PSoC 5 Interrupts

Purpose

This lab will introduce you to the hardware and software implementation of interrupts on the PSoC 5. It will also allow you to see the effects of interrupt priorities and the pitfalls of spending too much time in the interrupt service routine. You will do this by creating two timers, each with its own interrupt service routine. Each ISR will be responsible for toggling its own digital output pin. By adding computation to the higher-priority ISR you will be able to delay execution of the lower-priority interrupt handler.

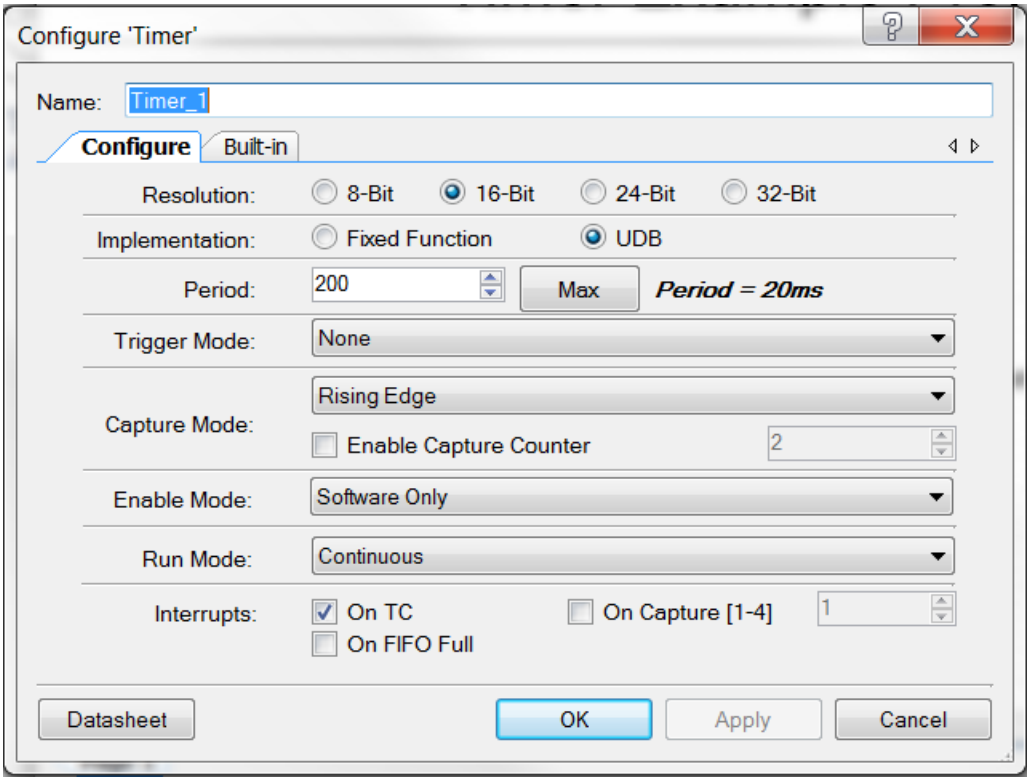
Procedure

Select the Timer example project using File->Open->Sample Project; set the architecture to PSoC 5LP to be sure that you get the right version of software. Before modifying the project, study the initial configuration:

- TopDesign.cysch: You will see Timer_1 and its connections. Its interrupt output is connected to TimerISR, which represents the link between the hardware interrupt signal and the interrupt service routine.
- main.c:
    - CY_ISR(InterruptHandler) is the interrupt handler for Timer_1. The call to Timer_1_STATUS clears the interrupt register so that the timer's next interrupt can occur. The update of InterruptCnt is part of the timer demo and not inherent to the timer operation.
    - main() enables global interrupts, then enables the timer interrupt, then initializes the timer before executing the demo for(;;) loop.
- TimerISR.c: One of the functions in this file is the definition of CY_ISR, which is actually implemented using a C macro. When you call this function, you should replace the formal parameter given here with the name you give to your ISR.

You will add several components to your TopDesign.cysch.

