Home

RTCC Interrupts

What is an Interrupt?

An interrupt is a defined condition that causes the microcontroller (MCU) to suspend what it is doing and go to a known memory location to execute a special block of code. The condition could be that a certain time has elapsed (Timed Interval) or a port (or ports) has changed state (Port Trigger Interrupt). Interrupts are used in computers to ensure that a specific task is done a certain number of times or that an alarm condition is handled IMMEDIATELY.

This chapter covers Timed Interrupts. For a discussion of interrupts that are generated based on an external event (i.e. an alarm is tripped), please see the chapter on Port Trigger Interrupts.

Overview

How fast is your car traveling? The standard way to determine that is to measure the distance traveled by some time constant. Thus, we state that a car is traveling "thirty miles per hour".

To a computer, time means nothing, it only knows clock cycles. You can fool a computer into thinking time is traveling faster or slower by simply changing the clock speed. Accurate time keeping is done by attaching a crystal or resonator to a chip and having the chip count the number of cycles. Yes, you can connect a resistor and a capacitor together and estimate elapsed time but it will not be nearly as accurate as a crystal or resonator. In fact, some microcontrollers in projects that are not time critical, use an internal RC circuit to estimate the passage of time.

The goal for this chapter is to demonstrate how to create an interrupt on the microcontroller that will accurately blink an LED once per second. Not a rocket science type project by my philosophy is to give you small programs that can be used as a shell around your larger programs.

Demonstration programs typically involve WAY too much code and forget that most programmers do not read a book from start to finish but rather JUMP INTO a chapter that they have some interest in. (yea, admit that you opened this chapter before you read the previous material!). In light of that (pun intented), I will demonstrate how to blink an LED. If you can get that working, they you can add your complex code later.

Blink an LED

- Let's start with a simple program that just blinks an LED using a 4mHz resonator. Connect RC.0 to a 220 ohm resistor thence to an LED thence to ground. The following program will blink an LED about once a second which is pretty easy to see.

```
DEVICE          SX28, OSC4MHZ, TURBO, STACKX, OPTIONX
FREQ            4_000_000 ' Identify the frequency of the Resonator
```

```
PROGRAM Start
Start:
   tris_C=%00000000    ' Set port direction (0=output, 1=input)
Main:
   RC = NOT RC
   pause 500
   goto main
```

Using an RTCC

Most programs start with an initialization routine and then go into some sort of main loop. The main loop controls the execution of the program and by way of IF statements and subroutines, carries on its normal work. Occationally, however you need a program that will do something IMMEDIATELY upon an outside event (i.e. an alarm is tripped) or PRECISELY on a schedule (i.e. so many times per second, guaranteed). That trick is implemented in something called an "interrupt".

In the SX processor, the interrupt is implemented with either the Real Time Clock Counter (RTCC) or the Watchdog Timer. In this chapter we will cover the RTCC.

The RTCC monitors the number of clock "cycles". With a 4mHz resonator, that means that one clock cycle happens every 0.0000002 seconds. When the RTCC is running, it increments once for each clock cycle. The RTCC register can only hold a number up to 255 and then it overflows to zero again. So when the RTCC overflows, it triggers an interrupt. Even at that rate the LED would blink over 15,000 times a second. We can only see an LED "blink" when it is slower than 50 times a second. This means we need to either slow down the processor or "scale" the timing. We are going to do the latter by using a technique called "pre-scaling". This is done using the OPTION register.

To get started, we want to tell the compiler that we want to use the RTCC. On page 174 of the SX-Key/Blitz Development System Manual 2.0 is a chart showing the various things the OPTION register controls, one of them is the RTCC.

Option Register							
7	6	5	4	3	2	1	0
RTW	RTI	RTS	RTE	PSA	**PS2**	**PS1**	**PS0**

For our purposes we want to set the RTW bit which is the left most bit. If this bit is a 0 then it means we want to use the Watchdog Timer. If it is a "1" then we want to use the RTCC. Here we are telling the compiler that we want to use the RTCC.

```
OPTION = %10000000  ' or we can use OPTION=$80
```

Pre-Scaling

Ok, some quick math. Let's say we tell the MCU that we want to blink the LED once every clock cycle. Using a 4mHz resonator, it would blink 4,000,000 times per second! We need to get it down so that we can even see a glimmer of a blink! If we turn on the RTCC, it will count up to 255 and then

overflow (go back to zero) causing the interrupt to fire. That brings it down to about 15,000 times a second. Next we can "scale" the RTCC so that it after a certain number of times of overflowing it will fire the interrupt. For example, we can tell the MCU that we want to fire the interrupt only after it overflows 128 times. Now let's do the math again.

```
4,000,000 resonator (4mHz)

The RTCC increments on each clock cycle.  When it
overflows (goes from 255 to 00) it triggers an interrupt.

4,000,000 resonator freq / 256 clock cycles = 15625
(an interrupt is triggered this many times per second)

Still too fast. Let's try dividing (scalling) that by 128).

15625 / 128 = 122.07 (thus, about 122 times a second)
```

The maximum blink rate you can discern is about 50 times a second. So even this is a little fast. Let's go one more step and use the scale of 256 instead of 128.

```
15625 / 256 = 61.035
```

This is close enough to try. Remember in our experiment at the beginning of this chapter, we blinked the LED once a second. We are going to try it at sixty times faster than that!

