Features

- Operating voltage:
  f_{SYS} = 6MHz: 2.2V~5.5V
  f_{SYS} = 12MHz: 4.0V~5.5V
- 4K×15 Program Memory
- 160×8 data memory RAM
- USB 2.0 Full Speed Compatible
- Single external interrupt input shared with I/O line
- Single 16-bit programmable Timer/Event Counters with overflow interrupt
- Single SPI interfaces (master and slave mode) shared with PA0~PA3
- Total of 4 Interrupts - $\overline{INT}$, Timer, SPI, USB
- Watchdog Timer function
- Power down and wake-up functions to reduce power consumption
- 16 channel 12-bit resolution A/D converter for HT82A620R
- 3-channel 12-bit PWM output shared with three I/O lines
- 24 Bidirectional I/O lines
- Up to 0.33μs instruction cycle with 12MHz system clock at V_{DD}=5V
- Max. 4 endpoints supported - endpoint 0 included
- All endpoints support Interrupt and bulk transfer
- Endpoint 0 supports control and interrupt
- All endpoints except endpoint 0 can be configured as 8, 16, 32, 64 FIFO size
- Endpoint 0 has 8 byte FIFO
- Total FIFO size: 64+8 bytes (RAM0: 48 bytes; RAM1:16 bytes, 8 bytes for endpoint0)
- 6-level subroutine nesting
- Bit manipulation instruction
- Table read instructions
- 63 powerful instructions
- All instructions executed in one or two instruction cycles
- Low voltage reset function (2.0V±0.1%)
- 20/24/28-pin SSOP, 32-pin QFN package
- PB7 can configure as GPIO or VDDIO by option. The power supply of the pin PB0~PB6 can be optioned as VDD or VDDIO

General Description

The HT82A520R/HT82A620R are 8-bit high performance RISC-like microcontrollers designed for USB keyboard mouse and joystick product applications.

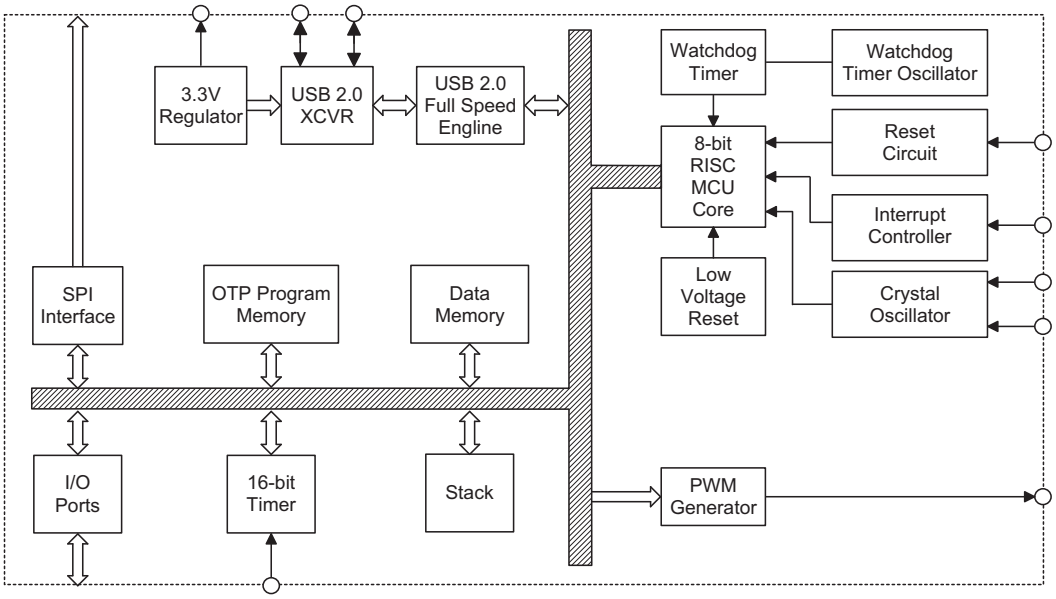
The advantages of low power consumption, I/O flexibility, programmable frequency divider, timer functions, oscillator options, multi-channel A/D Converter, Pulse Width Modulation function, Watchdog timer, SPI interfaces, Power Down and wake-up functions, enhance the versatility of these devices to suit a wide range of application possibilities such as sensor signal processing, motor driving, industrial control, consumer products, subsystem controllers, etc.

Selection Table

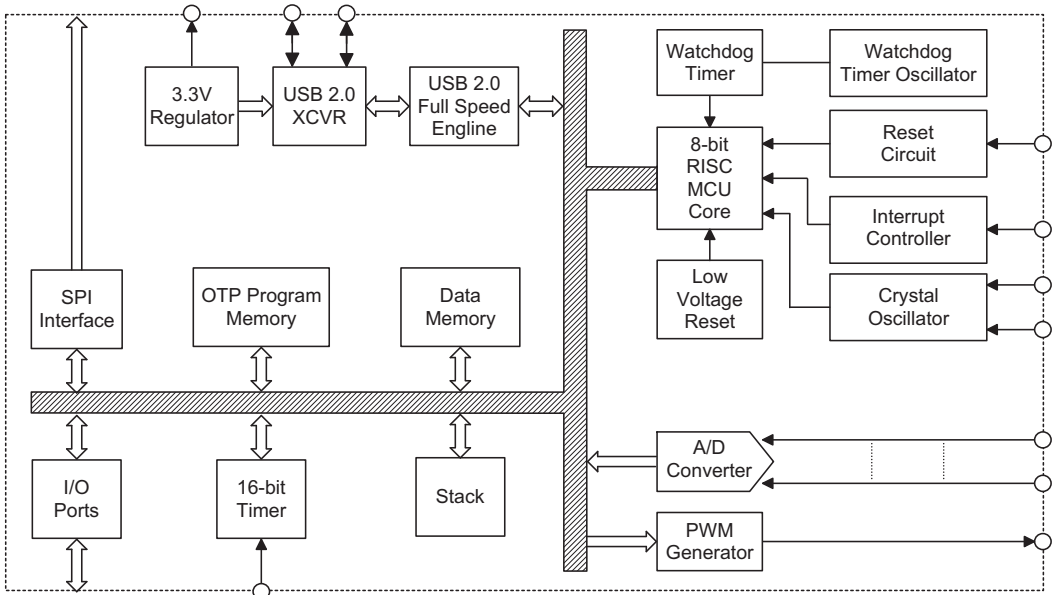
Part No.	Data Memory	Program Memory	I/O	Timer	A/D	PWM	SPI	Package
HT82A520R	160×8	4K×15	24	16-bit×1	—	12-bit×3	1	20/24/28SSOP, 32QFN
HT82A620R	160×8	4K×15	24	16-bit×1	12-bit×16	12-bit×3	1	20/24/28SSOP, 32QFN

Block Diagram

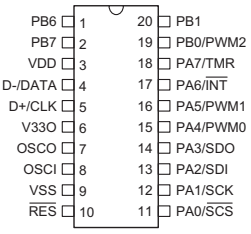
HT82A520R



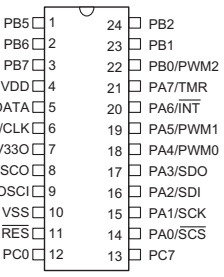
HT82A620R



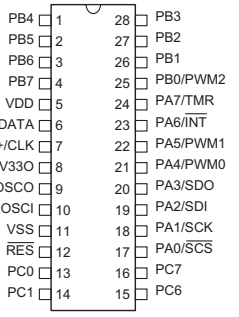
Pin Assignment



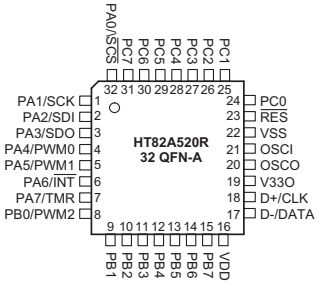
HT82A520R
20 SSOP-A



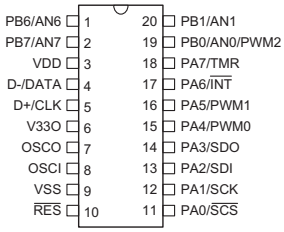
HT82A520R
24 SSOP-A



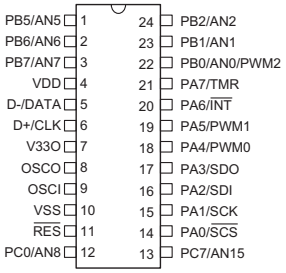
HT82A520R
28 SSOP-A



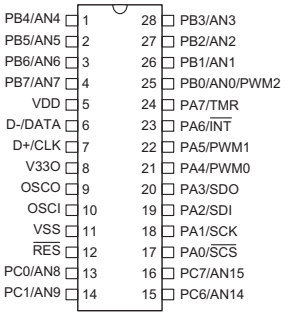
HT82A520R
32 QFN-A



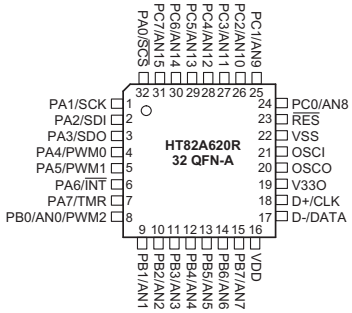
HT82A620R
20 SSOP-A



HT82A620R
24 SSOP-A



HT82A620R
28 SSOP-A



HT82A620R
32 QFN-A

Pin Description

HT82A520R

Pin Name	I/O	Options	Description
PA0/$\overline{SCS}$ PA1/SCK PA2/SDI PA3/SDO PA4/PWM0 PA5/PWM1 PA6/$\overline{INT}$ PA7/TMR	I/O	Pull-high Wake-up NMOS or CMOS	Bidirectional 8-bit input/output port. Each pin can be configured as a wake-up input by a configuration option. Software instructions determine if the pin is output or Schmitt Trigger input. When if output, Configuration option may choose CMOS or NMOS. Configuration options determine if the pins have pull-high resistors. The $\overline{INT}$ and TMR pins are pin-shared with PA6 and PA7, respectively. PA0~PA3 are shared with the SPI function. PA4~PA5 are shared with PWM0~PWM1.
PB0/PWM2 PB1~PB7	I/O	Pull-high Wake-up PB7: I/O or VDDIO PB0~PB6 power: from VDD or VDDIO	Bi-directional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if the pins have pull-high resistors. A configuration option determines whether PB7 is an I/O pin or a power supply pin for PB0~PB6. A configuration option determines whether the power supply for PB0~PB6 is sourced from VDD or VDDIO. PB0 shared with PWM2.
PC0~PC7	I/O	Pull-high Wake-up	Bi-directional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if the pins have pull-high resistors.
OSCI OSCO	I O	—	OSCI, OSCO are connected to an 6MHz or 12MHz Crystal/resonator (determined by software instructions) for the internal system clock
$\overline{RES}$	I	—	Schmitt Trigger reset input, active low
VDD	—	—	Positive power supply
D-/Data	I/O	—	USBD- line The USB function is controlled by software control registers
D+/CLK	I/O	—	USBD+ line The USB function is controlled by software control registers
V33O	O	—	3.3V regulator output
VSS	—	—	Negative power supply, ground

HT82A620R

Pin Name	I/O	Options	Description
PA0/$\overline{SCS}$ PA1/SCK PA2/SDI PA3/SDO PA4/PWM0 PA5/PWM1 PA6/$\overline{INT}$ PA7/TMR	I/O	Pull-high Wake-up NMOS or CMOS	Bi-directional 8-bit input/output port. Each bit can be configured as wake-up input by option (bit option). Software instructions determine if the pin is output or Schmitt Trigger input. When if output, Configuration option may choose CMOS or NMOS. Configuration options determine if the pins have pull-high resistors. The TMR and $\overline{INT}$ are pin-shared with PA6 and PA7, respectively. PA0~PA3 are shared with SPI function. PA4~PA5 shared with PWM0~PWM1.
PB0/AN0/PWM2 PB1/AN1 PB2/AN2 PB3/AN3 PB4/AN4 PB5/AN5 PB6/AN6 PB7/AN7	I/O	Pull-high Wake-up PB7: I/O or VDDIO PB0~PB6 power: from VDD or VDDIO	Bi-directional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if the pins have pull-high resistors. PB is pin shared with the A/D inputs. Once a PB line is selected as an A/D input, using software control, the I/O function and pull-high resistor are disabled automatically. A configuration option determines whether PB7 is an I/O pin or a power supply pin for PB0~PB6. A configuration option determines whether the power supply for PB0~PB6 is sourced from VDD or VDDIO PB0 shared with PWM2.
PC0/AN8 PC1/AN9 PC2/AN10 PC3/AN11 PC4/AN12 PC5/AN13 PC6/AN14 PC7/AN15	I/O	Pull-high Wake-up	Bi-directional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if the pins have pull-high resistors. PC is pin shared with the A/D inputs. Once a PB line is selected as an A/D input, using software control, the I/O function and pull-high resistor are disabled automatically
OSCI OSCO	I O	—	OSCI, OSCO are connected to an external 6MHz or 12MHz Crystal/resonator, determined by software instructions, for the internal system clock
RESB	I	—	Schmitt trigger reset input, active low
VDD	—	—	Positive power supply
D -/Data	I/O	—	USBD- line The USB function is controlled by a software control register
D +/CLK	I/O	—	USBD+ line The USB function is controlled by a software control register
V33O	O	—	3.3V regulator output
VSS	—	—	Negative power supply, ground

Absolute Maximum Ratings

Supply Voltage ..... V_{SS}–0.3V to V_{SS}+6.0V
Input Voltage ..... V_{SS}–0.3V to V_{DD}+0.3V
I_{OL} Total 150mA
Total Power Dissipation 500mW

Storage Temperature ..... –50°C to 125°C
Operating Temperature ..... –0°C to 70°C
I_{OH} Total ..... –100mA

Note: These are stress ratings only. Stresses exceeding the range specified under "Absolute Maximum Ratings" may cause substantial damage to the device. Functional operation of this device at other conditions beyond those listed in the specification is not implied and prolonged exposure to extreme conditions may affect device reliability.

D.C. Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	Operating Voltage	—	f_{SYS}=6MHz	2.2	—	5.5	V
			f_{SYS}=12MHz	4.0	—	5.5	V
I_{DD}	Operating Current	5V	No load, f_{SYS}=6MHz, ADC Off, DAC Off	—	8	—	mA
I_{sus1}	Suspend Current	5V	No load, system HALT, USB transceiver, 3.3V Register on and clr suspend2 (UCC.4 [22H])	—	330	—	μA
I_{sus2}	Suspend Current	5V	No load, system HALT, USB transceiver, 3.3V Register on and set suspend2 (UCC.4 [22H])	—	220	—	μA
I_{STB1}	Standby Current (WDT Disabled)	5V	No load, system HALT, USB disable	—	—	10	μA
V_{IL1}	Input Low Voltage for I/O Ports	5V	—	0	—	$0.3V_{DD}$	V
V_{IH1}	Input High Voltage for I/O Ports	5V	—	$0.7V_{DD}$	—	V_{DD}	V
V_{IL2}	Input Low Voltage ($\overline{RES}$)	5V	—	0	—	$0.4V_{DD}$	V
V_{IH2}	Input High Voltage ($\overline{RES}$)	5V	—	$0.9V_{DD}$	—	V_{DD}	V
V_{LVR0}	Low Voltage Reset	5V	—	1.9	2.0	2.1	V
V_{V33O}	3.3V Regulator Output	5V	I_{V33O}=−5mA	3.0	3.3	3.6	V
V_{OS*}	Offset Error	—	—	−2	—	2	mV
I_{OL}	I/O Port Sink Current	5V	V_{OL}=0.1V_{DD}	10	20	—	mA
I_{OH}	I/O Port Source Current	5V	V_{OH}=0.9V_{DD}	−5	−10	—	mA
R_{PH}	Pull-high Resistance	5V	—	10	30	50	kΩ
R_{PH1}	Pull-high Resistance for DATA	5V	—	—	4.5	—	kΩ
R_{PH2}	Pull-high Resistance for CLK	5V	—	—	4.5	—	kΩ
I_{ADC}*	Additional Power Consumption if A/D Converter is Used	5V	No load	—	1.5	3.0	mA
DNL*	A/D Differential Non-Linearity	—	—	—	—	±2	LSB
INL*	A/D Integral Non-Linearity	—	—		±2.5	±4.0	LSB
RESOLU*	Resolution	—	—			12	Bits

Note: "∗"for HT82A620R

A.C. Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
f_{SYS}	System Clock	—	2.2V~5.5V	—	6000	—	kHz
		—	4.0V~5.5V	—	12000	—	kHz
f_{TIMER}	Timer I/P Frequency (TMR)	—	f_{SYS}=6MHz	0	—	6000	kHz
			f_{SYS}=12MHz	0	—	12000	kHz
t_{WDTOSC}	Watchdog Oscillator Period	5V	—	—	31	—	μs
t_{RES}	External Reset Low Pulse Width	—	—	1	—	—	ms
t_{SST}	System Start-up Timer Period	—	Wake-up from HALT	—	1024	—	t_{SYS}
t_{OPD}	Option Load Timer Period	5V	—	33	70	140	ms
t_{INT}	Interrupt Pulse Width	—	—	1	—	—	μs
t_{AD}*	A/D Clock Period	—	—	1	—	—	μs
t_{ADC}*	A/D Conversion Time	—	—	—	16	—	t_{AD}
t_{ADCS}*	A/D Sample Time	—	—	—	8	—	t_{AD}
t_{CS_SK}	SPI $\overline{SCS}$ to SCK Time	—	—	50	—	—	ns
t_{SPICK}	SPI Clock Time	—	—	400	—	—	ns

Note: $t_{SYS}=1/f_{SYS}$

"*" for HT82A620R

System Architecture

A key factor in the high-performance features of the Holtek range of microcontrollers is attributed to the internal system architecture. The range of devices take advantage of the usual features found within RISC microcontrollers providing increased speed of operation and enhanced performance. The pipelining scheme is implemented in such a way that instruction fetching and instruction execution are overlapped, hence instructions are effectively executed in one cycle, with the exception of branch or call instructions. An 8-bit wide ALU is used in practically all operations of the instruction set. It carries out arithmetic operations, logic operations, rotation, increment, decrement, branch decisions, etc. The internal data path is simplified by moving data through the Accumulator and the ALU. Certain internal registers are implemented in the Data Memory and can be directly or indirectly addressed. The simple addressing methods of these registers along with additional architectural features ensure that a minimum of external components is required to provide a functional I/O and A/D control system with maximum reliability and flexibility.

Clocking and Pipelining

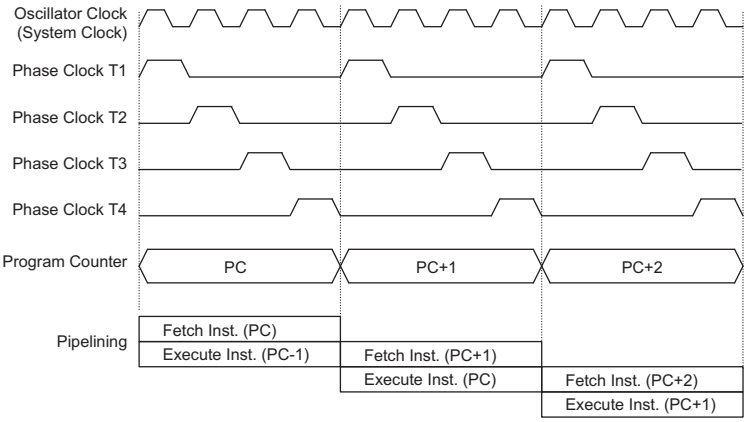
The main system clock, derived from a Crystal/Resonator is subdivided into four internally generated non-overlapping clocks, T1~T4. The Program Counter is incremented at the beginning of the T1 clock during which time a new instruction is fetched. The remaining T2~T4 clocks carry out the decoding and execution functions. In this way, one T1~T4 clock cycle forms one instruction cycle. Although the fetching and execution of instructions takes place in consecutive instruction cycles, the pipelining structure of the microcontroller ensures that instructions are effectively executed in one instruction cycle. The exception to this are instructions where the contents of the Program Counter are changed, such as subroutine calls or jumps, in which case the instruction will take one more instruction cycle to execute.

For instructions involving branches, such as jump or call instructions, two instruction cycles are required to complete instruction execution. An extra cycle is required as the program takes one cycle to first obtain the actual jump or call address and then another cycle to actually execute the branch. The requirement for this extra cycle should be taken into account by programmers in timing sensitive applications.

Program Counter

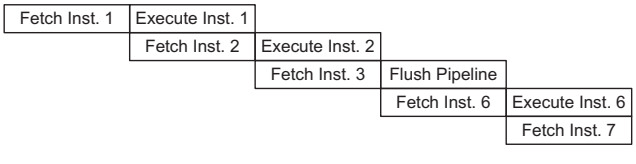
During program execution, the Program Counter is used to keep track of the address of the next instruction to be executed. It is automatically incremented by one each time an instruction is executed except for instructions, such as ″JMP″ or ″CALL″ that demand a jump to a non-consecutive Program Memory address. It must be noted that only the lower 8 bits, known as the Program Counter Low Register, are directly addressable by user.

When executing instructions requiring jumps to non-consecutive addresses such as a jump instruction,



System Clocking and Pipelining

```
1          MOV A,[12H]
2          CALL DELAY
3          CPL [12H]
4          :
5          :
6  DELAY:  NOP
```



Instruction Fetching

a subroutine call, interrupt or reset, etc., the microcontroller manages program control by loading the required address into the Program Counter. For conditional skip instructions, once the condition has been met, the next instruction, which has already been fetched during the present instruction execution, is discarded and a dummy cycle takes its place while the correct instruction is obtained.

The lower byte of the Program Counter, known as the Program Counter Low register or PCL, is available for program control and is a readable and writeable register. By transferring data directly into this register, a short program jump can be executed directly, however, as only this low byte is available for manipulation, the jumps are limited to the present page of memory, that is 256 locations. When such program jumps are executed it should also be noted that a dummy cycle will be inserted.

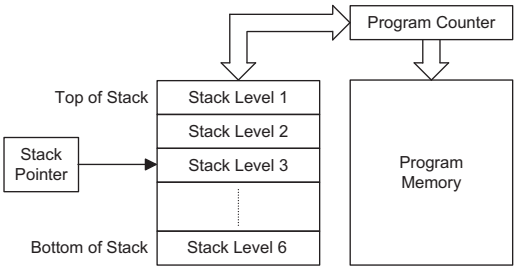
The lower byte of the Program Counter is fully accessible under program control. Manipulating the PCL might cause program branching, so an extra cycle is needed to pre-fetch. Further information on the PCL register can be found in the Special Function Register section.

Stack

This is a special part of the memory which is used to save the contents of the Program Counter only. The stack has 6 levels and is neither part of the data nor part of the program space, and is neither readable nor writeable. The activated level is indexed by the Stack Pointer, SP, and is neither readable nor writeable. At a subroutine call or interrupt acknowledge signal, the contents of the Program Counter are pushed onto the stack. At the end of a subroutine or an interrupt routine, signaled by a return instruction, RET or RETI, the Program Counter is restored to its previous value from the stack.

After a device reset, the Stack Pointer will point to the top of the stack.

If the stack is full and an enabled interrupt takes place, the interrupt request flag will be recorded but the acknowledge signal will be inhibited. When the Stack Pointer is decremented, by RET or RETI, the interrupt will be serviced. This feature prevents stack overflow allowing the programmer to use the structure more easily. However, when the stack is full, a CALL subroutine instruction can still be executed which will result in a stack overflow. Precautions should be taken to avoid such cases which might cause unpredictable program branching.



Arithmetic and Logic Unit – ALU

The arithmetic-logic unit or ALU is a critical area of the microcontroller that carries out arithmetic and logic operations of the instruction set. Connected to the main microcontroller data bus, the ALU receives related instruction codes and performs the required arithmetic or logical operations after which the result will be placed in the specified register. As these ALU calculation or operations may result in carry, borrow or other status changes, the status register will be correspondingly updated to reflect these changes. The ALU supports the following functions.

