] for Lavender or [19:10] for Jasmine, see figure 1-3). Lower bits are fixed to '0'. That restriction applies because of the same row can't be used for different layers. Domain size in X-dimension DSZ\_X is given in logical pixels. It is needed to calculate the pixels per line. The Y-dimension is not needed, there is no automatic layer size limitation implemented. Please note that there is no DSZ\_Y register implemented. Color space code CSPC\_CSC is a representation of the appropriate color format. It is converted internally to calculate the bits per pixel (bpp) by power of two, which is equal to the number of bits the pixel address has to be shifted to get the right word address.

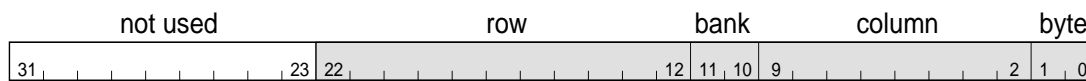
**Table 1-5:** Color Space Codes

Code	Color Space Type	bpp	Shift
0x0	1bpp	1	0
0x1	2bpp	2	1
0x2	4bpp	4	2
0x3	8bpp	8	3
0x4	RGB555	16	4
0x5	RGB565	16	4
0x6	RGB888	32	5
0x7	YUV422	16	4
0x8	YUV444	32	5
0x9-0xD	<i>reserved for GPU intermediate color space</i>		
0xE	YUV555 <sup>a</sup>	16	5
0xF	YUV656 <sup>a</sup>	16	5

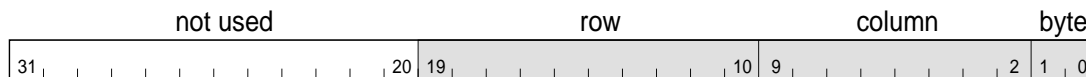
a.Jasmine only

The physical address is a combination of SDRAM bank, row and column address. The significance is pre-defined in following order from row over bank to column address. Thus the picture data is stored in a block- ing structure drawn in section 1.5.1, figures 1-6/1-7. Additional to physical word addressing there can be distinguished between several byte addresses. The complete physical address format is shown in figures 1-

3 and 1-4 as it is used for IPA and DPA access methods. For LDR entries of PHA only row bits are valid,



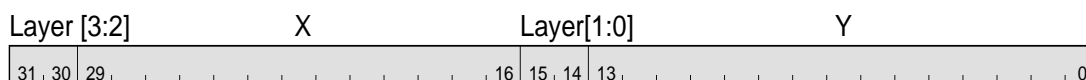
**Figure 1-3:** Physical address format (Lavender)



**Figure 1-4:** Physical address format (Jasmine)

lower bits are fixed to '0'. This is due to the physical start address is aligned on the row grid.

For comparison, logical address format is shown in figure 1-5.



**Figure 1-5:** Logical address format

Now back to the relationship between logical pixel address containing layer, X and Y location and the physical mapping of the pixel data. The needed bits per pixel line of the layer can be calculated as<sup>1</sup>

$$XBits = DomSzX \ll Shift = DomSzX * bpp$$

Remark: Expression ' $\ll Shift$ ' interpretable as ' $* bpp$ ' with the restriction

that  $bpp$  has a value range of power of two  $\{2^0, 2^1, 2^2, 2^3, 2^4, 2^5\}$

For determining Shift value from its Color Space Code see table 1-5. It depends on the layer number and how CSPACE is configured for it.

Layer memory can only be divided into whole numbered parts of rows in each dimension. Each row segment has a width of 8 words. Lavender uses 2 adjacent banks with same row number, thus a virtual row of 16 words is formed. From the number of bits the needed number of horizontal memory rows is

$$XRows = XBits[18:9] + (XBits[8:0] ? 1 : 0) \quad (\text{for Lavender})$$

$$XRows = XBits[18:8] + (XBits[7:0] ? 1 : 0) \quad (\text{for Jasmine})$$

Finally the row, bank and column addresses can be derived from the logical address components and this temporary values. Concatenation of row, bank and column address enhanced with 2 bits for byte addressing results in the physical address.

$$RA = Y[13:6] * XRows + (X \ll Shift)[18:9] \quad (\text{for Lavender})$$

$$RA = Y[13:5] * XRows + (X \ll Shift)[18:8] \quad (\text{for Jasmine})$$

$$BA = \{Y[5], (X \ll Shift)[8]\} \quad (\text{for Lavender only})$$

$$CA = \{Y[4:0], (X \ll Shift)[7:5]\}$$

1. Squared brackets stand for vector slices, curly braces are vector combinations.

## 1.5.1 Elucidations regarding Address Mapping

### 1.5.1.1 Block Structure of Pixel Data

DRAMs have not equal access timings if randomly accessed. If a ROW address is already activated, faster access can be done. Each ROW consists of 256 COL addresses.

As compromise between horizontal and vertical operation a block oriented access scheme is implemented. A block is identical with one ROW, each 256 COL addresses with faster access. Disadvantage of this block structure is a more complicated pixel addressing over direct physical access methods. Block size is defined to 8 words horizontal<sup>1</sup> and 32 lines vertical<sup>2</sup>. For example, this results to 32x32 pixel block size at 8 bpp.

If bank interleaving is used (on Lavender chip), same ROW address for each of the 4 banks are combined to a macro block with double size in horizontal and vertical dimension. This gives the chance to activate a ROW in another bank before reading from it during access is running on another bank. This hides row access time in most cases.

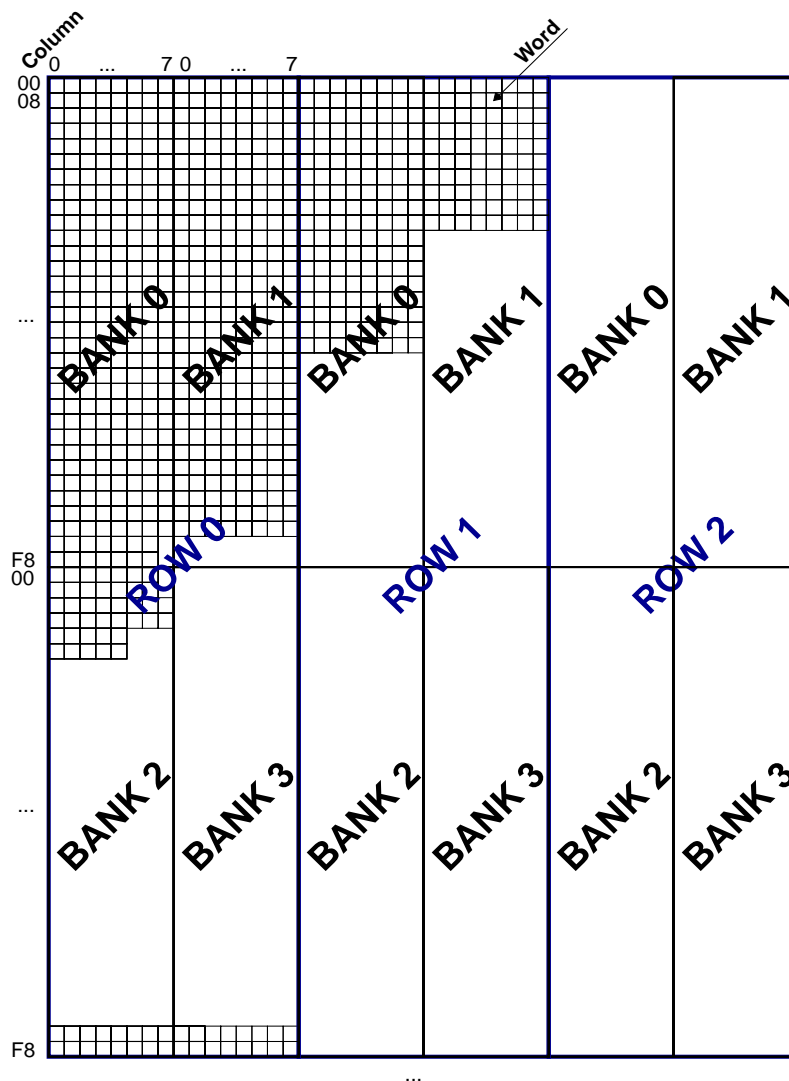
For access by pixel address the block structure is not relevant. It is mapped automatically by hardware to the right physical address. If non-picture data or data which should not be displayed is stored via physical access, address can be interpreted as linear space without rows, banks and columns. Only if physical access on graphic data is required, the block based philosophy should be considered.

---

1.number of pixels depending on color depth (bpp)

2.word and pixel have same meaning

---



**Figure 1-6:** Memory Mapping of Bank, Row and Column Address (Lavender)

### 1.5.1.2 Access Methods and Devices

This section is not SDC relevant but the reader can have benefit in better architectural understanding of the GDC device. It depicts setup of other GDC macros which are related to addressing data in memory.

- Pixel addressing for drawing commands

There are two main sections of command regarding addressing method. First group uses pixel address information over input FIFO, second group makes use of registers for pixel coordinates.

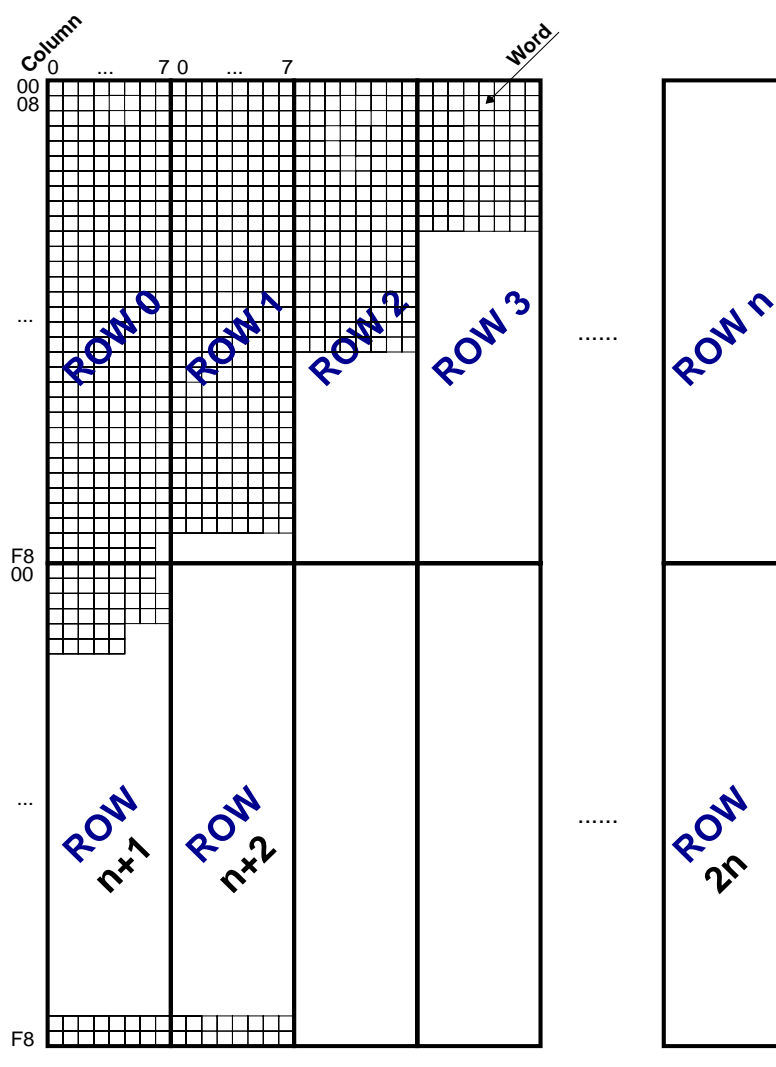
Commands Grp1 (FIFO): PutPixel, PutPxWd, PutPxFC, GetPixel, XChPixel, DwLine, DwPoly, DwRect, MemCP

If PPCMD\_ULAY register set to 0, complete logical address information is fed through Input FIFO. Format consists of Layer, X and Y position. The 32 bit pixel address word is combined to (from MSB to LSB) {L[3:2], X[13:0], L[1:0], Y[13:0]}.

If PPCMD\_ULAY register set to 1, layer information is used from layer register PPCMD\_LAY and is not evaluated from Input FIFO. The appropriate layer bits of Input FIFO data are don't care. Address Format consists of X and Y position. The 32 bit pixel address word is combined to {L[- -], X[13:0], L[- -], Y[13:0]}.

Commands Grp2 (REGs): PutBM, PutCP, PutTxtBM, PutTxtCP

These Commands have it's dedicated coordinate registers. No address information is fed through Input FIFO. They are using XYMIN, XYMAX, PPCMD\_LAY registers.



**Figure 1-7:** Memory Mapping of Row and Column Address (Jasmine)

- Direct memory mapped Physical Access (DPA)

Command Interface is not necessary for this access method. However some specialities should kept in consideration while using the DPA interface.

- DIPA clock is enabled
- DPA should be enabled by setting SDFLAG to '1'
- Two windows are possible to map into MCU address space (shares GDC Chip Select)
- Window address offset WNDOF and size WND SZ are mapped to required address space
- WNDSD is set to the section start address of video RAM which appears in the window

Mapped address calculates to

$$\text{PHY\_MAP} = \text{CS\_REGION} + \text{WNDOF} - \text{WNSD}$$

WNSZ limits size of accessible address range. If exceeded no write permission is granted and tri-state buffers kept close at reading.

DPA runs completely unbuffered. Additional there are no real-time or preferred data channels to the video RAM available, the normal SDC requesting and arbitration procedures apply. The normal case is that DPA

has to wait for higher prioritized jobs and the currently running task for completion. Thus only a slow access and difficult predictable timing results from this behaviour.

*The not predictable access time requires DPA\_RDY polling at writing due to the RDY line pull down feature is in general only supported for read access in GDC.*

With setting higher Priority for DPA access than GPU (display output) this situation can be improved. This can help when high bandwidth components are running continuously, i.e. GPU, PP and VIC. The risk of interrupting the real-time streams of GPU and VIC increases only negligibly, but beware of changing default priority when working at the upper limit of bandwidth consumption.

- Indirect Physical Access (IPA)

Commands: PutPA, GetPA

IPA makes full benefit of physical access while using burst transfer techniques. Additional no restrictions apply with address range limitation. Physical address is transferred to the Input FIFO. Data packets are also routed through Input or Output FIFOs. To achieve maximum data throughput physical address auto-increment is implemented for GetPA function.

If logical pixel data is transferred via physical access, be aware of physical address incrementing method. Due to the fact that single transfers with converted addresses (logical to physical) are not effective over this device, the user should check if block based transfers are possible. Pixel data have to be divided into segments even to 8 data words in X-dimension and then next line of block can be transferred. Burst transfer should start aligned on the block grid (8x32 words). With this method only one start address has to be converted and sent followed by a data block transfer of up to 256 words is possible. Another way is the transfer of 8-words line segments with a start address with only moderate amount of address overhead. This has the advantage that there is no need for restricting pixel position to the block grid in Y-dimension.

Problematic in any case is random access on pixels over physical addresses. Command and address calculation effort is too high. Dedicated Pixel commands should be used then.

If packetized block transfer is used, priority of IPA device has not that much influence on data rate compared to DPA. But increasing of IPA priority may cause interruption of real-time processes of VIC and GPU. Only two devices with highest priority setting are kept as real-time due to the arbitration scheme.

Important parameters for IPA are Input and Output FIFO data amount MIN/MAX thresholds (DIPAIF\_IFMAX, DIPAIF\_IFMIN, DIPAOF\_OFMAX, DIPAOF\_OFMIN). These thresholds control when a transfer is started/stopped (max) and adjust block size of memory transfers (min). Higher block size improves performance but increases risk of data stream interruption for other devices.

### 1.5.1.3 Program Example

Following example demonstrates address mapping and physical access with relationship to pixel coordinates. Intention is to draw a rectangular area with radiances and finally copy the drawn object per physical access.

```

/* ... before this section of code INIT_GDC from mkctrl tool was included */

ClrLayer(0x008080, 0);
ClrLayer(0x008080, 1);

/* ... here initialization of window handles W0...W2 is normally included */

xoff = 30;
yoff = 45;
aafen = 1;

/* DEMONSTRATION OF PHYSICAL ACCESS - DRAW ORIGINAL PATTERN */
puts (&W0, "[1] Drawing original pattern\n");
DrawRect (0xd0d000, 0, xoff, yoff, 100, 100);

GDC_CMD_NOP(); /* Cancel DwRect before AAF enable */
PXP_AAF_THE(aafen);

for (ang=0; ang<6.2828; ang+=0.15707) { /* draw radiancies */
    recx = (word) (50*sin(ang));
    recy = (word) (50*cos(ang));
    DrawLine(0x000000, 0,
        (xoff+50)<<aafen, (yoff+50)<<aafen, /* double size if AAF enabled */
        recx<<aafen, recy<<aafen);
}
GDC_CMD_NOP();
PXP_AAF_THE(0);

/* DEMONSTRATION OF PHYSICAL ACCESS - COPY TO DESTINATION W0WINDOW */
puts (&W0, "[2] Copy using physical access\n");

for (x=xoff; x<xoff+100; x++) {
    for (y=yoff; y<yoff+100; y++) {
        src = phy_address(0, x, y);
        dest = phy_address(1, x, y);
        *(dword *)dest = *(dword *)src;
    }
}

puts (&W0, "[3] finished.\n");

```

**Figure 1-8:** Excerpts from main program of the physical copy example

The example given in figure 1-8 did not use polling of DPA\_RDY flag. The information that the DPA device is ready for writing is implicitly given by the finished read access before. DPA has a two-stage buffer and read access is synchronized by using the hardware flow control over ULB\_RDY line. The following write access has at least one free buffer available.

```

/* build required format of pixel address */
dword pix_address (byte layer, word x, word y) {
    return (x << 16) + y + ((layer&0x0C) << 28) + ((layer&0x03) << 14);
}

```

**Figure 1-9:** Formatting logical address from Layer, X, Y

```
/* layer description record lookup and bit per pixel (bpp) mapping */
byte bpp_lookup(byte layer) {
    switch (G0CSPC_CSC(layer)) {
        case 0:          // 1bpp
            return 1;
        case 1:          // 2bpp
            return 2;
        case 2:          // 4bpp
            return 4;
        case 3:          // 8bpp
            return 8;
        case 4:          // RGB555
        case 5:          // RGB565
        case 7:          // YUV422
        case 0x0e:       // YUV555
        case 0x0f:       // YUV655
            return 16;
        default:         // (6) RGB888, (8) YUV444
            return 32;
    }
}
```

**Figure 1-10:** Figuring out bpp value from layer setting

```
void DrawRect (dword c, byte l, word x, word y, word sx, word sy) {
    dword corners[2];
    corners[0] = pix_address(l, x, y);
    corners[1] = pix_address(l, x+sx-1, y+sy-1);
    GDC_CMD_DwRt(c);
    GDC_FIFO_INP((dword*) corners, 2, 0);
}

void DrawLine (dword c, byte l, word x, word y, word lx, word ly) {
    dword corners[2];
    corners[0] = pix_address(l, x, y);
    corners[1] = pix_address(l, x+lx-1, y+ly-1);
    GDC_CMD_DwLn(c);
    GDC_FIFO_INP((dword*) corners, 2, 0);
}
```

**Figure 1-11:** Easy to use drawing functions for the example

The functions above are not that important for understanding address calculation but used in examples given. They are shown for completeness only.

Most interesting is `pix_address()` function given in figure 1-12. Any information is queried from GDC registers. This is done for better understanding of internal arithmetic only. If some settings are known before and fixed for the application, the algorithm can be simplified, which decreases execution time significantly. It also shows the differences for Lavender and Jasmine.



```

/* Build physical address from pixel address, function uses DRAM address */
/* window 0 only, pls ensure that DRAM mapping to MCU address is enabled */
dword phy_address (volatile byte layer, word x, word y) {
    byte bpp;          /* Color depth lookup, 1/2/4/8/16/32 */
    byte ca;           /* DRAM Column Address (8 bit) */
    word DomSzX;        /* Layer Size X-Dimension (14 bit) */
    dword XBits;        /* Number of Bits for one Line (19 bit) */
    word XRows;         /* Number of Row blocks in X-Dimension (10 bit) */
    word ra;           /* DRAM row address (without layer offset) */
    byte ba;           /* DRAM bank address, Lavender only (2 bit) */
    dword PhySDC;       /* Physical address SDRAM Controller view */
    dword PhyMCU;       /* Physical address MCU view */
    dword LayOffs;      /* Layer Offset */

    /* Determine Layer parameters and number of bits per line of pixels (x) */
    /* Formula: XBits = DomSzX << Shift = DomSzX * bpp */
    bpp = bpp_lookup(layer); /* layer color depth */
    DomSzX = G0DSZ_X(layer); /* layer pixel width */
    LayOffs = G0PHA(layer); /* layer start address */
    XBits = DomSzX * bpp; /* layer bitsize width */
    /* Column address, Formula: CA = {Y[4:0], (X << Shift)[7:5]} */
    /* Y: 32 lines, X: 8 words each block */
    ca = ((y & 0x1f) << 3) + (((x*bpp)&0xff)/32);

    switch (G0CLKPDR_ID) {
    case 1: /* ----- ID: Jasmine, GDC-DRAM, single bank ----- */
        /* Number of memory rows (grid of pixel blocks) in X-dimension */
        /* each partially used row has to be considered (add 1 if remainder) */
        /* Formula: XRows = XBits[18:8] + (XBits[7:0]? 1: 0) */
        XRows = (XBits>>8) + ((XBits & 0xff)? 1: 0);
        /* DRAM row address relative to layer start address, each row has a */
        /* block size of 256 bit in X-dimension and 32 lines in Y-dimension */
        /* Formula: RA = Y[13:5] * XRows + (X << Shift)[18:8] */
        ra = y/32*XRows + (x*bpp/256);
        /* combination of address bits: [19:10]ra, [9:2]ca, [1:0]byte */
        PhySDC = LayOffs + (ra<<10) + (ca<<2);
        break;
    case 0: /* ----- ID: Lavender, GDC with external DRAM, 4 bank ----- */
        /* Formula: XRows = XBits[18:9] + (XBits[8:0]? 1: 0) */
        XRows = (XBits>>9) + ((XBits & 0x1ff)? 1: 0);
        /* row block size is 512 bit in X-dim, 64 lines in Y-dim */
        /* Formula: RA = Y[13:6] * XRows + (X << Shift)[18:9] */
        ra = y/64*XRows + (x*bpp/512);
        /* there exist 4 banks with same row address, thus each row block */
        /* consits of 4 bannk parts */
        /* Formula: BA = {Y[5], (X << Shift)[8]} */
        ba = ((y>>5) & 0x1)*2 + (((x*bpp)>>8) & 0x1);
        /* combination of address: [22:12]ra, [11:10]ba, [9:2]ca, [1:0]byte */
        PhySDC = LayOffs + (ra<<12) + (ba<<10) + (ca<<2);
        break;
    default:
        return -1; /* err: wrong chip ID */
    }

    /* add GDC0 offset (G0CMD is first GDC address) and ULB settings */
    /* such as MCU Window 0 offset and subtract SDRAM offset */
    /* check consistency if memory window is mapped and reachable */
    if (!G0SDFLAG_EN)
        return -2; /* err: SDRAM mapping not enabled */
    if (PhySDC < G0WNDS0 || PhySDC >= G0WNDS0+G0WNDSZ0)
        return -3; /* err: pixel address outside mapped Video RAM space */
    /* DRAM address GDC base address */
    PhyMCU = (dword) (char *) PhySDC + (dword) (char *) &G0CMD
    /* MCU offset SDRAM offset */
    + (char *) G0WNDOF0 - (char *) G0WNDS0;
    return PhyMCU;
}

```

Figure 1-12: Logical to physical address conversion routine

Last example is an intelligent ClearLayer function, which determines the start address of the next layer automatically. In that way layer size in Y-dimension is calculated before drawing a monochrome rectangle filling the complete layer.

```
void ClrLayer(dword color, byte layer) {
    dword R0[2];
    int j;
    byte ppw;
    dword next_layer_phy, PhysSz;
    word DomSzXWrd, DomSzXBlk, DomSzYLin, DomSzY;

    /* ----- calculation of Domain Size Y ----- */
    next_layer_phy = 0x100000; /* max memory size */
    /* search next Layer start address
       and calculate physical size from actual to next layer */
    for (j=0; j<16; j++) {
        if ((layer!=j) && (G0PHA(j)>G0PHA(layer)) && (next_layer_phy>G0PHA(j))) {
            next_layer_phy = G0PHA(j);
        }
    }
    /* calculate max Domain Size Y to fit in Physical Layer Size */
    PhysSz = next_layer_phy - G0PHA(layer);
    ppw = 32/bpp_lookup(layer); /* pixel per word */
    DomSzXWrd = G0DSZ_X(layer)/ppw + (G0DSZ_X(layer)%ppw? 1: 0);
    if (G0CLKPDR_ID == 1) { /* Jasmine 8x32 row blocks */
        DomSzXBlk = DomSzXWrd/8 + ((DomSzXWrd & 0x7)? 1: 0);
        DomSzYLin = PhysSz/4/DomSzXBlk/8;
        DomSzY = DomSzYLin - (DomSzYLin & 0x1f);
    }
    if (G0CLKPDR_ID == 0) { /* Lavender 16x64 row blocks */
        DomSzXBlk = DomSzXWrd/16 + ((DomSzXWrd & 0xf)? 1: 0);
        DomSzYLin = PhysSz/4/DomSzXBlk/16;
        DomSzY = DomSzYLin - (DomSzYLin & 0x3f);
    }
    /* ----- Clea layer function ----- */
    R0[0] = pix_address(layer, 0, 0);
    R0[1] = pix_address(layer, G0DSZ_X(layer)-1, DomSzY-1);
    GDC_CMD_DwRt(color);
    GDC_FIFO_INP(R0, 2, 0);
}
```

**Figure 1-13:** ClearLayer with automatic layer size detection

#### 1.5.1.4 Address Calculation - Example with concrete {X,Y} Pixel

Mkctrl Tcl-GUI	Name in gdc_reg.h	Value
gdc_offset0	WNDOF0	0x40000
gdc_size0	WNDSZ0	0x80000
sdram_offset0	WNDS0	0x0
Physical Address Layer 0	PHA(0)	0x20000
Domain Size X	DSZ_X(0)	640
Color Space Code	CSPC_CSC(0)	RGB555 (16bpp)

This example calculation is for the Lavender Chip. First calculation returns number of row-blocks in X-dimension. We need 20 (0x14) rows side by side for storing 640 pixels with color depth 16 bit.

$$XBits = DomSzX * bpp = 640 * 16 = 10240 = 0x2800$$

$$XRows = XBits[18:9] + (XBits[8:0]? 1: 0) = 10240/512 + 0 = 20 = 0x14$$

Assumed first pixel {0,0} address is searched for, physical address is Layer 0 start address directly.

```
PhyMap  = GDC_CS_Area + WNDOF0 - WNDS0 = 0x30000000 + 0x40000 - 0
PhyMap  = 0x30040000
PhyAddr = PhyMap + PHA(0) = 0x30060000
```

Now for pixel position of Layer 0, X=115, Y= 250 physical address should be transformed.

```
RA = Y[13:6] * XRows + (X << Shift)[18:9]
RA = 250/64*20 + 115*16/512 = 3*20 + 1840/512 = 63 = 0x3f = 0b11 1111
BA = {Y[5], (X << Shift)[8]} = {bit 5 of 0xFA, bit 8 of 0x730} = 0b11
CA = {Y[4:0], (X << Shift)[7:5]} = {0b11010, 0b001} = 0b1101 0001
```

Concatenation of RA, BA and CA results in the physical SDRAM word address. Additional concatenation of two bits 0b00 convert it to a byte address (x 4).

```
PhySDRAM = {RA, BA, CA, 0b00} = 0b11 1111. 11.11 0100 01.00 = 0x3FF44
```

Finally physical address offset for Layer 0 has to be added.

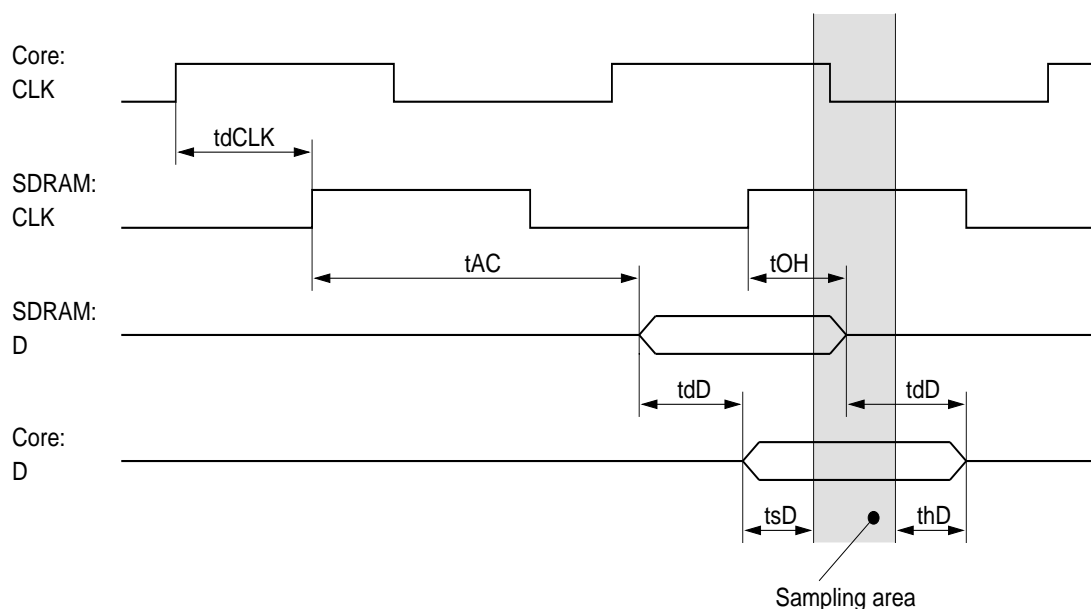
```
PhyAddr = 0x20000 + 0x3FF44 = 0x5FF44
```

This address can be used for IPA directly. For DPA GDC\_CS\_Area, WNDOF0 and WNDS0 have to be considered.

```
PhyAddrDPA = PhyMap + PhyAddr = 0x30040000 + 0x5FF44 = 0x3009FF44.
```

## 2 SDRAM Ports

From timing point of view critical is specially read accesses due to the fact, that delays of clock to the memory device and data back from memory to GDC have to be summarized relative to the internal rising clock edge. Figure 2-3 shows when read data from SDRAM are valid after feeding back to GDC. There is the additional restriction for keeping the setup and hold times of the data input registers, which reduces the valid region of sampling time additionally. That's why active clock edge of the input flip-flop has to be inside the gray marked region. Best decision is the mid of this to have enough space for deviations. There are different possibilities to compensate the part of total delay which is larger than one clock period. One is to insert a delay buffer in the clock lines of the input registers, another is to sample data on falling clock edge of input register. Additional to the input register clock delay there are to satisfy the hold time requirements of all SDRAM input signals (outputs from GDC). Due to the fact that the input signal timing has to be relative to the receiving clock at the SDRAM, this depends much on the delay of output clock buffer and clock wire delay.



**Figure 2-1:** SDRAM Interface Timing

### 2.1 External SDRAM I/O-Pads with configurable sampling Time (Lavender)

Additional to the tri state control there is a special timing feature implemented at the SDRAM ports. Inside the SDC there is a configurable hold time adjustment for the outputs and sampling time adjustment for the data input.

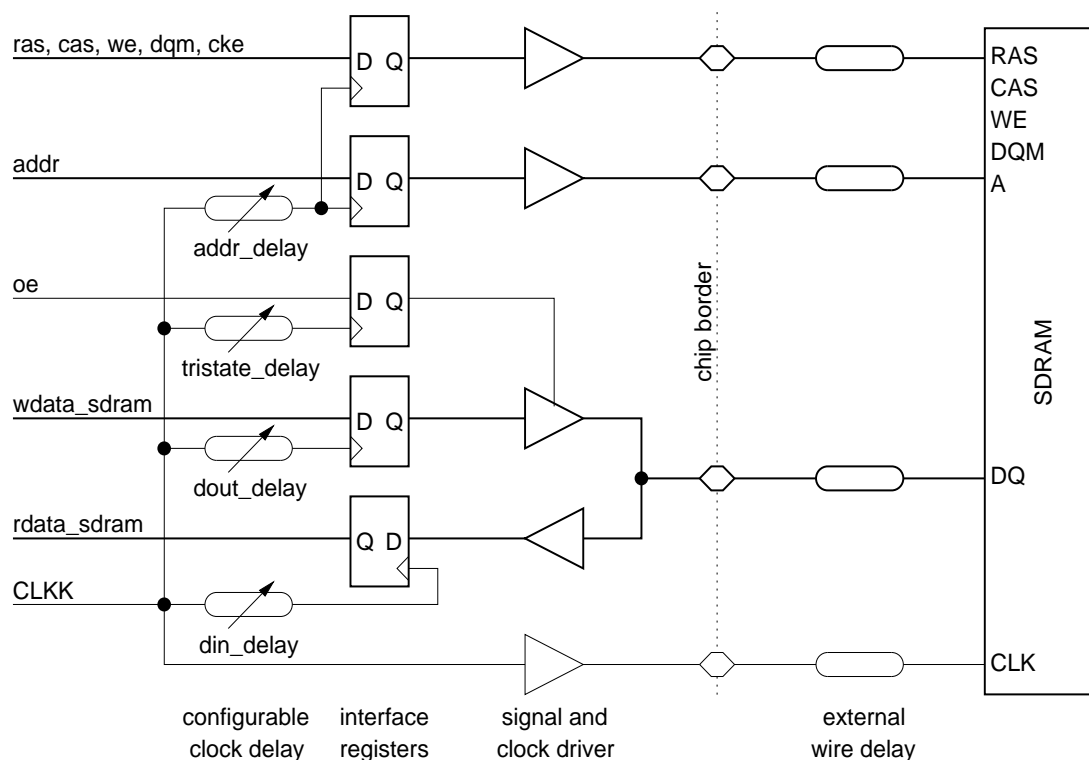
The implemented solution uses configurable delays for each signal group (addresses and control signals, tri state control, write data and read data). In this way it's possible to compensate the influence of the board layout in a comfortable way.

Registers for the SDRAM interface control the timing of the ports. Four different timings have to be satisfied:

- Hold time for SDRAM input pins address, command and DQM ( $t_{DCBTaout}$ )
- Hold time for SDRAM data input pins ( $t_{DCBTdout}$ )

- Sampling time for SDRAM data outputs ( $t_{DCBTdin}$ )
- Delay for switching the tri state buffer enable signal ( $t_{DCBToe}$ )

Figure 2-2 shows the schematic of the implemented SDRAM interface circuitry. Variable clock line delays



**Figure 2-2:** Design of SDRAM Interface

are implemented as buffer chains with multiplexed taps. The multiplexers respective the resulting delays are controlled by a two-bit value for each signal group in the delay configuration byte. Under typical conditions a programmable range from nearly 1ns up to 4ns is possible in steps of 1ns.

Recommended values for the interface setup are  $t_{DCBTaout}=2ns$ ,  $t_{DCBTdout}=2ns$ ,  $t_{DCBTdin}=3ns$  and  $t_{DCBToe}=1ns$ .

## 2.2 Integrated SDRAM Implementation (Jasmine)

Jasmine has no interface to external SDRAM. Delay adjustment is not needed and not implemented for the integrated solution.

## 3 Configuration

### 3.1 Register Summary

A summary of Initialization control registers is given in Table 3-1. Defaults are for 100 MHz operation frequency. Address definitions for symbolic word addresses are in the file 'cbp\_const.v'.

**Table 3-1:** Configuration Information of SDC

Symbol	Bits	Description	Reset Value
SDWAIT_OPT	[20]	Interleave Opt <sup>a</sup> : Execute Precharge and Activate during running Bursts of previous access.	1 unused Jasmine
SDWAIT_TRP	[19:16]	tRP: RAS Precharge Time (PRE -> ACTV, default 2 wait states)	2
SDWAIT_TRRD	[15:12]	tRRD <sup>b</sup> : RAS to RAS Bank Active Delay Time (ACTV -> ACTV, default 1 wait state)	1 unused Jasmine
SDWAIT_TRAS	[11:8]	tRAS: RAS Active Time (ACTV -> PRE, default 5 wait states)	5
SDWAIT_TRCD	[7:4]	tRCD: RAS to CAS Delay Time (ACTV -> READ WRIT, default 2 wait states)	2
SDWAIT_TRW	[3:0]	tRW: Read to Write Recovery Time (READ -> WRIT, default 7 wait states) Wrong reset value, Jasmine requires 7/5, Lavender 10/8, depending on used AAF or not.	3 !
SDINIT	[15:0]	Init Period: Power On Stabilization Time (default 20000)	20000
SDRFSH	[15:0]	Refresh Period: Single Row Refresh Period (default 1600)	1600
SDSEQRAM[] [0:63] Lavender [0:31] Jasmine	[13:0]	Sequencer RAM [13:0], 64 words [13:7]addr, [6:4]instr, [3:0]{ras,cas,we,ap} (Jasmine has [12:7] address argument)	undefined
SDMODE	[12:0]	MRS <sup>c</sup> : SDRAM Mode Register [9] burst write enable, [6:4] CL, [3] interleave burst, [2:0] burst length (0:3)=1,2,4,8 / 7=full (default 0x033)	0x0033 Lavender only
SDIF_TAO	[7:6]	tDCBTaout <sup>d</sup> : Address output register clock delay (controls also CKE, DQM, RAS, CAS, WE outputs)	0 Lavender only
SDIF_TDO	[5:4]	tDCBTdout: Data output register clock delay	0 Lavender only
SDIF_TDI	[3:2]	tDCBTdin: Data input register clock delay	0 Lavender only
SDIF_TOE	[1:0]	tDCBToe: Output enable register clock delay	0 Lavender only

**Table 3-1:** Configuration Information of SDC

Symbol	Bits	Description	Reset Value
SDCFLAG_BUSY	[0]	CBPbusy: Set these flag before changing the power on initialization period or the sequencer RAM. Reset it after the access make changes take affect.	0
SDCFLAG_DQMEN	[1]	DQM partial write feature <sup>e</sup> : Optimize byte/8bpp and half word/16bpp access if set to 1.	0 Jasmine only
PHA[0:15]	[22:12] [19:10]	Layer Start Addresses: Row address offset for start position of each layer. First bit positions for Lavender, second for Jasmine. [22/19:0] can be handled as byte address, but only shown bits are stored.	undefined
DSZ_X[0:15]	[29:16]	Layer Widths: X component of Layer Size in pixel	undefined
CSPC_CSC[0:15]	[3:0]	Color Depth Table: Color definition code for evaluation of the number of bits per pixel (bpp)	undefined

a.Lavender only. Jasmine works with single bank.

b.Has no effect for Jasmine.

c.Fixed and not accessible for Jasmine.

d.DCBT interface timing adjustable for Lavender only.

e.Jasmine only.

## 3.2 Core clock dependent Timing Configuration

### 3.2.1 General Setup

SDRAM access wait states and refresh periods are configurable to support a wide range of scalability in matter of system performance and power consumption. Additional to the row refresh time out value the refresh sequence in the micro program sequencer is adaptable for its dedicated core frequency.

The configuration tool generates optimized settings for a given core clock frequency to met best performance result. If a fixed setting should be used over a certain frequency range (i.e. if clock scaling is used without the effort spending for re-configuration) the minimum frequency is required to calculate the refresh period and the highest frequency should be used to calculate the values for the wait state timers. Thus refresh condition and minimum access timing is always satisfied.

### 3.2.2 Refresh Configuration for integrated DRAM (Jasmine)

Setup of minimum refresh rate is based on core clock cycles. Thus the value for the refresh period has to be configured for its dedicated core clock frequency. Additional to the core frequency, maximum junction temperature has significant influence on refresh period.

- $T_{jmax} = \text{up to } 100 \text{ degC} : t_{REF} = 16.4\text{ms}$
- $T_{jmax} = 101 \text{ to } 110 \text{ degC} : t_{REF} = 8.2\text{ms}$
- $T_{jmax} = 111 \text{ to } 125 \text{ degC} : t_{REF} = 4.1\text{ms}$

Assumed Row Refresh duration for all 1024 rows is evenly distributed to refresh all rows within above specification, refresh timing of 16/8/4 us have to be set up. For automotive temperature range 4 us are required.





---

## ***B-4 Pixel Processor (PP)***



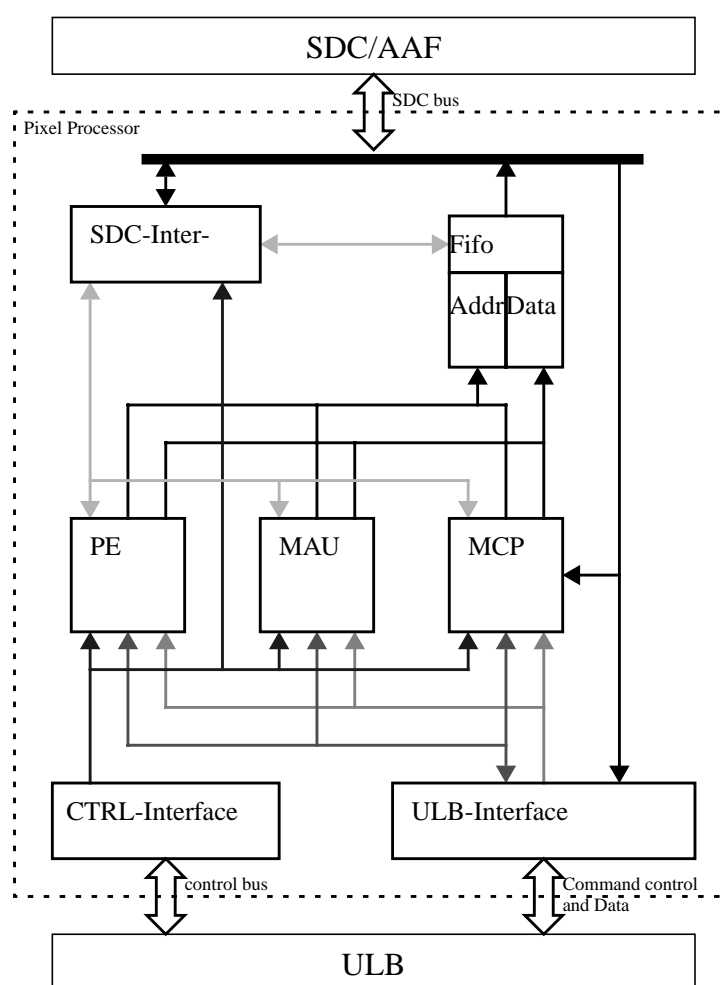
# 1 Functional Description

The Pixel Processor (PP) is a component of the Graphics Display Controller (GDC), which realizes the main functions “drawing of geometrical functions” and “writing and reading single pixel” to and from Video RAM. It is implemented in the GDC design between User Logic Bus Interface (ULB) and the SDRAM Controller (SDC)/Anti Aliasing Filter (AAF), in reference to block diagram in GDC specification. The PP consists of three separate devices (execution devices) and other submodules. The execution devices are the Pixel Engine (PE) for drawing function, the Memory Access Unit (MAU) for single pixel access and the Memory CoPy unit (MCP) for copying rectangular areas. The submodules are Control Interface, ULB Interface, the SDC Interface and the fifos for pixel addresses and pixel data.

More information about the internal module can be found in the corresponding chapters and sections.

## 1.1 Overview

### 1.1.1 PP Structure



**Figure 1-1:** PP block diagram

The internal structure of PP is shown in figure 1-1. It consists of the execution devices (PP, MAU, MCP) and the submodules Control interface, ULB interface and SDC interface. The ULB block in the figure 1-1

is only for understanding PP in context of GDC and represents the ULB functionality, such as command control signals and data bus for reading and writing.

### 1.1.2 Function of submodules

The PE is a unit, which realizes the main function of PP, which are drawing of geometrical figures and writing compressed and uncompressed pictures into the Video RAM. The supported commands are:

- Drawing Commands
  - **DwLine** (draw lines)
  - **DwRect** (draw arectangular areas)
  - **DwPoly** (draw a polygon)
- Bitmap Commands
  - **PutBM** (write an uncompressed bitmap into Video RAM)
  - **PutCP** (write an RLE compressed bitmap into Video RAM)
  - **PutTxtBM** (write an uncompressed 1 bit pixel mask as a bitmap with a higher colour depth into Video RAM)
  - **PutTxtCP** (write an RLE compressed 1 bit pixel mask as a bitmap with a higher colour depth into Video RAM).

The second unit in PP is the MAU. It realizes the single pixel access for reading and writing. Following commands can be used by the programmer:

- Pixel Commands
  - **PutPixel** (set one pixel on the display)
  - **PutPxFC** (set one pixel with a fixed colour on the display)
  - **XChPixel** (read-modify-write of a single pixel)
  - **GetPixel** (read one pixel from the VideoRam)
  - **PutPxWd** (write a 32bit data word with a number of pixels into the Video RAM; the number of pixels depends on the colour depth of the target address).

The third execution device is the MCP. It gives a simple way for the programmer to copy rectangular areas from the source layer to the target layer. The only command, which can be used is:

- Memory to Memory Commands
  - **MemCP** (copy rectangular area from one layer to another)

The SDCI is a component, which manages the access to the SDC. The SDCI collects pixel addresses and data to packages, which should be transferred to the Video RAM with one request. The addresses and data are collected in the fifos (address and data fifo), every one of them has a size of 32bit\*(64+2) words.

The Control interface is connected to the internal control bus system of the GDC, which allows the programmer to have access to the configuration registers of all sub modules or macros. All configuration registers of the PP are connected to the control bus system.

## 1.2 Configuration Registers

The following table 1-1 shows the configuration registers of PP and their initial values.

**Table 1-1:** register list of the PP

Address	Name	Width	Bits	Comment	Initial
1240h 1244h 1248h 124Ch 1250h 1254h 1258h 125Ch 1260h 1264h 1268h 126Ch 1270h 1274h 1278h 127Ch	CSPC_CSC[0:15]	4	[3:0]	color space code (to calculate color depth) for layer[0:15] color definition of layer 0 color definition of layer 1 color definition of layer 2 color definition of layer 3 color definition of layer 4 color definition of layer 5 color definition of layer 6 color definition of layer 7 color definition of layer 8 color definition of layer 9 color definition of layer 10 color definition of layer 11 color definition of layer 12 color definition of layer 13 color definition of layer 14 color definition of layer 15	0h
4100h	BGCOL_EN BGCOL_COL <sup>a</sup>	1 24	[24] [23:0]	enable background color background color data (depends on colour depth) background pixel colour data is used by commands PutTxtBM and PutTxtCP	0h
4104h	FGCOL	24	[23:0]	foreground colour data (depends on colour depth)	0h
4108h	IGNORCOL_EN IGNORCOL_COL	1 24	[24] [23:0]	enable ignore colour ignored colour data (depends on colour depth) ignored pixel colour data is used by commands PutBM and PutCP	0h
410Ch	LINECOL	24	[23:0]	line colour data (depends on colour depth) line pixel colour data is used by command DwLine	0h
4110h	PIXCOL	24	[23:0]	pixel colour data (depends on colour depth) pixel colour data is used by command PutPxFC	0h
4114h	PLCOL	24	[23:0]	polygon colour data (depends on colour depth) line pixel colour data is used by command DwPoly	0h

**Table 1-1:** register list of the PP

Address	Name	Width	Bits	Comment	Initial
4118h	RECTCOL	24	[23:0]	rectangle colour data (depends on colour depth) line pixel colour data is used by command DwRect	0h
411Ch	XYMAX_XMAX XYMAX_YMAX	14 14	[29:16] [13:0]	endpoint X-dimension endpoint Y-dimension bitmap shape description (stop address incrementation) for the commands PutBM, PutCP, PutTxtBM and PutTxtCP	0h
4120h	XYMIN_XMIN XYMIN_YMIN	14 14	[29:16] [13:0]	startpoint X-dimension startpoint Y-dimension bitmap shape description (start address incrementation) for the commands PutBM, PutCP, PutTxtBM and PutTxtCP	0h
4124h	PPCMD_ULAY PPCMD_LAY PPCMD_EN PPCMD_DIR PPCMD_MIR	1 4 1 1 2	[28] [27:24] [16] [8] [1:0]	use target layer register target layer enable anti aliasing filter bitmap direction bitmap mirror  Target layer information is used by PutBM, PutCP, PutTxtBM, PutTxtCP any time. It is used by DwLine, DwRect, DwPoly if ULAY enabled. Direction is relevant for PutBM, PutCP, PutTxtBM and PutTxtCP (0=horizontal, 1=vertical). Mirror <sup>b</sup> for PutBM, PutCP, PutTxtBM, PutTxtCP (0h=without mirroring, 1h=X-axis, 2h=Y-axis, 3h=X- and Y-axis).	0h
4128	SDCPRIO	3	[2:0]	PP priority for SDC-request	0h
412Ch	REQCNT	8	[7:0]	maximum number of data words, which should be transferred in one Video RAM access cycle (packet size). <sup>c</sup> word count = REQCNT+1	0h
4130h	READINIT	1	[0]	read config register, double buffer read access behavior: 0=internal buffered register, 1=config register. <sup>d</sup>	0h

a. Note! All colour data in the registers is right aligned (lsb) and depends on the colour depth/resolution.

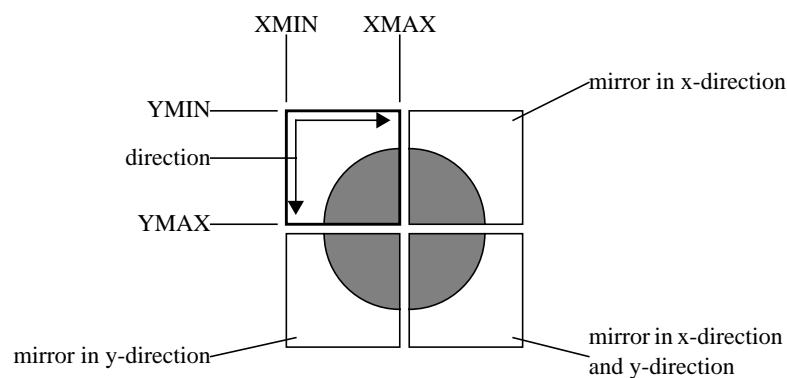
b. The mirror functionality means doubling at the limitation lines, which are defined by XMAX or YMAX (see figure 1-2).

- c. REQCNT is important for bandwidth considerations. Large values give fast PP data throughput but increasing risk of shortage for real-time devices GPU/VIC. If REQCNT is different from 0, data can remain in PP FIFOs (not sufficient amount for packetized transfer). The remaining data is flushed with start of the next command, i.e. if the drawing is finished with a NoOp command.
- d. To avoid consistency problems during read-modify-write access it is recommended to set READINIT behavior to '1'. Thus read and write accesses use the same register. The copy procedure from configuration register to the internal buffered register is controlled by the pixel processor. The internal value is fixed during running commands.

## 1.3 Special Command Options

### 1.3.1 Bitmap Mirror

Bitmap mirror option is for enhancement of symmetric objects. Original object is drawn in any case. It is possible in one or both X- and Y-direction. Thus only the half or the quarter of an bitmap needs to be transferred. Mirroring is usable for bitmap commands PutBM, PutCP, PutTxtBM and PutTxtCP. Mirroring axis are defined by the end point coordinates.



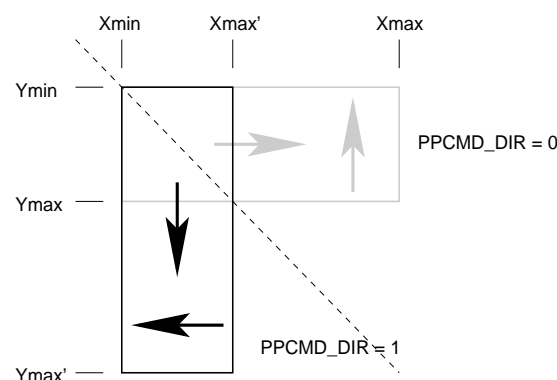
**Figure 1-2:** Mirroring Function

### 1.3.2 Bitmap Direction

Bitmap direction option swaps X- and Y- coordinates relative to the start point. Original bitmap is not drawn. It could be understood as flipping the object at an axis through the start point with an angle of 45 degree. If PPCMD\_DIR = 1 the new end point calculates to

$$X_{max}' = X_{min} + Y_{max} - Y_{min}$$

$$Y_{max}' = Y_{min} + X_{max} - X_{min}$$



**Figure 1-3:** Direction Option

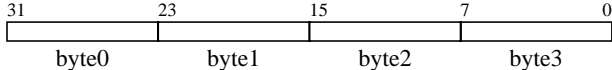
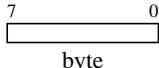
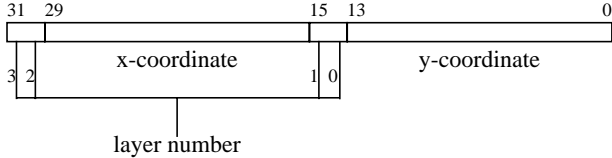
## 2 Format Definitions

This chapter describes the colour and data formats of all PP specific interfaces, such as ULB interface or SDC interface. You can also find explanation of all used symbols.

### 2.1 Legend of symbols

The table describes the mnemonics of data formats.

**Table 2-1: Mnemonics and Symbols**

Mnemonic & Symbols	Description
[...]	<p>data double word (32bit):</p>  <ul style="list-style-type: none"> <li>• [single colour data]: one pixel</li> <li>• [colour data]: more than one pixel in one word (uncompressed)</li> <li>• [coded data]: RLE compressed pixel data</li> <li>• [colour enable data]: 1bpp pixel mask (uncompressed)</li> <li>• [coded colour enable data]: RLE compressed 1bpp pixel mask</li> </ul>
<...>	<p>data byte (8bit):</p> 
[L, X, Y] or [L <sub>high</sub> , X, L <sub>low</sub> , Y]	<p>pixel address (32bit):</p> 
{...}	<ul style="list-style-type: none"> <li>• {X, Y}: pixel coordinates (2*14bit)</li> <li>• {L<sub>high</sub>, L<sub>low</sub>}: layer number (2*2bit)</li> <li>• {reg}: value of register, e.g. RCOLOR</li> </ul>



**Table 2-1:** Mnemonics and Symbols

Mnemonic & Symbols	Description
(...)	<p>data package: It consists of one or more 32bit data words, which contain all necessary information for the current command. e.g. ([X, Y], [L<sub>high</sub>, X, L<sub>low</sub>, Y]) This is the data (start point, end point and layer) for the command DwLine, two data words are used.</p>

## 2.2 Data Formats at ULB Interface

Color data at ULB interface (IFIFO/OFIFO) and for configuration registers is LSB aligned. Only data for PutPxWd is an exceptional case, it delivers data in physical form as stored in Video RAM.

The ULB delivers data in three formats:

- pixel addresses
- single pixel data: all single colour data is right aligned in the data word; the number of used bits depends on the colour depth.
- pixel data stream (compressed or decompressed):

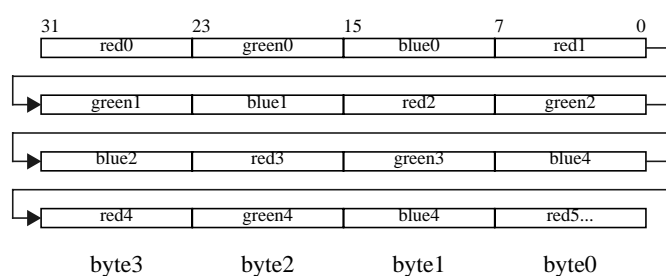
**Table 2-2:** Symbols of RLE Compression

Resolution	Mode	Command Byte	Colour Bytes
bpp1	compressed	<1NNN NNNC>	colour information in command byte (bit[0])
	uncompressed	<0NNN NNNN>	(NNNN NNN+1 mod 8) bytes with colour data <CCCC CCCC> (NNNNNNN+1 rem 8) > 0: last byte may be incomplete; e.g. 4: <CCCC xxxxx>
bpp2	compressed	<1NNN NNNN>	1 byte incomplete <CCxx xxxx>
	uncompressed	<0NNN NNNN>	(NNNNNNN+1 mod 4) bytes with colour data <CCCC CCCC> (NNNNNNN+1 rem 4) > 0: last byte may be incomplete; e.g. 2: <CCCC xxxxx>
bpp4	compressed	<1NNN NNNN>	1 byte incomplete <CCCC xxxx>
	uncompressed	<0NNN NNNN>	(NNNNNNN+1 mod 2) bytes with colour data <CCCC CCCC> (NNNNNNN+1 rem 2) > 0: last byte may be incomplete; e.g. 1: <CCCC xxxxx>
bpp8	compressed	<1NNN NNNN>	1 colour byte <CCCC CCCC>
	uncompressed	<0NNN NNNN>	(NNNNNNN + 1) bytes with colour data <CCCC CCCC>
RGB555, RGB565	compressed	<1NNN NNNN>	2 bytes colour data (<CCCC CCCC>, <CCCC CCCC>)
	uncompressed	<0NNN NNNN>	(NNNNNNN + 1)*2 bytes with colour data (<CCCC CCCC>, <CCCC CCCC>)

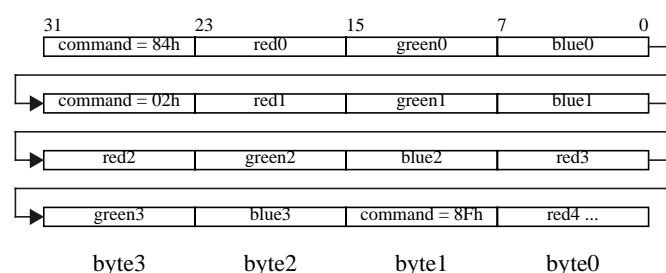
**Table 2-2:** Symbols of RLE Compression

Resolution	Mode	Command Byte	Colour Bytes
RGB888	compressed	<1NNN NNNN>	3 bytes colour data (<CCCC CCCC>, <CCCC CCCC>, <CCCC CCCC>)
	uncompressed	<0NNN NNNN>	(NNNNNNN + 1)*3 bytes with colour data (<CCCC CCCC>, <CCCC CCCC>, <CCCC CCCC>)

**PutBM:** uncompressed continuous data stream (RGB888)->[colour data]



**PutCP:** compressed continuous data stream (RGB888) ->[coded data]

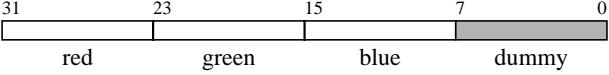
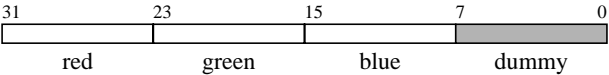
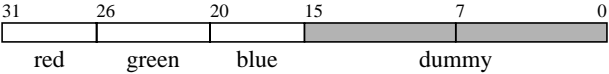
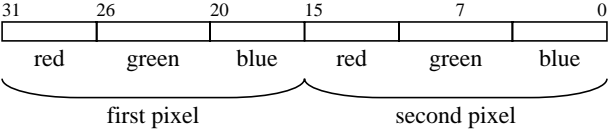
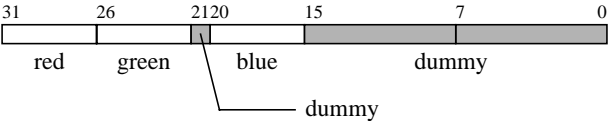
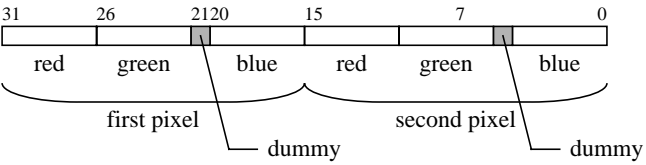
**Figure 2-1:** Examples of continuous data stream (ULB/MCU side)

## 2.3 Data Formats for Video RAM / SDC Interface

At the SDC interface all colour formats are possible and supported. Color data is transferred and stored in MSB aligned form. This is opposite to the IFIFO/OFIFO data at ULB interface.

All pixel data in block (single pixel) or burst mode are left aligned. In burst mode more than one pixel data can be in one data word.

**Table 2-3:** Color formats on the SDC interface

colour Format	Comment
RGB888	<p>true colour format</p> <ul style="list-style-type: none"> <li>single pixel ([single colour data])            </li> <li>pixel burst ([colour data])            </li> </ul>
RGB565	<p>high colour format</p> <ul style="list-style-type: none"> <li>single pixel ([single colour data])            </li> <li>pixel burst ([colour data], one data word of the burst and maximum of two pixel per word)            </li> </ul>
RGB555	<p>high colour format</p> <ul style="list-style-type: none"> <li>single pixel ([single colour data])            </li> <li>pixel burst ([colour data], one data word of the burst and maximum of two pixel per word)            </li> </ul>

**Table 2-3:** Color formats on the SDC interface

colour Format	Comment
bpp8, bpp4, bpp2, bpp1	<p>colour index format</p> <p>These colour formats represents index information for the CLUT (Colour Lock-up Table) and not colour information in RGB format. The maximum of numbers in one word depends on the colour resolution (bits per pixel).</p> <ul style="list-style-type: none"> <li>single pixel ([single colour data]) <div data-bbox="582 593 1204 750"> <p>31 23 15 7 0</p> <p>dummy</p> <p>colour index byte, depends on colour resolution; the left aligned bits are used</p> </div> </li> <li>pixel burst ([colour data]) <div data-bbox="582 840 1204 963"> <p>31 23 15 7 0</p> <p>first pixel</p> <p>31-(bpp-1)</p> </div> </li> </ul>

---

## ***B-5 Antialiasing Filter (AAF)***

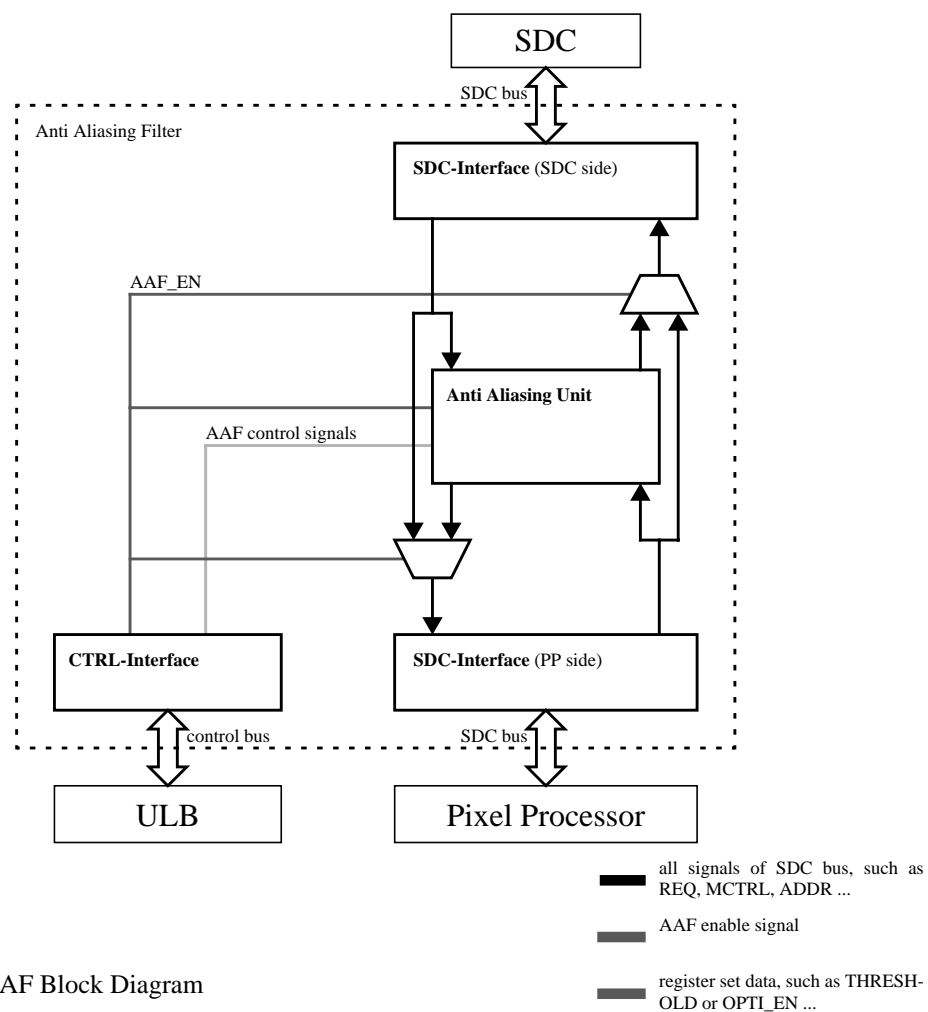


# 1 Functional Description

The Anti Aliasing Filter (AAF) is a component of the Graphics Display Controller (GDC), which realizes the anti aliasing filter function for some pixel commands. It is implemented in the GDC design between the Pixel Processor (PP) and the SDRAM Controller (SDC), in reference to block diagram in GDC Specification. The AAF processes data on the fly which should be stored in the Video RAM. Only for true or high colour resolution pixel data an applicable result could be expected. Averaging of indexed colours may lead to unexpected effects.

## 1.1 AAF Overview

### 1.1.1 Top Level Structure



**Figure 1-1:** AAF Block Diagram

The internal structure of AAF is shown in Figure 1-1. It consists of a bypass multiplexor, a control block, the antialiasing unit (AAU) itself and two compatible interfaces to the SDC and PP modules.

The bypass is needed to switch the AAF in a transparent (disabled) mode. If AAU is disabled and the bypass is active the clock supply of the whole AAF module can be switched off for power saving purposes. Additional if AAF is not used the SDRAM timing for read-to-write recovery can be made faster. The bypass

mode guarantees a direct connection between PP and SDC without pixel data and pixel address manipulation by AAF (AAU is off).

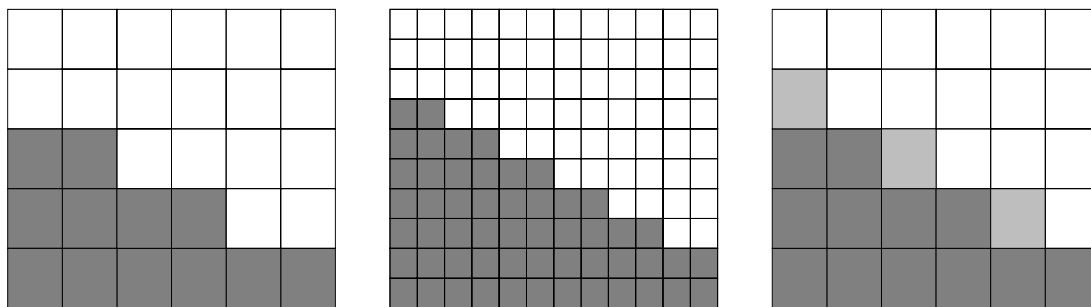
The control block realizes reading or writing of configuration registers. The anti aliasing unit manipulates the pixel data and manages the access to PP and SDC.

