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/)

Help

Next, let's put the pieces together and show actual code for edge-triggered interrupts. Again, the goal is to increment the global variable **`FallingEdges`** each time the SW1 switch is pressed using falling edge-triggered interrupts on Port F.

VIDEO 12.3C EDGE-TRIGGERED INTERRUPT EXAMPLE CODE DEMO

C12 3c Edge triggered interrupts YouTube

3:52 / 3:52 1.0x

DR. JONATHAN VALVANO: All right, let's put the pieces together.

This is the startup.s.

There's the Reset Handler, which is where it begins when you hit the reset.

Down here we see that SysTick_Handler, which we'll do next.

But the one we're doing today is the PortF_Handler.

So when we touch the switch, this location

will specify what software gets executed.

That is our vector.

All right, let's look at the initialization.

We're going to activate the clock, clear our counter.

The direction register is equal to a 0 because it's an input.

We're going to enable the pin for digital signals.

We don't care about the alternate function or PCTL, or the analog mux.

We do have to have a pull up because this

is PortF bit 4 on the LaunchPad, which requires a pull up.

And here are the four bits that we saw in the last video.

IS equals 0.

IBE equals 0.

IEV equals 0.

And to arm it, we're going to set the IM, or the arm bit equal to a 1.

When we write to the ICR, it will clear the flag.

This is the code which sets the priority 7

register to set the priority for this interrupt.

And then we're going to set bit 30 of the enable register

to enable the PortF edge trigger.

And lastly, we're going to enable all interrupts by clearing the I bit.

Here's our interrupt service routine.

Help

On the trigger, which is the falling of the switch,

we're going to first acknowledge by clearing the trigger

flag by writing a 1 to the IC register.

And then we're going to increment the counter, which

will count the number of times a switch is pressed.

And then we'll return from interrupt by just simply returning.

The main program doesn't do much.

It initializes the system and then waits in this wait loop.

DR. RAMESH YERRABALLI: Now let's run it.

DR. JONATHAN VALVANO: Yeah.

All right, to run it, we'll download it.

And we're going to run debugger.

This is the real board.

We're going to set up a watch down here so that we count the number of edges.

And then we're going to hit the Go button up here.

And we're going to see down here in this window how many times we

pressed the button.

DR. RAMESH YERRABALLI: Now I'm going

to press the button and release it.

So that's the touch caused an increment by 1.

I'm going to press and release a second increment.

DR. JONATHAN VALVANO: It works.

DR. RAMESH YERRABALLI: And press and release the third increment.

DR. JONATHAN VALVANO: All right.

So in summary, we've seen that important events are set to trigger an interrupt.

We'll set the priority of that interrupt.

We'll set its vector so we know where to go.

We will arm it to say we're interested in interrupts.

We'll enable it in the nested vector interrupt controller.

And we'll clear the I bit so all interrupts

Help

```
volatile unsigned long FallingEdges = 0;
void EdgeCounter_Init(void){
  SYSCTL_RCGC2_R |= 0x00000020; // (a) activate clock for port F
  FallingEdges = 0;             // (b) initialize count and wait for clock
  GPIO_PORTF_DIR_R &= ~0x10;    // (c) make PF4 in (built-in button)
  GPIO_PORTF_AFSEL_R &= ~0x10;  //     disable alt funct on PF4
  GPIO_PORTF_DEN_R |= 0x10;     //     enable digital I/O on PF4
  GPIO_PORTF_PCTL_R &= ~0x000F0000; //  configure PF4 as GPIO
  GPIO_PORTF_AMSEL_R &= ~0x10;  //    disable analog functionality on PF4
  GPIO_PORTF_PUR_R |= 0x10;     //     enable weak pull-up on PF4
  GPIO_PORTF_IS_R &= ~0x10;     // (d) PF4 is edge-sensitive
  GPIO_PORTF_IBE_R &= ~0x10;    //     PF4 is not both edges
  GPIO_PORTF_IEV_R &= ~0x10;    //     PF4 falling edge event
  GPIO_PORTF_ICR_R = 0x10;      // (e) clear flag4
  GPIO_PORTF_IM_R |= 0x10;      // (f) arm interrupt on PF4
  NVIC_PRI7_R = (NVIC_PRI7_R&0xFF00FFFF)|0x00A00000; // (g) priority 5
  NVIC_EN0_R = 0x40000000;      // (h) enable interrupt 30 in NVIC
  EnableInterrupts();           // (i) Enable global Interrupt flag (I)
}

void GPIOPortF_Handler(void){
  GPIO_PORTF_ICR_R = 0x10;      // acknowledge flag4
  FallingEdges = FallingEdges + 1;
}

int main(void){
    EdgeCounter_Init(); // initialize GPIO Port F interrupt
    while(1){
      WaitForInterrupt();
    }
  }
```

Program 12.4. Interrupt-driven edge-triggered input that counts rising edges of PF4 (C12_EdgeInterrupt).

This initialization is shown to enable interrupts in step (i). However, in most systems we would not enable interrupts in the device initialization. Rather, it is good design to initialize all devices in the system, then enable interrupts. All ISRs must acknowledge the interrupt by clearing the trigger flag that requested the interrupt. For edge-triggered PF4, the trigger flag is bit 4 of the **`GPIO_PORTF_RIS_R`** register. This flag can be cleared by writing a 0x10 to **`GPIO_PORTF_ICR_R`**.

If two or more triggers share the same vector, these requests are called **polled interrupts**, and the ISR must determine which trigger generated the interrupt. If the requests have separate vectors, then these requests are called **vectored interrupts** and the ISR knows which trigger caused the interrupt.

One of the problems with switches is called **switch bounce**. Many inexpensive switches will mechanically oscillate for up to a few milliseconds when touched or released. It behaves like an underdamped oscillator. These mechanical oscillations cause electrical oscillations such that a port pin will oscillate high/low during the bounce. In some cases this bounce should be removed. To remove switch bounce we can ignore changes in a switch that occur within 10 ms of each other. In other words, recognize a switch transition, disarm interrupts for 10ms, and then rearm after 10 ms. Alternatively, we could record the time of the switch transition. If the time between this transition and the previous transition is less than 10ms, ignore it. If the time is more than 10 ms, then accept and process the input as a real event.

Help