Mode	Program Counter Bits											
	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Initial Reset	0	0	0	0	0	0	0	0	0	0	0	0
USB Interrupt	0	0	0	0	0	0	0	0	0	1	0	0
External Interrupt	0	0	0	0	0	0	0	0	1	0	0	0
Timer/Event Counter Overflow	0	0	0	0	0	0	0	0	1	1	0	0
SPI Interrupt	0	0	0	0	0	0	0	1	0	0	0	0
Skip	Program Counter + 2											
Loading PCL	PC11	PC10	PC9	PC8	@7	@6	@5	@4	@3	@2	@1	@0
Jump, Call Branch	#11	#10	#9	#8	#7	#6	#5	#4	#3	#2	#1	#0
Return from Subroutine	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0

Program Counter

Note: PC11~PC8: Current Program Counter bits
#11~#0: Instruction code address bits
@7~@0: PCL bits
S11~S0: Stack register bits

- Arithmetic operations: ADD, ADDM, ADC, ADCM, SUB, SUBM, SBC, SBCM, DAA
- Logic operations: AND, OR, XOR, ANDM, ORM, XORM, CPL, CPLA
- Rotation RRA, RR, RRCA, RRC, RLA, RL, RLCA, RLC
- Increment and Decrement INCA, INC, DECA, DEC
- Branch decision, JMP, SZ, SZA, SNZ, SIZ, SDZ, SIZA, SDZA, CALL, RET, RETI

Program Memory

The Program Memory is the location where the user code or program is stored. The HT82A520R/HT82A620R is a One-Time Programmable, OTP, memory type device where users can program their application code into the device. By using the appropriate programming tools, OTP devices offer users the flexibility to freely develop their applications which may be useful during debug or for products requiring frequent upgrades or program changes. OTP devices are also applicable for use in applications that require low or medium volume production runs.

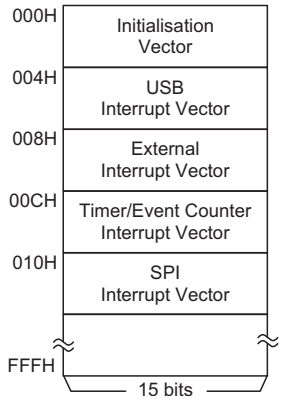
Structure

The Program Memory has a capacity of 4K by 15 bits. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries. Table data, which can be setup in any location within the Program Memory, is addressed by separate table pointer registers.

Special Vectors

Within the Program Memory, certain locations are reserved for special usage such as reset and interrupts.

- Location 000H
  This vector is reserved for use by the device reset for program initialisation. After a device reset is initiated, the program will jump to this location and begin execution.
- Location 004H
  This area is reserved for the USB interrupt service program. If the USB interrupt is activated, the interrupt is enabled and the stack is not full, the program begins execution at location 004H.
- Location 008H
  This vector is used by the external interrupt. If the $\overline{\text{INT}}$ external input pin on the device receives a high to low transition, the program will jump to this location and begin execution, if the interrupt is enabled and the stack is not full.
- Location 00CH
  This vector is used by the timer counter. If a counter overflow occurs, the program will jump to this location and begin execution if the timer interrupt is enabled and the stack is not full.
- Location 010H
  This vector is used by serial interface. When 8-bits of data have been received or transmitted successfully from serial interface, the program will jump to this location and begin execution if the interrupt is enabled and the stack is not full.



Program Memory Structure

Look-up Table

Any location within the Program Memory can be defined as a look-up table where programmers can store fixed data. To use the look-up table, one method is to first setup a low byte table pointer by placing the lower order address of the look up data to be retrieved in the low byte table pointer register, TBLP. This register defines the lower 8-bit address of the look-up table.

After setting up the table pointer, the table data can be retrieved from the current Program Memory page or last Program Memory page using the "TABRDC[m]" or "TABRDL [m]" instructions, respectively. When these instructions are executed, the lower order table byte from the Program Memory will be transferred to the user defined Data Memory register [m] as specified in the instruction. The higher order table data byte from the Program Memory will be transferred to the TBLH special register. Any unused bits in this transferred higher order byte will be read as "0".

The following diagram illustrates the addressing/data flow of the look-up table:

Table Program Example

Another method is to setup the full table address using both the TBLP and TBHP low and high byte table pointer registers to directly address any area in he Program Memory. In this way any page of data can be accessed directly using the TABRDL instruction. If the TBHP high byte table pointer register is to be used, then it must first be enabled with a configuration option.

The following example shows how the table pointer and table data is defined and retrieved from the microcontroller. This example uses raw table data located in the last page which is stored there using the ORG statement. The value at this ORG statement is "F00H" which refers to the start address of the last page within the 4K Program Memory of device. The table pointer is setup here to have an initial value of "06H". This will ensure that the first data read from the data table will be at the Program Memory address "F06H" or 6 locations after the start of the last page. Note that the value for the table pointer is referenced to the first address of the present page if the "TABRDC [m]" instruction is being used. The high byte of the table data which in this case is equal to zero will be transferred to the TBLH register automatically when the "TABRDL [m]" instruction is executed.

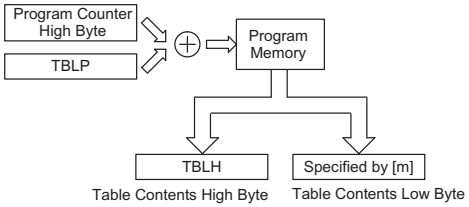


Table Read - TBLP Only

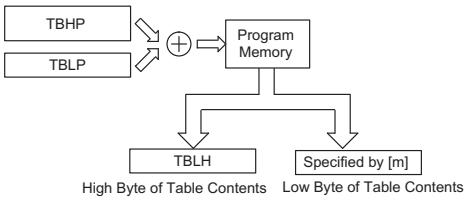


Table Read - TBLP / TBHP

```
tempreg1 db    ?    ; temporary register #1
tempreg2 db    ?    ; temporary register #2
     :
     :

mov a,06h           ; initialise table pointer - note that this address
                    ; is referenced

mov tblp,a          ; to the last page or present page
     :
     :

tabrdl   tempreg1   ; transfers value in table referenced by table pointer
                    ; to tempreg1
                    ; data at prog. memory address "F06H" transferred to
                    ; tempreg1 and TBLH

dec tblp            ; reduce value of table pointer by one

tabrdl   tempreg2   ; transfers value in table referenced by table pointer
                    ; to tempreg2
                    ; data at prog.memory address "F05H" transferred to
                    ; tempreg2 and TBLH
                    ; in this example the data "1AH" is transferred to
                    ; tempreg1 and data "0FH" to register tempreg2
                    ; the value "00H" will be transferred to the high byte
                    ; register TBLH
     :
     :

org F00h            ; sets initial address of last page

dc  00Ah, 00Bh, 00Ch, 00Dh, 00Eh, 00Fh, 01Ah, 01Bh
     :
     :
```

Instruction	Table Location Bits											
	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
TABRDC [m]	PC11	PC10	PC9	PC8	@7	@6	@5	@4	@3	@2	@1	@0
TABRDL [m]	1	1	1	1	@7	@6	@5	@4	@3	@2	@1	@0

Table Location

Note: PC11~PC8: Current Program Counter bits
@7~@0: Table Pointer TBLP bits
TBHP register bit3~bit0 when TBHP is enabled

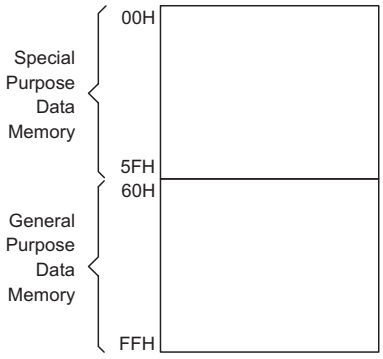
Because the TBLH register is a read-only register and cannot be restored, care should be taken to ensure its protection if both the main routine and Interrupt Service Routine use the table read instructions. If using the table read instructions, the Interrupt Service Routines may change the value of TBLH and subsequently cause errors if used again by the main routine. As a rule it is recommended that simultaneous use of the table read instructions should be avoided. However, in situations where simultaneous use cannot be avoided, the interrupts should be disabled prior to the execution of any main routine table-read instructions. Note that all table related instructions require two instruction cycles to complete their operation.

Data Memory

The Data Memory is a volatile area of 8-bit wide RAM internal memory and is the location where temporary information is stored. Divided into two sections, the first of these is an area of RAM where special function registers are located. These registers have fixed locations and are necessary for correct operation of the device. Many of these registers can be read from and written to directly under program control, however, some remain protected from user manipulation. The second area of Data Memory is reserved for general purpose use. All locations within this area are read and write accessible under program control.

Structure

The two sections of Data Memory, the Special Purpose and General Purpose Data Memory are located at consecutive locations. All are implemented in RAM and are 8 bits wide but the length of each memory section is dictated by the type of microcontroller chosen. The start address of the Data Memory for all devices is the address ″00H″. Registers which are common to all microcontrollers, such as ACC, PCL, etc., have the same Data Memory address.



Data Memory Structure

Note: Most of the Data Memory bits can be directly manipulated using the ″SET [m].i″ and ″CLR [m].i″ with the exception of a few dedicated bits. The Data Memory can also be accessed through the memory pointer register MP.

General Purpose Data Memory

All microcontroller programs require an area of read/write memory where temporary data can be stored and retrieved for use later. It is this area of RAM memory that is known as General Purpose Data Memory. This area of Data Memory is fully accessible by the user program for both read and write operations. By using the ″SET [m].i″ and ″CLR [m].i″ instructions individual bits can be set or reset under program control giving the user a large range of flexibility for bit manipulation in the Data Memory.

00H	IAR0	2EH	
01H	MP0	2FH	
02H	IAR1	30H	UFOEN
03H	MP1	31H	UFC0
04H		32H	
05H	ACC	33H	
06H	PCL	34H	
07H	TBLP	35H	
08H	TBLH	36H	
09H	WDTS	37H	
0AH	STATUS	38H	SBCR
0BH	INTC0	39A	SBDR
0CH		3AH	ADRL*
0DH		3BH	ADRH*
0EH		3CH	ADCR*
0FH	TMRH	3DH	ADSR*
10H	TMRL	3EH	
11H	TMRC	3FH	
12H	PA	40H	
13H	PAC	41H	
14H	PB	42H	
15H	PBC	43H	
16H	PC	44H	
17H	PCC	45H	
18H		46H	
19H		47H	
1AH		48H	
1BH		49H	
1CH	USB_STAT	4AH	
1DH	UINT	4BH	
1EH	INTC1	4CH	
1FH	TBHP	4DH	
20H	USC	4EH	
21H	USR	4FH	
22H	UCC	50H	
23H	AWR	51H	PWMBR0
24H	STALL	52H	PWM0DRL
25H	SIES	53H	PWM0DRH
26H	MISC	54H	PWMBR1
27H	UFIEN	55H	PWM1DRL
28H	FIFO0	56H	PWM1DRH
29H	FIFO1	57H	PWMBR2
2AH	FIFO2	58H	PWM2DRL
2BH	FIFO3	59H	PWM2DRH
2CH		5AH	SPIR
2DH			

: Unused Read as "00"

"*" for HT82A620R

Special Purpose Data Memory

This area of Data Memory is where registers, necessary for the correct operation of the microcontroller, are stored. Most of the registers are both readable and writable but some are protected and are readable only, the details of which are located under the relevant Special Function Register section. Note that for locations that are unused, any read instruction to these addresses will return the value ″00H″.

Special Function Registers

To ensure successful operation of the microcontroller, certain internal registers are implemented in the Data Memory area. These registers ensure correct operation of internal functions such as timers, interrupts, etc., as well as external functions such as I/O data control. The location of these registers within the Data Memory begins at the address 00H. Any unused Data Memory locations between these special function registers and the point where the General Purpose Memory begins is reserved and attempting to read data from these locations will return a value of 00H.

Indirect Addressing Register – IAR0, IAR1

The IAR0 and IAR1 register, although having their locations in normal RAM register space, do not actually physically exist as normal registers. The method of indirect addressing for RAM data manipulation uses these Indirect Addressing Registers and Memory Pointers, in contrast to direct memory addressing, where the actual memory address is specified. Actions on the IAR0 and IAR1 registers will result in no actual read or write operation to these registers but rather to the memory location specified by their corresponding Memory Pointer, MP0 or MP1. Acting as a pair, IAR0 and MP0 can together only access data from Bank 0, while the IAR1 and MP1 register pair can access data from both Bank 0 and Bank 1. As the Indirect Addressing Registers are not physically implemented, reading the Indirect Addressing Registers indirectly will return a result of ″00H″ and writing to the registers indirectly will result in no operation.

Memory Pointer – MP0, MP1

For all devices, two Memory Pointers, known as MP0 and MP1 are provided. These Memory Pointers are physically implemented in the Data Memory and can be manipulated in the same way as normal registers providing a convenient way with which to address and track data. When any operation to the relevant Indirect Addressing Registers is carried out, the actual address that the microcontroller is directed to, is the address specified by the related Memory Pointer.

```
data .section  'data'
adres1    db ?
adres2    db ?
adres3    db ?
adres4    db ?
block     db ?
code .section at 0 'code'
org   00h

start:
    mov a,04h           ; setup size of block
    mov block,a
    mov a,offset adres1; Accumulator loaded with first RAM address
    mov mp0,a           ; setup memory pointer with first RAM address

loop:
    clr IAR0               ; clear the data at address defined by MP0
    inc mp0                ; increment memory pointer
    sdz block           ; check if last memory location has been cleared
    jmp loop

continue:
```

The important point to note here is that in the example shown above, no reference is made to specific Data Memory addresses.

Accumulator – ACC

The Accumulator is central to the operation of any microcontroller and is closely related with operations carried out by the ALU. The Accumulator is the place where all intermediate results from the ALU are stored. Without the Accumulator it would be necessary to write the result of each calculation or logical operation such as addition, subtraction, shift, etc., to the Data Memory resulting in higher programming and timing overheads. Data transfer operations usually involve the temporary storage function of the Accumulator; for example, when transferring data between one user defined register and another, it is necessary to do this by passing the data through the Accumulator as no direct transfer between two registers is permitted.

Program Counter Low Register – PCL

To provide additional program control functions, the low byte of the Program Counter is made accessible to programmers by locating it within the Special Purpose area of the Data Memory. By manipulating this register, direct jumps to other program locations are easily implemented. Loading a value directly into this PCL register will cause a jump to the specified Program Memory location, however, as the register is only 8-bit wide, only jumps within the current Program Memory page are permitted. When such operations are used, note that a dummy cycle will be inserted.

Look-up Table Registers – TBLP, TBHP, TBLH

These three special function registers are used to control operation of the look-up table which is stored in the Program Memory. TBLP and TBHP are the table pointer low and high byte registers and indicate the location where the table data is located. There value must be setup before any table read commands are executed. Their value can be changed, for example using the ″INC″ or ″DEC″ instructions, allowing for easy table data pointing and reading. TBLH is the location where the high order byte of the table data is stored after a table read data instruction has been executed. Note that the lower order table data byte is transferred to a user defined location.

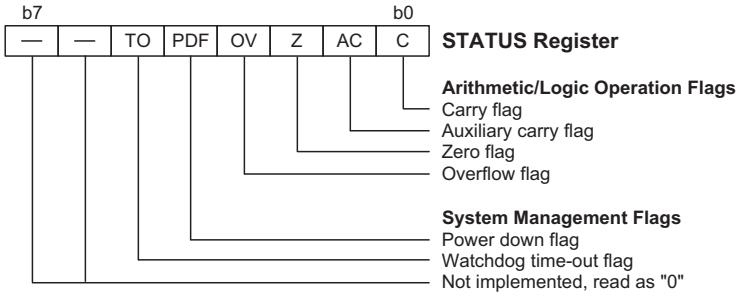
Status Register – STATUS

This 8-bit register contains the zero flag (Z), carry flag (C), auxiliary carry flag (AC), overflow flag (OV), power down flag (PDF), and watchdog time-out flag (TO). These arithmetic/logical operation and system management flags are used to record the status and operation of the microcontroller.

With the exception of the TO and PDF flags, bits in the status register can be altered by instructions like most other registers. Any data written into the status register will not change the TO or PDF flag. In addition, operations related to the status register may give different results due to the different instruction operations. The TO flag can be affected only by a system power-up, a WDT time-out or by executing the ″CLR WDT″ or ″HALT″ instruction. The PDF flag is affected only by executing the ″HALT″ or ″CLR WDT″ instruction or during a system power-up.

The Z, OV, AC and C flags generally reflect the status of the latest operations.

- **C** is set if an operation results in a carry during an addition operation or if a borrow does not take place during a subtraction operation; otherwise C is cleared. C is also affected by a rotate through carry instruction.
- **AC** is set if an operation results in a carry out of the low nibbles in addition, or no borrow from the high nibble into the low nibble in subtraction; otherwise AC is cleared.
- **Z** is set if the result of an arithmetic or logical operation is zero; otherwise Z is cleared.
- **OV** is set if an operation results in a carry into the highest-order bit but not a carry out of the highest-order bit, or vice versa; otherwise OV is cleared.
- **PDF** is cleared by a system power-up or executing the ″CLR WDT″ instruction. PDF is set by executing the ″HALT″ instruction.
- **TO** is cleared by a system power-up or executing the ″CLR WDT″ or ″HALT″ instruction. TO is set by a WDT time-out.



Status Register

In addition, on entering an interrupt sequence or executing a subroutine call, the status register will not be pushed onto the stack automatically. If the contents of the status registers are important and if the interrupt routine can change the status register, precautions must be taken to correctly save it.

Interrupt Control Registers – INTC0, INTC1

The microcontrollers provide one external interrupts, one internal timer/event counter overflow interrupt, one SPI interrupt and one USB interrupt. By setting various bits within these registers using standard bit manipulation instructions, the enable/disable function of each interrupt can be independently controlled. A master interrupt bit within this register, the EMI bit, acts like a global enable/disable and is used to set all of the interrupt enable bits on or off. This bit is cleared when an interrupt routine is entered to disable further interrupt and is set by executing the "RETI" instruction.

Timer/Event Counter Registers – TMRH, TMRL, TMRC

All devices possess one internal 16-bit count-up timer. An associated register pair known as TMRL/TMRH are the locations where the timer 16-bit values are located. These registers can also be preloaded with fixed data to allow different time intervals to be setup. Associated control registers, known as TMRC, contains the setup information for the timers, which determines in what mode the timer is to be used as well as containing the timer on/off control function.

Input/Output Ports and Control Registers

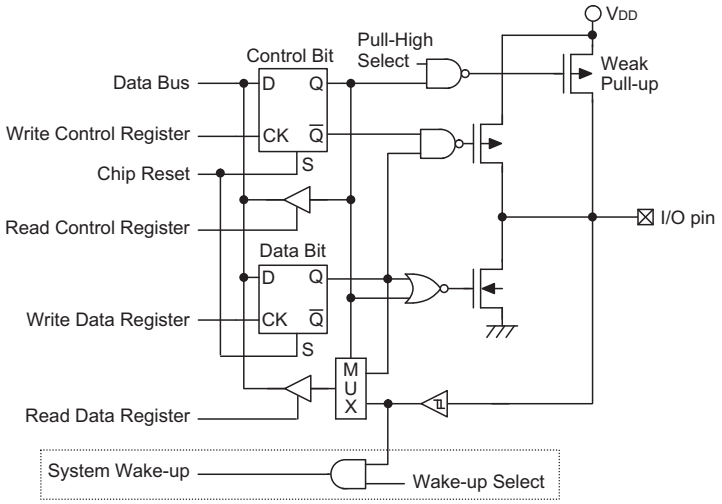
Within the area of Special Function Registers, the I/O registers and their associated control registers play a prominent role. All I/O ports have a designated register correspondingly labeled as PA, PB, PC. These labeled I/O registers are mapped to specific addresses within the Data Memory as shown in the Data Memory table, which are used to transfer the appropriate output or input data on that port. With each I/O port there is an associated control register labeled PAC, PBC, PCC, also mapped to specific addresses with the Data Memory. The control register specifies which pins of that port are set as inputs and which are set as outputs. To setup a pin as an input, the corresponding bit of the control register must be set high, for an output it must be set low. During program initialisation, it is important to first setup the control registers to specify which pins are outputs and which are inputs before reading data from or writing data to the I/O ports. One flexible feature of these registers is the ability to directly program single bits using the "SET [m].i" and "CLR [m].i" instructions. The ability to change I/O pins from output to input and vice versa by manipulating specific bits of the I/O control registers during normal program operation is a useful feature of these devices.

Input/Output Ports

Holtek microcontrollers offer considerable flexibility on their I/O ports. With the input or output designation of every pin fully under user program control, pull-high options for all ports and wake-up options on certain pins, the user is provided with an I/O structure to meet the needs of a wide range of application possibilities.

Depending upon which package is chosen, the microcontroller provides up to 24 bidirectional input/output lines labeled with port names PA, PB, PC.

This register is mapped to the Data Memory with an addresses as shown in the Special Purpose Data Memory table. Seven of these I/O lines can be used for input and output operations and one line as an input only. For input operation, these ports are non-latching, which



Generic Input/Output Ports

means the inputs must be ready at the T2 rising edge of instruction "MOV A,[m]", where m denotes the port address. For output operation, all the data is latched and remains unchanged until the output latch is rewritten.

Pull-high Resistors

Many product applications require pull-high resistors for their switch inputs usually requiring the use of an external resistor. To eliminate the need for these external resistors, I/O pins, when configured as an input have the capability of being connected to an internal pull-high resistor. The pull-high resistors are selectable via configuration options and are implemented using weak PMOS transistors. Each pin on all of I/O can be selected individually to have this pull-high Resistors feature and each nibble on each of the other ports.

Port Pin Wake-up

If the HALT instruction is executed, the device will enter the Power Down Mode, where the system clock will stop resulting in power being conserved, a feature that is important for battery and other low-power applications. Various methods exist to wake-up the microcontroller, one of which is to change the logic condition on one of the port pins from high to low. After a HALT instruction forces the microcontroller into entering the Power Down Mode, the processor will remain in a low-power state until the logic condition of the selected wake-up pin on the port pin changes from high to low. This function is especially suitable for applications that can be woken up via external switches. Each pin on PA has a bit wake-up configuration option and PB, PC have nibble wake-up configuration options.

I/O Port Control Registers

Each I/O port has its own control register PAC, PBC, PCC, to control the input/output configuration. With this control register, each CMOS output or input with or without pull-high resistor structures can be reconfigured dynamically under software control. Each of the I/O ports is directly mapped to a bit in its associated port control register. Note that several pins can be setup to have NMOS outputs using configuration options.

For the I/O pin to function as an input, the corresponding bit of the control register must be written as a "1". This will then allow the logic state of the input pin to be directly read by instructions. When the corresponding bit of the control register is written as a "0", the I/O pin will be setup as an output. If the pin is currently setup as an output, instructions can still be used to read the output register. However, it should be noted that the program will in fact only read the status of the output data latch and not the actual logic status of the output pin.

Port B VDDIO Function

The output drivers of most I/O pins use the VDD power supply line as their high voltage level. In this device pins PB0~PB6 can use a different voltage, other than VDD as their high level. This is supplied externally on pin PB7. This function is selected using a configuration option.

Pin-shared Functions

The flexibility of the microcontroller range is greatly enhanced by the use of pins that have more than one function. Limited numbers of pins can force serious design constraints on designers but by supplying pins with multi-functions, many of these difficulties can be overcome. For some pins, the chosen function of the multi-function I/O pins is set by configuration options while for others the function is set by application program control.

- External interrupt input
  The external interrupt pin $\overline{INT}$ is pin-shared with the I/O pin PA6. For applications not requiring an external interrupt input, the pin-shared external interrupt pin can be used as a normal I/O pin, however to do this, the external interrupt enable bits in the INTC0 register must be disabled.
- External Timer Clock Inputs
  The external timer pin TMR is pin-shared with an I/O pin. To configure this pin to operate as a timer input, the corresponding control bit in the timer control register must be correctly set. For applications that do not require an external timer input, this pin can be used as a normal I/O pin. Note that if used as a normal I/O pin the timer mode control bits in the timer control register must select the timer mode, which has an internal clock source, to prevent the input pin from interfering with the timer operation.
- PWM outputs
  The device contains three PWM outputs which are pin-shared with I/O pins. The PWM output functions are chosen via registers. Note that the corresponding bit of the port control register, PAC and PBC, must setup the pin as an output to enable the PWM output. If the PAC and PBC port control register has setup the pin as an input, then the pin will function as a normal logic input with the usual pull-high option, even if the PWM configuration option has been selected.

- A/D inputs - HT82A620R
  This device has 16 A/D converter inputs depending upon which package type is chosen. All of these analog inputs are pin-shared with I/O pins on Port B and Port C. If these pins are to be used as A/D inputs and not as normal I/O pins then the corresponding bits in the A/D Converter Control Register, ACSR, must be properly set. There are no configuration options associated with the A/D function. If used as I/O pins, then full pull-high resistor configuration options remain, however if used as A/D inputs then any pull-high resistor options associated with these pins will be automatically disconnected.

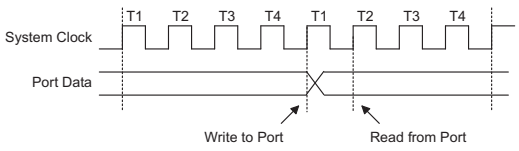
I/O Pin Structures

The vast range of I/O functions and pin-shared options results in a huge variety of I/O pin structure types. For this reason the generic Input/Output Port diagram provided here is for general reference only. As the exact logical construction of the I/O pin will differ from the drawing, they are supplied as a guide only to assist with the functional understanding of the basic I/O pins.