### 1.1.2 AAU Function

The main function of the anti aliasing algorithm is to reduce the staircasing effect. This effect is a result of projecting (drawing) objects onto the physical pixel resolution of the display. To get a higher image quality the implemented super sampling algorithm could be used to avoid stair effects on the edges of objects, e.g. lines or circles. In this case the anti aliasing is done on the fly, that means that during the drawing of objects the object itself is antialiased with the current background. This results in smoother edges by averaging foreground and background color information.

#### 1.1.2.1 Super Sampling

The objects are drawn at a higher resolution, which per definition leads to less staircasing. This virtual resolution is doubled in X and Y direction for 2x2 super sampling or four times in both dimensions for 4x4 super sampling. The image is then converted to the normal resolution by averaging the sub pixels to a new resulting pixel, which is stored in video RAM. Foreground pixels are generated by PixelProcessor or written through its interface. Background pixels are read from memory.



**Figure 1-2:** Antialiasing with super sampling. From left to right: Without antialiasing, doubled resolution with sub-pixels, averaging of sub-pixels.

#### 1.1.2.2 Drawing Resolution

The super sampling method requires doubled or four times the drawing resolution. If antialiasing is enabled, coordinates have to be doubled or multiplied by four. The Pixel Processor did not enhance the resolution automatically. It is the task of the application to work on the finer resolution of the sub-pixel grid.

A side-effect of the higher resolution is that the amount of generated pixel data is four or sixteen times higher compared to the original picture without antialiasing. Additionally each write access converts to a read-modify-write access due to the fact that background information is required. This has to be considered in terms of drawing performance. To reduce the effect of the high number of read-modify-write (RMW) accesses to the same pixel coordinate write-back optimization is implemented for subsequent writes to the same pixel address. It is recommended to switch on optimization by setting OPTI\_EN to '1'.

#### 1.1.2.3 Data and Address Processing

The algorithm is based on single pixel processing due to sequential averaging in video memory. Therefore PP is forced to block mode, each pixel data requires its own address information. Packet data formats are not supported (e.g. PutPxWd). Incoming PP requests of variable block size were divided into single RMW accesses to the SDC. That's why some restrictions apply during AAF usage, which will be discussed in section 1.3.1.



Formulas for sequential averaging of foreground and background data are shown in table 1-1. Pixel data from PP is fg, background from memory is bg and fg' is the new resulting pixel, which is written to video memory. The low precision of the 2x2 super sampling can be compensated by adaptive coefficient selection, controlled by color difference thresholds. The value of the thresholds depends on human eye sensitiv-

**Table 1-1:** Formulas for sequential averaging

Formula	Condition
$fg' = (2 fg + 2 bg) / 4$	2x2 super sampling size. If difference between fg and bg color is below threshold. Threshold1 applies if fg<bg, otherwise threshold2.
$fg' = (1.5 fg + 3 bg) / 4$	2x2 super sampling size. If difference between fg and bg color is greater or equal threshold. Threshold1 applies if fg<bg, otherwise threshold2.
$fg' = (1.5 fg + 15 bg) / 16$	4x4 super sampling size. No threshold based coefficient adaption required.

ity on chrominance contrast and differs for various types of graphics. Right setup values could be evaluated empirically.

Addresses of pixel coordinates are divided by 2 in 2x2 super sampling mode for each X/Y dimension. In 4x4 super sampling mode pixel address of resulting pixel is divided by 4. Thus 4 or 16 sub-pixels of the generated high-resolution object are mapped to the same pixel address in video memory. The sequential averaging is the result of the multiple applied formula on the same memory address.

## 1.2 Configuration Registers

The following table shows the configuration registers of AAF and their initial values.

**Table 1-2:** AAF related Registers

Address	Name	Width	Comment	Init. Value
4124h	PPCMD_EN	1	bit[16] - AAF_EN: enable anti aliasing filter (high active)	0h
4500h	AATR_TH2 AATR_TH1	16	bit[15:8] - threshold2 bit[7:0] -threshold1 for colour channel red	0h FFh
4504h	AATG_TH2 AATG_TH1	16	bit[15:8] - threshold2 bit[7:0] - threshold1 for colour channel green	0h FFh
4508h	AATB_TH2 AATB_TH1	16	bit[15:8] - threshold2 bit[7:0] - threshold1 for colour channel blue	0h FFh
450Ch	AAOE_WBO AAOE_BX4	1	bit[0] - OPTI_EN: write back optimization bit[1] - Operator size (0=2x2, 1=4x4)	0h

### Note!

The number of the used bits of threshold registers depends on colour depth. That means:

- RGB888: for all colour channels all bits of threshold registers are used by AAF algorithm
- RGB565: for the red and blue colour channel only the bit[7:3] are used for calculation and for the green channel the bit[7:2]

- RGB555: for all colour channels only the bit[7:3] are used for calculation

## 1.3 Application Notes

### 1.3.1 Restrictions due to Usage of AAF

The AAF does not support all modes of the PP and SDC functionality. For the using of AAF note there are some restrictions, which are described in this section.

Table 1-3 shows all pixel processor commands, which can be used or are not allowed to be used or make no sense commonly used with the AAF.

**Table 1-3:** Support of PP commands

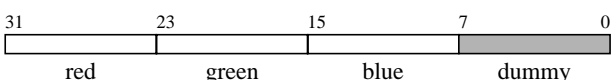
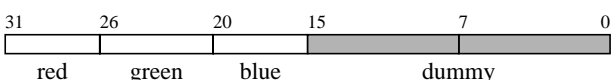
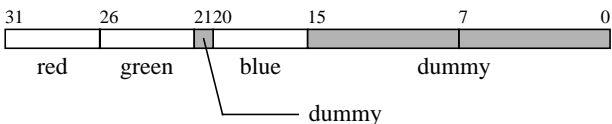
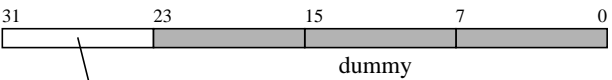
Command	Access Mode	Comment
DwLine, DwPoly, DwRect, PutBM, PutCP, PutTxtBm, PutTxtCP, PutTxtCP, PutPixel, PutPxFC	write (block mode)	pixel address manipulation (sub pixel access <sup>a</sup> ) and pixel data manipulation
XChPixel	read-modify-write (block mode)	pixel address manipulation (sub pixel access), no pixel data manipulation for read data and pixel data manipulation for write data <b>Note!</b> The RDATA information is stored in the ULB output Fifo. Do not use this command with enabled AAF!
GetPixel	only read (block mode)	no pixel address manipulation (real pixel access <sup>b</sup> ) and no pixel data manipulation; <b>Note!</b> The RDATA information is stored in the ULB output Fifo. Do not use this command with enabled AAF!
MemCP	sequential read and write (block mode)	read access: like GetPixel write access: like PutPixel <b>Note!</b> Make sure that the source area does not overlap with the target area!
PutPxWd	write (burst mode)	<b>Prohibition! (did not work)</b>

- Sub pixel access means that the pixel processor wants to read or write into the sub pixel space, which have the double resolution in horizontal and vertical direction (one real pixel on the display/layer have four sub pixel).
- Real pixel access means that the pixel processor wants to read or write a real pixel (a physical pixel) of a the display/layer.

### 1.3.2 Supported Colour Formats

The AAF supports only pixel data manipulation for true or high colour resolutions like RGB888, RGB565 and RGB555.

**Table 1-4:** supported colour formats

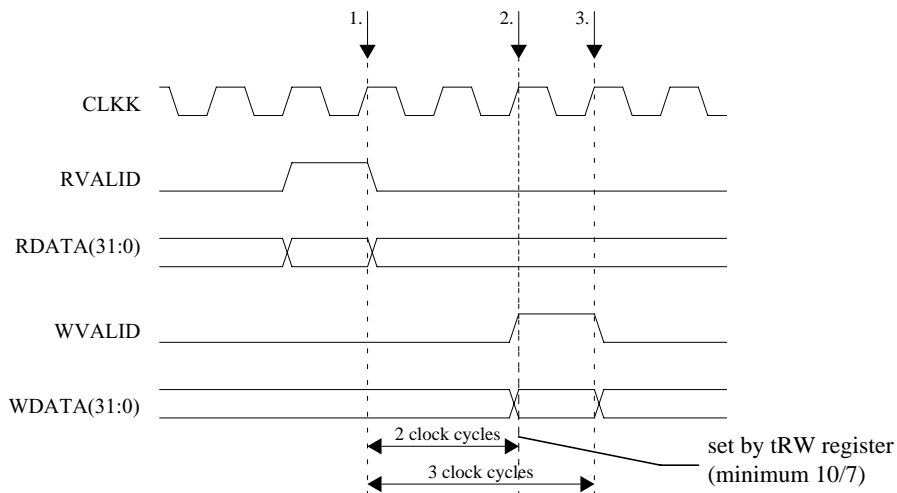
colour Format	Comment
RGB888	<p>true colour format (single pixel <sup>a)</sup>)</p> 
RGB565	<p>high colour format (single pixel)</p> 
RGB555	<p>high colour format (single pixel)</p> 
bpp8, bpp4, bpp2, bpp1	<p>colour index format (single pixel)</p>  <p>colour index byte, depend on colour resolution the left aligned bits are used</p> <p>This colour formats can not be correctly manipulated by the AAF, because the pixel data represents index information for the CLUT (Colour Lock-up Table) and not colour information in RGB format. If the AAF is used and one of this colour formats should be transferred to the SDC, the AAF calculates a new colour index, which is stored in the SDC. The visible result at the display may not be the intended one.</p> <p><b>Note!</b> To avoid such problems, the programmer should disable the AAF for this colour format access.</p>

- a. A single pixel means one pixel in a 32bit data word at the SDC bus (SDC format in block mode; pixel data is MSB aligned).

### 1.3.3 Related SDC Configuration

The information in this section is very important for the programmer, because wrong timing at the SDC can hang up the AAF. The 'read to write recovery time' (tRW) in the SDC should be set to the minimum value of 10 for Lavender or 7 for Jasmine. This timing parameter selects a delay between read and write access to the SDRAM and thus it influences the difference between read data and write data appear on the internal

bus connected to the AAF. AAF requires at minimum two extra clock cycles which is needed for internal data processing.



**Figure 1-3:** Timing for read-modify-write on SDC bus for AAF access

Description of events at clock cycles from figure 1-3:

1. At this edge the AAF takes the pixel data of the old pixel from RDATA bus (SDC side).
2. At this event the AAF sets the result at the WDATA bus.
3. SDC takes the pixel data of new pixel from the WDATA bus.

**Note!**

The delay between RVALID (1.) and WVALID (2.) can be bigger, but not smaller!

To gain more system performance, tRW is possible to reduce to 8 (Lavender) or 5 (Jasmine) if Antialiasing Filter is bypassed. No other module has any requirements to this read-write timing. The minimum values of 8/5 guarantee that there will be no bus conflict at the SDRAM interface itself. The setting of tRW is independent from core clock frequency.

**Table 1-5:** Minimum settings for tRW

	Jasmine	Lavender
<b>AAF on</b>	7	10
<b>AAF off</b>	5	8

There is no requirement to reduce tRW setting to its minimum. It is only an aspect of performance tuning. Additionally, it did not that much influence system performance, it is safe to setup a value of 10 or 7 all the time.

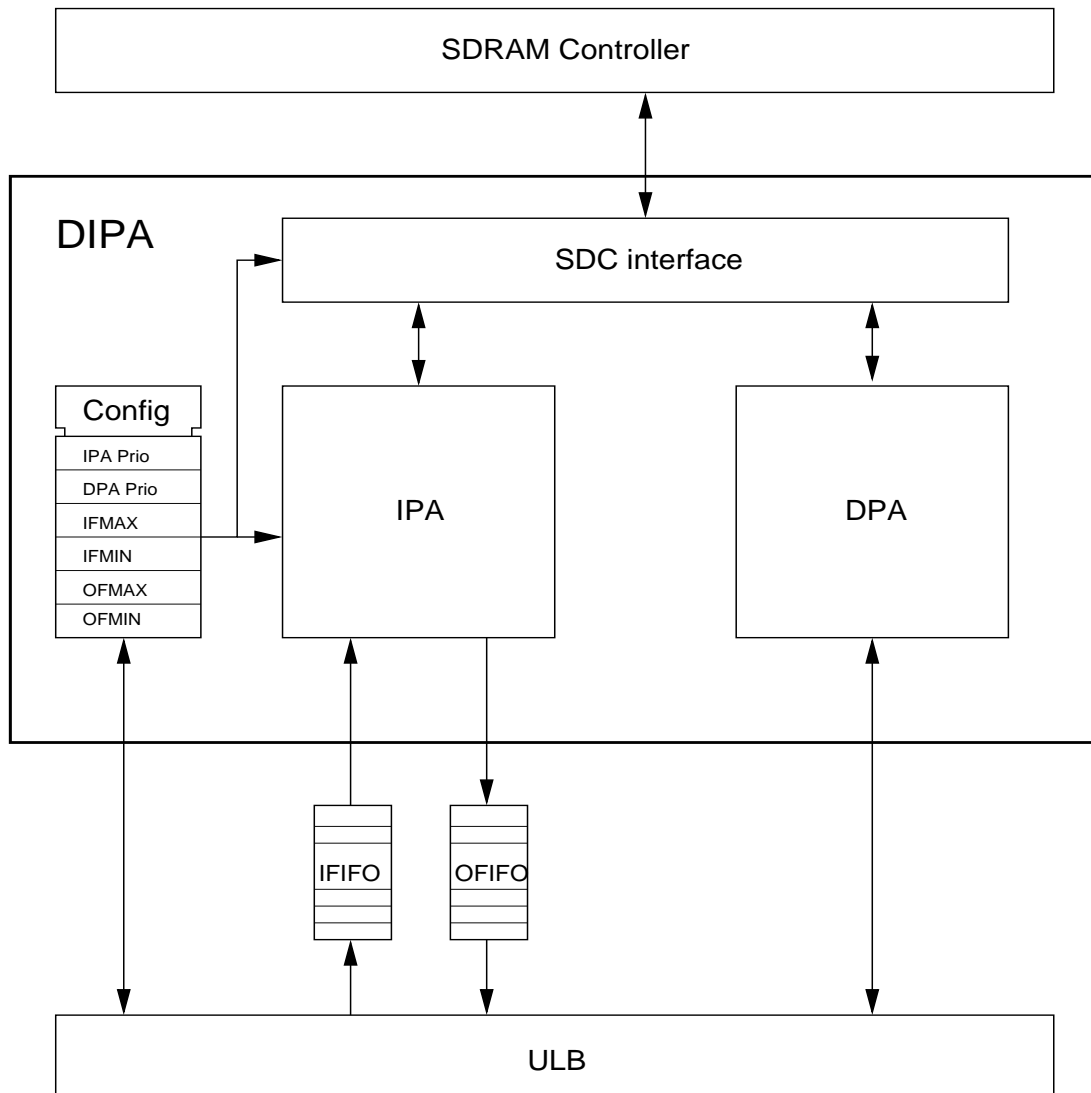
---

## ***B-6 Direct and Indirect Physical Memory Access Unit (DIPA)***



# 1 Overview

The purpose of the DIPA device is to provide video memory (frame buffer) access methods for the MCU without the need of the Pixel Processor inside the graphic display controller (GDC) to be used. This is performed by using physical SDRAM write and read access. The data words in video memory can be accessed with raw physical addresses.



**Figure 1-1:** DIPA Block Diagram within GDC environment

The function divides into two blocks, DPA (Direct Physical memory Access) and IPA (Indirect Physical memory Access).

DPA is for direct memory mapped video RAM access, address range of video RAM has a certain offset regarding GDC internal mapping and GDC Chip Select address mapping of MCU address space. Data transfer over DPA are single word, half word or byte accesses. Each single access has to be arbitrated with competing GDC devices GPU, VIC and PP by the SDRAM Controller (SDC), therefore this method of direct frame buffer access is relative slow.

IPA offers also the possibility to have physical access to the video RAM. Main advantage is that the transfer is executed in a buffered manner over the input and output FIFOs. Addressing is done without any address offset calculation required. Physical address is transferred as parameter from 0 to the upper bounds of video memory size. Data blocks can be transferred with a given block size and start address. The slow-down due

to the needed SDC arbitration time has much less influence when multiple data words are transferred as packet at once. Additional performance increase is reached because of FIFO buffering. IPA is used if the following commands are executed:

- PutPA, write address and data field of variable size via Input FIFO
- GetPA (n), read n data words via Output FIFO

The SDC interface multiplexes between IPA and DPA devices and controls its concurrence. DPA has precedence over IPA.

There is no necessity for physical access of video data only. Not used video memory, i.e. if the memory is not assigned in the Layer Description Record (LDR, see GDC registers), can be used for general purposes. If pixel data should be accessed the logical to physical address mapping has to be considered (see SDRAM Controller Specification). For non-image data addressing can be in a linear way.



## 2 Configuration Registers

### 2.1 Register List

Configuration of DIPA internal registers is needed for SDRAM arbitration priority information and to control IPAs FIFO buffering. This influences system performance and bandwidth balance only.

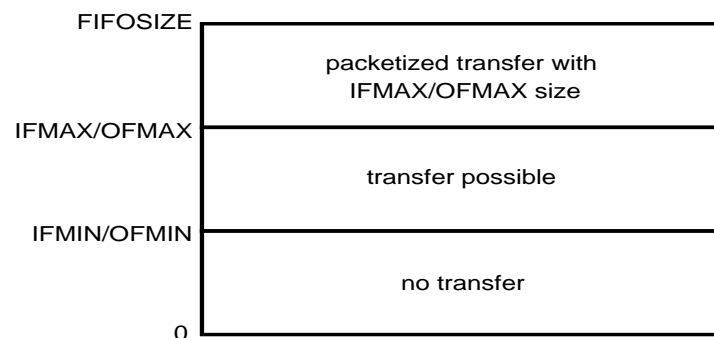
**Table 2-1:** DIPA performance adjustment registers

Register	Address	Bits	Function	Initial
DIPACTRL_PDPA	0x4200	[18:16]	DPA arbitration priority [0...7]	0x0
DIPACTRL_PIPA	0x4200	[2:0]	IPA arbitration priority [0...7]	0x0
DIPAIF_IFMIN	0x4204	[6:0]	Minimum block size for data transfer to the video RAM ( $\geq 2$ )	0x02
DIPAIF_IFMAX	0x4204	[22:16]	Maximum block size for data transfer to the video RAM ( $\geq$ IPAIF_MIN)	0x02
DIPAOF_OFMIN	0x4208	[6:0]	Minimum block size for data transfer from the video RAM ( $\geq 1$ )	0x01
DIPAOF_OFMAX	0x4208	[22:16]	Maximum block size for data transfer to the video RAM ( $\geq$ DIPAOF_MIN)	0x01

Initial values (reset-state) for IPA configuration are not default values and some of them are restricted for operation. An configuration of block sizes is required before IPA operation is possible.

### 2.2 Recommended Settings

Figure 2-1 shows the relationship between IPA settings for input or output FIFO and the current FIFO load. If the load is smaller than IFMIN or OFMIN no data transfer is performed. In the load range between IFMIN/OFFMIN and IFMAX/OFFMAX a single transfer with at least IFMIN/OFFMIN data words is triggered. Above IFMAX/OFFMAX FIFO load the transfer is packetized into packages of IFMAX/OFFMAX size.



**Figure 2-1:** DIPA settings for input and output FIFO

The configuration with respect to system behaviour depends on the bandwidth reserve for video RAM access (transfer rate and core clock dependent). Normally GPU and VIC have highest access arbitration priority due to the fact that these are real-time applications and their bandwidth violations would lead to data loss (drop of display or video data may be the result of wrong configuration).

DPA with its single word accesses has normally not that important influence on system performance. The granted access duration is very short. If the bandwidth requirements are not close to the edge, highest priority should be not problematic if assigned (i.e. DPA=7, GPU=6, VIC=5). If considered that bandwidth is very critical for GPU or VIC a setting of GPU =7, VIC=6 and DPA=5 should be assigned.

IPA can lock the video RAM interface relative long time, even if a large maximum block size is specified. Therefore a very low priority should be assigned to IPA, at least lower than the real-time devices GPU and VIC. A low value did not necessarily reduce IPA performance. The data stream is buffered through the input and output FIFOs. Only reaction time increases slightly for low priority requests. If bandwidth violations occur for GPU or VIC at IPA usage at lowest priority, the maximum block transfer size should be reduced. It may be possible that GPU FIFO under-run or VIC FIFO overflow occurs while IPA locks the RAM access with large packet transfers. Just at the usage of high resolution displays with large video bandwidth requirement this aspect should be considered.

Now some considerations for minimum block size settings. In general the minimum values must be less or equal to the maximum settings. These values are for controlling efficiency of IPA requests to the SDC. If the amount of data in the FIFOs is lower the given value, no action is initiated. The IPA device waits for a required, worthwhile packet size. The greater the packet size, the higher the throughput, but additional higher risk of GPU or VIC interruption due to video memory locking during longer transfer times.

## 2.3 Related Settings and Informations

From application point of view the setup of appropriate control information to the DIPA configuration registers is not sufficient for accessing video memory in physical address format. Additional settings are required for a complete configuration. Detailed information are in ULB and SDC documentation.

ULB handles the mapping of the video RAM address used for DPA (memory mapped). Other commands use the In- and Output FIFOs too. Additional command processing is controlled by ULB.

SDC did the logical to physical address conversion. If picture data should be accessed, SDC calculates based on Layer Description Record (LDR) information the physical video memory address. Layer start offsets and bit size of pixel data influences logical to physical address mapping. This knowledge is necessary to locate a given picture coordinate {Layer, X, Y} at its dedicated physical memory position.

If IPA is used in DMA mode both control mechanisms for DMA buffer sizes (DMA demand mode, see ULB description) and for IPA FIFO buffering should be considered to stay not in conflict with each other. Otherwise this could lead to deadlock situations in data flow.

---

## ***B-7 Video Interface Controller (VIC)***



# 1 Introduction

## 1.1 Video Interface Controller functions and features

The purpose of the Video Interface Controller (VIC) is to receive video data from an external video controller. The VIC operates only in synchronous mode, which means that the VIC clock should be provided by external video pixel decoder. Asynchronous modes where no clock is needed are not supported.

An external video controller converts an analog video signal into a digital component stream. Some of the controllers have additional features like video resizing, picture quality control (contrast, brightness etc.), anti-aliasing-filtering and color format conversion (YUV to RGB).

VIC reads digital video signals from video controller output, assorts data and writes them into graphic controller's Video memory (SDRAM) area reserved for external video data.

Due to a new bus shuffler<sup>1</sup>, byteswap and clock invert functions the video interface gets a wider range of flexibility.

The following points list all VIC features briefly:

- data port with 8 bit (Port A only) or 16 bit (Port A and Port B)
- different color formats: RGB555, RGB565, RGB888, YUV444, YUV655<sup>2</sup>, YUV555<sup>2</sup>, YUV422
- single clock and double clock mode (Port A only)
- different input timing (Videoscaler-Mode and CCIR) with a wide range of variations due to bus shuffler (Jasmine only)
- programmable polarity for control signals
- alpha key mapping with programmable color
- windowing function (Jasmine only)
- progressive and interleaced input
- skip function to reduce data rate
- frame management unit to synchronize video input and displayed video output
- programmable layer addresses
- picture start flag in flag register (Jasmine only)

## 1.2 Video data handling

VIC reads real-time video data from a special display controller interface and writes them into Video Memory (SDRAM) with help of SDRAM controller (SDC).

As video target a normal layer can be used. Within the display controller no special marking for video layers is necessary, it is treated as all other layer which contain drawings and bitmaps.

In order to synchronize input video frame rate with display frame rate VIC contains a frame synchronization controller that works together with GPU. For synchronization a two layer and a three layer mode is available. In GPU only one of the used layers needs to be set up. Which of these two or three layers is actually used is decided by frame synchronization controller in real time.

In two layer synchronization mode one write and one read layer is used for synchronization and the frame controller compares vertical read/write positions for actual layer.

In three layer synchronization mode only a complete written layer is displayed by GPU. It depends on the frame rate difference between video input and display output which layer is displayed how often. If GPU is

---

1. This shuffler is only available for Jasmine. Lavender does not contain this feature.

2. Jasmine only; see chapter 2.2

much faster than VIC it can happen that one layer is displayed more than one time. On the other hand, if VIC is much faster than GPU the display of some layers may be skipped. In any case no artefacts within one layer are visible.

To decrease memory size needed for video input all needed layers for synchronization (two or three) may be set to one layer number. In this case VIC and GPU uses only one layer. Of course no frame rate synchronization is done and artefacts may be visible on display within video.

As an additional feature VIC supports a so called 'alpha channel'. Via a special pin (VSC\_ALPHA) an external video scaler signals the VIC not to take the pixel color for the current coordinate but a special color previously defined in a special register (VICALPHA). This color is written to Video memory instead of video color.

Together with other features of display controller such as transparent color and blinking inside GPU various effects can be achieved.

Jasmine contains a window function where a rectangular area (window) can be defined which treats as clipping rectangle for video input data. Only data within this window are written to Video Memory.

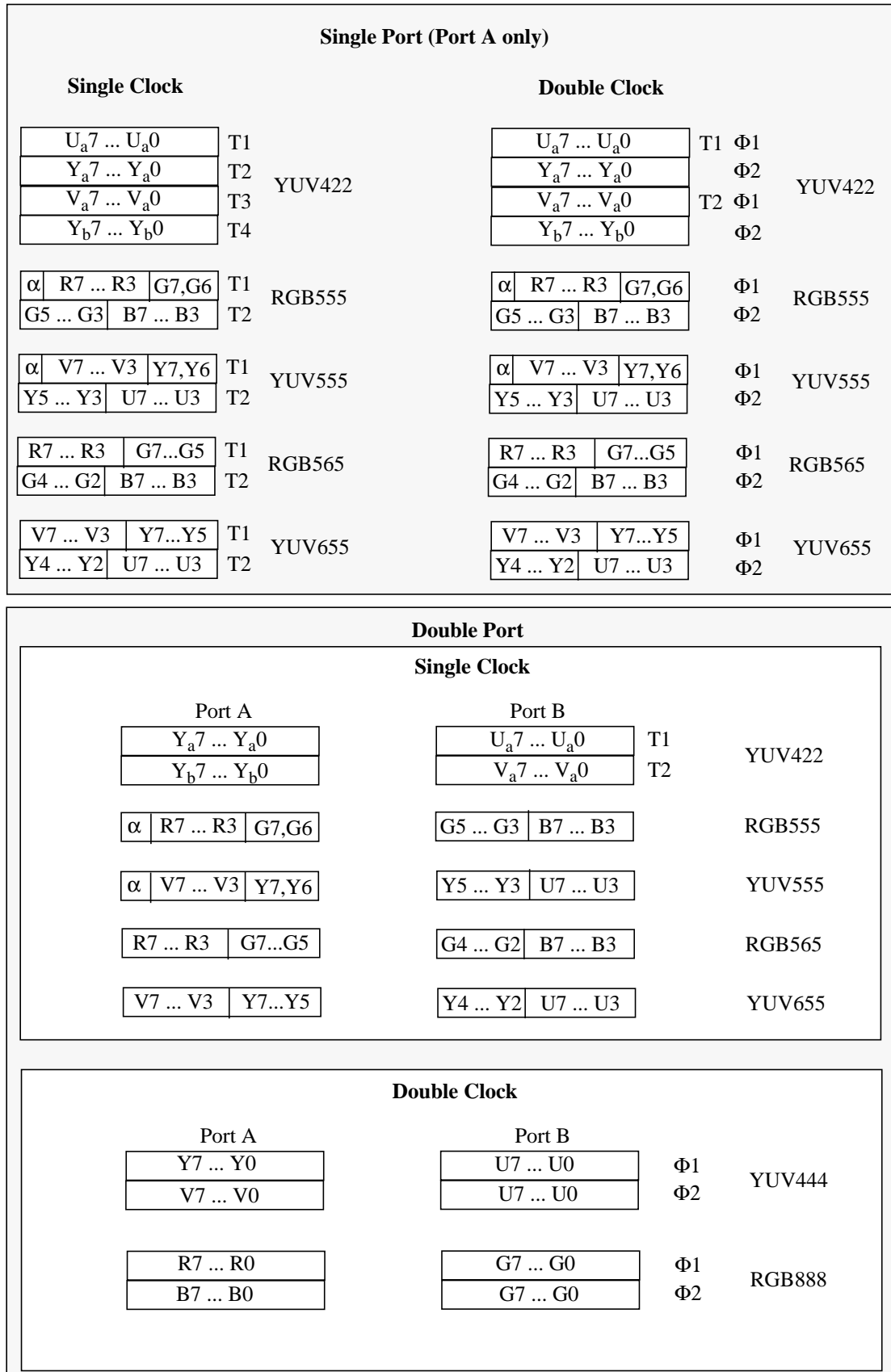
This function can be used to cut out a part of the video frame in order to save memory. It is also used to remove blanking information in special input timing modes (see chapter 2.3 for details).

## 2 VIC Description

### 2.1 Data Input Formats

Video Interface Controller supports different color formats displayed in figure 2-1 but does not convert color format into each other. Color depth for video target layers has to match color depth of incoming video data. Color conversion and mapping to display color depth will be done in output stage within GPU.

The Video Interface Controller has two data ports (port A and B), each 8 bit wide. In single port mode only port A will be used. Double port mode means that data will be received on both ports (A and B, 16 bit). Additionally VIC supports two different clock modes. Single clock mode uses only one clock phase (edge) while double clock mode uses both phases. In double clock mode, data on port A changes with every edge of clock. Port B (if enabled) and control signal only works in single clock mode. Different color modes depending on clock mode are shown in figure 2-1.

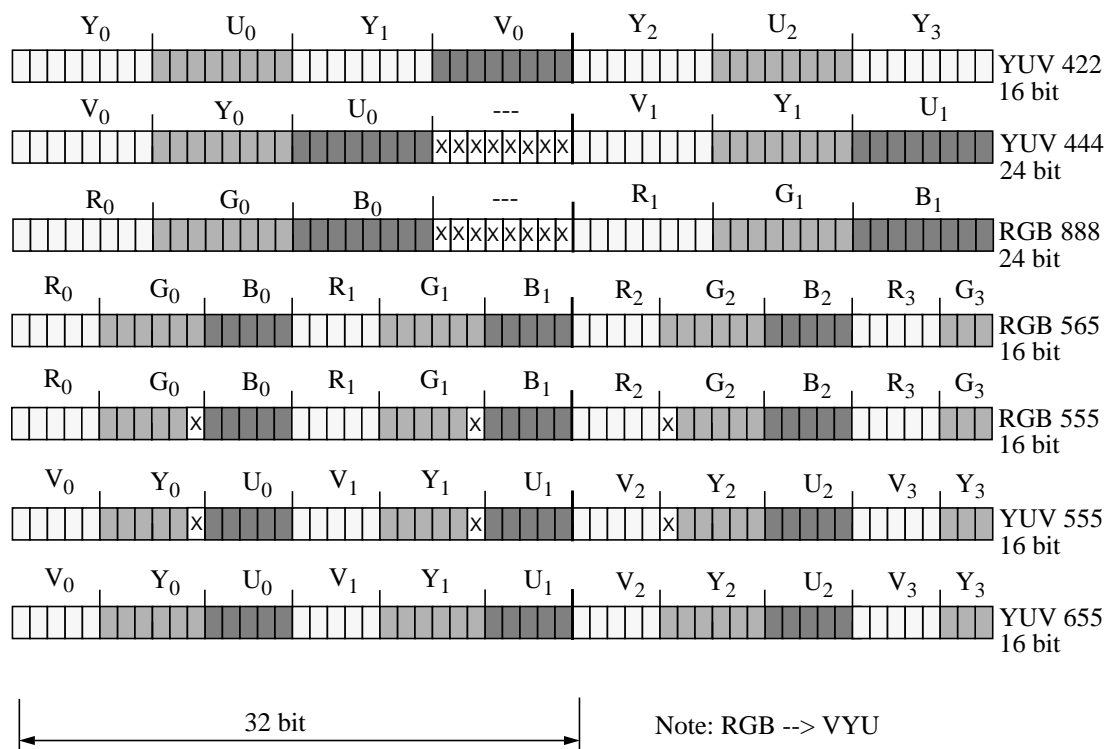
**Figure 2-1:** Color assignment in single - and double port mode



## 2.2 Data format in Video RAM (SDRAM)

Figure 2-2 shows the physical placement of all supported video color formats within a layer. These formats are equal to those of Graphic Processing Unit (GPU) and Pixel Processor (PP) so that video input PP drawings can be mixed within one layer. Of course a drawing will be overwritten with the input video picture but nonoverlapping merging of drawing and video is possible. Be also aware of layer synchronization between VIC and GPU; either only one layer is taken for synchronization or the drawing is copied on all two or three layers with help of MemCP command. If only one of two/three layers contains the drawing you will see a flicker on display output.

VIC performs a format mapping from video input format according to figure 2-1 to the Lavender/Jasmine internal format according to figure 2-2 for a given color depth. Note that some color formats are not supported by Lavender. Table 2-1 gives an overview on available formats for both display controllers.



**Figure 2-2:** Different color formats in video RAM

**Table 2-1:** Supported Video color formats for Lavender and Jasmine

Format	Lavender	Jasmine	Comment
YUV422	grayscale	yes	Lavender does not contain a YUV to RGB conversion therefore this format is converted to grayscale with 256 steps.
YUV444	grayscale	yes	Lavender does not contain a YUV to RGB conversion therefore this format is converted to grayscale with 256 steps.
RGB888	yes	yes	
RGB565	yes	yes	

**Table 2-1:** Supported Video color formats for Lavender and Jasmine

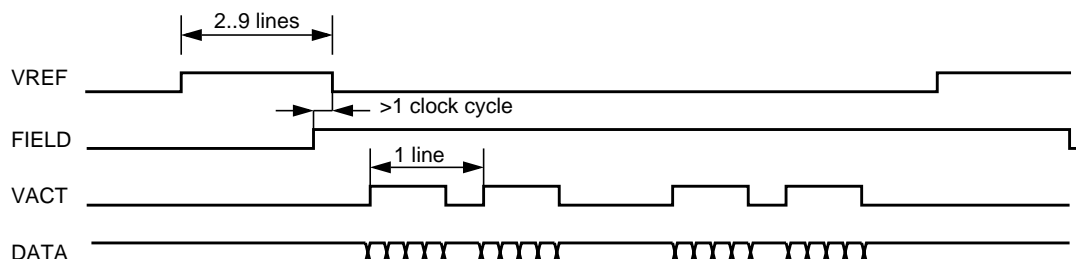
Format	Lavender	Jasmine	Comment
RGB555	yes	yes	
YUV555	-	yes	Lavender does not contain a YUV to RGB conversation (see GPU description)
YUV655	-	yes	Lavender does not contain a YUV to RGB conversation (see GPU description)

## 2.3 Data Input Timing

The VIC component supports three different input timing modes which can be selected using register VICVISYN\_SEL. The preferred and default mode is denoted "Videoscaler-Mode" in the following.

### 2.3.1 Videoscaler-Mode

Videoscaler-Mode has been implemented for VPX Video Pixel Decoder family (Micronas Intermetall), which provides up to 16 data bits (DATA; Pin: VSC\_D[15:0]), a video clock (CLKV; Pin: VSC\_CLKV), a vertical synchronization signal (VREF; Pin: VSC\_VREF), a field identification signal (FIELD; Pin: VSC\_IDENT), a horizontal video active signal (VACT; Pin: VSC\_VACT) and additional an optional alpha key signal (ALPHA; Pin: VSC\_ALPHA). Other video decoders are supported if they provide a similar interface and timing. See timing diagram (figure 2-3) for detailed information.

**Figure 2-3:** Videoscaler timing (general)

VREF pulse indicates the start of a new field. FIELD changes should happen at least one clock cycle before negative edge of VREF (see figure 2-3).

VACT marks an active video line. Due to downscale function of Videoscaler inactive lines are possible. Positive edge of VACT indicates the start of an active line, negative edge denotes the end of line. During VACT is active, VIC samples data received on data ports and writes them into Video Memory (SDRAM). Polarities of control signals are adjustable, see register VICPCTRL.

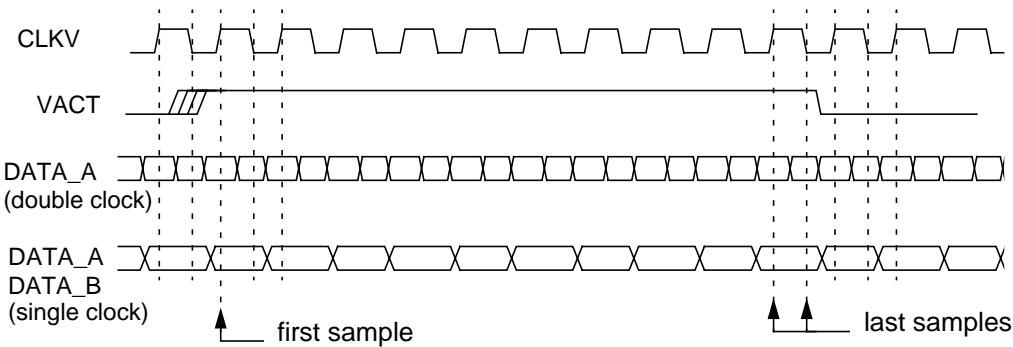


Figure 2-4: Videoscaler timing (detailed)

2.3.2 CCIR-Mode

In CCIR-Mode no explicit control signals are used. Control informations are directly merged into data stream (data port A). VIC component extracts control information from data and maps it to internal signals similar to Videoscaler-Mode.

The CCIR-Mode is only available for MB87P2020-A (Jasmine).

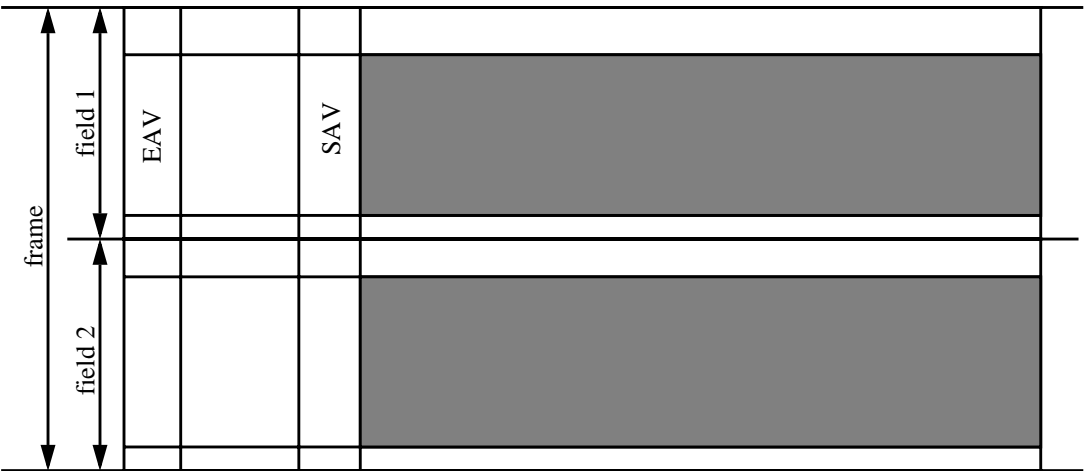


Figure 2-5: Definitions for CCIR mode

There are two timing reference codes, one at the beginning of each video data block (SAV: start of active video) and one at the end of video data block (EAV: end of active video). Each code consists of a four-byte-sequence. The first three bytes are fixed preamble. The fourth word carries information about field information (F), field blanking (V) and line blanking (H).

Table 2-2: Timing reference code sequence

Bit number	7	6	5	4	3	2	1	0
FIRST	1	1	1	1	1	1	1	1
SECOND	0	0	0	0	0	0	0	0
THIRD	0	0	0	0	0	0	0	0
FOURTH	1	F	V	H	P3	P2	P1	P0

F-bit:

- '0' during field number 1
- '1' during field number 2

V-bit:

- '1' during field blanking
- '0' else

H-bit:

- '0' in sequence SAV
- '1' in sequence EAV

Bits [3:0] (P3..P0) are protection bits. The value of those bits depends on bits F, V and H.

**Table 2-3:** Definition of SAV, EAV and protection bits

fixed	SAV/EAV			protection bits			
	F	V	H	P3	P2	P1	P0
1	0	0	0	0	0	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	0	1	1
1	0	1	1	0	1	1	0
1	1	0	0	0	1	1	1
1	1	0	1	1	0	1	0
1	1	1	0	1	1	0	0
1	1	1	1	0	0	0	1

The VIC implements an error detection and correction based on protection bits. It is possible to detect two-bit-errors and to correct one-bit-errors.

In case of a two-bit-error the controller takes F- and V-bit directly from current TRC (without error correction) and inverts H-bit received from previous TRC.

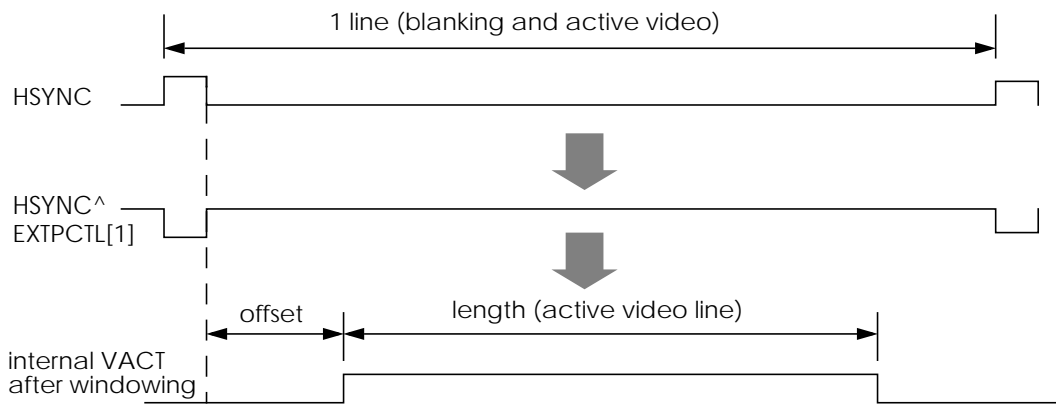
### 2.3.3 External-Timing-mode

The External-Timing-Mode is only available for MB87P2020-A (Jasmine).

External-Timing-mode is together with VIC windowing function (Register: VICLIMEN) a generalized Videoscaler-Mode. It is able to receive a horizontal synchronization signal (HSYNC) and a vertical synchronization signal (VSYNC) which is provided by many video devices. Note that the data has to be one of the formats described in chapter 2.1.

Polarities can be controlled by register EXTPCTRL. Control signal (HSYNC, [pin VSC\_VACT], VSYNC, [pin VSC\_VREF] and PARITY [pin VSC\_IDENT]) will be taken directly from input.

The HSYNC-pulse tags the begin of a new line and the VSYNC-pulse the begin of a new field/frame. Active video needs not to be indicated additionally. The start and length of the visible window should be controlled by windowing function for both dimensions.



**Figure 2-6:** External-Timing  
Related registers: VICLIMEN, VICLIMH, VICLIMV, EXTPCTRL



## 3 VIC settings

### 3.1 Register list

**Table 3-1:** VIC register description

Symbol	Address.	Description/Definition
VICSTART	h4000	input layer start coordinates X[29:16] : x coordinate ( <b>default: 0</b> ) Y[13:0] : y coordinate ( <b>default: 0</b> )
VICALPHA	h4004	Color to be mapped when alpha pin is active and alpha_mode=1 (VICCTRL[6]). Bit occupancy depending on color mode (LSB aligned). COL[23:0]: <ul style="list-style-type: none"> <li>— [7:0] : value for blue (U)</li> <li>— [15:8] : value for green (Y)</li> <li>— [23:16] : value for red (V)</li> </ul> default: 24'b0
VICCTRL	h4008	MODE[3:0]: color_mode <ul style="list-style-type: none"> <li>— 0100: RGB555</li> <li>— 0101: RGB565</li> <li>— 0110: RGB888</li> <li>— 0111: YUV422</li> <li>— 1000: YUV444</li> <li>— 1110: YUV555</li> <li>— 1111: YUV655</li> </ul> CLOCK[4]: clock_mode <ul style="list-style-type: none"> <li>— 0: single clock</li> <li>— 1: double clock</li> </ul> PORT[5]: port_mode <ul style="list-style-type: none"> <li>— 0: single port</li> <li>— 1: double port</li> </ul> ALLEN[6]: alpha_mode <ul style="list-style-type: none"> <li>— 0: don't use alpha information</li> <li>— 1: use alpha</li> </ul> BSWAP[7]: byte_swap <ul style="list-style-type: none"> <li>— 0: {A[15:8],B[7:0]}</li> <li>— 1: {B[15:8],A[7:0]}</li> </ul>

**Table 3-1:** VIC register description

Symbol	Address.	Description/Definition
VICCTRL	h400C	<p>FST[3:0]: primary_layer_address</p> <p>SEC[7:4]: secondary_layer_address</p> <p>TRD[11:8]: third_layer_address</p> <p>ODDEN[16]: enable_odd_fields</p> <ul style="list-style-type: none"> <li>— 0: disabled</li> <li>— 1: enabled</li> </ul> <p>EVENEN[17]: enable_even_fields</p> <ul style="list-style-type: none"> <li>— 0: disabled</li> <li>— 1: enabled</li> </ul> <p>VICEN[18]: vic_enable</p> <ul style="list-style-type: none"> <li>— 0: disabled</li> <li>— 1: enabled</li> </ul> <p>SKIP[20]: skip_fields_enable</p> <ul style="list-style-type: none"> <li>— 0: use every field</li> <li>— 1: skip every 2nd field of each type</li> </ul> <p>FRAME[21]: frame/field</p> <ul style="list-style-type: none"> <li>— 0: field mode (store one field in each layer)</li> <li>— 1: frame mode (interlocked storage of two fields in each layer)</li> </ul> <p>ODDFST[22]: odd_field_first</p> <ul style="list-style-type: none"> <li>— 0: even field is top field</li> <li>— 1: odd field is top field</li> </ul>
VICPCTRL	h4010	<p>Polarity control for VPX video synchronization signals. Those settings only effects if "VPX video source" is selected (VICVISYN[1:0] = 2'b00)</p> <p>VREF[0]: vref polarity</p> <ul style="list-style-type: none"> <li>— 0: high active</li> <li>— 1: low active</li> </ul> <p>VACT[1]: vact polarity</p> <ul style="list-style-type: none"> <li>— 0: high active</li> <li>— 1: low active</li> </ul> <p>FIELD[2]: field polarity</p> <ul style="list-style-type: none"> <li>— 0: polarity unchanged</li> <li>— 1: invert polarity</li> </ul> <p>ALPHA[3]: alpha polarity</p> <ul style="list-style-type: none"> <li>— 0: high active</li> <li>— 1: low active</li> </ul>



**Table 3-1:** VIC register description

Symbol	Address.	Description/Definition
VICFSYNC	h4014	SWL[14:0]: Switch Level (2-layer mode only) SYNC[15]: Layer Sync Mode — 0: 2-layer mode — 1: 3-layer mode REL[16]: Frame Rate Relation (2-layer mode only) — 0: GPU is faster than VIC — 1: VIC is faster than GPU
SDRAM	h401C	LP[2:0]: low priority <b>(default: 3'b010)</b> HP[6:4]: high priority <b>(default: 3'b110)</b>
VICBSTA	h4020	LOAD[6:0]: fifo load  REQ[8:7]: SDC request status — b00: IDLE — b01: REQ1 — b10: WAIT1  CLR[9]: clear fifo  ADD[12:10]: address register load — bx00: empty — bx01: 1 burst — bx10: 2 burst — bx11: 3 burst — bx00: 4 burst (full) — b0xx : NORM — b1xx : FULL, but no error (error flag will be set with next new address)  FSM[15:13]: fifo_fsm state — b000 : NORM — b001 : WAIT — b101 : WAIT1 — b010 : ERROR — b111 : RESET — b110 : ERROR1 — b011 : RESET1  AERR[16] : address error  FF[17] : fifo full  FE[18] : fifo empty

**Table 3-1:** VIC register description

Symbol	Address.	Description/Definition
VICRLAY	h4024	<p>AIVL[3:0]: actual written video layer</p> <p>LIVL[11:8]: anteriorly written video layer</p> <p>AOVL[23:16]: actual output video layer</p>
VICVISYN	h4028	<p>SEL[1:0] : video source select</p> <ul style="list-style-type: none"> <li>— b00 : VPX</li> <li>— b01 : CCIR - TRC</li> <li>— b10 : CCIR - ext. sync</li> </ul> <p>START[8] : selector for picture start flag (to ULB)</p> <ul style="list-style-type: none"> <li>— 0 : VIC - start writing picture</li> <li>— 1 : GPU - start reading picture</li> </ul> <p>SHUFF[20:16] : Data Bus Shuffler  <b>default: 5'b00000</b>  For details see chapter 3.2.</p> <p>DEL[25:24] : data delay</p> <ul style="list-style-type: none"> <li>— b00: delay data 0 cycle respective VSC_VACT</li> <li>— b01: delay data -1 cycles respective VSC_VACT</li> <li>— b10: delay data -2 cycles respective VSC_VACT</li> <li>— b11: delay data -3 cycles respective VSC_VACT</li> </ul>
VICLIMEN	h402C	<p>LIMENA[0] : limit size enable</p> <ul style="list-style-type: none"> <li>— 0: limit size disabled</li> <li>— 1: limit size enabled</li> </ul>
VICLIMH	h4030	<p>HOFF[10:0]: horizontal offset</p> <p>HEN[26:16]: horizontal window length (including offset!)</p> <p>Only effects if VICLIMEN[0] = 1!</p>
VICLIMV	h4034	<p>VOFF[10:0]: vertical offset</p> <p>VEN[26:16]: vertical window length (including offset!)</p> <p>Only effects if VICLIMEN[0] = 1!</p>

**Table 3-1:** VIC register description

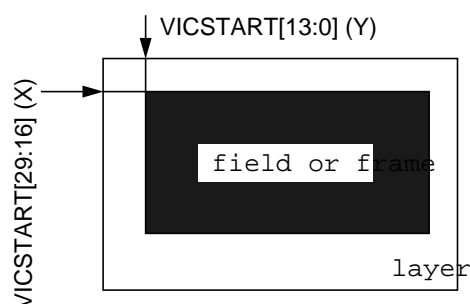
Symbol	Address.	Description/Definition
EXTPCTL	h4038	<p>Polarity control for external video synchronization signals. Those settings only effects if "external video source" is selected (VICVISYN[1:0] = 0b10)</p> <p>VREF[0]: vsync</p> <ul style="list-style-type: none"> <li>— 0: high active</li> <li>— 1: low active</li> </ul> <p>HREF[1]: hsync</p> <ul style="list-style-type: none"> <li>— 0: high active</li> <li>— 1: low active</li> </ul> <p>PARITY[2]: parity</p> <ul style="list-style-type: none"> <li>— 0: polarity unchanged</li> <li>— 1: invert polarity</li> </ul> <p>ALPHA[3]: alpha</p> <ul style="list-style-type: none"> <li>— 0: high active</li> <li>— 1: low active</li> </ul>
CLKPDR	hFC04	<p>VII[12]: video clock invert</p> <ul style="list-style-type: none"> <li>— 0: don't invert clock</li> <li>— 1: invert clock</li> </ul>

## 3.2 Register Description

VICSTART (4000H):

GDC stores video fields or frames in different memory areas called layers. The VICSTART register defines layer start coordinates in x- and y-direction. Note that coordinates are equal for all layers used for video input (one, two or three layer).

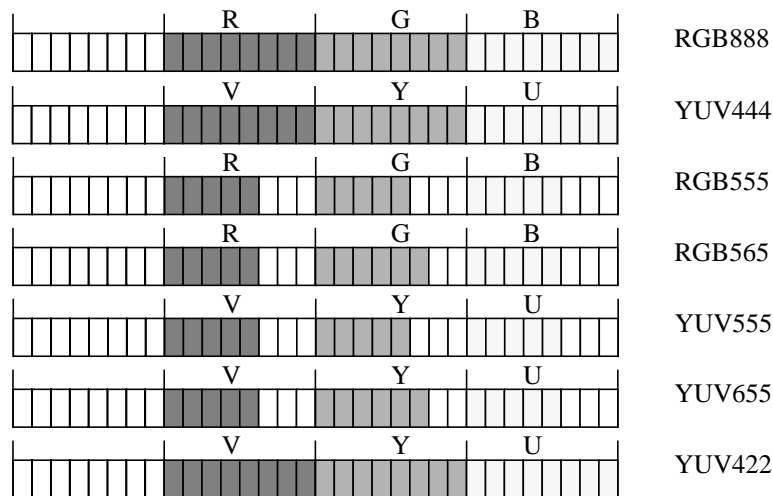
Related registers: VICFCTRL[11:8], VICFCTRL[7:4], VICFCTRL[3:0]

**Figure 3-1:** Start coordinate definitions

VICALPHA (4004H):

The VICALPHA register contains the value of color which will be mapped if alpha key signal is active and alpha\_mode is enabled (see register VICCTRL[6]). Note that bit occupancy depends on color mode (see figure 3-2).

Related register: VICCTRL[6]



**Figure 3-2:** Possible Color occupancy in register VICALPHA

#### VICCTRL (4008H):

The VICCTRL register contains settings for color mode, clock mode, port mode and byte swap. Those settings should be done according to settings of external video decoder. Please use only suggestive combinations of color-, clock- and port-mode.

#### VICFCTRL (400CH):

*bit [3:0]* (FST), *[7:4]* (SEC), *[11:8]* (TRD):

GDC's memory space is divided in up to 16 areas (layer). Each of them has a logical layer address and can contain video data. VIC implies a frame synchronization controller which manages two or three layers needed for frame synchronization (see register VICFSYNC). Layer addresses for VIC could be defined in register VICFCTRL. Note that same addresses are allowed, but field/frame synchronization won't work correctly.

Related registers: VICSTART, VICFSYNC

#### *bit 16 and 17* (EVENEN and ODDEN):

If those flags are set to zero, every field of the specified type (odd/even) will be skipped. So EVENEN (enable even fields) and ODDEN (enable odd fields) submit selection of field type. It is recommended to enable only one field type to suppress field jumping during displaying video layers. Function is to be seen in context with VICFCTRL[20] (SKIP). Those flags have no relevance if VICFCTRL[21] (FRAME) is set.

Related registers: VICFCTRL[18], VICFCTRL[22:20]

#### *bit 18* (VICEN):

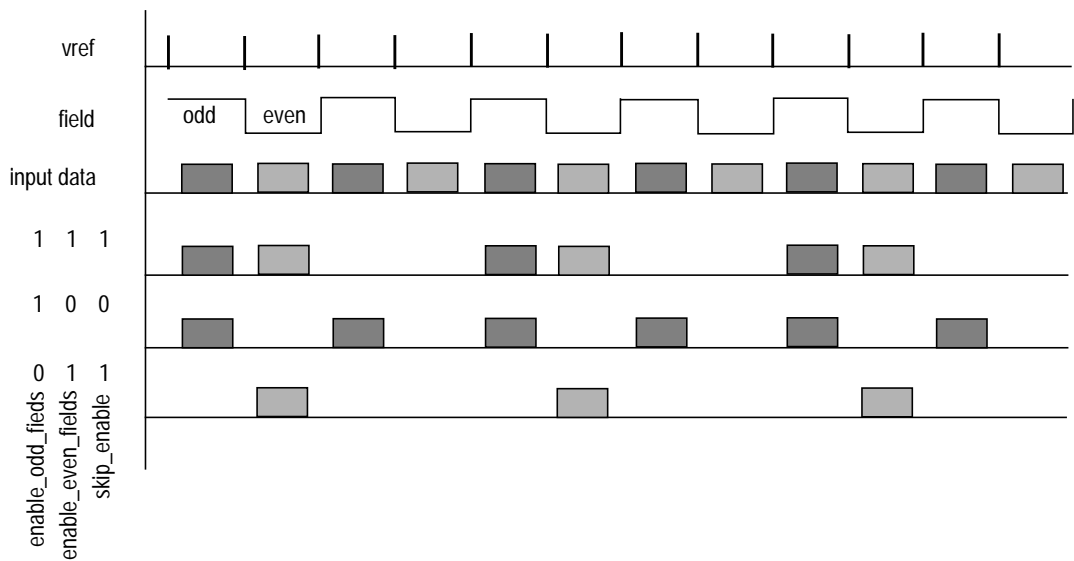
The VICEN (VIC enable) switches video interface unit on (1) or off (0), („main switch“). It does also enable or disable synchronization of video in- and output.

Related registers: VICFCTRL[17:16], VICFCTRL[22:20]

#### *bit 20* (SKIP):

This flag controls the skip function of VIC. If SKIP is set to 1, every second field of each field type is skipped. Skip enable function is to be seen in closed connection with VICFCTRL[16] (EVENEN) and VICFCTRL[17] (ODDEN). If both of them and SKIP are set to one, every second frame is skipped. So it is possible to reduce data rate between VIC and SDRAM to a half of the output data rate from external video pixel decoder. A quarter of input data rate can be achieved by enabling only one field and skip. An example is shown in the following figure (figure 3-3).

Related registers: VICFCTRL[18:16], VICFCTRL[22:21]

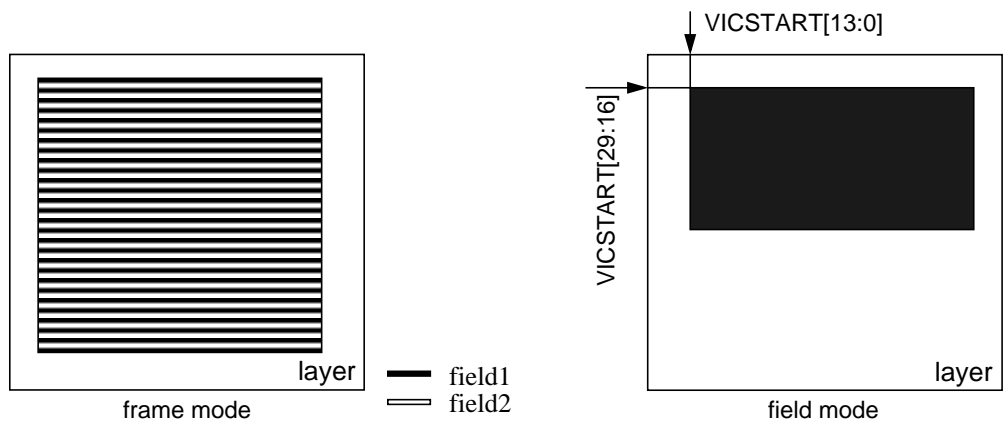


**Figure 3-3:** Reducing data rate by skipping fields (examples)

*bit 21* (FRAME):

The FRAME flag controls the storage format of input video in video RAM. If FRAME is set to 1, a complete frame (even and odd field) will be saved interlocked in each video layer. In case of FRAME=0 every layer contains one field and interleaved to progressive format conversion can be done in output process (line doubling).

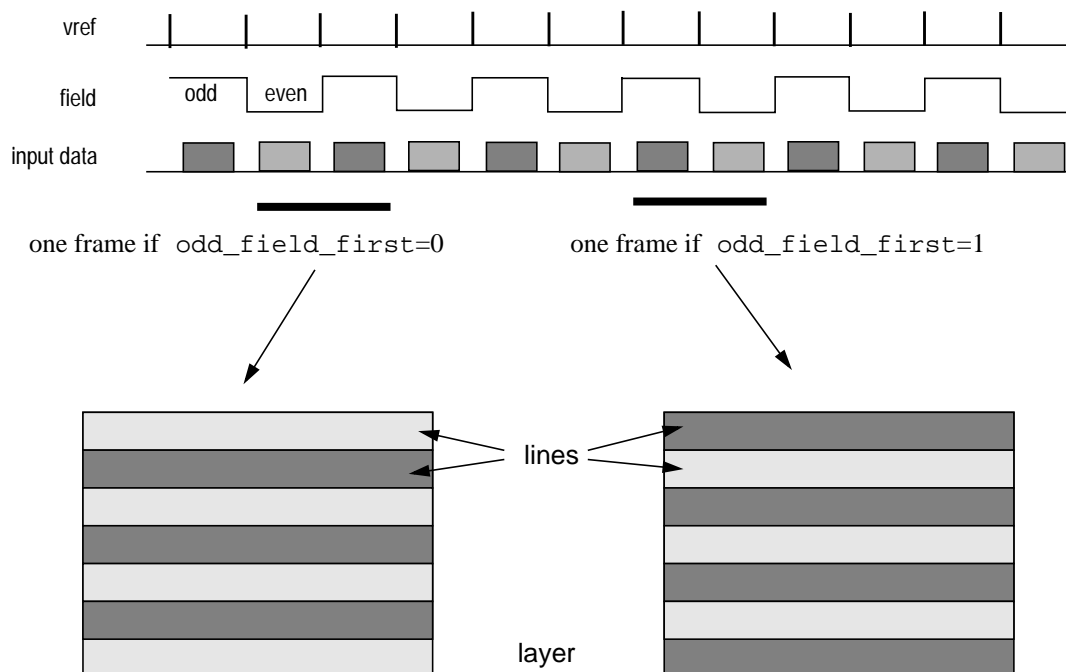
Related registers: VICFCTRL[18:16], VICFCTRL[20], VICFCTRL[22], VICPCTRL[2] or EXTPCTRL[2] in association with VICVISYN[1:0]



**Figure 3-4:** Frame mode vs. field mode

*bit 22:*

ODDFST (odd field first) determines the position of the fields in a frame related to time. The following figure illustrates the issue. This flag has no relevance if VICFCTRL[21] is set to 0.



**Figure 3-5:** odd\_field\_first flag: mode of action

VICPCTRL (4010H):

The flags in this word set the polarity of input control signals (vact, vref, alpha, field). Note that those bits only influence the behaviour of VIC if Videoscaler-Mode is enabled as video source (VICVISYN[1:0]=0b00).

Related register: VICVISYN[1:0]

VICFSYNC (4014H):

Synchronization of video input and video output is frame/field based. Frame synchronization unit is to ensure correct order of pictures and avoids overtaking of read and write pointer in the same active layer. Two modes are possible and can be selected by VICFSYNC[15] (SYNC).

*bit 15* (SYNC):

If Layer Sync Mode (SYNC) is set to 1, the 3 layer synchronization mode will be used. The used memory space is reduced by selecting three layer mode and setting all layer addresses (VICFCTRL[3:0], VICFCTRL[7:4], VICFCTRL[11:8]) to the same layer number. Then video input and output are not synchronized.

In two layer mode (SYNC=0) the synchronization will be performed by comparing vertical read/write position (GPU/VIC) in actual layer. There are two layers for video I/O (primary layer: VICFCTRL[3:0], secondary layer VICFCTRL[7:4]). The third layer address (VICFCTRL[11:8]) has no effect.

Related registers: VICFSYNC[16], VICSYN[14:0], VICFCTRL[3:0], VICFCTRL[7:4], VICFCTRL[11:8]

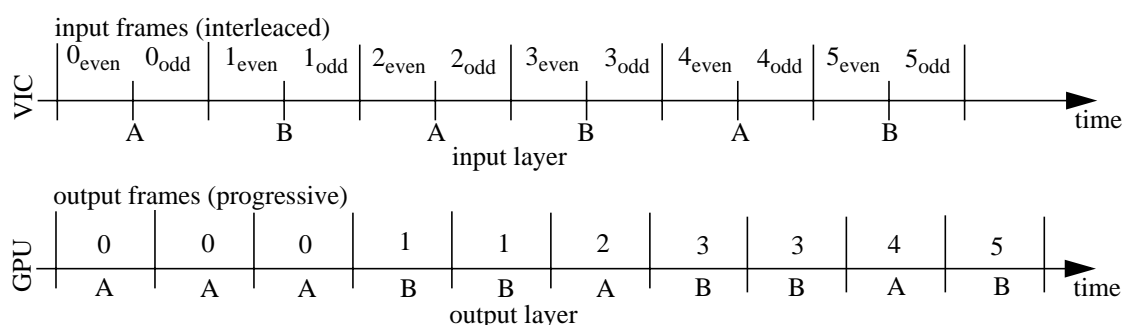
*bit 16* (REL):

This bit has relevance if VICFSYNC[15] is set to 0 (2-layer-mode).

The Frame Rate Relation (REL) flag indicates the relation between input (VIC) and output (GPU) frame rate. The faster unit toggles its layer each time a layer is fully processed. The other unit compares its line pointer with the switch level VICFSYN[14:0] (SWL) to decide, which layer using next. For example, if REL is set to 0, frame output rate is higher than frame input rate. The current active VIC layer toggles with

every new input frame. The next GPU layer is synchronized depending on current VIC layer and vertical position of write pointer (in this layer) which will be compared with SWL. If current vertical write position is above the programmable threshold SWL, the current VIC write layer will be allocated to GPU because an overtaking of read and write pointer in the same layer is excluded. Otherwise (SWL is less than VIC write pointer) the frame synchronization unit denotes the inactive VIC layer to be the next active GPU layer.

Related registers: VICFSYNC[15], VICSYN[14:0], VICFCTRL[3:0], VICFCTRL[7:4]



**Figure 3-6:** Example of frame synchronization: video output is faster than video input

*bits [14:0] (SWL):*

This bit has only effect if VICFSYNC[15] is set to 0 (2-layer-mode).

For explanation of SWL in context with VICFSYNC[16] (REL) see previous point. SWL will be compared with the vertical address. Note that this address (and SWL too) includes an offset (VICSTART[13:0]). If frame mode is enabled (VICFCTRL[21] = 1), SWL indicates the field (0: even, 1: odd) where SWL is located. In field mode (VICFCTRL[21] = 0) SWL[14] has no function (should be set to 0).

Related registers: VICFSYNC[16], VICSYN[15], VICFCTRL[3:0], VICFCTRL[7:4]

VICRLAY (4024H):

VICRLAY contains some layer information, which can be read from ULB (i.e. after a picture start was indicated). Three different informations will be given:

- VICRLAY[3:0] (AIVL) - this is the layer video data will be written in currently
- VICRLAY[11:8] (LIVL) - this is the previous written video layer
- VICRLAY[23:16] (AOVL) - this is the current displayed video layer

VICRLAY is a read only register.

