(https://www.edx.org)

UTAustinX: UT.6.01x Embedded Systems - Shape the World

KarenWest (/dashboard) ▼

Courseware (/courses/UTAustinX/UT.6.01x/1T2014/courseware) **Course Info (/courses/UTAustinX/UT.6.01x/1T2014/info)**

Discussion (/courses/UTAustinX/UT.6.01x/1T2014/discussion/forum) **Progress (/courses/UTAustinX/UT.6.01x/1T2014/progress)**

Questions (/courses/UTAustinX/UT.6.01x/1T2014/a3da417940af4ec49a9c02b3eae3460b/)

Syllabus (/courses/UTAustinX/UT.6.01x/1T2014/a827a8b3cc204927b6efaa49580170d1/)

Embedded Systems Community (/courses/UTAustinX/UT.6.01x/1T2014/e3df91316c544d3e8e21944fde3ed46c/)

VIDEO 12.3D. PERIODICSYSTICKINTS CODE DEMO

C12 3d SysTick periodic interrrupt software YouTube

Help

6:12 / 6:12 **1.0x**

DR. RAMESH YERRABALLI: This is Ramesh Yerraballi.

Let's take a look at another interrupt, which is the SysTick interrupt.

The SysTick interrupt is used to generate periodic events.

That's one of its uses.

So we'll see an example where we want to produce a wave, a square wave if you

will, where on port F, bit 2, we will produce a square wave like this.

It's going to be on and off for one millisecond, on one millisecond, off,

and repeats this process.

So that's our problem.

So the way we're going to solve this problem

is we will periodically interrupt.

And every time we interrupt, we will toggle.

The software action will be toggled PF2.

So this is our trigger and the action is toggle PF2.

So let's take a look at the components of our program.

The initialization involves our SysTick registers, the control register,

the reload register, and the current register.

So we will set our control register to 0, disable during set-up.

We will set our reload value to a calculated period.

This period corresponds to our 1 millisecond.

And the calculation is that our clock is at 16 megahertz.

And at 16 megahertz, 1 millisecond is 16,000 times

62.5, which is 1 millisecond.

So our count is going to be 16,000.

So that's what we call it.

And as Jon indicated in the previous program, we need to set the priority.

The priority for the SysTick interrupt is

in a different register, which is the priority three register of the Nested

Vectored Interrupt Control.

And we are setting the priority to be equal to 2 by this statement here.

And the last thing we will do is we will enable SysTick

because we finished the initialization.

And then we will enable interrupts by setting the I to be equal to 0.

DR. JONATHAN VALVANO: Professor Yerraballi, this 7 is new.

Last time, we wrote a different number to it.

Last time, we wrote a 5.

Why is it a 7 now?

DR. RAMESH YERRABALLI: So that's a good question,

because previously when we enabled SysTick, we wrote a value of 5 here,

which was a 101.

Now we have a different value, which is 111 instead of a 101

which corresponds to a 7.

So this bit here is to enable interrupts.

That is, we're saying that when SysTick crosses the boundary where

it goes from 1 to a 0 transition, we want an interrupt to be flagged.

And that's a bit to say that.

DR. JONATHAN VALVANO: Oh.

DR. RAMESH YERRABALLI: OK.

So let's take a look at how the code is set up.

So first, we initialize the SysTick interrupt

by calling the SysTick_Init with this value, 16,000,

which corresponds to 1 millisecond.

We enable interrupts.

And our while loop is just sitting there waiting for the SysTick

to be fired off every time it crosses the 1 0 boundary.

And this is just a wait for interrupts so that we are in a power saving mode.

So now when the interrupt trigger does occur, we execute the SysTick_Handler.

And within the SysTick_Handler, we perform a toggle on port F, bit 2.

And we're also incrementing a counter to keep track of how many times

we did this flipping of the bit PF2.

DR. JONATHAN VALVANO: So how are we going to test it?

DR. RAMESH YERRABALLI: So let's test this by running it in our simulation.

So we're going to make sure we're in simulation mode.

So we go to our debug.

It's in the simulator mode, OK.

And I'm going to go ahead and--

DR. JONATHAN VALVANO: Build it.

DR. RAMESH YERRABALLI: --build it.

And I'm going to run it.

And we have already set it up so that the Logic

Analyzer is monitoring port F, bit 2.

And we also have a viewer here that shows

us our SysTick timer, the registers corresponding to SysTick timer.

And this is on PF2 which we can look at here

or we can look at in the Logic Analyzer.

So let's go ahead and run it.