Programming Considerations

Within the user program, one of the first things to consider is port initialisation. After a reset, all of the data and port control register will be set high. This means that all I/O pins will default to an input state, the level of which depends on the other connected circuitry and whether pull-high options have been selected. If the PAC, PBC, PCC port control register, are then programmed to setup some pins as outputs, these output pins will have an initial high output value unless the associated PA, PB, PC port data registers are first programmed. Selecting which pins are inputs and which are outputs can be achieved byte-wide by loading the correct value into the port control register or by programming individual bits in the port control register using the ″SET [m].i″ and ″CLR [m].i″ instructions. Note that when using these bit control instructions, a read-modify-write operation takes place. The microcontroller must first read in the data on the entire port, modify it to the required new bit values and then rewrite this data back to the output ports.

The ports have the additional capability of providing wake-up functions. When the device is in the Power Down Mode, various methods are available to wake the device up. One of these is a high to low transition of any of the port pins. Single or multiple pins on the ports can be setup to have this function.



Read/Write Timing

Timer/Event Counters

The provision of timers form an important part of any microcontroller, giving the designer a means of carrying out time related functions. This device contains a single count-up timer of 16-bit capacity. As each timer has three different operating modes, they can be configured to operate as a general timer, an external event counter or as a pulse width measurement device.

There are two types of registers related to the Timer/Event Counters. The first is the register that contain the actual value of the Timer/Event Counter and into which an initial value can be preloaded, and is known as TMRH, TMRL. Reading from this register retrieves the contents of the Timer/Event Counter. The second type of associated register is the Timer Control Register, which defines the timer options and determines how the Timer/Event Counter is to be used, and has the name TMRC. This device can have the timer clocks configured to come from the internal clock sources. In addition, the timer clock sources can also be configured to come from the external timer pins.

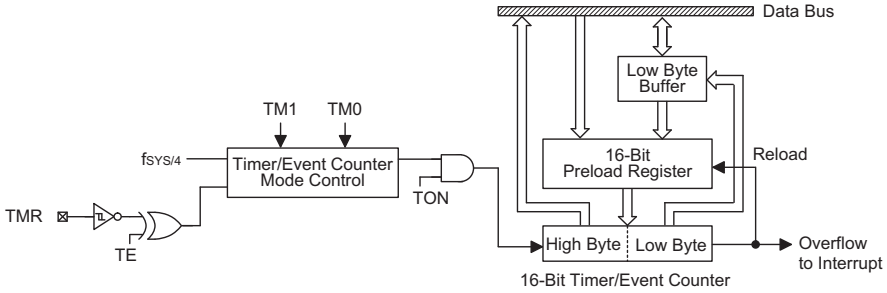
The external clock source is used when the Timer/Event Counter is in the event counting mode, the clock source being provided on the external timer pin. The external timer pin has the name TMR. Depending upon the condition of the TE bit in the Timer Control Register, each high to low, or low to high transition on the external timer input pin will increment the Timer/Event Counter by one.

Configuring the Timer/Event Counter Input Clock Source

The Timer/Event Counter′s clock can originate from various sources. The instruction clock source (system clock source divided by 4) is used when the Timer/Event Counter is in the timer mode or in the pulse width measurement mode. The external clock source is used when the Timer/Event Counter is in the event counting mode, the clock source being provided on the external timer pin, TMR. Depending upon the condition of the TE bit, each high to low, or low to high transition on the external timer pin will increment the counter by one.

Timer Registers – TMRL, TMRH

The timer registers are special function registers located in the Special Purpose RAM Data Memory and are the places where the actual timer values are stored. The timer registers are known as TMRL and TMRH. The value in the timer registers increases by one each time an internal clock pulse is received or an external transition occurs on the external timer pin. The timer will count from the initial value loaded by the preload register to the full count of FFFFH for the 16-bit timer at which point the timer overflows and an internal interrupt signal is generated. The timer value will then be reset with the initial preload register value and continue counting.



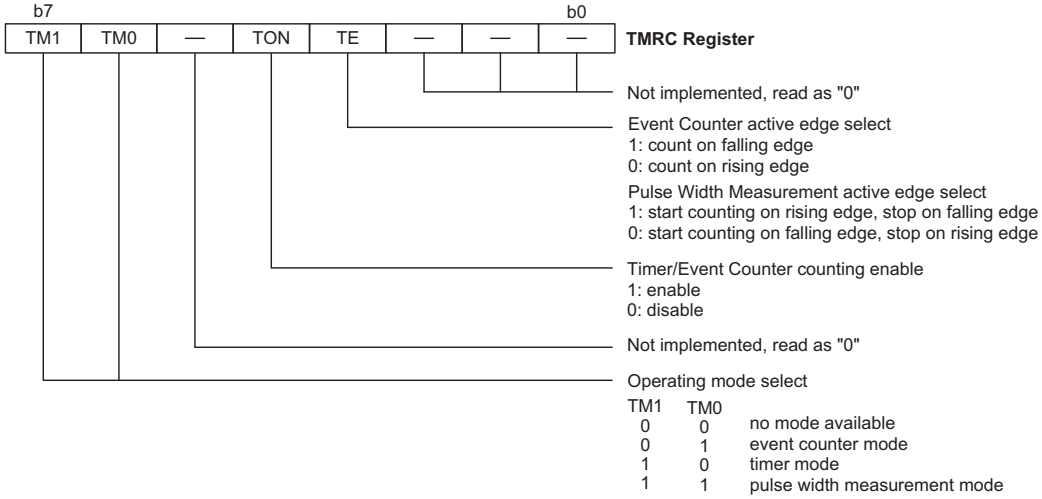
16-bit Timer/Event Counter Structure

To achieve a maximum full range count of FFFFH, the preload registers must first be cleared to all zeros. It should be noted that after power-on, the preload register will be in an unknown condition. Note that if the Timer/Event Counter is switched off and data is written to its preload registers, this data will be immediately written into the actual timer registers. However, if the Timer/Event Counter is enabled and counting, any new data written into the preload data registers during this period will remain in the preload registers and will only be written into the timer registers the next time an overflow occurs.

For the 16-bit Timer/Event Counter which has both low byte and high byte timer registers, accessing these registers is carried out in a specific way. It must be note when using instructions to preload data into the low byte timer register, namely TMRL, the data will only be placed in a low byte buffer and not directly into the low byte timer register. The actual transfer of the data into the low byte timer register is only carried out when a write to its associated high byte timer register, namely TMRH, is executed. On the other hand, using instructions to preload data into the high byte timer register will result in the data being directly written to the high byte timer register. At the same time the data in the low byte buffer will be transferred into its associated low byte timer register. For this reason, the low byte timer register should be written first when preloading data into the 16-bit timer registers. It must also be noted that to read the contents of the low byte timer register, a read to the high byte timer register must be executed first to latch the contents of the low byte timer register into its associated low byte buffer. After this has been done, the low byte timer register can be read in the normal way. Note that reading the low byte timer register will result in reading the previously latched contents of the low byte buffer and not the actual contents of the low byte timer register.

Timer Control Register – TMRC

The flexible features of the Holtek microcontroller Timer/Event Counters enable them to operate in three different modes, the options of which are determined by the contents of their control register, which has the name TMRC. It is the Timer Control Register together with its corresponding timer register that control the full operation of the Timer/Event Counter. Before the Timer/Event Counter can be used, it is essential that the Timer Control Register is fully programmed with the right data to ensure its correct operation, a process that is normally carried out during program initialisation.



Timer/Event Counter Control Register

To choose which of the three modes the Timer/Event Counter is to operate in, either in the timer mode, the event counting mode or the pulse width measurement mode, bits 7 and 6 of the Timer Control Register, which are known as the bit pair TM1/TM0, must be set to the required logic levels. The Timer/Event Counter on/off bit, which is bit 4 of the Timer Control Register and known as TON, provides the basic on/off control of the Timer/Event Counter. Setting the bit high allows the Timer/Event Counter to run, clearing the bit stops it running. If the Timer/Event Counter is in the event count or pulse width measurement mode, the active transition edge level type is selected by the logic level of bit 3 of the Timer Control Register which is known as TE.

Configuring the Timer Mode

In this mode, the Timer/Event Counter can be utilised to measure fixed time intervals, providing an internal interrupt signal each time the Timer/Event Counter overflows. To operate in this mode, the Operating Mode Select bit pair, TM1 or TM0, in the Timer Control Register must be set to the correct value as shown.

Control Register Operating Mode Select Bits for the Timer Mode	Bit7	Bit6
	1	0

In this mode the internal clock, $f_{SYS}/4$ is used as the internal clock for the Timer/Event Counter. After the other bits in the Timer Control Register have been setup, the enable bit TON, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter to run. Each time an internal clock cycle occurs, the Timer/Event Counter increments by one. When it is full and overflows, an interrupt signal is generated and the Timer/Event Counter will reload the value already loaded into the preload register and continue counting. The interrupt can be disabled by ensuring that the Timer/Event Counter Interrupt Enable bit in the Interrupt Control Register, INTC0, is reset to zero.

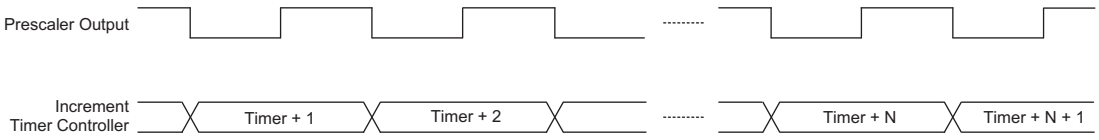
Configuring the Event Counter Mode

In this mode, a number of externally changing logic events, occurring on the external timer pin, can be recorded by the Timer/Event Counter. To operate in this mode, the Operating Mode Select bit pair, TM1/TM0, in the Timer Control Register must be set to the correct value as shown.

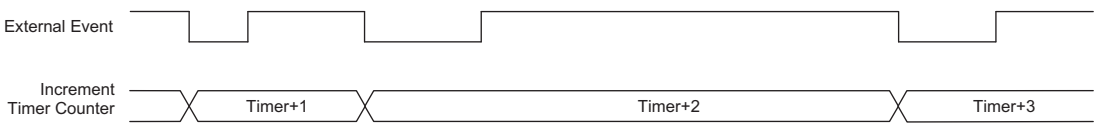
Control Register Operating Mode Select Bits for the Event Counter Mode	Bit7	Bit6
	0	1

In this mode, the external timer pin, TMR, is used as the Timer/Event Counter clock source, however it is not divided by the internal prescaler. After the other bits in the Timer Control Register have been setup, the enable bit TON, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter to run. If the Active Edge Select bit TE, which is bit 3 of the Timer Control Register, is low, the Timer/Event Counter will increment each time the external timer pin receives a low to high transition. If the Active Edge Select bit is high, the counter will increment each time the external timer pin receives a high to low transition. When it is full and overflows, an interrupt signal is generated and the Timer/Event Counter will reload the value already loaded into the preload register and continue counting. The interrupt can be disabled by ensuring that the Timer/Event Counter Interrupt Enable bit in the Interrupt Control Register, INTC0, is reset to zero.

As the external timer pin is an independent pin and not shared with an I/O pin, the only thing to ensure the timer operate as an event counter is to ensure that the Operating Mode Select bits in the Timer Control Register place the Timer/Event Counter in the Event Counting Mode. It should be noted that in the event counting mode, even if the microcontroller is in the Power Down Mode, the Timer/Event Counter will continue to record externally changing logic events on the timer input pin. As a result when the timer overflows it will generate a timer interrupt and corresponding wake-up source.



Timer Mode Timing Chart



Event Counter Mode Timing Chart

Configuring the Pulse Width Measurement Mode

In this mode, the Timer/Event Counter can be utilised to measure the width of external pulses applied to the external timer pin. To operate in this mode, the Operating Mode Select bit pair, TM1 or TM0, in the Timer Control Register must be set to the correct value as shown.

Control Register Operating Mode Select Bits for the Pulse Width Measurement Mode	Bit7	Bit6
	1	1

In this mode the internal clock, $f_{SYS}/4$ is used as the internal clock for the 16-bit Timer/Event Counters. After the other bits in the Timer Control Register have been setup, the enable bit TON, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter, however it will not actually start counting until an active edge is received on the external timer pin.

If the Active Edge Select bit TE , which is bit 3 of the Timer Control Register, is low, once a high to low transition has been received on the external timer pin, TMR, the Timer/Event Counter will start counting until the external timer pin returns to its original high level. At this point the enable bit will be automatically reset to zero and the Timer/Event Counter will stop counting. If the Active Edge Select bit is high, the Timer/Event Counter will begin counting once a low to high transition has been received on the external timer pin and stop counting when the external timer pin returns to its original low level. As before, the enable bit will be automatically reset to zero and the Timer/Event Counter will stop counting. It is important to note that in the Pulse Width Measurement Mode, the enable bit is automatically reset to zero when the external control signal on the external timer pin returns to its original level, whereas in the other two modes the enable bit can only be reset to zero under program control.

The residual value in the Timer/Event Counter, which can now be read by the program, therefore represents the length of the pulse received on the external timer pin. As the enable bit has now been reset, any further transitions on the external timer pin will be ignored. Not until the enable bit is again set high by the program can the timer begin further pulse width measurements. In this way, single shot pulse measurements can be easily made.

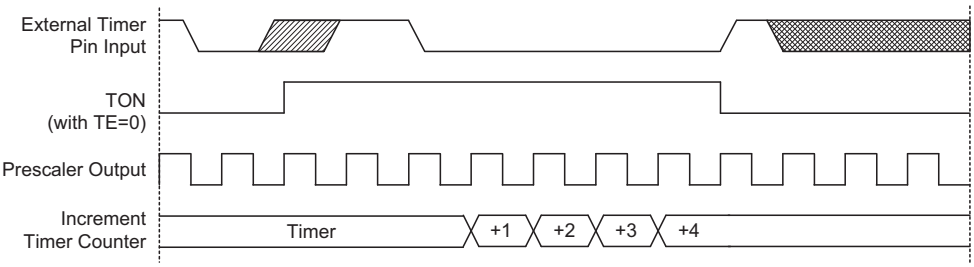
It should be noted that in this mode the Timer/Event Counter is controlled by logical transitions on the external timer pin and not by the logic level. When the Timer/Event Counter is full and overflows, an interrupt signal is generated and the Timer/Event Counter will reload the value already loaded into the preload register and continue counting. The interrupt can be disabled by ensuring that the Timer/Event Counter Interrupt Enable bit in the Interrupt Control Register, INTC0, is reset to zero.

I/O Interfacing

The Timer/Event Counter, when configured to run in the event counter or pulse width measurement mode, requires the use of an external pin for correct operation. This is implemented by ensuring that the mode select bits in the Timer/Event Counter control register, select either the event counter or pulse width measurement mode.

Programming Considerations

When configured to run in the timer mode, the instruction clock (system clock divided by 4) is used as the internal timer clock source for the two 16-bit timers and is therefore synchronized with the overall operation of the microcontroller. In this mode, when the appropriate timer register is full, the microcontroller will generate an internal interrupt signal directing the program flow to the respective internal interrupt vector. For the pulse width measurement mode, the instruction clock is also used as the timer clock source but the timer will only run when the correct logic condition appears on the external timer input pin. As this is an external event and not synchronised with the internal timer clock, the microcontroller will only see this external event when the next timer clock pulse arrives. As a result there may be small differences in measured values requiring programmers to take this into account during programming. The same applies if the timer is configured to be in the event counting mode which again is an external event and not synchronised with the internal system or timer clock.



Prescaler Output is sampled at every falling edge of T1.

Pulse Width Measure Mode Timing Chart

When the Timer/Event Counter is read or if data is written to the preload registers, the clock is inhibited to avoid errors, however as this may result in a counting error, this should be taken into account by the programmer. Care must be taken to ensure that the timers are properly initialised before using them for the first time. The associated timer interrupt enable bits in the interrupt control register must be properly set otherwise the internal interrupt associated with the timer will remain inactive. The edge select, timer mode and clock source control bits in timer control register must also be correctly set to ensure the timer is properly configured for the required application. It is also important to ensure that an initial value is first loaded into the timer register before the timer is switched on; this is because after power-on the initial value of the timer register is unknown. After the timer has been initialised the timer can be turned on and off by controlling the enable bit in the timer control register. Note that setting the timer enable bit high to turn the timer on, should only be executed after the timer mode bits have been properly setup. Setting the timer enable bit high together with a mode bit modification, may lead to improper timer operation if executed as a single timer control register byte write instruction.

When the Timer/Event counter overflows, its corresponding interrupt request flag in the interrupt control register will be set. If the timer interrupt is enabled this will in turn generate an interrupt signal. However irrespective of whether the timer interrupt is enabled or not, a Timer/Event counter overflow will also generate a wake-up signal if the device is in a Power-down condition. This situation may occur if the Timer/Event Counter is in the Event Counting Mode and if the external signal continues to change state. In such a case, the Timer/Event Counter will continue to count these external events and if an overflow occurs the device will be woken up from its Power-down condition. To prevent such a wake-up from occurring, the timer interrupt request flag should first be set high before issuing the HALT instruction to enter the Power Down Mode.

Timer Program Example

This program example shows how the Timer/Event Counter registers are setup, along with how the interrupts are enabled and managed. Note how the Timer/Event Counter is turned on, by setting bit 4 of the Timer Control Register. The Timer/Event Counter can be turned off in a similar way by clearing the same bit. This example program sets the Timer/Event Counter tobe in the timer mode, which uses the internal system clock as the clock source.

```
org 04h             ; USB interrupt vector
reti
org 08h             ; external interrupt vector
reti
org 0ch             ; Timer/Event Counter interrupt vector
jmp tmr0int         ; jump here when Timer/Event Counter overflows
org 010h            ; SPI  interrupt vector
reti
:
org 20h             ; main program
:
;internal Timer/Event Counter interrupt routine
tmrint:
:
; Timer/Event Counter main program placed here
:
reti
:

begin:
;setup Timer/Event Counter  registers
mov a,0e8h          ; setup low byte preload value for Timer/Event Counter
mov tmrl,a          ; low byte must be setup before high byte
mov a,09bh          ; setup high byte preload value for Timer/Event Counter
mov tmrh,a          ;
mov a,080h          ; setup Timer control register TMRC
mov tmrc,a          ; Timer/Event Counter clock source is fSYS/4
                    ; setup interrupt register
mov a,009h          ; enable master interrupt and timer interrupts
mov intc0,a
:
set tmrc.4          ; start Timer/Event Counter _ note mode bits must be previously setup
```

Interrupts

Interrupts are an important part of any microcontroller system. When an external interrupt pin transition or an internal function such as a Timer/Event Counter overflow, an USB interrupt, or transmission or reception of SPI data occurs, their corresponding interrupt will enforce a temporary suspension of the main program allowing the microcontroller to direct attention to their respective needs. Each device contains a single external interrupt and several internal interrupts functions. The external interrupt is controlled by the action of the external interrupt pins, while the internal interrupts are controlled by the Timer/Event Counter overflow, a USB interrupt and SPI data transmission or reception.

Interrupt Registers

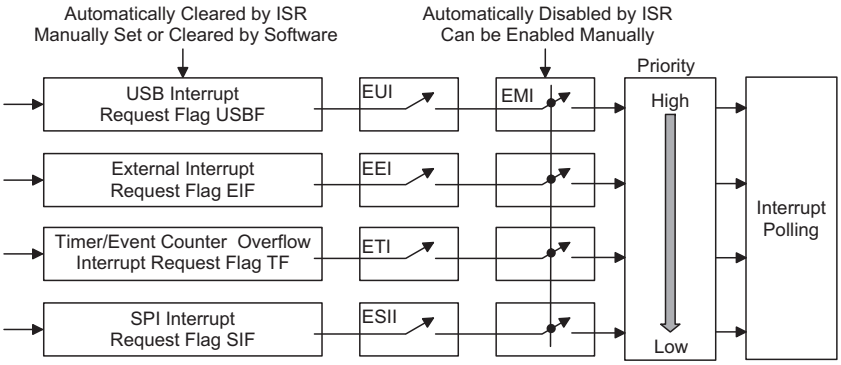
Overall interrupt control, which means interrupt enabling and request flag setting, is controlled by the two interrupt control registers, which are located in the Data Memory. By controlling the appropriate enable bits in these registers each individual interrupt can be enabled or disabled. Also when an interrupt occurs, the corresponding request flag will be set by the microcontroller. The global enable flag if cleared to zero will disable all interrupts.

Interrupt Operation

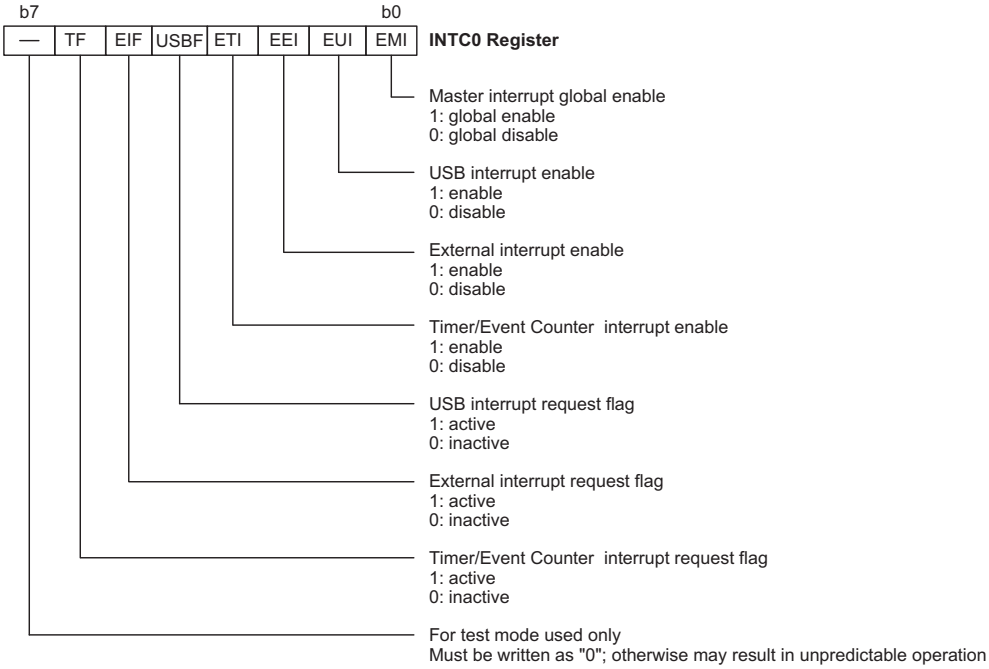
A USB interrupt, Timer/Event Counter overflow, 8-bits of data transmission or reception on either of the one SPI interfaces or an active edge on the external interrupt pin will all generate an interrupt request by setting their corresponding request flag, if their appropriate interrupt enable bit is set. When this happens, the Program Counter, which stores the address of the next instruction to be executed, will be transferred onto the stack. The Program Counter will then be loaded with a new address which will be the value of the corresponding interrupt vector. The microcontroller will then fetch its next instruction from this interrupt vector. The instruction at this vector will usually be a JMP statement which will jump to another section of program which is known as the interrupt service routine. Here is located the code to control the appropriate interrupt. The interrupt service routine must be terminated with a RETI statement, which retrieves the original Program Counter address from the stack and allows the microcontroller to continue with normal execution at the point where the interrupt occurred.

The various interrupt enable bits, together with their associated request flags, are shown in the accompanying diagram with their order of priority.

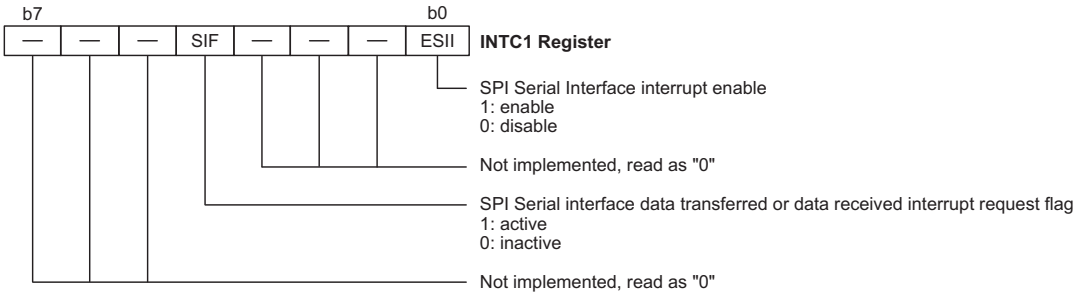
Once an interrupt subroutine is serviced, all the other interrupts will be blocked, as the EMI bit will be cleared automatically. This will prevent any further interrupt nesting from occurring. However, if other interrupt requests occur during this interval, although the interrupt will not be immediately serviced, the request flag will still be recorded. If an interrupt requires immediate servicing while the program is already in another interrupt service routine, the EMI bit should be set after entering the routine, to allow interrupt nesting. If the stack is full, the interrupt request will not be acknowledged, even if the related interrupt is enabled, until the Stack Pointer is decremented. If immediate service is desired, the stack must be prevented from becoming full.



