SIMULATION SESSION

INTERRUPTS AND POWER SAVING

INTRODUCTION

In this session you will begin to write your own code based on what you have learned so far. Any new concepts will be introduced on a needs basis.

At the end of this session, you should be able to do the following:

INTENDED LEARNING OUTCOMES

- Program and implement a hardware timer interrupt to interrupt execution at a regular interval
- Program and implement an external interrupt to interrupt execution asychronously
- Calculate the interrupt latency
- Deploy and debug a PIC24 program on real hardware
- Use a schematic to write C code to control physical hardware

TASK 1 - USING A HARDWARE TIMER INTERRUPT

The PIC24 has 5 hardware timers that can run independently of the main thread of code. We are once again going to use **TIMER1**. Refer to the lecture notes on how the timer is programmed. In this example, we shall look at configuring and using a timer **interrupt**. This differs from our previous experience of timers where we were responsible for monitoring the value of the timer.

FOR A TIMER INTERRUPT, THE **HARDWARE** MONITORS THE VALUE OF THE TIMER (TMR1) AND RAISES AN INTERRUPT WHEN IT REACHES A GIVEN VALUE

- The timer has a special '**period register**' – for TIMER1 this is register **PR1**.
- When the timer TMR1 == PR1, it recycles back to zero (TMR=0) and sets the appropriate interrupt flag.
- If the Timer Interrupt is enabled, this event raises an interrupt.
- If this interrupt has a higher priority than the current process, the CPU branches to the so-called **Interrupt Service Routine** (ISR). For TIMER1, this is always called **T1Interrupt** on the PIC24**.**
- The Timer still counts once every T seconds, where $T = \frac{2}{F_{osc}} \cdot Prescale$

The value of **Prescale** can be set to one of 4 values: 1, 8, 64 or 256.

SEE CHAPTER 5 IN (DI JASIO L., PROGRAMMING 16-BIT MICROCONTROLLERS IN C) FOR MORE DETAILS.

1. Open the project in the folder Task 1 called timer_interrupt.
    - This code is a working example of a timer interrupt. Study this code carefully and read all the comments.
2. Build and step through the code. Keep stepping until the program enters the **_T1Interrupt()** function

TASKS	
Q. What is the value of TMR1?	
Q. What caused the program to jump into the _T1Interrupt() function?	
Q. What is the purpose of the Idle() function? **Q. Referring to the PIC24FJ128 datasheet, find out the benefits of the idle mode.**	
Task. The current value of PR1=0x31. The current value of PRESCALE=1. **Calculate the current interval between timer interrupts**	**Answer** Interval =
Task. Use the built-in stopwatch to verify your answer	

3. You shall now change some parameters to raise an interrupt **once a second**.

TASK	
Calculate the following parameters to generate an interrupt once a second:	
Parameter	Value
PR1	
PRESCALE (1,8,64 or 256)	
From Section 10 of the PIC24FJ128 datasheet, what is the correct value for the Timer Clock PresScale parameter (TCKPS)	
TCKPS	
Modify and test your code. Use the stopwatch to verify the timer interrupt is working at the correct frequency	
Test your code visually. Switch to the Proteus VSM debugger and check the circuit behaves as expected.	

4. The hardware timer TIMER1 is now configured for 1Hz (1 cycle per second). The next task is to display a count on the two displays (in seconds).
5. Where it says //TASK 3, implement the code to perform **display updates**. You should display the elapsed time in seconds on the two displays (0..99)
   - Hint: use the `update_display` function
   - Use the integer divide '/' and modulus '%' to get the tens and units
6. Once this is working, check the following:

Questions

Q. Has this code affected the accuracy of the timer?

Q. Does the interval for display updates vary?

Q. In terms of accuracy and power consumption, how does this compare with using polling or blocking loops?

(discuss with the tutor if you are not sure)

Q. (Advanced)
From the time that TMR1 == PR1, how long does it take to enter the ISR?

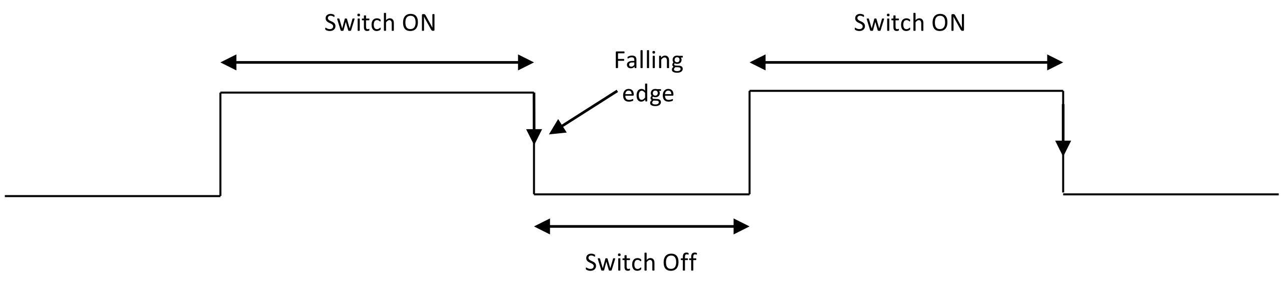
TASK 2 – EXTERNAL INTERRUPTS

The timer is an internal interrupt source. Most Microcontrollers and Microprocessors can also respond to external events. In this example we are going to respond to a simple push switch. Pressing the switch shall raise a hardware interrupt which will start or stop a timer. The question of interrupt priority is also raised – which should have higher priority, the switch or the timer?

1. Open the project file Task2\external_interrupt
2. Run the simulation and try clicking on SW2 on the schematic
   a. You should find that by **releasing** the switch, you switch the display on and off
3. Look through the code to ensure you understand it
   a. Single step / use breakpoints as appropriate

There are two interrupts used in this example:

- The first is the TIMER1 (used in the previous exercise) which generates an accurate count every second
- The second is an 'input capture' – currently configured to detect a **falling edge** on pin RD8



Questions

Q. What is the interrupt priority of the TIMER?

Q. What is the interrupt priority of the INPUT CAPTURE?

Comment on this

TASK. Make appropriate changes to the code – see section 12.1 in the PIC24FJ128 datasheet for details. Note your changes below:

It would seem more appropriate that the switch should be active on a rising edge.

4. Refer again to section 12.1 in the PIC24FJ128 datasheet. Also look at the source code under the comment that reads //Configure the input capture (IC1)

Task

Change the input capture to react on a rising edge. Note your changes below

Note that SW2 only toggles the display – it does NOT stop or start the counter. In the next exercise, you are to add another switch that starts and stops the counter.

TASK (OPTIONAL for those that like to tinker with electronics!)

TASK – modify your circuit to add an extra switch SW3 to drive Input Capture 2.

Modify the code you already have to do the following:

SW3 shall be is used to start, stop and reset the counter.

You should configure and create an interrupt service routine (ISR) to detect the rising edge of SW3 and to control the **state** of the system.

State 1 - When SW3 is pressed the first time, this should start the counter.
State 2 - When SW3 is pressed the second time, this should stop the counter.
State 3 - When SW3 is pressed the third time, the counter should be reset and return to state 1

TASK: Draw the state diagram
TASK: Implement the code to configure the Input pins, the Input Capture 2 control register and enable the interrupt associated with it.
TASK: Implement the interrupt service routine for Input Capture 2
TASK: Modify the other ISR's as appropriate.
Hint – the counter should not always be incremented in the TIMER routine.