Related registers: VICVISYN[8], VICFCTRL[11:8], VICFCTRL[7:4], VICFCTRL[3:0]

VICVISYN (4028H):

*bit [1:0] (SEL):*

Switches to select an input timing scheme. In Videoscaler-Mode (b'00) VIC interface uses explicit synchronization signals. Polarity of those signals can be selected by VICPCTRL. SEL=0b10 is a similar timing scheme as Videoscaler-Mode, polarity can be controlled by EXTPCTRL. CCIR-mode uses timing reference codes merged into the data stream. No explicit control signals are used.

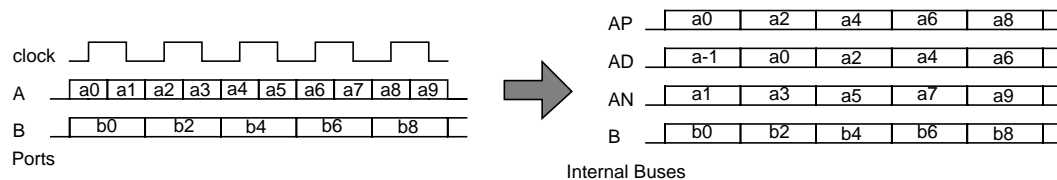
*bit[8] (START):*

Both, VIC and GPU, generate their own picture start signal. It indicates the begin of writing (VIC) or reading (GPU) a new picture into or from data RAM. One flag (FLNOM\_VICSYN) indicates the picture start to

ULB. START selects one of the two signals (VIC-start or GPU-start) and allocates it to FLNOM\_VICSYN. This is a useful feature to synchronize anything on picture start controlled by software.

Related register: VICRLAY

#### *bits[20:16]* Bus Shuffler (SHUF)

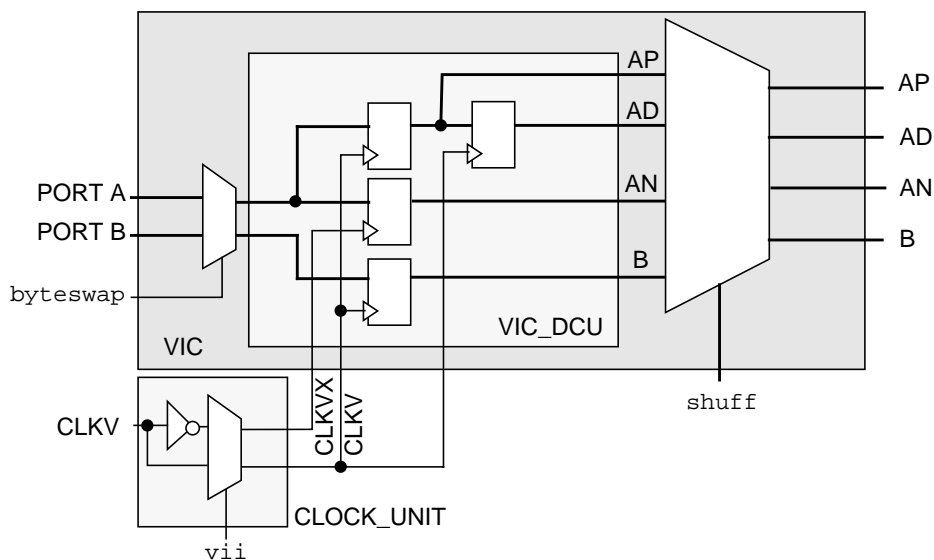


**Figure 3-7:** Definition of port - internal bus - assignment

VIC owns two data ports which are splitted into four internal busses (figure 3-7). To get more flexibility (i.e. connect another external video controller to VIC without changing the board) VIC includes various functions:

- clock invert function (CLKPDR[12])
- byte swap to exchange ports A and B
- bus shuffler to exchange internal busses (all combination are possible, see table 3-2)

A schematic overview about data path are given in figure 3-8. Note that contradictory settings can abrogate each other (i.e. setting SHUF=2 abrogates setting VICCTRL\_BSWAP=1).



**Figure 3-8:** Schematic overview about busshuffler function



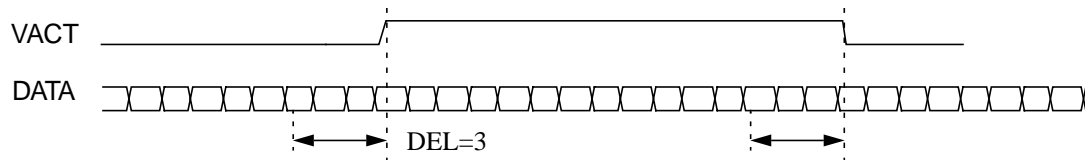
**Table 3-2:** VIC Bus Shuffler settings

<b>Bus Shuffler Value</b>	<b>BUS A, pos. edge (AP)</b>	<b>Bus A, delayed (AD)</b>	<b>Bus A, neg. edge (AN)</b>	<b>Bus B (B)</b>
<b>0 (default)</b>	<b>AP</b>	<b>AD</b>	<b>AN</b>	<b>B</b>
1	AD	AP	AN	B
2	B	AD	AN	AP
3	AP	B	AN	AD
4	AP	AD	B	AN
5	AN	AD	AP	B
6	AP	AN	AD	B
7	AP	B	AD	AN
8	AD	B	AN	AP
9	AD	B	AP	AN
10	AD	AN	AP	B
11	AD	AN	B	AP
12	AP	AN	B	AD
13	AD	AP	B	AN
14	B	AD	AP	AN
15	B	AN	AD	AP
16	B	AN	AP	AD
17	B	AP	AD	AN
18	B	AP	AN	AD
19	AN	AD	B	AP
20	AN	AP	AD	B
21	AN	AP	B	AD
22	AN	B	AP	AD
23	AN	B	AD	AP
others	AP	AD	AN	B

Related registers: VICCTRL[7], CLKPDR[12]

bits [25:24] (DEL):

DEL provides an option to shift data relative to pin VSC\_VACT with up to -3 cycles.



**Figure 3-9:** DEL definitions (example: DEL=3)

VICLIMEN (402CH):

The VIC-subunit VIC\_LIMIT is enabled or disabled by this switch. So a pan function can be realized. Another reason to enable VIC\_LIMIT could be the compensation of different input sync timing. Relating control registers are VICLIMH (horizontal offset and horizontal window length [video clocks per step]) and VICLIMV (vertical offset and vertical window length [lines per step]).

For horizontal parameters the context of input pixel per clock which depends on video colour depth and window offset/length should be considered. Since some colour formats transport less than one pixel per clock and the offset and length is set up in clock units it is necessary to calculate the setup value depending on colour depth. For each colour depth a factor 'f' for horizontal offset and length can be calculated as follows:

$$f = \frac{\text{Clocks}}{\text{Pixel}}$$

Registers VICLIMH\_HOFF and VICLIMH\_HEN have to be multiplied by 'f'. Note that for most colour formats 'f' can be set to one so that offset and length can be set directly in pixel dimensions. Table 3-3 lists all colour formats with their factor.

Additionally some colour formats need a special alignment for horizontal offset and length. Otherwise colour distortions may occur. Table 3-3 gives also an overview on necessary alignments.

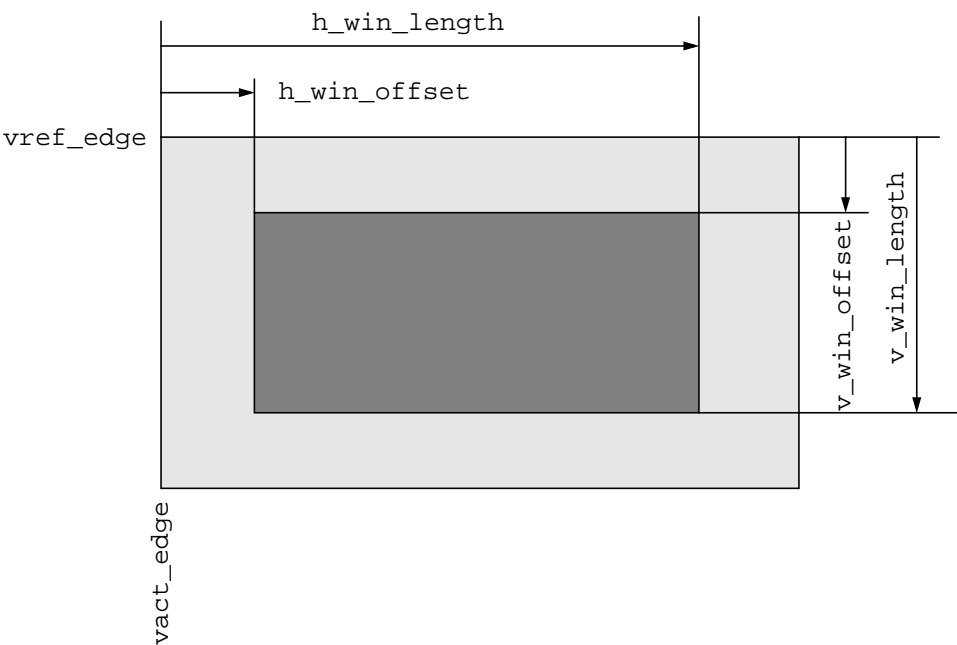
**Table 3-3:** Horizontal offset and length parameters for different colour depths

Format	Clock	Port	Factor 'f'	Alignment
YUV422	Single	Single	2	4
	Double		1	2
	Single	Double	1	2
RGB555	Single	Single	2	2
	Double		1	1
	Single	Double	1	1
YUV555	Single	Single	2	2
	Double		1	1
	Single	Double	1	1
RGB565	Single	Single	2	2
	Double		1	1
	Single	Double	1	1

**Table 3-3:** Horizontal offset and length parameters for different colour depths

Format	Clock	Port	Factor 'f'	Alignment
YUV655	Single	Single	2	2
	Double		1	1
	Single	Double	1	1
YUV444	Double	Double	1	1
RGB888	Double	Double	1	1

Related registers: VICLIMH, VICLIMV, VICVISYN[25:24]



**Figure 3-10:** window definitions

EXTPCTRL (4038H):

The flags in this word set the polarity of input control signals (hsync, vsync, alpha, parity). Note that those bits are only an effective if „ext. Sync“ is selected as video source (VICVISYN[1:0]=0b10).

Related registers: VICVISYN[1:0]



---

## ***B-8 Graphic Processing Unit (GPU)***



# 1 Functional Description

## 1.1 GPU Features

The GPU (Graphics Processing Unit) is a part of Fujitsu's display controllers MB87J2120 ('Lavender') and MB87P2020-A ('Jasmine'). In the following these two controllers are referred to as GDC (graphics display controller).

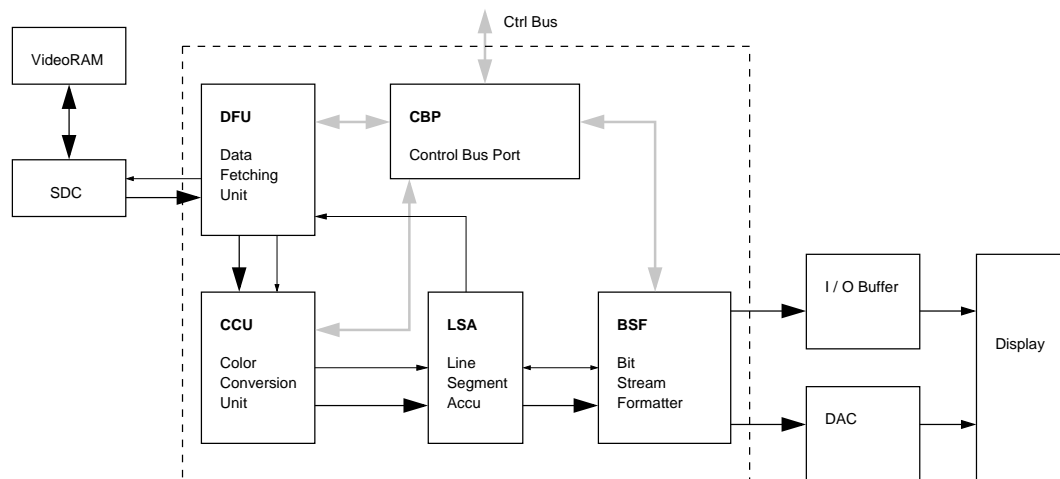
The GPU is responsible for producing visible images from pixel data stored in VideoRAM (SDRAM). Its features include:

- maximum flexibility by configuration registers for most functions
- wide range of supported display types and sizes (see chapter 7)
- twin display mode (digital and analog displays in parallel) (Jasmine only)
- up to four layers from a whole of 16 displayed simultaneously and overlapping in four planes
- size, color resolution and position on the display configurable on a per-layer basis
- blinking and transparency individual for each layer
- separate setting for display background color
- YUV to RGB matrix with configurable coefficients as well as a loadable gamma table included (Jasmine only)
- synchronization with the video interface controller (VIC) allowing frame rate conversion
- flexible generation of almost arbitrary sync signals
- individual settings for clamping values at digital and analog outputs (Lavender can only handle one clamping color)
- color key output
- internal and external pixel clock
- masking and division by 2 for internal pixel clock
- programmable interrupts
  - at specified display location
  - for insufficient memory bandwidth
- diagnostic read back
  - current display location
  - current blink state of layers
  - current input FIFO load

The image on display is produced in a three-step approach: fetching pixel data, transforming and merging the layers from their respective color space to one common, and finally formatting the bit stream appropriate for the actual display used.

## 1.2 GPU Overview

### 1.2.1 Top Level Structure



**Figure 1-1:** GPU top level structure. Thick black lines show the flow of pixel data, thin black lines the control flow. Thick gray lines represent register set data.

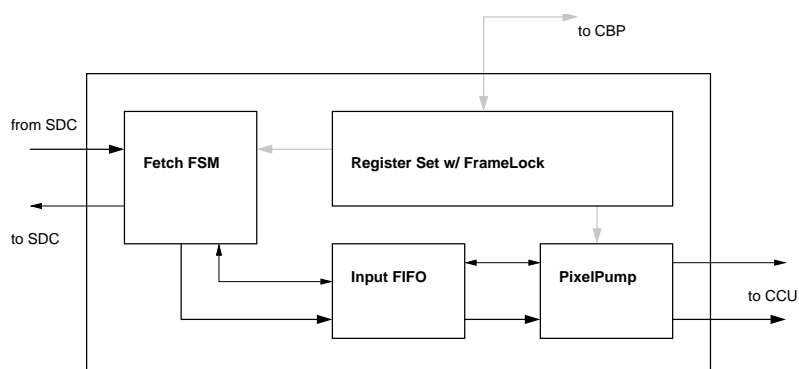
Figure 1-1 shows the GPU top level structure, which consists of the four main components DFU, CCU, LSA, and BSF. The CBP is actually only an auxiliary unit to couple the other units to the system-wide control bus. Control registers are assigned their respective units, so there is no GPU-global register set (although it appears as such for the programmer). The functions of each unit will be discussed briefly in the following sections.

### 1.2.2 DFU Function

The DFU (Data Fetching Unit) interacts with the SDC (SDRAM Controller) to transport pixel data from VideoRAM into the GPU pipeline. Since VideoRAM is a shared resource within GDC, the DFU is equipped with 4KBits of FIFO to balance data rate and optimize RAM usage by enabling burst accesses.

DFU structure is shown in figure 1-2. Pixels from layers displayed simultaneously are fetched sequentially according to their respective start points, display offsets, color resolutions, and vertical stacking. This is carried out by the Fetch FSM. The data obtained from SDC is put into the FIFO, from where it is read out later by the PixelPump and fed through the CCU into the LSA, where the vertical (Z-)order is actually realized. The PixelPump is also used to carry out the chrominance demultiplexing for YUV422 layers (cf. 3.7.1).

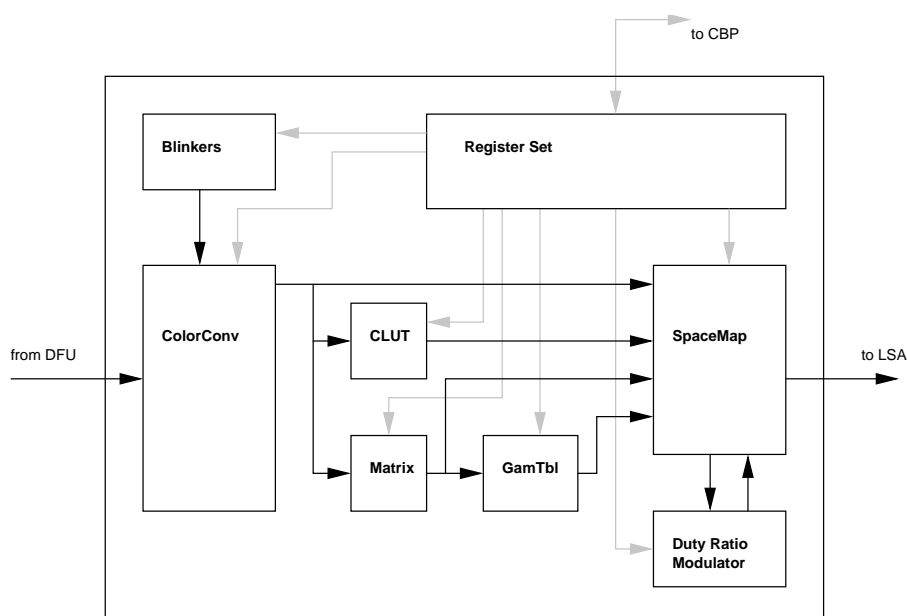




**Figure 1-2:** DFU structure. Pixel data is transported from SDC via the FetchFSM to the Input FIFO. From there it is read out by the PixelPump and fed into CCU. The register set contains all necessary geometric and color space values to fetch the pixels. Some of the registers are double-buffered and updated on a per-frame basis.

### 1.2.3 CCU Function

The CCU (Color Conversion Unit) handles all transformations necessary to convert the layers displayed from their respective color space to one common color space. Further, special colors for each layer can be defined that trigger transparency and blinking. These colors are detected by the CCU and the appropriate action is carried out.



**Figure 1-3:** CCU structure. Pixel data is fed in by DFU and travels through ColorConv, optionally through CLUT, and is prepared by SpaceMap for output to LSA.

The CCU contains seven functional sub-units plus its own register set. This register set stores all layer-specific color information which control the transformations. Figure 1-3 shows the structure of the CCU. The Blinkers component provides a pulse-width modulated blink signal for each layer. In the ColorConv component blink and transparent colors are detected. The CLUT (Color Look-Up Table) is used to convert layers with color resolutions of less than or equal to 8 bits to higher resolutions. There is only one CLUT for

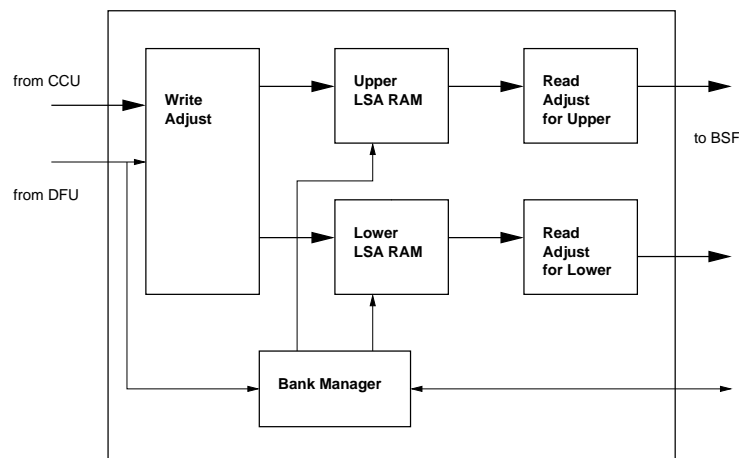
all layers, however, each layer may have its own offset for look-up. The Matrix is used for transforming YUV (YCbCr) video data into RGB colorspace (cf. 3.7.2). It is supplemented with a gamma table (GamTbl) to carry out a non-linear inverse gamma correction (cf. 3.7.3). The SpaceMap component merges the individual layer color spaces to one common intermediate color space and further to one physical color space. The Duty Ratio Modulator is used to provide pseudo gray levels (or colors) for displays with a low number of bits per pixel. Color space conversion is explained more detailed in chapter 2.

### 1.2.4 LSA Function

The LSA (Line Segment Accumulator) is used to realize the vertical layer order in up to four planes. It acts also as internal FIFO between GPU front end and back end, where clock domain transition between GDC core clock and pixel clock takes place. Figure 1-4 shows the LSA structure.

The LSA contains two RAM blocks controlled by the BankManager component. These components perform the actual FIFO functionality. Three adjustment components (one for writing, two for reading) complete the unit. They are used to efficiently employ the RAMs. There are two RAMs to allow for parallel read-out when interfacing dual-scan displays.

Pixel data is fed in sequentially via WriteAdjust to one of the RAMs. Read-out is performed in parallel through the respective ReadAdjust component. Z-ordering (vertical layer order) is achieved by writing pixels plane by plane sequentially into the LSA from bottom plane to topmost plane, thus overwriting pixels hidden underneath others in higher planes.



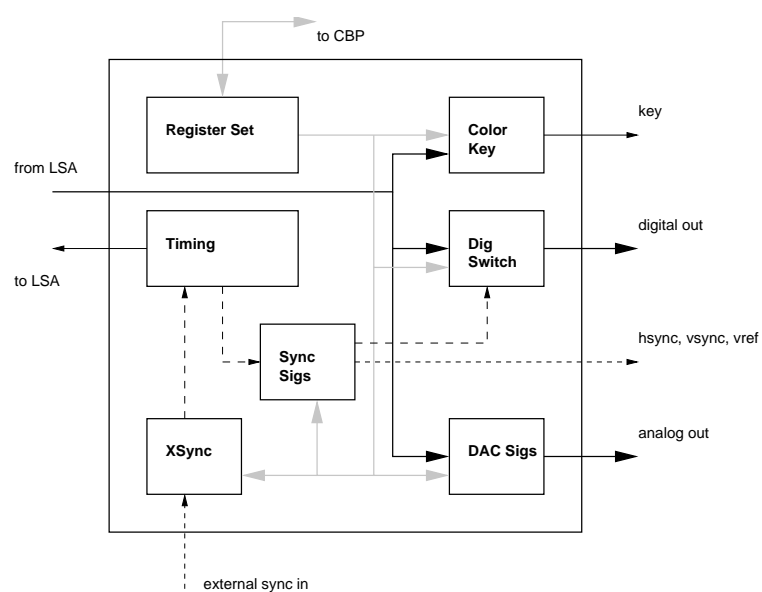
**Figure 1-4:** LSA structure. Pixel data is prepared by WriteAdjust and then stored in one of the RAMs. During read-out data is adjusted once again by the respective ReadAdjust.

### 1.2.5 BSF Function

The BSF (Bit Stream Formatter) unit performs the necessary preparations to actually output the picture on a physical display (either with analog or digital interface). The BSF contains components for signal preparation and for synchronization. The structure is shown in figure 1-5.

The Timing component produces the synchronization frame for image data (see 3.9). It controls the LSA read-out and provides positional information to the SyncSigs component. The XSync component supports Timing when GDC is running with external synchronization. The SyncSigs component generates synchronization signals for the physical display in a very flexible way (see 3.10). Further, it can produce a programmable interrupt. The components DigSwitch and DAC Sigs prepare pixel data for output on displays with digital or analog interface, respectively. They select necessary signals and perform a delay compensation between pixel and synchronization signals.

The ColorKey component allows the application of chroma-keying with GDC by detecting whether the pixel lie within a programmable color range (see 3.14.5).



**Figure 1-5:** BSF structure. Pixel data is fetched from LSA and prepared for output by DigSwitch and DACSigs. Timing controls this fetching and establishes a position reference for SyncSigs, which generates synchronization signals.

## 2 Color Space Concept

### 2.1 Background

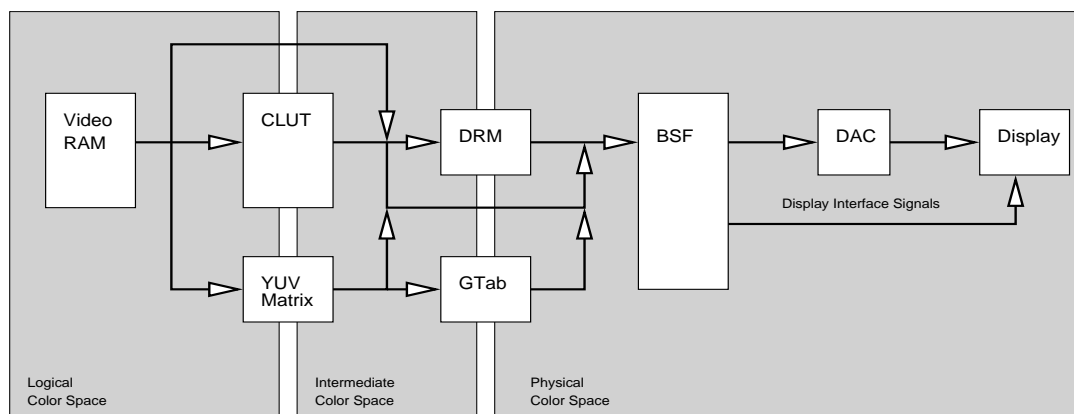
The GDC-ASIC allows to use several numbers of bits per pixel to represent pictures in its VideoRAM (1, 2, 4, 8, 16, and 24 bits per pixel). They may even be different in different planes. The color<sup>1</sup> space defined by these bits shall be referred to as *logical* color space.

Actual displays, however, might have color resolutions differing from those in the logical space. The number of bits per pixel as supported by the display actually wired to the GDC-ASIC shall be referred to as *physical* color space.

Obviously, there is the need to map the logical onto the physical color space. Additional internal units such as a *Color Look-up Table* (CLUT), a *YUV to RGB Matrix* (Jasmine only), and a *Duty Ratio Modulator* (DRM) lead to another internal representation which shall be referred to as *intermediate* color space. This internal representation is laid out as to provide some kind of common ground between logical and physical color space.

### 2.2 Data Flow for Color Space Conversion within GPU

Figure 2-1 gives an overview on the logical data flow for color space conversion. This does not represent actual GPU-pipeline stages but conceptual fields where the different color spaces apply.



**Figure 2-1:** Basic data flow for color space conversion.

Pixel data held in VideoRAM belongs to logical color space. It is mapped to intermediate color space through one of the ways listed in Table 2-1. Mapping to physical color space is carried out according to Table 2-2. Signals actually wired to the physical display are interpreted in this physical color space. However, there is one additional stage of conversion: bit stream formatting. Displays may require data in a stream with bit groups not necessarily equal to one pixel. Therefore, an appropriate postprocessing is indispensable (see 1.2.5 and 3.3).

1. "color" here refers to different hues as well as to different shades of monochrome

## 2.3 Mapping from Logical to Intermediate Color Space

Table 2-1 lists the available mappings from logical to intermediate color space. The term *direct* designates an immediate mapping wire by wire with more significant bits being clamped to zero as needed. For instance, logical 2bpp is mapped directly onto intermediate RGB111 this way:  $I[3:0] = "0" \& L[1:0]$ .

The term *lookup* means using the look-up table for mapping. The term *aligned* designates a mapping in which wires are connected in groups according to color channels with appropriate value scaling. For instance:

- logical RGB555 to intermediate RGB666 (i.e. expansion of color channel bit with):  
 $I[17:0] = L[14:10] \& L[14] \& L[9:5] \& L[9] \& L[4:0] \& L[4]$
- logical RGB555 to intermediate RGB333 (i.e. reduction of color channel bit with):  
 $I[8:0] = L[14:12] \& L[9:7] \& L[4:2]$

The term *matrix* designates application of a matrix multiplication. When transforming logical YUV422 to an RGB intermediate format an optional chrominance interpolation may be applied (see 3.7). The term *achromatic* describes a mapping from YUV to RGB color spaces by dropping chrominance information and transforming the luminance to gray levels. All matrix and achromatic mappings are auto aligned to their target bit with<sup>1</sup>.

The mapping from logical to intermediate color space is controlled by the settings for logical color space in the layer's respective layer description record and the setting of intermediate color space, matrix parameters, and transfer bit in the merging description record.

## 2.4 Mapping from Intermediate to Physical Color Space

Table 2-2 lists the mappings from intermediate to physical color space. The terms *direct* and *aligned* keep their meaning. The term *modulated* designates the usage of the Duty Ratio Modulator (see 3.8). The purpose of this unit is to provide additional (pseudo-) levels (shades of hue or gray) for output, i.e. virtually expand the color resolution in the physical color space. This is achieved by tuning the duty ratio of actually existing physical bits. Accordingly, the term *gamma* means employing the gamma tables for mapping.

The mapping is controlled by the settings of intermediate color space in the merging description record and the setting of physical color space in the display interface record. The resulting mapping from logical to physical color spaces is shown by table 2-3. It contains the possible path of transformation from pixel in VideoRAM to those on the physical display.

---

1. Since matrix results are always in RGB888 color space this auto aligning implies at most a reduction of bit-width whereas achromatic mapping can result in both, bit width reduction as well as expansion (see also 3.7.2).

Table 2-1: Mapping from logical to intermediate color space. Mappings in normal face are default, those in italics designate alternative mappings.

logical color space	intermediate color space codes							bits per pixel	code
	1bpp	RGB111	4bpp	RGB222	RGB333	RGB444	RGB666		
1bpp	lookup / <i>direct</i>	lookup						1	0
2bpp	lookup / <i>direct</i>		lookup / <i>aligned</i>	lookup				2	1
4bpp	lookup	lookup / <i>direct</i>		lookup				4	2
8bpp	lookup			lookup / <i>direct</i>	lookup / <i>aligned</i>	lookup		8	3
RGB555	n/a				aligned		aligned / <i>direct</i> <sup>a</sup>	16	4
RGB565					aligned		aligned / <i>direct</i> <sup>a</sup>	16	5
RGB888					aligned		direct	24	6
YUV422					matrix / <i>achromatic</i>			16	7
YUV444					matrix / <i>achromatic</i>			24	8
YUV555 <sup>b</sup>					matrix / <i>achromatic</i>			16	14
YUV655 <sup>b</sup>					matrix / <i>achromatic</i>			16	15
bits per pixel	1	3	4	6	9	12	18	24	
code	0	9	2	10	11	12	13	6	

a. Direct mapping is not available when RGB gamma forcing is applied.

b. This format is not available for Lavender.

Table 2-2: Mapping from intermediate to physical color space.

intermediate color space	physical color space							bits per pixel	code
	1bpp	RGB111	4bpp	RGB333	RGB444	RGB666	RGB888		
1bpp	direct	n/a						1	0
RGB111	n/a	direct	n/a					3	9
4bpp	modulated	direct		n/a				4	2
RGB222	n/a	modulated	n/a					6	10
RGB333	n/a			direct / $\gamma$ <sup>a</sup>	n/a			9	11
RGB444	n/a				direct / $\gamma$ <sup>a</sup>	n/a		12	12
RGB666	n/a					direct / $\gamma$ <sup>a</sup>	n/a	18	13
RGB888	n/a						direct / $\gamma$ <sup>a</sup>	24	6
bits per pixel	1	3	4	6	12	18	24		
code	0	9	2	11	12	13	6		

a. Selectable for intermediate color space stemming from one of the YUV logical color spaces, or if RGB gamma forcing is applied.

Table 2-3: Resulting mapping from logical to physical color space

logical color space	physical color space							bits per pixel	code
	1bpp	RGB111	4bpp	RGB333	RGB444	RGB666	RGB888		
1bpp	dd, ld, lm	ld, lm	ld	ld	ld	ld	ld	1	0
2bpp	dd, ld,am, lm	dd, ld, ad, lm	ad, ld	ld	ld, la	ld, la	ld	2	1
4bpp	ld, dm , lm	dd, ld, lm	dd, ld	ld	ld, la	ld, la	ld	4	2
8bpp	ld, lm	ld, dm, lm	ld	ad, ld	ld	ld	ld	8	3
RGB555				ad	ad	dd, ad	dd, ad	16	4
RGB565				ad	ad, aa	dd, ad	dd, ad	16	5
RGB888				ad	ad	ad	dd	24	6
YUV422				xd, yd	xd, yd	xd, yd	xd, yd	16	7
YUV444				xd, yd	xd, yd	xd, yd	xd, yd	24	8
YUV555 <sup>a</sup>				xd, yd	xd, yd	xd, yd	xd, yd	16	14
YUV655 <sup>a</sup>				xd, yd	xd, yd	xd, yd	xd, yd	16	15
bits per pixel	1	3	4	9	12	18	24		
code	0	9	2	11	12	13	6		

a. This format is not available for Lavender.



Meaning of abbreviations:

- d: direct mapping
- l: look-up mapping
- a: aligned mapping

- m: mapping by duty modulation
- x: matrix multiplication
- y: conversion to achromatic

## 3 GPU Control Information

The following sections describe the information necessary for output of video and picture data on a display wired to GDC. The tables give only the type of information and its respective size, they do not imply any assignment to GDC-registers. The intention is to sort information according to their conceptual place of application and to group data belonging together.

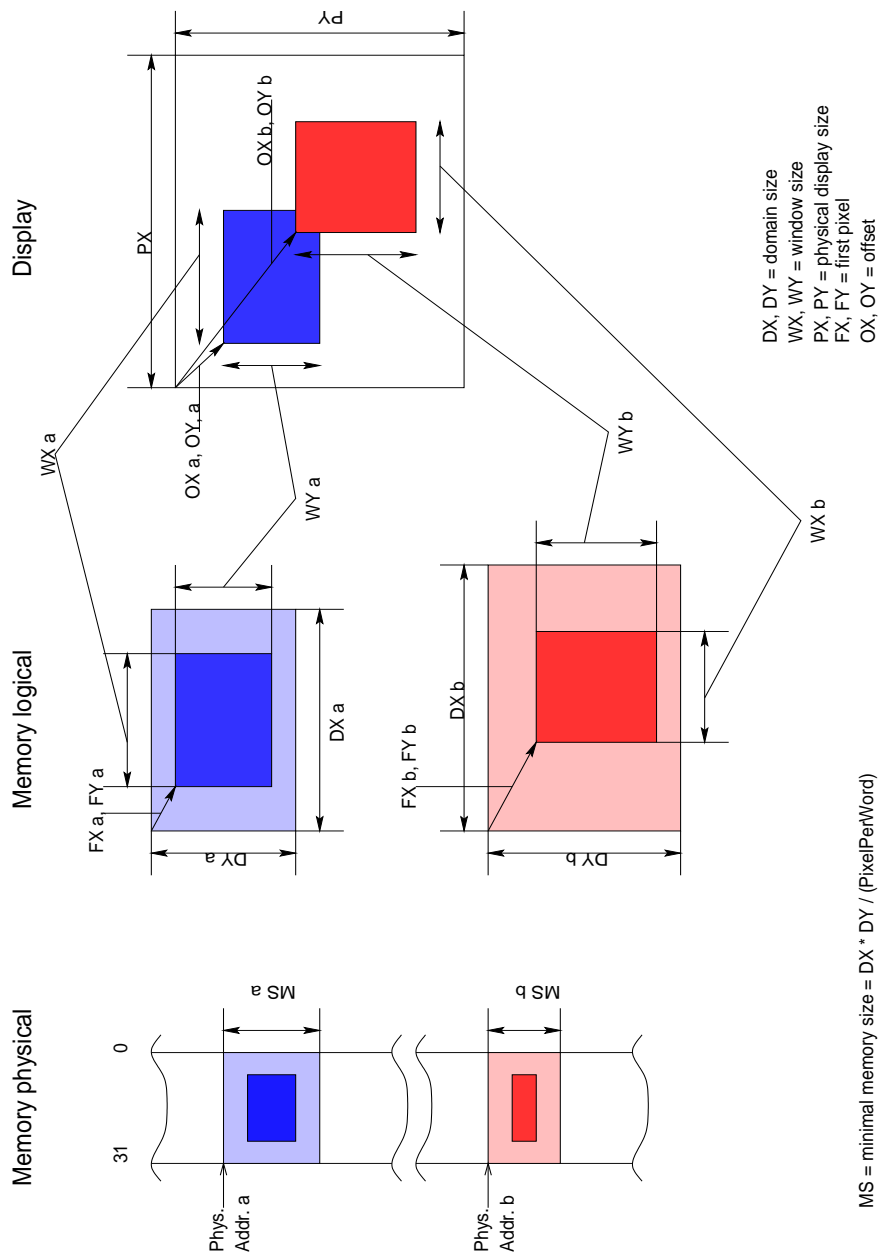
### 3.1 Layer Description Record

This group of data describes all information belonging to a single layer *l* (graphics or video) in VideoRAM (i.e. within logical color space).

**Table 3-1:** Layer Description Record

	Information	Size (bits)
1	<b>Physical address in VRAM<sup>a</sup></b> The address of the word containing the first pixel of the layer according to MCU address space.	32
2	<b>Domain size DX<sub>l</sub><sup>a</sup>, (DY<sub>l</sub>)</b> Logical area covered by the layer in VideoRAM. DY only pro forma, the user is responsible for providing enough RAM space per layer.	2 x 14 unsigned
3	<b>First pixel to be displayed FX<sub>l</sub>, FY<sub>l</sub></b> Upper left corner of area in VRAM actually to be displayed	2 x 14 unsigned
4	<b>Display window size WX<sub>l</sub>, WY<sub>l</sub></b> Size of area to be displayed	2 x 14 unsigned
5	<b>Display window offset OX<sub>l</sub>, OY<sub>l</sub></b> Offset of layer area from upper left corner of physical display.	2 x 14 signed
<i>See figure 3-1 for an illustration of geometry.</i>		
6	<b>Color space code</b> See section 3.12.1	4
7	<b>Transparent color</b> The color (in logical color space) to be ignored for display	1...24
8	<b>Transparency enable</b>	1
9	<b>Blink color</b> (confirming to 6)	1...24
10	<b>Blink alternative color</b> (confirming to 6) If a pixel in video RAM has a logical color equal to <i>blink color</i> , then it is displayed alternating in <i>blink color</i> / <i>alternative color</i> . Either color may also be the <i>transparent color</i> , in this case the pixel and the layer below it are displayed alternatively.	1...24
11	<b>Blink rate</b> # of frames on, # of frames off	2 x 8
12	<b>Blink enable</b>	1

a. This information is held in SDC



**Figure 3-1:** Illustration for geometric relations between VideoRAM, layer coordinates and display coordinates.

## 3.2 Merging Description Record

This record contains all data needed to convert each logical layer into intermediate and physical color space, together with appropriate handling of Z-order.

**Table 3-2:** Merging Description Record.

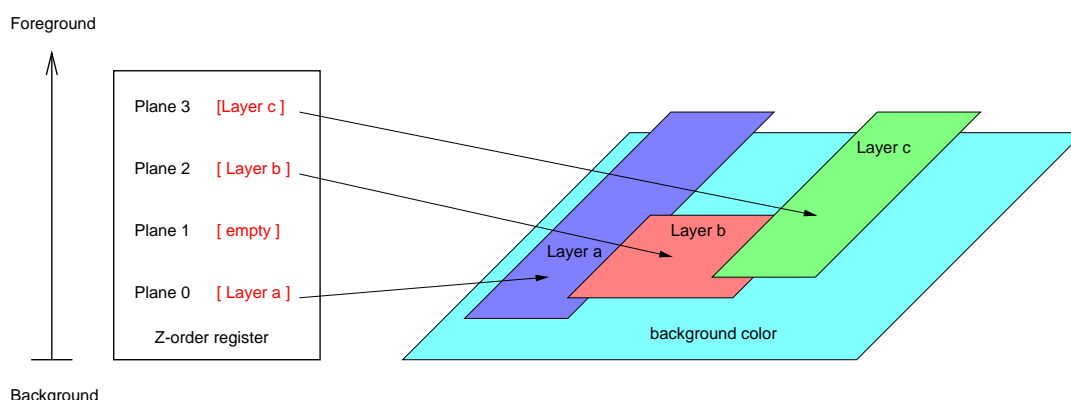
Information		Size (bits)
1	<b>Color Space Code</b> for intermediate color space	4
2	<b>Layer to intermediate transfer codes</b> (one per layer) Flag to decide between two ways of transforming the layer's logical color space into the intermediate (common) color space. Maximal two legal ways are defined for all combinations of logical to intermediate color space mapping (see section 2.3).	16 x 1
3	<b>Z-order register</b> Contains a place for each layer concurrently displayed. This defines the Z-order, i.e. the first place contains the number of the topmost layer, the second place the number of the layer immediately below and so on (see fig. 3-2).	4 x 4
4	<b>Plane enable</b> Flag to switch display of the respective plane on or off.	4
5	<b>CLUT offsets</b> (one per layer) This offset is added to the pixel value when forming the address for look-up.	16 x 9
6	<b>CLUT contents</b>	256 x 1..24 <sup>a</sup> 512 x 1..24 <sup>b</sup>
7	<b>Background color</b> (confirming to intermediate color space) The color to be displayed on locations where there is no pixel from any layer of the video RAM. This allows to keep the visible layers as small as possible to save RAM bandwidth yet still be able to control the appearance of the display background.	1...24
8	<b>Background enable</b> This flag can be used to improve GPU performance when there is at least one layer that covers the whole display area.	1
9	<b>Pseudo level duty ratios</b> These give the values used by the duty ratio modulator to produce pseudo levels. Since the modulator produces either 16 (4 bit to 1 conversion) or 3 x 4 (6 bit to 3 conversion) pseudo levels there is a maximum of 14 really modulated levels, two are simply on and off level.	14 x 6
10	<b>Pre-Matrix Biases (Jasmine only)</b> These values are added to Y, U, V before matrix multiplication	3 x 8
11	<b>Matrix Coefficients (Jasmine only)</b> Factors for matrix multiplication, all treated as unsigned integer in the range of 0...2 (see 3.7.2).	9 x 8
12	<b>Gamma Correction Table contents (Jasmine only)</b> Values used for Gamma correction after multiplication. Correction is done by table look-up.	3 x 256 x 8
13	<b>Gamma Table enable (Jasmine only)</b> Flag to directly map the matrix result into physical color space.	1

**Table 3-2:** Merging Description Record. (*Continued*)

Information		Size (bits)
14	<b>RGB Gamma Forcing (Jasmine only)</b> Flag to force pixels from RGB555, RGB565, RGB888 through the gamma table, valid only when gamma table enable is set.	1
15	<b>YUV422 Chroma interpolation enable (Jasmine only)</b> Since in logical color space YUV422 two pixels share their chrominance information linear interpolation between pairs of adjacent pixels is used to smooth the appearance of the picture.	1

- a. For MB87J2120 (Lavender).  
b. For MB87P2020-A (Jasmine).

**Remark:** The purpose of the background color and Z-order register is a clearer concept of visibility, a more user-friendly programming interface and a unified handling of all layers. If there was no Z-order register the user still would be able to swap layers by changing their respective VideoRAM start addresses. However, this would require more accesses than the single access to the Z-order register.

**Figure 3-2:** Illustration for Z-order register.

### 3.3 Display Interface Record

This group of data contains information necessary to describe the geometric and timing behaviour of the physical display as well as timing data to generate various synchronization signals.

**Table 3-3:** Display Interface Record.

Information		Size (bits)
1	<b>Display physical size PX, PY</b>	2 x 14
2	<b>Color space code</b> For physical color space, see 3.12.1	4
3	<b>Bit stream format code</b> Defines the number of bits written per video clock period (see section 3.12.2)	4
4	<b>Scan mode</b> Distinction between single / dual scan, special “Optrex-mode”	2

**Table 3-3:** Display Interface Record. (*Continued*)

<b>Information</b>		<b>Size (bits)</b>
5	Display mode Distinction between single and twin display mode.	1
6	R/B Swap Flag to allow for a swap between red and blue channel for RGB111 color space, only useful for bit stream formats S4 and S8	1
7	<b>Physical width in scan dots</b> = $PX * BitsPerPixel / BitsPerScanClock$	14
8	<b>Dual Scan Offset</b> Distance between the two lines which are concurrently output.	14
9	<b>External Sync Control</b> Control bits to define reactions on external sync signals, see 3.9	4
10	<b>Master Timing Definition</b> Positional parameters to control output and sync signal generation, see 3.9	6 x 15
11	<b>Sync Pulse Generator Control</b> Position parameters for pulse generators, see 3.10.2	6 x 4 x 31
12	<b>Sync Sequencer Control</b> Parameters for sequence-based signal generation, see 3.10.3	64 x 32 + 5
13	<b>Sync Mixer Control</b> Parameters that control the final generation of sync signals from merged pulse generator and sync sequencer outputs, see 3.10.4	8 x 47
14	<b>Sync Switch</b> Half cycle delay on/off flag for each sync mixer output	8
15	<b>Pixel Clock Gate Control</b> Bits to determine clock gating, divider and gating type, see 3.11	4
16	<b>Color Key Limits</b> Upper and lower color limit for which a match signal is generated, confirming to color space	2 x 3 x 1...8
17	Color Key Polarity Selects whether match signal is low or high active	1
18	<b>Blank Clamping Values</b> Values that are output at analog and digital outputs when blanking is active (no pixel data is output)	1 x 1..24 <sup>a</sup> 2 x 1..24 <sup>b</sup>
19	<b>Main Output Enable Flags</b> Bits to control tristate drivers for display signals	29

a. Lavender contains one clamping value for digital and analog output.

b. Jasmine contains two different clamping values for digital and analog output.

### 3.4 Supported Physical Color Space / Bit Stream Format Combinations

Table 3-4 gives an overview on the legal physical color space / bits stream format combinations for the primary display. These are derived from the features of the supported displays. An analog output for S9...S12 is also possible in *twin display mode* (see next section).

**Table 3-4:** Supported combinations of physical color space to bit stream format

Physical color space codes	Bit stream format code									
	S1	S2	S4	S6	S8	S9	S12	S18	S24	AN
1bpp	x	x	x							
RGB111			x	x	x					
4bpp			x		x					
RGB333						x				
RGB444							x			
RGB666								x		
RGB888									x	x

### 3.5 Twin Display Mode

Twin Display Mode is only available for MB87P2020-A (Jasmine). MB87J2120 (Lavender) does not support this feature.

The GPU offers the opportunity to run a digital and an analog display in parallel (to output according data at digital and analog output pins, respectively). This is referred to as *twin display mode* and may be used to concurrently output the picture and post-processing it (e.g. do some color keying). The implication, however, is that both data streams use identical resolutions and sync timing. Some sync signals may not be available at the multipurpose pins as well (see 3.14.1). In twin display mode one display becomes the *primary display*, the other the *secondary display*. The distinction which display is primary depends on the bit stream format as shown in table 3-5.

**Table 3-5:** Primary / secondary display distinction in twin display mode

Bit stream format	Primary display	Secondary display
AN	Analog	Digital
S9...S24	Digital	Analog
Others	Digital	<i>No twin display mode available.</i>

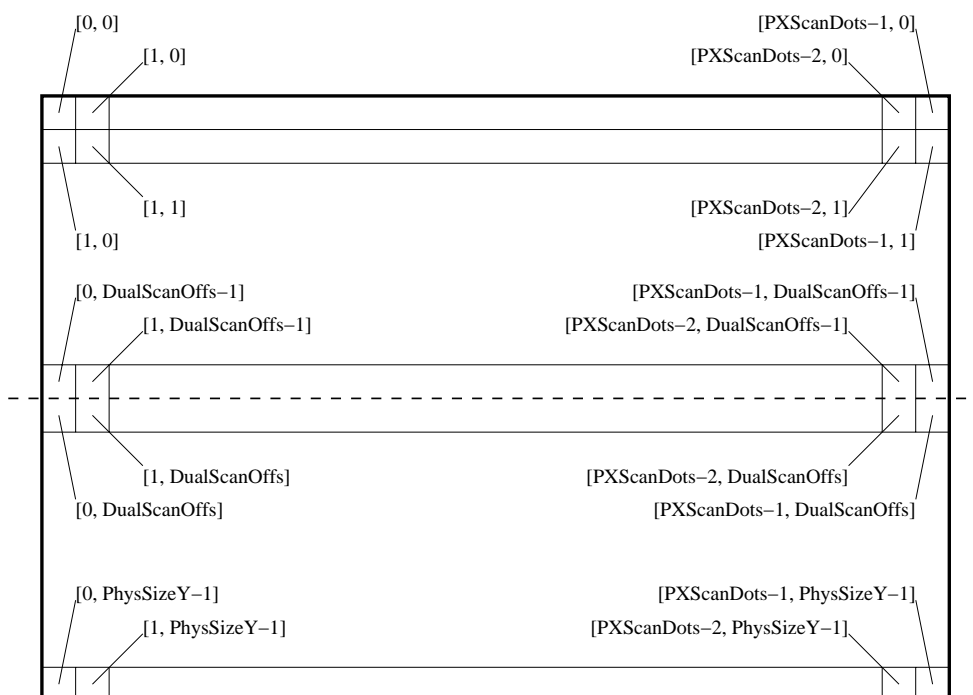
When the analog display is secondary pixel data are auto scaled to 8 bit per color channel for DAC outputs to maintain full voltage swing (bit with at the digital outputs remains according to selected bit stream format). This auto scaling works as explained for bit expansion in 2.3.

### 3.6 Scan Modes

Three scan modes are supported and determine the sequence pixel data that form the bit streams being output. Figure 3-3 gives an illustration on geometrical reference points on the display used later to describe the according bit streams.

4. **Single Scan:**  
Pixels are written in one continuous stream, starting at upper left corner, proceeding left to right, line by line.
5. **Dual Scan:**  
Pixels are written in two parallel streams. The first stream starts at the upper left corner, proceeding left to right, line by line. The second starts at the left margin of the line defined by DualScanOffset (usually  $PY/2$ ), proceeding left to write, line by line. Each stream contains DualScanOffset lines, i.e. the stream for the upper display segment determines the number of lines
6. **Zigzag Scan:**  
aka "Optrex-mode". Pixels are written in one stream, starting at the upper left corner, proceeding from left to right, however, line number sequence is peculiar: every other line number is biased with DualScanOffset, i.e. the first line written is line #0, the second line is line number DualScanOffset, the third line written is line #1, the fourth line written is line #(DualScanOffset+1) etc.

Table 3-6 shows the resulting bit streams according to the different scan modes. Please note that these streams, as well as the reference points in figure 3-3 count in scan dots, which are not necessarily pixels (this is determined by the bit stream format). A scan dot consists of all bits output during on display clock cycle and may contain the bits of more than one pixel or even bits from fractions of adjacent pixels.



**Figure 3-3:** Geometric reference points on the display.



**Table 3-6:** Bit stream output sequence according to scan mode, given as [x, y] pairs. Where not stated differently, one row is equivalent to one display clock cycle.

Single Scan	Dual Scan		Zigzag Scan	Note
	Upper Stream	Lower Stream		
	0...n cycles optional blanking (vertical and horizontal)			a
[0, 0]	[0, 0]	[0, DualScanOffs]	[0, 0]	b
[1, 0]	[1, 0]	[1, DualScanOffs]	[1, 0]	
	...			
[PXScanDots-2, 0]	[PXScanDots-2, 0]	[PXScanDots-2, DualScanOffs]	[PXScanDots-2, 0]	
[PXScanDots-1, 0]	[PXScanDots-1, 0]	[PXScanDots-1, DualScanOffs]	[PXScanDots-1, 0]	c
	0...n cycles optional blanking (horizontal)			d
[0, 1]	[0, 1]	[0, DualScanOffs+1]	[1, DualScanOffs]	e
[1, 1]	[1, 1]	[1, DualScanOffs+1]	[2, DualScanOffs]	
	...			
[PXScanDots-1, 1]	[PXScanDots-1, 1]	[PXScanDots-1, DualScanOffs+1]	[PXScanDots-1, DualScanOffs]	f
	0...n cycles optional blanking (horizontal)			g
[0, 2]	[0, 2]	[0, DualScanOffs+2]	[0, 1]	
[1, 2]	[1, 2]	[1, DualScanOffs+2]	[1, 1]	
	...			
[PXScanDots-2, 2]	[PXScanDots-2, 2]	[PXScanDots-2, DualScanOffs+2]	[PXScanDots-2, 1]	

**Table 3-6:** Bit stream output sequence according to scan mode, given as [x, y] pairs. Where not stated differently, one row is equivalent to one display clock cycle.

Single Scan	Dual Scan		Zigzag Scan	Note
	Upper Stream	Lower Stream		
[PXScanDots-1, 2]	[PXScanDots-1, 2]	[PXScanDots-1, DualScanOffs+2]	[PXScanDots-1, 1]	
	0...n cycles optional blanking (horizontal)			
[0, 3]	[0, 3]	[0, DualScanOffs+3]	[0, DualScanOffs+1]	
[1, 3]	[1, 3]	[1, DualScanOffs+3]	[1, DualScanOffs+1]	
	...		[2, DualScanOffs+1]	
	...			
[PXScanDots-2, PhysSizeY-1]	[PXScanDots-2, DualScanOffs-1]	[PXScanDots-2, 2*DualScanOffs-1]	[PXScanDots-2, DualScanOffs-1]	
[PXScanDots-1, PhysSizeY-1]	[PXScanDots-1, DualScanOffs-1]	[PXScanDots-1, 2*DualScanOffs-1]	[PXScanDots-1, DualScanOffs-1]	h
	0...n cycles optional blanking (horizontal and vertical)			i

- leading blanking before start of active frame area
- leftmost scan dot of topmost line (start of active display area)
- rightmost scan dot on first line, last scan dot of horizontal cycle for single and dual scan
- trailing horizontal blanking after first line for single and dual scan,  
for zigzag scan there is **no** blanking between rightmost scan dot of topmost line and leftmost scan dot of line number DualScanOffs
- start of second line (leftmost scan dot) for single and dual scan
- rightmost scan dot on second line for single and dual scan,  
for zigzag scan this is rightmost scan dot on line number DualScanOffs, i.e. last scan dot of horizontal cycle for zigzag scan
- trailing horizontal blanking
- last scan dot of last line, end of active display area
- trailing horizontal and vertical blanking

Please refer to section 3.9 for a more detailed explanation of display timing, especially blanking insertion.

### 3.7 YUV to RGB conversion

Full support for YUV to RGB conversion as described in this section is only available for MB87P2020-A (Jasmine).

MB87J2120 (Lavender) is able to handle YUV444 and YUV422 in a limited manner. It simply ignores the color components of the color value and displays it in 256 step grayscale format also called 'achromatic mapping'.

This behaviour is also available for Jasmine as alternative mapping for YUV422, YUV444, YUV555 and YUV655.

#### 3.7.1 YUV422 Demultiplexing and Chrominance Interpolation

The YUV422 format is special in the respect that it implies chroma sub-sampling and distribution of U(Cb) and V(Cr) values on every other pixel (see figure 3-4 below).

<i>Luminance</i>	...	$Y_{2i-2}$	$Y_{2i-1}$	$Y_{2i}$	$Y_{2i+1}$	$Y_{2i+2}$	$Y_{2i+3}$	...	8 bits each
<i>Chrominance</i>	...	$U_{2i-2}$	$V_{2i-2}$	$U_{2i}$	$V_{2i}$	$U_{2i+2}$	$V_{2i+2}$	...	8 bits each
		$Chroma_{2i-2}$		$Chroma_{2i}$		$Chroma_{2i+2}$			

**Figure 3-4:** Memory map for pixels in YUV422 format. Pixels on even positions carry U-chrominance, those on odd positions V-chrominance,  $i = 0, 1, 2, \dots$

Therefore, pairs of pixels have to be fetched from memory to obtain complete color information. This results in the constraints that layer's horizontal first pixels (FX) as well as their horizontal window size (WX) must not be odd. Due to internal processing of pixels in portions of display segments the horizontal display offset (OX) is also limited to even positions. These restrictions are enforced automatically by GPU by ignoring the LSBs of the parameters mentioned before.

In order to process the YUV422 format in the matrix (see next section) the PixelPump within the DFU does the necessary demultiplexing to assign each pixel the complete chrominance information, i.e. pixels arriving at the matrix have the structure shown in figure 3-5.

<i>Luminance</i>	...	$Y_{2i-2}$	$Y_{2i-1}$	$Y_{2i}$	$Y_{2i+1}$	$Y_{2i+2}$	$Y_{2i+3}$	...	8 bits each
<i>Chrominance</i>	...	$U_{2i-2}$		$U_{2i}$		$U_{2i+2}$		...	8 bits each
	...	$V_{2i-2}$		$V_{2i}$		$V_{2i+2}$		...	8 bits each

**Figure 3-5:** Pixel structure after YUV422 demultiplexing. Although now each pixel is assigned complete chrominance information there is still a chrominance sub sampling

The effect of chrominance sub sampling can be decreased and hence the appearance of the image improved by applying a linear interpolation between consecutive chrominance values as illustrated in figure 3-6. This interpolation is controlled by a configuration register.

<i>Luminance</i>	...	$Y_{2i-2}$	$Y_{2i-1}$	$Y_{2i}$	$Y_{2i+1}$	$Y_{2i+2}$	$Y_{2i+3}$	...
<i>Chrominance</i>	...	$U_{2i-2}$	$\frac{U_{2i-2} + U_{2i}}{2}$	$U_{2i}$	$\frac{U_{2i} + U_{2i+2}}{2}$	$U_{2i+2}$	$\frac{U_{2i+2} + U_{2i+4}}{2}$	...
	...	$V_{2i-2}$	$\frac{V_{2i-2} + V_{2i}}{2}$	$V_{2i}$	$\frac{V_{2i} + V_{2i+2}}{2}$	$V_{2i+2}$	$\frac{V_{2i+2} + V_{2i+4}}{2}$	...

**Figure 3-6:** Pixel structure after linear chrominance interpolation

### 3.7.2 Matrix Multiplication

GPU incorporates a YUV to RGB Matrix with prebiasing to convert pixels in YUV (YCbCr) color spaces to RGB color spaces. To flexibly adjust to different input standards, all matrix coefficients can be configured. The following equation is calculated for the YUV to RGB conversion:

$$\begin{pmatrix} R_{\text{mtx}} \\ G_{\text{mtx}} \\ B_{\text{mtx}} \end{pmatrix} = \begin{pmatrix} C_{YR} & C_{UR} & C_{VR} \\ C_{YG} & C_{UG} & C_{VG} \\ C_{YB} & C_{UB} & C_{VB} \end{pmatrix} \times \begin{pmatrix} Y + \text{PreBias}_Y \\ U + \text{PreBias}_U \\ V + \text{PreBias}_V \end{pmatrix} \quad (6)$$

All matrix coefficients  $C_{ij}$  are derived from their respective configuration registers  $P_{ij}$  in either of the following ways:

- a) Chroma coefficients for green channel  $C_{UG}$  and  $C_{VG}$ :

$$C_{ij} = -(P_{ij}/128) \quad (7)$$

- b) all others:

$$C_{ij} = (P_{ij}/128) \quad (8)$$

That is, all configuration registers are treated as 8 bit unsigned binary numbers, thus resulting in a range of  $-2 \dots 0$  for coefficients  $C_{UG}$  and  $C_{VG}$  and  $0 \dots 2$  for all other coefficients. All *PreBias* values are stored as 8 bit signed binary numbers in their configuration registers (cf. table 4-1).

The calculated intensity values for R, G and B are limited to 8 bit positive binary numbers with values outside the legal range being clipped to the respective margins, i.e. underflows (negative values) are mapped to zero, overflows (values  $>255$ ) to 255.

**Note:** There is only one set of matrix parameters (pre biases and coefficients) for all layers. YUV color spaces with less than 8 bits per channel (i.e. YUV555 and YUV655) are therefore auto scaled to YUV444 in advance to fit into a unified value range for all YUV color spaces. This scaling works similar to that explained for the mapping from logical to intermediate color spaces in section 2.3.

### 3.7.3 Inverse Gamma Correction

When processing YUV material originally produced for display on CRTs color channel intensities are calculated according to equation (6) may be distorted by a gamma function according to (9) to cope with the non linear electro-optical characteristics of the CRT's phosphorus.

$$I_{\text{mtx}} = c \cdot I_{\text{act}}^\gamma + I_0 \quad (9)$$

with  $I_{\text{mtx}}$  being the intensity of one color channel according to equation (6),  $I_{\text{act}}$  the actual (linear) intensity,  $c$  and  $I_0$  constants and  $\gamma \approx 2.2$ . However, a linear representation (i.e. no gamma distortion) might be demanded. Therefore, GPU includes three gamma look-up tables (one per color channel) to accomplish the inverse transformation to obtain  $I_{\text{act}}$  from  $I_{\text{mtx}}$  as well as any other (possibly non linear) post-matrix trans-

formation. This is done by loading the gamma tables with the appropriate contents. For instance, the inverse gamma correction would require the contents of address  $I_{\text{mtx}}$  to be:

$$I_{\text{act}} = [I_{\text{mtx}}] = ((I_{\text{mtx}} - I_0)/c)^{1/\gamma} \quad (10)$$

In a different application the gamma look-up tables might be used to adopt RGB color spaces to special display characteristics. This is referred to as *RGB gamma forcing*. In this case, the gamma tables are not available for YUV gamma correction anymore, although the matrix still can be used for YUV to RGB conversion.

## 3.8 Duty Ratio Modulation

### 3.8.1 Working Principle

This paragraph briefly describes the Duty Ratio Modulator integrated. The purpose of this unit is to provide additional (pseudo-) levels (shades of hue or gray) for output, i.e. virtually expand the color resolution in the physical color space. This is achieved by tuning the duty ratio of actually existing physical bits.

Assume there is a frame rate of 100 Hz on a black and white display (1 bit physical color space). Pseudo gray levels can be obtained when pixels are not set black all 100 frames per second but say 80 frames only. Thus, the pixel is seen in dark gray instead of black. The less time a pixel is set to black the lighter the gray becomes. This is equivalent to modulating the duty ratio of the pixel signal.

Vertical sync (i.e. the frame rate) is used to modulate the pixels, since this signal must be common for all pixels on display because otherwise artefacts may occur. If the decision whether to set a pixel to black or white to obtain the desired pseudo gray level is made e.g. on a line-by-line basis it depends on the ratio between the number of lines and the set pseudo gray level where (geometrically on the display) “gray” pixels are set to black or white. Unsuitable settings could then cause actually “gray” pixels to stay either black or white or visibly flicker. Therefore, all “gray” pixels of the frame of the same pseudo gray level are set to the same physical value of either black or white to achieve a unified optical impression.

The number of frames constituting a basic modulation period depends on the number of pseudo gray levels desired and on the linearity of the display characteristics. For instance, if just black, white and gray is intended the basic modulation period might be limited to two frames. Black pixels are set black every frame (100% duty ratio), white pixels are not set black at all (0% duty ratio) and “gray” pixels are set black every other frame (50% duty ratio). However, the “gray” optically perceived might not be 50% black. This is due to a possibly nonlinear electro-optical characteristics of the display. In order to cope with this adjustments must be made possible. An optically 50% black might be achieved with a 60% duty ratio, for instance. One should be able to adjust the duty ratio with an accuracy of some percent (5 or 6 bits precision). Hence, longer periods i.e. more frames (30...100) are needed.

When using a basic period of e.g. 100 frames a 50% duty ratio is equivalent to an overall 50 frames on and 50 frames off. This implies no statement about when to display the pixel as active and inactive, respectively. Setting the pixel first for 50 frames active and afterwards the next 50 frames inactive seems to be equivalent to setting the pixel active every other frame.

Unfortunately, this is not the case in general. Most displays feature a frame rate in the magnitude of 100Hz which means that the modulation period of 100 frames lasts for one second. Both methods are equivalent only for displays slow enough to integrate over this rather long period. On faster displays the first method (simple pulse width modulation with 50 consecutive frames activity) will cause a blinking of 2 Hz clearly visible for the human eye. Consequently, a simple pulse width modulation is inadequate and a distribution of on and off values as even as possible is required.

This is achieved using a derivation of Bresenham’s line algorithm. This algorithm was originally designed to draw a line from point  $A(x_a, y_a)$  to point  $B(x_b, y_b)$  on a lattice of discrete pixels with minimal deviation from the ideal graph  $y = m \cdot x + n$ . The basic idea is to scan the distance  $\Delta X = x_b - x_a$  step by step and draw pixels at appropriate  $y$ -values using a so-called decision variable. This variable tells for a given  $x(k+1)$  whether to draw the pixel on  $(x(k+1), y(k))$  or  $(x(k+1), y(k)+1)$ .

For the application within the GPU there are some simplifications: The basic equation is reduced to  $y = m \cdot x$ , with  $y$  representing the activity,  $x$  the modulation period and slope  $m$  the desired duty ratio. Further, the distance  $\Delta X$  can be limited to a power of 2. The value of the decision variable from Bresenham's algorithm distinguishing between  $y(k)$  and  $y(k) + 1$  immediately tells when to set the signal active and when inactive.

### 3.8.2 Usage

As listed in Table 2-2, there are two mappings for which the Duty Ratio Modulator is applied: from intermediate 4bpp to physical 1bpp and for intermediate RGB222 to physical RGB111. Usage differs for either case since a different number of pseudo levels is required.

#### Mapping from intermediate 4bpp to physical 1bpp

In this case there is a total of 16 different pixel values in intermediate color space which need be mapped to a total of only two different pixel values in physical color space. The 16 intermediate values are interpreted as gray scale, thus representing black, white and 14 levels of gray. These 14 levels have to be produced by the DRM. The duty ratios used for this 14 pseudo levels are given in GPU registers (see chapter 4). Table 3-7 gives the assignment between pixel value and pseudo levels.

**Table 3-7:** Assignment between pixel values in intermediate and physical color space when using duty ratio modulation for mapping from intermediate 4bpp to physical 1bpp.

pixel value		pixel value	
interm.	phys.	interm.	phys.
0	constant off	8	DRM 7 output
1	DRM 0 output	9	DRM 8 output
2	DRM 1 output	10	DRM 9 output
3	DRM 2 output	11	DRM 10 output
4	DRM 3 output	12	DRM 11 output
5	DRM 4 output	13	DRM 12 output
6	DRM 5 output	14	DRM 13 output
7	DRM 6 output	15	constant on

Please note that for a given DRM its output is either one or zero at any given time. What differs is the duty ratio, which is adjustable via GPU register.

#### Mapping from intermediate RGB222 to physical RGB111

In this case the pixel value is interpreted as three color channels R, G, and B which are mapped individually from intermediate to physical color space. Each intermediate color channel has a width of 2 bits, thus allowing four different values. As a whole, intermediate RGB222 can represent 64 different colors.

However, since each color channel is regarded separately, there are only two pseudo levels per channel, since the other two are constant on and off, respectively. Thus, a whole of six modulators is needed. Table 3-8 gives the assignment for this case.