Interrupt Structure



INTC0 Register



INTC1 Register

Interrupt Priority

Interrupts, occurring in the interval between the rising edges of two consecutive T2 pulses, will be serviced on the latter of the two T2 pulses, if the corresponding interrupts are enabled. In case of simultaneous requests, the following table shows the priority that is applied. These can be masked by resetting the EMI bit.

Interrupt Source	Priority	Vector
USB Interrupt	1	0004H
External Interrupt	2	0008H
Timer/Event Counter Overflow Interrupt	3	000CH
SPI Interrupt	4	0010H

Suitable masking of the individual interrupts using the interrupt registers can prevent simultaneous occurrences.

External Interrupt

For an external interrupt to occur, the global interrupt enable bit, EMI, and external interrupt enable bit, EEI, must first be set. An actual external interrupt will take place when the external interrupt request flag, EIF is set, a situation that will occur when a high to low transition appears on the interrupt pins. The external interrupt pin is pin-shared with the I/O pins PA6 can only be configured as an external interrupt pin if the corresponding external interrupt enable bits in the interrupt control register INTC0 have been set. The pins must also be setup as inputs by setting the corresponding PAC.6 bits in the port control register. When the interrupt is enabled, the stack is not full and a high to low transition appears on the external interrupt pin, a subroutine call to the external interrupt vector at location 08H will take place. When the interrupt is serviced, the external interrupt request flag, EIF will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts. Note that any pull-high resistor configuration options on these pins will remain valid even if the pins are used as external interrupt inputs.

Timer/Event Counter Interrupt

For a Timer/Event Counter interrupt to occur, the global interrupt enable bit, EMI, and the corresponding timer interrupt enable bit, ETI, must first be set. An actual Timer/Event Counter interrupt will take place when the Timer/Event Counter interrupt request flag, TF, is set, a situation that will occur when the Timer/Event Counter overflows. When the interrupt is enabled, the stack is not full and a Timer/Event Counter overflow occurs, a subroutine call to the timer interrupt vector at location 0CH , will take place. When the interrupt is serviced, the timer interrupt request flag, TF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

SPI Interrupt

For a SPI Interrupt to occur, the global interrupt enable bit, EMI, and the corresponding SPI interrupt enable bit, ESII must be first set. An actual SPI Interrupt will take place when one of the two SPI interrupt request flags, SIF is set, a situation that will occur when 8-bits of data are transferred or received from either of the SPI interfaces. When the interrupt is enabled, the stack is not full and an SPI interrupt occurs, a subroutine call to the SPI interrupt vector at location 10H, will take place. When the interrupt is serviced, the SPI interrupt request flag, SIF will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

USB Interrupt

A USB interrupts will be triggered by the following USB events, at which point the the related interrupt request flag, USBF in the INTC0 register, will be set.

- Accessing the corresponding USB FIFO from the PC
- A USB suspend signal from the PC
- A USB resume signal from the PC
- A USB Reset signal

When the interrupt is enabled, the stack is not full and the USB interrupt is active, a subroutine call to location 04H will occur. The interrupt request flag, USBF, and the EMI bit will be cleared to disable other interrupts.

When PC Host accesses the FIFO of the device, the corresponding request USR bit is set, and a USB interrupt is triggered. Therefore it can be determined which FIFO has been accessed. When the interrupt has been served, the corresponding bit should be cleared by the program. When the device receive a USB Suspend signal from the Host PC, the suspend line, bit0 of USC, is set and a USB interrupt is also triggered. Also when device receive a Resume signal from the Host PC, the resume line, bit3 of USC, is set and a USB interrupt is triggered.

Programming Considerations

By disabling the interrupt enable bits, a requested interrupt can be prevented from being serviced, however, once an interrupt request flag is set, it will remain in this condition in the interrupt control register until the corresponding interrupt is serviced or until the request flag is cleared by a software instruction.

It is recommended that programs do not use the ″CALL subroutine″ instruction within the interrupt subroutine. Interrupts often occur in an unpredictable manner or need to be serviced immediately in some applications. If only one stack is left and the interrupt is not well controlled, the original control sequence will be damaged once a ″CALL subroutine″ is executed in the interrupt subroutine.

All of these interrupts have the capability of waking up the processor when in the Power Down Mode.

Only the Program Counter is pushed onto the stack. If the contents of the accumulator or status register are altered by the interrupt service program, which may corrupt the desired control sequence, then the contents should be saved in advance.

Reset and Initialisation

A reset function is a fundamental part of any microcontroller ensuring that the device can be set to some predetermined condition irrespective of outside parameters. The most important reset condition is after power is first applied to the microcontroller. In this case, internal circuitry will ensure that the microcontroller, after a short delay, will be in a well defined state and ready to execute the first program instruction. After this power-on reset, certain important internal registers will be set to defined states before the program commences. One of these registers is the Program Counter, which will be reset to zero forcing the microcontroller to begin program execution from the lowest Program Memory address.

In addition to the power-on reset, situations may arise where it is necessary to forcefully apply a reset condition when the microcontroller is running. One example of this is where after power has been applied and the microcontroller is already running, the RES line is forcefully pulled low. In such a case, known as a normal operation reset, some of the microcontroller registers remain unchanged allowing the microcontroller to proceed with normal operation after the reset line is allowed to return high. Another type of reset is when the Watchdog Timer overflows and resets the microcontroller. All types of reset operations result in different register conditions being setup.

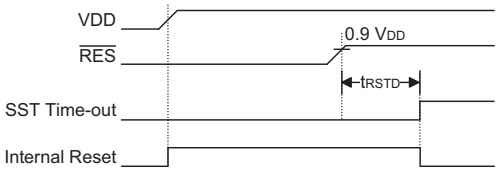
Another reset exists in the form of a Low Voltage Reset, LVR, where a full reset, similar to the RES reset is implemented in situations where the power supply voltage falls below a certain threshold.

Reset Functions

There are five ways in which a microcontroller reset can occur, through events occurring both internally and externally:

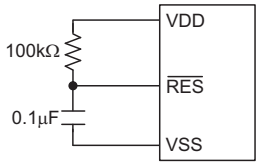
- Power-on Reset
  The most fundamental and unavoidable reset is the one that occurs after power is first applied to the microcontroller. As well as ensuring that the Program Memory begins execution from the first memory address, a power-on reset also ensures that certain other registers are preset to known conditions. All the I/O port and port control registers will power up in a high condition ensuring that all pins will be first set to inputs.

Although the microcontroller has an internal RC reset function, if the VDD power supply rise time is not fast enough or does not stabilise quickly at power-on, the internal reset function may be incapable of providing a proper reset operation. In such cases it is recommended that an external RC network is connected to the $\overline{RES}$ pin, whose additional time delay will ensure that the $\overline{RES}$ pin remains low for an extended period to allow the power supply to stabilise. During this time delay, normal operation of the microcontroller will be inhibited. After the $\overline{RES}$ line reaches a certain voltage value, the reset delay time t_{RSTD} is invoked to provide an extra delay time after which the microcontroller will begin normal operation. The abbreviation SST in the figures stands for System Start-up Timer.



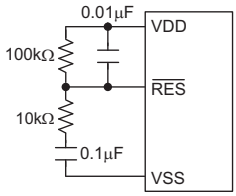
Power-On Reset Timing Chart

For most applications a resistor connected between VDD and the $\overline{RES}$ pin and a capacitor connected between VSS and the $\overline{RES}$ pin will provide a suitable external reset circuit. Any wiring connected to the $\overline{RES}$ pin should be kept as short as possible to minimise any stray noise interference.



Basic Reset Circuit

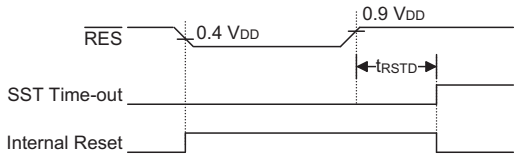
For applications that operate within an environment where more noise is present the Enhanced Reset Circuit shown is recommended.



Enhanced Reset Circuit

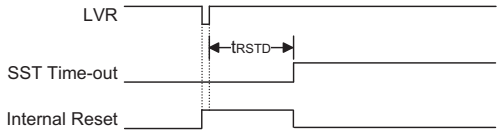
More information regarding external reset circuits is located in Application Note HA0075E on the Holtek website.

- $\overline{RES}$ Pin Reset
  This type of reset occurs when the microcontroller is already running and the $\overline{RES}$ pin is forcefully pulled low by external hardware such as an external switch. In this case as in the case of other reset, the Program Counter will reset to zero and program execution initiated from this point. Note that as the external reset pin is also pin-shared with PA7, if it is to be used as a reset pin, the correct reset configuration option must be selected.



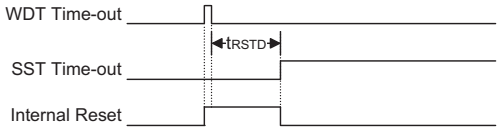
$\overline{RES}$ Reset Timing Chart

- Low Voltage Reset – LVR
  The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device. The LVR function is selected via a configuration option. If the supply voltage of the device drops to within a range of 0.9V~V_{LVR} such as might occur when changing the battery, the LVR will automatically reset the device internally. For a valid LVR signal, a low supply voltage, i.e., a voltage in the range between 0.9V~V_{LVR} must exist for a time greater than that specified by t_{LVR} in the A.C. characteristics. If the low supply voltage state does not exceed this value, the LVR will ignore the low supply voltage and will not perform a reset function. The actual V_{LVR} value can be selected via configuration options.



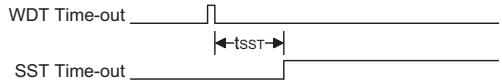
Low Voltage Reset Timing Chart

- Watchdog Time-out Reset during Normal Operation
  The Watchdog time-out Reset during normal operation is the same as a hardware $\overline{RES}$ pin reset except that the Watchdog time-out flag TO will be set to ″1″.



WDT Time-out Reset during Normal Operation Timing Chart

- Watchdog Time-out Reset during Power Down
  The Watchdog time-out Reset during Power Down is a little different from other kinds of reset. Most of the conditions remain unchanged except that the Program Counter and the Stack Pointer will be cleared to ″0″ and the TO flag will be set to ″1″. Refer to the A.C. Characteristics for t_{SST} details.



WDT Time-out Reset during Power Down Timing Chart

Reset Initial Conditions

The different types of reset described affect the reset flags in different ways. These flags, known as PDF and TO are located in the status register and are controlled by various microcontroller operations, such as the Power Down function or Watchdog Timer. The reset flags are shown in the table:

TO	PDF	RESET Conditions
0	0	$\overline{RES}$ reset during power-on
0	0	$\overline{RES}$ wake-up during Power Down
0	0	$\overline{RES}$ or LVR reset during normal operation
1	u	WDT time-out reset during normal operation
1	1	WDT time-out reset during Power Down

Note: ″u″ stands for unchanged

The following table indicates the way in which the various components of the microcontroller are affected after a power-on reset occurs.

Item	Condition After RESET
Program Counter	Reset to zero
Interrupts	All interrupts will be disabled
WDT	Clear after reset, WDT begins counting
Timer/Event Counter	Timer Counter will be turned off
Prescaler	The Timer Counter Prescaler will be cleared
Input/Output Ports	I/O ports will be setup as inputs
Stack Pointer	Stack Pointer will point to the top of the stack

The different kinds of resets all affect the internal registers of the microcontroller in different ways. To ensure reliable continuation of normal program execution after a reset occurs, it is important to know what condition the microcontroller is in after a particular reset occurs. The following table describes how each type of reset affects the microcontroller internal registers.

Register	Reset (Power-on)	WDT Time-out (Normal Operation)	$\overline{RES}$ Reset (Normal Operation)	$\overline{RES}$ Reset (HALT)	WDT Time-out (HALT)*	USB Reset (Normal)	USB Reset (HALT)
MP0	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu	xxxx xxxx	xxxx xxxx
MP1	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu	xxxx xxxx	xxxx xxxx
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCL	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
WDTS	---- -111	---- -111	---- -111	---- -111	---- -111	---- -111	---- -uuu
STATUS	--00 xxxx	--1u uuuu	--00 uuuu	--00 uuuu	--11 uuuu	--uu uuuu	--01 uuuu
INTC0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
TMRH	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu	xxxx xxxx	xxxx xxxx
TMRL	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu	xxxx xxxx	xxxx xxxx
TMRC	00-0 1---	00-0 1---	00-0 1---	00-0 1---	uu-u u---	uu-u u---	uu-u u---
PA	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
PAC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
PB	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
PBC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
PC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
PCC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
PWM0DRL	0000 --00	0000 --00	0000 --00	0000 --00	uuuu --uu	0000 --00	0000 --00
PWM0DRH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
PWMBR0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
PWM1DRL	0000 --00	0000 --00	0000 --00	0000 --00	uuuu --uu	0000 --00	0000 --00
PWM1DRH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
PWMBR1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
PWM2DRL	0000 --00	0000 --00	0000 --00	0000 --00	uuuu --uu	0000 --00	0000 --00
PWM2DRH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
PWMBR2	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
ADRL(**)	xxxx ----	xxxx ----	xxxx ----	xxxx ----	xxxx ----	xxxx ----	uuuu ----
ADRH(**)	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
ADCR(**)	0100 0000	0100 0000	0100 0000	0100 0000	0100 0000	0100 0000	0100 0000
ACSR(**)	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
USB_STAT	--xx 0000	--xx 0000	--xx 0000	--xx 0000	--xx 0000	--xx 0000	--xx 0000
UINT	0000 1111	0000 uuuu	0000 1111	0000 1111	0000 uuuu	0000 1111	0000 1111
TBHP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
USC	1000 0000	uuuu xuux	1000 x00x	1000 x00x	uuuu xuux	1000 0000	1000 0000
USR	0000 0000	uuuu uuuu	0000 0000	0000 0000	uuuu uuuu	00uu 0000	00uu 0000

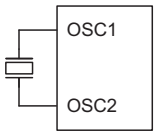
Register	Reset (Power-on)	WDT Time-out (Normal Operation)	$\overline{RES}$ Reset (Normal Operation)	$\overline{RES}$ Reset (HALT)	WDT Time-out (HALT)*	USB Reset (Normal)	USB Reset (HALT)
UCC	0000 0000	uuuu uuuu	0000 0000	0000 0000	uuuu uuuu	0uu0 u000	0uu0 u000
AWR	0000 0000	uuuu uuuu	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
STALL	0000 0000	uuuu uuuu	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
SIES	0000 0000	uuxx xxuu	0000 0000	0000 0000	uuxx xxuu	0u00 u000	0u00 u000
MISC	0000 0000	xxuu uuuu	0000 0000	0000 0000	xxuu uuuu	0000 0000	0000 0000
UFIEN	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
FIFO0	xxxx xxxx	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000
FIFO1	xxxx xxxx	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000
FIFO2	xxxx xxxx	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000
FIFO3	xxxx xxxx	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000
UFOEN	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
UFC0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
UFC1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	uuuu uuuu	uuuu uuuu
USVC	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu	0000 0000	0000 0000
SBCR	0110 0000	0110 0000	0110 0000	0110 0000	0110 0000	uuuu uuuu	uuuu uuuu
SBDR	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
SPIR	---- 0000	---- 0000	---- 0000	---- 0000	---- 0000	---- 0000	---- 0000

Note: "*" means "warm reset", "-" not implemented
"u" means "unchanged", "x" means "unknown"
"**" for HT82A620R

Oscillator

A crystal across OSC1 and OSC2 is needed to provide the feedback and phase shift required for the oscillator. No other external components are required.

The HT82A520R/HT82A620R can operate with 6MHz or 12MHz system clocks. In order to make sure that the USBSIE functions properly, the user should correctly configure the SYSCLK bit of the UCC Register. The default system clock is 12MHz



Crystal Oscillator

Power Down Mode and Wake-up

Power Down Mode

All of the Holtek microcontrollers have the ability to enter a Power Down Mode. When the device enters this mode, the normal operating current, will be reduced to an extremely low standby current level. This occurs because when the device enters the Power Down Mode, the system oscillator is stopped which reduces the power consumption to extremely low levels, however, as the device maintains its present internal condition, it can be woken up at a later stage and continue running, without requiring a full reset. This feature is extremely important in application areas where the microcontroller must have its power supply constantly maintained to keep the device in a known condition but where the power supply capacity is limited such as in battery applications.

Entering the Power Down Mode

There is only one way for the device to enter the Power Down Mode and that is to execute the "HALT" instruction in the application program. When this instruction is executed, the following will occur:

- The system oscillator will stop running and the application program will stop at the "HALT" instruction.
- The Data Memory contents and registers will maintain their present condition.
- The WDT will be cleared and resume counting if the WDT clock source is selected to come from the WDT oscillator. The WDT will stop if its clock source originates from the system clock.
- The I/O ports will maintain their present condition.
- In the status register, the Power Down flag, PDF, will be set and the Watchdog time-out flag, TO, will be cleared.

Standby Current Considerations

As the main reason for entering the Power Down Mode is to keep the current consumption of the microcontroller to as low a value as possible, perhaps only in the order of several micro-amps, there are other considerations which must also be taken into account by the circuit designer if the power consumption is to be minimised.

Special attention must be made to the I/O pins on the device. All high-impedance input pins must be connected to either a fixed high or low level as any floating input pins could create internal oscillations and result in increased current consumption. Care must also be taken with the loads, which are connected to I/O pins, which are setup as outputs. These should be placed in a condition in which minimum current is drawn or connected only to external circuits that do not draw current, such as other CMOS inputs.

If the configuration options have enabled the Watchdog Timer internal oscillator then this will continue to run when in the Power Down Mode and will thus consume some power. For power sensitive applications it may be therefore preferable to use the system clock source for the Watchdog Timer. If any I/O pins are configured as A/D analog inputs using the channel configuration bits in the ADCR register, then the A/D converter will be turned on and a certain amount of power will be consumed. It may be therefore desirable before entering te Power Down Mode to ensure that the A/D converter is powered down by ensuring that any A/D input pins are setup as normal logic inputs with pull-high resistors.Note: A/D function for HT82A620R.

Wake-up

After the system enters the Power Down Mode, it can be woken up from one of various sources listed as follows:

- An external reset
- An external falling edge on any of the I/O pins
- A system interrupt
- A WDT overflow

If the system is woken up by an external reset, the device will experience a full system reset, however, if the device is woken up by a WDT overflow, a Watchdog Timer reset will be initiated. Although both of these wake-up methods will initiate a reset operation, the actual source of the wake-up can be determined by examining the TO and PDF flags. The PDF flag is cleared by a system power-up or executing the clear Watchdog Timer instructions and is set when executing the "HALT" instruction. The TO flag is set if a WDT time-out occurs, and causes a wake-up that only resets the Program Counter and Stack Pointer, the other flags remain in their original status.

Each pins on Port A or any nibble on the other ports can be setup via an individual configuration option to permit a negative transition on the pin to wake-up the system. When a Port pins wake-up occurs, the program will resume execution at the instruction following the ″HALT″ instruction.

If the system is woken up by an interrupt, then two possible situations may occur. The first is where the related interrupt is disabled or the interrupt is enabled but the stack is full, in which case the program will resume execution at the instruction following the ″HALT″ instruction. In this situation, the interrupt which woke-up the device will not be immediately serviced, but will rather be serviced later when the related interrupt is finally enabled or when a stack level becomes free. The other situation is where the related interrupt is enabled and the stack is not full, in which case the regular interrupt response takes place. If an interrupt request flag is set to ″1″ before entering the Power Down Mode, the wake-up function of the related interrupt will be disabled.

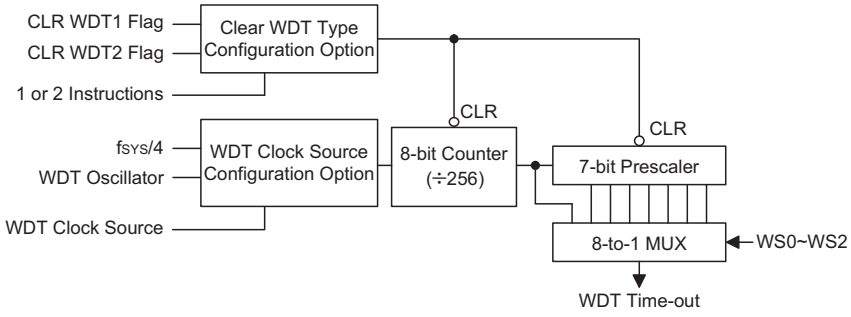
No matter what the source of the wake-up event is, once a wake-up situation occurs, a time period equal to 1024 system clock periods will be required before normal system operation resumes. However, if the wake-up has originated due to an interrupt, the actual interrupt subroutine execution will be delayed by an additional one or more cycles. If the wake-up results in the execution of the next instruction following the ″HALT″ instruction, this will be executed immediately after the 1024 system clock period delay has ended.

Watchdog Timer

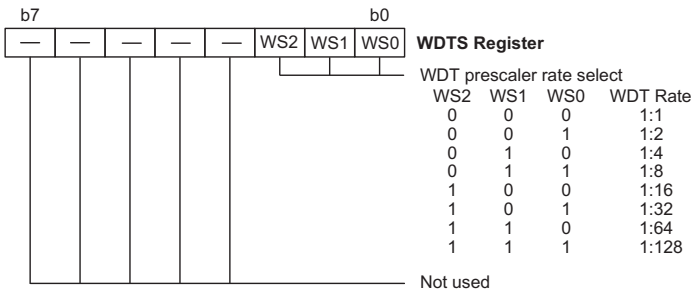
The Watchdog Timer is provided to prevent program malfunctions or sequences from jumping to unknown locations, due to certain uncontrollable external events such as electrical noise. It operates by providing a device reset when the WDT counter overflows. The WDT clock is supplied by one of two sources selected by configuration option: its own self-contained dedicated internal WDT oscillator, or the instruction clock which is the system clock divided by 4. Note that if the WDT configuration option has been disabled, then any instruction relating to its operation will result in no operation.

The internal WDT oscillator has an approximate period of 31μs at a supply voltage of 5V. If selected, it is first divided by 256 via an 8-stage counter to give a nominal period of 8ms. Note that this period can vary with VDD, temperature and process variations. For longer WDT time-out periods the WDT prescaler can be utilized. By writing the required value to bits 0, 1 and 2 of the WDTS register, known as WS0, WS1 and WS2, longer time-out periods can be achieved. With WS0, WS1 and WS2 all equal to 1, the division ratio is 1:128 which gives a maximum time-out period of about 1s.

A configuration option can select the instruction clock, which is the system clock divided by 4, as the WDT clock source instead of the internal WDT oscillator. If the instruction clock is used as the clock source, it must be noted that when the system enters the Power Down Mode, as the system clock is stopped, then the WDT clock source will also be stopped. Therefore the WDT will lose its protecting



Watchdog Timer



Watchdog Timer Register

purposes. In such cases the system cannot be restarted by the WDT and can only be restarted using external signals. For systems that operate in noisy environments, using the internal WDT oscillator is therefore the recommended choice.

Under normal program operation, a WDT time-out will initialise a device reset and set the status bit TO. However, if the system is in the Power Down Mode, when a WDT time-out occurs, only the Program Counter and Stack Pointer will be reset. Three methods can be adopted to clear the contents of the WDT and the WDT prescaler. The first is an external hardware reset, which means a low level on the $\overline{RES}$ pin, the second is using the watchdog software instructions and the third is via a ″HALT″ instruction.

There are two methods of using software instructions to clear the Watchdog Timer, one of which must be chosen by configuration option. The first option is to use the single ″CLR WDT″ instruction while the second is to use the two commands ″CLR WDT1″ and ″CLR WDT2″. For the first option, a simple execution of ″CLR WDT″ will clear the WDT while for the second option, both ″CLR WDT1″ and ″CLR WDT2″ must both be executed to successfully clear the WDT. Note that for this second option, if ″CLR WDT1″ is used to clear the WDT, successive executions of this instruction will have no effect, only the execution of a ″CLR WDT2″ instruction will clear the WDT. Similarly, after the ″CLR WDT2″ instruction has been executed, only a successive ″CLR WDT1″ instruction can clear the Watchdog Timer.

Pulse Width Modulator

The device contains three Pulse Width Modulation PWM outputs. Useful for such applications such as motor speed control, the PWM function provides an output with a variable frequency, and with a duty cycle that can be varied by setting particular values into the corresponding register pair.

Channel	PWM Mode	Output Pin	Register Names
1	8+4	PA4	PWM0DRL~ PWM0DRH
2	8+4	PA5	PWM1DRL~ PWM1DRH
3	8+4	PB0	PWM2DRL~ PWM2DRH

PWM Registers

Three register, located in the Data Memory are assigned to each Pulse Width Modulator output and are known as the PWM registers. It is in each register pair that the 12-bit value, which represents the overall duty cycle of one modulation cycle of the output waveform, should be placed. The PWM registers also contain the enable/disable control bit for the PWM outputs. To increase the PWM modulation frequency, each modulation cycle is modulated into sixteen individual modulation sub-sections, known as the 8+4 mode. Note that it is only necessary to write the required modulation value into the corresponding PWM register as the subdivision of the waveform into its sub-modulation cycles is implemented automatically within the microcontroller hardware.