DR. JONATHAN VALVANO: Ooh.

DR. RAMESH YERRABALLI: And we see the

SysTick causing interrupts.

And we can measure the time between our two interrupts.

And that's 5.6 to 5.690, so that's pretty close to 1 millisecond.

DR. JONATHAN VALVANO: Works.

DR. RAMESH YERRABALLI: It works.

And we can also see the LED here, but this

is a visual that doesn't tell us whether it's actually

going on and at 1 millisecond.

So we looked at how the SysTick can be used to generate a square wave.

But maybe there's some interesting applications for this.

Jon, what do you think?

DR. JONATHAN VALVANO: Yeah.

We can use this type of wave to control

Table 12.6 shows the SysTick registers used to create a periodic interrupt. SysTick has a 24-bit counter that decrements at the bus clock frequency. Let f_{BUS} be the frequency of the bus clock, and let n be the value of the **RELOAD** register. The frequency of the periodic interrupt will be

$$f_{BUS}/(n+1)$$

First, we clear the **ENABLE** bit to turn off SysTick during initialization. Second, we set the **RELOAD** register. Third, we write any value to **`NVIC_ST_CURRENT_R`** to clear the counter. Lastly, we write the desired mode to the control register,**`NVIC_ST_CTRL_R`**. We must set **CLK_SRC**=1, because **CLK_SRC**=0 external clock mode is not implemented on the TM4C microcontroller family. We set **INTEN** to enable interrupts. We need to set the **ENABLE** bit so the counter will run. In summary, we will write a 7 to the **`NVIC_ST_CTRL_R`** register.

We establish the priority of the SysTick interrupts using the TICK field in the **`NVIC_SYS_PRI3_R`** register (bits 31-29). When the **CURRENT** value counts down from 1 to 0, the **COUNT** flag is set. On the next clock, the **CURRENT** is loaded with the **RELOAD** value. In this way, the SysTick counter (**CURRENT**) is continuously decrementing. If the **RELOAD** value is n, then the SysTick counter operates at modulo n+1 (…n, n-1, n-2 … 1, 0, n, n-1, …). In other words, it rolls over every n+1 counts. Thus, the **COUNT** flag will be set every n+1 counts. Program 12.5 shows a simple example of SysTick. SysTick is the only interrupt on the TM4C that has an automatic acknowledge. Notice there is no explicit software step in the ISR to clear the **COUNT** flag.

Address	31-24	23-17	16	15-3	2	1	0	Name
$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	NVIC_ST_CTRL_R

$E000E014	0	24-bit RELOAD value	NVIC_ST_RELOAD_R
$E000E018	0	24-bit CURRENT value of SysTick counter	NVIC_ST_CURRENT_R

Address	31-29	28-24	23-21	20-8	7-5	4-0	Name
$E000ED20	TICK	0	PENDSV	0	DEBUG	0	NVIC_SYS_PRI3_R

Table 12.6. SysTick registers.

```
volatile unsigned long Counts=0;
void SysTick_Init(unsigned long period){ // priority 2
  NVIC_ST_CTRL_R = 0;         // disable SysTick during setup
  NVIC_ST_RELOAD_R = period-1;// reload value
  NVIC_ST_CURRENT_R = 0;      // any write to current clears it
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000;
  NVIC_ST_CTRL_R = 0x07; // enable SysTick with core clock and interrupts
  EnableInterrupts();
}
void SysTick_Handler(void){
  GPIO_PORTF_DATA_R ^= 0x04;       // toggle PF2
  Counts = Counts + 1;
}
int main(void){ // running at 16 MHz
  SYSCTL_RCGC2_R |= 0x00000020; // activate port F
  Counts = 0;
  GPIO_PORTF_DIR_R |= 0x04;   // make PF2 output (PF2 built-in LED)
  GPIO_PORTF_AFSEL_R &= ~0x04;// disable alt funct on PF2
  GPIO_PORTF_DEN_R |= 0x04;   // enable digital I/O on PF2
  GPIO_PORTF_PCTL_R = (GPIO_PORTF_PCTL_R&0xFFFFF0FF)+0x00000000;
  GPIO_PORTF_AMSEL_R = 0;     // disable analog functionality on PF
  SysTick_Init(16000);        // initialize SysTick timer, every 1ms
  EnableInterrupts();
  while(1){                   // interrupts every 1ms, 500 Hz flash
    WaitForInterrupt();
  }
}
```

Program 12.5 Implementation of a periodic interrupt using SysTick (C12_PeriodicSysTickInts).

Help