- Set the period of Timer_1 to 200. Its parameters should now look like this:



- Add two digital output pins, heartbeat1 and heartbeat2. Be sure to assign each pin to a location on the connector. These pins will be operated only by software so uncheck the HW Connection box.
- Add interrupt handler Timer2ISR.
- Add a second timer Timer_2. Use the same parameters values for Timer_2 as for Timer_1: 16-bit resolution, UDB, Period = 200, Trigger Mode none, Capture Mode Rising Edge, Enable Mode Software Only, Run Mode Continuous, Interrupts On TC. Connect its reset, capture, and clock pins to the corresponding clock pins on Timer_1.

You also need to change your main.c:

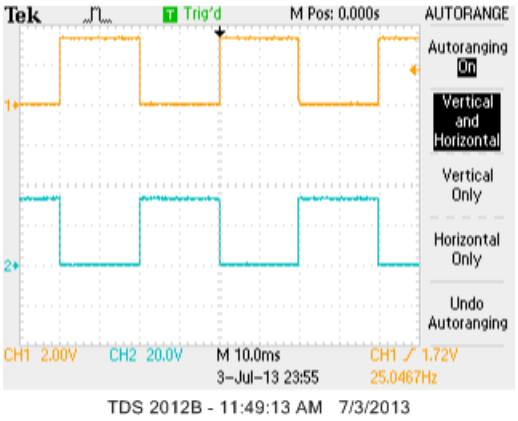
- Add a global variable heartbeat and initialize it to 0.
- The interfaces to the two digital output pins are contained in heartbeat1.h and heartbeat2.h. Add code to the Timer_1 ISR to toggle the value of the heartbeat variable and write it to the heartbeat1 pin:

```
heartbeat = ~heartbeat;
heartbeat1_Write(heartbeat);
```

- Add an ISR for Timer_2 called Timer_2_Handler. Add code to write the complement of the heartbeat variable's value to the heartbeat2 pin. This routine should not modify the value of the heartbeat variable.
- Add code to initialize the Timer_2 interrupt handler and to start Timer_2.
- Set the value of the CyDelay parameter to 1u.

- You may want to modify the display code to write the values of both timers. The parameters to LCD_Position() are the row and column of the first character of the output.

You should verify your setup by compiling and downloading it. When you connect an oscilloscope or logic analyzer to your two output pins, you should see that heartbeat2 has the opposite value of heartbeat1 and that their rising and falling edges are aligned:



Experiments

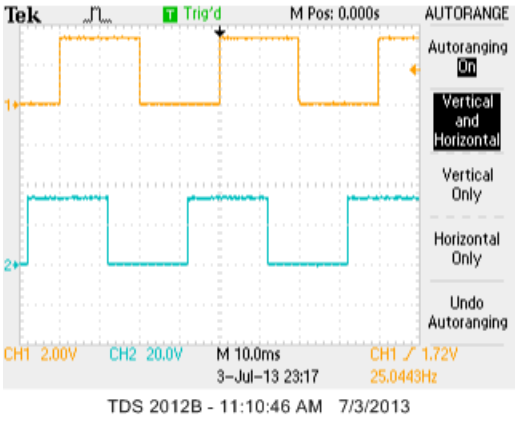
You will now extend the execution time of one of the ISRs and see its effect on the output pins.

You will add delay to the Timer_1 ISR with a simple loop:

```
#if 1
    for (idelay=0, scum=0; idelay<NDELAY; idelay++) scum = scum+1;
#endif
```

You can use the #if declaration to turn the code on and off during compilation. You should declare idelay, scum, and NDELAY as globals. For starters, set NDELAY=1000.

Recompile and reload your design. You should see the phase of the heartbeat2 signal delayed:



Q1: What value of NDELAY completely blocks execution of the Timer_2 ISR?

Q2: What happens to the heartbeat2 waveform at your maximum value of NDELAY? Why does this happen?

Now turn off the delay code in the Timer_1 ISR and add equivalent code to the Timer_2 ISR.

Q3: What is the phase relationship between heartbeat1 and heartbeat2?

You Should Turn In

1. Your main.c code.
2. Answers to Q1, Q2, Q3.

As always, output snapshots are cool.

Super-duper bonus points: use the potentiometer to change the value of NDELAY on-the-fly.