This method of dividing the original modulation cycle into a further 16 sub-cycles enables the generation of higher PWM frequencies, which allow a wider range of applications to be served. As long as the periods of the generated PWM pulses are less than the time constants of the load, the PWM output will be suitable as such long time constant loads will average out the pulses of the PWM output. The difference between what is known as the PWM cycle frequency and the PWM modulation frequency should be understood.

As the PWM clock is defined by register PWMBR and the system clock f_{SYS} or $f_{SYS}/4$ (chosen via the PWM_S bit), and as the PWM value is 12-bits wide, the overall PWM cycle frequency (f_{PWM}) as the following equation and the corresponding PWM modulation frequency for 8+4 mode is f_{PWM} /256 .

$$f_{PWM}= \frac{1}{(1/f_{SYS})x(PWMBRn+1)} \text{ , when PWM\_S=0}$$

$$f_{PWM}= \frac{1}{(4/f_{SYS})x(PWMBRn+1)} \text{ , when PWM\_S=1}$$

where PWMBRn=0~255 and f_{SYS} may be 6MHz, 12MHz , n=0~2 , according register whether it is PWM0, PWM1 or PWM2

PWM Modulation Frequency	PWM Cycle Frequency	PWM Cycle Duty
$f_{PWM}/256$	$f_{PWM}/4096$	(PWM register value)/4096

8+4 PWM Mode Modulation

Each full PWM cycle, as it is 12-bits wide, has 4096 clock periods. However, in the 8+4 PWM mode, each PWM cycle is subdivided into sixteen individual sub-cycles known as modulation cycle 0 ~ modulation cycle 15, denoted as "i" in the table. Each one of these sixteen sub-cycles contains 256 clock cycles. In this mode, a modulation frequency increase of sixteen is achieved. The 12-bit PWM register value, which represents the overall duty cycle of the PWM waveform, is divided into two groups. The first group which consists of bit4~bit11 is denoted here as the DC value. The second group which consists of bit0~bit3 is known as the AC value. In the 8+4 PWM mode, the duty cycle value of each of the two modulation sub-cycles is shown in the following table.

Parameter	AC (0~15)	DC (Duty Cycle)
Modulation cycle i (i=0~15)	i<AC	$\frac{DC+1}{256}$
	i≥AC	$\frac{DC}{256}$

8+4 Mode Modulation Cycle Values

The accompanying diagram illustrates the waveforms associated with the 8+4 mode of PWM operation. It is important to note how the single PWM cycle is subdivided into 16 individual modulation cycles, numbered 0~15 and how the AC value is related to the PWM value.

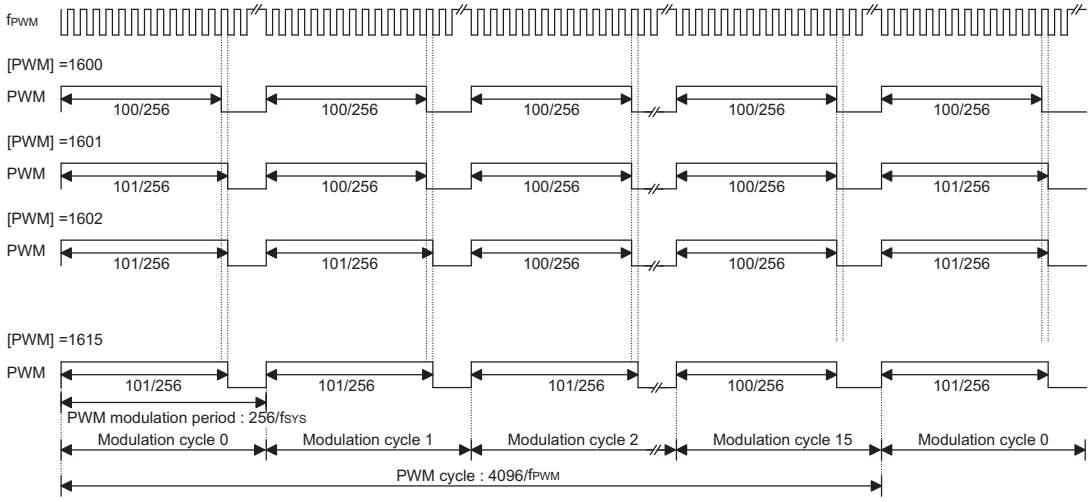
PWM Output Control

The three outputs, PWM0, PWM1, and PWM2 are shared with pins PA4, PA5 and PB0. To operate as a PWM output and not as an I/O pin, bit 0 of the relevant PWM low byte register bit must be set high. A zero must also be written to the corresponding bit in the PAC and PBC port control register, to ensure that the PWM0 output pin is setup as an output. After these two initial steps have been carried out, and of course after the required PWM 12-bit value has been written into the PWM register pair register, setting the corresponding bit in the PA and PB data register high will enable the PWM data to appear on the pin. Writing a zero to the bit will disable the PWM output function and force the output low. In this way, the Port A and Port B data output register bits, can also be used as an on/off control for the PWM function. Note that if the enable bit in the PWM register is set high to enable the PWM function, but if the corresponding bit in the PAC and PBC control register is high to configure the pin as an input, then the pin can still function as a normal input line, with pull-high resistor selections.

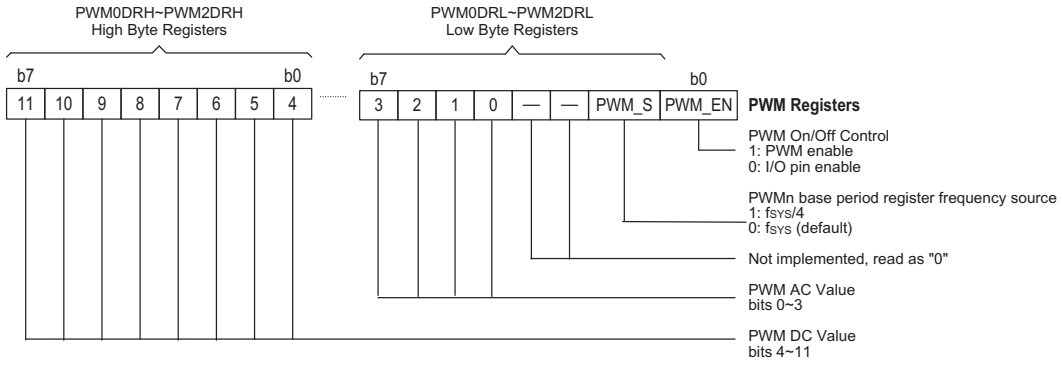
PWM Programming Example

The following sample program shows how the output is setup and controlled.

```
mov a,64h       ; setup PWM0 value to 1600 decimal which is 640H
mov pwm0h,a     ; setup PWM0H register value
clr pwm0l       ; setup PWM0L register value
clr pac.4       ; setup pin PA4 as an output
set pwm0en      ; set the PWM0 enable bit
set pa.4        ; Enable the PWM0 output
  :   :
  :   :
clr pa.4        ; PWM0 output disabled – PA4 will remain low
```



8+4 PWM Mode



PWM Register Pairs

Bit	R/W	Description
7~0	R/W	Used to define the base period of the PWM Range =1 ~ 256 × (1/f_{SYS} or 4/f_{SYS} chosen via the PWM_S bit) where PWMBRn=0~255 (n= 0~2)

PWM Base Period Register PWMBR0 ~ PWMBR2

Analog to Digital Converter – HT82A620R

The need to interface to real world analog signals is a common requirement for many electronic systems. However, to properly process these signals by a microcontroller, they must first be converted into digital signals by A/D converters. By integrating the A/D conversion electronic circuitry into the microcontroller, the need for external components is reduced significantly with the corresponding follow-on benefits of lower costs and reduced component space requirements.

A/D Overview

The device contains a multi-channel channel analog to digital converter which can directly interface to external analog signals, such as that from sensors or other control signals and convert these signals directly into an 12-bit digital value. The number of available channels depends upon which package type is chosen.

The A/D block diagram shows the overall internal structure of the A/D converter, together with its associated registers.

A/D Converter Data Registers – ADRL, ADRH

The device, which contain a single12-bit A/D converter, requires two data registers, known as ADRL and ADRH. After the conversion process takes place, these registers can be directly read by the microcontroller to obtain the digitised conversion value.

In the following tables, D0~D7 are the A/D conversion data result bits.

Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADRL	D3	D2	D1	D0	—	—	—	—
ADRH	D11	D10	D9	D8	D7	D6	D5	D4

Note: D11~D0 is the A/D conversion result data bit MSB~LSB.

A/D Data Register

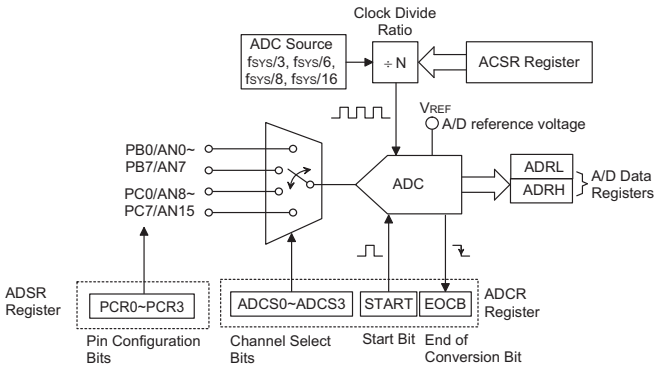
A/D Converter Control Register – ADCR

To control the function and operation of the A/D converter, control registers known as ADCR and ADSR are provided. These 8-bit registers define functions such as the selection of which analog channel is connected to the internal A/D converter, which pins are used as analog inputs and which are used as normal I/Os as well as controlling the start function and monitoring the A/D converter end of conversion status.

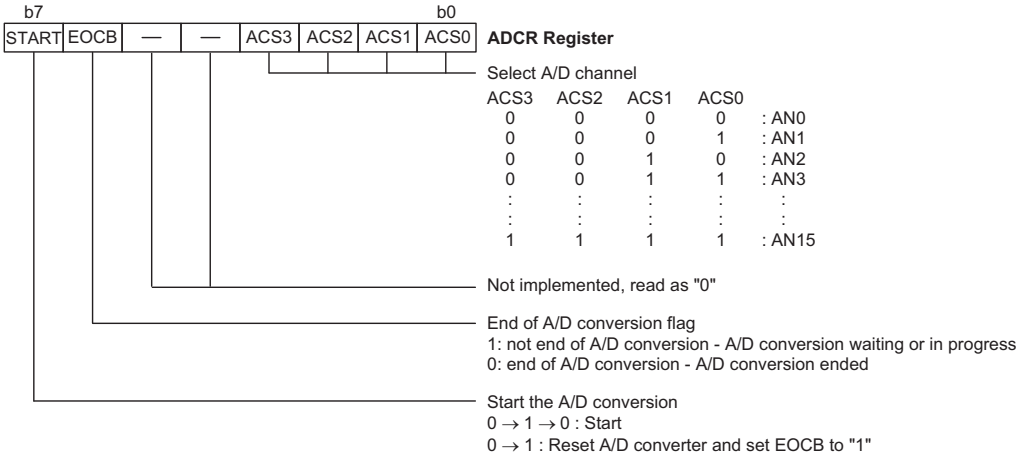
One section of this register contains the bits ACS3~ACS0 which define the channel number. As each of the devices contains only one actual analog to digital converter circuit, each of the individual analog inputs must be routed to the converter. It is the function of the ACS3~ACS0 bits in the ADCR register to determine which analog channel is actually connected to the internal A/D converter.

The START bit in the ADCR register is used to start and reset the A/D converter. When the microcontroller sets this bit from low to high and then low again, an analog to digital conversion cycle will be initiated. When the START bit is brought from low to high but not low again, the EOCB bit in the ADCR register will be set to a "1" and the analog to digital converter will be reset. It is the START bit that is used to control the overall on/off operation of the internal analog to digital converter.

The EOCB bit in the ADCR register is used to indicate when the analog to digital conversion process is complete. This bit will be automatically set to "0" by the microcontroller after a conversion cycle has ended. In addition, the corresponding A/D interrupt request flag will be set in the interrupt control register, and if the interrupts are enabled, an appropriate internal interrupt signal will be generated. This A/D internal interrupt signal will direct the program flow to the associated A/D internal interrupt address for processing. If the A/D internal interrupt is disabled, the microcontroller can be used to



A/D Converter Structure



A/D Converter Control Register

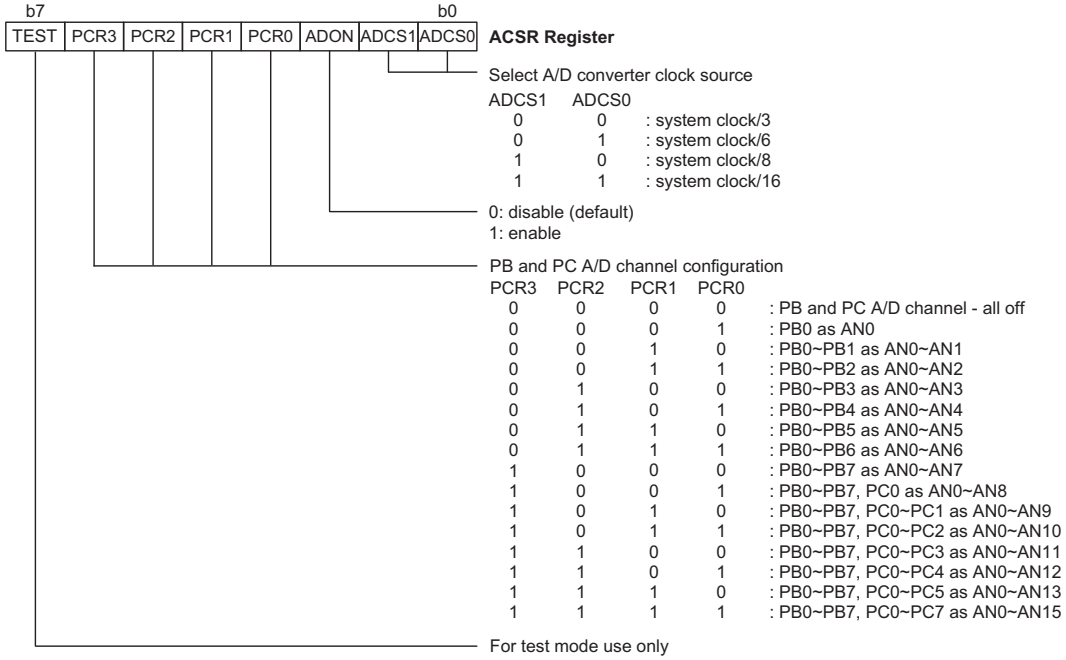
poll the EOCB bit in the ADCR register to check whether it has been cleared as an alternative method of detecting the end of an A/D conversion cycle.

A/D Converter Clock Source Register – ACSR

The clock source for the A/D converter, which originates from the system clock f_{SYS}, is first divided by a division ratio, the value of which is determined by the ADCS1 and ADCS0 bits in the ACSR register.

The ACSR control register also contains the PCR3~PCR0 bits which determine which pins on Port B and Port C are used as analog inputs for the A/D converter and which pins are to be used as normal I/O pins. If the 4-bit address on PCR3~PCR0 has a value of ″1111″ or higher, then all 16 pins, namely AN0~ AN15 will all be set as analog inputs. Note that if the PCR3~PCR0 bits are all set to zero, then all the Port B and Port C pins will be setup as normal I/Os and the internal A/D converter circuitry will be powered off to reduce the power consumption.

Although the A/D clock source is determined by the system clock f_{SYS}, and by bits ADCS1 and ADCS0, there are some limitations on the maximum A/D clock source speed that can be selected. As the minimum value of permissible A/D clock period, t_{AD}, is 0.5μs, care must be taken for system clock speeds in excess of 4MHz. For system clock speeds in excess of 4MHz, the ADCS1 and ADCS0 bits should not be set to ″00″. Doing so will give A/D clock peri-



A/D Converter Clock Source Register

ods that are less than the minimum A/D clock period which may result in inaccurate A/D conversion values. Refer to the following table for examples, where values marked with an asterisk * show where, depending upon the device, special care must be taken, as the values may be less than the specified minimum A/D Clock Period

A/D Input Pins

All of the A/D analog input pins are pin-shared with the I/O pins on Port B and Port C. Bits PCR3~PCR0 in the ACSR register, determine whether the input pins are setup as normal Port B and Port C input/output pins or whether they are setup as analog inputs. In this way, pins can be changed under program control to change their function from normal I/O operation to analog inputs and vice versa. Pull-high resistors, which are setup through configuration options, apply to the input pins only when they are used as normal I/O pins, if setup as A/D inputs the pull-high resistors will be automatically disconnected. Note that it is not necessary to first setup the A/D pin as an input in the PBC or PCC port control register to enable the A/D input as when the PCR3~PCR0 bits enable an A/D input, the status of the port control register will be overridden.

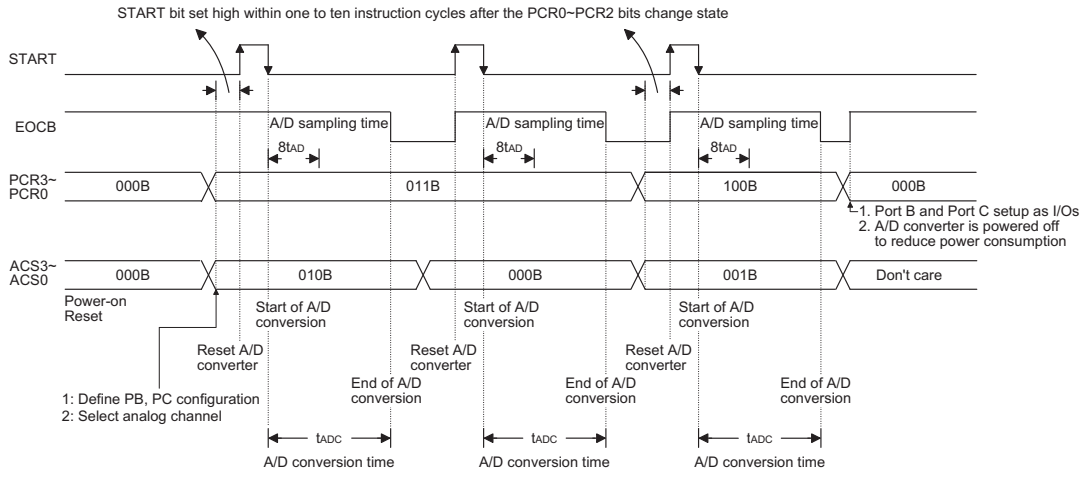
Initialising the A/D Converter

The internal A/D converter must be initialised in a special way. Each time the Port B and Port C A/D channel selection bits are modified by the program, the A/D converter must be re-initialised. If the A/D converter is not initialised after the channel selection bits are changed, the EOCB flag may have an undefined value, which may produce a false end of conversion signal. To initialise the A/D converter after the channel selection bits have changed, then, within a time frame of one to ten instruction cycles, the START bit in the ADCR register must first be set high and then immediately cleared to zero. This will ensure that the EOCB flag is correctly set to a high condition.

Summary of A/D Conversion Steps

The following summarises the individual steps that should be executed in order to implement an A/D conversion process.

- Step 1
  Select the required A/D conversion clock by correctly programming bits ADCS1 and ADCS0 in the ACSR register.
- Step 2
  Select which channel is to be connected to the internal A/D converter by correctly programming the ACS3~ACS0 bits which are also contained in the ADCR register.
- Step 3
  Select which pins on Port B and Port C are to be used as A/D inputs and configure them as A/D input pins by correctly programming the PCR3~PCR0 bits in the ACSR register.
- Step 4
  The analog to digital conversion process can now be initialised by setting the START bit in the ADCR register from "0" to "1" and then to "0" again. Note that this bit should have been originally set to "0".
- Step 5
  To check when the analog to digital conversion process is complete, the EOCB bit in the ADCR register can be polled. The conversion process is complete when this bit goes low. When this occurs the A/D data registers ADR can be read to obtain the conversion value. As an alternative method if the interrupts are enabled and the stack is not full, the program can wait for an A/D interrupt to occur.



Note: A/D clock must be fSYS/3, fSYS/6, fSYS/8, fSYS/16

A/D Conversion Timing

Note: When checking for the end of the conversion process, if the method of polling the EOCB bit in the ADCR register is used, the interrupt enable step above can be omitted.

The A/D conversion timing diagram shows graphically the various stages involved in an analog to digital conversion process and its associated timing.

The setting up and operation of the A/D converter function is fully under the control of the application program as there are no configuration options associated with the A/D converter. After an A/D conversion process has been initiated by the application program, the microcontroller internal hardware will begin to carry out the conversion, during which time the program can continue with other functions. The time taken for the A/D conversion is $16t_{AD}$ where t_{AD} is equal to the A/D clock period.

Programming Considerations

When programming, special attention must be given to the A/D channel selection bits in the ADSR register. If these bits are all cleared to zero no external pins will be selected for use as A/D input pins allowing the pins to be used as normal I/O pins. When this happens the power supplied to the internal A/D circuitry will be reduced resulting in a reduction of supply current. This ability to reduce power by turning off the internal A/D function by clearing the A/D channel selection bits may be an important consideration in battery powered applications.

Another important programming consideration is that when the A/D channel selection bits change value, the A/D converter must be re-initialised. This is achieved by pulsing the START bit in the ADCR register immediately after the channel selection bits have changed state. The exception to this is where the channel selection bits are all cleared, in which case the A/D converter is not required to be re-initialised.

A/D Programming Example

The following two programming examples illustrate how to setup and implement an A/D conversion. In the first example, the method of polling the EOCB bit in the ADCR register is used to detect when the conversion cycle is complete, whereas in the second example, the A/D interrupt is used to determine when the conversion is complete.

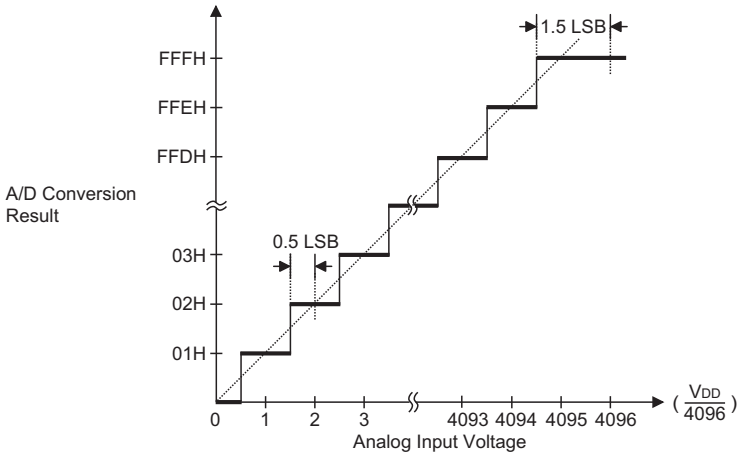
A/D Transfer Function

As the device contain a 12-bit A/D converter, its full-scale converted digitised value is equal to FFFH. Since the full-scale analog input value is equal to the VDD voltage, this gives a single bit analog input value of $V_{DD}/4096$. The diagram show the ideal transfer function between the analog input value and the digitised output value for the A/D converter.

Note that to reduce the quantisation error, a 2.5 LSB offset is added to the A/D Converter input. Except for the digitised zero value, the subsequent digitised values will change at a point 2.5 LSB below where they would change without the offset, and the last full scale digitised value will change at a point 2.5 LSB below the V_{DD} level.

Example: using an EOCB polling method to detect the end of conversion

```
    clr  EADI                  ; disable ADC interrupt
    mov  a,00011001B
    mov  acsr,a                ; setup the ACSR register to select fsys/6 as the A/D clock
                               ; setup the ACSR register to configure Port PB0~PB2 as A/D
                               ; Inputs
    mov  a,00000000B
    mov  adcr,a                ; setup the ADCR register and select AN0 to be connected to
                               ; the A/D converter
                               ; As the Port B and Port C channel bits have changed the
                               ; following START signal (0→1→0) must be issued within
                               ; 10 instruction cycles
    Set  ADON                  ; A/D on
Start_conversion:
    clr  START
    set  START                 ; reset A/D
    clr  START                 ; start A/D
Polling_EOC:
    sz   EOCB                  ; poll the ADCR register EOCB bit to detect end
                               ; of A/D conversion
    jmp  polling_EOC           ; continue polling
    mov  a,ADR                 ; read conversion result value
    mov  adrl_buffer,a         ; save result to user defined register
         :
    jmp  start_conversion      ; start next A/D conversion
```



Ideal A/D Transfer Function

SPI Serial Interface

The device includes one SPI Serial Interfaces. The SPI interface is a full duplex serial data link, originally designed by Motorola, which allows multiple devices connected to the same SPI bus to communicate with each other. The devices communicate using a master/slave technique where only the single master device can initiate a data transfer. A simple four line signal bus is used for all communication.