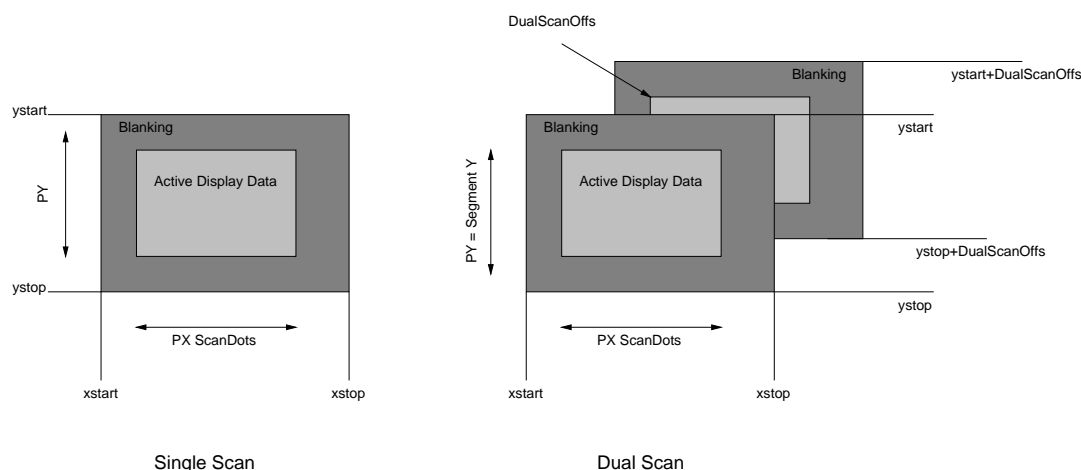
**Table 3-8:** Assignment between pixel values in intermediate and physical color space when using duty ratio modulation for mapping from intermediate RGB222to physical RGB111.

pixel value					
channel R		channel G		channel B	
interm., bits [5:4]	phys., bit [2]	interm., bits [3:2]	phys., bit [1]	interm., bits [1:0]	phys., bit [0]
0	constant off	0	constant off	0	constant off
1	DRM 4 output	1	DRM 2 output	1	DRM 0 output
2	DRM 5 output	2	DRM 3 output	2	DRM 1 output
3	constant on	3	constant on	3	constant on

### 3.9 Master Timing Information

The generation of the correct timing is a rather complex issue, due to the great variety of displays supported. Therefore, a coherent, flexible and easy-to-use approach is extremely important. This is achieved by separating the GPU internal timing (for fetching and bit stream generation) from external timing (for sync signals).

The main timing frame for all display types is completely defined by four or six values. Fig. 3-7 gives a geometrical interpretation of the four primary values.



**Figure 3-7:** Interpretation of main timing parameters in relation to geometrical parameters of the physical display.

The general idea is to assign every scan dot (i.e. the bits written during one display clock cycle) a pair of integer numbers, one for the horizontal and one for vertical timing, thus representing display timing as coordinates within a two-dimensional scan area. X-values smaller than zero represent horizontal blanking before visible pixels, X-values greater than or equal to `PXScanDots` horizontal blanking after visible pixels. Active display area (i.e. visible pixels, either background or layer) is designated by values from 0 to `(PXScanDots - 1)`. The same applies to Y-values. Those below zero designate leading vertical blanking, those greater than or equal to `PY` trailing vertical blanking, and those between zero and `(PY - 1)` visible lines.

There is a special case: TV-conform timing requires that every other frame consists of an odd number of lines to achieve an odd sum of lines for two consecutive frames (representing the two fields of an interlaced TV picture). Therefore, an additional coordinate had to be introduced, referred to as *field bit*, thus actually expanding the coordinate space to three dimensions. Even for this case the number of visible pixels remain constant, what differs is the number of blanking lines.

These “timing coordinates” form the basis for all timing-related signal generation. For a complete description a maximum six timing and three geometrical parameters are necessary. The geometrical parameters are the number of scan dots in each direction, i.e. the area with visible pixels on the physical display, and the extra `DualScanOffs` parameter for dual scan displays (i.e. the number of lines of each display segment).

The number of necessary timing parameters is either four or six, depending whether or not field toggling is enabled (i.e. frames with differing number of blanking lines are used). For normal operation a set of four timing parameters is sufficient: `xstart`, `xstop`, `ystart`, and `ystop`. The “timing coordinates” mentioned above cover a range from [`xstart`, `ystart`] to and including [`xstop`, `ystop`]. If field toggling is used, then there is a pair of (`ystart`, `ystop`) values for every frame type, which is used alternatively.

This explanation applies also to dual scan displays with the distinction that the Y values describe only the upper display segment, the lower segment simply is assigned the same timing (since its respective pixel stream is output in parallel to the upper segment).

## 3.10 Generation of Sync Signals

### 3.10.1 Overview

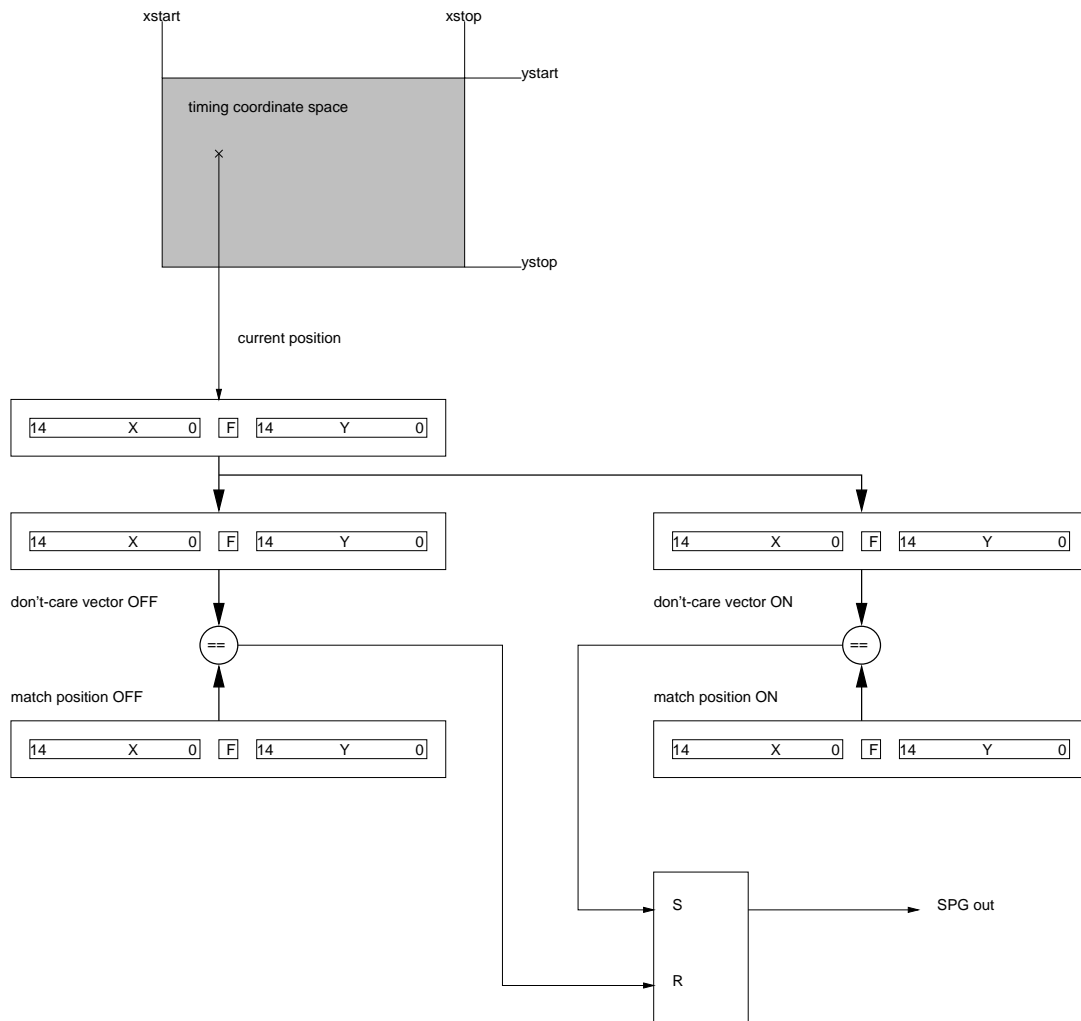
To achieve maximal flexibility, generation of sync signals is a three stage approach. In a first stage, signals are generated which carry positional timing information (according to the timing coordinates described in the section before). There are two methods to obtain these signals. The second stage combines them to form more complex waveforms. The third stage is used for a programmable delay of half a pixel clock cycle.

During display operation the sync generating components are provided with the current timing position as integer numbers in 2's complement representation.

### 3.10.2 Position Matching

One way to form first-stage signals is a simple position matching to trigger an RS-flip-flop. This is done by an array of six identical sync pulse generators (SPGs). Fig. 3-8 shows the working principle.





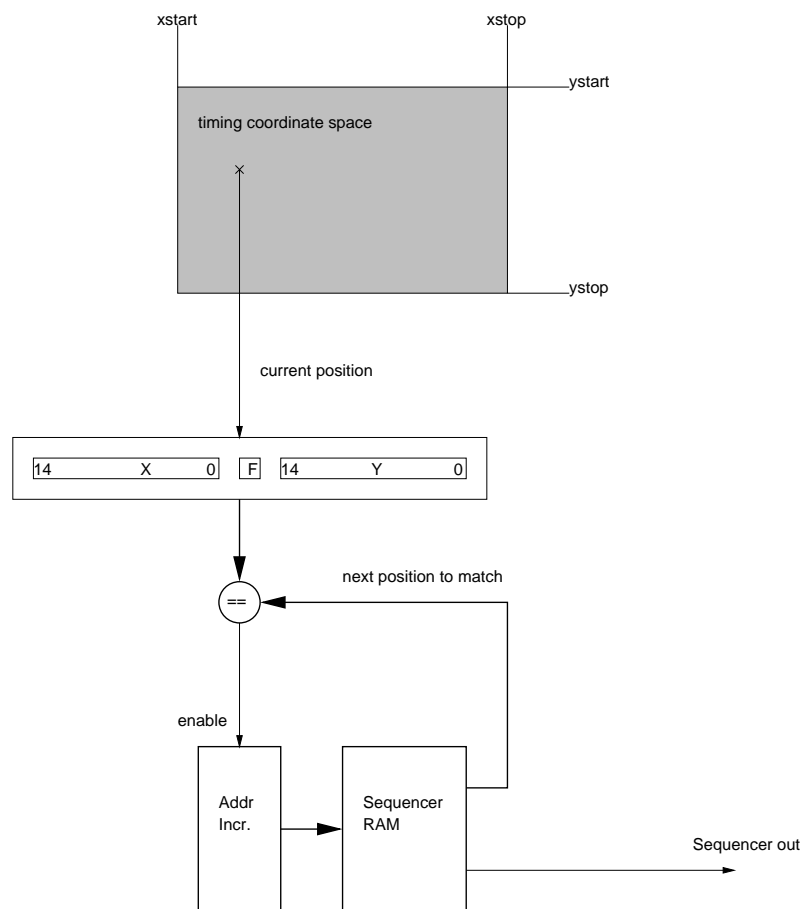
**Figure 3-8:** Matching positions with sync pulse generators.

The output of a sync pulse generator is set or reset if the current position equals the respective programmable position in all bits for which its don't-care-vector (which is also programmable) contains zeros. The Off-matching is dominant, i.e. when both, On and Off position are matched at the same time, the output of the sync pulse generator is reset.

### 3.10.3 Sequence Matching

A more sophisticated yet more powerful approach to form first-stage signals is the application of a sequencer RAM to match a whole sequence of positions. Fig. 3-9 shows the principle of operation.

There is one sync sequencer (SyncSeq) which is able to follow an arbitrary sequence of timing positions and generate an appropriate output signal. The length of the sequence as well as the contents of the RAM, consisting of position and the assigned output value are programmable.

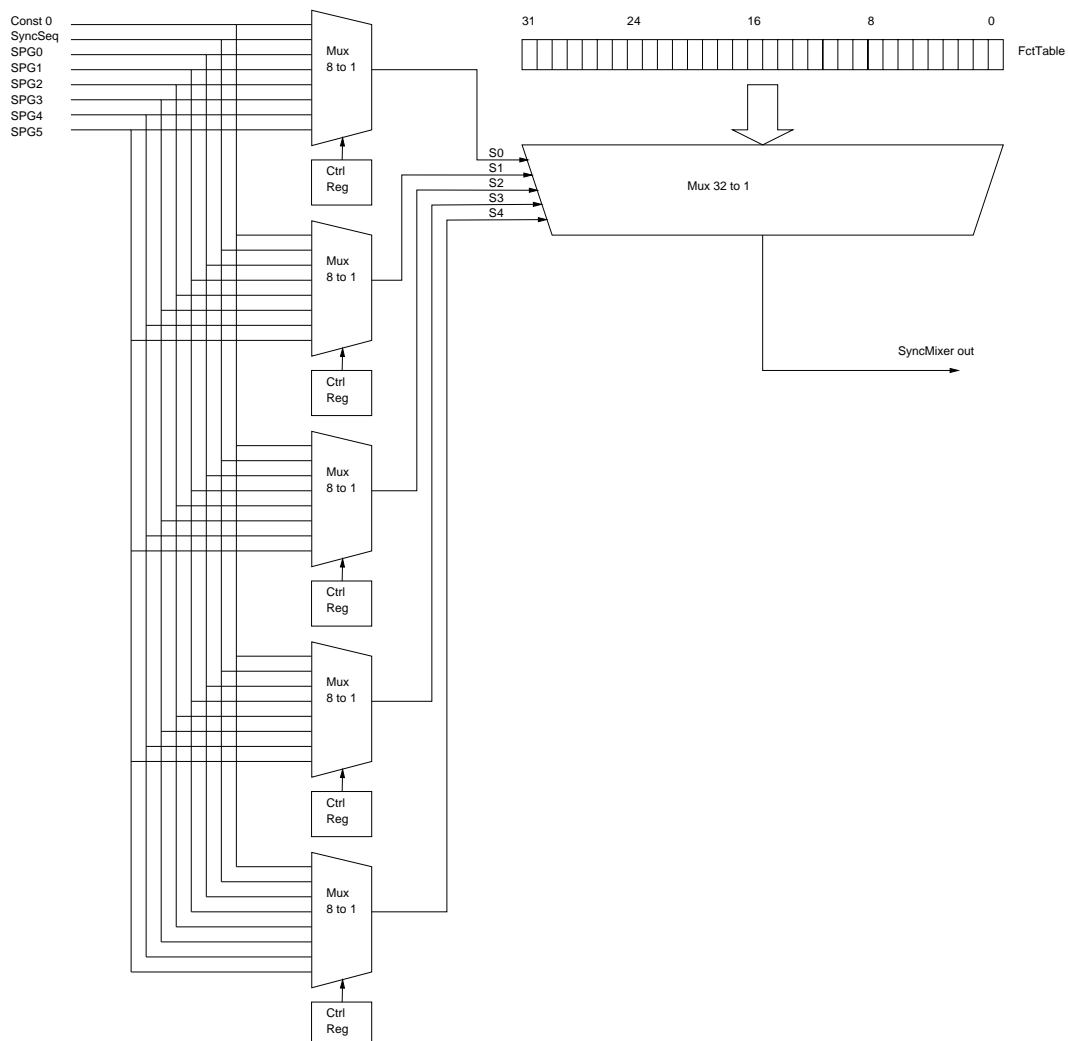


**Figure 3-9:** Matching whole sequences of positions with the Sync Sequencer.

The operation is as follows: At the begin, the address counter is reset to zero and the RAM outputs the first position to match and the output value for this position. If the comparator signals a match, the RAM address is incremented, the preset output value is propagated and the RAM now outputs the next position to match. This match/address increment cycle continues until the programmed sequence length is reached. If the last position is matched, the address counter is reset to zero again and the cycle starts anew. It is thus possible to generate arbitrarily complex waveforms with up to 64 edges (which is the maximum sequence length).

### 3.10.4 Combining First-Stage Sync Signals

As shown above, there are six sync pulse generator outputs and one sync sequencer output. To obtain more complex waveforms, these signals can be combined in a second stage. Here, an array of eight sync mixers (SMx) is used to calculate Boolean functions of first-stage signals. Each sync mixer can form any Boolean function of up to five inputs. The basic structure of one such mixer is depicted in fig. 3-10



**Figure 3-10:** Basic structure of a Sync Mixer. Each of the five address lines of the 32 to 1 multiplexer can be individually selected from any of the seven (plus one constant zero) first-stage signals. The output is the result of a table look-up. The register `FctTable` contains the truth table of the Boolean function calculated.

The concept of the sync mixers needs some explanation. In a first step the signals to be combined are selected. These are referred to then as  $S0...S4$  and form the address for the function table. This function table is used to look up the result of the Boolean operation the five selected signals shall be subject to.

An example may help understand the topic. Assuming the outputs of three Sync Pulse Generators shall form a combined signal with the function  $SMx = SyncSeq \wedge SPG0 \wedge \overline{SPG1}$ , one would proceed as follows. At first, the Sync Mixer signals  $S0...S4$  are assigned the Sync Pulse Generator outputs or constant zero by programming the respective multiplexers. The next step is to build the function's truth table, as shown in table 3-9. Since the intended function has only three inputs, only eight entries need be specified.

**Table 3-9:** Function table for the Sync Mixer example.

Selected First-stage Signals					Desired Output
S4 = 0	S3 = 0	S2 = SPG1	S1 = SPG0	S0 = SyncSeq	SMx = f(S0....S4)
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	0
Combinations [S4...S0] = 10000...11111 can never occur since S4 and S3 are selected constantly zero					need not be specified

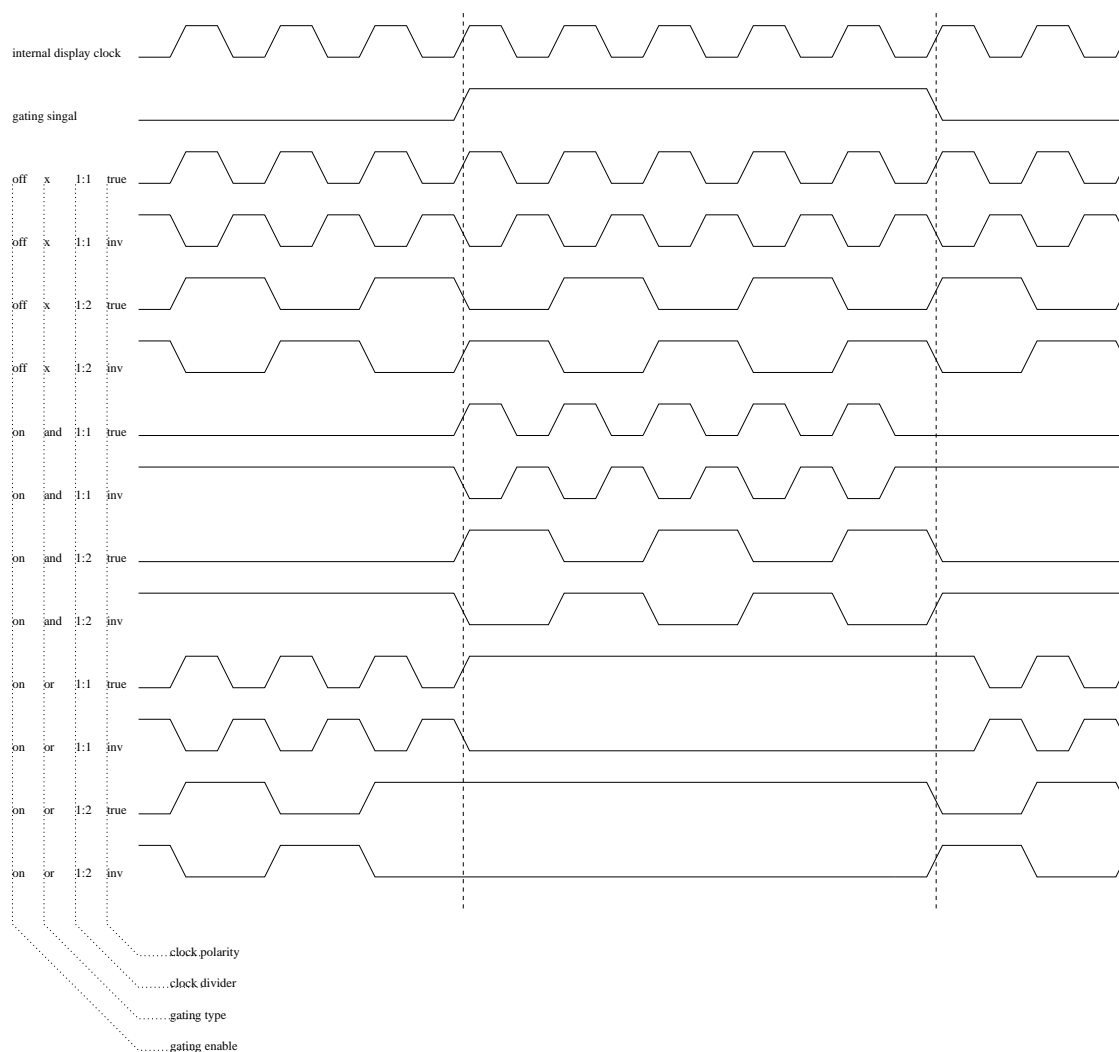
It is recommended that S4...S0 are listed in order of binary number representation. This allows to take the function result row immediately as register contents for the Sync Mixer function table, i.e. the last row is interpreted as binary 32 bit number with the LSB in the first row and the MSB in the last. For the example this would be [xxxx xxxx xxxx xxxx xxxx xxxx 0000 1000] binary, with x's denoting arbitrarily set or reset bits, since these will never be read out of the function table.

### 3.10.5 Sync Signal Delay Adjustment

Before the outputs of the eight Sync Mixers are connected to actual GDC pins, they are fed through a programmable delay stage. This allows the signals either to be left untouched or delayed for half a pixel clock cycle. This delay can be set for each of the eight Sync Mixer output signals individually with the Sync Switch register.

## 3.11 Pixel Clock Gating

The GPU allows to flexibly shape the pixel clock output at the respective pin. This is done with a programmable control register that defines how the output of Sync Mixer 7 affects the output pixel clock. Fig. 3-11 shows the resulting waveforms according to gate control settings. It should be mentioned that the control by the mixer output is the only way to determine the position of clock edges when using the divided clock. If the output clock is not gated, but divided, then the first rising clock edge occurs exactly at the same time with the first rising edge of the pure internal pixel clock after reset.



## 3.12 Numerical Mnemonic Definitions

### 3.12.1 Color Space Code

This code is intended to designate color resolutions in the different color spaces uniquely. It provides information on the current color space as well as for transformations between these spaces.

**Table 3-10:** Color space code definitions

Code	Color Space		Code	Color Space	
	Mnemonic	Bits occupied		Mnemonic	Bits occupied
0	1bpp	1	8	YUV444	32
1	2bpp	2	9	RGB111	2
2	4bpp	4	10	RGB222	6
3	8bpp	8	11	RGB333	9

**Table 3-10:** Color space code definitions

Code	Color Space		Code	Color Space	
	Mnemonic	Bits occupied		Mnemonic	Bits occupied
4	RGB555	16	12	RGB444	12
5	RGB565	16	13	RGB666	18
6	RGB888	32	14	YUV555 <sup>a</sup>	16
7	YUV422	16	15	YUV655 <sup>a</sup>	16

a. This format is only available for MB87P2020-A (Jasmine).

### 3.12.2 Bit Stream Format Code

This code describes number of bits actually written to the display (some displays require more than one pixel written at once).

**Table 3-11:** Bit stream format code definitions

Code	Bit stream		Code	Bit stream	
	Mnemonic	Bits written		Mnemonic	Bits written
0	S1	1	6	S12	12
1	S2	2	7	S18	18
2	S4	4	8	S24	24
3	S6	6	9	AN	analog output
4	S8	8	10-15	reserved	
5	S9	9			

### 3.12.3 Scan Mode Code

This code determines the scan mode of the display, i.e. the number and layout of pixel streams written in parallel to the physical display.

**Table 3-12:** Scan mode definitions.

Code	0	2	3	1
Mnemonic	SglScan	DualScan	Zzagscan	illegal

### 3.13 Bit to Color Channel Assignment

Table 3-13 gives the assignment of the bits to color channels (RGB or YUV) corresponding to bit width (i.e. the number of bits occupied by one pixel) as used GPU internal for chromatic intermediate and physical color spaces. This assignment is important for direct mapping (to determine which GDC pins carry the desired signal) and for CLUT and gamma table loading.

**Table 3-13:** Bit to RGB/YUV assignment for chromatic color spaces.

Color space	Bits assigned to					
	R	G	B	Y	U(Cb)	V(Cr)
RGB111 <sup>a</sup>	[2]	[1]	[0]			
RGB111 <sup>b</sup>	[0]	[1]	[2]			
RGB222	[5:4]	[3:2]	[1:0]			
8bpp <sup>c</sup>	[7:5]	[4:2]	[1:0]			
RGB333	[8:6]	[5:3]	[2:0]			
RGB444	[11:8]	[7:4]	[3:0]			
RGB555	[15:11]	[10:6]	[4:0]			
RGB565	[15:11]	[10:5]	[4:0]			
RGB666	[17:12]	[11:6]	[5:0]			
RGB888	[23:16]	[15:8]	[7:0]			
YUV422 <sup>d</sup>				[15:8]	[7:0]	
YUV444				[15:8]	[7:0]	[23:16]
YUV555 <sup>e</sup>				[10:6]	[4:0]	[15:11]
YUV655 <sup>e</sup>				[10:5]	[4:0]	[15:11]

a. Default assignment with no R/B swap

b. Alternative assignment with active R/B swap

c. This assignment is valid only when 8bpp is mapped aligned to intermediate RGB333

d. Bits [7:0] contain U and V alternating for every other pixel.

e. This format is only available for MB87P2020-A (Jasmine).

## 3.14 GPU Signal to GDC Pin Assignment

### 3.14.1 Multi-Purpose Digital Signals

Some GDC pins are shared by GPU signals, since pins are a limited package resource. This sharing depends on the physical display connected to GDC. Table 3-14 shows the assignments.

**Table 3-14:** GPU multi-purpose digital signal to GDC pin assignment. Pixel data designates actual colors or gray scales as well as the blanking clamp value.

GDC pin	GPU signal usage
<i>a) Display with scan mode SglScan</i>	
DIS_D[23:19]	Digital pixel data [23:19] when bit stream format is either S24, or AN and twin display mode active (digital display is secondary), otherwise delay adjusted outputs of Sync Mixer 7...3
DIS_D[18:0]	Digital pixel data
<i>b) Display with scan mode DualScan</i>	
DIS_D[23:19]	Delay adjusted outputs of Sync Mixer 7...3
DIS_D[18:16]	Constant zero
DIS_D[15:8]	Lower display segment digital pixel data
DIS_D[7:0]	Upper display segment digital pixel data
<i>c) Display with scan mode ZzagScan</i>	
DIS_D[23:19]	Delay adjusted outputs of Sync Mixer 7...3
DIS_D[18:8]	Constant zero
DIS_D[7:0]	Digital pixel data
<b>Note 1:</b> For cases with DIS_D[23:19] being not connected to the delay adjusted outputs of Sync Mixers 3 to 7 these signals are not available at GDC pins. <b>Note 2:</b> Independent of the GPU signals being connected to the pins their output is further controlled by the respective output enable settings.	
DIS_PIXCLK	When used as output the pin carries the pixel clock as formed due to gating and divider settings (cf. 3.11)

### 3.14.2 Analog Pixel Data

If scan mode is set to SglScan and bit stream format is AN (analog display is primary) or twin display mode (Jasmine only) is active and bit stream format is S9...S24 (analog display is secondary), pins A\_RED, A\_GREEN, and A\_BLUE carry analog pixel data, i.e. colors or gray scales as well as the blanking clamp value, otherwise they are high-Z. They are high-Z as well when the DAC output enable bit is reset.

### 3.14.3 Dedicated Sync Signals

Three pins for dedicated sync signals are available. Table 3-15 gives the assignment.

**Table 3-15:** GPU sync signal to GDC pin assignment.

GDC pin	GPU signal usage
DIS_HSYNC	This pin is always connected to the delay adjusted output of Sync Mixer 0.



**Table 3-15:** GPU sync signal to GDC pin assignment.

GDC pin	GPU signal usage
DIS_VSYNC	This pin is always connected to the delay adjusted output of Sync Mixer 1.
DIS_VREF	This pin is always connected to the delay adjusted output of Sync Mixer 2.
<b>Note:</b> Although the pin names seem to imply a certain signal function this is not the case. In fact, any of the pins above can output any arbitrary waveform generated with the respective Sync Mixer.	

### 3.14.4 Sync Mixer connections

Table 3-16 shows the mapping of sync mixer outputs to GDC pins or its internal connections.

**Table 3-16:** Sync mixer connections

Sync mixer	GDC pin	Internal connection
Sync mixer 0	DIS_HSYNC	
Sync mixer 1	DIS_VSYNC	
Sync mixer 2	DIS_VREF	
Sync mixer 3	DIS_D[19]	
Sync mixer 4	DIS_D[20]	
Sync mixer 5	DIS_D[21]	CCFL synchronization (MB87P2020-A only)
Sync mixer 6	DIS_D[22]	Flag FLNOM_GSYNC (MB87P2020(-A) only)
Sync mixer 7	DIS_D[23]	Clock gating

Sync mixer 7 is used for pixel clock gating as already described in chapter 3.11.

Sync mixer 6 is connected to the flag FLNOM\_GSYNC which can be used by an application to trigger on an event regarding display output. For example with help of this flag an application can synchronize its drawing on display frame rate. It is also possible to generate a MCU interrupt with this flag (see ULB description for details about flags and interrupt handling).

Note that this connection is only available for MB87P2020 and MB87P2020-A.

Sync mixer 5 can be taken as synchronization input for the CCFL driver. This allows a flexible CCFL flash rate synchronization with display output refresh rate. See CCFL description for further details about synchronization settings.

Note that this connection is only available for MB87P2020-A.

Internal connections for sync mixers are always established independent from display settings.

### 3.14.5 Color Key Output

The GDC pin DIS\_CK carries the output of the color key unit. It is high-Z when the respective output enable bit is reset.

#### Behaviour:

The output becomes active with a user-controlled polarity when data for visible pixels is output that lies within limits that are also defined by the user. It is inactive during blanking periods. The behaviour further depends on physical color space / bit stream format combinations:

- For combinations that lead to an output of one pixel per pixel clock, the output is updated on a per-pixel basis.

- For combinations with more than one pixel per pixel clock cycle (either all complete pixels or with partial remainders, i.e. not all bits of one pixel are included) the output corresponds to the geometrically leftmost complete pixel.
- For combinations with pixel clock cycles for which no complete pixel is output (instead remaining bits from the previous pixel clock and some bits of the next pixel) the output corresponds to the geometrically leftmost complete previous pixel.

The matching is done separately for each color channel. A pixel matches the key when the following condition holds:

$$Limit_{Red, Lower} \leq Pixel_{Red} \leq Limit_{Red, Upper} \text{ AND}$$

$$Limit_{Green, Lower} \leq Pixel_{Green} \leq Limit_{Green, Upper} \text{ AND}$$

$$Limit_{Blue / Mono, Lower} \leq Pixel_{Blue / Mono} \leq Limit_{Blue / Mono, Upper}$$

The pixel's color channels are extracted automatically corresponding to the physical color space.

## 4 GPU Register Set

### 4.1 Description

Addresses are hexadecimal byte addresses in MCU address space, initial values are given as mnemonic or hexadecimal, as appropriate. Bit slices are according to the *word* starting at address *Addr*. Registers marked in *Lock* are read-only during DIR\_MTimingOn set to 1.

**Table 4-1:** GPU Register Set (*continued*).

Mnemonic (API-Names)	Addr	Lock	Function	Initial Value
<b>Layer description record information</b>				
<i>LDR0_PhAddr ...</i> <i>LDR15_PhAddr</i> PHA(i)	1000 ... 103C		Layer 0 to Layer 15 physical address in VideoRAM (These are actually SDC registers, included here for completeness of layer description records only.)	undef.
<i>LDR0_DomSz ...</i> <i>LDR15_DomSz</i> DSZ_X(i)	1040 ... 107C		Layer 0 to Layer 15 domain size, i.e. size of whole area covered by the layer [29:16] = DX = width (unsigned integer) (These are actually SDC registers, included here for completeness of layer description records only. There is no DY stored, it is the user's responsibility to provide sufficient RAM space for each layer.)	undef.
<i>LDR0_1stPxl ...</i> <i>LDR15_1stPxl</i> DP1_X(i) DP1_Y(i)	1080 ... 10BC		Layer 0 to Layer 15 first pixel to be displayed, i.e. offset within logical layer domain [29:16] = FX (unsigned integer) [13:0] = FY (unsigned integer)	undef.
<i>LDR0_WinSz ...</i> <i>LDR15_WinSz</i> WSZ_X(i) WSZ_Y(i)	10C0 ... 10FC		Layer 0 to Layer 15 window size, i.e. size of visible area [29:16] = WX = width (unsigned integer) [13:0] = WY = height (unsigned integer)	undef.
<i>LDR0_Offs ...</i> <i>LDR15_Offs</i> WOF_X(i) WOF_Y(i)	1100 ... 113C		Layer 0 to Layer 15 display offset, i.e. offset from upper left corner of physical display [29:16] = OX (signed integer) [13:0] = OY (signed integer)	undef.
<i>LDR0_TrCol ...</i> <i>LDR15_TrCol</i> LTC(i)	1140 ... 117C		Layer 0 to Layer 15 transparent color, i.e. color for which a pixel is not to be displayed (= transparent) [23:0] = color (according to color space)	undef.
<i>LDR0_BlCol ...</i> <i>LDR15_BlCol</i> LBC(i)	1180 ... 11BC		Layer 0 to Layer 15 blink color, i.e. color for which a pixel is displayed either in this or blink alternative color [23:0] = color (according to color space)	undef.
<i>LDR0_BlAlt ...</i> <i>LDR15_BlAlt</i> LAC(i)	11C0 ... 11FC		Layer 0 to Layer 15 blink alternative color [23:0] = color (according to color space)	undef.

**Table 4-1:** GPU Register Set (*continued*).

Mnemonic (API-Names)	Addr	Lock	Function	Initial Value
<i>LDR0_BIRate ...</i> <i>LDR15_BIRate ...</i> LBR_OFF(i) LBR_ON(i)	1200 ... 123C		Layer 0 to Layer 15 blink rate definitions [15:8] = (#-1) of frames for blink alternative color [7:0] = (#-1) of frames for blink color	undef.
<i>LDR0_CSpace ...</i> <i>LDR15_CSpace ...</i> CSPC_CSC(i) CSPC_TE(i) CSPC_LDE(i)	1240 ... 127C		Layer 0 to Layer 15 color space & format definitions [3:0] = color space code [8] = transparency enable [9] = line doubling enable (for video field/frame conversion)	undef.
<b>Merging description record information</b>				
<i>MDR_Format</i> CFORMAT_CSC CFORMAT_GFORCE CFORMAT_IPOLEN CFORMAT_GAMEN CFORMAT_LITC	1300		Intermediate color space & conversion definition [3:0] = intermediate color space [5] = RGB gamma forcing (valid only when gamma table enable is set) [6] = YUV422 chroma interpolation enable [7] = gamma table enable [31:16] = layer to intermediate transfer codes (0 = normal transfer, 1 = alternative transfer)	1bpp off  off off 0000
<i>MDR_Bgnd</i> BACKCOL_COL BACKCOL_EN	1304		Background color, i.e. color to display for pixels not covered by any layer [23:0] = color (according to intermediate color space) [24] = background enable	0000 0
<i>MDR_BlinkCtrl</i> MBC_EN MBC_CBS	1308		Blink enable bits & blink state read-back [15:0] = blink enable (one bit per layer) [31:16] = current blink state (read-only)	0000
<i>MDR_ZOrder</i> ZORDER_EN0 ZORDER_EN1 ZORDER_EN2 ZORDER_EN3 ZORDER_TM0 ZORDER_TM1 ZORDER_TM2 ZORDER_TM3	130C		Z-order register, i.e. four places to describe stacking of layers [3:0] = # of downmost layer (plane 0) [7:4] = # of next layer (plane 1) [11:8] = # of top but one layer (plane 2) [15:12] = # of topmost layer (plane 3) [16] = enable plane 0 [17] = enable plane 1 [18] = enable plane 2 [19] = enable plane 3	0 0 0 0 off off off off
<i>MDR_ClutOffs0 ...</i> <i>MDR_ClutOffs15 ...</i> CLUTOF(i)	1340 ... 137C		Table of 16 color table offsets [8:0] = offset into look-up table for layer <i>i</i>	undef.
<i>MDR_DRM0 ...</i> <i>MDR_DRM13 ...</i> DRM(i)	1380 ... 13B4		table of 14 definitions for pseudo gray levels [5:0] = duty ratio definition	undef.
<b>YUV to RGB Matrix Parameters (Jasmine only)</b>				

Table 4-1: GPU Register Set (*continued*).

Mnemonic (API-Names)	Addr	Lock	Function	Initial Value
<i>MTX_PreBias</i> PREBIAS_Y PREBIAS_CB PREBIAS_CR	1400		Pre matrix bias values [23:16] = PreBiasY [15:8] = PreBiasCb [7:0] = PreBiasCr	F0 80 80
<i>MTX_CoeffR</i> COEFFR_YXR COEFFR_CBXR COEFFR_CRXR	1404		Matrix coefficients to red channel [23:16] = YxR [15:8] = CBxR [7:0] = CRxR	80 0 B3
<i>MTX_CoeffG</i> COEFFG_YXG COEFFG_CBXG COEFFG_CRXG	1408		Matrix coefficients to green channel [23:16] = YxG [15:8] = CBxG [7:0] = CRxG	80 2C 5B
<i>MTX_CoeffB</i> COEFFB_YXB COEFFB_CBXB COEFFB_CRXB	140C		Matrix coefficients to blue channel [23:16] = YxB [15:8] = CBxB [7:0] = CRxB	80 E2 0
<i>CLUT</i> CLUT(i)	2000 ... 27FC		Color look-up table with 512 (Jasmine) or 256 (Lavender) entries [23:0] = color (according to intermediate color space)	undef.
<b>Gamma correction tables (Jasmine only)</b>				
<i>Gamma</i> GAMMA_R GAMMA_G GAMMA_B	2800 ... 2BFC		Look-up tables for inverse gamma correction [23:16] = R [15:8] = G [7:0] = B	undef undef undef
<b>Display interface record information</b>				
<i>DIR_PhSize</i> PHSIZE_X PHSIZE_Y	3000	x	Display physical size, i.e. visible pixel area [29:16] = PX = width [13:0] = PY = height	0 0
<i>DIR_Format</i> PHFRM_CSC PHFRM_RBSW PHFRM_BSC PHFRM_FTE PHFRM_SM PHFRM_TDM PHFRM_IES PHFRM_HSYAE PHFRM_VSYAE PHFRM_POL	3004	x	Output format definitions [3:0] = physical color space [4] = R/B swap [11:8] = bit stream format code [12] = field toggle enable (distinguishes between odd & even fields) [17:16] = scan mode [18] = twin display mode (1 = enable) [24] = int./ext. sync (0 = int. sync, i.e. hsync,vsync, vref are outputs) [25] = xhsync active edge (0 = low/high edge) [26] = xvsync active edge [26] [27] = xfref polarity (0 = odd field ref. is low active)	1bpp off S1 off SglSca n off int 0 0 0

**Table 4-1:** GPU Register Set (*continued*).

Mnemonic (API-Names)	Addr	Lock	Function	Initial Value
<i>DIR_PXScanDots</i> PHSCAN_SCLK	3008	x	Physical size in scan dots (PX*BitsPerPixel/BitsPerScanlock) [29:16] = SX	0
<i>DIR_DualScanOffs</i> DUALSCOF	300C	x	Dual scan Y offset (usually PY/2) [13:0] = offset	0
<i>DIR_MTimOddStart</i> MTIMODD_X(0) MTIMODD_Y(0)	3010	x	Master timing: odd/only field start (in scan clock cycles, with respect to visible area) [30:16] = x (2's complement) [14:0] = y (2's complement)	0 0
<i>DIR_MTimOddStop</i> MTIMODD_X(1) MTIMODD_Y(1)	3014	x	Master timing: odd/only field stop (in scan clock cycles, with respect to visible area) [30:16] = x (2's complement) [14:0] = y (2's complement)	0 0
<i>DIR_MTimEvenStart</i> MTIMEVEN_Y(0)	3018	x	Master timing: even field start (in scan clock cycles, with respect to visible area) [14:0] = y (2's complement)	0
<i>DIR_MTimEvenStop</i> MTIMEVEN_Y(1)	301C	x	Master timing: even field stop (in scan clock cycles, with respect to visible area) [14:0] = y (2's complement)	0
<i>DIR_MTimingOn</i> MTIMON	3020		Master timing switch [0] = switch (0 = off, 1 = on)	off
<i>DIR_TimingDiag</i> TIMDIAG_X TIMDIAG_FIELD TIMDIAG_Y	3024		Diagnostic register, contains current timing position	read-only
<i>DIR_SPG0PosOn</i> SPGPSON_X(i) SPGPSON_FIELD(i) SPGPSON_Y(i)	3030		Sync pulse generator 0, "switch-on" position [30:16] = X scan position (2's complement) [14:0] = Y scan position (2's complement) [15] = field (0=odd, 1=even field)	0 0 0
<i>DIR_SPG0MaskOn</i> SPGMKON_X(i) SPGMKON_FIELD(i) SPGMKON_Y(i)	3034		Sync pulse generator 0, "switch-on" don't care vector [30:0] = mask bits (1= do not include this bit into position matching)	0
<i>DIR_SPG0PosOff</i> SPGPSOF_X(i) SPGPSO_FIELD(i) SPGPSO_Y(i)	3038		Sync pulse generator 0, "switch-off" position [30:16] = X scan position (2's complement) [14:0] = Y scan position (2's complement) [15] = field (0 = odd, 1 = even field)	0 0 0
<i>DIR_SPG0MaskOff</i> SPGMKOF_X(i) SPGMKOF_FIELD(i) SPGMKOF_Y(i)	303C		Sync pulse generator 0, "switch-off" don't care vector [30:0] = mask bits (1= do not include this bit into position matching)	0
<i>DIR_SPG1PosOn</i>	3040		Sync pulse generator 1, "switch-on" position (cf. DIR_SPG0PosOn)	

**Table 4-1:** GPU Register Set (*continued*).

<b>Mnemonic (API-Names)</b>	<b>Addr</b>	<b>Lock</b>	<b>Function</b>	<b>Initial Value</b>
<i>DIR_SPG1MaskOn</i>	3044		Sync pulse generator 1, “switch-on” don’t care vector (cf. DIR_SPG0PosOn)	
<i>DIR_SPG1PosOff</i>	3048		Sync pulse generator 1, “switch-off” position (cf. DIR_SPG0PosOn)	
<i>DIR_SPG1MaskOff</i>	304C		Sync pulse generator 1, “switch-off” don’t care vector (cf. DIR_SPG0PosOn)	
<i>DIR_SPG2PosOn</i>	3050		Sync pulse generator 2, “switch-on” position (cf. DIR_SPG0PosOn)	
<i>DIR_SPG2MaskOn</i>	3054		Sync pulse generator 2, “switch-on” don’t care vector (cf. DIR_SPG0PosOn)	
<i>DIR_SPG2PosOff</i>	3058		Sync pulse generator 2, “switch-off” position (cf. DIR_SPG0PosOn)	
<i>DIR_SPG2MaskOff</i>	305C		Sync pulse generator 2, “switch-off” don’t care vector (cf. DIR_SPG0PosOn)	
<i>DIR_SPG3PosOn</i>	3060		Sync pulse generator 3, “switch-on” position (cf. DIR_SPG0PosOn)	
<i>DIR_SPG3MaskOn</i>	3064		Sync pulse generator 3, “switch-on” don’t care vector (cf. DIR_SPG0PosOn)	
<i>DIR_SPG3PosOff</i>	3068		Sync pulse generator 3, “switch-off” position (cf. DIR_SPG0PosOn)	
<i>DIR_SPG3MaskOff</i>	306C		Sync pulse generator 3, “switch-off” don’t care vector (cf. DIR_SPG0PosOn)	
<i>DIR_SPG4PosOn</i>	3070		Sync pulse generator 4, “switch-on” position (cf. DIR_SPG0PosOn)	
<i>DIR_SPG4MaskOn</i>	3074		Sync pulse generator 4, “switch-on” don’t care vector (cf. DIR_SPG0PosOn)	
<i>DIR_SPG4PosOff</i>	3078		Sync pulse generator 4, “switch-off” position (cf. DIR_SPG0PosOn)	
<i>DIR_SPG4MaskOff</i>	307C		Sync pulse generator 4, “switch-off” don’t care vector (cf. DIR_SPG0PosOn)	
<i>DIR_SPG5PosOn</i>	3080		Sync pulse generator 5, “switch-on” position (cf. DIR_SPG0PosOn)	
<i>DIR_SPG5MaskOn</i>	3084		Sync pulse generator 5, “switch-on” don’t care vector (cf. DIR_SPG0PosOn)	
<i>DIR_SPG5PosOff</i>	3088		Sync pulse generator 5, “switch-off” position (cf. DIR_SPG0PosOn)	
<i>DIR_SPG5MaskOff</i>	308C		Sync pulse generator 5, “switch-off” don’t care vector (cf. DIR_SPG0PosOn)	
<i>DIR_SSqCycle</i> SSQCYLE	30FC		Sequencer cycle length [4:0] = (#-1) of sequencer cycles	0

Table 4-1: GPU Register Set (*continued*).

Mnemonic (API-Names)	Addr	Lock	Function	Initial Value
<i>DIR_SSqCnts</i> SSQCNTS_OUT(i) SSQCNTS_SEQX(i) SSQCNTS_FIELD(i) SSQCNTS_SEQY(i)	3100 ... 31FC		Sequencer position definitions [30:16] = X scan position (2's complement) [14:0] = Y scan position (2's complement) [15] = field (0=odd, 1=even field) [31] = output value, when position is reached	undef.
<i>DIR_SMx0Sigs</i> SMXSIGS_S4(i) SMXSIGS_S3(i) SMXSIGS_S2(i) SMXSIGS_S1(i) SMXSIGS_S0(i)	3200		Sync mixer 0 signal select [14:12] = s4 [11:9] = s3 [8:6] = s2 [5:3] = s1 [2:0] = s0 s <sub>i</sub> == 0: const. zero s <sub>i</sub> == 1 sync sequencer output s <sub>i</sub> == 2...7 sync pulse generator 0...5 output	0 0 0 0 0
<i>DIR_SMx0FctTable</i> SMXFCT(i)	3204		Sync mixer 0 function table [31:0] = output value mixer output = function table [a] $a = s_4 * 2^4 + s_3 * 2^3 + s_2 * 2^2 + s_1 * 2^1 + s_0 * 2^0$	0
<i>DIR_SMx1Sigs</i>	3208		Sync mixer 1 signal select (cf. DIR_SMx0Sigs)	
<i>DIR_SMx1FctTable</i>	320C		Sync mixer 1 function table (cf. DIR_SMx0FctTable)	
<i>DIR_SMx2Sigs</i>	3210		Sync mixer 2 signal select (cf. DIR_SMx0Sigs)	
<i>DIR_SMx2FctTable</i>	3214		Sync mixer 2 function table (cf. DIR_SMx0FctTable)	
<i>DIR_SMx3Sigs</i>	3218		Sync mixer 3 signal select (cf. DIR_SMx0Sigs)	
<i>DIR_SMx3FctTable</i>	321C		Sync mixer 3 function table (cf. DIR_SMx0FctTable)	
<i>DIR_SMx4Sigs</i>	3220		Sync mixer 4 signal select (cf. DIR_SMx0Sigs)	
<i>DIR_SMx4FctTable</i>	3224		Sync mixer 4 function table (cf. DIR_SMx0FctTable)	
<i>DIR_SMx5Sigs</i>	3228		Sync mixer 5 signal select (cf. DIR_SMx0Sigs)	
<i>DIR_SMx5FctTable</i>	322C		Sync mixer 5 function table (cf. DIR_SMx0FctTable)	
<i>DIR_SMx6Sigs</i>	3230		Sync mixer 6 signal select (cf. DIR_SMx0Sigs)	
<i>DIR_SMx6FctTable</i>	3234		Sync mixer 6 function table (cf. DIR_SMx0FctTable)	
<i>DIR_SMx7Sigs</i>	3238		Sync mixer 7 signal select (cf. DIR_SMx0Sigs)	
<i>DIR_SMx7FctTable</i>	323C		Sync mixer 7 function table (cf. DIR_SMx0FctTable)	
<i>DIR_SSwitch</i> SSWITCH	3240	x	Sync switch [7:0] = delay select (0=no, 1=0.5 cycle delay)	0



Table 4-1: GPU Register Set (*continued*).

Mnemonic (API-Names)	Addr	Lock	Function	Initial Value
<i>DIR_PixClkGate</i> PIXCLKGT_GT PIXCLKGT_GON PIXCLKGT_CP PIXCLKGT_HC	3248	x	Pixel clock gate control (by output of sync mixer 7) [0] = gate type (0=AND, 1=OR) [1] = gate enable (0=off) [2] = clock polarity (0=true, 1=inverted) [3] = divider (0=1:1, 1=1:2)	0 0 0 0
<i>DIR_CKlow</i> CKLOW_LLRL CKLOW_LLGL CKLOW_LLBL CKLOW_OPL	3250		Color key lower limits (according to physical color space) [23:16] = red channel [15:8] = green channel [7:0] = blue/monochrome channel [24] = key out polarity (0=active high, 1=active low)	0 0 0 0
<i>DIR_CKup</i> CKUP_ULRL CKUP_ULGL CKUP_ULBL	3254		Color key upper limit (according to physical color space) [23:16] = red channel [15:8] = green channel [7:0] = blue/monochrome channel Pin xo_ckey is activated, when all pixel channels lie within (including limits) their respective limits	0 0 0
<i>DIR_AClamp</i> ACLAMP_ACLRL ACLAMP_ACLGL ACLAMP_ACLBL	3258		Jasmine: Clamping value for analog outputs Lavender: Clamping value for analog and digital output [23:16] = red channel [15:8] = green channel [7:0] = blue channel	0 0 0
<i>DIR_DClamp</i> DCLAMP	325C		Clamping value for digital outputs (Jasmine only) [23:0] = value output during blanking	0
<i>DIR_MainEnable</i> MAINEN_DOEL MAINEN_HSOEL MAINEN_VSOEL MAINEN_VROEL MAINEN_CKOE MAINEN_DACOE MAINEN_REFOEL	3260		Main display output enable flags [23:0] = digital data output enable [24] = hsync output enable (when int. sync) [25] = vsync output enable (when int. sync) [26] = vref output enable (when int. sync) [27] = ckey output enable [28] = DACs output enable [29] = reference voltage enable	0 0 0 0 0 0 0 0
<b>SDRAM Controller Interaction</b>				
<i>GPU_SDCPrio</i> SDCP_LP SDCP_HP SDCP_IFL	3270		SDRAM Controller request priorities [2:0] = low priority value (not used) [6:4] = high priority value [15:8] = input FIFO load (read-only)	3 7

## 4.2 Determination of Register Contents

### 4.2.1 Values Derived from Display Specs

#### Resolution and Formats

The display spec states the number of physically visible pixels in each direction. These numbers are the values used in `DIR_PhSize`, i.e. the number of visible pixels per line is the `PX` value, the number of lines the `PY` value.

The code for physical color space is derived from the number of bits per pixel (color resolution). After physical color space is defined, intermediate color space has to be chosen from the legal mappings, given in table 2-2, section 2.4.

The number of pixels the display expects per pixel clock cycle (or the number of bits expected per pixel clock in relation to the color resolution, as actually stated), together with the display interface type (digital or analog) determine the bit stream format.

Once derived, the values for `PX` and bit stream format are used to determine the physical size in scan dots, i.e. `PXScanDots`. This value is calculated this way:  $PXScanDots = \left\lfloor \frac{PX \times BitsPerPixel}{BitsPerScanClock} \right\rfloor$ . That

means that the value of `PXScanDots` is rounded towards the nearest integer that is less than or equal to the fractional value. This does *not* prevent a fractional number of pixels per scan clock being output as required by some displays.

#### Timing and Sync-Signals

After entry of resolution and formats timing is the next item to specify. This has to be done in accordance to the pixel clock set up in the Clock Unit (CU).

As was elaborated in section 3.9, all timing is described in terms of the timing coordinate space. The first task is therefore to establish its basic unit, i.e. the time needed per scan dot. This value can be obtained either directly from the display's spec sheet or need to be calculated from other values given there. Due to the fact that only discrete pixel clock rates are possible, this unit time has to be rounded to the nearest value allowed by the actual pixel clock. Hence,  $t_{ScanDot} \approx 1 / PixelClock$

Now the values for `xstart` and `xstop` (bits [30:16], 2's complement, addresses 0x3010 and 0x3014) can be calculated. The first of these contains the start time of a line in relation to the begin of active display (i.e. the first visible pixel) expressed in timing coordinates, i.e. the number of pixel clock cycles. This time is obtained from the display's spec sheet either as time or number of clocks. If given as time, the value is determined this way:  $xstart = (t_{LineStart} - t_{ActiveDisplay}) / t_{ScanDot}$ . Please note that for leading blanking this value is negative, so if the spec sheet gives the number of clock cycles directly, it has to be entered as negative number as well.

Depending on how the length of a line is stated in the display spec, the `xstop` value can be obtained in two ways:

from a time value:  $xstop = xstart + (TimePerLine / t_{ScanDot}) - 1$

from the number of clock cycles:  $xstop = xstart + CyclesPerLine - 1$

The corresponding `ystart` and `ystop` values are calculated in a similar manner. If no TV-conform timing is needed then the respective "odd" values are sufficient (bits [14:0], 2's complement, addresses 0x3010 and 0x3014). The value for `ystart` is basically the number of leading blanking lines entered as negative value (cf. note for `xstart`). If only a period of time for leading blanking is given in the spec the value can be calculated as follows:

$$\begin{aligned} ystart &= VertLeadBlank / TimePerLine \\ &= VertLeadBlank / ((xstop - xstart + 1) / PixelClock) \end{aligned}$$

Depending on whether the global number of display lines (active and blanking) or the frame period is given in the spec, the `ystop` value can be calculated these ways:

using a global number of lines:  $ystop = NumberOfLines + ystart - 1$

using the frame period:  $ystop = (T_{Frame} / TimePerLine) + ystart - 1$

---

In case of TV-conform timing being required two different pairs of `ystart` and `ystop` values are needed, referred to as “odd” and “even”. They will be used alternating every other frame. Usually, the `ystart` values will be the same, whilst the `ystop` values differ by one. Thus it is possible to achieve an odd number of lines for two consecutive frames.

**Note:** The complete entry of all necessary start and stop values is indispensable regardless of whether internal or external generation of sync signals is specified.

After determination of start and stop values sync signals can be specified as obtained from the display spec if internal generation of sync signals is requested. For the majority of cases usage of sync pulse generators as described in 3.10.2 should be sufficient. The basic task for their application is to determine the values for start and duration of the required sync signals. These values are then converted to the “timing coordinates” and entered into the appropriate GPU registers (addresses 0x3030 to 0x308C). For instance, a horizontal sync signal that starts a display line and lasts for  $t_{\text{hsync}}$  would result in a value for its “switch-on” position of  $p_{x, \text{on}} = xstart$  and a “switch-off” position of  $p_{x, \text{off}} = p_{x, \text{on}} + (t_{\text{hsync}} / t_{\text{ScanDot}})$ , whilst the respective y-components set to “don’t care”. Hence, this signal will provide a pulse of length  $t_{\text{hsync}}$  at the beginning of each line. Position calculations are similar when the sync sequencer shall be used.

The required polarity and optional half cycle delay at the respective GDC pin is produced by the appropriate sync mixer and sync switch settings as explained in 3.10.4 and 3.10.5.

#### External pixel clock

Depending on the specified GDC clock source for pixel clock (received from pin or derived from internal clocks) the GPU can produce a tailored pixel clock signal at the clock pin for the display. Some displays may require a gated clock that has periodical gaps. This behaviour is stated in the display spec. If necessary, it can be produced using the clock gating described in 3.11 in conjunction with the settings for sync mixer 7 that controls this gate. The gate control signal is produced in the same manner as described above for the sync signals.

#### Size, Timing, and Sync Dependencies

Although the values for physical display size, timing, and sync generation are programmable within wide ranges, they are strictly interlocked for a certain display. Therefore, it is important to enter consistent values. The previous paragraphs gave the rules to determine the values, here, table 4-2 shall conclude them as a quick overview to check for correct magnitudes in correspondence to different scan modes for a display with a given pixel resolution of  $X \times Y$  (see also section 3.6). The timing values are approximative and have to be tuned for blanking.

**Table 4-2:** Dependencies for size and timing according to scan mode for a display of given resolution.

	Values	Scan Mode		
		SglScan	DualScan	ZzagScan
Size	PX	X		
	PY	Y		
	PXScanDots	$\left\lfloor \frac{PX \times \text{BitsPerPixel}}{\text{BitsPerScanClock}} \right\rfloor$		
Timing	basic horizontal (hsync) cycle, i.e. $xstop - xstart$	$\approx PXScanDots$		$\approx 2 \times PXScanDots$
	basic vertical (vsync) cycle, i.e. $ystop - ystart$	$\approx Y$	$\approx Y/2$	
	resulting frame period	$\approx PXScanDots \times Y$	$\approx \frac{PXScanDots \times Y}{2}$	$\approx PXScanDots \times Y$

### Interface to Physical Display

The electrical interface of the actual display wired to the GDC determines the number, type and possibly the direction of signals at the pins. Direction information is needed for sync signals, as the GPU supports external and internal synchronization. In case of external synchronization the pins DIS\_HSYNC, DIS\_VSYNC, and DIS\_VREF are inputs, otherwise outputs.

The digital multi-purpose pins DIS\_D[23:0] carry either pixel data or sync signals as given in table 3-14, section 3.14. Pins not necessary for operation can be switched to high-Z by resetting their corresponding output enable bit in the DIR\_MainEnable word (address 0x3260). This applies also to the color key output pin DIS\_CK as well as to the sync signal pins DIS\_HSYNC, DIS\_VSYNC, and DIS\_VREF when in internal synchronization mode.

In case of a display with analog inputs connected to GDC the pins A\_RED, A\_GREEN, and A\_BLUE carry the output if the internal DACs when the following conditions hold (otherwise they are high-Z):

- scan mode (bits [17:16] of DIR\_Format) is set to SglScan AND
- DAC output enable as well as reference voltage enable (bits [28] and [29] of DIR\_MainEnable, address 0x3260) is set AND

*Either:*

- Twin display mode (bit [18] of address 0x3004) is off
- physical color space (bits [3:0] of DIR\_Format, address 0x3004) is set to RGB888,
- bit stream format (bits [11:8] of DIR\_Format) is set to AN (analog display is primary),

*Or (Jasmine only)*

- Twin display mode is on
- physical color space is RGB333...RGB888
- bit stream format is set to S9...S24 (analog display is secondary)

When connecting analog displays to GDC via capacitors a clamping value for defined levels is necessary. This value is entered into the DIR\_AClamp register (address 0x3258) and is output during blanking periods. The similar value DIR\_DClamp (address 0x325C) exists for the digital pins and may there be useful for external glue circuitry.

## 4.2.2 Values Determined by Application

### Matrix Settings (Jasmine only)

If image data is processed in one of the YUV formats appropriate matrix parameters and possibly gamma table settings are necessary. First, the pre-matrix biases (bits [23:0] of address 0x1400) need be determined. These are signed 8 bit numbers in 2's complement which are used to adjust the value ranges of luminance and chrominance before matrix multiplication. Adjustments should be made in a manner that:

- Luminance (Y) values are mapped to a monotonous range of 0...0xFF (zero to full scale)
- Chrominance (U, V) values are mapped to a monotonous range of 0x80...0...0x7F (most negative...neutral gray...most positive)

If the image data comply to the ITUR-601 standard then the default values need not be changed.

Next, the matrix coefficients have to be calculated, i.e. their values are mapped to 8 bit binary numbers. This is accomplished using the following formula:

$$ConfigReg = \lfloor \text{abs}(Coeff) \cdot 128 \rfloor \bmod 256 \quad (11)$$

Please be aware that (cf. 3.7.2)

- The absolute value of any coefficient must lie in the range of 0...2
- The values of coefficients CBxG and CRxG (bits [15:8] and [7:0] of address 0x1408) are internally treated as negative.

If the image data comply to the ITUR-601 standard then the default values need not be changed.

**Gamma Settings (Jasmine only)**

If there is any post processing required after matrix multiplication then this can be accomplished for any function  $I_{\text{gam}} = f(I_{\text{mtx}})$  by loading appropriate values into the gamma tables (bits [23:0] of addresses 0x2800...0x2BFC). In order for the gamma conversion to take effect the gamma enable flag (bit [7] of address 0x1300) must be set.

**4.2.3 User preferences**

In general, all layer specific settings (positions, color resolution, blink/transparency color, blink rate, line doubling) are unlimited at the user's command<sup>1</sup>. Further, blink enables, Z-order, and CLUT offsets can also be chosen arbitrarily. When displaying YUV422 image data chroma interpolation can be switched on to improve image quality. It is also up to the user to switch on line doubling, especially for video data fetched by VIC.

The contents of the color look-up table is also free to choose, with the only limitation that the effective bit width of the contents is determined by the intermediate color space. This limitation applies also to the background color. A similar limitation holds for the color key limits, but with the physical color space.

Duty ratio modulation (cf. 3.8) may be applied to improve the impression of the image displayed. Its usage depends on the physical color space and intermediate color space. For the mappings [intermediate 4bbp to physical 1bbp] and [intermediate RGB222 to physical RGB111] duty ratio modulation is used, hence, the values for the pseudo levels to be generated have to be entered into the respective registers (addresses 0x1380 to 0x13B4).

**4.3 GPU Initialization Sequence**

In order to ensure a correct operation of the connected display as well as for the GPU itself control data has to be entered in a certain sequence. Further, not every control register may be written at any given time.

Basically, information that defines display timing can not be changed after MasterTimingOn has been set to one.

Initialization should be carried out as follows:

7. Determine parameters of the display physically connected to the GDC and derive timing characteristics. Load the values into the appropriate registers.
8. Determine what interface signals are needed and enable them by setting the values in DIR\_MainEnable.
9. Load the contents of the CLUT, if needed.
10. Derive the necessary intermediate color space (dependent on the physical color space), using tables 2-2 and 2-3.
11. Determine whether matrix multiplication is necessary. If so, load pre matrix biases and matrix coefficients (when differing from defaults). If post processing of matrix results is required load gamma tables with appropriate contents.
12. Load the Duty Ratio Modulator registers, if needed.
13. Set layer geometry by loading the respective layer description record registers.
14. Now, MasterTimingOn may be set, thus starting signal generation at the output pins. If the Z-order register remained unchanged in reset state, all pixels are displayed in background color.

During normal operation, color attributes (layer, blinking, transparent, background, CLUT contents) may be set at any time and take effect immediately. Changing layer color spaces, layer position parameters or Z-order register contents affects the display at the start of the next frame being fetched from video RAM<sup>2</sup>

---

1. There is a limitation concerning the color space in that there has to be a legal mapping to the intermediate color space, as given in table 2-1.

## 5 Bandwidth Considerations

### 5.1 Processing Bandwidth

#### 5.1.1 Average Bandwidth

For a first go/no go decision an estimation of average GPU bandwidth demands might be sufficient. It is done on the basis of the number of pixels processed per frame and the frame period. In general, the condition

$$T_{\text{proc}} \leq T_{\text{frame}} \quad (12)$$

must hold for the GPU to work properly. These two periods are calculated as follows:

a) Frame period:

$$T_{\text{frame}} = T_{\text{DisplayClock}} \cdot \Delta x \cdot \Delta y \quad (13)$$

$$\Delta x = X_{\text{Stop}} - X_{\text{Start}} \quad (14)$$

$$\Delta y = \begin{cases} Y_{\text{oddStop}} - Y_{\text{oddStart}} + 1 & \text{for disabled field-toggling} \\ \frac{1}{2} \cdot ((Y_{\text{oddStop}} - Y_{\text{oddStart}} + 1) + (Y_{\text{evenStop}} - Y_{\text{evenStart}} + 1)) & \text{for enabled field-toggling} \end{cases} \quad (15)$$

b) Processing period:

$$\begin{aligned} T_{\text{proc}} &= T_{\text{background}} + T_{\text{layers}} \\ &= T_{\text{CoreClock}} \cdot (DisplayArea + LayerArea) \\ &= T_{\text{CoreClock}} \cdot \left( (PhysX \cdot PhysY) + \sum_{\text{visible layers}} (WX_i \cdot WY_i) \right) \end{aligned} \quad (16)$$

**Note 1:** *DisplayArea* counts only when background is enabled.

**Note 2:** *LayerArea* counts only for potentially visible pixels, i.e. parts of layers outside the physical display area need not be processed and therefore do not count here. However, pixels hidden underneath others do count.

#### 5.1.2 Peak Bandwidth

To have a more reliable statement about whether or not confirming to bandwidth restrictions the peak bandwidth has to be calculated, since this is virtually the limiting constraint. Therefore, a more detailed elaboration into GPU function is necessary.

Pixels are processed in portions of one *line segment*. The size of one such line segment is a function of the visible layers logical color spaces, the physical color space, an internal burst limit (512 pixels) for VideoRAM accesses, and the actual number of pixels per line.

$$SegLen = \min(32 \cdot \min(ppw_{\text{log}}, l), 32 \cdot ppw_{\text{phys}}, 512, PhysSizeX) \quad (17)$$

The values for the number of pixels per word (either per VideoRAM word or per LSA word) according to color spaces are given in tables 5-1 and 5-2. In the special case that no layer is active  $\min(ppw_{\text{log}}, l)$  defaults to 1.

---

2. Due to internal FIFOs the physical display is affected with a delay dependent on picture size, visible layers and their color resolution.

**Table 5-1:** Number of pixels per VideoRAM word according to logical color space.

Logical Color Space Code	1bpp	2bpp	4bpp	8bpp	RGB555	RGB565	RGB888	YUV422	YUV444	YUV555 <sup>a</sup>	YUV655 <sup>a</sup>
<b>ppw<sub>log</sub></b>	32	16	8	4	2	2	1	2	1	2	2

a. This format is only available for MB87P2020-A (Jasmine).

