elektrorevue
ISSN 1213 - 1539

Random Number Generator in MSP430 x5xx Families

Radek Fujdiak, Jiří Mišurec, Petr Mlýnek, Ondřej Rášo

Department of Telecommunications
Faculty of Electrical Engineering and Communication
Brno University of Technology
Email: fujdiak@phd.feec.vutbr.cz, misurec@feec.vutbr.cz, mlynek@feec.vutbr.cz, raso@feec.vutbr.cz

Abstract *– This article describes a method for generating random numbers on low-power microcontroller MSP430x5xx Families from Texas Instruments Company. This generator can be used to generate numbers for encryption or decryption methods in cryptography (for example in the cryptosystem Diffie-Hellman). This article provides a short theoretical introduction, some practical examples with used modules of MSP430 and in the last chapter a description how to create a random number generator for this microcontroller and some tips how to grow the randomness.*

1 Introduction

Random number generators (RNG) are today used in everyday life in banking, transferring data or in general in security systems (cryptosystems). Two main types of number generating are existing, software and hardware. Software generating is mainly faster with smaller randomness, that is mean only pseudo-random number line and only algorithm needs. Hardware generating is slower with higher randomness, which is mean truly random number line with special modules needs for creating random process [1].

This article is focused on random number generators for low-power devices, concretely the microcontroller MSP430x5xxx from Texas Instruments Company. It is used in many devices, as it has good energy properties like saving battery in portable devices [2].

The article concisely describes the work with the microcontroller and explains how to create the random generator. First one type of generator is chosen from all possibilities and in the next part its basic parts, properties and functionality are described.

2 RNG in MSP430

It exists two basic dividing for the random number generators. The first dividing looks where is created the randomness and using terms the software and the hardware random generators. The software generators use mostly mathematical model, which is able to generate random numbers. The hardware generators use some physical event (noise in space etc.) for generating random numbers. The second dividing looks if the generated sequence is truly random or not and using terms the truly-random generator and the pseudo-random generator. The truly-random generator is truly random, which means is not possible to find a sequence, which is repeated in loop independent to the time. The pseudo-random generator is not truly random, which means they generate some sequence of bits (bigger or smaller, depend on the generator) which is always repeated. From the description of software generator is evident that this is the pseudo-random generators. It is coming from the fact that it is using the mathematical model. Some specific methods exist for improving the randomness of these generators, for example to use better seeds, generate new seeds in specific rounds etc. But also with these improvements the generator is still only pseudo-random generator. The hardware random generator can be on the other hand truly random generator but it is not necessary. It is important in hardware generating control the real event (on which the generator is based) if does not change its attributes and does not become in some easily predictable event. If all necessary conditions are met, the hardware random generator can be called as truly-random generator.

From the description of the generators also the requirements and the attributes of these generators are coming. The software generator, if it has enough computing capacity, is faster with lower randomness (pseudo-random) and the hardware generator, if they using good random event, having better randomness and it does not need special computing capacity.

The random number generator for MSP430 can be created with the basic function `rand()`, which is included in the basic library `stdlib`. This solution can be used for some basic examples of generating numbers or some beta-testing, but for practical devices the randomness of this function is very insufficient.

It can be also used many software generators from various different creators, but this generators will every time generate only pseudo-random combinations of numbers [3]. The second problem for the using the software generators in MSP430 is special computing needs. The MSP430 is the ultra-low-power microcontroller and it is evident that needs to save as much as possible from the performance. That is the reasons for choosing some hardware solution. The hardware generators are using truly random events, which can generate truly random numbers. If the random generator is used in practical devices, then it needs to be counted also with the final prize of the product. Each external module, which will be extra added, will increase the final prize of this product. That is reason for using an internal module.

elektrorevue
ISSN 1213 - 1539

2.1 Modules used for RNG

This is a basic introduction to the modules, which are used for our Random Number Generator. In this part will be showed the work with them and described how to set them.

2.1.1 Diode and Watchdog

It is possible to work only with one green diode in MSP430x5xxx. This diode can be used as a control device, for example with some conditions, where the diode will blink in every case with a different frequency. In Code 1 the basic control of the diode is showed. For blinking is necessary to put the control orders in some loop function [4].

```
P1DIR |= 0x41; // Set the diode PIN
P1OUT = 1; // 1 for ON, 0 for OFF diode
P1OUT ^= 0x41; // negotion of diode PIN
```

Code 1: Diode control

Watch dog is an automatically reset system for timers. In most situations it is better to have them under personal control, and then it is better to set the watch dog off (Code 2) [5].

```
WDTCTL = WDTPW + WDTHOLD; // Stop WDT
```

Code 2: Stop watch-dog timer

2.1.2 Clock Modules

MSP430 has three basic clock modules (Figure 1). Two of them can be used as a signal for Timers [6].