SPI Interface Communication

Four lines are used for each function. These are, SDI - Serial Data Input, SDO - Serial Data Output, SCK - Serial Clock and $\overline{SCS}$ - Slave Select. Note that the condition of the Slave Select line is conditioned by the CSEN bit in the SBCR control register. If the CSEN bit is high then the $\overline{SCS}$ line is active while if the bit is low then the $\overline{SCS}$ line will be in a floating condition. The accompanying timing diagram depicts the basic timing protocol of the SPI bus.

SPI Registers

There are three registers for control of the SPI Interface. These are the SBCR register which is the control register and the SBDR which is the data register and SPIR register which is the SPI mode control register. The SBCR register is used to setup the required setup parameters for the SPI bus and also used to store associated operating flags, while the SBDR register is used for data storage.

The SPIR register is used to select SPI mode, clock polarity edge selection and SPI enable or disable selection.

After Power on, the contents of the SBDR register will be in an unknown condition while the SBCR register will default to the condition below:

CKS	M1	M0	SBEN	MLS	CSEN	WCOL	TRF
0	1	1	0	0	0	0	0

Note that data written to the SBDR register will only be written to the TXRX buffer, whereas data read from the SBDR register will actual be read from the register.

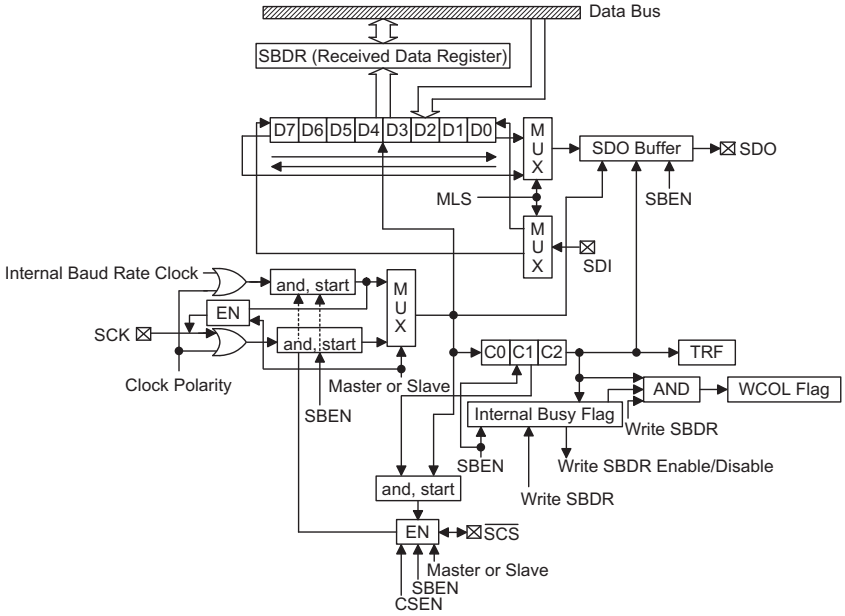
SPI Bus Enable/Disable

To enable the SPI bus, the SBEN bit should be set high, then wait for data to be written to the SBDR (TXRX buffer) register. For the Master Mode, after data has been written to the SBDR (TXRX buffer) register then transmission or reception will start automatically. When all the data has been transferred, the TRF bit should be set. For the Slave Mode, when clock pulses are received on SCK, data in the TXRX buffer will be shifted out or data on SDI will be shifted in.

To Disable the SPI bus SCK, SDI, SDO, $\overline{SCS}$ should be I/O mode.

Bit No.	Label	R/W	Function
0	SPI_CPOL	R/W	1: clock polarity rising 0: clock polarity falling (default falling)
1	SPI_MODE	R/W	0: SPI output the data in the rising edge(polarity=1) or falling edge (polarity=0); SPI read data in the in the falling edge(polarity=1) or rising edge (polarity=0); (default) 1: SPI first output the data immediately after the SPI is enable. And SPI output the data in the falling edge(polarity=1) or rising edge (polarity=0); SPI read data in the in the rising edge(polarity=1) or falling edge (polarity=0)
2	SPI_CSEN	R/W	1: SPI_CSEN Enable , this bit is used to enable/disable software CSEN function 0: SPI_CSEN disable, $\overline{SCS}$ define as GPIO (default disable)
3	SPI_EN	R/W	This bit control the shared PIN ($\overline{SCS}$, SDI, SDO and SCK) is SPI or GPIO mode 0: I/O mode (default) 1: SPI mode
7~4	Reserved bit	R/W	Always 0

SPIR Register



SPI Block Diagram

Note: WCOL: set by SPI cleared by users

CSEN: enable/disable chip selection function pin
master mode: 1/0 = with/without $\overline{SCS}$ output function
Slave mode: 1/0 = with/without $\overline{SCS}$ input control function

SBEN: enable/disable serial bus (0: initialise all status flags)
when SBEN=0, all status flags should be initialised
when SBEN=1, all SPI related function pins should stay at floating state

TRF: 1 = data transmitted or received, 0= data is transmitting or still not received

CPOL: I/O = clock polarity rising/falling edge: For SPIR Register.

If clock polarity set to rising edge (SPI_CPOL=1), serial clock timing follow SCK, otherwise (SPI_CPOL=0) SCK is the serial clock timing.

SPI Operation

All communication is carried out using the 4-line interface for both Master or Slave Mode. The timing diagram shows the basic operation of the bus.

The CSEN bit in the SBCR register controls the SCSB line of the SPI interface. Setting this bit high, will enable the SPI interface by allowing the $\overline{SCS}$ line to be active, which can then be used to control the SPI inteface. If the CSEN bit is low, the $\overline{SCS}$ line will be in a floating condition and can therefore not be used for control of the SPI interface. The SBEN bit in the SBCR register must also be high which will place the SDI line in a floating condition and the SDO line high. If in the Master Mode the SCK line will be either high or low depending upon the clock polarity configuration option. If in the Slave Mode the SCK line will be in a floating condition. If SBEN is low then the bus will be disabled and $\overline{SCS}$, SDI, SDO and SCK will all be in a floating condition.

In the Master Mode, the Master will always generate the clock signal. The clock and data transmission will be initiated after data has been written to the SBDR register. In the Slave Mode, the clock signal will be received from an external master device for both data transmission or reception. The following sequences show the order to be followed for data transfer in both Master and Slave Mode:

- Master Mode
  Step 1. Select the clock source using the CKS bit in the SBCR control register
  Step 2. Setup the M0 and M1 bits in the SBCR control register to select the Master Mode and the required Baud rate. Values of 00, 01 or 10 can be selected.
  Step 3. Setup the CSEN bit and setup the MLS bit to choose if the data is MSB or LSB first, this must be same as the Slave device.
  Step 4. Setup the SBEN bit in the SBCR control register to enable the SPI interface.
  Step 5. For write operations: write the data to the SBDR register, which will actually place the data into the TXRX buffer. Then use the SCK and $\overline{SCS}$ lines to output the data. Goto to step 6.For read operations: the data transferred in on the SDI line will be stored in the TXRX buffer until all the data has been received at which point it will be latched into the SBDR register.
  Step 6. Check the WCOL bit, if set high then a collision error has occurred so return to step5. If equal to zero then go to the following step.
  Step 7. Check the TRF bit or wait for an SPI serial bus interrupt.
  Step 8. Read data from the SBDR register.
  Step 9. Clear TRF.
  Step10. Goto step 5.
- Slave Mode:
  Step 1. The CKS bit has a don't care value in the slave mode.
  Step 2. Setup the M0 and M1 bits to 11 to select the Slave Mode. The CKS bit is don't care.
  Step 3. Setup the CSEN bit and setup the MLS bit to choose if the data is MSB or LSB first, this must be same as the Master device.
  Step 4. Setup the SBEN bit in the SBCR control register to enable the SPI interface.
  Step 5. For write operations: write data to the SBDR register, which will actually place the data into the TXRX register, then wait for the master clock and $\overline{SCS}$ signal. After this goto Step 6.
  For read operations: the data transferred in on the SDI line will be stored in the TXRX buffer until all the data has been received at which point it will be latched into the SBDR register.
  Step 6. Check the WCOL bit, if set high then a collision error has occurred so return to step5. If equal to zero then goto the following step.
  Step 7. Check the TRF bit or wait for an SPI serial bus interrupt.
  Step 8. Read data from the SBDR register.
  Step 9. Clear TRF
  Step10. step 5

SPI Configuration Options and Status Control

One option is to enable the operation of the WCOL, write collision bit, in the SBCR register. Some control in SPIR register. The SPI_CPOL select the clock polarity of the SCK line. The SPI_MODE select SPI data output mode.

SPI include four pins , can share I/O mode status . The status control combine with four bits for SPIR and SBCR register. Include SPI_CSEN , SPI_EN for SPIR register and CSEN, SBEN for SBCR register.

Control Bit for Register				SPI Share Function Pins Status			
SPI_EN	SPI_CSEN	SBEN	CSEN	$\overline{SCS}$	SCK	SDO	SDI
0	x	x	x	I/O mode	I/O mode	I/O mode	I/O mode
1	0	0	x	I/O mode	I/O mode	I/O mode	I/O mode
1	0	1	x	I/O mode	SPI mode	SPI mode	SPI mode (Z)
1	1	0	x	I/O mode	I/O mode	I/O mode	I/O mode
1	1	1	0	SPI mode (Z)	SPI mode	SPI mode	SPI mode (Z)
1	1	1	1	SPI mode	SPI mode	SPI mode	SPI mode (Z)

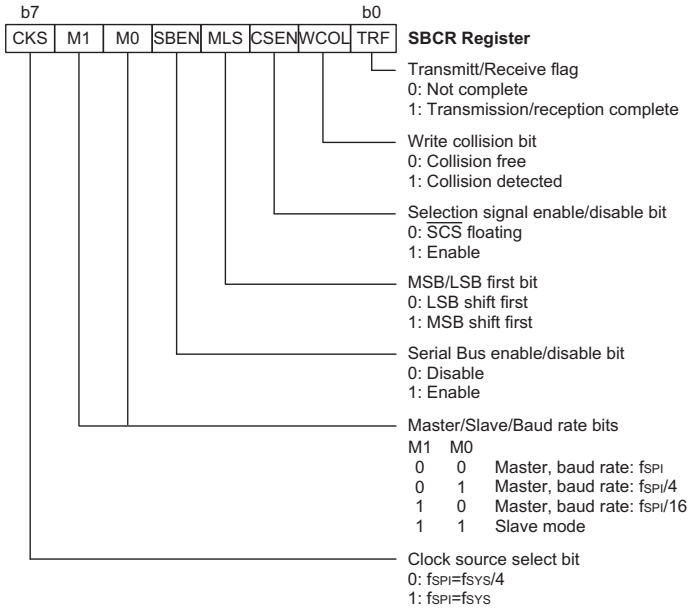
Note: X: don't care
(Z) floating

Error Detection

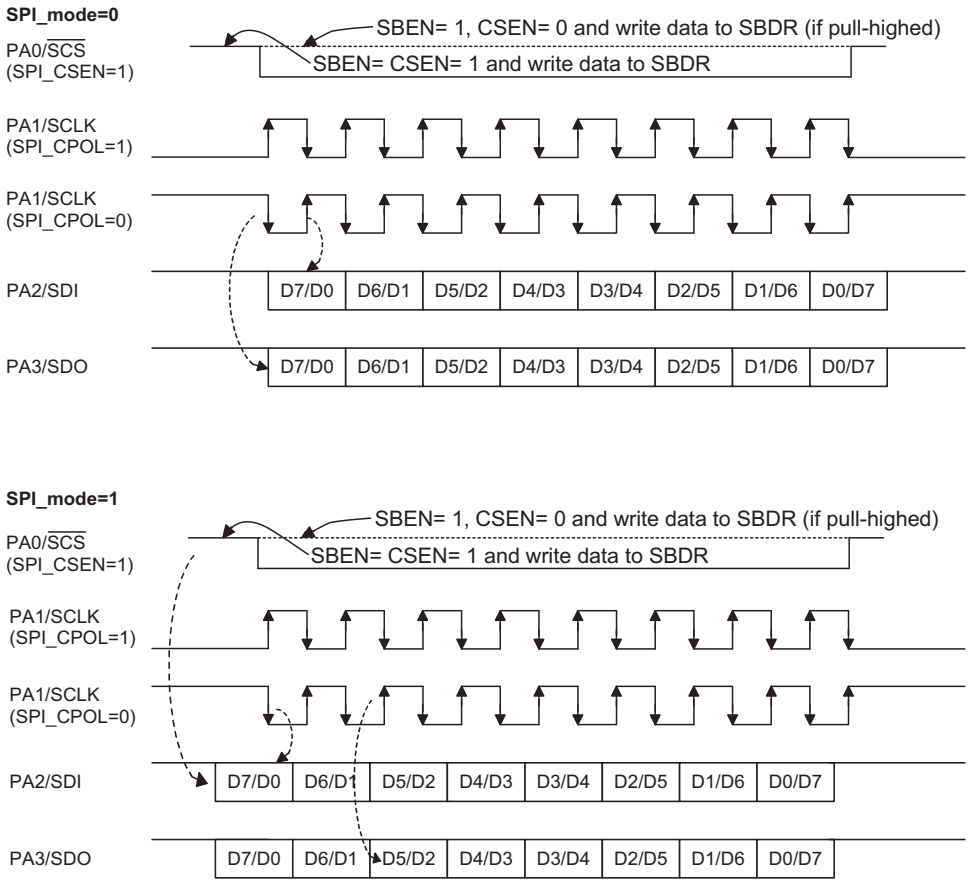
The WCOL bit in the SBCR register is provided to indicate errors during data transfer. The bit is set by the Serial Interface but must be cleared by the application program. This bit indicates a data collision has occurred which happens if a write to the SBDR register takes place during a data transfer operation and will prevent the write operation from continuing. The bit will be set high by the Serial Interface but has to be cleared by the user application program. The overall function of the WCOL bit can be disabled or enabled by a configuration option.

Programming Considerations

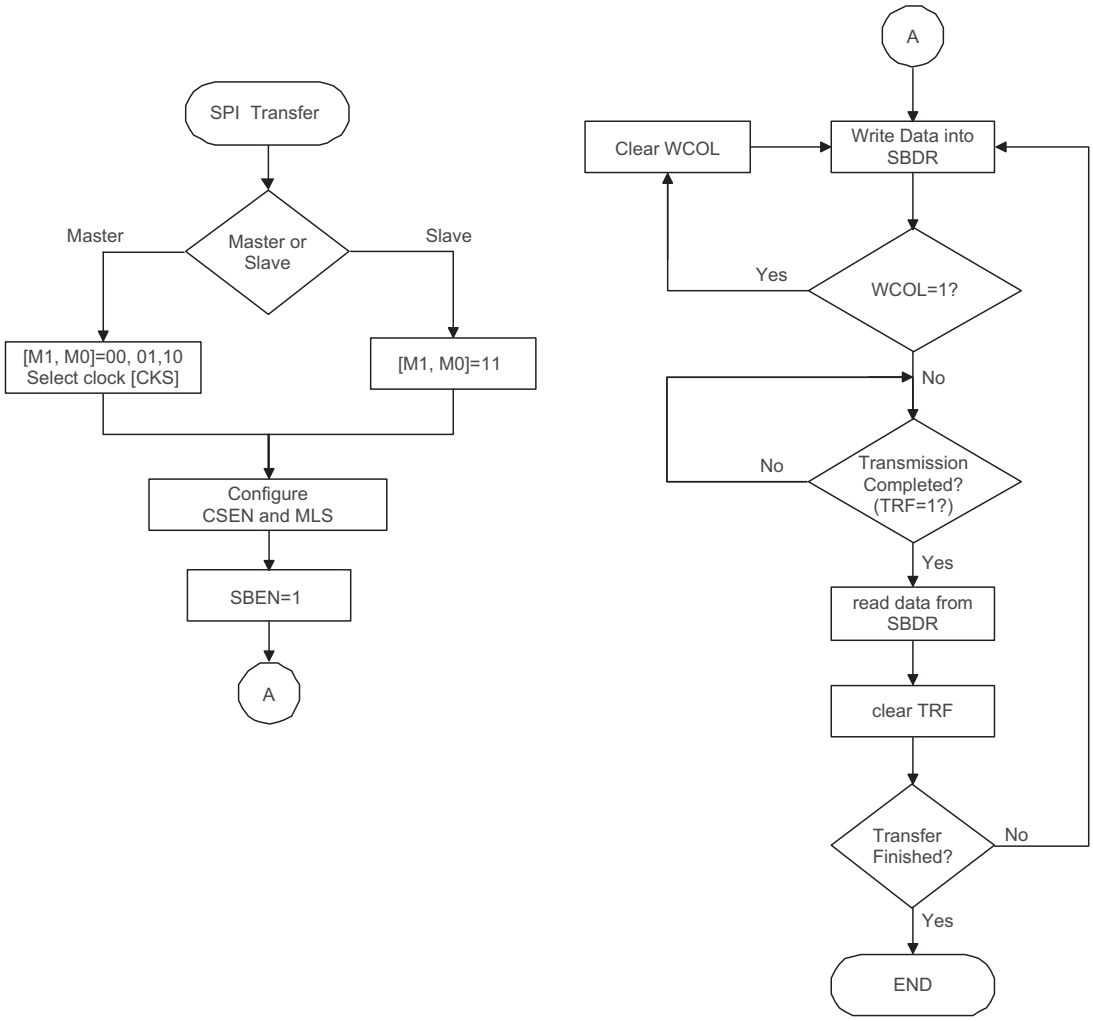
When the device is placed into the Power Down Mode note that data reception and transmission will continue. The TRF bit is used to generate an interrupt when the data has been transferred or received.



SPI Interface Control Register



SPI Bus Timing



SPI Transfer Control Flowchart

USB Interface

The device includes a USB interface function allowing for the convenient design of USB peripheral products.

HT82A620R/HT82A520R Power Plane

There are three power planes for the device: USB SIE VDD, VDDIO and the MCU VDD. For the USB SIE VDD will supply all circuits related to the USB SIE and be sourced from pin "UBUS". Once the USB is removed from the USB and there is no power in the USB BUS, the USB SIE circuit is no longer operational.

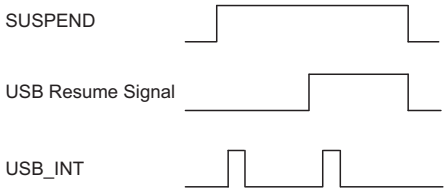
For the PB port, it can be configured using a configuration option to define the if the pins PB0~PB7 are supplied by either the MCU VDD, or if pins PB0~PB6 are supplied by the power pin VDDIO, in which case power will be supplied on pin PB7. In the latter configuration, PB7 will be configured as a power pin VDDIO and not a normal I/O pin.

For the MCU VDD, it supplies all the HT82A520R/HT82A620R circuit except the USB SIE which is supply by UBUS.

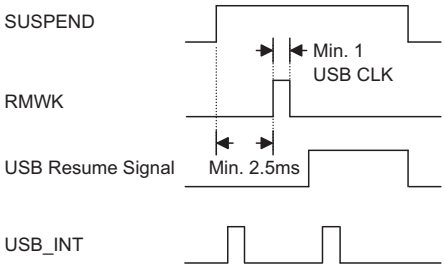
USB Suspend Wake-Up Remote Wake-Up

If there is no signal on the USB bus for over 3ms, the device will enter a suspend mode. The Suspend flag, SUSP, in the USC register, will then be set high and a USB interrupt will be generated to indicate that the device should jump to the suspend state to meet the requirements of the USB suspend current spec. In order to meet the requirements of the suspend current, the firmware should disable the USB clock by clearing the USBCKEN bit to zero.

The suspend current can be further decreased by setting the SUSP2 bit in the UCC register. When the resume signal is sent out by the host, the device will be woken up the by the USB interrupt and the Resume bit in the USC register will be set. To ensure correct device operation, the program must set the USBCKEN bit in the UCC register high and clear the SUSP2 bit in the UCC register. The Resume signal will be cleared before the Idle signal is sent out by the host and the Suspend line in the USC register will change to zero. So when the MCU detects the Suspend bit in the USC register, the condition of the Resume line should be noted and taken into consideration.



The device has a remote wake up function which can wake-up the USB Host by sending a wake-up pulse through RMWK in the USC register. Once the USB Host receives a wake-up signal from the device it will send a Resume signal to the device.



USB Interface Operation

The device has 4 Endpoints, EP0~EP3. EP0 supports Control transfer. All EP1~EP3 support Interrupt or Bulk transfer.

All endpoints except EP0 can be configured as 8, 16, 32 or 64 FIFO size using the register UFC0 and UFC1. EP0 has an 8-byte FIFO size. The Total FIFO size is 64+8 bytes. The URD in the USC register is the USB reset signal control function definition bit.

Bit No.	Label	R/W	Function
0	—	—	Unused bit, read as ″0″
1	PUB	R/W	1: D+, and D- have a 500kΩ pull-high 0: no pull-high (default on MCU reset)
2	SE0	R/W	This bit is used to indicate the SIE has detect a SE0 noise in the USB bus. This bit is set by SIE and clear by F/W.
3	SE1	R/W	This bit is used to indicate the SIE has detect a SE1 noise in the USB bus. This bit is set by SIE and clear by F/W.
4	PS2/DAI	R	USBD-/DATA input.
5	PS2/CKI	R	USBD+/CLK input.
6	PS2/DAO	W	Output for driving USBD-/DATA pin, when work under 3D PS2 mouse function. Default value is ″1″.
7	PS2/CKO	W	Output for driving USBD+/CLK pin, when work under 3D PS2 mouse function. Default value is ″1″.

USB_STAT Register

Bit No.	Label	R/W	Function
0	EP0EN	R/W	Controls the USB endpoint0 interrupt. Default value is ″0″. 1: enabled 0: disabled
1	EP1EN	R/W	Controls the USB endpoint1 interrupt. Default value is ″0″. 1: enabled 0: disabled
2	EP2EN	R/W	Controls the USB endpoint2 interrupt. Default value is ″0″. 1: enabled 0: disabled
3	EP3EN	R/W	Controls the USB endpoint3 interrupt. Default value is ″0″. 1: enabled 0: disabled
4~7	—	—	Unused bit, read as ″0″

UINT1 Register

Bit No.	Label	R/W	Function
0	SUSP	R	Read only, USB suspend indication. When this bit is set to ″1″ (set by SIE), it indicates that the USB bus has entered the suspend mode. The USB interrupt is also triggered when this bit changes from low to high. Default value is ″0″.
1	RMWK	R/W	USB remote wake-up command. It is set by MCU to leave the USB host leaving the suspend mode.
2	URST	R/W	USB reset indication. This bit is set/cleared by the USB SIE. This bit is used to detect a USB reset event on the USB bus. When this bit is set to ″1″, this indicates that a USB reset has occurred and that a USB interrupt will be initialized. Default value is ″0″. Default value is ″0″.
3	RESUME	R	USB resume indication. When the USB leaves the suspend mode, this bit is set to ″1″ (set by SIE). When the RESUME is set by SIE, an interrupt will be generated to wake-up the MCU. In order to detect the suspend state, the MCU should set USBCKEN and clear SUSP2 (in the UCC register) to enable the SIE detect function. RESUME will be cleared when the SUSP goes to ″0″. When the MCU is detecting the SUSP, the condition of RESUME (causes the MCU to wake-up) should be noted and taken into consideration. Default value is ″0″.
4	V33O	R/W	0/1: Turn-off/on V33O output. Default value is ″0″.
5	PLL	R/W	0: Turn-on PLL (default), 1: turn off PLL. Default value is ″0″.
6	SELPS2	R/W	When set to ′1′, indicated the chip work under PS2 mode. Default value is ″0″. Default value is ″0″.
7	URD	R/W	USB reset signal control function definition. 1: USB reset signal will reset MCU. (Default) 0: USB reset signal cannot reset MCU.

USC Register

The USR register which is the endpoint interrupt status register, is used to indicate which endpoint is accessed and to select the USB bus. The endpoint request flags, EP0F, EP1F, EP2F and EP3F, are used to indicate which endpoints are accessed. If an endpoint is accessed, the related endpoint request flag will be set high and a USB interrupt will be generated, if the USB interrupt is enabled and the stack is not full. When the active endpoint request flag is serviced, the endpoint request flag has to be cleared to zero using the program.