**Table 5-2:** Number of pixels per LSA word according to physical color space.

Physical Color Space Code	1bpp	RGB111	4bpp	RGB333	RGB444	RGB666	RGB888
<b>ppw<sub>phys</sub></b>	24	8	6	2	2	1	1

The limiting constraint (12) can now be refined to

$$\max(T_{\text{seg, proc}}) \leq T_{\text{seg, out}} \quad (18)$$

since every line segment has to be fully processed during a time shorter or equal to the time a segment is output. These two periods of time are calculated as follows:

a) Time to output a line segment:

$$T_{\text{seg, out}} = (T_{\text{DisplayClock}} \cdot \text{SegLen} / \text{StreamFactor}) \quad (19)$$

$$\begin{aligned} \text{StreamFactor} &= (\text{PhysSizeX} / \text{PXScanDots}) \cdot \text{ScanFactor} \\ &= (\text{bpsd} / \text{bpp}_{\text{phys}}) \cdot \text{ScanFactor} \end{aligned} \quad (20)$$

$$\text{ScanFactor} = \begin{cases} 1 & \text{for ScanMode} \neq \text{DualScan} \\ 2 & \text{for ScanMode} = \text{DualScan} \end{cases} \quad (21)$$

The values for bits per scan dot (bpsd) and bits per physical pixel (bpp<sub>phys</sub>) are given in tables 5-3 and 5-4. Please note that only certain combinations of bit stream formats and physical color spaces are supported (cf. table 3-4, p. 165).

**Table 5-3:** Number of bits per scan dot according to bit stream format.

Bit Stream Format Code	S1	S2	S4	S6	S8	S9	S12	S18	S24	AN
<b>bpsd</b>	1	2	4	6	8	9	12	18	24	24

**Table 5-4:** Number of bits per physical pixel according to physical color space.

Physical Color Space Code	1bpp	RGB111	4bpp	RGB333	RGB444	RGB666	RGB888
<b>bpp<sub>phys</sub></b>	1	3	4	9	12	18	24

b) Maximum time to process a segment:

$$\max(T_{\text{seg, proc}}) = T_{\text{CoreClock}} \cdot \text{MaxSegPixel} \quad (22)$$

To obtain the value `MaxSegPixel` the segment with the maximum number of pixels processed has to be found. This is achieved following the algorithm shown in figure 5-1.

```
MaxSegPixel := 0
for y := 0 to (PhysSizeY-1) do
begin
  act_layers:= {}
  foreach l in vis_layers do
  begin
    if y >= OffsY(l) and y < (OffsY(l)+WY(l)) then
      act_layers := {act_layers, l}
    end
  end
  if act_layers == {} then continue
  segm_start := 0
  while (segm_start < PhysSizeX) do
  begin
    segm_stop := min((segm_start + SegLen), PhysSizeX)
    seg_pixels := 0
    foreach l in act_layers do
    begin
      if OX(l) >= segm_stop or (OX(l) + WX(l)) < segm_start then
        continue
      if OX(l) < segm_start then
        layer_start := segm_start
      else
        layer_start := OX(l)
      if (OX(l) + WX(l)) >= segm_stop then
        layer_stop := segm_stop
      else
        layer_stop := OX(l) + WX(l)
      seg_pixels := seg_pixels + (layer_stop - layer_start)
    end
    if logical_color_space(l) == YUV422 then
      seg_pixels := seg_pixels + 2
    if background_enabled then
      seg_pixels := seg_pixels + (segm_stop - segm_start)
    if seg_pixels > MaxSegPixel then
      MaxSegPixel := seg_pixels
    end
  end
  segm_start := segm_stop
end
end
```

**Figure 5-1:** Algorithm to determine the maximum number of processed pixels per line segment.

As a result of this the minimal frequency ratio of core clock to display clock is derived as:

$$\frac{f_{\text{CoreClock}}}{f_{\text{DisplayClock}}} = \frac{T_{\text{DisplayClock}}}{T_{\text{CoreClock}}} \geq \frac{\text{MaxSegPixel}}{\text{SegLen}} \cdot \text{ScanFactor} \quad (23)$$

## 5.2 Memory Bandwidth

Although memory access is more unlikely to become the bottleneck in contrast to processing it may turn out as limiting factor when memory traffic of other GDC units (e.g. Pixel Processor or Direct Access) increases. Therefore, estimations for memory bandwidth are given here.



### 5.2.1 Average Bandwidth

This estimation is done on the basis of the visible<sup>1</sup> layer area, i.e. the area of active layers within the geometrical borders of the physical display, since all pixels displayed had to be fetched.

The mean number of words (which is equal to the number of core clock cycles) transferred from memory is calculated as follows:

$$W_{\text{GPU}} = \sum_{\text{visible layers}} \left( Area_l \cdot \frac{1}{ppw_{\log, l}} \right) \quad (24)$$

The number of pixels per VideoRAM word was already given in table 5-1. For correct GDC function condition (25) must hold:

$$T_{\text{frame}} \geq T_{\text{CoreClock}} \cdot (W_{\text{GPU}} + W_{\text{other units}}) \quad (25)$$

### 5.2.2 Peak Bandwidth

As already mentioned in section 5.1.2, GPU processes data in portions of a line segment. This implies that the segment with the maximum number of words fetched has to be found (it should be noted that this is not necessarily the segment with the maximum number of pixels). The algorithm used to find this segment is

---

1. The area counts also as visible if a layer is (partially) covered by another layer.

shown in figure 5-2. It is quite similar to that in figure 5-1. Segment length is determined using equation (17).

```

MaxSegWords := 0
for y := 0 to (PhysSizeY-1) do
begin
  act_layers:= {}
  foreach l in vis_layers do
  begin
    if y >= OffsY(l) and y < (OffsY(l)+WY(l)) then
      act_layers := {act_layers, l}
    end
  end
  if act_layers == {} then continue
  segm_start := 0
  while (segm_start < PhysSizeX) do
  begin
    segm_stop := min((segm_start + SegLen), PhysSizeX)
    seg_words := 0
    foreach l in act_layers do
    begin
      if OX(l) >= segm_stop or (OX(l) + WX(l)) < segm_start then
        continue
      if OX(l) < segm_start then
        layer_start := segm_start
      else
        layer_start := OX(l)
      if (OX(l) + WX(l)) >= segm_stop then
        layer_stop := segm_stop
      else
        layer_stop := OX(l) + WX(l)
      if logical_color_space(l) == YUV422 then
        layer_stop := layer_stop + 1
      seg_words := seg_words +
        truncate((layer_stop - layer_start + ppwlog(l)-1) / ppwlog(l))
    end
    if seg_words > MaxSegWords then
      MaxSegWords := seg_words
    end
  end
end

```

**Figure 5-2:** Algorithm to determine the maximum number of words fetched per segment.

With this result, condition (25) is refined to

$$T_{\text{seg, out}} \geq T_{\text{CoreClock}} \cdot (W_{\text{peak, GPU}} + W_{\text{peak, other units}}) \quad (26)$$

Where  $T_{\text{seg, out}}$  is the time to output one segment as calculated in equation (19), and  $W_{\text{peak, GPU}}$  is simply MaxSegWords.

## 5.3 Recommendations

In order to ensure correct GPU function it is important to estimate the necessary bandwidth for pixel processing and output. However, one has to bear in mind that not all influences are known in advance, especially the behaviour of other units the GPU shares the SDC and VideoRAM with. Therefore, one should provide for sufficient safety range when determining GDC setup.

There are some issues that can help save memory and processing bandwidth. Following the recommendations given below it may be possible to operate at lower core clock / display clock ratios, thus improving EMC and reducing power consumption.

- When the whole display area is covered by non-transparent layer pixels background painting can be switched off.
- Multiple layers do not increase bandwidth when they do not cover each other.

- Large transparent areas should be avoided, instead the layer's window size should be limited to the smallest rectangle including all non-transparent pixels.
- Color resolution (logical color space) should be reduced to the minimal number of colors needed simultaneously, a more colourful display can be achieved with useful CLUT entries as well. This approach decreases the necessary memory bandwidth.
- Longer blanking periods decrease average bandwidth demands.
- For a given size and display clock dual scan displays demand a higher memory and processing bandwidth.
- Displays with greater StreamFactors according to equation (20) demand higher bandwidth.
- Video display should be used with care, since it produces high memory traffic (from VIC to RAM and from RAM to GPU). VIC provides some means to reduce to necessary bandwidth (dropping certain frames and fields).

## 6 Functional Peculiarities

### 6.1 Configuration Constraints

Due to the great flexibility provided by the extensive programmability *there is no built-in precaution against illegal, not supported or absurd register settings*. Programming has to be carried out with a thorough understanding of the features explained in this document. Items to pay attention to include (but are not limited to):

- Window size does not exceed domain size in every dimension
- Window size greater than zero
- First pixel lies within window size
- Offset within physical display dimensions
- CLUT offset may cause wrap-around for look-up
- Color space settings confirming to tables 2-1 and 2-2, chapter 2
- Non-transparent layers covering others
- Layers multiply exposed (i.e. identical settings for Z-order register entries, this is a waste of memory and processing bandwidth)
- Disabled background will lead to garbage pixels in areas not covered by non-transparent layer contents
- Values for master timing X stop must be greater than or equal to `PXScanDots`, master timing Y stop values must be greater than or equal to `PhysSizeY`, otherwise internal FSMs will hang
- Sync pulse generator and sync sequencer entries should lie within timing range determined by the master timing registers
- Combination of physical color space and bit stream format confirming to table 3-4, chapter 3
- All necessary output pins enabled for connected physical display
- Color key limits according to physical color space
- Register set entries that influence bit stream timing (master timing values, physical color space code, bit stream format code, display dimensions etc.) are locked with `MasterTimingOn` to prevent internal FSMs from hanging. These values may only be set or changed when `MasterTimingOn` is reset. These registers are marked in table 4-1.

### 6.2 Bandwidth

- Bandwidth is an important issue when it comes to usage at performance limits. Unfortunately, it is hard to estimate in advance whether certain settings will cause constraint violations, since other GDC-units, the GPU shares the VideoRAM with, have an influence hard to predict. The rules given in chapter 5 should help checking for possible bottlenecks.

As a support for software development GPU can trigger an interrupt when it suffers from critical data shortage. The interrupt is actually raised when the LSA runs empty (an empty DFU input FIFO does not constitute a data shortage as such, since this state can only be temporary and recoverable). In the

interrupt case the GPU ceases data fetching for the remaining frame period and triggers the ULB flag BWVIO (bit 18 of ULB flag word). However, display output continues with blank pixel data to prevent possible damage at the connected display. This state is left either by soft-reset (i.e. switching MasterTimingOn off and back on again) or at the begin of the next frame period. If bandwidth is still too small the interrupt will be raised again.

- The priority with which GPU requests data from SDC can be controlled with the settings in the GPU\_SDCPrio register (bits [6:4] and [2:0]<sup>1</sup>, address 0x3270). Per default the GPU has the highest priority amongst all GDC units. *Any changes of this value should be done with utmost care*, since a continuous display timing requires a continuous data stream from RAM be maintained.

## 6.3 Image Processing

There are some points to be aware of when processing image data in YUV color space:

- Matrix coefficients CBxG and CRxG are treated as having negative sign
- YUV555 and YUV655 color spaces are auto scaled to YUV444 before matrix multiplication
- Processing YUV422 data restricts horizontal first pixel, window size and offset to even values
- There is only either a mapping through matrix or achromatic mapping from YUV color spaces to intermediate and eventually to physical color space. Therefore, if a direct mapping from YUV to GPU pins is desired a special approach is necessary. It consists of a re-labelling of layers containing YUV data to corresponding RGB color spaces with an equal number of bits per pixel (table 6-1) and then directly mapping these to the physical color space and thus to the pins (cf. table 3-14). Note also that YUV422 uses chrominance multiplexing.

**Table 6-1:** Compatibility list for YUV to RGB re-labelling.

YUV spaces	Corresponding RGB spaces
YUV422	RGB565
YUV444	RGB888
YUV555	RGB555
YUV655	RGB565

- Image quality for pictures in YUV422 may be improved by using chrominance interpolation.
- To improve image appearance for video pictures grabbed by VIC, GPU offers *line doubling*. With this feature (which can be switched on individually for each layer) vertical resolution is reduced to the half by displaying each line in VideoRAM twice on the display (e.g. line #0 from RAM is displayed on display lines #0 and #1, RAM line #1 on display lines #2 and #3 and so on). This approach eliminates inter-field jitter. When used in conjunction with field dropping in VIC it additionally reduces memory bandwidth.
- Pixels of layers in color space RGB555, RGB565 and RGB888 may be fed through the gamma tables to apply further color processing. This mode is enabled when both flags, gamma enable and RGB gamma forcing, are set. Similar arbitrary color processing is achieved for color spaces with less than 16 bits per pixel by simply using the CLUT.

1. Bits [2:0] for low priority are not used in the current implementation, i.e. GPU always uses high priority requests.

## 6.4 Data Output

- GPU pixel output is always LSB-aligned, i.e. the geometrically leftmost pixel is represented in the least significant bit(s). For cases where displays demand MSB-alignment this has to be produced by appropriate wiring on the PCB. There are two exceptional cases: physical color space RGB111 and bit stream format S4 or S8. Then fractions of a pixel are output and the alignment swap can not be obtained by wiring only. Therefore, GPU provides the flag “R/B-swap” which exchanges the red with the blue color channel (exchange of a pixel’s bit 0 and 2).
- Since GPU is designed for LCDs and equivalent *progressive scan*<sup>1</sup> display types it can not produce TV-conform *interlaced* output streams. However, it can produce TV-conform *timing* signals (hsync and vsync, cf. section 3.10) for displays which mimic TV behaviour.
- For bit stream formats other than S24 it is possible to control the values at the pins DIS\_D[23:19] directly by the user (aka port expansion mode). This is achieved by setting the respective sync mixer signal selects all to constant zero. Then bit 0 of the mixer’s function table represents the pin value directly. This direct control works for all bit stream formats on pins DIS\_HSYNC, DIS\_VSYNC, and DIS\_VREF, since these are always available.
- It is possible to run both, an analog and a digital display, concurrently with the same timing (only valid for MB87P2020-A (Jasmine)). This is accomplished using the *twin display mode* (cf. 3.5). However, this works only for single scan displays and pins DIS\_D[23:19] may not be available for sync signal output then.

## 6.5 Diagnostics

GPU provides some diagnostic information in read-only registers that might help during software development. The following data are available:

- Current blink state (bits[31:16], register MDR\_BlinkCtrl, address 0x1308). A ‘1’ at a bit slice designates that the corresponding layer would use its alternative blink color. The information may be used to synchronize different layers or to synchronize drawing and display. A simultaneous start of blinking for all layers can be achieved by simultaneous setting of the blink enable bits (bits [15:0] of MDR\_BlinkCtrl).
- Current timing position (register DIR\_TimingDiag, address 0x3024) in “timing coordinates” as explained in section 3.9. The information is useful to check for correct register settings.
- Current input FIFO load (bits [15:8], register GPU\_SDCPrio, address 0x3270). This information allows to check for potential bandwidth bottlenecks. During normal operation the FIFO will be full (load  $\approx 130$ ) most of the time. However, it does not constitute a critical state as such when it runs empty. Only if there is not enough memory bandwidth left to refill it for longer periods of time the LSA will consequently run empty and raise an interrupt.

---

1. The term *progressive scan* means that a complete frame is output line by line, whilst *interlaced* means that a complete frame is output as two *fields*, where one field contains all lines with odd and the other field all lines with even numbers.

---

## 7 Supported Displays

The tables in the following sections list the result of a display survey done to derive necessary and useful GPU features. They are divided in sections for Passive Matrix LCD, Active Matrix (TFT) LCD, Electroluminescent Displays and Field Emission Displays. Some potential problems for support encountered during this survey are explained in section 7.5.

The tables in sections 7.1 to 7.4 list only those analog inputs necessary for normal display operation. Some of the displays feature extra inputs e.g. for dimming. These pins usually require potentiometers (or current D/A converters). The type of digital interface signals is abbreviated as follows: *C5* denotes 5V CMOS level, *C3* denotes 3.3V CMOS level, and *T5* denotes 5V TTL level. Color resolution is abbreviated this way: *m* denotes monochrome, *c* denotes low color resolution (less than 24 bits per pixel), and *t* denotes true color (16 million colors).

## 7.1 Passive Matrix LCD

Vendor	Type	Res. X x Y	Bits per Pixel	Colors	Scan	# of pixels written at once	digital i/f signals		max. PixClk (MHz)	Framerate (Hz)		Power Supply (V)
							#	Type		min	max	
Optrex	DMF-50970NC	160 x 113	3	c	single <sup>a</sup>	2 2/3	13	C3 / C5 <sup>b</sup>	10...20 <sup>b</sup>	?	?	?
Sharp	LM32P10	320 x 240	1	m	single	4	7	C5	6.5	70	80	5; -20
Sharp	LM64P89	640 x 480	1	m	dual parallel	2 x 4	12	C5	6.5	59	125	5; -19
Sharp	LM64C142	640 x 480	3	c	dual parallel	2 x (2 2/3)	20	T5	12.2	59	120	5; 27
Sanyo	DG24320C5PC	320 x 240	3	c	dual parallel	2 x (1 1/3) 2 x (2 2/3)	21	C5	12.2	73	73	5; 26

a. special pixel order: pixels 1....160,y and 1....160, y+58 are written as one line

b. derived from used column driver ICs

None of the Passive Matrix Displays includes an inverter to produce the high voltage needed for the CCFL backlight.



## 7.2 Active Matrix (TFT) Displays

Vendor	Type	Res. X x Y	Bits per Pixel	Colors	# of pixels written at once	i/f signals			max. PixClk (MHz)	Framerate (Hz)		Power Supply (V)	CCFL Inv. incl.
						analog #	digital			min	max		
							#	Type					
FPD	LDE052T-02	320 x 240	12	t	1	-	20	C5	26.8	55	65	5; 12	no
FPD	LDE052T-12	320 x 240	12	t	1	-	20	C5	26.8	55	65	5; 12	yes
FPD	LDE052T-32	320 x 240	24	t	1	3	8	C5	26.8	55	65	5; 12	yes
FPD	LDE052T-52	320 x 240	24	t	1	3	8	C5	26.8	55	65	5; 12	no
Sharp	LQ5AW116	320 x 234	24	t	1	3	9	C5	7.6	55	65	8; -5	no
Hosiden	HLD0909	640 x 480	3	m	2	-	11	C5	13.3	50	70	5	no
NEC	NL3224AC35-01	320 x 240	24	t	1	3	9	C5	8.4	48	52	9.5	yes
NEC	NL6448AC33-18	640 x 480	18	c	1	-	22	C3/C5	29	58	62	3.3/5; 12	yes
NEC	NL6448AC20-06	640 x 480	12 / 18	c	1	-	17 / 23	C5	29	58	62	5	yes
Sanyo	ALP401RDD	320 x 240	24	t	1	3	5	T5	20.2	51	71	12	yes
Sanyo	ALP401RDV	320 x 240	24	t	1	-	29	T5	6.6	56	63	12	no

All Active Matrix Displays work in single scan mode.

### 7.3 Electroluminescent Displays

Vendor	Type	Res. X x Y	Bits per Pixel	Colors	# of pixels written at once	digital i/f signals		max. PixClk (MHz)	Framerate (Hz)		Power Sup- ply (V)
						#	Type		min	max	
Sharp	LJ32H028	320 x 240	1	m	4	7	C5	6,6	60	120	5; 12
Planar	EL160.80.50	160 x 80	1	m	4	8	C5	7.1	0	240	5; 12
Planar	EL240.64 / EL240.64-SD	240 x 64	1	m	1	6	C5	4.0	0	240	(5) <sup>a</sup> ; 12
Planar	EL4737HB / EL4737HB-ICE	320 x 128	1	m	1	5	T5	5.1	0	120	(5); 12
Planar	EL320.240.36	320 x 240	1	m	4	9	C5	7.1	0	120	(5); 12
Planar	EL320.256-F6 EL320.256-FD6	320 x 256	1	m	1 / 2	9	T5	25		75	5; 11....30 <sup>b</sup>
Planar	EL320.256-FD7	320 x 256	1	m	1 / 2	8	T5	25		120	5; 11....30
Planar	EL512.256-H	512 x 256	1	m	1 / 2	7	T5	30		75	5; 11....30
Planar	EL640.400-CB1/ CD4	640 x 400	1	m	1 / 2	6	T5	30		70	5; 12
Planar	EL640.400-CB3	640 x 400	1	m	1 / 2	6	T5	30		70	5; 24
Planar	EL640.400-C2, - C3, -C4	640 x 400	1	m	1 / 2	7	T5	30		70	5; 11....30
Planar	EL640.400-CEx	640 x 400	1	m	1 / 2 x 4	16	C5	6.6		160	5; 12

Vendor	Type	Res. X x Y	Bits per Pixel	Colors	# of pixels written at once	digital i/f signals		max. PixClk (MHz)	Framerate (Hz)		Power Sup- ply (V)
						#	Type		min	max	
Planar	EL640.480-AF1, - AG1	640 x 480	1	m	2 x 4	11	C5	6.5		120	5; 12
Planar	EL640.480-AM8	640 x 480	1	m	2 x 4	11	C5	6.5		120	5; 24
Planar	EL640.480-AA1	640 x 480	4	c	1	10	C5	30		80	5; 12
Planar	EL640.480-ASB	640 x 480	4	m	1 / 2	16	T5	30		70	12

a. optional

b. arbitrary within range

Except the Planar EL640.400-CEEx, which has an optional dual scan mode, all electroluminescent displays work in single scan mode.

## 7.4 Field Emission Displays

Vendor	Type	Res. X x Y	Bits per Pixel	Colors	# of pixels written at once	digital i/f signals		max. PixClk (MHz)	Framerate (Hz)		Power Sup- ply (V)
						#	Type		min	max	
Motorola	D07SPD310	128 x 160	9	c	1	14	C3	1.6	60	75	3.3; 7.2
PixTech	FE532S-M1	320 x 240	1	m	4	9	T5	6.0	50	350	5; 12

Both field emission displays work in single scan mode.

## 7.5 Limitations for support

### 7.5.1 No Support due to VCOM Inversion

There were another two displays in the survey, the Fujitsu FLC15ADCAW and the Sharp LQ7BW506, which suffer from a rather “substrate-oriented” interface. These displays do not have the necessary circuitry to prevent DC from the liquid crystal. Therefore, they require RGB input voltage and common electrode voltage ( $V_{COM}$ ) be reversed every line. This inversion works as follows: For one line a voltage of e.g. -1.9V is applied to the common electrode. A voltage of 0.6V on the RGB inputs produces then a white, a voltage of 4.6V a black display. The next line, the common electrode is clamped to 5.9V. Now, a voltage of 0.6V on RGB produces black and a voltage of 4.6V produces white. In other words: common electrode voltage is alternated between -1.9 and 5.9V, while the voltage *difference* for RGB remains at 4V but with alternating result (color) for every line. This causes the following problems:

- There is *no way* to produce the voltages mentioned above on-chip within GDC, since they exceed the limits of supply voltage by far. Therefore, additional external circuitry is indispensable (Sharp recommend their dedicated video signal polarity reversing IC IR3Y29A).
- As the spec sheets state, the voltages actually used need to be fine-tuned for *every individual display* to obtain optimum contrast (i.e.  $V_{COM}$  DC component (mean level),  $V_{COM}$  AC component (voltage difference), as well as RGB DC and AC components). This implies external potentiometers to produce the bias voltages.

### 7.5.2 Problems due to 5V CMOS Interfaces

There is a number of displays, which demand 5V CMOS levels for their digital interface signals (marked with “C5” in the respective column). This is problematic since the GDC-ASIC is *not able* to guarantee the correct “High”-level at its outputs since the ASIC runs at 3.3V.

**Work-around:** Either external pull-up resistors or special level shifter ICs are used as e.g. the Philips 74ALVC164245 dual supply translating transceiver.

---

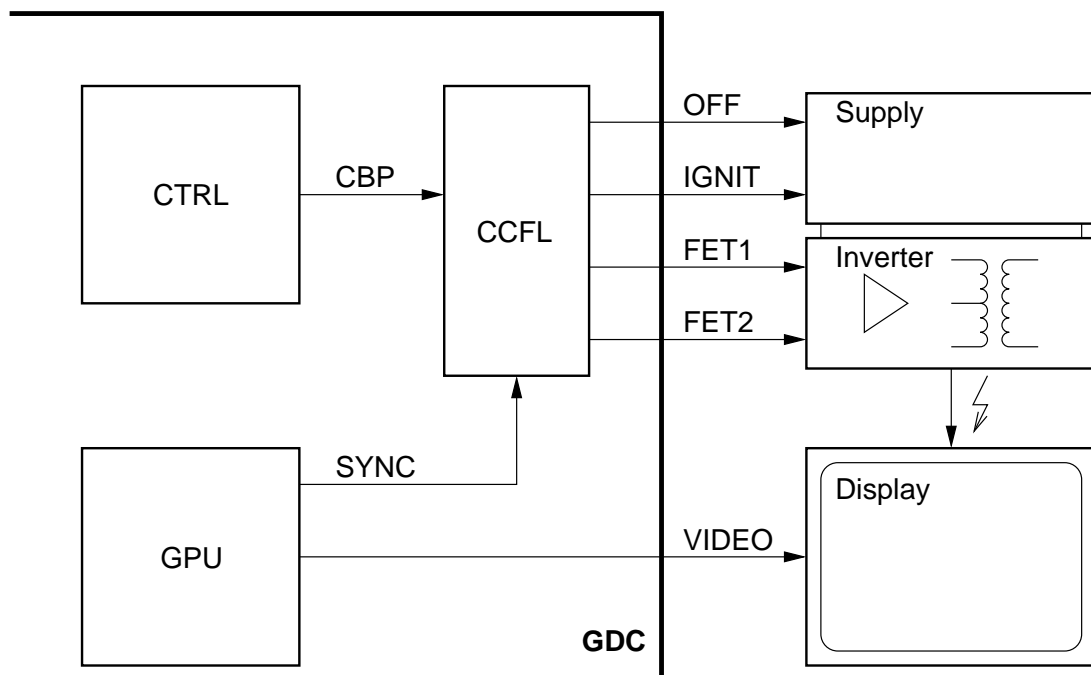
## ***B-9 Cold Cathode Fluorescence Light Driver (CCFL)***



# 1 Introduction

The Cold Cathode Fluorescence Light Driver is used to generate the needed signals for controlling a switched power supply and ionisation circuit of a cold cathode lamp. The intention was that brightness of the lamp should be adjustable in a wide range.

Following figure shows a block diagram of the circuit driving the cold cathode lamp which can be used for generating high voltage supplying the displays back light.



**Figure 1-1:** Application overview for the CCFL circuit

In general CCFL is independent from other GDC components. It is initialized via the Control Bus Port (CBP) interface which is used to setup almost all GDC control registers. After initialization CCFL works as stand-alone circuit.

To avoid interferences between back light flashing frequency and picture refresh rate of the display a synchronization input is provided which internally is connected with the video frame synchronization signal of the Graphic Processing Unit (GPU). Another synchronization method is possible by the connected MCU over the control interface when writing to the synchronization flag of the control register.

MB87P2020-A has a connection to GPU sync mixer 5 as third synchronization possibility. Because a sync mixer can match many display coordinates within one GPU frame (see GPU description for details about sync signal generation) a higher CCFL frequency compared to simple frame synchronization can be achieved without losing synchronization between CCFL and display output.

In short terms the CCFL has the following synchronization possibilities:

- Internal GPU frame synchronization
- Software synchronization
- Internal sync mixer synchronization (MB87P2020-A only)

Simply spoken the CCFL circuit is a kind of pulse width modulation of lightning duration of the fluorescence lamp within one frame period of the display. If the lamp is switched on CCFL provides bi-phase pulse signals (pins CCFL\_FET1, CCFL\_FET2) for direct controlling the complementary FET gates of primary inductance of the voltage transverter. Additional to controlling the lightning and off durations the ignition of the lamp is handled (pins: CCFL\_OFF, CCFL\_IGNIT).

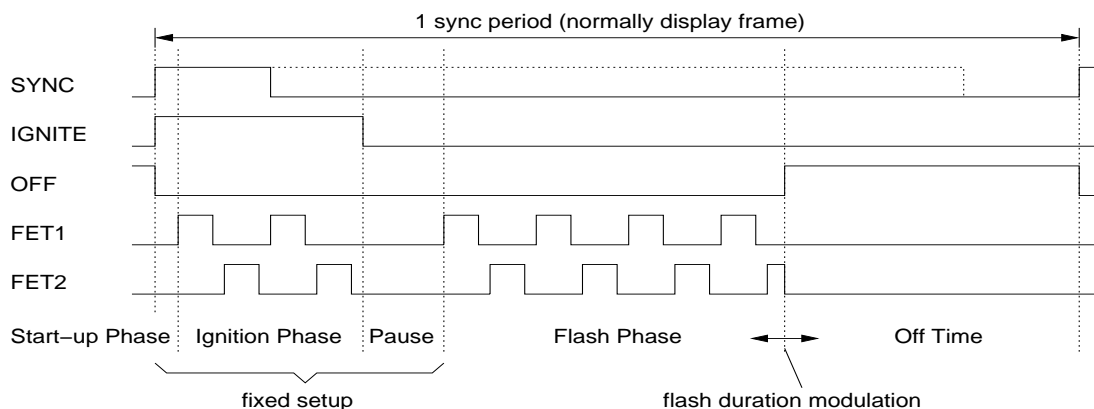
If a simple PWM output is needed (for CCFL supplies without direct FET connection or different voltage converter implementations) a low active PWM signal for brightness control is provided by the CCFL\_OFF pin.



## 2 Signal Waveform

### 2.1 General Description

The following figure shows a waveform example for the external signals and the synchronisation signal. One cycle consists of five phases, a start-up phase, ionisation phase, pause, flash phase and the remaining time until the rising edge of the synchronisation signal.



**Figure 2-1:** Principle waveform of output signals relative to SYNC

The duration of the start-up phase is fixed, the duration of the following three phases is determined by the values of the registers IGNITE, PAUSE and FLASH. Normally IGNITE and PAUSE values are fixed setup to be optimal for the lamp characteristics. Brightness modulation is done by changing the duration of the FLASH phase.

Table 2-1 describes the values of the external signals in each phase. While FET1 and FET2 directly controlling the voltage converter, IGNIT and OFF are for controlling the height of input voltage or to disable it completely.

**Table 2-1:** Phases of one CCFL cycle

Signal	Value				
	Start-up phase	Ionisation phase	Pause	Flash phase	Remaining time
FET1/FET2	low	pulsed <sup>a</sup>	low	pulsed <sup>a</sup>	low
IGNIT	high	high	low	low	low
OFF	low	low	low	low	high

a.see section 2.3

### 2.2 Duration of the Phases

The duration of the certain phases is determined by the value of the prescaler register SCALE and the values of the registers IGNITE, PAUSE and FLASH. The value of the prescaler register defines a base period, which is used to derive pulse shapes and duration of the phases.

$$t_{\text{bper}} = t_{\text{CLK}} \cdot \text{SCALE}$$

The duration of the appropriate phases calculates as follows:

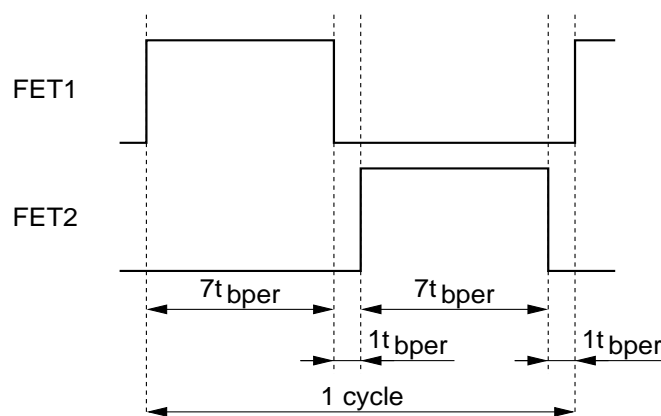
15. Start-up phase:  $t_{\text{start}} = t_{\text{bper}} \cdot 8$
16. Ionization phase:  $t_{\text{ion}} = t_{\text{bper}} \cdot 4 \cdot \text{IGNITE}$
17. Pause:  $t_{\text{pause}} = t_{\text{bper}} \cdot 4 \cdot \text{PAUSE}$
18. Flash phase:  $t_{\text{flash}} = t_{\text{bper}} \cdot 4 \cdot \text{FLASH}$

The fifth phase equals to the remaining off time until the next rising edge of the synchronisation signal

Note: If FLASH duration exceeds display frame duration the next synchronization will be dropped and lamp is off the next period after previous FLASH duration is ended. This could be result in flickering or malfunction of the lamp.

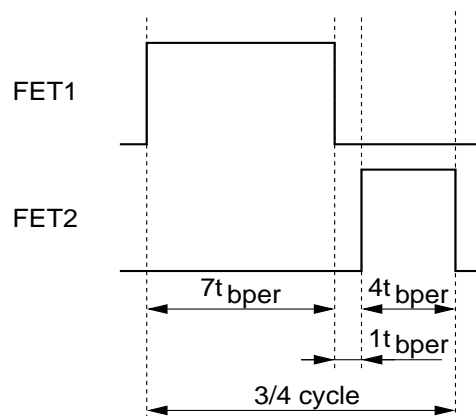
## 2.3 Pulse shape of FET1 and FET2

The following figure shows the pulse shape of pulses, which have to be generated during Ionisation and Flash phase.



**Figure 2-2:** Timing definition of FET control signals

If the end of a phase is reached during a high period of a pulse, a shortened pulse has to be generated as shown in the following figure.



**Figure 2-3:** Timing definition of FET control signals for fractured number of cycles

## 3 Register Description

### 3.1 Overview

**Table 3-1:** Control Register CCFL1

Name	Description	Position	Reset Value	Access
CCFL1_SSEL <sup>a</sup>	Sync select 1	[28]	0	R/W
CCFL1_EN	CCFL enable	[27]	0	R/W
CCFL1_PROT	Protect settings	[26]	0	R/W
CCFL1_SNCS	Sync select 0	[25]	0	R/W
CCFL1_SYNC	Sync software trigger	[24]	0	R/W
-	reserved	[23:8]		
CCFL1_SCL	Prescaler Register	[7:0]	0	R/W

a. MB87P2020-A only

**Table 3-2:** Duration Register CCFL2

Name	Description	Position	Reset Value	Access
CCFL2_FLS	Flash duration	[31:16]	0	R/W
CCFL2_PSE	Pause duration	[15:8]	0	R/W
CCFL2_IGNT	Ionisation duration (ignition)	[7:0]	0	R/W

### 3.2 Control Bits

For MB87P2020-A an additional control bit for selecting sync mixer synchronization was necessary. Note that this new control bit (CCFL1\_SSEL) takes precedence over SNCS flag.

For other devices than MB87P2020-A writing to this control bit is ignored while reading returns always '0'.

**Table 3-3:** Control bit description

Bit	Name	Description
28 <sup>a</sup>	CCFL1_SSEL	SYNCSEL. If this bit is not set (0), synchronization is controlled by SNCS. Otherwise (1) the output of GPU sync mixer 5 is used for synchronization.
27	CCFL1_EN	COCADEN. If this bit is set, the Cold Cathode Driver is enabled. Otherwise the pins have its inactive state (FET1/2=0, IGNIT=0, OFF=1).
26	CCFL1_PROT	PROTECT. If this bit is set, write access to the duration register changes the value of the register, but the durations of the phases are not influenced until this bit will be cleared. If this bit is not set, a write access changes the durations immediately.

**Table 3-3:** Control bit description

Bit	Name	Description
25	CCFL1_SNCS	SYNCSEL. If this bit is not set (0), VSYNC display synchronisation is enabled at each first active line. Otherwise (1) synchronisation by the EXTYSYNC flag (software) will be used.
24	CCFL1_SYNC	EXTSYNC. Setting these bit is interpreted as synchronization pulse if SYNCSEL=1. The reset has to be done by software too.

a. MB87P2020-A only

## 4 Application Notes

### 4.1 CCFL Setup Example

The time base of CCFL Driver should be near 800 ns. Assumed we have a system clock frequency CLKK of 60 MHz (16.6 ns).

$$t_{\text{bper}} = 800 \text{ ns}$$

$$\text{CCFL1\_SCL} = t_{\text{CLKK}} / t_{\text{bper}} = 800 \text{ ns} / 16.6 \text{ ns} = 48 = 0x30$$

We have to setup a prescaler value of 0x30 in the SCALE register.

Possible values for Ignition duration and Pause can be 0x10 and 0x80 for instance. This depends mainly on the lamp characteristics and is not too critical for setting up the right value.

If we have a frame rate of 60 Hz so we get nearly

$$1 / 60\text{Hz} / 800\text{ns} = 20833 \text{ } t_{\text{bper}} = 0x5161 \text{ } t_{\text{bper}} \text{ cycles.}$$

Due to the fact that setup duration values are interpreted as macro cycles of 4 times  $t_{\text{bper}}$  one period has a duration of

$$0x5161 \text{ } t_{\text{bper}} \text{ cycles} / 4 = 0x1458 \text{ macro cycles.}$$

After subtraction of 0x10 ignition time and 0x80 pause duration and 1 for the start-up phase we got the resulting dimming range of 0x0000 to 0x13C7 macro cycles from dark to full luminescence.

Be careful not to setup a larger value for FLASH duration, then the lamp ignition may happen each other frame only and is flickering then.

### 4.2 CCFL Protection

During initialization of setup values there is no need to switch off the CCFL driver. To avoid inconsistency of configuration data a protection mechanism can be used. This method offers a way that no forbidden output timing is generated and the lamp or the supply circuitry would not be damaged. To do a secure configuration following sequence should be used:

1. Switch CCFL\_PROT to '1'
2. Configure CCFL2 duration registers.
3. Release protection by setting CCFL\_PROT = '0'

This gives the possibility to modulate the durations without switching the lamp off.



---

## ***PART C - Pinning and Electrical Specification***





# 1 Pinning and Buffer Types

## 1.1 Pinning for MB87P2020-A

### 1.1.1 Pinning

Table 1-1 shows the pinning for MB87P2020-A (redesigned Jasmine) sorted by pin number while table 1-2 is sorted by names. Both tables refer to a standard Fujitsu 208-pin (L)QFP package FPT-208P-Mxx.

**Table 1-1:** MB87P2020-A pinning sorted by pin number

Pin	Name	Buffer type	Description
1	MODE[0]	BFNNQLX	Mode Pin
2	MODE[1]	BFNNQLX	Mode Pin
3	GND	GND	
4			Place holder for analog area
5	A_GREEN	OTAMX	Analog Green
6	DAC2_VSSA1	ITAVSX	DAC Ground
7	DAC2_VDDA1	ITAVDX	DAC Supply 2.5V
8	A_BLUE	OTAMX	Analog Blue
9	DAC1_VDDA1	ITAVDX	DAC Supply 2.5V
10	DAC1_VSSA1	ITAVSX	DAC Ground
11	A_VRO	OTAMX	DAC Full Scale Adjust
12	DAC1_VSSA	ITAVSX	DAC Ground
13	DAC1_VDDA	ITAVDX	DAC Supply 2.5V
14	A_RED	OTAMX	Analog Red
15	DAC3_VDDA1	ITAVDX	DAC Supply 2.5V
16	DAC3_VSSA1	ITAVSX	DAC Ground
17	VREF	ITAMX	DAC test pin VREF
18			Place holder for analog area
19	VDDI	VDDI#	Core supply 2.5 V
20	TEST	ITCHX	Fujitsu test pin
21	VSC_CLKV	ITFHX	Video Scaler Clock
22	RESETX	ITFHX	Reset
23	OSC_OUT	BX4MFSR2X	XTAL output
24	VDDE[0]		IO supply 3.3V
25	OSC_IN	IXN3X	XTAL input

**Table 1-1:** MB87P2020-A pinning sorted by pin number

Pin	Name	Buffer type	Description
26	GND	GND	
27	VDDI	VDDI#	Core supply 2.5 V
28	RDY_TRIEN	B3NNLMR2X	Control RDY pin behaviour
29	APLL_AVDD		APLL supply 2.5V
30	APLL_AVSS		APLL GND
31	RCLK	ITFHX	Reserved clock
32	ULB_A[20]	BFNNQLX	ULB Interface Address
33	ULB_A[19]	BFNNQLX	ULB Interface Address
34	VDDI	VDDI#	Core supply 2.5 V
35	ULB_A[18]	BFNNQLX	ULB Interface Address
36	ULB_A[17]	BFNNQLX	ULB Interface Address
37	ULB_A[16]	BFNNQLX	ULB Interface Address
38	GND	GND	GND
39	ULB_A[15]	BFNNQLX	ULB Interface Address
40	ULB_A[14]	BFNNQLX	ULB Interface Address
41	ULB_A[13]	BFNNQLX	ULB Interface Address
42	ULB_A[12]	BFNNQLX	ULB Interface Address
43	VDDE	VDDE#	IO supply 3.3V
44	ULB_CLK	ITFHX	ULB Interface Clock
45	ULB_A[11]	BFNNQLX	ULB Interface Address
46	ULB_A[10]	BFNNQLX	ULB Interface Address
47	ULB_A[9]	BFNNQLX	ULB Interface Address
48	ULB_A[8]	BFNNQLX	ULB Interface Address
49	ULB_A[7]	BFNNQLX	ULB Interface Address
50	GND	GND	GND
51	ULB_A[6]	BFNNQLX	ULB Interface Address
52	ULB_A[5]	BFNNQLX	ULB Interface Address
53	ULB_A[4]	BFNNQLX	ULB Interface Address
54	ULB_A[3]	BFNNQLX	ULB Interface Address
55	ULB_A[2]	BFNNQLX	ULB Interface Address
56	ULB_A[1]	BFNNQLX	ULB Interface Address
57	ULB_A[0]	BFNNQLX	ULB Interface Address
58	ULB_CS	BFNNQLX	ULB Interface Chip Select

**Table 1-1:** MB87P2020-A pinning sorted by pin number

Pin	Name	Buffer type	Description
59	ULB_RDX	BFNNQLX	ULB Interface Read
60	GND	GND	GND
61	VDDE	VDDE#	IO supply 3.3V
62	ULB_DACK	BFNNQLX	ULB Interface DMA Acknowledge
63	ULB_D[31]	BFNNQMX	ULB Interface Data
64	ULB_D[30]	BFNNQMX	ULB Interface Data
65	ULB_D[29]	BFNNQMX	ULB Interface Data
66	GND[1]		GND
67	VDDE[1]		IO supply 3.3V
68	ULB_D[28]	BFNNQMX	ULB Interface Data
69	ULB_D[27]	BFNNQMX	ULB Interface Data
70	ULB_D[26]	BFNNQMX	ULB Interface Data
71	ULB_D[25]	BFNNQMX	ULB Interface Data
72	GND	GND	GND
73	VDDI	VDDI#	Core supply 2.5 V
74	VDDE[2]		IO supply 3.3V
75	ULB_D[24]	BFNNQMX	ULB Interface Data
76	ULB_D[23]	BFNNQMX	ULB Interface Data
77	ULB_D[22]	BFNNQMX	ULB Interface Data
78	ULB_D[21]	BFNNQMX	ULB Interface Data
79	VDDI	VDDI#	Core supply 2.5 V
80	GND[2]		GND
81	ULB_D[20]	BFNNQMX	ULB Interface Data
82	ULB_D[19]	BFNNQMX	ULB Interface Data
83	ULB_D[18]	BFNNQMX	ULB Interface Data
84	ULB_D[17]	BFNNQMX	ULB Interface Data
85	GND	GND	GND
86	VDDI	VDDI#	Core supply 2.5 V
87	ULB_D[16]	BFNNQMX	ULB Interface Data
88	ULB_D[15]	BFNNQMX	ULB Interface Data
89	ULB_D[14]	BFNNQMX	ULB Interface Data
90	ULB_D[13]	BFNNQMX	ULB Interface Data
91	GND[3]		GND

**Table 1-1:** MB87P2020-A pinning sorted by pin number

Pin	Name	Buffer type	Description
92	VDDE[3]		IO supply 3.3V
93	ULB_D[12]	BFNNQMX	ULB Interface Data
94	ULB_D[11]	BFNNQMX	ULB Interface Data
95	ULB_D[10]	BFNNQMX	ULB Interface Data
96	GND	GND	GND
97	VDDE	VDDE#	IO supply 3.3V
98	ULB_D[9]	BFNNQMX	ULB Interface Data
99	ULB_D[8]	BFNNQMX	ULB Interface Data
100	ULB_D[7]	BFNNQMX	ULB Interface Data
101	ULB_D[6]	BFNNQMX	ULB Interface Data
102	VDDE[4]		IO supply 3.3V
103	GND[4]		GND
104	ULB_D[5]	BFNNQMX	ULB Interface Data
105	ULB_D[4]	BFNNQMX	ULB Interface Data
106	ULB_D[3]	BFNNQMX	ULB Interface Data
107	GND	GND	GND
108	ULB_D[2]	BFNNQMX	ULB Interface Data
109	VDDE[5]		IO supply 3.3V
110	SDRAM_VCC[0]		SDRAM supply 2.5V
111	ULB_D[1]	BFNNQMX	ULB Interface Data
112	ULB_D[0]	BFNNQMX	ULB Interface Data
113	GND[5]		GND
114	VDDE	VDDE#	IO supply 3.3V
115	SDRAM_VCC[1]		SDRAM supply 2.5V
116	ULB_RDY	OTFTQMX	ULB Interface Ready
117	ULB_DSTP	BFNNQMX	ULB Interface DMA Stop
118	SDRAM_VCC[2]		SDRAM supply 2.5V
119	GND	GND	GND
120	ULB_DREQ	OTFTQMX	ULB Interface DMA Request
121	ULB_INTRQ	OTFTQMX	ULB Interface Interrupt Request
122	ULB_WRX[0]	ITFHX	ULB Interface Write Enable (D[31:24])
123	VDDI	VDDI#	Core supply 2.5 V
124	ULB_WRX[1]	ITFHX	ULB Interface Write Enable (D[23:16])

**Table 1-1:** MB87P2020-A pinning sorted by pin number

Pin	Name	Buffer type	Description
125	SDRAM_VCC[3]		SDRAM supply 2.5V
126	ULB_WRX[2]	ITFHX	ULB Interface Write Enable (D[15:8])
127	ULB_WRX[3]	ITFHX	ULB Interface Write Enable (D[7:0])
128	SDRAM_PBI	IPBIX	SDRAM Test mode
129	VDDE[7]		IO supply 3.3V
130	GND	GND	GND
131	VDDI	VDDI#	Core supply 2.5 V
132	SDRAM_TBST	ITBSTX	SDRAM test mode
133	SDRAM_TTST	ITTSTX	SDRAM test mode
134	VPD	VPDX	Fujitsu Tester Pin
135	DIS_D[0]	B3NNLMR2X	Display Data
136	DIS_D[1]	B3NNLMR2X	Display Data
137	DIS_D[2]	B3NNLMR2X	Display Data
138	VDDI	VDDI#	Core supply 2.5 V
139	DIS_D[3]	B3NNLMR2X	Display Data
140	DIS_D[4]	B3NNLMR2X	Display Data
141	DIS_D[5]	B3NNLMR2X	Display Data
142	GND	GND	GND
143	DIS_D[6]	B3NNLMR2X	Display Data
144	DIS_D[7]	B3NNLMR2X	Display Data
145	DIS_D[8]	B3NNLMR2X	Display Data
146	DIS_D[9]	B3NNLMR2X	Display Data
147	VDDE	VDDE#	IO supply 3.3V
148	DIS_D[10]	B3NNLMR2X	Display Data
149	DIS_D[11]	B3NNLMR2X	Display Data
150	DIS_D[12]	B3NNLMR2X	Display Data
151	DIS_D[13]	B3NNLMR2X	Display Data
152	DIS_D[14]	B3NNLMR2X	Display Data
153	DIS_D[15]	B3NNLMR2X	Display Data
154	GND	GND	GND
155	VDDE[8]		IO supply 3.3V
156	DIS_D[16]	B3NNLMR2X	Display Data
157	DIS_D[17]	B3NNLMR2X	Display Data

**Table 1-1:** MB87P2020-A pinning sorted by pin number

Pin	Name	Buffer type	Description
158	DIS_D[18]	B3NNLMR2X	Display Data
159	DIS_D[19]	B3NNLMR2X	Display Data
160	DIS_D[20]	B3NNLMR2X	Display Data
161	DIS_D[21]	B3NNLMR2X	Display Data
162	DIS_D[22]	B3NNLMR2X	Display Data
163	DIS_D[23]	B3NNLMR2X	Display Data
164	GND	GND	GND
165	VDDE	VDDE#	IO supply 3.3V
166	DIS_CKEY	B3NNLMR2X	Display Colour Key
167	DIS_PIXCLK	B3NNLMR2X	Display Pixel Clock (programmable in/out)
168	DIS_VSYNC	B3NNLMR2X	Display programmable sync
169	DIS_HSYNC	B3NNLMR2X	Display programmable sync
170	DIS_VREF	B3NNLMR2X	Display programmable sync
171	VSC_D[0]	BFNNQLX	Video Scaler Data Input
172	VSC_D[1]	BFNNQLX	Video Scaler Data Input
173	VSC_D[2]	BFNNQLX	Video Scaler Data Input
174	VSC_D[3]	BFNNQLX	Video Scaler Data Input
175	VSC_D[4]	BFNNQLX	Video Scaler Data Input
176	GND	GND	GND
177	VDDI	VDDI#	Core supply 2.5 V
178	VDDE[9]		IO supply 3.3V
179	MODE[2]	BFNNQLX	Mode Pin
180	MODE[3]	BFNNQLX	Mode Pin
181	VSC_D[5]	BFNNQLX	Video Scaler Data Input
182	VSC_D[6]	BFNNQLX	Video Scaler Data Input
183	VDDI	VDDI#	Core supply 2.5 V
184	VSC_D[7]	BFNNQLX	Video Scaler Data Input
185	VSC_D[8]	BFNNQLX	Video Scaler Data Input
186	VSC_D[9]	BFNNQLX	Video Scaler Data Input
187	VSC_D[10]	BFNNQLX	Video Scaler Data Input
188	VSC_D[11]	BFNNQLX	Video Scaler Data Input
189	GND	GND	GND
190	VDDI	VDDI#	Core supply 2.5 V

**Table 1-1:** MB87P2020-A pinning sorted by pin number

Pin	Name	Buffer type	Description
191	VSC_D[12]	BFNNQLX	Video Scaler Data Input
192	VSC_D[13]	BFNNQLX	Video Scaler Data Input
193	VSC_D[14]	BFNNQLX	Video Scaler Data Input
194	VSC_D[15]	BFNNQLX	Video Scaler Data Input
195	VSC_VREF	BFNNQLX	Video Scaler Vertical Reference
196	VSC_VACT	BFNNQLX	Video Scaler VACT
197	VSC_ALPHA	BFNNQLX	Video Scaler ALPHA
198	VSC_IDENT	BFNNQLX	Video Scaler Field identification
199	SPB_BUS	BFNNQHX	SPB Interface
200	GND	GND	GND
201	VDDE	VDDE#	IO supply 3.3V
202	SPB_TST	BFNNQHX	SPB Test
203	CCFL_FET2	OTFTQMX	CCFL FET driver
204	CCFL_FET1	OTFTQMX	CCFL FET driver
205	CCFL_IGNIT	OTFTQMX	CCFL supply control IGNITION
206	GND[6]		GND
207	VDDE[10]		IO supply 3.3V
208	CCFL_OFF	OTFTQMX	CCFL supply control OFF

**Table 1-2:** MB87P2020-A pinning sorted by name

Pin	Name	Buffer type	Description
4			Place holder for analog area
18			Place holder for analog area
8	A_BLUE	OTAMX	Analog Blue
5	A_GREEN	OTAMX	Analog Green
14	A_RED	OTAMX	Analog Red
11	A_VRO	OTAMX	DAC Full Scale Adjust
29	APLL_AVDD		APLL supply 2.5V
30	APLL_AVSS		APLL GND
204	CCFL_FET1	OTFTQMX	CCFL FET driver
203	CCFL_FET2	OTFTQMX	CCFL FET driver
205	CCFL_IGNIT	OTFTQMX	CCFL supply control IGNITION

**Table 1-2:** MB87P2020-A pinning sorted by name

Pin	Name	Buffer type	Description
208	CCFL_OFF	OTFTQMX	CCFL supply control OFF
13	DAC1_VDDA	ITAVDX	DAC Supply 2.5V
9	DAC1_VDDA1	ITAVDX	DAC Supply 2.5V
12	DAC1_VSSA	ITAVSX	DAC Ground
10	DAC1_VSSA1	ITAVSX	DAC Ground
7	DAC2_VDDA1	ITAVDX	DAC Supply 2.5V
6	DAC2_VSSA1	ITAVSX	DAC Ground
15	DAC3_VDDA1	ITAVDX	DAC Supply 2.5V
16	DAC3_VSSA1	ITAVSX	DAC Ground
166	DIS_CKEY	B3NNLMR2X	Display Colour Key
135	DIS_D[0]	B3NNLMR2X	Display Data
148	DIS_D[10]	B3NNLMR2X	Display Data
149	DIS_D[11]	B3NNLMR2X	Display Data
150	DIS_D[12]	B3NNLMR2X	Display Data
151	DIS_D[13]	B3NNLMR2X	Display Data
152	DIS_D[14]	B3NNLMR2X	Display Data
153	DIS_D[15]	B3NNLMR2X	Display Data
156	DIS_D[16]	B3NNLMR2X	Display Data
157	DIS_D[17]	B3NNLMR2X	Display Data
158	DIS_D[18]	B3NNLMR2X	Display Data
159	DIS_D[19]	B3NNLMR2X	Display Data
136	DIS_D[1]	B3NNLMR2X	Display Data
160	DIS_D[20]	B3NNLMR2X	Display Data
161	DIS_D[21]	B3NNLMR2X	Display Data
162	DIS_D[22]	B3NNLMR2X	Display Data
163	DIS_D[23]	B3NNLMR2X	Display Data
137	DIS_D[2]	B3NNLMR2X	Display Data
139	DIS_D[3]	B3NNLMR2X	Display Data
140	DIS_D[4]	B3NNLMR2X	Display Data
141	DIS_D[5]	B3NNLMR2X	Display Data
143	DIS_D[6]	B3NNLMR2X	Display Data
144	DIS_D[7]	B3NNLMR2X	Display Data
145	DIS_D[8]	B3NNLMR2X	Display Data



Table 1-2: MB87P2020-A pinning sorted by name

Pin	Name	Buffer type	Description
146	DIS_D[9]	B3NNLMR2X	Display Data
169	DIS_HSYNC	B3NNLMR2X	Display programmable sync
167	DIS_PIXCLK	B3NNNMR2X	Display Pixel Clock (programmable in/out)
170	DIS_VREF	B3NNLMR2X	Display programmable sync
168	DIS_VSYNC	B3NNLMR2X	Display programmable sync
3	GND	GND	
26	GND	GND	
38	GND	GND	GND
50	GND	GND	GND
60	GND	GND	GND
72	GND	GND	GND
85	GND	GND	GND
96	GND	GND	GND
107	GND	GND	GND
119	GND	GND	GND
130	GND	GND	GND
142	GND	GND	GND
154	GND	GND	GND
164	GND	GND	GND
176	GND	GND	GND
189	GND	GND	GND
200	GND	GND	GND
66	GND[1]		GND
80	GND[2]		GND
91	GND[3]		GND
103	GND[4]		GND
113	GND[5]		GND
206	GND[6]		GND
1	MODE[0]	BFNNQLX	Mode Pin
2	MODE[1]	BFNNQLX	Mode Pin
179	MODE[2]	BFNNQLX	Mode Pin
180	MODE[3]	BFNNQLX	Mode Pin
25	OSC_IN	IXN3X	XTAL input

Table 1-2: MB87P2020-A pinning sorted by name

Pin	Name	Buffer type	Description
23	OSC_OUT	BX4MFSR2X	XTAL output
31	RCLK	ITFHX	Reserved clock
28	RDY_TRIEN	B3NNLMR2X	Control RDY pin behaviour
22	RESETX	ITFHX	Reset
128	SDRAM_PBI	IPBIX	SDRAM Test mode
132	SDRAM_TBST	ITBSTX	SDRAM test mode
133	SDRAM_TTST	ITTSTX	SDRAM test mode
110	SDRAM_VCC[0]		SDRAM supply 2.5V
115	SDRAM_VCC[1]		SDRAM supply 2.5V
118	SDRAM_VCC[2]		SDRAM supply 2.5V
125	SDRAM_VCC[3]		SDRAM supply 2.5V
199	SPB_BUS	BFNNQHX	SPB Interface
202	SPB_TST	BFNNQHX	SPB Test
20	TEST	ITCHX	Fujitsu test pin
57	ULB_A[0]	BFNNQLX	ULB Interface Address
46	ULB_A[10]	BFNNQLX	ULB Interface Address
45	ULB_A[11]	BFNNQLX	ULB Interface Address
42	ULB_A[12]	BFNNQLX	ULB Interface Address
41	ULB_A[13]	BFNNQLX	ULB Interface Address
40	ULB_A[14]	BFNNQLX	ULB Interface Address
39	ULB_A[15]	BFNNQLX	ULB Interface Address
37	ULB_A[16]	BFNNQLX	ULB Interface Address
36	ULB_A[17]	BFNNQLX	ULB Interface Address
35	ULB_A[18]	BFNNQLX	ULB Interface Address
33	ULB_A[19]	BFNNQLX	ULB Interface Address
56	ULB_A[1]	BFNNQLX	ULB Interface Address
32	ULB_A[20]	BFNNQLX	ULB Interface Address
55	ULB_A[2]	BFNNQLX	ULB Interface Address
54	ULB_A[3]	BFNNQLX	ULB Interface Address
53	ULB_A[4]	BFNNQLX	ULB Interface Address
52	ULB_A[5]	BFNNQLX	ULB Interface Address
51	ULB_A[6]	BFNNQLX	ULB Interface Address
49	ULB_A[7]	BFNNQLX	ULB Interface Address

Table 1-2: MB87P2020-A pinning sorted by name

Pin	Name	Buffer type	Description
48	ULB_A[8]	BFNNQLX	ULB Interface Address
47	ULB_A[9]	BFNNQLX	ULB Interface Address
44	ULB_CLK	ITFHX	ULB Interface Clock
58	ULB_CS	BFNNQLX	ULB Interface Chip Select
112	ULB_D[0]	BFNNQMX	ULB Interface Data
95	ULB_D[10]	BFNNQMX	ULB Interface Data
94	ULB_D[11]	BFNNQMX	ULB Interface Data
93	ULB_D[12]	BFNNQMX	ULB Interface Data
90	ULB_D[13]	BFNNQMX	ULB Interface Data
89	ULB_D[14]	BFNNQMX	ULB Interface Data
88	ULB_D[15]	BFNNQMX	ULB Interface Data
87	ULB_D[16]	BFNNQMX	ULB Interface Data
84	ULB_D[17]	BFNNQMX	ULB Interface Data
83	ULB_D[18]	BFNNQMX	ULB Interface Data
82	ULB_D[19]	BFNNQMX	ULB Interface Data
111	ULB_D[1]	BFNNQMX	ULB Interface Data
81	ULB_D[20]	BFNNQMX	ULB Interface Data
78	ULB_D[21]	BFNNQMX	ULB Interface Data
77	ULB_D[22]	BFNNQMX	ULB Interface Data
76	ULB_D[23]	BFNNQMX	ULB Interface Data
75	ULB_D[24]	BFNNQMX	ULB Interface Data
71	ULB_D[25]	BFNNQMX	ULB Interface Data
70	ULB_D[26]	BFNNQMX	ULB Interface Data
69	ULB_D[27]	BFNNQMX	ULB Interface Data
68	ULB_D[28]	BFNNQMX	ULB Interface Data
65	ULB_D[29]	BFNNQMX	ULB Interface Data
108	ULB_D[2]	BFNNQMX	ULB Interface Data
64	ULB_D[30]	BFNNQMX	ULB Interface Data
63	ULB_D[31]	BFNNQMX	ULB Interface Data
106	ULB_D[3]	BFNNQMX	ULB Interface Data
105	ULB_D[4]	BFNNQMX	ULB Interface Data
104	ULB_D[5]	BFNNQMX	ULB Interface Data
101	ULB_D[6]	BFNNQMX	ULB Interface Data

Table 1-2: MB87P2020-A pinning sorted by name

Pin	Name	Buffer type	Description
100	ULB_D[7]	BFNNQMX	ULB Interface Data
99	ULB_D[8]	BFNNQMX	ULB Interface Data
98	ULB_D[9]	BFNNQMX	ULB Interface Data
62	ULB_DACK	BFNNQLX	ULB Interface DMA Acknowledge
120	ULB_DREQ	OTFTQMX	ULB Interface DMA Request
117	ULB_DSTP	BFNNQMX	ULB Interface DMA Stop
121	ULB_INTRQ	OTFTQMX	ULB Interface Interrupt Request
59	ULB_RDY	BFNNQLX	ULB Interface Read
116	ULB_RDY	OTFTQMX	ULB Interface Ready
122	ULB_WRX[0]	ITFHX	ULB Interface Write Enable (D[31:24])
124	ULB_WRX[1]	ITFHX	ULB Interface Write Enable (D[23:16])
126	ULB_WRX[2]	ITFHX	ULB Interface Write Enable (D[15:8])
127	ULB_WRX[3]	ITFHX	ULB Interface Write Enable (D[7:0])
43	VDDE	VDDE#	IO supply 3.3V
61	VDDE	VDDE#	IO supply 3.3V
97	VDDE	VDDE#	IO supply 3.3V
114	VDDE	VDDE#	IO supply 3.3V
147	VDDE	VDDE#	IO supply 3.3V
165	VDDE	VDDE#	IO supply 3.3V
201	VDDE	VDDE#	IO supply 3.3V
24	VDDE[0]		IO supply 3.3V
207	VDDE[10]		IO supply 3.3V
67	VDDE[1]		IO supply 3.3V
74	VDDE[2]		IO supply 3.3V
92	VDDE[3]		IO supply 3.3V
102	VDDE[4]		IO supply 3.3V
109	VDDE[5]		IO supply 3.3V
129	VDDE[7]		IO supply 3.3V
155	VDDE[8]		IO supply 3.3V
178	VDDE[9]		IO supply 3.3V
19	VDDI	VDDI#	Core supply 2.5 V
27	VDDI	VDDI#	Core supply 2.5 V
34	VDDI	VDDI#	Core supply 2.5 V

Table 1-2: MB87P2020-A pinning sorted by name

Pin	Name	Buffer type	Description
73	VDDI	VDDI#	Core supply 2.5 V
79	VDDI	VDDI#	Core supply 2.5 V
86	VDDI	VDDI#	Core supply 2.5 V
123	VDDI	VDDI#	Core supply 2.5 V
131	VDDI	VDDI#	Core supply 2.5 V
138	VDDI	VDDI#	Core supply 2.5 V
177	VDDI	VDDI#	Core supply 2.5 V
183	VDDI	VDDI#	Core supply 2.5 V
190	VDDI	VDDI#	Core supply 2.5 V
134	VPD	VPDX	Fujitsu Tester Pin
17	VREF	ITAMX	DAC test pin VREF
197	VSC_ALPHA	BFNNQLX	Video Scaler ALPHA
21	VSC_CLKV	ITFHX	Video Scaler Clock
171	VSC_D[0]	BFNNQLX	Video Scaler Data Input
187	VSC_D[10]	BFNNQLX	Video Scaler Data Input
188	VSC_D[11]	BFNNQLX	Video Scaler Data Input
191	VSC_D[12]	BFNNQLX	Video Scaler Data Input
192	VSC_D[13]	BFNNQLX	Video Scaler Data Input
193	VSC_D[14]	BFNNQLX	Video Scaler Data Input
194	VSC_D[15]	BFNNQLX	Video Scaler Data Input
172	VSC_D[1]	BFNNQLX	Video Scaler Data Input
173	VSC_D[2]	BFNNQLX	Video Scaler Data Input
174	VSC_D[3]	BFNNQLX	Video Scaler Data Input
175	VSC_D[4]	BFNNQLX	Video Scaler Data Input
181	VSC_D[5]	BFNNQLX	Video Scaler Data Input
182	VSC_D[6]	BFNNQLX	Video Scaler Data Input
184	VSC_D[7]	BFNNQLX	Video Scaler Data Input
185	VSC_D[8]	BFNNQLX	Video Scaler Data Input
186	VSC_D[9]	BFNNQLX	Video Scaler Data Input
198	VSC_IDENT	BFNNQLX	Video Scaler Field identification
196	VSC_VACT	BFNNQLX	Video Scaler VACT
195	VSC_VREF	BFNNQLX	Video Scaler Vertical Reference

## 1.1.2 Buffer types

Table 1-3 shows all used buffers for MB87P2020-A.

**Table 1-3:** Buffer types for MB87P2020-A

Buffer type	Description
B3NNLMR2X	Bidirectional True buffer (3.3V CMOS, IOL=4mA, Low Noise type, ESD improved)
B3NNNMR2X	Bidirectional True buffer (3.3V CMOS, IOL=4mA, ESD improved)
BFNNQHX	Bidirectional True buffer (5V Tolerant, IOL=8mA, High speed type)
BFNNQLX	Bidirectional True buffer (5V Tolerant, IOL=2mA, High speed type)
BFNNQMX	5V tolerant, bidirectional true buffer 3.3V CMOS, IOL/IOH=4mA
BX4MFSR2X	Oscillator Output (ESD improved)
IPBIX	Input True Buffer for DRAM TEST (2.5V CMOS with 25K Pull-up) (SDRAM test only)
ITAMX	Analog Input buffer
ITAVDX	Analog Power Supply
ITAVSX	Analog GND
ITBSTX	Input True Buffer for DRAM TEST (2.5V CMOS with 25K Pull-down) (SDRAM test only)
ITCHX	Input True buffer (2.5V CMOS)
ITFHX	5V tolerant 3.3V CMOS Input
ITTSTX	Input True buffer for DRAM TEST Control (2.5V CMOS with 25K Pull-down)
IXN3X	Oscillator Input (ESD improved)
OTAMX	Analog Output
OTFTQMX	5 V tolerant 3.3V tri-state output, IOL/IOH=4mA
VPDX	3.3V CMOS input, disable input for Pull up/down resistors, connect to GND

## 1.2 Pinning for MB87P2020

### 1.2.1 Pinning

Table 1-4 shows the pinning for MB87P2020 (Jasmine) sorted by pin number while table 1-5 is sorted by names. Both tables refer to a standard Fujitsu 208-pin (L)QFP package FPT-208P-Mxx.