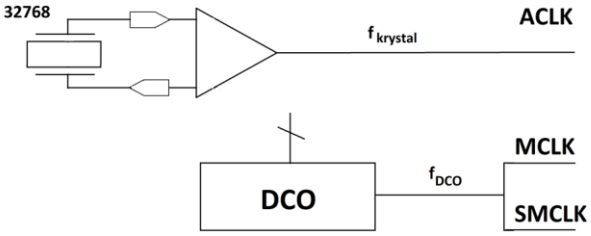

Figure 1: Clock system in MSP430

The first SMCLK (subsystem master clock) is using a digital controlled oscillator (DCO), which in basic runs on a frequency around 1 MHz. The DCO can be also resetted to 1, 8, 12 and 16 MHz. To change the frequency of the clock module registers `BCSCTL` (basic clock system control registers) and `DCOCTL` should be setted (Code 3 showing how to set a 1MHz).

```
BCSCTL1 = CALBC1_1MHZ;// Set range
DCOCTL = CALDCO_1MHZ; // DCO step,modul.
```

Code 3: Setting 1 MHz in DCO

The second clock is ACLK (auxiliary clock). This clock is running on a much lower frequency oscillator - VLO (in basic 12 KHz). Similar to the DCO the frequency of the VLO can be resetted, but in this case into any other value. It can be also setted in the register `BCSCTL1`, but with a different variable. All of the registers can be found in the library `MSP430f5438`.

2.1.3 Timers and Interrupts

Two different timers `TIMER_A` and `TIMER_B` are existing. Both can be used. The differences between them are only question about queue priority of some register [7], what is really irrelevant. `TIMER_A` (Figure 2) will be described because in the RNG this one is used.

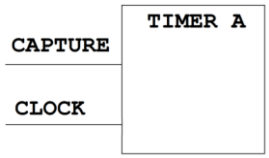

Figure 2: TIMER A

Timer_A it is a 16-bit counter, having 4 modes of operation (stop, up, continuous and up/down), 3 capture/compare registers (`CCR1 – CCR4`) and 2 interrupts vectors (`TACCR0` and `TAIV`).

The stop mode is counting to the value and when the last value is reached, then the interrupt flag is setted (if it is enabled). Up mode doing a similar operation, but after reaching the last value and setting the interrupt flag, it is resetting all values and the counting process is starting again from zero. Up/Down mode is doing the same like up mode, but it is counting to the last value and back to zero. The interrupt flag is after that setted, all the values are resetted and the process starts to count again from zero. The continuous mode is showed on the Figure 3. On the Figure the last value FFFFh (0xFFFF) and the same frequency in the timer clock FFFFh are setted.

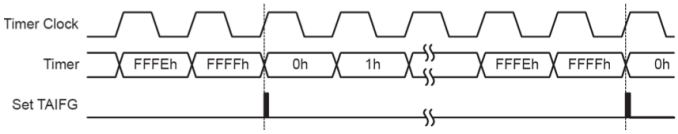

Figure 3: Interrupt flag with continuous mode [8]

The timers are having three key parts. First is the clock input, which has its own source (ticks at a specified rate). Second the counter, which has a specified mode and interrupt procedure, which is called when it reaches the last value.

For a better understanding of how to set a timer, it is possible to look in the library `MSP430f5438`, where the basic registers of `Timer_A` can be found.

For example when the count limit should be setted to 12000 (`TA0CCR0`), with enabled interrupt flag (`TA0CCTL0`) and sources (`TA0CTL`) ACLK with 12 KHz (`TASSEL_1 + MC_2`), the Code 4 should be followed.

```
TA0CCR0 = 12000; // count lim.,used CCR0
TA0CCTL0 = 0x10; // enable interrupts
TA0CTL = TASSEL_1 + MC_2;
```

Code 4: Settings for Timer_A0

The Timers of MSP430 are very complicated and it is no space to show all possibilities. This could be a topic for a new article.

elektrorevue
ISSN 1213 - 1539

3 Hardware RNG using Timers

The random number generator for MSP430, which will be described, is using two clocks (SMCLK, ACLK) and one timer (TIMER_A and two of threads of it). The clocks SMCLK and ACLK are two independent clocks and having each own timing source. This fact can be used for generating numbers if it is done right.

ACLK running in basic on 12 KHz, the SMCLK clock running in basic on 1 MHz (in real the frequency circling around 0.9 to 1.1 MHz). This mean one of the clocks is ticking slower and one faster, then when the slower (ACLK) will have first tick the faster (SMCLK) will have n-ticks (Figure 4).