Bit No.	Label	R/W	Function
0	EP0F	R/W	When this bit is set to ″1″ (set by SIE), it indicates that endpoint 0 has been accessed and a USB interrupt will occur. When the interrupt has been serviced, this bit should be cleared by software. Default value is ″0″.
1	EP1F	R/W	When this bit is set to ″1″ (set by SIE), it indicates that endpoint 1 has been accessed and a USB interrupt will occur. When the interrupt has been serviced, this bit should be cleared by software. Default value is ″0″.
2	EP2F	R/W	When this bit is set to ″1″ (set by SIE), it indicates that endpoint 2 has been accessed and a USB interrupt will occur. When the interrupt has been serviced, this bit should be cleared by software. Default value is ″0″.
3	EP3F	R/W	When this bit is set to ″1″ (set by SIE), it indicates that endpoint 3 has been accessed and a USB interrupt will occur. When the interrupt has been serviced, this bit should be cleared by software. Default value is ″0″. Default value is ″0″.
4~7	—	—	Unused bit, read as ″0″

USR Register

There is a system clock control register to select the clock used in the MCU. This register consists of a USB clock control bit, USBCKEN, a second suspend mode control bit, SUSP2, and a system clock selection bit, SYSCLK.

The endpoint selection is determined by EPS2, EPS1 and EPS0.

Bit No.	Label	R/W	Function
0 1 2	EPS0 EPS1 EPS2	R/W	Accessing endpoint FIFO selection. Default value is ″000″. EPS2, EPS1, EPS0: 000: Select endpoint 0 FIFO (control) 001: Select endpoint 1 FIFO 010: Select endpoint 2 FIFO 011: Select endpoint 3 FIFO If the selected endpoints do not exist, the related functions will be absent.
3	USBCKEN	R/W	USB clock control bit. When this bit is set to ″1″, it indicates that the USB clock is enabled. Otherwise, the USB clock is turned-off. Default value is ″0″.
4	SUSP2	R/W	This bit is used for reducing power consumption in suspend mode. In normal mode, clean this bit to ″0″. (default) In halt mode, set this bit to ″1″ for reducing power consumption.
5	FSYS16MHz	R/W	This bit is used to define if the MCU system clock comes form an external OSC or comes from the PLL output 16MHz clock. 0: system clock sourced from OSC. (default) 1: system clock sourced from the PLL output 16MHz.
6	SYSCLK	R/W	This bit is used to specify the MCU system clock oscillator frequency. For a 6MHz crystal oscillator or resonator, set this bit to ″1″. For a 12MHz crystal oscillator or resonator, clear this bit to ″0″. (default)
7	RCTRL	R/W	This bit is used to control whether there is 7.5kΩ resistor between D+ and Vbus. 0: no 7.5kΩ between D+ and Vbus (default) 1: has 7.5kΩ between D+ and Vbus

UCC Register

The AWR register contains the current address and a remote wake up function control bit. The initial value of AWR is ″00H″. The address value extracted from the USB command has not to be loaded into this register until the SETUP stage has finished.

Bit No.	Label	R/W	Function
0	WKEN	R/W	USB remote-wake-up enable/disable (1/0). Default value is ″0″.
1~7	AD0~AD6	R/W	USB device address. Default value is ″0000000″.

AWR Register

The STALL register shows if the corresponding endpoint has worked properly or not. As soon as endpoint improper operation occurs, the related bit in the STALL register has to be set high. The STALL register bits will be cleared by a USB reset signal and a setup token event.

Bit No.	Label	R/W	Function
0~3	STL0~ STL3	R/W	Set by the users when related USB endpoints were stalled. Cleared by a USB reset. The STL0 is also cleared by a Setup Token event. Default value is ″0000″.
4~7	—	—	Unused bit, read as ″0″

STALL Register

The SIES register is only used for EP0 except for the NMI bit, which can control all endpoints

Bit No.	Label	R/W	Function
0	ASET	R/W	This bit is used to configure the SIE to automatically change the device address by the value stored in the AWR register. When this bit is set to ″1″ by firmware, the SIE will update the device address by the value stored in the AWR register after the PC host has successfully read the data from he device by an IN operation. Otherwise, when this bit is cleared to ″0″, the SIE will update the device address immediately after an address is written to the AWR register. So, in order to work properly, the firmware has to clear this bit after a next valid SETUP token is received. Default value is ″0″.
1	ERR	R/W	This bit is used to indicate that some errors have occurred when the FIFO is accessed. This bit is set by SIE and should be cleared by firmware. This bit is used for all endpoint. Default value is ″0″.
2	OUT	R/W	This bit is used to indicate the OUT token (except the OUT zero length token) has been received. The firmware clears this bit after the OUT data has been read. Also, this bit will be cleared by SIE after the next valid SETUP token is received. Default value is ″0″.
3	IN	R	This bit is used to indicate the current USB receiving signal from PC host is IN token.
4	NO ACK	R	This bit will set to ″1″ once SIE discover ther are some error condition so the SIE is not response (NAK or ACK or DATA) for the USB token. This bit is set by SIE and clear by F/W.
5	—	—	Unused bit, read as ″0″
6	CRCF	R/W	This bit will set to ″1″ when there are the following three condition is happened: CRC error, PID error, Bit stuffing error. This bit is set by SIE and clear by F/W. Default value is ″0″.
7	NMI	R/W	NAK token interrupt mask flag. If this bit set, when the device sent a NAK token to the host, an interrupt will be disabled. Otherwise if this bit is cleared, when the device sends a NAK token to the host, it will enter the interrupt sub-routine. This bit is used for all endpoint. Default value is ″0″.

SIES Register

The MISC register combines command and status to control the desired endpoint FIFO action and to show the status of the desired endpoint FIFO. MISC will be cleared by a USB reset signal.

Bit No.	Label	R/W	Function
0	REQUEST	R/W	After setting the status of the desired one, FIFO can be requested by setting this bit high. After finishing, this bit must be set low. Default value is ″0″.
1	TX	R/W	To represent the direction and transition end MCU access. When set to logic 1, the MCU desires to write data to the FIFO. After finishing, this bit must be set to logic 0 before terminating request to represent transition end. For an MCU read operation, this bit must be set to logic 0 and set to logic 1 after finishing. Default value is ″0″.
2	CLEAR	R/W	MCU requests to clear the FIFO, even if the FIFO is not ready. After clearing the FIFO, the USB interface will send force_tx_err to tell the Host that data under-run if the Host wants to read data. Default value is ″0″.
3~4	—	—	Unused bit, read as ″0″
5	SETCMD	R/W	To show that the data in the FIFO is a setup command. This bit is set by Hardware and clear by Firmware. Default value is ″0″.
6	READY	R	To show that the desired FIFO is ready.
7	LEN0	R	To show that the host sent a 0-sized packet to the MCU. This bit must be cleared by a read action to the corresponding FIFO.

MISC Register

Bit No.	Label	R/W	Function
0	FIFO_def	R/W	Once this bit set to ″1″ by Firmware, The SIE should redefine the FIFO configuration. This bit is automatically cleared by SIE
1	SETI1*	R/W	Input FIFO for EP1 enable 1/disable 0; default disable
2	SETI2*	R/W	Input FIFO for EP2 enable 1/disable 0; default disable
3	SETI3*	R/W	Input FIFO for EP3 enable 1/disable 0; default disable
4~7	—	—	Unused bit, read as ″0″

Note: ″*″ It is only required to set the data pipe as an input pile or output pile. The purpose of this function is to avoid the host sending an abnormal IN or OUT token and disabling the endpoint.

UFIEN Register, USB Endpoint 1~Endpoint 3 set IN Pipe Enable Register.

Bit No.	Label	R/W	Function
0	—	—	Unused bit, read as ″0″
1	SETO1**	R/W	Output FIFO for EP1 enable 1/disable 0; default disable
2	SETO2**	R/W	Output FIFO for EP2 enable 1/disable 0; default disable
3	SETO3**	R/W	Output FIFO for EP3 enable 1/disable 0; default disable
4~7	—	—	Unused bit, read as ″0″

Note: ″*″ USB definition: when the host sends a ″set Configuration″, the Data pipe should send the DATA0 (about the Data toggle) first. So, when the Device receives a ″set configuration″ setup command, the user needs to toggle this bit as the following data will send a Data0 first.

″**″ It is only required to set the data pipe as an input pile or output pile. The purpose of this function is to avoid the host sending a abnormal IN or OUT token and disabling the endpoint.

UFOEN Register, USB Endpoint 1~Endpoint 3 set OUT pipe enable register

Bit No.	Label	R/W	Function
0 1	RAM_def0 RAM_def1	R/W	00: RAM0 input FIFO, RAM1 output FIFO (default) 01: Both RAM0 and RAM1 are output FIFO 10: Both RAM0 and RAM1 are input FIFO 11: RAM0 output FIFO, RAM1 input FIFO
2 3	E1FS0 E1FS1	R/W	Define endpoint 1 FIFO size E1FS1, E1FS0: 00: 8-byte (default) 01: 16-byte 10: 32-byte 11: 64-byte
4 5	E2FS0 E2FS1	R/W	Define endpoint 2 FIFO size E2FS1, E2FS0: 00: 8-byte (default) 01: 16-byte 10: 32-byte 11: 64-byte
6 7	E3FS0 E3FS1	R/W	Define endpoint 3 FIFO size E3FS1, E3FS0: 00: 8-byte (default) 01: 16-byte 10: 32-byte 11: 64-byte

UFC0 USB FIFO Size Control Register 0

The total FIFO size is 64+8 bytes. All endpoints except EP0 can be defined by registers UFOEN, UFIEN, UFC0 and UFC. There are three FIFO mapped as follow:

8 bytes FIFO for Endpoint0

RAM0 FIFO for other input Endpoint (1~3)

RAM1 FIFO for other output Endpoint (1~3)

Label	R/W	Function
FIFO0~FIFO3	R/W	EPi accessing register (i=0~3). When an endpoint is disabled, the corresponding accessing register should be disabled.

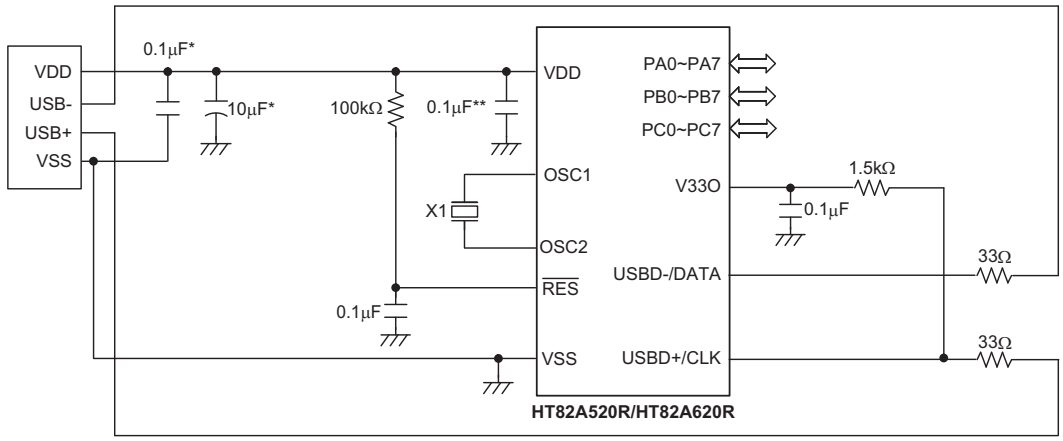
FIFO0~FIFO3 USB Endpoint Accessing Registers Definitions

Configuration Options

No.	Options
1	PA pull-high enable/disable by bit
2	PB pull-high enable/disable by nibble
3	PC pull-high enable/disable by nibble
4	PB wake-up enable/disable by nibble
5	PC wake-up enable/disable by nibble
6	USB D- pin internal 1.5kΩ resistor enable/disable
7	USB D+ pin internal 7.5kΩ resistor enable/disable
8	TBHP enable or disable
9	Low voltage reset: enable/disable
10	WDT enable/disable
11	WDT clock source: $f_{SYS}/4$ or WDTOSC (32K RC)
12	CLR WDT instructions: one or two clear WDT instruction(s)
13	PA NMOS or CMOS output type
14	PA wake-up enable/disable by bit
16	PB7 mode: GPIO or V_{DDIO} pin for PB0~PB6 power source
17	VDD PB0~PB6: PB0~PB6 power source from V_{DD} or V_{DDIO}
18	SPI_WCOL: enable/disable

Application Circuits

Crystal or Ceramic Resonator for Multiple I/O Applications



Note: The resistance and capacitance for reset circuit should be designed in such a way as to ensure that VDD is stable and remains within a valid operating voltage range before bringing $\overline{RES}$ high.

X1 can use 6MHz or 12MHz, X1 as close OSC1 & OSC2 as possible.

* These capacitors should be placed close to the USB connector.

** This capacitor should be placed close to the MCU.

Instruction Set

Introduction

Central to the successful operation of any microcontroller is its instruction set, which is a set of program instruction codes that directs the microcontroller to perform certain operations. In the case of Holtek microcontrollers, a comprehensive and flexible set of over 60 instructions is provided to enable programmers to implement their application with the minimum of programming overheads.

For easier understanding of the various instruction codes, they have been subdivided into several functional groupings.

Instruction Timing

Most instructions are implemented within one instruction cycle. The exceptions to this are branch, call, or table read instructions where two instruction cycles are required. One instruction cycle is equal to 4 system clock cycles, therefore in the case of an 8MHz system oscillator, most instructions would be implemented within 0.5μs and branch or call instructions would be implemented within 1μs. Although instructions which require one more cycle to implement are generally limited to the JMP, CALL, RET, RETI and table read instructions, it is important to realize that any other instructions which involve manipulation of the Program Counter Low register or PCL will also take one more cycle to implement. As instructions which change the contents of the PCL will imply a direct jump to that new address, one more cycle will be required. Examples of such instructions would be ″CLR PCL″ or ″MOV PCL, A″. For the case of skip instructions, it must be noted that if the result of the comparison involves a skip operation then this will also take one more cycle, if no skip is involved then only one cycle is required.

Moving and Transferring Data

The transfer of data within the microcontroller program is one of the most frequently used operations. Making use of three kinds of MOV instructions, data can be transferred from registers to the Accumulator and vice-versa as well as being able to move specific immediate data directly into the Accumulator. One of the most important data transfer applications is to receive data from the input ports and transfer data to the output ports.

Arithmetic Operations

The ability to perform certain arithmetic operations and data manipulation is a necessary feature of most microcontroller applications. Within the Holtek microcontroller instruction set are a range of add and subtract instruction mnemonics to enable the necessary arithmetic to be carried out. Care must be taken to ensure correct handling of carry and borrow data when results exceed 255 for addition and less than 0 for subtraction. The increment and decrement instructions INC, INCA, DEC and DECA provide a simple means of increasing or decreasing by a value of one of the values in the destination specified.

Logical and Rotate Operations

The standard logical operations such as AND, OR, XOR and CPL all have their own instruction within the Holtek microcontroller instruction set. As with the case of most instructions involving data manipulation, data must pass through the Accumulator which may involve additional programming steps. In all logical data operations, the zero flag may be set if the result of the operation is zero. Another form of logical data manipulation comes from the rotate instructions such as RR, RL, RRC and RLC which provide a simple means of rotating one bit right or left. Different rotate instructions exist depending on program requirements. Rotate instructions are useful for serial port programming applications where data can be rotated from an internal register into the Carry bit from where it can be examined and the necessary serial bit set high or low. Another application where rotate data operations are used is to implement multiplication and division calculations.

Branches and Control Transfer

Program branching takes the form of either jumps to specified locations using the JMP instruction or to a subroutine using the CALL instruction. They differ in the sense that in the case of a subroutine call, the program must return to the instruction immediately when the subroutine has been carried out. This is done by placing a return instruction RET in the subroutine which will cause the program to jump back to the address right after the CALL instruction. In the case of a JMP instruction, the program simply jumps to the desired location. There is no requirement to jump back to the original jumping off point as in the case of the CALL instruction. One special and extremely useful set of branch instructions are the conditional branches. Here a decision is first made regarding the condition of a certain data memory or individual bits. Depending upon the conditions, the program will continue with the next instruction or skip over it and jump to the following instruction. These instructions are the key to decision making and branching within the program perhaps determined by the condition of certain input switches or by the condition of internal data bits.

Bit Operations

The ability to provide single bit operations on Data Memory is an extremely flexible feature of all Holtek microcontrollers. This feature is especially useful for output port bit programming where individual bits or port pins can be directly set high or low using either the "SET [m].i" or "CLR [m].i" instructions respectively. The feature removes the need for programmers to first read the 8-bit output port, manipulate the input data to ensure that other bits are not changed and then output the port with the correct new data. This read-modify-write process is taken care of automatically when these bit operation instructions are used.

Table Read Operations

Data storage is normally implemented by using registers. However, when working with large amounts of fixed data, the volume involved often makes it inconvenient to store the fixed data in the Data Memory. To overcome this problem, Holtek microcontrollers allow an area of Program Memory to be setup as a table where data can be directly stored. A set of easy to use instructions provides the means by which this fixed data can be referenced and retrieved from the Program Memory.

Other Operations

In addition to the above functional instructions, a range of other instructions also exist such as the "HALT" instruction for Power-down operations and instructions to control the operation of the Watchdog Timer for reliable program operations under extreme electric or electromagnetic environments. For their relevant operations, refer to the functional related sections.

Instruction Set Summary

The following table depicts a summary of the instruction set categorised according to function and can be consulted as a basic instruction reference using the following listed conventions.

Table conventions:

x: Bits immediate data

m: Data Memory address

A: Accumulator

i: 0~7 number of bits

addr: Program memory address

Mnemonic	Description	Cycles	Flag Affected
Arithmetic			
ADD A,[m]	Add Data Memory to ACC	1	Z, C, AC, OV
ADDM A,[m]	Add ACC to Data Memory	1Note	Z, C, AC, OV
ADD A,x	Add immediate data to ACC	1	Z, C, AC, OV
ADC A,[m]	Add Data Memory to ACC with Carry	1	Z, C, AC, OV
ADCM A,[m]	Add ACC to Data memory with Carry	1Note	Z, C, AC, OV
SUB A,x	Subtract immediate data from the ACC	1	Z, C, AC, OV
SUB A,[m]	Subtract Data Memory from ACC	1	Z, C, AC, OV
SUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory	1Note	Z, C, AC, OV
SBC A,[m]	Subtract Data Memory from ACC with Carry	1	Z, C, AC, OV
SBCM A,[m]	Subtract Data Memory from ACC with Carry, result in Data Memory	1Note	Z, C, AC, OV
DAA [m]	Decimal adjust ACC for Addition with result in Data Memory	1Note	C
Logic Operation			
AND A,[m]	Logical AND Data Memory to ACC	1	Z
OR A,[m]	Logical OR Data Memory to ACC	1	Z
XOR A,[m]	Logical XOR Data Memory to ACC	1	Z
ANDM A,[m]	Logical AND ACC to Data Memory	1Note	Z
ORM A,[m]	Logical OR ACC to Data Memory	1Note	Z
XORM A,[m]	Logical XOR ACC to Data Memory	1Note	Z
AND A,x	Logical AND immediate Data to ACC	1	Z
OR A,x	Logical OR immediate Data to ACC	1	Z
XOR A,x	Logical XOR immediate Data to ACC	1	Z
CPL [m]	Complement Data Memory	1Note	Z
CPLA [m]	Complement Data Memory with result in ACC	1	Z
Increment & Decrement			
INCA [m]	Increment Data Memory with result in ACC	1	Z
INC [m]	Increment Data Memory	1Note	Z
DECA [m]	Decrement Data Memory with result in ACC	1	Z
DEC [m]	Decrement Data Memory	1Note	Z

Mnemonic	Description	Cycles	Flag Affected
Rotate			
RRA [m]	Rotate Data Memory right with result in ACC	1	None
RR [m]	Rotate Data Memory right	1Note	None
RRCA [m]	Rotate Data Memory right through Carry with result in ACC	1	C
RRC [m]	Rotate Data Memory right through Carry	1Note	C
RLA [m]	Rotate Data Memory left with result in ACC	1	None
RL [m]	Rotate Data Memory left	1Note	None
RLCA [m]	Rotate Data Memory left through Carry with result in ACC	1	C
RLC [m]	Rotate Data Memory left through Carry	1Note	C
Data Move			
MOV A,[m]	Move Data Memory to ACC	1	None
MOV [m],A	Move ACC to Data Memory	1Note	None
MOV A,x	Move immediate data to ACC	1	None
Bit Operation			
CLR [m].i	Clear bit of Data Memory	1Note	None
SET [m].i	Set bit of Data Memory	1Note	None
Branch			
JMP addr	Jump unconditionally	2	None
SZ [m]	Skip if Data Memory is zero	1Note	None
SZA [m]	Skip if Data Memory is zero with data movement to ACC	1note	None
SZ [m].i	Skip if bit i of Data Memory is zero	1Note	None
SNZ [m].i	Skip if bit i of Data Memory is not zero	1Note	None
SIZ [m]	Skip if increment Data Memory is zero	1Note	None
SDZ [m]	Skip if decrement Data Memory is zero	1Note	None
SIZA [m]	Skip if increment Data Memory is zero with result in ACC	1Note	None
SDZA [m]	Skip if decrement Data Memory is zero with result in ACC	1Note	None
CALL addr	Subroutine call	2	None
RET	Return from subroutine	2	None
RET A,x	Return from subroutine and load immediate data to ACC	2	None
RETI	Return from interrupt	2	None
Table Read			
TABRDC [m](4)	Read ROM code (locate by TBLP and TBHP) to data memory and TBLH	2Note	None
TABRDC [m](5)	Read ROM code (current page) to data memory and TBLH	2Note	None
TABRDL [m]	Read table (last page) to TBLH and Data Memory	2Note	None
Miscellaneous			
NOP	No operation	1	None
CLR [m]	Clear Data Memory	1Note	None
SET [m]	Set Data Memory	1Note	None
CLR WDT	Clear Watchdog Timer	1	TO, PDF
CLR WDT1	Pre-clear Watchdog Timer	1	TO, PDF
CLR WDT2	Pre-clear Watchdog Timer	1	TO, PDF
SWAP [m]	Swap nibbles of Data Memory	1Note	None
SWAPA [m]	Swap nibbles of Data Memory with result in ACC	1	None
HALT	Enter power down mode	1	TO, PDF

Note: 1. For skip instructions, if the result of the comparison involves a skip then two cycles are required, if no skip takes place only one cycle is required.

2. Any instruction which changes the contents of the PCL will also require 2 cycles for execution.

3. For the ″CLR WDT1″ and ″CLR WDT2″ instructions the TO and PDF flags may be affected by the execution status. The TO and PDF flags are cleared after both ″CLR WDT1″ and ″CLR WDT2″ instructions are consecutively executed. Otherwise the TO and PDF flags remain unchanged.

4. Configuration option ″TBHP option″ is enabled

5. Configuration option ″TBHP option″ is disabled

Instruction Definition

ADC A,[m]	Add Data Memory to ACC with Carry
Description	The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the Accumulator.
Operation	ACC ← ACC + [m] + C
Affected flag(s)	OV, Z, AC, C
ADCM A,[m]	Add ACC to Data Memory with Carry
Description	The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the specified Data Memory.
Operation	[m] ← ACC + [m] + C
Affected flag(s)	OV, Z, AC, C
ADD A,[m]	Add Data Memory to ACC
Description	The contents of the specified Data Memory and the Accumulator are added. The result is stored in the Accumulator.
Operation	ACC ← ACC + [m]
Affected flag(s)	OV, Z, AC, C
ADD A,x	Add immediate data to ACC
Description	The contents of the Accumulator and the specified immediate data are added. The result is stored in the Accumulator.
Operation	ACC ← ACC + x
Affected flag(s)	OV, Z, AC, C
ADDM A,[m]	Add ACC to Data Memory
Description	The contents of the specified Data Memory and the Accumulator are added. The result is stored in the specified Data Memory.
Operation	[m] ← ACC + [m]
Affected flag(s)	OV, Z, AC, C
AND A,[m]	Logical AND Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical AND operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″AND″ [m]
Affected flag(s)	Z
AND A,x	Logical AND immediate data to ACC
Description	Data in the Accumulator and the specified immediate data perform a bitwise logical AND operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″AND″ x
Affected flag(s)	Z
ANDM A,[m]	Logical AND ACC to Data Memory
Description	Data in the specified Data Memory and the Accumulator perform a bitwise logical AND operation. The result is stored in the Data Memory.
Operation	[m] ← ACC ″AND″ [m]
Affected flag(s)	Z

CALL addr	Subroutine call
Description	Unconditionally calls a subroutine at the specified address. The Program Counter then increments by 1 to obtain the address of the next instruction which is then pushed onto the stack. The specified address is then loaded and the program continues execution from this new address. As this instruction requires an additional operation, it is a two cycle instruction.
Operation	Stack ← Program Counter + 1 Program Counter ← addr
Affected flag(s)	None
CLR [m]	Clear Data Memory
Description	Each bit of the specified Data Memory is cleared to 0.
Operation	[m] ← 00H
Affected flag(s)	None
CLR [m].i	Clear bit of Data Memory
Description	Bit i of the specified Data Memory is cleared to 0.
Operation	[m].i ← 0
Affected flag(s)	None
CLR WDT	Clear Watchdog Timer
Description	The TO, PDF flags and the WDT are all cleared.
Operation	WDT cleared TO ← 0 PDF ← 0
Affected flag(s)	TO, PDF
CLR WDT1	Pre-clear Watchdog Timer
Description	The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT2 and must be executed alternately with CLR WDT2 to have effect. Repetitively executing this instruction without alternately executing CLR WDT2 will have no effect.
Operation	WDT cleared TO ← 0 PDF ← 0
Affected flag(s)	TO, PDF
CLR WDT2	Pre-clear Watchdog Timer
Description	The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT1 and must be executed alternately with CLR WDT1 to have effect. Repetitively executing this instruction without alternately executing CLR WDT1 will have no effect.
Operation	WDT cleared TO ← 0 PDF ← 0
Affected flag(s)	TO, PDF

CPL [m] Complement Data Memory

Description Each bit of the specified Data Memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice versa.