To set the "pre-scaller" we need to set bits PS2, PS1 and PS0 in our OPTION register. These are the right most bits in the OPTION register and we set them at the same time we tell the compiler we want to use the RTCC timer. We need to use a value of %10000111 meaning to use the RTCC (left most bit) and a maximum value for the pre-scaller (the right most bits). We do this by using the command:

```
OPTION = %10000111  ' Set RTCC with 1:256 pre-scaller
```

Sometimes programmers use hexadecimal and it would look like:

```
OPTION = $87
```

Blink an LED using an Interrupt

Now let's create the program. Most of the action happens after the FREQ command and before the PROGRAM START.

- First we add a command called IRC_CAL. This calibrates the internal RC Oscillator to the external crystal or resonator on your board and helps you to calcalate precise interrupt timing. If you don't do this, the interrupt will not be in perfect sync with the resonator.

```
IRC_CAL   IRC_SLOW
```

- Next we add the INTERRUPT code. Here we simply set the LED to the opposite of what it was. If it was on, we turn it off, if it was off, we turn it on.

```
INTERRUPT
   ' toggle the LED on or off
   RC = NOT RC
RETURNINT
```

- Inside our main program, we set the OPTION register to %10000111 (or $87) to indicate that we want to use the RTCC and that we want to scale it to 1:256.
- Finally, we do not need anything in the main loop because the toggling is going to be done in the interrupt.
- The code is below and will toggle the LED about 61 times per second. I don't know about you but the LED is blinking faster that I can recognize. This is due to a phenomenon called "Persistence of Vision" which basically states that when something happens too fast it is invisible (a very un-scientific description, but it gets the idea across). Thus, type in the program and debug it. Remember, we still have a little ways to go.

```
DEVICE          SX28, OSC4MHZ, TURBO, STACKX, OPTIONX
FREQ            4_000_000 ' Identify the frequency of the Resonator
IRC_CAL         IRC_SLOW  ' Calibrate internal RC Oscilator to Resonator

INTERRUPT
ISR_Start:
   ' This interrupt is called once for every 256*256 clock cycles
   ' or about 61 times a second and still too fast to see
   RC = NOT RC
ISR_Exit:
   RETURNINT ' {cycles}

Program Start
Start:
   ' initialization code here
   tris_C=%00000000    ' Set port direction (0=output, 1=input)
   option = %10000111  ' Set the RTCC scale to 1:256
   rc = $FF            ' Set an initial value to RC
Main:
   ' Later we will put additional code here
   ' For now we just want to loop forever
   goto main
```

Slow it Down More!

We have already set the pre-scaller at its maximum value of 256 so what do we do now? Well, in the interrupt we can add code that will only act on every third call. Set up a BYTE variable called MyCounter. Change the interrupt code as follows and in the initialization part of your program, set MyCounter equal to zero.

```
DEVICE          SX28, OSC4MHZ, TURBO, STACKX, OPTIONX
FREQ            4_000_000 ' Identify the frequency of the Resonator
IRC_CAL         IRC_SLOW  ' Calibrate internal RC Oscilator to Resonator

MyCounter VAR BYTE

INTERRUPT
 ISR_Start:
```

```
   ' This interrupt is called once for every 256*256 clock cycles
   MyCounter = MyCounter + 1
   IF MyCounter = 3 THEN
      MyCounter = 0
      RC = NOT RC
   ENDIF
ISR_Exit:
   RETURNINT ' {cycles}

Program Start
Start:
   ' initialization code here
   tris_C=%00000000    ' Set port direction (0=output, 1=input)
   option = %10000111  ' Set the RTCC scale to 1:256
   rc = $FF            ' Set an initial value to RC
   MyCounter = 0       ' Initialize to zero
Main:
   ' Later we will put additional code here
   ' For now we just want to loop forever
   goto main
```

Ah! NOW we can see the blink. With a little experimenting and using a count of 36 I was able to blink the LED about once per second. To get it to blink at EXACTLY once per second would involve some more exprimentation. In the next chapter, we will turn an LED on or off when some external event happens like an alarm being tripped.

Other Values for the OPTION register

- RTW - as stated above, 0=WatchDog Timer, 1=RTTC
- RTI - 0=rollover is enabled (which we want), 1=disable rollover
- RTS - 0=increment on instruction cycle (which we want), 1=increment on RTCC pin on MCU
- RTE - 0=increment on low-to-high transition, 1=increment on high-to-low
- PSA - 0=prescaler is assigned to RTCC, 1=prescaler assigned to Watchdog Timer
- PS2 - these last three bits determine the prescaler as powers of 2, from 2 to 256.
- PS1
- PS0

*File: **RTCCInterrupts.mkd** updated: 04/04/2010 18:29*