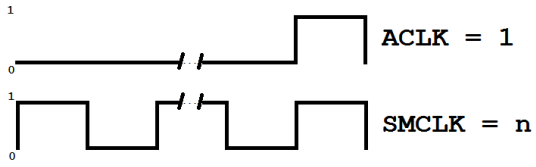

Figure 4: Comparison of ACLK and SMCLK

The number `n` (ticks of DCO) will be always little bit different (even or odd), but always unpredictable also when the previous state is known. This fact is guaranteed because the frequency of the DCO is not static and circling around 1 MHz and also with fact the both clock are independent with own sources. This timing differences between these two clock systems can be used as a source for generating a random number sequences [9].

For the odd or even number of the DCO ticks is used the LSB (least significant bit). If this LSB from the number `n` is used, one single random bit will be created. This means, that if this operation is done in loop, it can be easily generate a sequence of random bits. The basic scheme of functionality for this type of generating is showed in the Figure 5 and the following text will be focused to the implementation important parts (and code part).

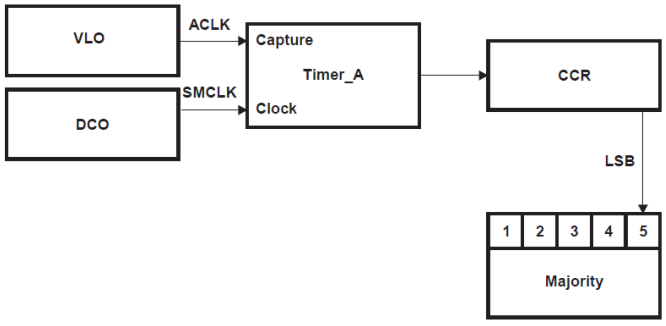

Figure 5: RNG in MSP430 using timers

It will be used the Timer_A, two clocks, CCR register for load a LSB and left-shifter register for saving random numbers (single LSBs). First it is necessary to set an input of `Timer_A` (concretely two threads of `Timer_A`), first `Timer_A0` for VLO (Code 6) and second `Timer_A1` for DCO (Code 5).

```
TA0CCTL0 = CAP | CM_1 | CCSIS_1;
TA0CTL = TASSEL_2 | MC_2;
```

Code 5: Setting DCO

`TA0` shows to which thread of `Timer_A` belongs the register. `CCTL0` register is choosing the mode of the counter and `CTL` register is choosing the clock source.

```
TA1CCR0 = 1200;
TA1CCTL0 = CCIE | OUTMOD_3;
TA1CTL = TASSEL_1 | MC_1;
```

Code 6: Setting VLO

`TA1`, `CCTL` and `CTL` do the same as in the case before. Register `CCR0` is setting the maximum value of the counter. This is necessary because this thread will count the ticks of ACLK.

In the Figure 6 some little differences from Figure 5 are showed. These differences will be closer described in the following.

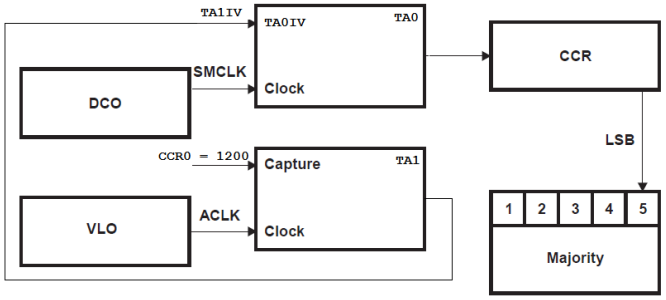

Figure 6: Real schematic of RNG

As it was showed, two threads of the Timer_A (TA0, TA1) are used. For the TA0 the DCO is used and for the TA1 the VLO. The TA1 is a counter, which is counting (in up mode) ticks of the VLO, till 1200 ticks are reached (smaller number accelerate generating process, but the VLO will have less time to stabilize, bigger number slows generating process, but is giving some time for the VLO stabilizing). When the TA1IV (Timer_A1 interrupt vector) is setted to one, then in the same time TA0 and the TA1 should be paused from any action (Code 7).

```
#pragma vector=TIMER1_A0_VECTOR
  __interrupt void Timer1_A0 (void) {
// TA1IV interrupt service routine
  TA0CTL = MC_0;  // Pause TA0
  TA1CTL = MC_0;  // Pause TA1
```

Code 7: Interrupt vector of Timer1_A0

All action should be paused and all registers resetted, but before is necessary to read register of CCR for LSB and put it into a left-shifted register (Code 8).

The left shifted register `rn` is created with number `n` (number of bits in `rn`) and `rn` is the random number. The saving process started from `n` (MSB of `rn`) and is going to zero (LSB of `rn`). This process simulates a left shifted register.

elektrorevue
ISSN 1213 - 1539

```
A0 = TA0R; // save number of ticks SMCLK
TA0R = 0; // reset cnt. of SMCLK ticks
if (A0 & (1 << 0)) {// comp. if LSB is 1
 rn |= (1 << n);
 n--}
// if yes save LSB = 1 to left-shifted
// register rn
 else {rn |= (0 << n)
       n--};
// if not save LSB = 0 to left-shifted
// register rn
```

Code 8: Saving randomness

It is good to use the diode for controlling and put in the program some condition for protecting the memories area, choose the appropriate `CCR0`, choose the number of bits `n` and some other small things, but in general this idea of random number generator is functionally and ready for use in practical products. Before it will be used, it is necessary to put this generator in some analytic process, what will show if this generator is creating sufficient randomized numbers. In the next subchapter it is described how it is possible to put in more randomize environment.

