Keywords: DS2790, DS2792, Random Number Generator, RNG

Sep 14, 2006

APPLICATION NOTE 3922

Generating a Random Number with the DS2790

Abstract: This application note describes how to use the lowest 4 bits of the DS2790's temperature register to generate a 16-bit random value. This note also provides example C code for generating a 16-bit random value.

Introduction

Random numbers are used in many encryption and authentication schemes. Generation of a random number involves finding a random behavior and using this behavior to generate a random value. Consequently, generating a random number with a microcontroller can be a difficult task, as microcontrollers frequently exhibit repetitive behavior.

Fortunately, the DS2790 allows designers to use the lowest 4 bits of the temperature register to generate a 16-bit random value. Although only the highest 11 bits of the temperature register are defined as temperature information in the Data Sheet, all 16 bits are reported. This functionality, along with the fact that the lowest 4 bits of the register are highly random, makes them ideal for generating a 16-bit random value.

Is the Extended Temperature Data Random?

Table 1 was taken by submersing a DS2790 in a controlled temperature bath. Placing the part in such a tightly controlled temperature environment quickly exposes nonrandom temperature values. The data below shows the lowest 4 bits of the address in memory where temperature is stored.

Table 1. Extended Temperature Data for the DS2790

Value	Count	Delta-Counts	Delta-Avg
0	23657	-453.625	-1.8814%
1	23822	-288.625	-1.1971%
2	24422	311.375	1.2914%
3	24091	-19.625	-0.0814%
4	24222	111.375	0.4619%
5	23994	-116.625	-0.4837%
6	24218	107.375	0.4453%
7	24258	147.375	0.6112%
8	24612	501.375	2.0795%
9	23984	-126.625	-0.5252%
A	23974	-136.625	-0.5667%
B	24005	-105.625	-0.4381%
C	24178	67.375	0.2794%
D	24066	-44.625	-0.1851%
E	23954	-156.625	-0.6496%
F	24313	202.375	0.8394%
Total Counts:	385770		
1/16th of Total:	24110.625		

The Value column shows the value of the lowest 4 bits of the temperature register in hexadecimal. The Count column shows the number of times each value occurred in approximately 24 hours. In a perfectly random system, given infinite time to monitor the DS2790's behavior, we would see each value occur 1/16th of the time. The Delta-Counts column shows the absolute error in the number of times each value occurred compared to the ideal 1/16th value. The Delta-Avg column shows the same error as a percentage of the ideal 1/16th value—that is, the percentage difference between the ideal random behavior and the value occurring 1/16th of the time. The very low percentage values demonstrate that the lowest 4 bits of the temperature register are highly rando

Generating a 16-Bit Random Number

To ensure that a random number is available when required, the example updates the random value every time a temperature-conversion result is completed. To generate a 16-bit random number, use the following steps.

1. Configure the DS2790 to generate an interrupt when a temperature conversion completes.
2. Note when the temperature interrupt occurs.
3. When the interrupt occurs, shift the lowest 4 bits of the temperature register into a variable.

These steps result in the DS2790 generating a temperature-conversion interrupt every 220ms (nominally). Since each temperature conversion provides 4 bits, and we are generating a 16-bit number, a completely new random number is available every 880ms (nominally).

Example C Code

The example C code that follows generates a 16-bit random value, which is available to the user as the global variable rand_num. Note that the part must be initialized to enable interrupts on completion of a temperature interrupt. The temperature interrupt is defined as EINT_ti in the code.

```
unsigned short rand_num;                /* Global random number */

/* This is the main Interrupt Service Routine*/
void FuelGaugeISR(void) __interrupt
{
char temp_low4bits;                     /* Store the low 4 bits of temperature here */

        /* Stay in the loop if there is an interrupt. */
        while (IIR & IMR_IM0 || IIR & IMR_IM1)
        {
                /* The IIR signifies if a Module 0 or Module 1 interrupt has occured.*/
                /* Module 0 will be treated with priority. */

                if (IIR & IMR_IM0) /* Module 0 Interrupt */
                {
                        /* Detect and Service Higher Priority Module 0 Interrupts here. */

                        if (EINT & EINT_ti) /* A temperature conversion completion int occurs every 220ms. */
                        {
                                EINT &= ~EINT_ti; /* Reset the temperature interrupt bit. */

                                /* This code builds a new 16 bit random number every 4 temperature interrupts */
                                /* The global variable rand_num is fully updated every 880ms */
                                /* Next statement clears all but low 4 bits of the temperature register */
                                temp_low4bits = (char)((pADC->TEMPERATURE) & 0x000F);

                                /* Next statement shifts the old random number left by 4 bits. */
                                rand_num = rand_num << 4;

                                /* Next statement "shifts" the new bits in by adding them. */
                                rand_num = rand_num + temp_low4bits;
                        }
                }
                else /* This is a module 1 interrupt. */
                {
                        /* Handle module 1 interrupts here. */
                }
        }
}
```

More Information
For technical questions and support: http://www.maxim-ic.com/support
For samples: http://www.maxim-ic.com/samples
Other questions and comments: http://www.maxim-ic.com/contact

Related Parts

DS2790: QuickView -- Full (PDF) Data Sheet

DS2792: QuickView -- Full (PDF) Data Sheet

AN3922, AN 3922, APP3922, Appnote3922, Appnote 3922