Operation $[m] \leftarrow \overline{[m]}$

Affected flag(s) Z

CPLA [m] Complement Data Memory with result in ACC

Description Each bit of the specified Data Memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice versa. The complemented result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation $ACC \leftarrow \overline{[m]}$

Affected flag(s) Z

DAA [m] Decimal-Adjust ACC for addition with result in Data Memory

Description Convert the contents of the Accumulator value to a BCD (Binary Coded Decimal) value resulting from the previous addition of two BCD variables. If the low nibble is greater than 9 or if AC flag is set, then a value of 6 will be added to the low nibble. Otherwise the low nibble remains unchanged. If the high nibble is greater than 9 or if the C flag is set, then a value of 6 will be added to the high nibble. Essentially, the decimal conversion is performed by adding 00H, 06H, 60H or 66H depending on the Accumulator and flag conditions. Only the C flag may be affected by this instruction which indicates that if the original BCD sum is greater than 100, it allows multiple precision decimal addition.

Operation $[m] \leftarrow ACC + 00H$ or
$[m] \leftarrow ACC + 06H$ or
$[m] \leftarrow ACC + 60H$ or
$[m] \leftarrow ACC + 66H$

Affected flag(s) C

DEC [m] Decrement Data Memory

Description Data in the specified Data Memory is decremented by 1.

Operation $[m] \leftarrow [m] - 1$

Affected flag(s) Z

DECA [m] Decrement Data Memory with result in ACC

Description Data in the specified Data Memory is decremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.

Operation $ACC \leftarrow [m] - 1$

Affected flag(s) Z

HALT Enter power down mode

Description This instruction stops the program execution and turns off the system clock. The contents of the Data Memory and registers are retained. The WDT and prescaler are cleared. The power down flag PDF is set and the WDT time-out flag TO is cleared.

Operation $TO \leftarrow 0$
$PDF \leftarrow 1$

Affected flag(s) TO, PDF

INC [m]	Increment Data Memory
Description	Data in the specified Data Memory is incremented by 1.
Operation	[m] ← [m] + 1
Affected flag(s)	Z
INCA [m]	Increment Data Memory with result in ACC
Description	Data in the specified Data Memory is incremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.
Operation	ACC ← [m] + 1
Affected flag(s)	Z
JMP addr	Jump unconditionally
Description	The contents of the Program Counter are replaced with the specified address. Program execution then continues from this new address. As this requires the insertion of a dummy instruction while the new address is loaded, it is a two cycle instruction.
Operation	Program Counter ← addr
Affected flag(s)	None
MOV A,[m]	Move Data Memory to ACC
Description	The contents of the specified Data Memory are copied to the Accumulator.
Operation	ACC ← [m]
Affected flag(s)	None
MOV A,x	Move immediate data to ACC
Description	The immediate data specified is loaded into the Accumulator.
Operation	ACC ← x
Affected flag(s)	None
MOV [m],A	Move ACC to Data Memory
Description	The contents of the Accumulator are copied to the specified Data Memory.
Operation	[m] ← ACC
Affected flag(s)	None
NOP	No operation
Description	No operation is performed. Execution continues with the next instruction.
Operation	No operation
Affected flag(s)	None
OR A,[m]	Logical OR Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical OR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″OR″ [m]
Affected flag(s)	Z

OR A,x Logical OR immediate data to ACC

Description: Data in the Accumulator and the specified immediate data perform a bitwise logical OR operation. The result is stored in the Accumulator.

Operation: ACC ← ACC "OR" x

Affected flag(s): Z

ORM A,[m] Logical OR ACC to Data Memory

Description: Data in the specified Data Memory and the Accumulator perform a bitwise logical OR operation. The result is stored in the Data Memory.

Operation: [m] ← ACC "OR" [m]

Affected flag(s): Z

RET Return from subroutine

Description: The Program Counter is restored from the stack. Program execution continues at the restored address.

Operation: Program Counter ← Stack

Affected flag(s): None

RET A,x Return from subroutine and load immediate data to ACC

Description: The Program Counter is restored from the stack and the Accumulator loaded with the specified immediate data. Program execution continues at the restored address.

Operation: Program Counter ← Stack
ACC ← x

Affected flag(s): None

RETI Return from interrupt

Description: The Program Counter is restored from the stack and the interrupts are re-enabled by setting the EMI bit. EMI is the master interrupt global enable bit. If an interrupt was pending when the RETI instruction is executed, the pending Interrupt routine will be processed before returning to the main program.

Operation: Program Counter ← Stack
EMI ← 1

Affected flag(s): None

RL [m] Rotate Data Memory left

Description: The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0.

Operation: [m].(i+1) ← [m].i; (i = 0~6)
[m].0 ← [m].7

Affected flag(s): None

RLA [m] Rotate Data Memory left with result in ACC

Description: The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation: ACC.(i+1) ← [m].i; (i = 0~6)
ACC.0 ← [m].7

Affected flag(s): None

RLC [m] Rotate Data Memory left through Carry

Description The contents of the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into bit 0.

Operation
[m].(i+1) ← [m].i; (i = 0~6)
[m].0 ← C
C ← [m].7

Affected flag(s) C

RLCA [m] Rotate Data Memory left through Carry with result in ACC

Description Data in the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into the bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation
ACC.(i+1) ← [m].i; (i = 0~6)
ACC.0 ← C
C ← [m].7

Affected flag(s) C

RR [m] Rotate Data Memory right

Description The contents of the specified Data Memory are rotated right by 1 bit with bit 0 rotated into bit 7.

Operation
[m].i ← [m].(i+1); (i = 0~6)
[m].7 ← [m].0

Affected flag(s) None

RRA [m] Rotate Data Memory right with result in ACC

Description Data in the specified Data Memory and the carry flag are rotated right by 1 bit with bit 0 rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation
ACC.i ← [m].(i+1); (i = 0~6)
ACC.7 ← [m].0

Affected flag(s) None

RRC [m] Rotate Data Memory right through Carry

Description The contents of the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7.

Operation
[m].i ← [m].(i+1); (i = 0~6)
[m].7 ← C
C ← [m].0

Affected flag(s) C

RRCA [m] Rotate Data Memory right through Carry with result in ACC

Description Data in the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation
ACC.i ← [m].(i+1); (i = 0~6)
ACC.7 ← C
C ← [m].0

Affected flag(s) C

SBC A,[m] Subtract Data Memory from ACC with Carry

Description The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation $ACC \leftarrow ACC - [m] - \overline{C}$

Affected flag(s) OV, Z, AC, C

SBCM A,[m] Subtract Data Memory from ACC with Carry and result in Data Memory

Description The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation $[m] \leftarrow ACC - [m] - \overline{C}$

Affected flag(s) OV, Z, AC, C

SDZ [m] Skip if decrement Data Memory is 0

Description The contents of the specified Data Memory are first decremented by 1. If the result is 0 the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation $[m] \leftarrow [m] - 1$
Skip if $[m] = 0$

Affected flag(s) None

SDZA [m] Skip if decrement Data Memory is zero with result in ACC

Description The contents of the specified Data Memory are first decremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction.

Operation $ACC \leftarrow [m] - 1$
Skip if $ACC = 0$

Affected flag(s) None

SET [m] Set Data Memory

Description Each bit of the specified Data Memory is set to 1.

Operation $[m] \leftarrow FFH$

Affected flag(s) None

SET [m].i Set bit of Data Memory

Description Bit i of the specified Data Memory is set to 1.

Operation $[m].i \leftarrow 1$

Affected flag(s) None

SIZ [m]	Skip if increment Data Memory is 0
Description	The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	[m] ← [m] + 1 Skip if [m] = 0
Affected flag(s)	None
SIZA [m]	Skip if increment Data Memory is zero with result in ACC
Description	The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	ACC ← [m] + 1 Skip if ACC = 0
Affected flag(s)	None
SNZ [m].i	Skip if bit i of Data Memory is not 0
Description	If bit i of the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction.
Operation	Skip if [m].i ≠ 0
Affected flag(s)	None
SUB A,[m]	Subtract Data Memory from ACC
Description	The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	ACC ← ACC – [m]
Affected flag(s)	OV, Z, AC, C
SUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory
Description	The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	[m] ← ACC – [m]
Affected flag(s)	OV, Z, AC, C
SUB A,x	Subtract immediate data from ACC
Description	The immediate data specified by the code is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.
Operation	ACC ← ACC – x
Affected flag(s)	OV, Z, AC, C

SWAP [m]	Swap nibbles of Data Memory
Description	The low-order and high-order nibbles of the specified Data Memory are interchanged.
Operation	[m].3~[m].0 ↔ [m].7 ~ [m].4
Affected flag(s)	None
SWAPA [m]	Swap nibbles of Data Memory with result in ACC
Description	The low-order and high-order nibbles of the specified Data Memory are interchanged. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.
Operation	ACC.3 ~ ACC.0 ← [m].7 ~ [m].4 ACC.7 ~ ACC.4 ← [m].3 ~ [m].0
Affected flag(s)	None
SZ [m]	Skip if Data Memory is 0
Description	If the contents of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	Skip if [m] = 0
Affected flag(s)	None
SZA [m]	Skip if Data Memory is 0 with data movement to ACC
Description	The contents of the specified Data Memory are copied to the Accumulator. If the value is zero, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.
Operation	ACC ← [m] Skip if [m] = 0
Affected flag(s)	None
SZ [m].i	Skip if bit i of Data Memory is 0
Description	If bit i of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction.
Operation	Skip if [m].i = 0
Affected flag(s)	None
TABRDC [m]	Read table (current page) to TBLH and Data Memory
Description	The low byte of the program code (current page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.
Operation	[m] ← program code (low byte) TBLH ← program code (high byte)
Affected flag(s)	None
TABRDC [m]	Move the ROM code (locate by TBLP and TBHP) to TBLH and data memory (ROM code TBHP is enabled)
Description	The low byte of ROM code addressed by the table pointers (TBLP and TBHP) is moved to the specified data memory and the high byte transferred to TBLH directly.
Operation	[m] ← program code (low byte) TBLH ← program code (high byte)
Affected flag(s)	None

TABRDL [m]	Read table (last page) to TBLH and Data Memory
Description	The low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.
Operation	[m] ← program code (low byte) TBLH ← program code (high byte)
Affected flag(s)	None

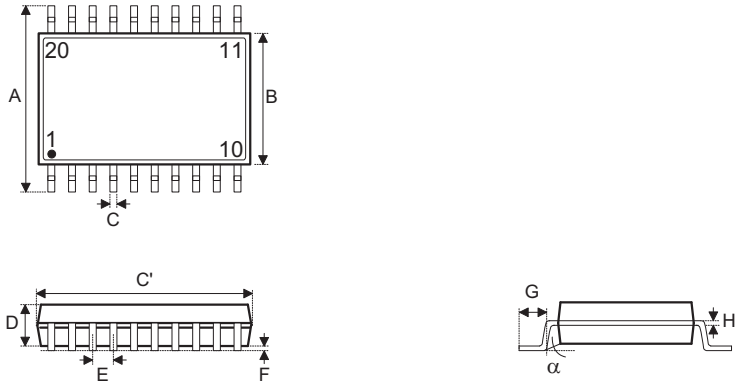
XOR A,[m]	Logical XOR Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical XOR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″XOR″ [m]
Affected flag(s)	Z

XORM A,[m]	Logical XOR ACC to Data Memory
Description	Data in the specified Data Memory and the Accumulator perform a bitwise logical XOR operation. The result is stored in the Data Memory.
Operation	[m] ← ACC ″XOR″ [m]
Affected flag(s)	Z

XOR A,x	Logical XOR immediate data to ACC
Description	Data in the Accumulator and the specified immediate data perform a bitwise logical XOR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″XOR″ x
Affected flag(s)	Z

Package Information

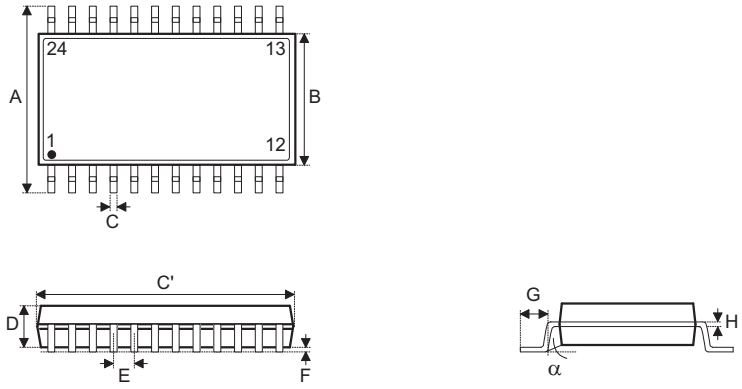
20-pin SSOP (150mil) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.228	—	0.244
B	0.150	—	0.158
C	0.008	—	0.012
C′	0.335	—	0.347
D	0.049	—	0.065
E	—	0.025	—
F	0.004	—	0.010
G	0.015	—	0.050
H	0.007	—	0.010
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	5.79	—	6.20
B	3.81	—	4.01
C	0.20	—	0.30
C′	8.51	—	8.81
D	1.24	—	1.65
E	—	0.64	—
F	0.10	—	0.25
G	0.38	—	1.27
H	0.18	—	0.25
α	0°	—	8°

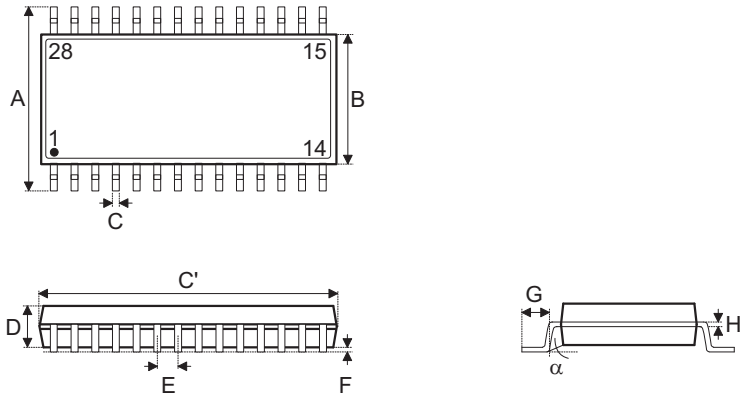
24-pin SSOP (150mil) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.228	—	0.244
B	0.150	—	0.157
C	0.008	—	0.012
C′	0.335	—	0.346
D	0.054	—	0.060
E	—	0.025	—
F	0.004	—	0.010
G	0.022	—	0.028
H	0.007	—	0.010
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	5.79	—	6.20
B	3.81	—	3.99
C	0.20	—	0.30
C′	8.51	—	8.79
D	1.37	—	1.52
E	—	0.64	—
F	0.10	—	0.25
G	0.56	—	0.71
H	0.18	—	0.25
α	0°	—	8°

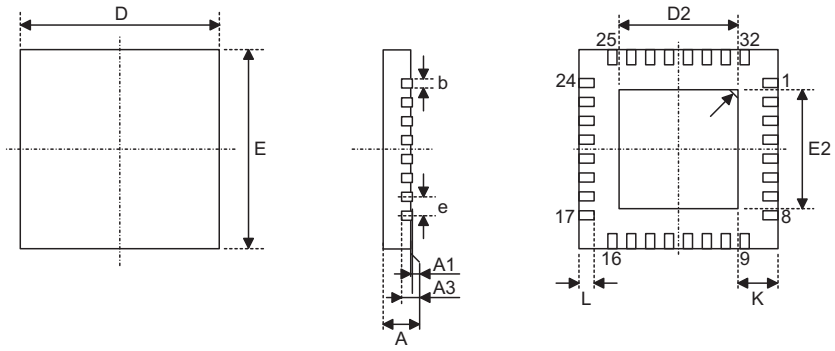
28-pin SSOP (150mil) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.228	—	0.244
B	0.150	—	0.157
C	0.008	—	0.012
C′	0.386	—	0.394
D	0.054	—	0.060
E	—	0.025	—
F	0.004	—	0.010
G	0.022	—	0.028
H	0.007	—	0.010
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	5.79	—	6.20
B	3.81	—	3.99
C	0.20	—	0.30
C′	9.80	—	10.01
D	1.37	—	1.52
E	—	0.64	—
F	0.10	—	0.25
G	0.56	—	0.71
H	0.18	—	0.25
α	0°	—	8°

SAW Type 32-pin (5mm×5mm) QFN Outline Dimensions

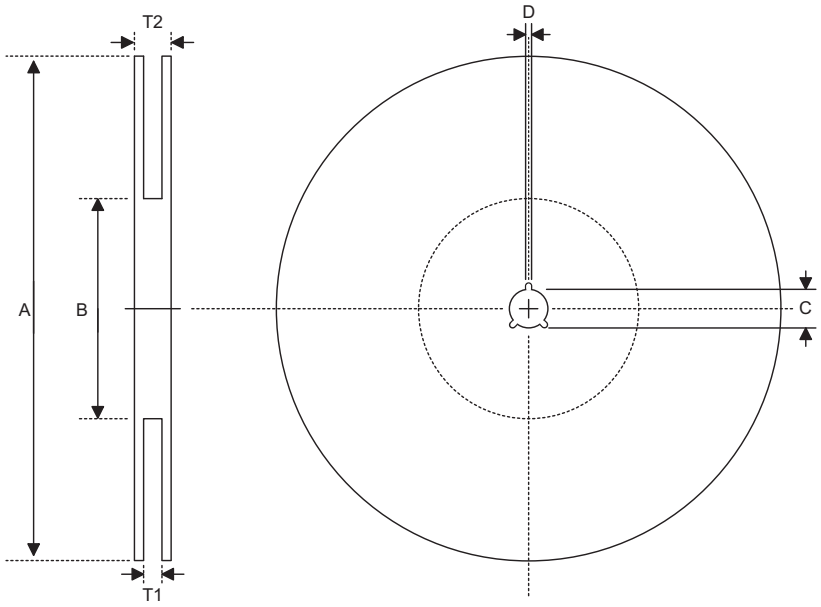


Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.028	—	0.031
A1	0.000	—	0.002
A3	—	0.008	—
b	0.007	—	0.012
D	—	0.197	—
E	—	0.197	—
e	—	0.020	—
D2	0.049	—	0.128
E2	0.049	—	0.128
L	0.012	—	0.020
K	—	—	—

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	0.70	—	0.80
A1	0.00	—	0.05
A3	—	0.20	—
b	0.18	—	0.30
D	—	5.00	—
E	—	5.00	—
e	—	0.50	—
D2	1.25	—	3.25
E2	1.25	—	3.25
L	0.30	—	0.50
K	—	—	—

Product Tape and Reel Specifications

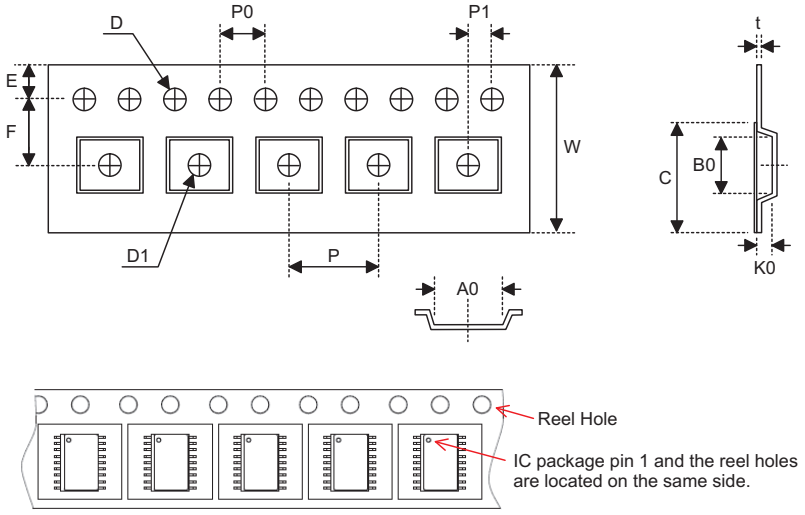
Reel Dimensions



SSOP 20S (150mil), SSOP 24S (150mil), SSOP 28S (150mil)

Symbol	Description	Dimensions in mm
A	Reel Outer Diameter	330.0±1.0
B	Reel Inner Diameter	100.0±1.5
C	Spindle Hole Diameter	$13.0^{+0.5/-0.2}$
D	Key Slit Width	2.0±0.5
T1	Space Between Flange	$16.8^{+0.3/-0.2}$
T2	Reel Thickness	22.2±0.2

Carrier Tape Dimensions



SSOP 20S (150mil)

Symbol	Description	Dimensions in mm
W	Carrier Tape Width	16.0 $^{+0.3/-0.1}$
P	Cavity Pitch	8.0±0.1
E	Perforation Position	1.75±0.10
F	Cavity to Perforation (Width Direction)	7.5±0.1
D	Perforation Diameter	1.5 $^{+0.1/-0.0}$
D1	Cavity Hole Diameter	1.50 $^{+0.25/-0.00}$
P0	Perforation Pitch	4.0±0.1
P1	Cavity to Perforation (Length Direction)	2.0±0.1
A0	Cavity Length	6.5±0.1
B0	Cavity Width	9.0±0.1
K0	Cavity Depth	2.3±0.1
t	Carrier Tape Thickness	0.30±0.05
C	Cover Tape Width	13.3±0.1

SSOP 24S (150mil)

Symbol	Description	Dimensions in mm
W	Carrier Tape Width	16.0 $^{+0.3/-0.1}$
P	Cavity Pitch	8.0±0.1
E	Perforation Position	1.75±0.10
F	Cavity to Perforation (Width Direction)	7.5±0.1
D	Perforation Diameter	1.5 $^{+0.1/-0.0}$
D1	Cavity Hole Diameter	1.50 $^{+0.25/-0.00}$
P0	Perforation Pitch	4.0±0.1
P1	Cavity to Perforation (Length Direction)	2.0±0.1
A0	Cavity Length	6.5±0.1
B0	Cavity Width	9.5±0.1
K0	Cavity Depth	2.1±0.1
t	Carrier Tape Thickness	0.30±0.05
C	Cover Tape Width	13.3±0.1

SSOP 28S (150mil)

Symbol	Description	Dimensions in mm
W	Carrier Tape Width	16.0±0.3
P	Cavity Pitch	8.0±0.1
E	Perforation Position	1.75±0.1
F	Cavity to Perforation (Width Direction)	7.5±0.1
D	Perforation Diameter	1.55 $^{+0.10/-0.00}$
D1	Cavity Hole Diameter	1.50 $^{+0.25/-0.00}$
P0	Perforation Pitch	4.0±0.1
P1	Cavity to Perforation (Length Direction)	2.0±0.1
A0	Cavity Length	6.5±0.1
B0	Cavity Width	10.3±0.1
K0	Cavity Depth	2.1±0.1
t	Carrier Tape Thickness	0.30±0.05
C	Cover Tape Width	13.3±0.1

Holtek Semiconductor Inc. (Headquarters)
No.3, Creation Rd. II, Science Park, Hsinchu, Taiwan
Tel: 886-3-563-1999
Fax: 886-3-563-1189
http://www.holtek.com.tw

Holtek Semiconductor Inc. (Taipei Sales Office)
4F-2, No. 3-2, YuanQu St., Nankang Software Park, Taipei 115, Taiwan
Tel: 886-2-2655-7070
Fax: 886-2-2655-7373
Fax: 886-2-2655-7383 (International sales hotline)

Holtek Semiconductor Inc. (Shenzhen Sales Office)
5F, Unit A, Productivity Building, No.5 Gaoxin M 2nd Road, Nanshan District, Shenzhen, China 518057
Tel: 86-755-8616-9908, 86-755-8616-9308
Fax: 86-755-8616-9722

Holtek Semiconductor (USA), Inc. (North America Sales Office)
46729 Fremont Blvd., Fremont, CA 94538, USA
Tel: 1-510-252-9880
Fax: 1-510-252-9885
http://www.holtek.com