3.1 Possibilities for growing random factor of RNG

In this subchapter will be given some tips for increasing the random factor of the generator. In the Figure 7 some additional modules/register are showed, which can be used for increasing the randomness.

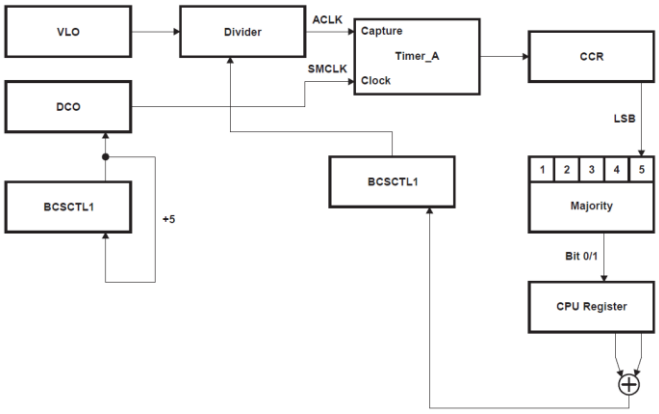

Figure 7: Additional modules for RNG [9]

The register BCSCTL1 as it was showed in Chapter 2.1.2 is used for setting the speed of the DCO/VLO. Number five can be used for change the speed of clocks every time when the LSB is shifted (any other number can be used too, but five shows the biggest differences between DCO/VLO clock steps).

The next thing what might be done is changing `n` (number of ticks). It could be static or random change in every round, which generates the LSB bit, but in this case is necessary to be careful, because this number even affects the speed of the number generating process.

4 Conclusion

After a theoretical introduction, the work with basic modules of MSP430 is showed. The biggest part of this article deals with how to create a hardware random number generator for MSP430 devices. It starts with the theoretical field, finishes with practical examples and gives some tips how to increase the randomness of the generator. It is possible to use this generator in practical devices, which are working even in the cryptography field. But the article did not deal with the analytical part and it is necessary to do some analytical tests before it will be taken from laboratory environment.

Acknowledge

Thanks to the Project TAČR 02020856 – Application research of intelligent systems for monitoring energy networks and to the University of Technology Brno.

References

[1] TURNER, Noah. *Software vs. Hardware RNG's*. 2005. In: [online]. [cit. 2013-12-15]. Available from: http://www.tstglobal.com/assets/downloads/1268986797a16.pdf

[2] Texas Instruments. *MSP430 Ultra-Low-Power Microcontroller*. 2008. In: [online]. [cit. 2013-12-15]. Available from: http://www.ti.com/lit/sg/slab034w/slab034w.pdf

[3] PRESS, William H., Saul A. TEUKOLSKY, William T. VETTERLING a Brian P. FLANNERY. *Numerical Recipes in C: The Art of Scientific Computing*. Second Edition. New York: Cambridge University Press, 1992. ISBN 0-521-43108-5.

[4] LITOVSKY, Gustav. *Beginning Microcontrollers with the MSP430*. 2010. In: [online]. [cit. 2013-12-15]. Available from: http://www.glitovsky.com/Tutorialv0_2.pdf

[5] Recursive Labs. *Programming the watchdog timer*. 2011. In: [online]. [cit. 2013-12-15]. Available from: http://recursive-labs.com/static/courses/rl100/samples/watchdog.pdf

[6] WISMAN, Ray. *MSP430 Timers and PWM*. 2012. In: [online]. [cit. 2013-12-15]. Available from: http://homepages.ius.edu/RWISMAN/C335/HTML/msp430Timer.HTM

[7] QUIRING, Keith. *MSP430 Timers In-Depth*. 2006. In: [online]. [cit. 2013-12-15]. Available from: http://www.ti.com/lit/ml/slap113/slap113.pdf

[8] WANG, Yin. *MSP430 Clock System and Timer*. 2007. In: [online]. [cit. 2013-12-15]. Available from: http://www.ccs.neu.edu/home/noubir/Courses/CSU610/S07/MSP430-Clock-Timers.pdf

elektrorevue
ISSN 1213 - 1539

[9] Texas Instruments. *Random Number Generation Using the MSP430*. 2006. In: [online]. [cit. 2013-12-15]. Available from: http://www.ti.com/lit/an/slaa338/slaa338.pdf