**Table 1-4:** MB87P2020 pinning sorted by pin number

Pin	Name	Buffer type	Description
1	MODE[0]	BFNNQLX	Mode Pin
2	MODE[1]	BFNNQLX	Mode Pin
3	GND	GND	

**Table 1-4:** MB87P2020 pinning sorted by pin number

Pin	Name	Buffer type	Description
4			Place holder for analog area
5	A_GREEN	OTAMX	Analog Green
6	DAC2_VSSA1	ITAVSX	DAC Ground
7	DAC2_VDDA1	ITAVDX	DAC Supply 2.5V
8	A_BLUE	OTAMX	Analog Blue
9	DAC1_VDDA1	ITAVDX	DAC Supply 2.5V
10	DAC1_VSSA1	ITAVSX	DAC Ground
11	A_VRO	OTAMX	DAC Full Scale Adjust
12	DAC1_VSSA	ITAVSX	DAC Ground
13	DAC1_VDDA	ITAVDX	DAC Supply 2.5V
14	A_RED	OTAMX	Analog Red
15	DAC3_VDDA1	ITAVDX	DAC Supply 2.5V
16	DAC3_VSSA1	ITAVSX	DAC Ground
17	VREF	ITAMX	DAC test pin VREF
18			Place holder for analog area
19	VDDI	VDDI#	Core supply 2.5 V
20	TEST	ITCHX	Fujitsu test pin
21	VSC_CLKV	ITFHX	Video Scaler Clock
22	RESETX	ITFUHX	Reset (pull up)
23	OSC_OUT	YB002AAX	XTAL output
24	VDDE[0]		IO supply 3.3V
25	OSC_IN	YI002AEX	XTAL input
26	GND	GND	
27	VDDI	VDDI#	Core supply 2.5 V
28	RDY_TRIEN	B3NNLMX	Control RDY pin behaviour
29	APLL_AVDD		APLL supply 2.5V
30	APLL_AVSS		APLL GND
31	RCLK	ITFHX	Reserved clock
32	ULB_A[20]	BFNNQLX	ULB Interface Address
33	ULB_A[19]	BFNNQLX	ULB Interface Address
34	VDDI	VDDI#	Core supply 2.5 V
35	ULB_A[18]	BFNNQLX	ULB Interface Address
36	ULB_A[17]	BFNNQLX	ULB Interface Address

**Table 1-4:** MB87P2020 pinning sorted by pin number

Pin	Name	Buffer type	Description
37	ULB_A[16]	BFNNQLX	ULB Interface Address
38	GND	GND	GND
39	ULB_A[15]	BFNNQLX	ULB Interface Address
40	ULB_A[14]	BFNNQLX	ULB Interface Address
41	ULB_A[13]	BFNNQLX	ULB Interface Address
42	ULB_A[12]	BFNNQLX	ULB Interface Address
43	VDDE	VDDE#	IO supply 3.3V
44	ULB_CLK	ITFHX	ULB Interface Clock
45	ULB_A[11]	BFNNQLX	ULB Interface Address
46	ULB_A[10]	BFNNQLX	ULB Interface Address
47	ULB_A[9]	BFNNQLX	ULB Interface Address
48	ULB_A[8]	BFNNQLX	ULB Interface Address
49	ULB_A[7]	BFNNQLX	ULB Interface Address
50	GND	GND	GND
51	ULB_A[6]	BFNNQLX	ULB Interface Address
52	ULB_A[5]	BFNNQLX	ULB Interface Address
53	ULB_A[4]	BFNNQLX	ULB Interface Address
54	ULB_A[3]	BFNNQLX	ULB Interface Address
55	ULB_A[2]	BFNNQLX	ULB Interface Address
56	ULB_A[1]	BFNNQLX	ULB Interface Address
57	ULB_A[0]	BFNNQLX	ULB Interface Address
58	ULB_CS	BFNNQLX	ULB Interface Chip Select
59	ULB_RDY	BFNNQLX	ULB Interface Read
60	GND	GND	GND
61	VDDE	VDDE#	IO supply 3.3V
62	ULB_DACK	BFNNQLX	ULB Interface DMA Acknowledge
63	ULB_D[31]	BFNNQMX	ULB Interface Data
64	ULB_D[30]	BFNNQMX	ULB Interface Data
65	ULB_D[29]	BFNNQMX	ULB Interface Data
66	GND[1]		GND
67	VDDE[1]		IO supply 3.3V
68	ULB_D[28]	BFNNQMX	ULB Interface Data
69	ULB_D[27]	BFNNQMX	ULB Interface Data



**Table 1-4:** MB87P2020 pinning sorted by pin number

Pin	Name	Buffer type	Description
70	ULB_D[26]	BFNNQMX	ULB Interface Data
71	ULB_D[25]	BFNNQMX	ULB Interface Data
72	GND	GND	GND
73	VDDI	VDDI#	Core supply 2.5 V
74	VDDE[2]		IO supply 3.3V
75	ULB_D[24]	BFNNQMX	ULB Interface Data
76	ULB_D[23]	BFNNQMX	ULB Interface Data
77	ULB_D[22]	BFNNQMX	ULB Interface Data
78	ULB_D[21]	BFNNQMX	ULB Interface Data
79	VDDI	VDDI#	Core supply 2.5 V
80	GND[2]		GND
81	ULB_D[20]	BFNNQMX	ULB Interface Data
82	ULB_D[19]	BFNNQMX	ULB Interface Data
83	ULB_D[18]	BFNNQMX	ULB Interface Data
84	ULB_D[17]	BFNNQMX	ULB Interface Data
85	GND	GND	GND
86	VDDI	VDDI#	Core supply 2.5 V
87	ULB_D[16]	BFNNQMX	ULB Interface Data
88	ULB_D[15]	BFNNQMX	ULB Interface Data
89	ULB_D[14]	BFNNQMX	ULB Interface Data
90	ULB_D[13]	BFNNQMX	ULB Interface Data
91	GND[3]		GND
92	VDDE[3]		IO supply 3.3V
93	ULB_D[12]	BFNNQMX	ULB Interface Data
94	ULB_D[11]	BFNNQMX	ULB Interface Data
95	ULB_D[10]	BFNNQMX	ULB Interface Data
96	GND	GND	GND
97	VDDE	VDDE#	IO supply 3.3V
98	ULB_D[9]	BFNNQMX	ULB Interface Data
99	ULB_D[8]	BFNNQMX	ULB Interface Data
100	ULB_D[7]	BFNNQMX	ULB Interface Data
101	ULB_D[6]	BFNNQMX	ULB Interface Data
102	VDDE[4]		IO supply 3.3V

**Table 1-4:** MB87P2020 pinning sorted by pin number

Pin	Name	Buffer type	Description
103	GND[4]		GND
104	ULB_D[5]	BFNNQMX	ULB Interface Data
105	ULB_D[4]	BFNNQMX	ULB Interface Data
106	ULB_D[3]	BFNNQMX	ULB Interface Data
107	GND	GND	GND
108	ULB_D[2]	BFNNQMX	ULB Interface Data
109	VDDE[5]		IO supply 3.3V
110	SDRAM_VCC[0]		SDRAM supply 2.5V
111	ULB_D[1]	BFNNQMX	ULB Interface Data
112	ULB_D[0]	BFNNQMX	ULB Interface Data
113	GND[5]		GND
114	VDDE	VDDE#	IO supply 3.3V
115	SDRAM_VCC[1]		SDRAM supply 2.5V
116	ULB_RDY	OTFTQMX	ULB Interface Ready
117	ULB_DSTP	BFNNQMX	ULB Interface DMA Stop
118	SDRAM_VCC[2]		SDRAM supply 2.5V
119	GND	GND	GND
120	ULB_DREQ	OTFTQMX	ULB Interface DMA Request
121	ULB_INTRQ	OTFTQMX	ULB Interface Interrupt Request
122	ULB_WRX[0]	ITFHX	ULB Interface Write Enable (D[31:24])
123	VDDI	VDDI#	Core supply 2.5 V
124	ULB_WRX[1]	ITFHX	ULB Interface Write Enable (D[23:16])
125	SDRAM_VCC[3]		SDRAM supply 2.5V
126	ULB_WRX[2]	ITFHX	ULB Interface Write Enable (D[15:8])
127	ULB_WRX[3]	ITFHX	ULB Interface Write Enable (D[7:0])
128	SDRAM_PBI	IPBIX	SDRAM Test mode
129	VDDE[7]		IO supply 3.3V
130	GND	GND	GND
131	VDDI	VDDI#	Core supply 2.5 V
132	SDRAM_TBST	ITBSTX	SDRAM test mode
133	SDRAM_TTST	ITTSTX	SDRAM test mode
134	VPD	VPDX	Fujitsu Tester Pin
135	DIS_D[0]	B3NNLMX	Display Data

**Table 1-4:** MB87P2020 pinning sorted by pin number

Pin	Name	Buffer type	Description
136	DIS_D[1]	B3NNLMX	Display Data
137	DIS_D[2]	B3NNLMX	Display Data
138	VDDI	VDDI#	Core supply 2.5 V
139	DIS_D[3]	B3NNLMX	Display Data
140	DIS_D[4]	B3NNLMX	Display Data
141	DIS_D[5]	B3NNLMX	Display Data
142	GND	GND	GND
143	DIS_D[6]	B3NNLMX	Display Data
144	DIS_D[7]	B3NNLMX	Display Data
145	DIS_D[8]	B3NNLMX	Display Data
146	DIS_D[9]	B3NNLMX	Display Data
147	VDDE	VDDE#	IO supply 3.3V
148	DIS_D[10]	B3NNLMX	Display Data
149	DIS_D[11]	B3NNLMX	Display Data
150	DIS_D[12]	B3NNLMX	Display Data
151	DIS_D[13]	B3NNLMX	Display Data
152	DIS_D[14]	B3NNLMX	Display Data
153	DIS_D[15]	B3NNLMX	Display Data
154	GND	GND	GND
155	VDDE[8]		IO supply 3.3V
156	DIS_D[16]	B3NNLMX	Display Data
157	DIS_D[17]	B3NNLMX	Display Data
158	DIS_D[18]	B3NNLMX	Display Data
159	DIS_D[19]	B3NNLMX	Display Data
160	DIS_D[20]	B3NNLMX	Display Data
161	DIS_D[21]	B3NNLMX	Display Data
162	DIS_D[22]	B3NNLMX	Display Data
163	DIS_D[23]	B3NNLMX	Display Data
164	GND	GND	GND
165	VDDE	VDDE#	IO supply 3.3V
166	DIS_CKEY	B3NNLMX	Display Colour Key
167	DIS_PIXCLK	B3NNLMX	Display Pixel Clock (programmable in/out)
168	DIS_VSYNC	B3NNLMX	Display programmable sync

**Table 1-4:** MB87P2020 pinning sorted by pin number

Pin	Name	Buffer type	Description
169	DIS_HSYNC	B3NNLMX	Display programmable sync
170	DIS_VREF	B3NNLMX	Display programmable sync
171	VSC_D[0]	BFNNQLX	Video Scaler Data Input
172	VSC_D[1]	BFNNQLX	Video Scaler Data Input
173	VSC_D[2]	BFNNQLX	Video Scaler Data Input
174	VSC_D[3]	BFNNQLX	Video Scaler Data Input
175	VSC_D[4]	BFNNQLX	Video Scaler Data Input
176	GND	GND	GND
177	VDDI	VDDI#	Core supply 2.5 V
178	VDDE[9]		IO supply 3.3V
179	MODE[2]	BFNNQLX	Mode Pin
180	MODE[3]	BFNNQLX	Mode Pin
181	VSC_D[5]	BFNNQLX	Video Scaler Data Input
182	VSC_D[6]	BFNNQLX	Video Scaler Data Input
183	VDDI	VDDI#	Core supply 2.5 V
184	VSC_D[7]	BFNNQLX	Video Scaler Data Input
185	VSC_D[8]	BFNNQLX	Video Scaler Data Input
186	VSC_D[9]	BFNNQLX	Video Scaler Data Input
187	VSC_D[10]	BFNNQLX	Video Scaler Data Input
188	VSC_D[11]	BFNNQLX	Video Scaler Data Input
189	GND	GND	GND
190	VDDI	VDDI#	Core supply 2.5 V
191	VSC_D[12]	BFNNQLX	Video Scaler Data Input
192	VSC_D[13]	BFNNQLX	Video Scaler Data Input
193	VSC_D[14]	BFNNQLX	Video Scaler Data Input
194	VSC_D[15]	BFNNQLX	Video Scaler Data Input
195	VSC_VREF	BFNNQLX	Video Scaler Vertical Reference
196	VSC_VACT	BFNNQLX	Video Scaler VACT
197	VSC_ALPHA	BFNNQLX	Video Scaler ALPHA
198	VSC_IDENT	BFNNQLX	Video Scaler Field identification
199	SPB_BUS	BFNNQHX	SPB Interface
200	GND	GND	GND
201	VDDE	VDDE#	IO supply 3.3V

**Table 1-4:** MB87P2020 pinning sorted by pin number

Pin	Name	Buffer type	Description
202	SPB_TST	BFNNQHX	SPB Test
203	CCFL_FET2	OTFTQMX	CCFL FET driver
204	CCFL_FET1	OTFTQMX	CCFL FET driver
205	CCFL_IGNIT	OTFTQMX	CCFL supply control IGNITION
206	GND[6]		GND
207	VDDE[10]		IO supply 3.3V
208	CCFL_OFF	OTFTQMX	CCFL supply control OFF

**Table 1-5:** MB87P2020 pinning sorted by name

Pin	Name	Buffer type	Description
4			Place holder for analog area
18			Place holder for analog area
8	A_BLUE	OTAMX	Analog Blue
5	A_GREEN	OTAMX	Analog Green
14	A_RED	OTAMX	Analog Red
11	A_VRO	OTAMX	DAC Full Scale Adjust
29	APLL_AVDD		APLL supply 2.5V
30	APLL_AVSS		APLL GND
204	CCFL_FET1	OTFTQMX	CCFL FET driver
203	CCFL_FET2	OTFTQMX	CCFL FET driver
205	CCFL_IGNIT	OTFTQMX	CCFL supply control IGNITION
208	CCFL_OFF	OTFTQMX	CCFL supply control OFF
13	DAC1_VDDA	ITAVDX	DAC Supply 2.5V
9	DAC1_VDDA1	ITAVDX	DAC Supply 2.5V
12	DAC1_VSSA	ITAVSX	DAC Ground
10	DAC1_VSSA1	ITAVSX	DAC Ground
7	DAC2_VDDA1	ITAVDX	DAC Supply 2.5V
6	DAC2_VSSA1	ITAVSX	DAC Ground
15	DAC3_VDDA1	ITAVDX	DAC Supply 2.5V
16	DAC3_VSSA1	ITAVSX	DAC Ground
166	DIS_CKEY	B3NNLMX	Display Colour Key
135	DIS_D[0]	B3NNLMX	Display Data

**Table 1-5:** MB87P2020 pinning sorted by name

Pin	Name	Buffer type	Description
148	DIS_D[10]	B3NNLMX	Display Data
149	DIS_D[11]	B3NNLMX	Display Data
150	DIS_D[12]	B3NNLMX	Display Data
151	DIS_D[13]	B3NNLMX	Display Data
152	DIS_D[14]	B3NNLMX	Display Data
153	DIS_D[15]	B3NNLMX	Display Data
156	DIS_D[16]	B3NNLMX	Display Data
157	DIS_D[17]	B3NNLMX	Display Data
158	DIS_D[18]	B3NNLMX	Display Data
159	DIS_D[19]	B3NNLMX	Display Data
136	DIS_D[1]	B3NNLMX	Display Data
160	DIS_D[20]	B3NNLMX	Display Data
161	DIS_D[21]	B3NNLMX	Display Data
162	DIS_D[22]	B3NNLMX	Display Data
163	DIS_D[23]	B3NNLMX	Display Data
137	DIS_D[2]	B3NNLMX	Display Data
139	DIS_D[3]	B3NNLMX	Display Data
140	DIS_D[4]	B3NNLMX	Display Data
141	DIS_D[5]	B3NNLMX	Display Data
143	DIS_D[6]	B3NNLMX	Display Data
144	DIS_D[7]	B3NNLMX	Display Data
145	DIS_D[8]	B3NNLMX	Display Data
146	DIS_D[9]	B3NNLMX	Display Data
169	DIS_HSYNC	B3NNLMX	Display programmable sync
167	DIS_PIXCLK	B3NNLMX	Display Pixel Clock (programmable in/out)
170	DIS_VREF	B3NNLMX	Display programmable sync
168	DIS_VSYNC	B3NNLMX	Display programmable sync
3	GND	GND	
26	GND	GND	
38	GND	GND	GND
50	GND	GND	GND
60	GND	GND	GND
72	GND	GND	GND

**Table 1-5:** MB87P2020 pinning sorted by name

Pin	Name	Buffer type	Description
85	GND	GND	GND
96	GND	GND	GND
107	GND	GND	GND
119	GND	GND	GND
130	GND	GND	GND
142	GND	GND	GND
154	GND	GND	GND
164	GND	GND	GND
176	GND	GND	GND
189	GND	GND	GND
200	GND	GND	GND
66	GND[1]		GND
80	GND[2]		GND
91	GND[3]		GND
103	GND[4]		GND
113	GND[5]		GND
206	GND[6]		GND
1	MODE[0]	BFNNQLX	Mode Pin
2	MODE[1]	BFNNQLX	Mode Pin
179	MODE[2]	BFNNQLX	Mode Pin
180	MODE[3]	BFNNQLX	Mode Pin
25	OSC_IN	YI002AEX	XTAL input
23	OSC_OUT	YB002AAX	XTAL output
31	RCLK	ITFHX	Reserved clock
28	RDY_TRIEN	B3NNLMX	Control RDY pin behaviour
22	RESETX	ITFUHX	Reset (pull up)
128	SDRAM_PBI	IPBIX	SDRAM Test mode
132	SDRAM_TBST	ITBSTX	SDRAM test mode
133	SDRAM_TTST	ITTSTX	SDRAM test mode
110	SDRAM_VCC[0]		SDRAM supply 2.5V
115	SDRAM_VCC[1]		SDRAM supply 2.5V
118	SDRAM_VCC[2]		SDRAM supply 2.5V
125	SDRAM_VCC[3]		SDRAM supply 2.5V

Table 1-5: MB87P2020 pinning sorted by name

Pin	Name	Buffer type	Description
199	SPB_BUS	BFNNQHX	SPB Interface
202	SPB_TST	BFNNQHX	SPB Test
20	TEST	ITCHX	Fujitsu test pin
57	ULB_A[0]	BFNNQLX	ULB Interface Address
46	ULB_A[10]	BFNNQLX	ULB Interface Address
45	ULB_A[11]	BFNNQLX	ULB Interface Address
42	ULB_A[12]	BFNNQLX	ULB Interface Address
41	ULB_A[13]	BFNNQLX	ULB Interface Address
40	ULB_A[14]	BFNNQLX	ULB Interface Address
39	ULB_A[15]	BFNNQLX	ULB Interface Address
37	ULB_A[16]	BFNNQLX	ULB Interface Address
36	ULB_A[17]	BFNNQLX	ULB Interface Address
35	ULB_A[18]	BFNNQLX	ULB Interface Address
33	ULB_A[19]	BFNNQLX	ULB Interface Address
56	ULB_A[1]	BFNNQLX	ULB Interface Address
32	ULB_A[20]	BFNNQLX	ULB Interface Address
55	ULB_A[2]	BFNNQLX	ULB Interface Address
54	ULB_A[3]	BFNNQLX	ULB Interface Address
53	ULB_A[4]	BFNNQLX	ULB Interface Address
52	ULB_A[5]	BFNNQLX	ULB Interface Address
51	ULB_A[6]	BFNNQLX	ULB Interface Address
49	ULB_A[7]	BFNNQLX	ULB Interface Address
48	ULB_A[8]	BFNNQLX	ULB Interface Address
47	ULB_A[9]	BFNNQLX	ULB Interface Address
44	ULB_CLK	ITFHX	ULB Interface Clock
58	ULB_CS	BFNNQLX	ULB Interface Chip Select
112	ULB_D[0]	BFNNQMX	ULB Interface Data
95	ULB_D[10]	BFNNQMX	ULB Interface Data
94	ULB_D[11]	BFNNQMX	ULB Interface Data
93	ULB_D[12]	BFNNQMX	ULB Interface Data
90	ULB_D[13]	BFNNQMX	ULB Interface Data
89	ULB_D[14]	BFNNQMX	ULB Interface Data
88	ULB_D[15]	BFNNQMX	ULB Interface Data



**Table 1-5:** MB87P2020 pinning sorted by name

Pin	Name	Buffer type	Description
87	ULB_D[16]	BFNNQMX	ULB Interface Data
84	ULB_D[17]	BFNNQMX	ULB Interface Data
83	ULB_D[18]	BFNNQMX	ULB Interface Data
82	ULB_D[19]	BFNNQMX	ULB Interface Data
111	ULB_D[1]	BFNNQMX	ULB Interface Data
81	ULB_D[20]	BFNNQMX	ULB Interface Data
78	ULB_D[21]	BFNNQMX	ULB Interface Data
77	ULB_D[22]	BFNNQMX	ULB Interface Data
76	ULB_D[23]	BFNNQMX	ULB Interface Data
75	ULB_D[24]	BFNNQMX	ULB Interface Data
71	ULB_D[25]	BFNNQMX	ULB Interface Data
70	ULB_D[26]	BFNNQMX	ULB Interface Data
69	ULB_D[27]	BFNNQMX	ULB Interface Data
68	ULB_D[28]	BFNNQMX	ULB Interface Data
65	ULB_D[29]	BFNNQMX	ULB Interface Data
108	ULB_D[2]	BFNNQMX	ULB Interface Data
64	ULB_D[30]	BFNNQMX	ULB Interface Data
63	ULB_D[31]	BFNNQMX	ULB Interface Data
106	ULB_D[3]	BFNNQMX	ULB Interface Data
105	ULB_D[4]	BFNNQMX	ULB Interface Data
104	ULB_D[5]	BFNNQMX	ULB Interface Data
101	ULB_D[6]	BFNNQMX	ULB Interface Data
100	ULB_D[7]	BFNNQMX	ULB Interface Data
99	ULB_D[8]	BFNNQMX	ULB Interface Data
98	ULB_D[9]	BFNNQMX	ULB Interface Data
62	ULB_DACK	BFNNQLX	ULB Interface DMA Acknowledge
120	ULB_DREQ	OTFTQMX	ULB Interface DMA Request
117	ULB_DSTP	BFNNQMX	ULB Interface DMA Stop
121	ULB_INTRQ	OTFTQMX	ULB Interface Interrupt Request
59	ULB_RDY	BFNNQLX	ULB Interface Read
116	ULB_RDY	OTFTQMX	ULB Interface Ready
122	ULB_WRX[0]	ITFHX	ULB Interface Write Enable (D[31:24])
124	ULB_WRX[1]	ITFHX	ULB Interface Write Enable (D[23:16])

Table 1-5: MB87P2020 pinning sorted by name

Pin	Name	Buffer type	Description
126	ULB_WRX[2]	ITFHX	ULB Interface Write Enable (D[15:8])
127	ULB_WRX[3]	ITFHX	ULB Interface Write Enable (D[7:0])
43	VDDE	VDDE#	IO supply 3.3V
61	VDDE	VDDE#	IO supply 3.3V
97	VDDE	VDDE#	IO supply 3.3V
114	VDDE	VDDE#	IO supply 3.3V
147	VDDE	VDDE#	IO supply 3.3V
165	VDDE	VDDE#	IO supply 3.3V
201	VDDE	VDDE#	IO supply 3.3V
24	VDDE[0]		IO supply 3.3V
207	VDDE[10]		IO supply 3.3V
67	VDDE[1]		IO supply 3.3V
74	VDDE[2]		IO supply 3.3V
92	VDDE[3]		IO supply 3.3V
102	VDDE[4]		IO supply 3.3V
109	VDDE[5]		IO supply 3.3V
129	VDDE[7]		IO supply 3.3V
155	VDDE[8]		IO supply 3.3V
178	VDDE[9]		IO supply 3.3V
19	VDDI	VDDI#	Core supply 2.5 V
27	VDDI	VDDI#	Core supply 2.5 V
34	VDDI	VDDI#	Core supply 2.5 V
73	VDDI	VDDI#	Core supply 2.5 V
79	VDDI	VDDI#	Core supply 2.5 V
86	VDDI	VDDI#	Core supply 2.5 V
123	VDDI	VDDI#	Core supply 2.5 V
131	VDDI	VDDI#	Core supply 2.5 V
138	VDDI	VDDI#	Core supply 2.5 V
177	VDDI	VDDI#	Core supply 2.5 V
183	VDDI	VDDI#	Core supply 2.5 V
190	VDDI	VDDI#	Core supply 2.5 V
134	VPD	VPDX	Fujitsu Tester Pin
17	VREF	ITAMX	DAC test pin VREF

**Table 1-5:** MB87P2020 pinning sorted by name

Pin	Name	Buffer type	Description
197	VSC_ALPHA	BFNNQLX	Video Scaler ALPHA
21	VSC_CLKV	ITFHX	Video Scaler Clock
171	VSC_D[0]	BFNNQLX	Video Scaler Data Input
187	VSC_D[10]	BFNNQLX	Video Scaler Data Input
188	VSC_D[11]	BFNNQLX	Video Scaler Data Input
191	VSC_D[12]	BFNNQLX	Video Scaler Data Input
192	VSC_D[13]	BFNNQLX	Video Scaler Data Input
193	VSC_D[14]	BFNNQLX	Video Scaler Data Input
194	VSC_D[15]	BFNNQLX	Video Scaler Data Input
172	VSC_D[1]	BFNNQLX	Video Scaler Data Input
173	VSC_D[2]	BFNNQLX	Video Scaler Data Input
174	VSC_D[3]	BFNNQLX	Video Scaler Data Input
175	VSC_D[4]	BFNNQLX	Video Scaler Data Input
181	VSC_D[5]	BFNNQLX	Video Scaler Data Input
182	VSC_D[6]	BFNNQLX	Video Scaler Data Input
184	VSC_D[7]	BFNNQLX	Video Scaler Data Input
185	VSC_D[8]	BFNNQLX	Video Scaler Data Input
186	VSC_D[9]	BFNNQLX	Video Scaler Data Input
198	VSC_IDENT	BFNNQLX	Video Scaler Field identification
196	VSC_VACT	BFNNQLX	Video Scaler VACT
195	VSC_VREF	BFNNQLX	Video Scaler Vertical Reference

### 1.2.2 Buffer types

Table 1-6 shows all used buffers for MB87P2020.

**Table 1-6:** Buffer types for MB87P2020

Buffer type	Description
B3NNLMX	Bidirectional True buffer (3.3V CMOS, IOL=4mA, Low Noise type)
B3NNNMX	Bidirectional True buffer (3.3V CMOS, IOL=4mA)
BFNNQHXX	Bidirectional True buffer (5V Tolerant, IOL=8mA, High speed type)
BFNNQLX	Bidirectional True buffer (5V Tolerant, IOL=2mA, High speed type)
BFNNQMX	5V tolerant, bidirectional true buffer 3.3V CMOS, IOL/IOH=4mA
IPBIX	Input True Buffer for DRAM TEST (2.5V CMOS with 25K Pull-up) (SDRAM test only)

**Table 1-6:** Buffer types for MB87P2020

Buffer type	Description
ITAMX	Analog Input buffer
ITAVDX	Analog Power Supply
ITAVSX	Analog GND
ITBSTX	Input True Buffer for DRAM TEST (2.5V CMOS with 25K Pull-down) (SDRAM test only)
ITCHX	Input True buffer (2.5V CMOS)
ITFHX	5V tolerant 3.3V CMOS Input
ITFUHX	5V tolerant 3.3V CMOS Input, 25 k Pull-up
ITTSTX	Input True buffer for DRAM TEST Control (2.5V CMOS with 25K Pull-down)
OTAMX	Analog Output
OTFTQMX	5 V tolerant 3.3V tri-state output, IOL/IOH=4mA
VPDX	3.3V CMOS input, disable input for Pull up/down resistors, connect to GND
YB002AAX	Oscillator Output
YI002AEX	Oscillator Input

## 1.3 Pinning for MB87J2120

### 1.3.1 Pinning

Table 1-7 shows the pinning for MB87J2120 (Lavender) sorted by pin number while table 1-8 is sorted by names. Both tables refer to a standard Fujitsu 256-pin BGA package (BGA-256P-M01).

**Table 1-7:** MB87J2120 pinning sorted by pin number

Pin	Name	Buffer Type	Description
1	GND	GND	GND
2	DAC3_VSSA1_2	ITAVSX	DAC Ground
3	A_BLUE	OTAMX	Analog Blue
4			Spacer, n.c.
5	A_GREEN	OTAMX	Analog Green
6	DAC1_VDDA	ITAVDX	DAC Supply 2.5V
7	VPD	VPDX	Fujitsu Tester Pin
8	DIS_D[1]	YB3NNLMX	Display Data
9	DIS_D[6]	YB3NNLMX	Display Data
10	DIS_D[8]	YB3NNLMX	Display Data

**Table 1-7:** MB87J2120 pinning sorted by pin number

Pin	Name	Buffer Type	Description
11	VDDE[0]	OVDD3X	IO supply 3.3V
12	DIS_D[13]	YB3NNLMX	Display Data
13	GND[0]	OVSSX	GND
14	DIS_D[14]	YB3NNLMX	Display Data
15	DIS_D[19]	YB3NNLMX	Display Data
16	GND[1]	OVSSX	GND
17	DIS_D[22]	YB3NNLMX	Display Data
18	DIS_VSYNC	YB3NNLMX	Display programmable sync
19	DIS_D[20]	YB3NNLMX	Display Data
20	GND	GND	GND
21	ULB_D[14]	BFNNQMX	ULB Interface Data
22	ULB_D[8]	BFNNQMX	ULB Interface Data
23	ULB_D[4]	BFNNQMX	ULB Interface Data
24	ULB_D[5]	BFNNQMX	ULB Interface Data
25	ULB_D[11]	BFNNQMX	ULB Interface Data
26	VDDE[9]	OVDD3X	IO supply 3.3V
27	ULB_D[20]	BFNNQMX	ULB Interface Data
28	ULB_WRX[2]	ITFHX	ULB Interface Byte 2 Write Enable
29	ULB_WRX[1]	ITFHX	ULB Interface Byte 1 Write Enable
30	ULB_D[26]	BFNNQMX	ULB Interface Data
31	ULB_D[30]	BFNNQMX	ULB Interface Data
32	ULB_D[25]	BFNNQMX	ULB Interface Data
33	ULB_D[31]	BFNNQMX	ULB Interface Data
34	ULB_CLK	ITFHX	ULB Interface Clock
35	ULB_A[7]	ITFHX	ULB Interface Address
36	ULB_WRX[3]	ITFHX	ULB Interface Byte 3 Write Enable
37	ULB_A[10]	ITFHX	ULB Interface Address
38	ULB_A[2]	ITFHX	ULB Interface Address
39	GND	GND	GND
40	ULB_INTRQ	OTFTQMX	ULB Interface Interrupt Output
41	GND[3]	OVSSX	GND
42	ULB_A[17]	ITFHX	ULB Interface Address
43	ULB_A[18]	ITFHX	ULB Interface Address

**Table 1-7:** MB87J2120 pinning sorted by pin number

Pin	Name	Buffer Type	Description
44	ULB_DACK	ITFHXX	ULB Interface DMA Acknowledge
45	SDC_D[3]	YB3NNLMX	SDRAM Data
46	SDC_D[7]	YB3NNLMX	SDRAM Data
47	SDC_D[11]	YB3NNLMX	SDRAM Data
48	VDDE[4]	OVDD3X	IO supply 3.3V
49	GND[10]	OVSSX	GND
50	SDC_D[15]	YB3NNLMX	SDRAM Data
51	SDC_CKE	YB3DNLMX	SDRAM Clock Enable
52	SDC_D[16]	YB3NNLMX	SDRAM Data
53	SDC_D[21]	YB3NNLMX	SDRAM Data
54	GND[5]	OVSSX	GND
55	SDC_A[0]	YB3NNLMX	SDRAM Address
56	SDC_D[28]	YB3NNLMX	SDRAM Data
57	SDC_D[22]	YB3NNLMX	SDRAM Data
58	GND	GND	GND
59	SDC_CAS	YB3NNLMX	SDRAM CAS
60	SDC_A[7]	YB3NNLMX	SDRAM Address
61	SDC_A[3]	YB3NNLMX	SDRAM Address
62	SDC_A[4]	YB3NNLMX	SDRAM Address
63	SDC_A[10]	YB3NNLMX	SDRAM Address
64	SBP_BUS	BFUNQHX	SPB Interface
65	MODE[3]	ITFUHX	GDC Mode Pin (pull up)
66	CCFL_FET2	OTFTQMX	CCFL FET driver
67	CCFL_IGNIT	OTFTQMX	CCFL supply control IGNITION
68	GND[11]	OVSSX	GND
69	MODE[0]	ITFUHX	GDC Mode Pin (pull up)
70	MODE[1]	ITFUHX	GDC Mode Pin (pull up)
71	AVDD[5]	APLL	APLL supply 2.5V
72	VSC_D[7]	ITFHXX	Video Scaler Data Input
73	VSC_ALPHA	ITFHXX	Video Scaler ALPHA
74	VSC_CLKV	ITFHXX	Video Scaler Clock
75	VSC_D[14]	ITFHXX	Video Scaler Data Input
76	VSC_D[6]	ITFHXX	Video Scaler Data Input

**Table 1-7:** MB87J2120 pinning sorted by pin number

Pin	Name	Buffer Type	Description
77	VSC_VREF	ITFHX	Video Scaler Vertical Reference
78	A_RED	OTAMX	Analog Red
79	DAC2_VSSA1_2	ITAVSX	DAC Ground
80	VSC_D[8]	ITFHX	Video Scaler Data Input
81	A_VRO	OTAVX	DAC Full Scale Adjust
82	VREF	ITAMX	DAC Testpin VREF
83	VSC_D[1]	ITFHX	Video Scaler Data Input
84	DIS_D[4]	YB3NNLMX	Display Data
85	DIS_D[5]	YB3NNLMX	Display Data
86	DIS_D[7]	YB3NNLMX	Display Data
87	DIS_D[15]	YB3NNLMX	Display Data
88	DIS_D[10]	YB3NNLMX	Display Data
89	DIS_D[16]	YB3NNLMX	Display Data
90	DIS_D[23]	YB3NNLMX	Display Data
91	DIS_D[2]	YB3NNLMX	Display Data
92	DIS_CK	YB3NNLMX	Display Color Key
93	ULB_WRX[0]	ITFHX	ULB Interface Byte 0 Write Enable
94	ULB_D[1]	BFNNQMX	ULB Interface Data
95	ULB_D[12]	BFNNQMX	ULB Interface Data
96	ULB_D[6]	BFNNQMX	ULB Interface Data
97	ULB_D[3]	BFNNQMX	ULB Interface Data
98	VDDE[2]	OVDD3X	IO supply 3.3V
99	ULB_D[15]	BFNNQMX	ULB Interface Data
100	ULB_D[18]	BFNNQMX	ULB Interface Data
101	ULB_D[22]	BFNNQMX	ULB Interface Data
102	ULB_D[23]	BFNNQMX	ULB Interface Data
103	ULB_D[24]	BFNNQMX	ULB Interface Data
104	ULB_CS	ITFHX	ULB Interface Chip Select
105	ULB_D[27]	BFNNQMX	ULB Interface Data
106	ULB_A[0]	ITFHX	ULB Interface Address
107	ULB_A[5]	ITFHX	ULB Interface Address
108	ULB_A[13]	ITFHX	ULB Interface Address
109	ULB_A[12]	ITFHX	ULB Interface Address

**Table 1-7:** MB87J2120 pinning sorted by pin number

Pin	Name	Buffer Type	Description
110	ULB_A[4]	ITFHX	ULB Interface Address
111	VDDE[10]	OVDD3X	IO supply 3.3V
112	ULB_DEOP	BFNNQMX	ULB Interface DMA Stop Output (DSTOP)
113	ULB_A[19]	ITFHX	ULB Interface Address
114	ULB_A[16]	ITFHX	ULB Interface Address
115	ULB_RDY	OTFTQMX	ULB Interface Ready
116	SDC_D[2]	YB3NNLMX	SDRAM Data
117	SDC_D[6]	YB3NNLMX	SDRAM Data
118	SDC_D[9]	YB3NNLMX	SDRAM Data
119	SDC_D[10]	YB3NNLMX	SDRAM Data
120	GND[4]	OVSSX	GND
121	SDC_D[17]	YB3NNLMX	SDRAM Data
122	SDC_D[12]	YB3NNLMX	SDRAM Data
123	SDC_D[18]	YB3NNLMX	SDRAM Data
124	SDC_D[25]	YB3NNLMX	SDRAM Data
125	SDC_D[31]	YB3NNLMX	SDRAM Data
126	SDC_D[30]	YB3NNLMX	SDRAM Data
127	SDC_D[24]	YB3NNLMX	SDRAM Data
128	SDC_D[20]	YB3NNLMX	SDRAM Data
129	SDC_A[11]	YB3NNLMX	SDRAM Address
130	SDC_A[5]	YB3NNLMX	SDRAM Address
131	SDC_A[2]	YB3NNLMX	SDRAM Address
132	VDDE[6]	OVDD3X	IO supply 3.3V
133	SDC_RAS	YB3NNLMX	SDRAM RAS
134	MODE[2]	ITFUHX	GDC Mode Pin (pull up)
135	VDDE[8]	OVDD3X	IO supply 3.3V
136	CCFL_FET1	OTFTQMX	CCFL FET driver
137	CCFL_OFF	OTFTQMX	CCFL supply control OFF
138	AVSS[5]	APLL	APLL GND
139	OSC_IN	YI002AEX	XTAL input
140	VSC_IDENT	ITFHX	Video Scaler Field identification
141	VSC_D[3]	ITFHX	Video Scaler Data Input



**Table 1-7:** MB87J2120 pinning sorted by pin number

Pin	Name	Buffer Type	Description
142	VSC_D[11]	ITFHX	Video Scaler Data Input
143	VSC_D[12]	ITFHX	Video Scaler Data Input
144	VSC_D[4]	ITFHX	Video Scaler Data Input
145			Spacer, n.c.
146	DAC1_VSSA	ITAVSX	DAC Ground
147	VSC_D[0]	ITFHX	Video Scaler Data Input
148	DAC2_VDDA1	ITAVDX	DAC Supply 2.5V
149	DAC3_VDDA1	ITAVDX	DAC Supply 2.5V
150	DIS_PIXCLK	B3NNNMX	Display Pixel Clock (programmable in/out)
151	DIS_D[0]	YB3NNLMX	Display Data
152	DIS_D[3]	YB3NNLMX	Display Data
153	DIS_D[11]	YB3NNLMX	Display Data
154	DIS_D[17]	YB3NNLMX	Display Data
155	DIS_D[12]	YB3NNLMX	Display Data
156	DIS_D[21]	YB3NNLMX	Display Data
157	DIS_VREF	YB3NNLMX	Display programmable sync
158	DIS_D[18]	YB3NNLMX	Display Data
159	ULB_D[0]	BFNNQMX	ULB Interface Data
160	ULB_D[16]	BFNNQMX	ULB Interface Data
161	ULB_D[10]	BFNNQMX	ULB Interface Data
162	ULB_D[2]	BFNNQMX	ULB Interface Data
163	ULB_D[7]	BFNNQMX	ULB Interface Data
164	ULB_D[13]	BFNNQMX	ULB Interface Data
165	ULB_D[17]	BFNNQMX	ULB Interface Data
166	ULB_D[19]	BFNNQMX	ULB Interface Data
167	ULB_D[21]	BFNNQMX	ULB Interface Data
168	ULB_D[28]	BFNNQMX	ULB Interface Data
169	ULB_A[1]	ITFHX	ULB Interface Address
170	ULB_D[29]	BFNNQMX	ULB Interface Data
171	ULB_A[3]	ITFHX	ULB Interface Address
172	ULB_A[11]	ITFHX	ULB Interface Address
173	ULB_A[14]	ITFHX	ULB Interface Address

**Table 1-7:** MB87J2120 pinning sorted by pin number

Pin	Name	Buffer Type	Description
174	ULB_A[6]	ITFHX	ULB Interface Address
175	ULB_A[20]	ITFHX	ULB Interface Address
176	ULB_DREQ	OTFTQMX	ULB Interface DMA Request
177	ULB_A[15]	ITFHX	ULB Interface Address
178	SDC_D[0]	YB3NNLMX	SDRAM Data
179	SDC_D[1]	YB3NNLMX	SDRAM Data
180	SDC_D[4]	YB3NNLMX	SDRAM Data
181	SDC_D[5]	YB3NNLMX	SDRAM Data
182	SDC_D[8]	YB3NNLMX	SDRAM Data
183	SDC_D[13]	YB3NNLMX	SDRAM Data
184	SDC_D[19]	YB3NNLMX	SDRAM Data
185	SDC_D[14]	YB3NNLMX	SDRAM Data
186	SDC_D[23]	YB3NNLMX	SDRAM Data
187	SDC_D[29]	YB3NNLMX	SDRAM Data
188	SDC_WE	YB3NNLMX	SDRAM Write Enable
189	SDC_D[26]	YB3NNLMX	SDRAM Data
190	SDC_DMQ	YB3NNLMX	SDRAM DQM
191	SDC_A[9]	YB3NNLMX	SDRAM Address
192	SDC_A[1]	YB3NNLMX	SDRAM Address
193	SDC_A[6]	YB3NNLMX	SDRAM Address
194	SDC_A[12]	YB3NNLMX	SDRAM Address
195	TEST	YB3DNLMX	Fujitsu Testpin
196	RSTX	ITFUHX	GDC Reset (pull up)
197	GND[8]	OVSSX	GND
198	VDDE[7]	OVDD3X	IO supply 3.3V
199	VSC_D[9]	ITFHX	Video Scaler Data Input
200	OSC_OUT	YB002AAX	XTAL output
201	VSC_D[5]	ITFHX	Video Scaler Data Input
202	VSC_D[13]	ITFHX	Video Scaler Data Input
203	VSC_D[10]	ITFHX	Video Scaler Data Input
204	VSC_D[2]	ITFHX	Video Scaler Data Input
205	GND	GND	GND
206	DAC1_VSSA1_2	ITAVSX	DAC Supply 2.5V

**Table 1-7:** MB87J2120 pinning sorted by pin number

Pin	Name	Buffer Type	Description
207	VDDI	VDDI#	Core supply 2.5 V
208	DAC1_VDDA1	ITAVDX	DAC Supply 2.5V
209	GND	GND	GND
210	VDDE	VDDE#5	IO supply 3.3V
211	VDDI	VDDI#	Core supply 2.5 V
212	DIS_D[9]	YB3NNLMX	Display Data
213	VDDE	VDDE#	IO supply 3.3V
214	GND	GND	GND
215	DIS_HSYNC	YB3NNLMX	Display programmable sync
216	VDDE	VDDE#	IO supply 3.3V
217	VDDE[1]	OVDD3X	IO supply 3.3V
218	GND	GND	GND
219	GND[2]	OVSSX	GND
220	VDDI	VDDI#	Core supply 2.5 V
221	ULB_D[9]	BFNNQMX	ULB Interface Data
222	GND	GND	GND
223	VDDI	VDDI#	Core supply 2.5 V
224	VDDE	VDDE#	IO supply 3.3V
225	GND[9]	OVSSX	GND
226	VDDI	VDDI#	Core supply 2.5 V
227	GND	GND	GND
228	ULB_A[9]	ITFHX	ULB Interface Address
229	VDDE	VDDE#	IO supply 3.3V
230	ULB_A[8]	ITFHX	ULB Interface Address
231	GND	GND	GND
232	ULB_RDY	ITFHX	ULB Interface Read
233	VDDI	VDDI#	Core supply 2.5 V
234	VDDE[3]	OVDD3X	IO supply 3.3V
235	GND	GND	GND
236	VDDE	VDDE#	IO supply 3.3V
237	VDDI	VDDI#	Core supply 2.5 V
238	SDC_CLK	B3NNNMXX	SDRAM Clock
239	VDDE	VDDE#	IO supply 3.3V

**Table 1-7:** MB87J2120 pinning sorted by pin number

Pin	Name	Buffer Type	Description
240	GND	GND	GND
241	SDC_D[27]	YB3NNLMX	SDRAM Data
242	VDDE	VDDE#	IO supply 3.3V
243	VDDE[5]	OVDD3X	IO supply 3.3V
244	GND	GND	GND
245	GND[6]	OVSSX	GND
246	VDDI	VDDI#	Core supply 2.5 V
247	SDC_A[8]	YB3NNLMX	SDRAM Address
248	GND	GND	GND
249	VDDI	VDDI#	Core supply 2.5 V
250	VDDE	VDDE#	IO supply 3.3V
251	GND[7]	OVSSX	GND
252	VDDI	VDDI#	Core supply 2.5 V
253	GND	GND	GND
254	VSC_D[15]	ITFHX	Video Scaler Data Input
255	VDDE	VDDE#	IO supply 3.3V
256	VSC_VACT	ITFHX	Video Scaler VACT

**Table 1-8:** MB87J2120 pinning sorted by name

Pin	Name	Buffer Type	Description
4			Spacer, n.c.
145			Spacer, n.c.
3	A_BLUE	OTAMX	Analog Blue
5	A_GREEN	OTAMX	Analog Green
78	A_RED	OTAMX	Analog Red
81	A_VRO	OTAVX	DAC Full Scale Adjust
71	AVDD[5]	APLL	APLL supply 2.5V
138	AVSS[5]	APLL	APLL GND
136	CCFL_FET1	OTFTQMX	CCFL FET driver
66	CCFL_FET2	OTFTQMX	CCFL FET driver
67	CCFL_IGNIT	OTFTQMX	CCFL supply control IGNITION
137	CCFL_OFF	OTFTQMX	CCFL supply control OFF

**Table 1-8:** MB87J2120 pinning sorted by name

Pin	Name	Buffer Type	Description
6	DAC1_VDDA	ITAVDX	DAC Supply 2.5V
208	DAC1_VDDA1	ITAVDX	DAC Supply 2.5V
146	DAC1_VSSA	ITAVSX	DAC Ground
206	DAC1_VSSA1_2	ITAVSX	DAC Supply 2.5V
148	DAC2_VDDA1	ITAVDX	DAC Supply 2.5V
79	DAC2_VSSA1_2	ITAVSX	DAC Ground
149	DAC3_VDDA1	ITAVDX	DAC Supply 2.5V
2	DAC3_VSSA1_2	ITAVSX	DAC Ground
92	DIS_CK	YB3NNLMX	Display Color Key
151	DIS_D[0]	YB3NNLMX	Display Data
88	DIS_D[10]	YB3NNLMX	Display Data
153	DIS_D[11]	YB3NNLMX	Display Data
155	DIS_D[12]	YB3NNLMX	Display Data
12	DIS_D[13]	YB3NNLMX	Display Data
14	DIS_D[14]	YB3NNLMX	Display Data
87	DIS_D[15]	YB3NNLMX	Display Data
89	DIS_D[16]	YB3NNLMX	Display Data
154	DIS_D[17]	YB3NNLMX	Display Data
158	DIS_D[18]	YB3NNLMX	Display Data
15	DIS_D[19]	YB3NNLMX	Display Data
8	DIS_D[1]	YB3NNLMX	Display Data
19	DIS_D[20]	YB3NNLMX	Display Data
156	DIS_D[21]	YB3NNLMX	Display Data
17	DIS_D[22]	YB3NNLMX	Display Data
90	DIS_D[23]	YB3NNLMX	Display Data
91	DIS_D[2]	YB3NNLMX	Display Data
152	DIS_D[3]	YB3NNLMX	Display Data
84	DIS_D[4]	YB3NNLMX	Display Data
85	DIS_D[5]	YB3NNLMX	Display Data
9	DIS_D[6]	YB3NNLMX	Display Data
86	DIS_D[7]	YB3NNLMX	Display Data
10	DIS_D[8]	YB3NNLMX	Display Data
212	DIS_D[9]	YB3NNLMX	Display Data

Table 1-8: MB87J2120 pinning sorted by name

Pin	Name	Buffer Type	Description
215	DIS_HSYNC	YB3NNLMX	Display programmable sync
150	DIS_PIXCLK	B3NNNMX	Display Pixel Clock (programmable in/out)
157	DIS_VREF	YB3NNLMX	Display programmable sync
18	DIS_VSYNC	YB3NNLMX	Display programmable sync
1	GND	GND	GND
20	GND	GND	GND
39	GND	GND	GND
58	GND	GND	GND
205	GND	GND	GND
209	GND	GND	GND
214	GND	GND	GND
218	GND	GND	GND
222	GND	GND	GND
227	GND	GND	GND
231	GND	GND	GND
235	GND	GND	GND
240	GND	GND	GND
244	GND	GND	GND
248	GND	GND	GND
253	GND	GND	GND
13	GND[0]	OVSSX	GND
49	GND[10]	OVSSX	GND
68	GND[11]	OVSSX	GND
16	GND[1]	OVSSX	GND
219	GND[2]	OVSSX	GND
41	GND[3]	OVSSX	GND
120	GND[4]	OVSSX	GND
54	GND[5]	OVSSX	GND
245	GND[6]	OVSSX	GND
251	GND[7]	OVSSX	GND
197	GND[8]	OVSSX	GND
225	GND[9]	OVSSX	GND

**Table 1-8:** MB87J2120 pinning sorted by name

Pin	Name	Buffer Type	Description
69	MODE[0]	ITFUHX	GDC Mode Pin (pull up)
70	MODE[1]	ITFUHX	GDC Mode Pin (pull up)
134	MODE[2]	ITFUHX	GDC Mode Pin (pull up)
65	MODE[3]	ITFUHX	GDC Mode Pin (pull up)
139	OSC_IN	YI002AEX	XTAL input
200	OSC_OUT	YB002AAX	XTAL output
196	RSTX	ITFUHX	GDC Reset (pull up)
64	SBP_BUS	BFUNQHX	SPB Interface
55	SDC_A[0]	YB3NNLMX	SDRAM Address
63	SDC_A[10]	YB3NNLMX	SDRAM Address
129	SDC_A[11]	YB3NNLMX	SDRAM Address
194	SDC_A[12]	YB3NNLMX	SDRAM Address
192	SDC_A[1]	YB3NNLMX	SDRAM Address
131	SDC_A[2]	YB3NNLMX	SDRAM Address
61	SDC_A[3]	YB3NNLMX	SDRAM Address
62	SDC_A[4]	YB3NNLMX	SDRAM Address
130	SDC_A[5]	YB3NNLMX	SDRAM Address
193	SDC_A[6]	YB3NNLMX	SDRAM Address
60	SDC_A[7]	YB3NNLMX	SDRAM Address
247	SDC_A[8]	YB3NNLMX	SDRAM Address
191	SDC_A[9]	YB3NNLMX	SDRAM Address
59	SDC_CAS	YB3NNLMX	SDRAM CAS
51	SDC_CKE	YB3DNLMX	SDRAM Clock Enable
238	SDC_CLK	B3NNNMX	SDRAM Clock
178	SDC_D[0]	YB3NNLMX	SDRAM Data
119	SDC_D[10]	YB3NNLMX	SDRAM Data
47	SDC_D[11]	YB3NNLMX	SDRAM Data
122	SDC_D[12]	YB3NNLMX	SDRAM Data
183	SDC_D[13]	YB3NNLMX	SDRAM Data
185	SDC_D[14]	YB3NNLMX	SDRAM Data
50	SDC_D[15]	YB3NNLMX	SDRAM Data
52	SDC_D[16]	YB3NNLMX	SDRAM Data
121	SDC_D[17]	YB3NNLMX	SDRAM Data

**Table 1-8:** MB87J2120 pinning sorted by name

Pin	Name	Buffer Type	Description
123	SDC_D[18]	YB3NNLMX	SDRAM Data
184	SDC_D[19]	YB3NNLMX	SDRAM Data
179	SDC_D[1]	YB3NNLMX	SDRAM Data
128	SDC_D[20]	YB3NNLMX	SDRAM Data
53	SDC_D[21]	YB3NNLMX	SDRAM Data
57	SDC_D[22]	YB3NNLMX	SDRAM Data
186	SDC_D[23]	YB3NNLMX	SDRAM Data
127	SDC_D[24]	YB3NNLMX	SDRAM Data
124	SDC_D[25]	YB3NNLMX	SDRAM Data
189	SDC_D[26]	YB3NNLMX	SDRAM Data
241	SDC_D[27]	YB3NNLMX	SDRAM Data
56	SDC_D[28]	YB3NNLMX	SDRAM Data
187	SDC_D[29]	YB3NNLMX	SDRAM Data
116	SDC_D[2]	YB3NNLMX	SDRAM Data
126	SDC_D[30]	YB3NNLMX	SDRAM Data
125	SDC_D[31]	YB3NNLMX	SDRAM Data
45	SDC_D[3]	YB3NNLMX	SDRAM Data
180	SDC_D[4]	YB3NNLMX	SDRAM Data
181	SDC_D[5]	YB3NNLMX	SDRAM Data
117	SDC_D[6]	YB3NNLMX	SDRAM Data
46	SDC_D[7]	YB3NNLMX	SDRAM Data
182	SDC_D[8]	YB3NNLMX	SDRAM Data
118	SDC_D[9]	YB3NNLMX	SDRAM Data
190	SDC_DMQ	YB3NNLMX	SDRAM DQM
133	SDC_RAS	YB3NNLMX	SDRAM RAS
188	SDC_WE	YB3NNLMX	SDRAM Write Enable
195	TEST	YB3DNLMX	Fujitsu Testpin
106	ULB_A[0]	ITFHX	ULB Interface Address
37	ULB_A[10]	ITFHX	ULB Interface Address
172	ULB_A[11]	ITFHX	ULB Interface Address
109	ULB_A[12]	ITFHX	ULB Interface Address
108	ULB_A[13]	ITFHX	ULB Interface Address
173	ULB_A[14]	ITFHX	ULB Interface Address



**Table 1-8:** MB87J2120 pinning sorted by name

Pin	Name	Buffer Type	Description
177	ULB_A[15]	ITFHX	ULB Interface Address
114	ULB_A[16]	ITFHX	ULB Interface Address
42	ULB_A[17]	ITFHX	ULB Interface Address
43	ULB_A[18]	ITFHX	ULB Interface Address
113	ULB_A[19]	ITFHX	ULB Interface Address
169	ULB_A[1]	ITFHX	ULB Interface Address
175	ULB_A[20]	ITFHX	ULB Interface Address
38	ULB_A[2]	ITFHX	ULB Interface Address
171	ULB_A[3]	ITFHX	ULB Interface Address
110	ULB_A[4]	ITFHX	ULB Interface Address
107	ULB_A[5]	ITFHX	ULB Interface Address
174	ULB_A[6]	ITFHX	ULB Interface Address
35	ULB_A[7]	ITFHX	ULB Interface Address
230	ULB_A[8]	ITFHX	ULB Interface Address
228	ULB_A[9]	ITFHX	ULB Interface Address
34	ULB_CLK	ITFHX	ULB Interface Clock
104	ULB_CS	ITFHX	ULB Interface Chip Select
159	ULB_D[0]	BFNNQMX	ULB Interface Data
161	ULB_D[10]	BFNNQMX	ULB Interface Data
25	ULB_D[11]	BFNNQMX	ULB Interface Data
95	ULB_D[12]	BFNNQMX	ULB Interface Data
164	ULB_D[13]	BFNNQMX	ULB Interface Data
21	ULB_D[14]	BFNNQMX	ULB Interface Data
99	ULB_D[15]	BFNNQMX	ULB Interface Data
160	ULB_D[16]	BFNNQMX	ULB Interface Data
165	ULB_D[17]	BFNNQMX	ULB Interface Data
100	ULB_D[18]	BFNNQMX	ULB Interface Data
166	ULB_D[19]	BFNNQMX	ULB Interface Data
94	ULB_D[1]	BFNNQMX	ULB Interface Data
27	ULB_D[20]	BFNNQMX	ULB Interface Data
167	ULB_D[21]	BFNNQMX	ULB Interface Data
101	ULB_D[22]	BFNNQMX	ULB Interface Data
102	ULB_D[23]	BFNNQMX	ULB Interface Data

Table 1-8: MB87J2120 pinning sorted by name

Pin	Name	Buffer Type	Description
103	ULB_D[24]	BFNNQMX	ULB Interface Data
32	ULB_D[25]	BFNNQMX	ULB Interface Data
30	ULB_D[26]	BFNNQMX	ULB Interface Data
105	ULB_D[27]	BFNNQMX	ULB Interface Data
168	ULB_D[28]	BFNNQMX	ULB Interface Data
170	ULB_D[29]	BFNNQMX	ULB Interface Data
162	ULB_D[2]	BFNNQMX	ULB Interface Data
31	ULB_D[30]	BFNNQMX	ULB Interface Data
33	ULB_D[31]	BFNNQMX	ULB Interface Data
97	ULB_D[3]	BFNNQMX	ULB Interface Data
23	ULB_D[4]	BFNNQMX	ULB Interface Data
24	ULB_D[5]	BFNNQMX	ULB Interface Data
96	ULB_D[6]	BFNNQMX	ULB Interface Data
163	ULB_D[7]	BFNNQMX	ULB Interface Data
22	ULB_D[8]	BFNNQMX	ULB Interface Data
221	ULB_D[9]	BFNNQMX	ULB Interface Data
44	ULB_DACK	ITFHX	ULB Interface DMA Acknowledge
112	ULB_DEOP	BFNNQMX	ULB Interface DMA Stop Output (DSTOP)
176	ULB_DREQ	OTFTQMX	ULB Interface DMA Request
40	ULB_INTRQ	OTFTQMX	ULB Interface Interrupt Output
232	ULB_RDY	ITFHX	ULB Interface Read
115	ULB_RDY	OTFTQMX	ULB Interface Ready
93	ULB_WRX[0]	ITFHX	ULB Interface Byte 0 Write Enable
29	ULB_WRX[1]	ITFHX	ULB Interface Byte 1 Write Enable
28	ULB_WRX[2]	ITFHX	ULB Interface Byte 2 Write Enable
36	ULB_WRX[3]	ITFHX	ULB Interface Byte 3 Write Enable
210	VDDE	VDDE#5	IO supply 3.3V
213	VDDE	VDDE#	IO supply 3.3V
216	VDDE	VDDE#	IO supply 3.3V
224	VDDE	VDDE#	IO supply 3.3V
229	VDDE	VDDE#	IO supply 3.3V
236	VDDE	VDDE#	IO supply 3.3V

**Table 1-8:** MB87J2120 pinning sorted by name

Pin	Name	Buffer Type	Description
239	VDDE	VDDE#	IO supply 3.3V
242	VDDE	VDDE#	IO supply 3.3V
250	VDDE	VDDE#	IO supply 3.3V
255	VDDE	VDDE#	IO supply 3.3V
11	VDDE[0]	OVDD3X	IO supply 3.3V
111	VDDE[10]	OVDD3X	IO supply 3.3V
217	VDDE[1]	OVDD3X	IO supply 3.3V
98	VDDE[2]	OVDD3X	IO supply 3.3V
234	VDDE[3]	OVDD3X	IO supply 3.3V
48	VDDE[4]	OVDD3X	IO supply 3.3V
243	VDDE[5]	OVDD3X	IO supply 3.3V
132	VDDE[6]	OVDD3X	IO supply 3.3V
198	VDDE[7]	OVDD3X	IO supply 3.3V
135	VDDE[8]	OVDD3X	IO supply 3.3V
26	VDDE[9]	OVDD3X	IO supply 3.3V
207	VDDI	VDDI#	Core supply 2.5 V
211	VDDI	VDDI#	Core supply 2.5 V
220	VDDI	VDDI#	Core supply 2.5 V
223	VDDI	VDDI#	Core supply 2.5 V
226	VDDI	VDDI#	Core supply 2.5 V
233	VDDI	VDDI#	Core supply 2.5 V
237	VDDI	VDDI#	Core supply 2.5 V
246	VDDI	VDDI#	Core supply 2.5 V
249	VDDI	VDDI#	Core supply 2.5 V
252	VDDI	VDDI#	Core supply 2.5 V
7	VPD	VPDX	Fujitsu Tester Pin
82	VREF	ITAMX	DAC Testpin VREF
73	VSC_ALPHA	ITFHX	Video Scaler ALPHA
74	VSC_CLKV	ITFHX	Video Scaler Clock
147	VSC_D[0]	ITFHX	Video Scaler Data Input
203	VSC_D[10]	ITFHX	Video Scaler Data Input
142	VSC_D[11]	ITFHX	Video Scaler Data Input
143	VSC_D[12]	ITFHX	Video Scaler Data Input

**Table 1-8:** MB87J2120 pinning sorted by name

Pin	Name	Buffer Type	Description
202	VSC_D[13]	ITFHX	Video Scaler Data Input
75	VSC_D[14]	ITFHX	Video Scaler Data Input
254	VSC_D[15]	ITFHX	Video Scaler Data Input
83	VSC_D[1]	ITFHX	Video Scaler Data Input
204	VSC_D[2]	ITFHX	Video Scaler Data Input
141	VSC_D[3]	ITFHX	Video Scaler Data Input
144	VSC_D[4]	ITFHX	Video Scaler Data Input
201	VSC_D[5]	ITFHX	Video Scaler Data Input
76	VSC_D[6]	ITFHX	Video Scaler Data Input
72	VSC_D[7]	ITFHX	Video Scaler Data Input
80	VSC_D[8]	ITFHX	Video Scaler Data Input
199	VSC_D[9]	ITFHX	Video Scaler Data Input
140	VSC_IDENT	ITFHX	Video Scaler Field identification
256	VSC_VACT	ITFHX	Video Scaler VACT
77	VSC_VREF	ITFHX	Video Scaler Vertical Reference

### 1.3.2 Buffer Types

Table 1-9 shows all used buffers for MB87J2120.

**Table 1-9:** Buffer types for MB87J2120

Buffer type	Description
B3NNLMX	Bidirectional True buffer (3.3V CMOS, IOL=4mA, Low Noise type)
B3NNNMX	Bidirectional True buffer (3.3V CMOS, IOL=4mA)
BFNNQMX	5V tolerant, bidirectional true buffer 3.3V CMOS, IOL/IOH=4mA
BFUNQHX	Bidirectional True buffer (5V Tolerant, 25K Pull-up, IOL=8mA, High speed type)
ITAMX	Analog Input buffer
ITAVDX	Analog Power Supply
ITAVSX	Analog GND
ITFHX	5V tolerant 3.3V CMOS Input
ITFUHX	5V tolerant 3.3V CMOS Input, 25 k Pull-up
OTAMX	Analog Output
OTAVX	Analog Output

**Table 1-9:** Buffer types for MB87J2120

Buffer type	Description
OTFTQMX	5 V tolerant 3.3V tri-state output, IOL/IOH=4mA
OVDD3X	Power Supply for 3.3V VDD
OVSSX	Power Supply for VSS
VPDX	3.3V CMOS input, disable input for Pull up/down resistors, connect to GND
YB002AAX	Oscillator Output
YB3DNLMX	Bidirectional True buffer (3.3V CMOS, 25K Pull-down, IOL=4mA,Low Noise type)
YB3NNLMX	Bidirectional True buffer (3.3V CMOS, IOL=4mA,Low Noise type)
YI002AEX	Oscillator Pin Input

## 2 Electrical Specification

### 2.1 Maximum Ratings

The maximum ratings are the limit values that must never be exceeded even for an instant. As long as the device is used within the maximum ratings specified range, it will never be damaged.

The Cx71 series of CMOS ASICs has five types of output buffers for driving current values, each of which has a different maximum output current rating.

**Table 2-1: Maximum Ratings**

Parameter	Symbol	Requirements	Unit
Supply voltage	VDDI VDDE SDRAM_VCC APLL_AVDD DAC_VDDA	-0.5 to +3.0 -0.5 to +4.0 VDDI VDDI -0.5 to +3.0	V
Input voltage	VI	-0.5 to VDD + 0.5 ( $\leq 4.0V$ ) <sup>a</sup> -0.5 to VDDE + 4.0 ( $\leq 6.0V$ ) <sup>b</sup>	V
Output voltage	VO	-0.5 to VDD + 0.5 ( $\leq 4.0V$ ) <sup>a</sup> -0.5 to VDDE + 4.0 <sup>b</sup> <L/H- State> -0.5 to VDDE + 4.0 ( $\leq 6.0V$ ) <sup>b</sup> <Z- State>	V
Storage temperature	TST	-55 to +125	°C
Junction temperature	Tj	-40 to +125	
Ambient temperature	Ta	-40 to +85	
Output current <sup>c</sup>	IO	+/-13	mA
Supply pin current for one VDD/GND pin	ID	60	mA

a.for 3.3V interface

b.for 5.0V tolerant

c.The maximum output current which always flows in the circuit.

#### 2.1.1 Power-on sequence

Lavender and Jasmine are dual power supply devices. For power ON/OFF sequence, there is no specific restriction, but the following sequences are recommended:

Power-ON:VDDI (internal, 2.5V)-> VDDE (external, 3.3V)-> Signal

Power-OFF:Signal-> VDDE (external, 3.3V)-> VDDI (internal, 2.5V)

It is restricted that VDDE only is supplied continuously for more than 1 minute while VDDI/DRAM supply is off. If the time exceeds 1 minute, it may affect the reliability of the internal transistors.

When VDDE is changed from off to on, the internal state of the circuit may not be maintained due to the noise by power supply. Therefore, the circuit should be initialized after power is on.

### **2.1.2 External Signal Levels**

External signal levels must not be higher than power supply voltage by 0.5V or more (3.3V inputs). If a signal with 0.5V or more than VDDE is given to an input buffer the current will flow internally to supply, which can give a permanent damage to the LSI.

In addition, when power supply becomes on or off, signal levels must not be higher than the power supply voltage by 0.5V or more. This means that signals must not be applied before power on / after power off.

If an external signal (5V) is input at a 5V tolerant input before the device in question is powered-on, it will give the LSI a permanent damage.

### **2.1.3 APLL Power Supply Level**

APLL (Analog PLL) power supply level must not be higher than power supply voltage VDDI. Please take care of APLL power supply not to be over VDDI level at Power ON/OFF sequence.

### **2.1.4 DAC supply**

DAC supply is isolated from other 2.5V supply.

### **2.1.5 SDRAM Supply**

SDRAM supply must be as same level as VDDI (Jasmine only).

## 2.2 Recommended Operating Conditions

The recommended operating conditions are the recommended values for assuring normal logic operation.

As long as the device is used within the recommended operating conditions, the electrical characteristics described below are assured.

**Table 2-2:** Operating conditions

Parameter		Symbol	Requirements			Unit
			Min	Typ	Max	
Supply voltage		VDDE	3.0	3.3	3.6	V
		VDDI	2.3	2.5	2.7	
		DAC_VDDA	2.3	2.5	2.7	
High-level input voltage	3.3V	VIH	2.0	-	VDDE+ 0.3	V
	5V Tolerant		2.0	-	5.5	
Low-level input voltage	3.3V	VIL	-0.3	-	0.8	V
	5V Tolerant		-0.3	-	0.8	
Junction temperature		Tj	-40	-	125	°C
Ambient temperature		Ta	-40	-	85	°C

## 2.3 DC Characteristics

The DC characteristics assure the worst values of the static characteristics of input/output buffers within the range specified at the recommended operating conditions.

**Table 2-3:** DC characteristics

Parameter	Symbol	Test conditions	Requirements			Unit
			Min	Typ	Max	
Supply current <sup>a b</sup>	IDDS	ASIC master type T7	-	-	0.2	mA
High-level output voltage	VOH	IOH= -100uA	VDDE-0.2	-	VDDE	V
Low-level output voltage	VOL	IOL= 100uA	0	-	0.2	V
High-level output current	IOH	L type VOH=VDDE- 0.4V	-2	-	-	mA
		M type VOH=VDDE- 0.4V	-4	-	-	mA
		H type VOH=VDDE- 0.4V	-8	-	-	mA



**Table 2-3:** DC characteristics

Parameter	Symbol	Test conditions	Requirements			Unit
			Min	Typ	Max	
Low-level output current	IOL	L type VOL=0.4V	2.0	-	-	mA
		M type VOL=0.4V	4.0	-	-	mA
		H type VOL=0.4V	8.0	-	-	mA
Input leakage current per pin <sup>b</sup>	IL		-	-	+/-5	uA
Input pull-up/pull-down resistor <sup>c</sup>	RP		10	25	70	kOhm
Output short-circuit current <sup>d</sup>	IOS	L type	-	-	+/-40	mA
		M type	-	-	+/-60	mA
		H type	-	-	+/-120	mA

a. VIH = VDD and VIL = VSS, memory is in stand-by mode, Analog cells (APLL, DACs, DAC-VREF) are at power down mode, Tj = 25°C

b. Input pins have to be static. If an input buffer with pull-up/pull-down resistor is used, the input leakage current may exceed the above value

c. Either a buffer without a resistor or with a pull-up/pull-down resistor can be selected from the input and bidirectional buffers.

d. Maximum supply current at the short circuit of output and VDD or VSS. For 1 second per pin.

Following table shows current/power consumption for Jasmine under special operating conditions. Core clock, which has most influence is varied over specified range. Please note, if other parameters varied that given values can be exceeded.

**Table 2-4:** Maximum core current consumption for Jasmine

Frequency [MHz]	APLL Divider/Multiplier Setup <sup>a</sup>	Core/Analog supply [mA]	DRAM supply [mA]	Power consumption [mW]
16	5 / 7	84.1	9.2 (3.8) <sup>b</sup>	360 (346)
20	5 / 9	104.4	11.5 (4.8)	421 (403)
36	6 / 20	180.7	20.7 (9.1)	652 (621)
48	5 / 23	232.4	27.6 (10.6)	811 (765)
64	2 / 15	294.0	36.8 (12.6)	1001 (936)

a. Values interpreted with n+1

b. Values for DRAM supply in parenthesis are measured while running an usual application.

Measurement conditions:

- Oscillator 12.0 MHz
- Video clock 13.5 MHz, Pixel Clock (display) 6.0 MHz, ULB\_CLK 16 MHz
- VDDI = 2.7V, VDDE = 3.6V

- I/O current assumed 30 mA, this varies in given environments/applications. Part of I/O power consumption was  $30\text{mA} * 3.6\text{V} = 108\text{mW}$  (fixed within this measurement environment).

## 2.4 Mounting / Soldering

Mounting and soldering is explained in Fujitsu package data book, chapter 2.

## 2.5 AC Characteristics

### 2.5.1 Measurement Conditions

For the determination of GDC timing values three independent conditions are used:

4. **External capacities:** 20pF and 50pF.  
Internal capacitance of the pin itself is included in this measurement condition (20/50pF already including 5...7pF pin capacitance) and has to be subtracted from external capacitive load.
5. **Operating conditions:** *maximum conditions* (min. voltage/max. temperature/slowest process) and *minimum conditions* (max. voltage/min. temperature/fastest process).  
The allowed values for voltage and temperature are described in the operating condition chapter.
6. **Timing path:** *longest* and *shortest* (internal) path to/from a given output/input pin.

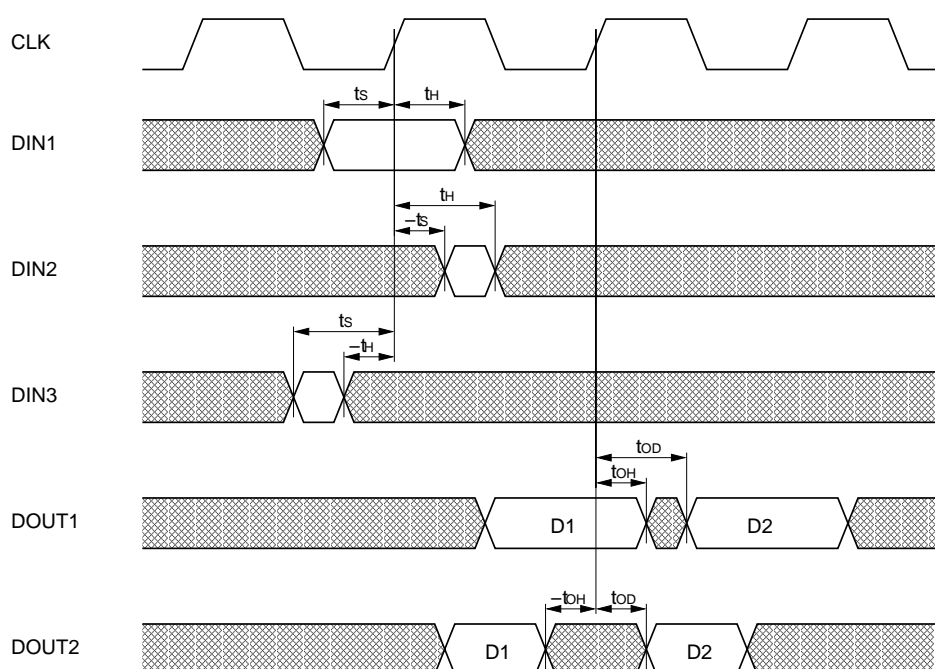
For every timing type for input and output signals always the worst case for the environment circuits is listed in this specification. Table 2-5 shows the used conditions for all signal types in this specification.

Input setup and hold timings are minimum requirements from the application point of view, thus specified in the “Min” column. However the input parameters are evaluated at maximum operating conditions and longest timing path (see table 2-5).

For tri-state outputs the timing is evaluated for the 0 -> Z and 1 -> Z transitions without any load capacitance dependency. Measurement points are defined if current through the output pin becomes below 5% of its nominal value. The voltage level at the output pin is not of interest when switching into high-Z state. The timing of voltage level in high-Z state depends only on the external RC combination and is not related to this AC specification.

### 2.5.2 Definitions

Figure 2-1 shows the definition of setup and hold relations for inputs (DIN1...3) and hold and delay relations for outputs (DOUT1...2) regarding the interface clock pin. DIN2, DIN3 and DOUT2 are special cases when negative setup or hold timings are given.



**Figure 2-1:** Input and Output Timing Relations

Table 2-5 shows the kind of timing values defined in this specification, its symbols and the evaluation conditions used to determine a specific value according to chapter 2.5.1.

For a given pin the symbol name is expanded by the pin name (e.g.  $t_{OD<Pin>}$ ). Sometimes also a shortcut for the pinname is used but the timing value has a unique name.

**Table 2-5:** Symbolic names and evaluation conditions

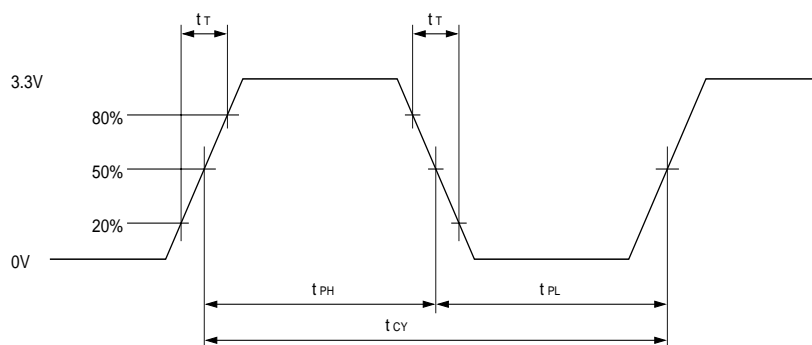
Timing	Symbol	Conditions as described in chapter 2.5.1
Input setup time	$t_S$	1: independent from external capacity 2: max conditions 3: longest path
Input hold time	$t_H$	1: independent from external capacity 2: max conditions 3: longest path
Output delay time	$t_{OD}$	1: separate value for 20pF and 50pF 2: max conditions 3: longest path
Output hold time	$t_{OH}$	1: separate value for 20pF and 50pF 2: min. conditions 3: shortest path

### 2.5.3 Clock inputs

OSC\_IN, OSC\_OUT are dedicated ports for crystal oscillator connection. When OSC\_IN is used as direct clock input, the specification applies as stated for the other possible clock inputs.

ULB\_CLK, RCLK, VSC\_CLKV, DIS\_PIXCLK give the ability to feed in an external clock directly. For usage of different clock inputs see Clock Unit specification.

- DIS\_PIXCLK can be configured as input or output



**Figure 2-2:** AC characteristics measurement conditions

- Transition Time  $t_T$  max. 2ns
- $V_{IH} = 2.0V$ ,  $V_{IL} = 0.8V$  (3.3V CMOS Interface Input)

**Table 2-6:** Timing Specification

Clock	Parameter	Symbol	Min	Max
ULB_CLK (routed to CLKM, MCU interface)	ULB Clock Cycle Time	$t_{CYU}$	15.625 ns	-
	ULB Clock High Pulse Width	$t_{PHU}$	7 ns	-
	ULB Clock Low Pulse Width	$t_{PLU}$	7 ns	-

**Table 2-6:** Timing Specification

<b>Clock</b>	<b>Parameter</b>	<b>Symbol</b>	<b>Min</b>	<b>Max</b>
RCLK (routed to CLKK, core clock without APLL)	RSV Clock Cycle Time	$t_{CYR}$	15.625 ns	-
	RSV Clock High Pulse Width	$t_{PHR}$	7 ns	-
	RSV Clock Low Pulse Width	$t_{PLR}$	7 ns	-
VSC_CLKV (video interface clock)	Video Clock Cycle Time	$t_{CYV}$	18.50 ns	-
	Video Clock High Pulse Width	$t_{PHV}$	9.25 ns	-
	Video Clock Low Pulse Width	$t_{PLV}$	9.25 ns	-
DIS_PIXCLK (routed to CLKD as external display clock)	Display Clock Cycle Time	$t_{CYD}$	18.50 ns	-
	Display Clock High Pulse Width	$t_{PHD}$	9.25 ns	-
	Display Clock Low Pulse Width	$t_{PLD}$	9.25 ns	-

## 2.5.4 MCU User Logic Bus Interface

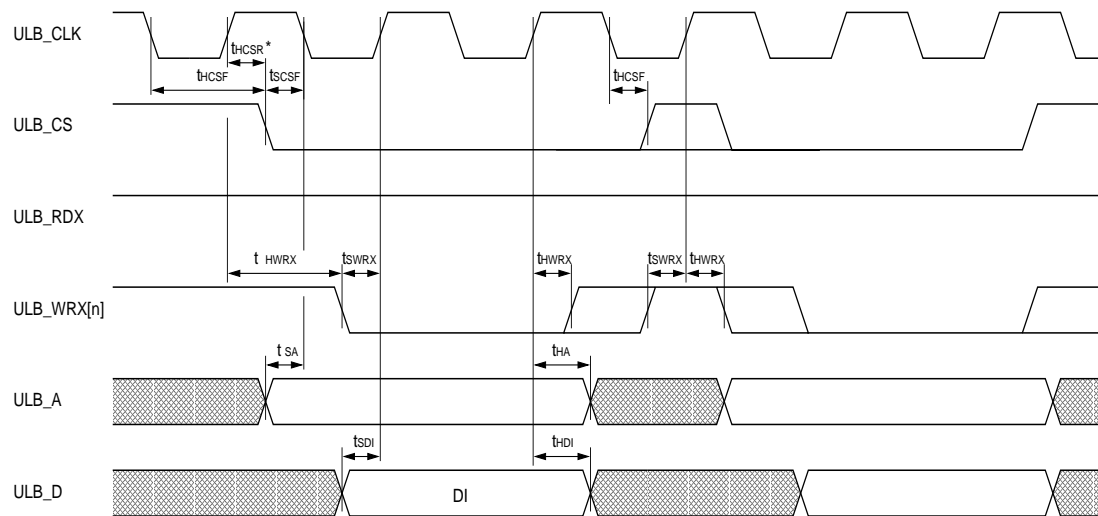


Figure 2-3: ULB write access (followed by another write)

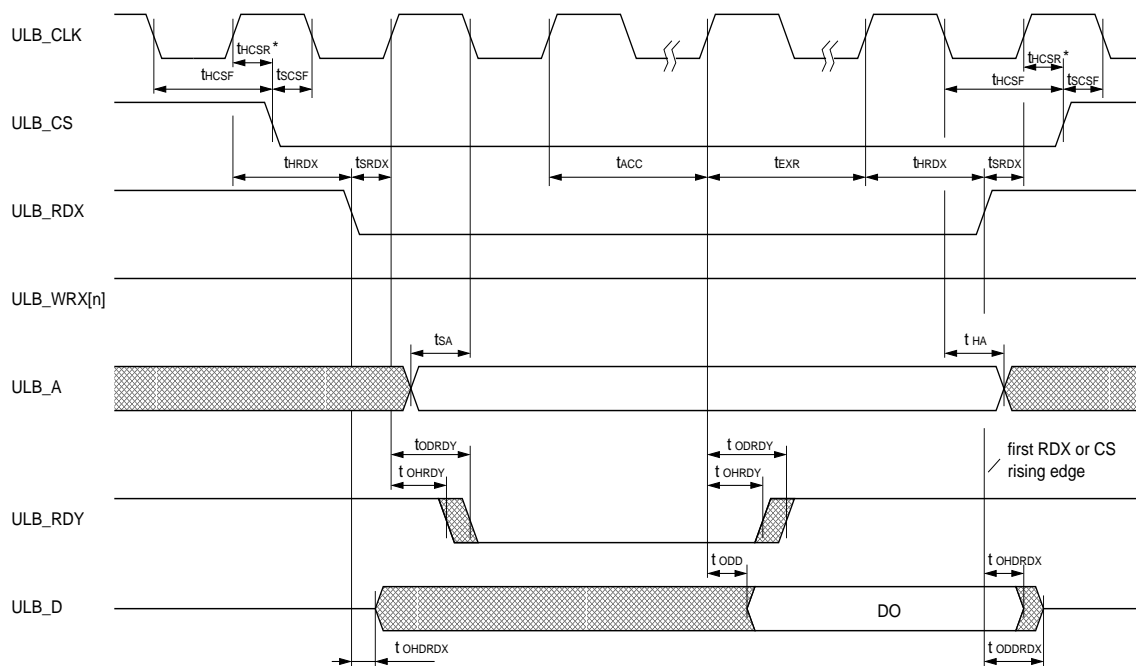


Figure 2-4: ULB read access

Table 2-7: ULB Input Signal Timing Specification

Parameter	Symbol	Min [ps]		Max [ps]	
		Jas	Lav	Jas	Lav
Chip Select (ULB_CS) Setup Time (falling)	$t_{SCSF}$	1450	410	-	
Chip Select (ULB_CS) Hold Time (falling)	$t_{HCSF}$	-130	2380	-	
Chip Select (ULB_CS) Hold Time (rising) <sup>a</sup>	$t_{HCSR}^*$	1800	3120	-	

**Table 2-7:** ULB Input Signal Timing Specification

Parameter	Symbol	Min [ps]		Max [ps]	
		Jas	Lav	Jas	Lav
Read (ULB_RDX) Setup Time	$t_{SRDX}$	980	5020	-	
Read (ULB_RDX) Hold Time	$t_{HRDX}$	1890	2620	-	
Write (ULB_WRX) Setup Time	$t_{SWRX}$	5820	7890	-	
Write (ULB_WRX) Hold Time	$t_{HWRX}$	1700	3850	-	
Address (ULB_A) Setup Time <sup>b</sup>	$t_{SA}$	2600	2410	-	
Address (ULB_A) Hold Time	$t_{HA}$	1910	3610	-	
Input Data (ULB_D) Setup Time	$t_{SDI}$	5200	1070	-	
Input Data (ULB_D) Hold Time	$t_{HDI}$	2130	3750	-	
Access Time	$t_{ACC}$	1 $T_{ULB\_CLK}$		variable <sup>c</sup>	
External Reaction Time on ULB_RDY	$t_{EXR}$	0 $T_{ULB\_CLK}$		-	

a. More restrictive specification to rising edge of CLK\_ULB is only needed if SPB will be used.

b. Address setup timing related to falling edge.

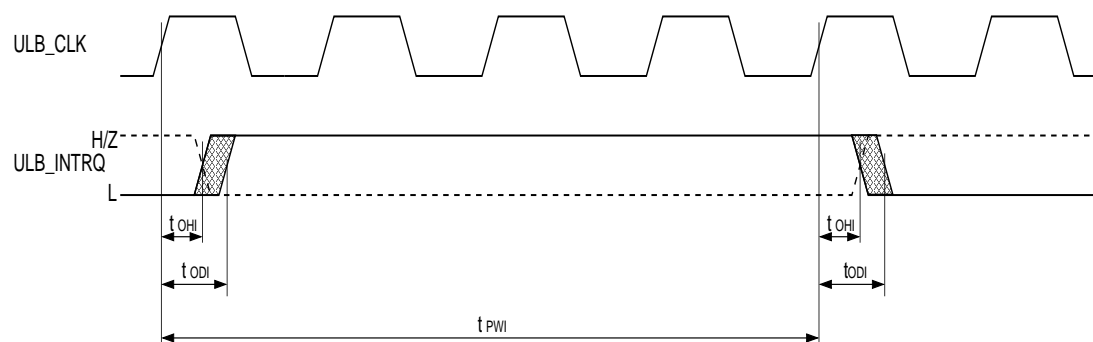
c. Access time varies with whole number of ULB\_CLK periods for different register addresses. Required timing is controlled with ULB\_RDY handshake.

**Table 2-8:** ULB Timing Specification, Output Characteristics

Parameter	Symbol	@20pF [ps]		@50pF [ps]	
		Jas	Lav	Jas	Lav
max. Ready (ULB_RDY) Output Delay Time	$t_{ODRDY}$	13960	13070	15690	15110
min. Ready (ULB_RDY) Output Hold Time	$t_{OHRDY}$	4550	4560	4660	5190
max. Output Data (ULB_D) Delay Time (regard. CLK)	$t_{ODD}$	17980	21160	20060	23230
min. Output Data (ULB_D) Hold Time (regard. CLK)	$t_{OHD}$	2640	3980	2640	4120
max. Output Data (ULB_D) Delay Time (regard. RDX)	$t_{ODDRDX}$	12010	16490	14120	18560
min. Output Data (ULB_D) Hold Time (regard. RDX)	$t_{OHDRDX}$	2450	3560	3120	3750

## 2.5.5 Interrupt

For a functional description of GDC interrupt controller and supported settings see User Logic Bus (ULB) controller description. This specification describes only the physical timing of interrupt request signal.



**Figure 2-5:** Interrupt output timing

**Table 2-9:** INTRQ Timing Specification

Parameter	Symbol	Min [ps]		Max [ps]	
		Jas	Lav	Jas	Lav
Interrupt (ULB_INTRQ) Pulse Width	$t_{PWI}$	$2T / INTREQ^a$		-	

a. Setup value in periods of ULB\_CLK, for edge sensitive mode only. In Level sensitive mode minimum 2 cycles.

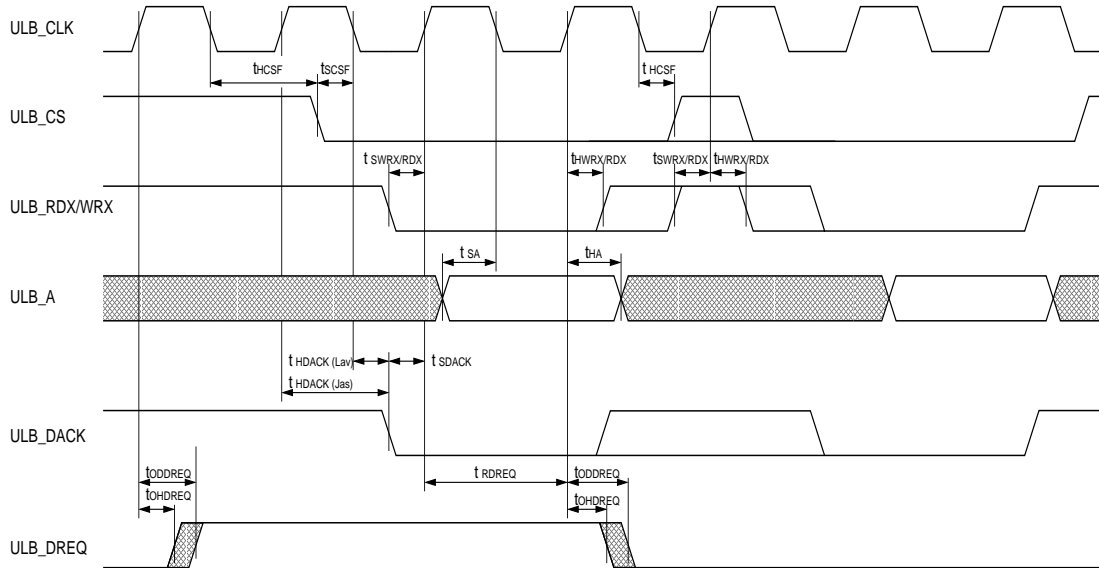
**Table 2-10:** INTRQ Timing Specification, Output Characteristics

Parameter	Symbol	@20pF [ps]		@50pF [ps]	
		Jas	Lav	Jas	Lav
max. Interrupt (ULB_INTRQ) output delay time	$t_{ODI}$	12360	12870	14400	14900
min. Interrupt (ULB_INTRQ) output hold time	$t_{OHI}$	4020	4420	4360	4640



## 2.5.6 DMA Control Ports

For a functional description of GDC DMA controller and supported settings see User Logic Bus (ULB) controller description. This specification describes only the physical timing of DMA control signals.



**Figure 2-6:** DMA block/burst access

**Table 2-11:** DMA Timing Specification

Parameter	Symbol	Min [ps]		Max [ps]	
		Jas	Lav	Jas	Lav
DMA acknowledge (ULB_DACK) Setup Time (rising)	$t_{SDACK}$	-80	50	-	-
DMA acknowledge (ULB_DACK) Hold Time (rising) <sup>a</sup>	$t_{HDACK}$	1600	-	-	-
DMA acknowledge (ULB_DACK) Hold Time (falling) <sup>b</sup>	$t_{HDACK}$	-	3130	-	-
DMA request (ULB_DREQ) Reaction Time	$t_{RDREQ}$	1T / 3T <sup>c</sup>		-	

a. Jasmine has ULB\_DACK timing requirement for rising clock edge only.

b. Lavender has ULB\_DACK timing requirement to both edge types. DACK 1->0 transition only allowed between falling and rising clock edge.

c. ULB\_DREQ reaction time on ULB\_DACK. 1 cycle in block/step/burst mode. Minimum 3 cycles in demand mode.

**Table 2-12:** DMA Timing Specification, Output Characteristics

Parameter	Symbol	@20pF [ps]		@50pF [ps]	
		Jas	Lav	Jas	Lav
max. DMA request (ULB_DREQ) Output Delay Time	$t_{ODDREQ}$	14140	15480	16170	17530
min. DMA request (ULB_DREQ) Output Hold Time	$t_{OHDREQ}$	4200	4460	4600	4690

## 2.5.7 Display Interface

For display settings, supported colour formats, functional timing and configuration possibilities of display interface see Graphic Processing Unit (GPU) description. This specification describes only the physical timing of display signals.

Pixel clock output pin (DIS\_PIXCLK) has a clock invert option. Thus timings are valid for both clock edges.

**Table 2-13: Pixel Clock Output Timing**

Parameter	Symbol	@20pF [ps]		@50pF [ps]	
		Jas	Lav	Jas	Lav
max. Pixel Clock (DIS_PIXCLK) Output Delay (r/f) <sup>a</sup>	t <sub>ODPIXCLK</sub>	15740	11240	17900	13410
min. Pixel Clock (DIS_PIXCLK) Output Hold	t <sub>OHPIXCLK</sub>	5240	3610	6090	4460

a. Related to internal PIXEL\_CLK

The timing values given in table 2-14 and 2-15 are related to Display clock output pin (DIS\_PIXCLK). The values are only valid if the capacitive load is the same for clock and data pins.

Because the driving clock edge for display clock (DIS\_PIXCLK) is programmable (see GPU description for more details) all possible edge combinations have to be taken into account for timing calculation.

**Table 2-14: Digital Display Interface, Output Characteristics**

Parameter	Symbol	@20pF [ps]		@50pF [ps]	
		Jas	Lav	Jas	Lav
max. Display Data (DIS_D) Output Delay Time (r/f)	t <sub>ODDISD</sub>	3510	4540	4020	4930
min. Display Data (DIS_D) Output Hold Time (r/f)	t <sub>OHDISD</sub>	-690	-820	-430	-600
max. Color Key (DIS_CKEY/DIS_CK) Output Delay (r/f)	t <sub>ODCKEY</sub>	1960	2880	2450	3320
min. Color Key (DIS_CKEY/DIS_CK) Output Hold (r/f)	t <sub>OHCKEY</sub>	-800	200	-530	300
max. DIS_VSYNC Output Delay (r/f)	t <sub>ODVSYNC</sub>	2800	3770	3220	4200
min. DIS_VSYNC Output Hold (r/f)	t <sub>OHVSYNC</sub>	-4770	-1610	-6930	-3780
max. DIS_HSYNC Output Delay (r/f)	t <sub>ODHSYNC</sub>	2420	3720	2840	4140
min. DIS_HSYNC Output Hold (r/f)	t <sub>OHHSYNC</sub>	-5050	-1720	-7210	-3890
max. DIS_VREF Output Delay (r/f)	t <sub>ODVREF</sub>	2560	3710	2980	4140
min. DIS_VREF Output Hold (r/f)	t <sub>OHVREF</sub>	-4920	-1720	-7080	-3890

**Table 2-15:** Analog Display Interface, Output Characteristics

Parameter	Symbol	@20pF [ps]		@50pF [ps]	
		Jas	Lav	Jas	Lav
max. Disp. Analog Output Delay Time (r) <sup>a</sup>	t <sub>ODDISA</sub>	14200	14430	13350	13580
min. Display Analog Output Hold Time	t <sub>OHDISA</sub>	8410	9080	6250	6910

a. Related to DIS\_PIXCLK output pin. The capacitive influence on the pixel clock output is greater than the influence on the analog outputs. Due to subtracted clock delay the relative delay values are smaller in the 50pF column.

## 2.5.8 Video Input

For a functional timing description of video input signals and for a description of supported video modes see Video Interface Controller (VIC) description. This specification describes only the physical timing of video input signals.

**Table 2-16:** Video Input Timing

Parameter	Symbol	Min [ps]		Max [ps]	
		Jas	Lav	Jas	Lav
Video Data (VSC_D) Setup Time	t <sub>SVID</sub>	-360	-1660	-	-
Video Data (VSC_D) Hold Time	t <sub>HVID</sub>	2360	2800	-	-
VSC_VREF Setup Time	t <sub>SVIVREF</sub>	-1940	-1580	-	-
VSC_VREF Hold Time	t <sub>HVIVREF</sub>	2660	2450	-	-
VSC_VACT Setup	t <sub>SVIVACT</sub>	-2100	-1610	-	-
VSC_VACT Hold	t <sub>HVIVACT</sub>	2820	2510	-	-
VSC_ALPHA Setup	t <sub>SVIALPHA</sub>	-2100	-1590	-	-
VSC_ALPHA Hold	t <sub>HVIALPHA</sub>	2840	2540	-	-
VSC_IDENT Setup	t <sub>SVIDENTI</sub>	-2100	-1620	-	-
VSC_IDENT Hold	t <sub>HVIDENTI</sub>	2850	2540	-	-

## 2.5.9 CCFL FET Driver

For pulse shape and functional timing of CCFL FET driver see CCFL Description. This information is for capacitance influence on operating conditions and skew estimation only.

**Table 2-17:** CCFL Driver Outputs, Output Characteristics

Parameter	Symbol	@20pF [ps]		@50pF [ps]	
		Jas	Lav	Jas	Lav
max. CCFL_FET1 Output Delay Time <sup>a</sup>	t <sub>ODCCFET1</sub>	8460	5310	10490	7340
min. CCFL_FET1 Output Hold Time	t <sub>OHCCFET1</sub>	3040	1830	3670	2460
max. CCFL_FET2 Output Delay Time	t <sub>ODCCFET2</sub>	8370	5270	10420	7290

**Table 2-17:** CCFL Driver Outputs, Output Characteristics

Parameter	Symbol	@20pF [ps]		@50pF [ps]	
		Jas	Lav	Jas	Lav
min. CCFL_FET2 Output Hold Time	t <sub>OHCCFET2</sub>	3020	1830	3640	2450
max. CCFL_OFF Output Delay Time	t <sub>ODCCOFF</sub>	8130	4730	10160	6760
min. CCFL_OFF Output Hold Time	t <sub>OHCCOFF</sub>	2920	1730	3540	2350
max. CCFL_IGNIT Output Delay Time	t <sub>ODCCIGNIT</sub>	8200	4740	10250	6770
min. CCFL_IGNIT Output Hold Time	t <sub>OHCCIGNIT</sub>	2940	1710	3570	2330

a. For Jasmine related to internal MASTER\_CLK, rising edge; for Lavender only the path delay was used.

## 2.5.10 Serial Peripheral Bus

For pulse shape and functional timing see SPB documentation. SPB timing regarding ULB\_CLK is not of interest. It is given for completeness only.

**Table 2-18:** SPB Pin Timing

Parameter	Symbol	Min [ps]		Max [ps]	
		Jas	Lav	Jas	Lav
SPB_BUS Input Setup Time (falling)	t <sub>SSPB</sub>	-1530	-3780	-	-
SPB_BUS Input Hold Time	t <sub>HSPB</sub>	3100	4870	-	-

**Table 2-19:** SPB Pin Timing, Output Characteristics

Parameter	Symbol	@20pF [ps]		@50pF [ps]	
		Jas	Lav	Jas	Lav
max. SPB_BUS Output Delay Time	t <sub>ODSPB</sub>	11470	14890	12500	14890
min. SPB_BUS Output Hold Time	t <sub>OHSPB</sub>	2990	4670	3010	5110

## 2.5.11 Special and Mode Pins

Mode[3:0], RDY\_TRIEN, VPD, TEST are static configuration and test pins.

RESETX is asynchronous, thus no timing relation to any CLK can be specified. Maximum low pulse width suppressed by spike filter is 5.5 ns. Only Jasmine has a spike filter implemented.

## 2.5.12 SDRAM Ports (Lavender)

See SDC documentation for description of external SDRAM interface timing configuration. Different from measurement conditions above, SDRAM interface timing is evaluated from load capacitance connection of C<sub>L</sub> = 10pF for MIN conditions and C<sub>L</sub> = 30pF for MAX conditions.

SDRAM clock output delay regarding internal core clock is shown in table 2-20. Other SDRAM interface signals are specified regarding the SDC\_CLK clock output pin. Same capacitive load is assumed for clock and address/command/data pins. If different load capacitance is connected to the SDC\_CLK pin than to the

other SDRAM interface pins timing is not guaranteed to follow this specification. Thus all connections to the SDRAM should have same wire length.

**Table 2-20:** SDRAM Clock Output Timing

Parameter	Symbol	@10pF [ps]	@30pF [ps]
max. SDRAM Clock (SDC_CLK) Output Delay Time	$t_{ODSDCCLK}$	7100	8470
min. SDRAM Clock (SDC_CLK) Output Hold Time	$t_{OHSDCCLK}$	2850	3480

Due to configurable SDRAM interface timing, setup value of SDIF register should be considered. The configuration register has two bits for each signal group:

- SDIF\_TAO for address, control (ras/cas/we), CKE and DQM output delays
- SDIF\_TDO for data output delay
- SDIF\_TDI for data input sampling delay and
- SDIF\_TOE for controlling tri-state enable of the data output buffers.

Table 2-21 specifies the possible delay shifts, commonly valid for address, command, CKE, DQM and data ports.

**Table 2-21:** Configurable Delay Shifts by SDIF

SDIF Setup	Min Delay Shift [ps]	Max Delay Shift [ps]
00 <sub>b</sub>	0	0
01 <sub>b</sub>	650	1920
10 <sub>b</sub>	1240	3620
11 <sub>b</sub>	1720	5030

SDRAM port timing specifications in table 2-22 are valid for a configured delay value 00<sub>b</sub>.

The user can add the given delay shifts to the appropriate minimum and maximum values for different configurations.

Note that additional delay shifts have to be subtracted for input setup timings and delay shifts have to be added for output delay, output hold and input hold timings.

For data input this could be used to shift the data sampling point later with regard to the internal clock.

For calculating shifted output parameters the user should take minimum delay shifts for MIN delay/hold and maximum delay shifts for MAX delay/hold. However input characteristics are always valid under MAX operating conditions and have to be shifted by delay shift value given in MAX column. Input characteristics are given in the MIN column because they are minimum requirements.

The timing values given in table 2-22 and table 2-23 are related to Lavender SDRAM clock pin (SDC\_CLK).

Because the clock output path depends on external capacitance also the input setup and hold times are capacitance dependent which is normally not the case (see also chapter 2.5.2).

**Table 2-22:** SDRAM Port Timing for Lavender

Parameter	Symbol	@10pF[ps]	@30pF[ps]
SDRAM Data (SDC_D) Input Setup Time (max cond.) <sup>a</sup>	t <sub>SSDCD</sub>	2520	3890
SDRAM Data (SDC_D) Input Hold Time (max cond.)	t <sub>HSDCD</sub>	-850	-1480

a. Related to SDRAM clock pin (SDC\_CLK).

**Table 2-23:** SDRAM Port Timing for Lavender, Output Characteristics

Parameter	Symbol	@10pF [ps]	@30pF [ps]
max. SDRAM Address (SDC_A) Output Delay Time	t <sub>ODSDCA</sub>	3870	4250
min. SDRAM Address (SDC_A) Output Hold Time	t <sub>OHSDCA</sub>	1280	1410
max. SDRAM Command Output Delay Time (SDC_RAS/SDC_CAS/SDC_WE)	t <sub>ODSDCCMD</sub>	3820	4180
min. SDRAM Command Output Hold Time (SDC_RAS/SDC_CAS/SDC_WE)	t <sub>OHSDCCMD</sub>	1270	1400
max. SDRAM Clock Enable (SDC_CKE) Output Delay Time	t <sub>ODSDCCKE</sub>	3520	3930
min. SDRAM Clock Enable (SDC_CKE) Output Hold Time	t <sub>OHSDCCKE</sub>	1150	1290
max. SDRAM Data Enable/Mask (SDC_DMQ) Output Delay Time	t <sub>ODSDCDQM</sub>	3800	4180
min. SDRAM Data Enable/Mask (SDC_DMQ) Output Hold Time	t <sub>OHSDCDQM</sub>	1280	1410
max. SDRAM Data (SDC_D) Output Delay Time	t <sub>ODSDCD</sub>	6160	6590
min. SDRAM Data (SDC_D) Output Hold Time	t <sub>OHSDCD</sub>	1240	660

---

## ***PART D - Appendix***





---

## ***D-1 Jasmine Command and Register Description***



# 1 Register Description

Table 1-2 shows all registers and bit groups of MB87P2020-A (Jasmine) and MB87J2120 (Lavender) with a short explanation. A more detailed description can be found in the component specific manual parts.

In table 1-2 valid registers or bitgroups for every device are marked in additional columns. Some registers or bitgroups are listed twice, separated for every device, since they are different for Lavender and Jasmine. Additionally some special registers exist which are marked in table 1-2 with a special code explained in table 1-1.

**Table 1-1:** Marking for special registers

Mark	Description
R	Read only register or bitgroup
W	Write only register or bitgroup
F	Hardware flag (value can be manipulated by hardware)
M	Register write access is locked if MTIMON_ON=1 (GPU master switch)
C	These registers have an internal shadow register which is synchronized to command execution. If a command is running the shadow register is locked, written values are stored and take effect only when next command is started. For reading always the user writeable register is returned.
D	Like 'C' but reading is configurable with READINIT_RCR

**Table 1-2:** Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
ULB (User Logic Bus Controller)								
CMD	0x0000						Command register	
		31:8	PAR	o	o	C	Command parameter	0
		7:0	CODE	o	o	C	Command code (for writing check FLNOM_CWEN)	0xFF (NoOp)
IFIFO	0x0004	31:0	–	o	o	W	Input FIFO Only word access is allowed.	–
OFIFO	0x0008	31:0	–	o	o	R	Output FIFO Only word access is allowed.	–
FLNOM	0x000C	31:0	–	o	o	F	Flag register (normal write access) <sup>a</sup>	0x20400000
FLRST	0x0010	31:0	–	o	o	F	Flag register (reset write access) <sup>a</sup> 1: Reset flag at this position	0x20400000

Table 1-2: Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
FLSET	0x0014	31:0	–	o	o	F	Flag register (set write access) <sup>a</sup> 1: Set flag at this position	0x20400000
INTNOM	0x0018	31:0	–	o	o		Interrupt mask register (normal write access) <sup>a</sup> 1: use flag for interrupt	0
INTRST	0x001C	31:0	–	o	o		Interrupt mask register (reset write access) <sup>a</sup> 1: Reset mask at this position	0
INTSET	0x0020	31:0	–	o	o		Interrupt mask register (set write access) <sup>a</sup> 1: Set mask at this position	0
INTLVL	0x0024	31:0		o	o		Interrupt level/edge settings <sup>a</sup> 1: positive edge of flag triggers interrupt 0: high level of flag triggers interrupt	0
INTREQ	0x0028						Interrupt request length	
		5:0	INTCLK		o		Interrupt request length in ULB clocks (CLKM)	0x10
RDYTO	0x002C						RDY timeout control register	
		7:0	RDYTO		o		RDY timeout length in ULB clocks (CLKM)	0xFF
		8	RDTOEN		o		RDY timeout enable 1: enable RDY timeout	1
RDYADDR	0x0030						RDY timeout address register (read only)	
		20:0	ADDR		o	R	Address where RDY timeout occurred	0

**Table 1-2:** Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
IFCTRL	0x0034						MCU interface control register	
		9:8	SMODE		0		Sample mode for bus control signals 00: 3 point mode 01: 2 of 3 point mode 10: 2 point mode 11: 1 point mode	0
		5	DRINV		0		1: invert ULB_DREQ	0
		4	DSINV		0		1: invert ULB_DSTP	0
		3	INTINV		0		1: invert ULB_INTRQ	0
		2	DRTRI		0		1: open drain for ULB_DREQ	0
		1	DSTRI		0		1: open drain for ULB_DSTP	0
		0	INTTRI		0		1: open drain for ULB_INTRQ	0
WNDOF0	0x0040	20:0	OFF	0	0		MCU offset for SDRAM window 0	0x10'0000
WNDSZ0	0x0044	20:0	SIZE	0	0		Size of SDRAM window 0	0x10'0000
WNDOF1	0x0048	20:0	OFF	0	0		MCU offset for SDRAM window 1	0x10'0000
WNDSZ1	0x004C	20:0	SIZE				Size of SDRAM window 1	0x10'0000
WNDS0	0x0050	19:0	OFF		0		SDRAM offset for SDRAM window 0	0
		22:0		0				
WNDS1	0x0054	19:0	OFF		0		SDRAM offset for SDRAM window 1	0
		22:0		0				
SDFLAG	0x0058	0	EN	0	0		'1': enable SDRAM space '0': any access to SDRAM space is ignored	0
IFUL	0x0080						Input FIFO limits	
		22:16	UL		0		Input FIFO upper limit for flag- or interrupt controlled flow control IFH=1 if IFLOAD >= IFUL_UL	0x0C
		23:16		0				
		6:0	LL		0		Input FIFO lower limit for flag- or interrupt controlled flow control IFL=1 if IFLOAD <= IFUL_LL	0x03
		7:0		0				

Table 1-2: Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
OFUL	0x0084						Output FIFO limits	
		22:16	UL		o		Output FIFO upper limit for flag- or interrupt controlled flow control OFH=1 if OFLOAD >= OFUL_UL	0x3C
		23:16		o				
		6:0	LL				Output FIFO lower limit for flag- or interrupt controlled flow control OFL=1 if OFLOAD <= IFUL_LL	0x0F
IFDMA	0x0088						Input FIFO limits for DMA transfer	
		22:16	UL		o		Upper limit for DMA access to input FIFO (not used)	0x0A
		23:16		o				
		6:0	LL		o		Lower limit for DMA access to input FIFO	0x3C
		7:0		o				
OFDMA	0x008C						Output FIFO limits for DMA transfer	
		22:16	UL		o		Upper limit for DMA access from output FIFO	0x0A
		23:16		o				
		6:0	LL		o		Lower limit for DMA access from output FIFO (not used)	0x3C
		7:0		o				

**Table 1-2:** Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
DMAFLAG	0x0090						DMA flag register	
		12:8	DSTP	o	o		Duration of DSTP signal. This value can be set in order to ensure a save MCU-DMAC reset. Normally the default value should work.	7
		4	TRI	o			1: Open drain for ULB_DREQ, ULB_DSTP, ULB_INTRQ	0
		3	INV	o			1: Invert ULB_DREQ, ULB_DSTP, ULB_INTRQ	0
		2	MODE	o	o		'1': DMA demand mode '0': DMA block/step- or burst mode	0
		1	EN	o	o		'1': enable DMA	0
		0	IO	o	o		'1': use DMA for input FIFO '0': use DMA for output FIFO	1
FLAGRES	0x0094						Flag behaviour register <sup>a</sup>	
		31:0	-	o	o		1: set flag to dynamic behaviour <sup>b</sup> 0: set flag to static behaviour	0x20400000
ULBDEB	0x0098						FIFO debug register (read only)	
		23:16	IFLC	o		R	Input FIFO load for current command <b>Attention:</b> This value changes with core clock; correct sampling by MCU can't be ensured.	0x00
		22:16			o			
		15:8	OF	o		R	Output FIFO load <b>Attention:</b> This value changes with core clock; correct sampling by MCU can't be ensured.	0x00
		14:8			o			
		7:0	IF	o		R	Input FIFO load independent from current command <b>Attention:</b> This value changes with core clock; correct sampling by MCU can't be ensured.	0x00
		6:0			o			

Table 1-2: Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
CMDDEB	0x009C						Command debug register (read only)	
		31:8	PAR		o	R	Parameter for currently executed command	0x000000
		7:0	CMD		o	R	Code for currently executed command	0xFF
SDC (SDRAM Controller)								
SDSEQRAM [32] (Jasmine)  SDSEQRAM [64] (Lavender)	0x0100-0x017C (Jasmine)  0x0100-0x01FC (Lavender)						SDRAM sequencer RAM (32 words)	
		12:7	ADDR	o			Microcode sequencer address argument	undef
		11:7			o			
		6:4	INST	o	o		Microcode instruction: run, ret, call, loop, srw, rrw, pde, pdx - coded 0 to 7)	undef
		3	RAS	o	o		Container command for SDRAM (RAS)	undef
		2	CAS	o	o		Container command for SDRAM (CAS)	undef
		1	WE	o	o		Container command for SDRAM (WE)	undef
SDMODE	0x0200						SDRAM Mode Register (external SDRAM MRS register; Lavender only), Bit [12:10, 8:7] reserved, set to '0'	
		9	BRST	o			Write enable 0: Burst 1: Single	0
		6:4	CL	o			CAS latency (2/3)	0x3
		3	ILB	o			1: Interleave burst 0: Sequential burst	0
		2:0	BLEN	o			Burst length (0=1, 1=2, 2=4, 3=8, 7=full)	0x3
SDINIT	0x0204	15:0	IP	o	o		Sysclocks of SDRAM power up initialization period (200 us)	0x4E20
SDRFSH	0x0208	15:0	RP	o	o		Sysclocks of SDRAM row refresh period (16 us)	0x0640



**Table 1-2:** Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
SDWAIT	0x020C						SDRAM timings (refer to SDRAM manual)	
		20	OPT	o			Bank interleave optimization	1
		19:16	TRP	o	o		RAS Precharge Time - 1	0x2
		15:12	TRRD	o			RAS to RAS Bank Active Delay Time - 1	1
		11:8	TRAS	o	o		RAS Active Time - 1	0x5
		7:4	TRCD	o	o		RAS to CAS Delay Time - 1	0x2
		3:0	TRW	o	o		Read to Write recovery time Recommended values: Lavender: 10 (8) <sup>c</sup> Jasmine: 7 (5) <sup>c</sup>	0x3 Value for- bidden! Use recom- mended val- ues!
SDIF	0x0210						SDRAM port interface timing (scalable clock delay) - is ignored by Jasmine	
		7:6	TAO	o			Address output delay	0
		5:4	TDO	o			Data output delay	0
		3:2	TDI	o			Data sampling delay	0
		1:0	TOE	o			Tristate control delay	0
SDCFLAG	0x0214						SDC control register	
		1	DQMEN		o		1: Enable DQM partial write optimization	0
		0	BUSY	o	o		Set busy flag during micro-program upload	0
GPU (Graphic Processing Unit)								
GPU-LDR (Layer Description Record)								
PHA[16]	0x1000 - 0x103C						Physical layer address in SDRAM	
		19:0	OFS		o		Address offset (RA, CA, BYTE) Bits[9:0] fixed to zero	undef
		22:0		o		Address offset (RA, BA, CA, BYTE) Bits[11:0] fixed to zero		
DSZ[16]	0x1040 - 0x107C						Layer domain size	
		29:16	o	o	o		o dimension of layer Size	undef

Table 1-2: Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
DP1[16]	0x1080 - 0x10BC						First pixel in domain (memory offset)	
		29:16	o	o	o		Offset for o dimension	undef
		13:0	Y	o	o		Offset for Y dimension	undef
WSZ[16]	0x10C0 - 0x10FC						Display window size for layer	
		29:16	o	o	o		Size in o dimension	undef
		13:0	Y	o	o		Size in Y dimension	undef
WOF[16]	0x1100 - 0x113C						Layer offset for display	
		29:16	o	o	o		Offset for o dimension	undef
		13:0	Y	o	o		Offset for Y dimension	undef
LTC[16]	0x1140 - 0x117C						Transparent colour for layer	
		23:0	COL	o	o		Colour code depending on layer colour depth (LSB aligned)	undef
LBC[16]	0x1180 - 0x11BC						Blink colour for layer	
		23:0	COL	o	o		Colour code depending on layer colour depth (LSB aligned)	undef
LAC[16]	0x11C0 - 0x11FC						Blink alternative colour for layer	
		23:0	COL	o	o		Colour code depending on layer colour depth (LSB aligned)	undef
LBR[16]	0x1200 - 0x123C						Blink rate for layer	
		15:8	OFF	o	o		Number of frames for alternate colour	undef
		7:0	ON	o	o		Number of frames for blink colour	undef
CSPC[16]	0x1240 - 0x127C						Colour space code and flag register	
		9	LDE	o	o		Line doubling enable	undef
		8	TE	o	o		Transparency enable	undef
		3:0	CSC	o	o		Colour space code	undef
GPU-MDR (Merging Description Record)								

**Table 1-2:** Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
CFORMAT	0x1300	31:16	LITC	o	o		Layer to intermediate transfer codes (one bit for each layer)	0x0000
		7	GAMEN		o		1: Gamma table enable	0
		6	IPOLEN		o		1: Interpolation for YUV422 colour code enable	0
		5	GFORCE		o		1: Force Gamma conversation for RGB>= 16bpp	0
		3:0	CSC	o	o		Intermediate colour space code	0x0
BACKCOL	0x1304						Background colour register	
		24	EN	o	o		1: Background colour enable	1
		23:0	COL	o	o		Background colour depending on colour depth for intermediate colour space	0x000000
MBC	0x1308						Blink control register	
		15:0	EN	o	o		Blink enable (one bit for each layer)	0x0000
		31:16	CBS	o	o	R	Current blink state (read only)	0x0000
ZORDER	0x130C						Z-Order register	
		19:16	EN	o	o		Enable (one bit for each plane)	0x0
		15:12	TM3	o	o		Topmost layer number	0x0
		11:8	TM2	o	o			0x0
		7:4	TM1	o	o			0x0
		3:0	TM0	o	o		Bottom layer number	0x0
CLUTOF [16]	0x1340 - 0x137C						16 CLUT offsets (1 per layer)	
		7:0	OFS	o			CLUT offset for layer	undef
		8:0			o			
DRM[14]	0x1380 - 0x13B4						Duty ratio modulator pseudo levels, 14 words	
		5:0	PGL	o	o		Pseudo level (PGL/64 Frames)	0x00

Table 1-2: Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
PREBIAS	0x1400						Pre Matrix bias values	
		23:16	Y		o		Y value	0xF0
		15:8	CB		o		Cb value	0x80
		7:0	CR		o		Cr value	0x80
COEFFR	0x1404						Matrix coefficients to red channel	
		23:16	YXR		o		Y to Red	0x80
		15:8	CBXR		o		Cb to Red	0x00
		7:0	CRXR		o		Cr to Red	0xB3
COEFFG	0x1408						Matrix coefficients to green channel	
		23:16	YXG		o		Y to Green	0x80
		15:8	CBXG		o		Cb to Green	0x2C
		7:0	CRXG		o		Cr to Green	0x5B
COEFFB	0x140C						Matrix coefficients to blue channel	
		23:16	YXB		o		Y to Blue	0x80
		15:8	CBXB		o		Cb to Blue	0xE2
		7:0	CRXB		o		Cr to Blue	0x00
CLUT [512] (Jas- mine)	0x2000 - 0x27FC (Jas- mine)						Colour look-up table	
		23:0	COL	o	o		Colour for intermediate colour space	undef
CLUT [256] (Laven- der)	0x2000 - 0x23FC (Laven- der)							
GAMMA [256]	0x2800 - 0x2BFC						Gamma Table	
		23:16	R		o		Red mapping	undef
		15:8	G		o		Green mapping	undef
		7:0	B		o		Blue mapping	undef
GPU-DIR (Display Interface Record)								
PHSIZE	0x3000						Display physical size	
		29:16	o	o	o	M	Width	0x0000
		13:0	Y	o	o	M	Height	0x0000

Table 1-2: Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
PHFRM	0x3004						Physical format register	
		27	POL	o	o	M	Xfref polarity 0=odd field ref is low	0
		26	VSYAE	o	o	M	Xvsync edge (0=Low/High edge, 1=High/Low edge)	0
		25	HSYAE	o	o	M	Xhsync edge (0=Low/High edge, 1=High/Low edge)	0
		24	IES	o	o	M	0: Internal Sync 1: External Sync	0
		18	TDM		o	M	1: Enable Twin Display mode	0
		17:16	SM	o	o	M	Scan mode	0
		12	FTE	o	o	M	1: Field toggle enable	0
		11:8	BSC	o	o	M	Bitstream format code	0x0
		4	RBSW	o	o	M	Swap R/B channel for RGB111	0
		3:0	CSC	o	o	M	Physical colour space code	0x0
PHSCAN	0x3008	29:16	SCLK	o	o	M	Physical size in scan clocks (Pixel*BPP/BitsPerScan- clock)	0
DUALSCOF	0x300C	13:0	OFS	o	o	M	Dual scan Y offset	0
MTIMODD [2]	0x3010 - 0x3014						Master timing, odd field, First word start, next word stop.	
		30:16	o	o	o	M	o dimension (2's comple- ment)	0x0000
		14:0	y	o	o	M	Y dimension (2's comple- ment)	0x0000
MTMEVEN [2]	0x3018 - 0x301C						Master timing, even field, First word start, next word stop. (o values from MTI - MODD[0,1])	
		14:0	Y	o	o	M	Y dimension (2's comple- ment)	0x0000
MTIMON	0x3020	0	ON	o	o		Master timing switch (0=off,1=on)	0

Table 1-2: Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
TIMDIAG	0x3024						Diagnostic timing position output (read only)	
		30:16	o	o	o	R	Current o position (2's complement)	0x0000
		15	FIELD	o	o	R	Current field	0
		14:0	Y	o	o	R	Current Y position (2's complement)	0x0000
SPG [6,4]	0x3030 – 0x308C						These registers contain 6 Sync Pulse Generators (SPG0-SPG5) with each 4 registers.	0
SPGPSON [6]	0: 0x3030, 1: 0x3040, 2: 0x3050, 3: 0x3060, 4: 0x3070, 5: 0x3080						Position to switch on	
		30:16	o	o	o		o position (2's complement)	0x0000
		15	FIELD	o	o		Field flag 0: odd; 1: even	0
		14:0	Y	o	o		Y position (2's complement)	0x0000
SPGMKON	0: 0x3034, 1: 0x3044, 2: 0x3054, 3: 0x3064, 4: 0x3074, 5: 0x3084						Don't care vector for 'Position on' match. (1=do not include this bit into position matching)	
		30:16	o	o	o		o mask	0x0000
		15	FIELD	o	o		Field mask	0
		14:0	Y	o	o		Y mask	0x0000
SPGPSOF	0: 0x3038, 1: 0x3048, 2: 0x3058, 3: 0x3068, 4: 0x3078, 5: 0x3088						Position to switch off	
		30:16	o	o	o		o position (2's complement)	0x0000
		15	FIELD	o	o		Field flag (0=odd, 1=even field)	0
		14:0	Y	o	o		Y position (2's complement)	0x0000

**Table 1-2:** Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
SPGMKOF	0: 0x303C,						Don't care vector for 'Position off' match (1=do not include this bit into position matching)	
	1: 0x304C,							
	2: 0x305C,	30:16	o	o	o		o mask	0x0000
	3: 0x306C,	15	FIELD	o	o		Field mask	0
	4: 0x307C,	14:0	Y	o	o		Y mask	0x0000
	5: 0x308C							
SSQCYCLE	0x30FC	5:0	SC		o		Actual length of sequencer cycle (SC = Num -1)	0x0
		4:0		o				
SSQCNTS [64]	0x3100 - 0x31FC						Sync sequencer RAM contents (64 Words)	
		31	OUT	o	o		Output value for scan position	undef
		30:16	SEQX	o	o		o scan position (2's complement)	undef
		15	FIELD	o	o		Field flag (0: odd;1: even)	undef
		14:0	SEQY	o	o		Y scan position (2's complement)	undef
SMX [8,2]	0x3200 - 0x323C						Array definition for sync mixers (SMX0-SMX7 with each 2 registers)	0
SMXSIGS	0: 0x3200,						Sync mixer	
	1: 0x3208,	14:12	S4	o	o		Signal select Signal to select: 0:const.zero 1:Sequencer out 2...7: Output of SPG0...SPG5	0
	2: 0x3210,	11:9	S3	o	o			0
	3: 0x3218,	8:6	S2	o	o			0
	4: 0x3220,	5:3	S1	o	o			0
	5: 0x3228,	2:0	S0	o	o			0
	6: 0x3230,							
	7: 0x3238							

Table 1-2: Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
SMXFCT	0:						Function table	
	0x3204, 1: 0x320C, 2: 0x3214, 3: 0x321C, 4: 0x3224, 5: 0x322C, 6: 0x3234, 7: 0x323C	31:0	FT	o	o		Output value = function_table[a] $a = S4 * 2^4 + S3 * 2^3 + S2 * 2^2 + S1 * 2^1 + S0 * 2^0$ Sn: Value (binary) of signal selected with SMXSIGS_Sn	0x00000000
SSWITCH	0x3240						Output signal delay (sync switch) register	
		7:0	CD	o	o	M	Sync switch, (0=no; 1=0.5 display clock (CLKD) cycles delay)	0x00
PIXCLKGT	0x3248						Pixel clock gate register; gate is output of SM7	
		3	HC	o	o	M	Clock divider (0=1:1; 1=1:2)	0
		2	CP	o	o	M	Clock polarity (0=true; 1=inverted)	0
		1	GON	o	o	M	Clock gate enable (1=on/ 0=off)	0
		0	GT	o	o	M	Gate type (0=And; 1=Or)	0
CKLOW	0x3250						Colour key lower limits (according to physical colour space) Pin DIS_CKEY is activated when all pixel channels lie within their limits (including limits).	
		24	OP	o	o		Key out polarity (0=active high, 1=active low)	0
		23:16	LLR	o	o		Red channel	0x00
		15:8	LLG	o	o		Green channel	0x00
		7:0	LLB	o	o		Blue/monochrome channel	0x00



**Table 1-2:** Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
CKUP	0x3254						Colour key upper limits (according to physical colour space) Pin DIS_CKEY is activated when all pixel channels lie within their limits (including limits).	
		23:16	ULR	o	o		Red channel	0x00
		15:8	ULG	o	o		Green channel	0x00
		7:0	ULB	o	o		Blue/monochrome channel	0x00
ACLAMP	0x3258						Jasmine: Clamping values for analog outputs (DACs) Lavender: Clamping values for analog outputs (DACs) and digital output	
		23:16	ACLR		o		Red channel	0x00
		15:8	ACLG		o		Green channel	0x00
		7:0	ACLB		o		Blue channel	0x00
		23:0	DAO	o			Clamping value for analog and digital output	0x000000
DCLAMP	0x325C	23:0	DCL		o		Blanking clamping value for digital outputs	0x000000
MAINEN	0x3260						Main display output enable flags	
		29	REFOE		o		1: (internal) reference voltage disable 0: (internal) reference voltage enable For Lavender this reference is connected to DACOE.	0
		28	DACOE	o	o		1: DAC output (A_BLUE, A_GREEN, A_RED) enable	0
		27	CKOE	o	o		1: DIS_CKEY output enable	0
		26	VROE	o	o		1: DIS_VREF output enable	0
		25	VSOE	o	o		1: DIS_VSYNC output enable	0
		24	HSOE	o	o		1: DIS_HSYNC output enable	0
		23:0	DOE	o	o		1: DIS_D[23:0] output enable	0x000000

**Table 1-2:** Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
GPU-SDCR (SDC Record)								
SDCP	0x3270						SDRAM Controller request priorities	
		15:8	I FL	o	o	R	Input FIFO load (read-only)	0x00
		6:4	HP	o	o		High priority	0x7
		2:0	LP	o	o		Low priority	0x3
VIC (Video Interface Controller)								
VICSTART	0x4000						Input layer start coordinates (memory offset for video)	
		29:16	o	o	o		o offset	0x0000
		13:0	Y	o	o		Y offset	0x0000
VICALPHA	0x4004						Replace colour when alpha pin (VSC_ALPHA) is active Colour depth depends on layer	
		23:0	COL	o	o		Alpha colour	0x000000

**Table 1-2:** Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
VICCTRL	0x4008						Input control word	
		7	BSWAP		0		Swap external channels A and B to internal channels iA and iB. 0: A->iA, B->iB 1: A->iB, B->iA	0
		6	ALEN	0	0		1:Enable alpha; 0:Disable alpha	0
		5	PORT	0	0		Port mode 1: Double port (port iA and iB) 0: Single port (port iA only)	1
		4	CLOCK	0	0		Video Clock (VSC_CLKV) mode for data sampling 1: Double clock mode (both edges) 0: Single clock mode (rising edge only)	0
		3 : 0	MODE	0	0		Colour mode for video input: 0x4: RGB555 0x5: RGB565 0x6: RGB888 0x7: YUV422 0x8: YUV444 0xE: YUV555 0xF: YUV655	0x6

Table 1-2: Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
VICFCTRL	0x400C						Field control word	
		22	ODDFST	o	o		Field order within a Frame: 1:Odd field is top field 0:Even field is top field	0
		21	FRAME	o	o		Video mode 1:Frame mode (interleave fields in one layer) 0:Field mode (store one field in one layer)	0
		20	SKIP	o	o		Field skip enable for selected fields 1: Skip every 2nd filed of each type 0:Use every field	0
		18	VICEN	o	o		1: General VIC enable	0
		17	EVENEN	o	o		1: Enable even fields	0
		16	ODDEN	o	o		1: Enable odd fields	0
		11:8	TRD	o	o		3rd layer number	0x7
		7:4	SEC	o	o		2nd layer number	0x0
		3:0	FST	o	o		1st layer number	0x6
VICPCTRL	0x4010						Polarity settings for video control pins	
		3	ALPHA	o	o		VSC_ALPHA: 1: Low active 0: High active	0
		2	FIELD	o	o		VSC_IDENT: 1: Odd field low active 0: Odd field high active	0
		1	VACT	o	o		VSC_VACT: 1: Low active, 0: High active	0
		0	VREF	o	o		VSC_VREF: 1: Low active, 0: High active	0

**Table 1-2:** Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
VICFSYNC	0x4014						Control register for video I/O synchronization	
		16	REL	o	o		1: VIC is faster than GPU 0: GPU is faster than VIC	0
		15	SYNC	o	o		1: Three layer mode 0: Two layer mode	0
		14:0	SWL	o	o		Switch level for layer switch in two layer sync mode	0
VICCSYNC	0x4018						Control Word for clock synchronization (Lavender only)	
		1:0	MODE	o			Sync Mode 00: Core Clock > Video Clock 01: Core Clock > 2*Video Clock 10: Core Clock = Video Clock 11: Reserved	00
SDRAM	0x401C						SDRAM request priority control register	
		2:0	LP	o	o		Low priority	0x2
		6:4	HP	o	o		High priority	0x6
VICBSTA	0x4020			o	o	R	VIC status register for test purpose (read only)	
VICRLAY	0x4024						Video layer debug register for test purpose (read only)	
		19:16	AOVL		o	R	Current output layer (to GPU)	undef
		11:8	LIVL		o	R	Last input layer (VIC)	undef
		3:0	AIVL		o	R	Current input layer (VIC)	undef

Table 1-2: Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
VICVISYN	0x4028						Video path control register	
		25:24	DEL		o		Negative data delay with respect to VSC_VACT signal	0
		20:16	SHUFF		o		Bus shuffler for data ports iA, iB, iA_delayed and iA_negedge	0
		8	START		o		Selector for VIC_SYNC flag <sup>a</sup> source: 0: Video write start 1: Video read start	0
		1:0	SEL		o		Selector for video input protocol: 00: VPX 01: CCIR-TRC 10: CCIR external sync 11: reserved	00
VICLIMEN	0x402C	0	LIMENA		o		1: Enable video limitation unit	0
VICLIMH	0x4030						Horizontal limitation settings	
		26:16	HEN		o		Horizontal window length including offset	0x000
		10:0	HOFF		o		Horizontal window offset	0x000
VICLIMV	0x4034						Vertical limitation settings	
		26:16	VEN		o		Vertical window length including offset	0x000
		10:0	VOFF		o		Vertical window offset	0x000

**Table 1-2:** Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
EXTPCTRL	0x4038						Polarity control register for CCIR mode with external video synchronization.	
		3	ALPHA		o		Alpha (pin VSC_ALPHA) 0: high active 1: low active	0
		2	PARITY		o		Parity (pin VSC_IDENT) 0: polarity unchanged 1: invert polarity	0
		1	HREF		o		Horizontal reference signal (pin VSC_VACT) 0: high active 1: low active	0
		0	VREF		o		Vertical reference signal (pin VSC_VREF) 0: high active 1: low active	0
PP (Pixel Processor)								
BGCOL	0x4100						Background pixel colour	
		24	EN	o	o	D	1: Enable background colour	0
		23:0	COL	o	o	D	Background colour data	0x000000
FGCOL	0x4104	23:0		o	o	D	Foreground pixel colour	0x000000
IGNORCOL	0x4108						Ignored pixel colour (PutBM, PutCP)	
		24	EN	o	o	D	1: enable ignore colour	0
		23:0	COL	o	o	D	Ignore colour data	0x000000
LINECOL	0x410C	23:0	COL	o	o	D	Line colour (DwLine)	0x000000
PIXCOL	0x4110	23:0	COL	o	o	D	Colour for pixel with fixed colour (PutPxFC)	0x000000
PLCOL	0x4114	23:0	COL	o	o	D	Polygon colour (DwPoly)	0x000000
RECTCCOL	0x4118	23:0	COL	o	o	D	Rectangle colour (DwRect)	0x000000
XYMAX	0x411C						Pixel stop address for pixel processor bitmap commands (PutBM, PutCP, PutTxtBM, PutTxtCP)	
		29:16	XMAX	o	o	D	o dimension for stop point	0x0000
		13:0	YMAX	o	o	D	Y dimension for stop point	0x0000

Table 1-2: Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
XYMIN	0x4120						Start address for pixel processor bitmap commands (PutBM, PutCP, PutTxtBM, PutTxtCP)	
		29:16	XMIN	o	o	D	o dimension for start point	0x0000
		13:0	YMIN	o	o	D	Y dimension for start point	0x0000
PPCMD	0x4124						Configuration for pixel processor commands	
		28	ULAY		o	D	1: Use target layer for drawing commands	0
		27:24	LAY	o	o	D	Target layer for commands PutBM, PutCP, PutTxtBM, PutTxtCP	0x0
		8	DIR	o	o	D	Direction for sequential commands PutBM, PutCP, PutTxtBM, PutTxtCP 0: Horizontal, 1:Vertical	0
		1:0	MIR	o	o	D	Mirror for sequential commands PutBM, PutCP, PutTxtBM, PutTxtCP 00: No mirror, 01: o-mirror, 10: Y-mirror, 11:XY-mirror	0x0
SDCPRIO	0x4128	2:0	PRI0	o	o	D	Priority for SDC interface	0x0
REQCNT	0x412C	7:0	MFB	o	o	D	MFB+1: Minimum FIFO-block size before activating a SDRAM request (0<=MFB<64)	0x00
READINIT	0x4130	0	RCR	o	o		Read control registers for pixel processor (address range: 0x4100-0x4130) 0: read back PP internal register 1: read back PP user writeable register	0
<b>DIPA (Direct and Indirect Physical memory Access)</b>								
DIPACtrl	0x4200						DIPA control register	
		18:16	PDPA	o	o		DPA priority for SDC access	0x0000
		2:0	PIPA	o	o		IPA priority for SDC access	0x0000



**Table 1-2:** Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
DIPAIF	0x4204						IPA input FIFO control register	
		23:16	MAX	o			Input FIFO max. block size	0x0002
		22:16			o			
		7:0	MIN	o			Input FIFO min. block size	0x0002
		6:0			o			
DIPAOF	0x4208						IPA output FIFO control register	
		23:16	MAX	o			Output FIFO max. block size	0x0001
		22:16			o			
		7:0	MIN	o			Output FIFO min. block size	0x0001
		6:0			o			
CCFL (Cold Cathode Fluorescence Light Driver)								
CCFL1	0x4400						CCFL control register	
		28	SSEL		o		Synchronization select: 0: use SNCS flag 1: sync to output of GPU sync mixer 5	0
		27	EN	o	o		1: CCFL enable	0
		26	PROT	o	o		1: Protect old settings during configuration	0
		25	SNCS	o	o		Synchronization select: 0: internal (vsync from GPU), 1: software	0
		24	SYNC	o	o		1: Synchronization trigger by software	0
		7:0	SCL	o	o		Timebase scale factor (Derived from System clock (CLKK))	0x00
CCFL2	0x4404						CCFL Duration Register (Unit: 4 Timebase Clocks)	
		31:16	FLS	o	o		Flash Duration	0x0000
		15:8	PSE	o	o		Pause Duration	0x00
		7:0	IGNT	o	o		Ignition Duration	0x00
AAF (Anti Aliasing Filter)								

**Table 1-2:** Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
AATR	0x4500						Thresholds for red channel	
		15:8	TH2	○	○		Threshold2	0x00
		7:0	TH1	○	○		Threshold1	0xFF
AATG	0x4504						Thresholds for green channel	
		15:8	TH2	○	○		Threshold2	0x00
		7:0	TH1	○	○		Threshold1	0xFF
AATB	0x4508						Thresholds for blue channel	
		15:8	TH2	○	○		Threshold2	0x00
		7:0	TH1	○	○		Threshold1	0xFF
AAOE	0x450C						AAF options	
		1	BX4		○		1: AAF block size 4x4 0: AAF block size 2x2	0
		0	WB0	○	○		1: Enable write back optimization	0
CU (Clock Unit)								

**Table 1-2:** Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
CLKCR	0xFC00						Clock configuration register	
		31:30	DCS	o	o		Direct clock source 00: Crystal (pins OSC_IN and OSC_OUT), 01: Pixel clock (pin DIS_PXCLK) 10: MCU clock (pin ULB_CLK) 11: Reserved clock (pin RCLK)	00
		29:24	SCP	o	o		System clock (CLKK) prescaler	0
		23:22	PCS	o	o		PLL clock source 00: Crystal (pins OSC_IN and OSC_OUT), 01: Pixel clock (pin DIS_PXCLK) 10: MCU clock (pin ULB_CLK) 11: Reserved clock (pin RCLK)	00
		21:16	PFD	o	o		PLL feedback divider	0x00
		15	SCSL	o	o		System clock (CLKK) select 0: direct clock source 1: PLL clock source	0
		14	PCSL	o	o		Pixel clock (CLKD) select 0: direct clock source 1: PLL clock source	0
		13	IPC	o	o		Pixel clock (CLKD) invert 0: not inverted 1: inverted	0
		12	PCOD	o	o		Pixel clock output (DIS_PXCLK) disable 0: internal pixel clock (CLKD output) 1: external pixel clock (DIS_PXCLK input)	1
		11	DBG		o		1: Core clock (CLKK) debug mode (output at pin SPB_TST)	0
		10:0	PCP	o	o		Pixel clock prescaler value	0x000

Table 1-2: Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
CLKPDR	0xFC04						Clock power down register	
		31:24	ID	o	o	R	Chip ID (read only) 00: MB87J2120 (Lavender) 01: MB87P2020(-A) (Jasmine)	01
		23:16	SID	o	o	R	Chip Sub-ID (read only) 00: MB87J2120 (Lavender) and MB87P2020 (Jasmine) 01: MB87P2020-A (Jasmine Redesign)	01
		15	MRST		o	W	Master hardware reset Write 1: Start Reset Write 0: Stop Reset Read: returns always '0'	0
				o		F	Master hardware reset Write 1: Start Reset (stops automatically) Write 0: No effect Read: returns current value	
		14	LCK		o	R	PLL lock (read only) 1: PLL has locked to input frequency	undef
		13	-		o		reserved; set to 0	0
		12	VII		o		1: Invert Video Clock (CLKV)	0
				o		R	PLL lock (read only) 1: PLL has locked to input frequency	undef
		11	RUN	o	o		PLL enable 1: PLL on 0: PLL off	0
		10	GPU	o	o		1: Enable GPU clocks	0
		9	ULB	o	o		1: Enable ULB clocks	0
		8	SPB	o	o		1: Enable SPB clock	0
		7	CCFL	o	o		1: Enable CCFL clock	0
		6	SDC	o	o		1: Enable SDC clock	0
		5	VIS	o	o		1: Enable VIC clocks	0
		4	DIPA	o	o		1: Enable DIPA clock	0
		3	AAF	o	o		1: Enable AAF clock	0
		2	MCP	o	o		1: Enable PP-MCP clock	0

**Table 1-2:** Register address space for Jasmine and Lavender

Register		Bits	Group Name	Lavender	Jasmine	Special	Description	Default value
Name	Address							
		1	MAU	0	0		1: Enable PP-MAU clock	0
		0	PE	0	0		1: Enable PP-PE clock	0

- a. See chapter 2 for a description of bit groups for flags.
- b. Dynamic behaviour means that a hardware flag reset is possible.
- c. Setting in '()' is an optimized setting if no AAF is used.



## 2 Flag Description

Table 2-1 contains all flags for MB87P2020-A (Jasmine) and MB87J2120 (Lavender).

In order to avoid data inconsistencies during bit masking within flag register a mask (and/or gating) process is implemented in hardware for flag register. To distinguish between flag set-, reset- and direct write access different addresses are used<sup>1</sup>:

- FLNOM (0x000C):normal write operation
- FLRST (0x0010):reset operation (1: reset flag on specified position; 0: don't touch)
- FLSET (0x0014):set operation (1: set flag on specified position; 0: don't touch)

All of these three addresses write physically to one register with three different methods.

For reading all three addresses return the value of flag register.

Every flag can have a different reset behaviour. With help of register FLAGRES the application can choose whether the hardware is allowed to reset the desired flag (*dynamic* behaviour, FLAGRES\_x=1) or not (*static* behaviour, FLAGRES\_x=0). With dynamic behaviour the flag follows the driving hardware signal while with static behaviour the application is responsible for resetting the flag in order to catch next event. In table 2-1 the default reset behaviour at system start up is given in last column.

Note that some flags are only available for Jasmine..

**Table 2-1:** Flags for MB87J2120 (Lavender) and MB87P2020-A (Jasmine)

Name	Bit	Lavender	Jasmine	Description	Default behaviour (FLAGRES)
VICSYN	31		X	A frame or field has been written to or read from SDRAM Which event is signalled by this flag depends on VIC settings: VICFCTRL_FRAME determines the storage type within SDRAM (field or frame). For details see VIC description and register list. VICVISYN_START determines whether a write or read start should trigger the flag	static (0)
ERDY	30		X	RDY timeout error has occurred See ULB description and register description for RDYTO and RDYADDR	static (0)
RDPA	29	X	X	1: DPA write access is enabled. This flag has to be polled before each DPA (write-) access to ensure a save SDRAM access <sup>a</sup> . <b>Otherwise data loss may occur.</b>	dynamic (1)
FDPA	28	X	X	1: DPA has finished SDRAM access and is ready for next one.	static (0)
RIPA	27	X	X	1: IPA is ready for command execution	static (0)
RMCP	26	X	X	1: MCP is ready for command execution	static (0)

1. Additionally all access types (word, halfword and byte) are possible for each of these addresses.

**Table 2-1:** Flags for MB87J2120 (Lavender) and MB87P2020-A (Jasmine)

Name	Bit	Lavender	Jasmine	Description	Default behaviour (FLAGRES)
RMAU	25	X	X	1: MAU is ready for command execution	static (0)
RPE	24	X	X	1: PE is ready for command execution	static (0)
EBPP	23	X	X	1: Colour depth for source- and target layer is different during MemCP command In this error case MCP reads data from input FIFO but performs no further actions.	static (0)
CWEN	22	X	X	1: Command register can be written. This flag has to be polled before a command can be written <sup>a</sup> . <b>Otherwise data loss and synchronization loss between Jasmine/Lavender and MCU may occur.</b>	dynamic (1)
EINT0	21	X	X	1: Lavender/Jasmine external interrupt occurred This interrupt is currently assigned to SPB device which is implemented on chip but outside the Lavender/Jasmine core.	static (0)
STOUT	20	X	X	After reset: Flag is set ('1') when SDRAM initialisation time is over Else: Flag is set ('1') when SDC forces SDRAM refresh. This indicates a high SDRAM bus load.	static (0)
GSYNC	19	X	X	This flag is directly connected to GPU Sync Mixer 6 <sup>b</sup> . The Sync Mixer default settings generate no sync signal.	static (0)
BWVIO	18	X	X	1: GPU bandwidth violation occurred which means that the GPU didn't receive requested data from SDRAM. This flag indicates a high SDRAM bus load.	static (0)
EOV	17	X	X	1: A command pipeline overflow has occurred. A command was sent to Jasmine while CWEN flag was '0'.	static (0)
ECODE	16	X	X	1: Wrong error code was written to command register This command code is internally treated as NoOp command.	static (0)
EDATA	15		X	1: Execution device (PP or IPA) tried to read from empty input FIFO This may indicate a wrong behaviour of execution device.	static (0)
BDPA	12	X	X	1: DPA is performing an SDRAM access.	static (0)
BIPA	11	X	X	1: IPA is executing a command	static (0)
BMCP	10	X	X	1: MCP is executing a command	static (0)
BMAU	9	X	X	1: MAU is executing a command	static (0)



**Table 2-1:** Flags for MB87J2120 (Lavender) and MB87P2020-A (Jasmine)

Name	Bit	Lavender	Jasmine	Description	Default behaviour (FLAGRES)
BPE	8	X	X	1: PE is executing a command	static (0)
OFL	7	X	X	1: Output FIFO load is equal or lower than programmable output FIFO lower limit (OFUL_LL).	static (0)
OFH	6	X	X	1: Output FIFO load is equal or higher than programmable output FIFO upper limit (OFUL_UL).	static (0)
OFE	5	X	X	1: Output FIFO is empty.	static (0)
OFF	4	X	X	1: Output FIFO is full.	static (0)
IFL	3	X	X	1: Input FIFO load is equal or lower than programmable input FIFO lower limit (IFUL_LL).	static (0)
IFH	2	X	X	1: Input FIFO load is equal or higher than programmable input FIFO upper limit (IFUL_UL).	static (0)
IFE	1	X	X	1: Input FIFO is empty.	static (0)
IFF	0	X	X	1: Input FIFO is full.	static (0)

- a. Alternative this flag can cause an interrupt and writing can be done inside Interrupt Service Routine (ISR).
- b. See GPU description for details about Sync Mixer settings.





**Table 3-1:** Command List

Mnemonic	Code	Device	Function	Registers
PutCP	02h	PE	<p>Store an compressed bitmap (TGA run-length coded) in Video RAM (finite command).</p> <p>Source:  IFIFO with n*[bitmap RLE data]  n depends on compression factor. If number of decompressed pixels is not sufficient  <math>(X_{max}-X_{min}+1)*(Y_{max}-Y_{min}+1)</math>, PE will not become ready and waits for data.</p> <p>Target:  Video RAM area which is defined by {Xmax, Xmin}, {Ymax, Ymin}, PPCMD_LAY and PPCMD_MIR. Pixel with IGNORCOL_COL are not written if IGNORCOL_EN = '1'.</p>	XYMAX XYMIN IGNORCOL_EN IGNORCOL_COL PPCMD_LAY PPCMD_EN PPCMD_DIR PPCMD_MIR  IFIFO
DwLine	03h	PE	<p>Draw Lines. The calculated pixel data are stored into Video RAM (infinite command). Colour is defined by LINECOL.</p> <p>Source:  ULB input fifo with start and end points n*([Xs,Ys], [Le,Xe,Ye]), n = number of lines. Layer of end point ignored if PPCMD_ULAY is set. Then PPCMD_LAY is used instead of.</p> <p>Target:  Line of pixels between start and end point in selected layer in Video RAM.</p> <p>Notes:  LINECOL or PPCMD changes apply after new command only. Layer information in pixel address can change with each line. Only relevant bits of LINECOL register are used, depending on colour depth.</p>	LINECOL PPCMD_LAY PPCMD_ULAY PPCMD_EN  IFIFO

**Table 3-1:** Command List

Mnemonic	Code	Device	Function	Registers
DwRect	04h	PE	<p>Draw Rectangles. The calculated pixel data are stored into Video RAM (infinite command). Colour is defined by RECTCOL.</p> <p>Source: ULB input fifo with start and end point <math>n * ([Xs, Ys], [Le, Xe, Ye])</math>, <math>n</math> = number of rectangles. Layer of endpoint ignored if PPCMD_ULAY is set. Then PPCMD_LAY is used instead of.</p> <p>Target: Rectangular area of pixels between start and end point in selected layer in Video RAM.</p> <p>Notes: RECTCOL or PPCMD changes apply after new command only. Layer information in pixel address can change with each line. Only relevant bits of RECTCOL register are used, depending on colour depth.</p>	<p>RECTCOL PPCMD_LAY PPCMD_ULAY PPCMD_EN</p> <p>IFIFO</p>
PutTxtBM	05h	PE	<p>Store uncompressed pixel data with fixed foreground or background Colour into Video RAM (finite command).</p> <p>Source: ULB input fifo with <math>n * [\text{colour enable data}]</math>  <math>n = (X_{\text{max}} - X_{\text{min}} + 1) * (Y_{\text{max}} - Y_{\text{min}} + 1) / 32</math></p> <p>Target: Rectangular area of pixels between start and end point in selected layer in Video RAM. Background pixels are written only if BGCOL_EN = '1'.</p> <p>Note: Only the relevant bits of BGCOL and FGCOL register are used (depends on the colour depth).</p>	<p>XYMAX XYMIN FGCOL BGCOL_EN BGCOL_COL PPCMD_LAY PPCMD_EN PPCMD_DIR PPCMD_MIR</p> <p>IFIFO</p>

**Table 3-1:** Command List

Mnemonic	Code	Device	Function	Registers
PutTxtCP	06h	PE	<p>Store compressed pixel data with fixed foreground or background colour into Video RAM (finite command).</p> <p>Source: ULB input fifo with n*[coded colour enable data], n depends on compression factor</p> <p>Target: Rectangular area of pixels between start and end point in selected layer in Video RAM. Background pixels are written only if BGCOL_EN = '1'.</p> <p>Note: Only the relevant bits of BGCOL and FGCOL register are used (depends on the colour depth).</p>	<p>XYMAX XYMIN FGCOL BGCOL_EN BGCOL_COL PPCMD_LAY PPCMD_EN PPCMD_DIR PPCMD_MIR</p> <p>IFIFO</p>
PutPixel	07h	MAU	<p>Transfer single pixel data into Video RAM (infinite command).</p> <p>Source: IFIFO with address/data pairs. n*([L,X,Y], [single colour data]), n = number of pixels</p> <p>Target: Addressed pixel in Video RAM.</p> <p>Note: IFIFO: LSB aligned pixel data, however physical data in Video RAM is MSB aligned</p>	<p>PPCMD_EN</p> <p>IFIFO</p>
PutPxWd	08h	MAU	<p>Transfer packetized pixel in data words (pixel bursts) into Video RAM (infinite command).</p> <p>Source: IFIFO with n*([L,X,Y],[data word]) n = number of packet words</p> <p>Target: Successive pixels in Video RAM on addressed position.</p> <p>Note: The number of pixels in the 32-bit word depends on colour depth/layer. In Opposition to PutPixel command IFIFO data is directly in Video RAM format. Especially for 24 bit colour depth IFIFO data is MSB aligned in this case.</p>	<p>PPCMD_EN</p> <p>IFIFO</p>

**Table 3-1:** Command List

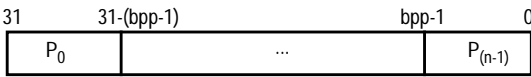
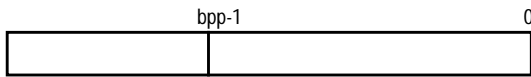
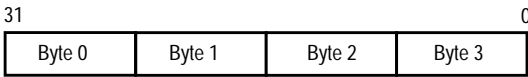
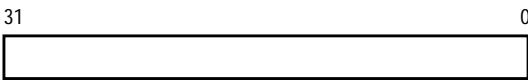
Mnemonic	Code	Device	Function	Registers
PutPxFC	09h	MAU	<p>Set fixed coloured pixels in Video RAM at appropriate addresses (infinite command). Colour is defined by PIXCOL register.</p> <p>Source: IFIFO with pixel addresses <math>n * ([L, X, Y])</math>, <math>n</math> = number of pixels</p> <p>Target: Addressed pixels in Video RAM.</p> <p>Note: Only the relevant bits of PIXCOL register are used (depends on the colour depth).</p>	PIXCOL PPCMD_EN  IFIFO
GetPixel	0Ah	MAU	<p>Read pixel data from Video RAM (infinite command).</p> <p>Source: IFIFO with pixel addresses, <math>n * ([L, X, Y])</math>, <math>n</math> = number of pixels. Video RAM with pixel data from pixel address.</p> <p>Target: OFIFO, <math>n * ([\text{single colour data}])</math>.</p>	IFIFO  OFIFO
XChPixel	0Bh	MAU	<p>Transfer single pixel data into Video RAM and read the old value from Video RAM (infinite command).</p> <p>Source: IFIFO with pixel address/data pairs, <math>n * ([L, X, Y], [\text{single colour data}])</math>, <math>n</math> = number of pixels. Old pixel data from Video RAM.</p> <p>Target: Video RAM for new pixel from IFIFO. OFIFO for old pixel from Video RAM.</p>	IFIFO  OFIFO

**Table 3-1:** Command List

Mnemonic	Code	Device	Function	Registers
MemCP	0Ch	MCP	<p>Copy rectangular region from Video RAM to Video RAM (infinite command).</p> <p>Source: Video RAM area defined by start point [Ls,Xs,Ys] and end point [Xe,Ye]. IFIFO control parameter: n*([Ls,Xs,Ys], [Xe,Ye], [Lt,Xt,Yt]), n = number of rectangular areas to copy</p> <p>Target: VideoRam area with start point [Lt,Xt,Yt] of same size as source area.</p> <p>Note: Layer information of source end point is ignored in pixel address. There is no end point of target area to specify. Source and target layer should have same colour depth. If mismatch detected, the error flag FLNOM_EBPP is activated.</p>	IFIFO
PutPA	0Dh	IPA	<p>Store data in Video RAM physically with address auto-increment (infinite command).</p> <p>Source: IFIFO with ([physical address], n*[physical data]).</p> <p>Target: Video RAM on physical address and succeeding. Physical address means there is no direct point relation.</p>	IFIFO
GetPA	0Eh	IPA	<p>Load data from Video RAM with address auto-increment, stop after CMD_PAR words.</p> <p>Source: IFIFO with [physical address] Video RAM address starting at physical address and succeeding words. Number of words given by CMD_PAR.</p> <p>Target: Physical data in OFIFO.</p>	CMD_PAR IFIFO OFIFO



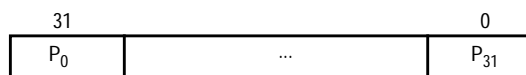
Mnemonic	Code	Device	Function	Registers
DwPoly	0Fh	PE	<p>Draw polygon lines. Calculated pixel are stored in Video RAM (infinite command).</p> <p>Source:  IFIFO with start and next points of a polygon (<math>[X_s, Y_s]</math>, <math>n*[L_n, X_n, Y_n]</math>).  <math>n</math> = number of polygon lines.  Each 'next' point is the start point of the next polygon line.</p> <p>Target:  Polygon pixels in Video RAM.</p> <p>Note:  Drawing of more than one polygon on different layers with one DwPoly command is possible by changing layer of next point.  The colour of the polygon can't change in same command cycle.  Only the relevant bits of PLCOL register are used (depends on colour depth).</p>	PLCOL PPCMD_LAY PPCMD_ULAY PPCMD_EN  IFIFO
NoOp	FFh	ULB	No operation. Dummy cycle for command controller.	-



24	1 *
16	2
8	4
4	8
2	16
1	32

\* In case of 24 bpp colour data is packet over word boundaries. There are no lower bits left empty and first pixel is MSB aligned on the 32-bit word grid.

- colour enable data

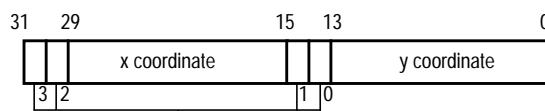


(...) Source or Target FIFO data

[...]n Deliver data in '['...' n times

[...]+ Deliver data in '['...' at least one time

- {L,X,Y} Pixel coordinates, point definition



Layer (to be ignored if layer register exists)

If L is not stated i.e. in {X,Y} form, information in bits 31, 30, 15, 14 kept as don't care.

## 3.2 Command and I/O Control

Command queue consists of two registers, an internal actual processed command and an additional buffer for the next planned command. Before writing to command register the FLNOM\_CWEN (command write enable) flag polling is required to ensure that the new command can be written. All execution devices provide ready or busy information via FLNOM flag register. Normally this dedicated device information is not needed to evaluate. Command controller takes care on operation.

Also the IFIFO and OFIFO has its status flags in FLNOM. These are usable for I/O flow control by polling or interrupt. Additional DMA support is possible to transfer data from/to the FIFOs.

---

## ***D-2 Hints and restrictions for Laven- der and Jasmine***



# 1 Special hints

## 1.1 IPA resistance against wrong settings

Invalid settings for minimum transfer block sizes lead to IPA deadlock from which it can only recover by reset. At minimum a value of 1 has to be given for the output FIFO (block size of zero makes no sense) and a minimum value of 2 for the input FIFO should be setup. Input FIFO needs at least one address/data pair for IPA.

Table 1-1 gives an overview and a classification about the described problem.

**Table 1-1:** Overview for IPA resistance

Subject	Description
Description	Smaller block sizes does not make sense since a block should contain at least one word. Input FIFO contains address and at least one data word. Output FIFO contains at least one data word.
Classification	Hint
Effects without workaround	The system may hang with wrong settings. Do not use forbidden settings. Escape only with HW-Reset.
Solution/Workaround	No workaround required.
Concerned devices	MB87J2120 (Lavender) MB87P2020 (Jasmine) fixed for MB87P2020-A (Jasmine redesign)
Testcase	EMDC: IPA.1 and SDC.3 (Limits can be set in 'mkctrl'.)

## 1.2 ULB\_DREQ pin timing to host MCU

ULB\_DREQ reaction time is critical if only one wait state for User Logic Bus (ULB) cycle is set up. Writing per DMA 'demand mode' can lead to additional transferred data after ULB\_DREQ tied low. Jasmine can handle this additional data by an two words deep overflow buffer. Thus no data will be lost.

For Lavender the count of waitstates can be set equal or greater than two.

Writing to input FIFO in DMA block/step/burst mode and reading from output FIFO with DMA (both block/step/burst and demand mode) transfers correct amount of data in any case for Jasmine and Lavender.

A detailed description about this topic can be found in hardware manual (chapter B-2.1.7) and in a DMA application note.

Table 1-2 gives an overview and a classification about the described problem.

**Table 1-2:** Overview for ULB\_DREQ timing

Subject	Description
Description	Reaction time for ULB_DREQ pin for write accesses is critical if one wait-state is set up.
Classification	Hint

**Table 1-2:** Overview for ULB\_DREQ timing

Subject	Description
Effects without workaround	One data word more than expected can be delivered by MCU.
Solution/Workaround	For MB87P2020 (Jasmine) a hardware solution is already implemented with a two word overflow buffer. For MB87J2120 (Lavender) the count of waitstates during DMA transfer should be set equal or larger than two.
Concerned devices	MB87J2120 (Lavender) fixed for MB87P2020 (Jasmine) fixed for MB87P2020-A (Jasmine redesign)
Testcase	EMDC: ULB.5

### 1.3 CLKPDR master reset

CLKPDR, bit[15] is write only. Read access returns always '0'. Writing '1' initiates global master reset, writing a '0' releases master reset.

Table 1-3 gives an overview and a classification about the described problem.

**Table 1-3:** Overview for master reset

Subject	Description
Description	The reset status can not read back.
Classification	Hint
Effects without workaround	CLKPDR_MRST is write only. Reading always return '0'.
Solution/Workaround	If the reset status has to be memorized it has to be done externally. Normally this sequence should reset Jasmine/Lavender safely: G0CLKPDR_MRST=1; G0CLKPDR_MRST=0;
Concerned devices	MB87J2120 (Lavender) MB87P2020 (Jasmine) MB87P2020-A (Jasmine redesign)
Testcase	EMDC: CTRL.1 (I/O march)

### 1.4 MAU (Memory Access Unit) commands

For Lavender drawing commands (PutPixel, PutPxWd, PutPxFC, GetPixel, XChPixel, DwLine, DwRect and DwPoly) the target layer has to be always included into data stream (pixel addresses).

For Jasmine only commands PutPixel, PutPxWd, PutPxFC, GetPixel and XChPixel can't use the layer register PPCMD\_LAY while the other drawing commands can use it. MAU has its own dedicated layer management and could not have benefit from the enhancement.<sup>1</sup>

1. The target layer register PPCMD\_LAY can be used for pixel engine commands if PPCMD\_ULAY is set to '1'.

A good workaround is to configure Layer 0 with the same parameters as the target layer<sup>1</sup> and make all drawings to layer 0. The advantage is that no layer number has to be merged into pixel addresses. After drawing access to the layer data is possible via target layer configuration. Also GPU can be set up to target layer.

Table 1-4 gives an overview and a classification about the described problem.

**Table 1-4:** Overview for MAU commands

Subject	Description
Description	For Jasmine target layer register (PPCMD_LAY) is ignored for MAU commands even if PPCMD_ULAY is set to '1'.
Classification	Hint
Effects without workaround	Commands PutPixel, PutPxWd, PutPxFC, GetPixel and XChPixel uses only the target layer delivered in input FIFO. See command list for a detailed command syntax description.
Solution/Workaround	Use Layer 0 as a temporary drawing layer (see description for further details).
Concerned devices	MB87J2120 (Lavender) partly fixed for MB87P2020 (Jasmine) partly fixed for MB87P2020-A (Jasmine redesign)
Testcase	EMDC: SDC.LOG2PHY

## 1.5 Pixel Processor (PP) double buffering

The Pixel Processor contains a double buffering mechanism for its registers which is synchronized to command execution.

Data are always written to configuration register which is not used for command execution. Instead of configuration register directly an internal register is used for data storing during command execution. The time to write internal register is determined by hardware.

For reading from PP registers an application can choose whether to read from configuration register (READINIT\_RCR=1) or to read from internal register (READINIT\_RCR=0).

For bitwise modification of pixel processor configuration registers the double buffering mechanism has to be considered. If parts of the registers should be changed with read-modify-write operations there is the risk to read from internal double buffered register and write the result to the external register back. Data inconsistency could be the result. It is recommended to initialize READINIT\_RCR=1 to have read access to the external registers too (same register is accessed for reading and writing). Note that the reset value is READINIT\_RCR=0 and has not the recommended value.

Table 1-5 gives an overview and a classification about the described problem.

**Table 1-5:** Overview for PP double buffering

Subject	Description
Description	Application note for READINT register and its consequences.
Classification	Hint

1. In this case layer 0 is a temporary layer only; it should not be used for normal drawings.

**Table 1-5:** Overview for PP double buffering

Subject	Description
Effects without workaround	READINIT_RCR=0: read internal register (reset value) READINIT_RCR=1: read configuration register Be careful with Read-Modify-Write accesses; use READINIT_RCR=1 in this case.
Solution/Workaround	No workaround required.
Concerned devices	MB87J2120 (Lavender) MB87P2020 (Jasmine) MB87P2020-A (Jasmine redesign)
Testcase	EMDC: SDC.LOG2PHY

## 1.6 Robustness of ULB\_RDY signal

Disturbances at external bus interface (ULB interface) can cause the ULB\_RDY pin to stay in active state (logic '0') during a bus read access. This stops a MB91360 series MCU completely so that no further commands will be executed (blocking of external bus). Only a MCU watchdog can reset the system.

To avoid this problem the PCB layout should be done very carefully especially for the signals ULB\_CS, ULB\_RDY and ULB\_WRX[3:0]. The following hints may help:

- Place MCU and Lavender as close as possible together on PCB.
- Use separate ground plane on PCB in order to avoid interferences with other signals.
- Place SDRAM and Lavender as close as possible together in order to avoid interferences from SDRAM interface to ULB interface

The interface circuit has been redesigned for Jasmine so that this problem should not occur with this device. Additionally for Jasmine an ULB\_RDY timeout has been added so that a 'hanging' ULB\_RDY can not block the whole system. See hardware manual for further details.

Table 1-6 gives an overview and a classification about the described problem.

**Table 1-6:** Overview for ULB\_RDY robustness

Subject	Description
Description	ULB_RDY signal may hang as a result of signal distortions on PCB.
Classification	Hint
Effects without workaround	A hanging ULB_RDY pin blocks the command execution of a MB91360 series MCU.
Solution/Workaround	Keep special attention to PCB layout. This problem should not occur with Jasmine. Additionally a timeout for ULB_RDY has been added.
Concerned devices	MB87J2120 (Lavender) fixed for MB87P2020 (Jasmine) fixed for MB87P2020-A (Jasmine redesign)
Testcase	Every testcase.



## 1.7 Robustness of command pipeline against software errors

As described in detail in hardware manual a command should only be written to Lavender/Jasmine when the flag FLNOM\_CWEN is '1'.

If this rule is followed the command execution of Lavender works correctly but if a command is written although the FLNOM\_CWEN flag is '0' the command execution of Lavender may hang (command pipeline overflow). Only a hardware reset can solve this situation.

In case of a command pipeline overflow the command dataflow via FIFOs is not automatically corrected but the application can detect a command pipeline overflow with help of the flag FLNOM\_EOV and should perform corrective actions.

Please note that not only wrong written software can cause this problem but also signal distortions on ULB data bus (ULB\_D).

Normally an application polls the FLNOM\_CWEN flag so it may be read many times inside a loop. If at one of these read accesses this flag changes its value from '0' to '1' due to PCB disturbances the software stops polling and sends a new command to the graphic controller. Lavender has not finished executing previous command and a command pipeline overflow occurs.

To avoid this communication problem between MCU and Lavender signal distortions on ULB\_D should be reduced by improving PCB layout as already described in chapter 1.6. Please note that additionally to the signals described in chapter 1.6 the ULB data bus (ULB\_D) is also a critical signal for PCB layout.

The command controller for Jasmine has been redesigned and a command pipeline overflow can not cause the command execution to hang. But the application has still to take care about the command dataflow synchronization as described above.

Table 1-7 gives an overview and a classification about the described problem.

**Table 1-7:** Overview for command pipeline robustness

Subject	Description
Description	If a command is written to Lavender while the flag FLNOM_CWEN is '0' the command pipeline may hang. This can be watched with FLNOM_EOV flag.
Classification	Hint
Effects without workaround	The command controller does not work properly after sending a new command while FLNOM_CWEN = '0'. Only a hardware reset can solve this situation. For Jasmine no hang-up can occur but the command<-> data synchronization can be disturbed.
Solution/Workaround	Write only commands to Lavender when FLNOM_CWEN is '1'. Keep special attention to PCB layout for ULB data bus (ULB_D).
Concerned devices	MB87J2120 (Lavender) fixed for MB87P2020 (Jasmine) fixed for MB87P2020-A (Jasmine redesign)
Testcase	Every testcase with command execution.

## 1.8 DMA resistance against wrong settings

In DMA demand mode the DMA controller inside Lavender/Jasmine counts the amount of data words to be transferred. See hardware manual chapter B-2.1.7 (ULB description) for more details about DMA.

If the amount of data to be transferred in DMA demand mode is equal to '0' the Lavender DMA controller may hang. The data amount of '0' can be calculated<sup>1</sup> if:

- IFDMA\_LL is set to max. input FIFO size (128 for Lavender) for DMA to input FIFO

- OFDMA\_UL is set to '0' for DMA from output FIFO.

In order to avoid this problem the described settings are forbidden for Lavender.

This problem has been solved for Jasmine.

Table 1-8 gives an overview and a classification about the described problem.

**Table 1-8:** Overview for DMA resistance against wrong settings

Subject	Description
Description	The DMA controller for Lavender may hang if transfer size in DMA demand mode is equal to '0'.
Classification	Hint
Effects without workaround	No DMA transfer is started even if the start condition is met <sup>a</sup> .
Solution/Workaround	Avoid critical settings IFDMA_LL = 128 and OFDMA_UL = 0. For Jasmine this problem has been fixed.
Concerned devices	MB87J2120 (Lavender) fixed for MB87P2020 (Jasmine) fixed for MB87P2020-A (Jasmine redesign)
Testcase	EMDC: ULB.5 and ULB.13

- a. Start condition for input FIFO is: FIFO load  $\leq$  IFDMA\_LL; for output FIFO: FIFO load  $\geq$  OFDMA\_UL.

---

1. For calculation of data amount not the programmed limits are used but the current FIFO load. Therefore the data amount is not necessarily '0' if the described settings are applied.

## 2 Restrictions

### 2.1 ESD characteristics for I/O buffers

Table 2-1 shows the ESD characteristics for all I/O buffers for Lavender and Jasmine. For a complete pinning table see hardware manual.

(A) Listed values are determined by using the **Human Body Model**(**C=100pF**, **R=1.5k ohm**). The test procedure is described in MIL-STD-883.

(B) The worst values among the four conditions (VDD positive, VDD negative, VSS positive, and VSS negative) are shown.

(C) The pass/fail criteria of 1uA leakage current was used for all the I/Os except for the I/Os described with „Leak mode“.

(D) In the case of „Leak mode“, the leakage current will increase but the I/Os will continue to be functional.

(E) The I/Os marked with „Destruction mode“, however, may be destroyed if the higher voltage than shown in the table is applied. Once they are destroyed, they may become non-functional.

**Table 2-1: ESD performance for buffer types**

Buffer Type	Description	ESD performance of MB87P2020
B3NNLMX	Bidirectional True buffer (3.3V CMOS, IOL=4mA, Low Noise type)	+/-500V, Destruction mode
B3NNNMX	Bidirectional True buffer (3.3V CMOS, IOL=4mA)	+/-500V, Destruction mode
BFNNQHX	Bidirectional True buffer (5V Tolerant, IOL=8mA, High speed type)	+/-2000V, Leak mode
BFNNQLX	Bidirectional True buffer (5V Tolerant, IOL=2mA, High speed type)	+/-2000V, Leak mode
BFNNQMX	5V tolerant, bidirectional true buffer 3.3V CMOS, IOL/IOH=4mA	+/-2000V, Leak mode
IPBIX	Input True Buffer for DRAM TEST (2.5V CMOS with 25K Pull-up) (SDRAM test only)	+/-2000V
ITAMX	Analog Input buffer	+/-2000V
ITAVDX	Analog Power Supply	N.A.
ITAVSX	Analog GND	N.A.
ITBSTX	Input True Buffer for DRAM TEST (2.5V CMOS with 25K Pull-down) (SDRAM test only)	+/-2000V
ITCHX	Input True buffer (2.5V CMOS)	+/-2000V

**Table 2-1:** ESD performance for buffer types

Buffer Type	Description	ESD performance of MB87P2020
ITFHX	5V tolerant 3.3V CMOS Input	+/-2000V, Leak mode
ITFUHX	5V tolerant 3.3V CMOS Input, 25 k Pull-up	+/-2000V, Leak mode
ITTSTX	Input True buffer for DRAM TEST Control (2.5V CMOS with 25K Pull-down)	+/-2000V
OTAMX	Analog Output	+/-2000V
OTAVX	Analog Output buffer	+/-2000V
OTFTQMX	5 V tolerant 3.3V tri-state output, IOL/IOH=4mA	+/-2000V, Leak mode
VPDX	3.3V CMOS input, disable input for Pull up/down resistors, connect to GND	+/-2000V
YB002AAX	Oscillator Output	+/-500V, Destruction mode
YB3DNLMX	Bidirectional True buffer (3.3V CMOS, 25K Pull-down, IOL=4mA, Low Noise type)	+/- 500V, Destruction mode
YB3NNLMX	Bidirectional True buffer (3.3V CMOS, IOL=4mA, Low Noise type)	+/- 500V, Destruction mode
YI002AEX	Oscillator Pin Input	+/-2000V

Table 2-2 gives an overview and a classification about the ESD characteristics.

**Table 2-2:** Overview for ESD characteristics

Subject	Description
Description	Some I/O buffer show a weak ESD performance (see table 2-1).
Classification	HW limitation
Effects without workaround	Affected buffers may be destroyed or may have a higher leakage current if a voltage higher than specified in table 2-1 is applied to the buffer.
Solution/Workaround	No workaround possible.
Concerned devices	MB87J2120 (Lavender) MB87P2020 (Jasmine) fixed for MB87P2020-A (Jasmine redesign)
Testcase	EMDC: ULB.5 and ULB.13

## 2.2 Command FSM

During emulation for Jasmine development a hang-up of the command FSM has been observed while the commands GetPA and NoOp have been executed within a loop. This hang-up has never been observed for Lavender.

Based on this emulation result the command FSM for Jasmine has been redesigned in order to avoid this problem in the future.

At the present time it can not securely excluded that the timing gap problem is NOT present on Lavender. Therefore further investigation and verification is necessary to check the Lavender behaviour with respect to this problem.

Table 2-3 gives an overview and a classification about the described problem.

**Table 2-3:** Overview for timing gap problem

Subject	Description
Description	A command FSM hang up has been observed for an intermediate version <sup>a</sup> of Jasmine.
Classification	HW limitation
Effects without workaround	The command execution is stopped at next command. Only a hardware reset can terminate this state.
Solution/Workaround	For Jasmine this problem is fixed. For Lavender it is not clear whether this problem occurs. Therefore further investigation is necessary.
Concerned devices	MB87J2120 (Lavender) fixed for MB87P2020 (Jasmine) fixed for MB87P2020-A (Jasmine redesign)
Testcase	EMDC: ULB.1

a. The term 'intermediate' means a RTL version between Lavender and Jasmine.

## 2.3 GPU mastertiming synchronization

For Lavender the internal GPU mastertiming signal (master switch for GPU) is not synchronized to display clock domain (output pixel clock) properly.

If the pixel clock runs asynchronous to core clock (e.g. external pixel clock) and the flag MTIMON\_ON is switched on display output may hang afterwards.

In order to avoid this problem no asynchronous display clock should be used. Asynchronous display clock can either be an external display clock or an internal clock derived from another clock than core clock. With other words core and display clock should have the same clock source. See chapter B-1 (Clock Unit (CU)) description in hardware manual for further details about Lavender's clock concept.

For Jasmine this problem is solved.

Table 2-4 gives an overview and a classification about the described problem.

**Table 2-4:** Overview for GPU mastertiming

Subject	Description
Description	If an asynchronous display clock (compared to core clock) is used, turning on/off the GPU with help of MTIMON_ON can cause the display output to hang.

**Table 2-4:** Overview for GPU mastertiming

Subject	Description
Classification	HW limitation
Effects without workaround	A wrong output picture is visible. Only a hardware reset can solve this problem.
Solution/Workaround	No asynchronous display clock should be used for Lavender. Core and display clock should have the same clock source <sup>a</sup> . For Jasmine this problem is solved.
Concerned devices	MB87J2120 (Lavender) fixed for MB87P2020 (Jasmine) fixed for MB87P2020-A (Jasmine redesign)
Testcase	EMDC: GPU-P10

a. As common input clock the pins OSC\_IN, ULB\_CLK, DIS\_PIXCLK or RCLK/MODE[3] can be used.

## 2.4 Read limitation for 16 Bit data interface to MCU

The User Logic Bus interface of MB87J2120 to MB91xxxx MCUs has read limitations in 16Bit data mode (Pin MODE[2]=0). Read access in 32-Bit data mode (Pin MODE[2]=1) is fully functional as also write accesses in all modes and access types. Writing to registers works with all modes properly.

This read limitation affects only half word (16 Bit) and byte (8 Bit) read access to Lavender internal registers (address range 0x0000-0xFBFF) in register space and direct SDRAM read access. It does not affect a word (32 Bit) read access. In this case the MCU splits the word into two half words (16 Bit) which are submitted sequentially. For this transmission the order has to be MSB first (Big Endian format). Table 2-5 gives an overview on all possible modes and read access types.

**Table 2-5:** Overview for read accesses in different modes

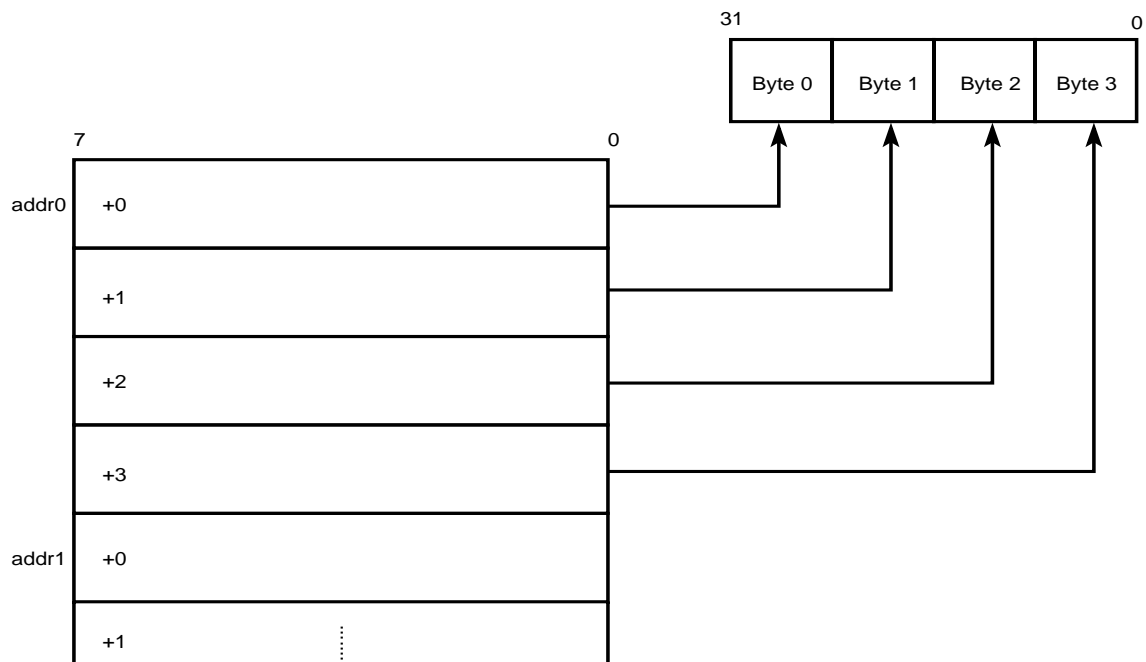
Read access Mode	Word access (32Bit)	Halfword access (16Bit)	Byte access (8Bit)
32Bit mode read (MODE[2]=1)	supported	supported	supported
16Bit mode read (MODE[2]=0)	supported <sup>a</sup>	not supported for offset 2	not supported for offset 2 and 3

a. since MCU delivers data in Big Endian order (MSB first).

An access to address offset 2 or 3 may lead to delivery of wrong values, depending on address of previous read access. Figure 2-1 shows these offsets relative to a given address (addr0, addr1) in byte addressed MCU memory space. The right part of the figure shows the mapping from this memory to internal 32 Bit registers.

There is a possible work-around because reading from an address with offset 0 or 1 to an aligned word address (divisible by 4) is fully operable. After the access to an address with offset 0 or 1, read access from offset 2 or 3 is possible and delivers correct data. If not otherwise guaranteed, a dummy read access on offset 0 or 1 should be included before reading from offset 2 or 3 in order to prepare the correct internal read sequence and deliver correct values also for offsets 2 and 3.<sup>1</sup> Alternative to the described procedure read accesses can be limited to 32-Bit accesses only (see table 2-5).

1. Make sure that this sequence is not interrupted by other read accesses (for instance by an interrupt request with flag read access).



**Figure 2-1:** Address offsets in byte addressed memory and its mapping to Jasmine/Lavender internal registers

In C-API the described workaround is already implemented. Within API read accesses from address offset 2 and 3 are limited to registers `FLNOM_IFF` and `FLNOM_OFE`. These accesses are implemented as 32-Bit read accesses.

Table 2-6 gives an overview and a classification about the described problem.

**Table 2-6:** Overview for read limitation in 16Bit data mode

Subject	Description
Description	Read access from <code>addr0+2</code> or <code>addr0+3</code> does not work reliable.
Classification	HW limitation
Effects without workaround	Halfword and byte read access from <code>addr0+2</code> or <code>addr0+3</code> may deliver wrong values in 16Bit data mode ( <code>MODE[2]=0</code> ).
Solution/Workaround	Read data from <code>addr0+0</code> or <code>addr0+1</code> before reading from <code>addr0+2</code> or <code>addr0+3</code> . OR Limit read accesses to 32Bit read accesses only. All C-API functions which have to read back values implement this workaround already.
Concerned devices	MB87J2120 (Lavender) MB87P2020 (Jasmine) MB87P2020-A (Jasmine redesign)
Testcase	EMDC: CTRL.1 (I/O march) and IPA.1 in 16Bit data mode.

## 2.5 SDC sequencer readback

For Lavender the SDC sequencer is not readable in 16bit data mode (pin MODE[2]=0). Normally the sequencer data will only be written once at initialization time. A readback of sequencer data is usually not necessary.

This problem has been fixed for Jasmine.

Table 2-7 gives an overview and a classification about the described problem.

**Table 2-7:** Overview for SDC sequencer readback

Subject	Description
Description	The SDC sequencer can not be read back in 16bit data mode.
Classification	HW limitation
Effects without workaround	Lavender delivers wrong data when the SDC sequencer is read in 16bit data mode.
Solution/Workaround	None. This problem has been fixed for Jasmine.
Concerned devices	MB87J2120 (Lavender) fixed for MB87P2020 (Jasmine) fixed for MB87P2020-A (Jasmine redesign)
Testcase	EMDC: CTRL.1 (I/O march)

## 2.6 Direct SDRAM access with 16bit and 8bit data mode

Lavender and Jasmine can perform a memory mapped SDRAM access (see hardware manual sections for ULB and DIPA for further details). In the case of 16bit (halfword) or 8bit (byte) read access from SDRAM wrong read data may be delivered for some addresses for Lavender.

Direct SDRAM write access works for all data sizes (32,16 and 8bit).

To avoid this problem 32bit (word) access should always be used for SDRAM read access. There is no restriction for read access from register space<sup>1</sup>.

For Jasmine this problem is solved and every data size can be used for reading from SDRAM.

Table 2-8 gives an overview and a classification about the described problem.

**Table 2-8:** Overview for direct SDRAM access with 16bit and 8bit data mode

Subject	Description
Description	Read access from SDRAM space with 16bit (halfword) and 8bit (byte) data size may deliver wrong values for some addresses.
Classification	HW limitation
Effects without workaround	The MCU reads wrong data from an address mapped to Lavender SDRAM space when reading a 16bit (halfword) or 8bit (byte) value. Reading a 32bit (word) value from SDRAM space works as well as reading from register space.

1. The GDC address space is divided into register and SDRAM space. See chapter B-2.1.4 in hardware manual for further details.



**Table 2-8:** Overview for direct SDRAM access with 16bit and 8bit data mode

Subject	Description
Solution/Workaround	Always read 32bit (word) values from Lavender SDRAM space and mask the data afterwards if necessary. For Jasmine this problem is solved.
Concerned devices	MB87J2120 (Lavender) fixed for MB87P2020 (Jasmine) fixed for MB87P2020-A (Jasmine redesign)
Testcase	EMDC: SDC.3 and DPA.1

## 2.7 Input FIFO read in 16bit mode

One hidden feature of Lavender and Jasmine is the reading from input FIFO<sup>1</sup>. Normally reading from input FIFO makes no sense and is therefore not officially supported but it may be the reason for a MCU blocking which can occur when reading from this address is performed by accident.

In 16bit data mode (Pin MODE[2]=0) reading from input FIFO can cause the ULB\_RDY pin to stay in active state (logic '0'). As already described in chapter 1.6 this can block the command execution of MCU. To avoid this problem reading from input FIFO should not be used in 16bit data mode.

This problem occurs on both GDC devices but for Jasmine the ULB\_RDY timeout can be set so that no system blocking can occur.

Table 2-9 gives an overview and a classification about the described problem.

**Table 2-9:** Overview for input FIFO read access in 16bit data mode

Subject	Description
Description	Reading from input FIFO in 16bit data mode causes ULB_RDY = 0.
Classification	HW limitation
Effects without workaround	The hanging ULB_RDY signal blocks the command execution of a MB91360 series MCU.
Solution/Workaround	Do not read from input FIFO (address 0x0004) in 16bit data mode. For Jasmine the ULB_RDY timeout can be activated.
Concerned devices	MB87J2120 (Lavender) MB87P2020 (Jasmine) MB87P2020-A (Jasmine redesign)
Testcase	EMDC: CTRL.1 (I/O march)

## 2.8 ULB\_DSTP pin function

ULB\_DSTP output of Lavender/Jasmine must not be used because it can not be guaranteed that ULB\_DSTP signal is always generated at DMA interruption by SW-Reset or falling edge of DMAFLAG\_EN<sup>2</sup>. It is rec-

- 
1. The read value is the current FIFO output value which will be delivered to Pixel Processor (PP) with next FIFO read access. Note that the PP reads with core clock data from input FIFO while MCU reading can only be performed with ULB clock. Because ULB clock is always slower (or at most equal) than core clock not all FIFO output data can be sampled.
  2. See ULB specification in hardware manual for further details.

ommended to disable/interrupt MCU-DMAC directly. See table 2-10 for a reset sequence example. The reset of MCU-DMAC is already included in C-API function `ULB_DMA_HDG`.

In case of direct MCU reset `DMAFLAG_DSTP` should be set to '0' in order to avoid unintentional interruption. With this setting no `ULB_DSTP` impulse is generated by the graphic controller.

The `ULB_DSTP` pin must not be used; it will not be supported any longer.

Table 2-10 gives an overview and a classification about the described problem.

**Table 2-10:** Overview for DSTP function

Subject	Description
Description	<code>ULB_DSTP</code> signal is sometimes not generated.
Classification	HW limitation
Effects without workaround	MCU DMA-Controller is not reset correctly. Additional data may be sent after <code>SWReset</code> or DMA turn off.
Solution/Workaround	<p><code>ULB_DSTP</code> signal must not be used. Instead turn off MCU DMA-Controller before sending <code>SWReset</code> to Jasmine/Lavender or disable DMA via <code>DMAFLAG_EN</code>.</p> <p>MCU-DMAC reset sequence:</p> <ol style="list-style-type: none"> <li>1.) <code>DMACA[30] = 0</code></li> <li>2.) <code>DMACA[31] = 0</code></li> </ol> <p>GDC-API function <code>ULB_DMA_HDG</code> takes care of this issue.</p>
Concerned devices	<p>MB87J2120 (Lavender)</p> <p>MB87P2020 (Jasmine)</p> <p>MB87P2020-A (Jasmine redesign)</p>
Testcase	EMDC: <code>ULB.5</code> and <code>ULB.13</code>

## 2.9 Software Reset for command execution

For Jasmine the `SwReset` command does not reset parts of command controller correctly. As a consequence the following commands are not executed properly. They may be finished too early.

Lavender does not have this problem due to a different command controller structure.

As workaround a `NoOp` can be executed after `SwReset`. This is already included into C-API.

Table 2-11 gives an overview and a classification about the described problem.

**Table 2-11:** Overview for Software Reset (incomplete reset)

Subject	Description
Description	Parts of Command Controller will not be reset with <code>SWReset</code> command.
Classification	HW limitation
Effects without workaround	After <code>SWReset</code> next command may not be executed properly.
Solution/Workaround	<p>Send a <code>NoOp</code> command after <code>SWReset</code> command.</p> <p>The <code>NoOp</code> command is already included in API function '<code>GDC_CMD_SwRs</code>'.</p>
Concerned devices	<p>not present for MB87J2120 (Lavender)</p> <p>MB87P2020 (Jasmine)</p> <p>MB87P2020-A (Jasmine redesign)</p>

**Table 2-11:** Overview for Software Reset (incomplete reset)

Subject	Description
Testcase	EMDC: ULB.1, ULB.2 and SDC.6

In case of software reset and anti aliasing (AAF) enabled, data for logical address [Layer 0, Pixel {0,0}] can be destroyed.

The SwReset command is executed by AAF despite a pending SDRAM access. Therefore layer and pixel address is reset which causes a wrong address for SDRAM access.

For this restriction two possible workarounds exist which are described in table 2-12.

Table 2-12 gives an overview and a classification about the described problem.

**Table 2-12:** Overview for Software Reset (AAF)

Subject	Description
Description	If AAF is enabled SwReset command causes an AAF reset without waiting for SDC.
Classification	HW limitation
Effects without workaround	If AAF is enabled data for [Layer 0, Pixel {0,0}] can be destroyed.
Solution/Workaround	<p>Workaround 1:</p> <ul style="list-style-type: none"> <li>Do not display [Layer 0, Pixel {0,0}] (either do not use entire layer or do not use Row 0 and Column 0 for layer 0)</li> </ul> <p>Workaround 2:</p> <ul style="list-style-type: none"> <li>Save physical word at start address of layer 0. This word contains Pixel {0,0}. Physical access does not need command control.</li> <li>Optional: Disable MCU interrupts and wait for GPU frame sync in order to avoid visible effects on display.</li> <li>Send command SwReset (including a NoOp according to table 2-11) to Jasmine/Lavender</li> <li>Restore physical word at start address of layer 0. Pixel {0,0} will be re-stored in this way.</li> </ul>
Concerned devices	MB87J2120 (Lavender) MB87P2020 (Jasmine) MB87P2020-A (Jasmine redesign)
Testcase	Code review

## 2.10 AAF settings double buffering

The command synchronization for AAF settings (register: PPCMD) works not properly. It can happen that the currently executed command uses the settings from previous command.

In order to avoid this problem a NoOp command should be inserted before AAF settings are manipulated. A sequence for a drawing command with AAF (turn on before command execution and turn off afterwards) looks as follows:

```
GDC_CMD_NOP();
G0PPCMD_EN = 1; // AAF enable
G0AAOE     = ..; // AAF init
GDC_CMD_Dw...; // Drawing command
GDC_CMD_NOP();
G0PPCMD_EN = 0; // AAF disable
```

Please note that the C-API functions already include the necessary flag polling of FLNOM\_CWEN. The C-API function 'PXP\_AAF' implements the described workaround already and should be used to turn on or off AAF within an application.

A special situation occurs when a command, which uses the AAF, is interrupted by a software reset (C-API function 'GDC\_CMD\_SwRs'). In this case even a NoOp command can not ensure reinitialization of AAF. It is necessary to execute a dummy drawing command after changing the AAF settings in order to restore double buffered values inside AAF. Note that AAF is enabled during dummy command and the options of the command executed before SWReset are activated. After the dummy command the new settings are applied to AAF.

Table 2-13 gives an overview and a classification about the described problem.

**Table 2-13:** Overview for AAFEN double buffering

Subject	Description
Description	The AAF double buffer for AAFEN signal works not properly.
Classification	HW limitation
Effects without workaround	The AAF settings from previous command are still valid for currently executed command.
Solution/Workaround	Workaround if no SWReset is used: <ul style="list-style-type: none"> <li>• Execute a NoOp command before AAF settings will be changed</li> </ul> If the drawing command with AAF is interrupted by a SWReset a special workaround is needed: <ul style="list-style-type: none"> <li>• Execute a NoOp command after SWReset (see also chapter 2.9)</li> <li>• Execute dummy draw command</li> </ul>
Concerned devices	MB87J2120 (Lavender) MB87P2020 (Jasmine) MB87P2020-A (Jasmine redesign)
Testcase	Code review

## 2.11 Pixel Engine (PE) Commands

There are four PE commands which load rectangular shapes of bitmaps or textures to video memory. PutTxtBM, PutTxtCP, PutBM and PutCP commands have the option to disable the write process of background data to video memory. **For PutTxtBM and PutTxtCP relevant configuration is BGCOL\_EN = 0. For PutBM and PutCP it is IGNORCOL\_EN = 1.** In both cases situation can occur where the data stream to video memory is finished before procession of the command is terminated. This is the case if not transferred background or ignore-coloured pixels were generated at the end.

Due to the finished data transfer the PP output FIFOs are empty already. If the command has finished the ready condition could be propagated to ULB command controller immediately, also if ULB has not stopped the PE device. Resulting from this too early sent "command ready" the ULB command controller fails.

Only in the special situations with suppressed background pixels an additional {GDC\_CMD\_NOP(); GDC\_CMD\_SwRs();} command sequence is required as work around after using the described commands with its appropriate parameters where the error may occur (example for PutTxtBM):

```
GDC_CMD_TxBM (xmin,xmax,ymin,ymax,FgCol,BgCol,BgEn,0,0,Lay);
GDC_FIFO_INP(p_pat, Font->Offset, 0);
if (BgEn == 0) {
    GDC_CMD_NOP(); /* finish PutTxtBM without data loss */
    GDC_CMD_SwRs(); /* PutTxtBM with BGCOL_EN=0 work around */
}
```

}

The inserted NOP command is for de-coupling PutTxtBM from Software Reset. All data were processed finally and closed. To bring the ULB command processor the software reset function is called afterwards. GDC\_CMD\_SwRs() itself executes the SWRES and NOP commands to bring all command processing devices to an initial state (see also section 2.9).

An insertion of GDC\_CMD\_NOP() before each command register write (introduced in C-API former versions) is not required. Only at dedicated points in the software at situations described above the work around is required. Other commands and with background enabled the error can not occur. Application examples encountered only marginal performance loss at transparent text output over background drawings. For bit-map transfer the additional two commands are not important.

Table 2-14 gives an overview and a classification about the described problem.

**Table 2-14:** Pixel engine commands

Subject	Description
Description	Pixel Engine signals end of command execution without stop signal from command controller. This causes possible malfunction of command controller for the next command processed.
Classification	HW limitation
Effects without workaround	Commands PutTxtBM, PutTxtCP, PutBM and PutCP with background disable (BGCOL_EN=0) or ignore colour enable (IGNORCOL_EN=1) can cause command control malfunction. This may lead to an incorrect execution of following commands.
Solution/Workaround	Send NoOp-SwReset command sequence after PutTxt[BM CP] if configured with BGCOL_EN=0 or after Put[BM CP] with IGNORCOL_EN=1. This description is valid for Lavender and Jasmine.
Concerned devices	MB87J2120 (Lavender) MB87P2020 (Jasmine) MB87P2020-A (Jasmine redesign)
Testcase	EMDC: SDC.6

Because of the splitting between command and data write functions within API the NoOp-SwReset command sequence can not be included selectively after affected commands. This can only be done in target application or in a higher level API. The C-API can't solve this problem by inserting a GDC\_CMD\_NOP ( ) and GDC\_CMD\_SwRs() before each command register write activity without performance loss. It is recommended to solve this problem only at dedicated points in the higher level software if special background conditions are relevant only.

## 2.12 Pixel read back commands (GetPixel, XChPixel)

Internal data flow blocks if REQCNT+1 data words does not fit into output FIFO. Therefore the full output FIFO size is not useable in some cases.

If an application wants to collect data in output FIFO in order to transfer a **complete block at once** without checking FIFO load for every data word for instance via DMA it is possible that not all data appear in output FIFO and the initialized limit is not reached. Even if the next command was sent after GetPixel or XChPixel which is normally suitable to flush input FIFO data flow blocking is not escaped.

Note that this behaviour does **not** occur if output FIFO is read with **flag polling for every data word** because the amount of words in output FIFO falls below the limit FIFOSIZE-REQCNT-1 at a certain time. GetPA is not affected because it uses other registers than REQCNT for block size calculation.

Table 2-15 gives an overview and a classification about the described problem.

**Table 2-15:** Pixel read back commands

Subject	Description
Description	Internal data flow blocks if REQCNT+1 data words does not fit into output FIFO. Therefore the full output FIFO size is not useable in some cases.
Classification	HW restriction
Effects without workaround	If an application collects data in output FIFO, for some package sizes last transfer to output FIFO is not started even if input FIFO flush has been forced by a new command.
Solution/Workaround	Ensure that free space in output FIFO is always greater REQCNT. This can be achieved if allowed package size is calculated according to (27). If REQCNT stays constant for an application package size can be calculated offline.
Concerned devices	MB87J2120 (Lavender) MB87P2020 (Jasmine) MB87P2020-A (Jasmine redesign)
Testcase	EMDC: PP.9

A possibility to utilize the full output FIFO size is to ensure that always REQCNT+1 words can be placed in output FIFO. This limits the maximal package size (number of words to transfer for one output FIFO fill) for a given REQCNT. The maximal package size can be calculated according to (1).

$$\text{pkg\_size} \leq \text{trunc}\left(\frac{\text{FIFOSIZE}}{\text{REQCNT} + 1}\right) \times (\text{REQCNT} + 1) \quad (27)$$

The function 'trunc' in (27) means that only the natural part of this fraction should be taken for calculation. The parameter 'FIFOSIZE' is the size of output FIFO. Note that 'pkg\_size' is the maximal package size, sizes smaller than the calculated size can be used.

Figure 2-2 shows an example on how to calculate the correct package size based on a given REQCNT and to read back a block of data. In order to keep the example simple flag polling for FLNOM\_OFH is used but it is also possible to generate an interrupt with this flag or to set up OFDMA\_UL with the package size for DMA transfer.

```
// -----
// Calculate optimal package size for
// a given Request-Count
// -----
reqcnt = G0REQCNT;
// Set FIFO size
if (G0CLKPDR_ID==0) { // Lavender
    of_size = 128;
} else {                // Jasmine
    of_size = 64;
}
pkg_size = ((byte)(of_size/(reqcnt+1)))*(reqcnt+1);
// -----
// Write command to display controller and
// set command registers for GetPixel
// (no registers required)
// -----
GDC_CMD_GtPx();
// -----
// write data to input FIFO
// -----
for (jj=0;jj<pkg_size;jj++) {
```

```

GDC_FIFO_INP((dword*)BuildIfData(x,y,layer),1,0);
}
// -----
// send NoOp command to force FIFO flush
// -----
GDC_CMD_NOP();
// -----
// wait for data in OF
// -----
G0OFUL_UL = pkg_size;
while (G0FLNOM_OFH==0);
// read data from output FIFO
for (jj=0;jj<pkg_size;jj++) {
    data[amount++] = G0OFIFO;
}

```

**Figure 2-2:** Example for package size calculation and read back

## 2.13 Display Interface Re-configuration

If modification of the display interface record (DIR, part of GPU) is used to re-configure the physical size and display timing (PHSIZE, MTIMODD, MTIMEVEN) it can happen that display output can't be activated again. This is important for multisync systems which work with different video resolutions (hot configurable in running application). The error was discovered only if switching from higher to lower display resolution.

Intended configuration method was:

1. MTIMON = 0
2. reconfigure DIR
3. MTIMON = 1

To avoid this error it is recommended to use general hardware master reset and reconfigure the whole GDC device:

1. CLKPDR\_MRST = 1
2. CLKPDR\_MRST = 0
3. reconfigure whole GDC
4. MTIMON = 1

**Table 2-16:** GPU sync timing change

Subject	Description
Description	If GPU display resolution is changed GPU may hang with no output (black display).
Classification	HW restriction
Effects without workaround	GPU may hang at resolution change from higher to lower resolutions.
Solution/Workaround	Only a hardware reset (RESETX pin or CLKPDR_MRST) and re-configuration reactivates GPU. This has to be considered if changing sync timing at runtime is required.

**Table 2-16:** GPU sync timing change

Subject	Description
Concerned devices	MB87J2120 (Lavender) MB87P2020 (Jasmine) partly fixed for MB87P2020-A (Jasmine redesign)
Concerned devices	MB87J2120 (Lavender) MB87P2020 (Jasmine) MB87P2020-A (Jasmine redesign)
Testcase	EMDC: SPB.1



---

## ***D-3 Abbreviations***



# Abbreviations

AAF	Anti Aliasing Filter
APLL	Analog Phase Locked Loop
BPP	Bits Per Pixel (colour depth)
BSF	Bit Stream Formatter
CBP	Control Bus Port
CCFL	Cold Cathode Fluorescent Lamp
CCU	Color Conversion Unit
CLUT	Color Look-up Table
CRT	Cathode Ray Tube
CTRL	ConTRoL unit
CU	Clock Unit
DAC	Digital to Analog Converter
DFU	Data Fetching Unit
DIPA	Direct and Indirect Physical memory Access
DMA	Direct Memory Access
DMAC	DMA Controller
DPA	Direct Physical memory Access
DRM	Duty Ratio Modulator/Modulation
EMC	Electromagnetic Compatibility
FET	Field Effect Transistor
FSM	Finite State Machine
GDC	Graphic Display Controller (Lavender and/or Jasmine in this manual)
GPU	Graphic Processing Unit
HW	HardWare
IPA	Indirect Physical memory Access
ISR	Interrupt Service Routine
LCD	Liquid Crystal Display
LSA	Line Segment Accumulator
MAU	Memory Access Unit
MCP	Memory CoPy Unit
MCU	Micro Controller Unit
PCB	Printed Circuit Board

PE	Pixel Engine
PP	Pixel Processor
RGB	Red / Green / Blue Color Format
RLE	Run Length Encoding
SDC	SDRAM Controller
SPB	Serial Peripheral Bus
SPG	Sync Pulse Generator
SW	SoftWare
TFT	Thin Film Transistor
TGA	TGA is a trademark of Truevision, Inc.
ULB	User Logic Bus (Controller)
VIC	Video Interface Controller
YUV	Luminance / Chrominance Color Format