Lab 3 - Part 3
HCS12 Timer Input Capture and Output Compare

Introduction and Objectives

Last week you wrote programs using the MC9S12 Timer Overflow Interrupt and Real Time Interrupt. This week you will work with the timer Input Capture and Timer Output Compare functions. To demonstrate their usage you will design a program to measure your reaction time and display it on the 7-seg display.

1   The Lab

1. Start a new project. This time select "float is IEEE32".

2. Start with the following program, which is just a do-nothing infinite loop:

```
#include "derivative.h"
#include "vectors12.h"
#define TRUE 1

main()
{
    while(TRUE)
    {
        __asm(wai);
    }
}
```

3. Add to your program a global variable called `value`. Add the following Real Time Interrupt ISR, and add code to the main part of the program to generate a real time interrupt every 2ms. In your main loop, set `value` to a known value (e.g., `0x1234`), and verify that this is correctly displayed on the seven segment LEDs.

```
interrupt void RTI_isr(void)
{
    static unsigned char digit = 0;
    const char c2seven_seg[] = { 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D,
                                 0x7D, 0x07, 0x7F, 0x6F, 0x77, 0x7c,
                                 0x58, 0x5e, 0x79, 0x71 };
    switch (digit){
        case 0: PTP = 0x0E;
                PORTB = c2seven_seg[(value>>12)&0x0F];
                break;
        case 1: PTP = 0x0D;
```

```
                        PORTB = c2seven_seg[(value>>8)&0x0F];
                        break;
               case 2: PTP = 0x0B;
                        PORTB = c2seven_seg[(value>>4)&0x0F];
                        break;
               case 0: PTP = 0x07;
                        PORTB = c2seven_seg[(value)&0x0F];
                        break;
     }
     if (++digit >= 4) digit = 0;
     CRGFLG = 0x80;
```

4. Now lets measure your reaction time and display it on the 7-seg display.

   (a) Make Bit 0 of Port A an output, and write a 0 to it. Make Bit 4 of Port A an input. Run a wire from `PA4` to `PT0`. Connect a 10 K resistor from `PT0` to VCC. When this is done, `PT0` will be high. When Key 1 is pressed on the keypad, `PT0` will go low.

   (b) Make `PTH` an input port. Make sure all DIP switches are off. Run a wire from `PH3` to `PT2`. When this is done, `PT2` will be high. When SW2 (below the DIP switches) is pressed, `PT2` will go low.

   (c) Make Bit 2 of Port E an output, and write a 0 to it. This will allow Bit 2 of Port E to turn on and off the LED next to the relay.

   (d) Set up `PT0` as input capture, capture falling edge. Enable `PT0` interrupt.

   (e) Set up `PT2` as input capture, capture falling edge. Disable `PT2` interrupt.

   (f) In an infinite loop, wait until the falling edge on `PT2`, then calculate the time between the edge on `PT2` and the edge on `PT0`. Convert this from clock cycles to milliseconds, and convert to BCD to display on the seven-segment LEDs. Clear the `TC0` and `TC2` flags and enable the Channel 0 interrupt when `PH0` goes low. Also, set value to 0 when `PH0` goes low. This allow you to start again by pressing `SW5`.

   You can let the MC9S12 convert clock cycles to time, and display the time between two edges on the seven-segment LEDs:

   - Need to divide the number of clock cycles by (24,000,000/prescaler)
   - Easiest to do this using floating point numbers
   - Need to convert back to fixed-point number to display on seven-segment LEDs
   - Can display number of milliseconds on seven segment LEDs
   - This will display time in hexadecimal. It makes sense to humans to display time in decimal convert hexadecimal to BCD

```
#define clock_freq 24000000.0
#define prescaler ((float)(1 << (TSCR2&0x07)))
unsigned int t_first, t_second, dt;
unsigned int value;              // Value to display on seven-segment LEDs
...
     dt = ((float)(t_second - t_first))*prescaler)/clock_freq * 1000.0;
     value = hex2bcd(dt);
...
unsigned int hex2bcd(unsigned int x)
{
     unsigned int d3,d2,d1,d0;
     if (x > 9999) return 0xFFFF;
     d3 = x/1000;
     x = x-d3*1000;
```

```
                    d2 = x/100;
                    x = x−d2∗100;
                    d1 = x/10;
                    x = x−d1∗10;
                    d0 = x;
                    return d3∗16∗16∗16 + d2∗16∗16 + d1∗16 + d0;
            }
```

    (g) Write an ISR to respond the the falling edge of `PT0`. In the ISR, save the time in a global variable, and set Bit 2 of Port E. This will turn on the LED by the relay. Disable the `TC0` interrupt, clear the `TC1` flag and enable the `TC1` interrupt.

    (h) Write an ISR to respond to the falling edge of `PT2`. In the ISR, save the time in a global variable, and clear Bit 2 of Port E. This will turn off the LED by the relay. Disable the `TC1` interrupt.

    (i) Test your program.

5. Write an output compare ISR for `PT4` to generate the waveform shown in Figure 1. Use a logic probe to verify that it works.
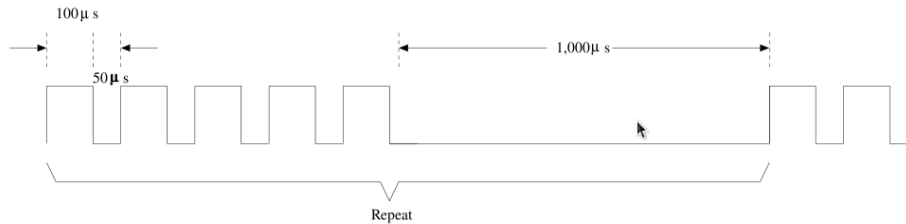

Figure 1: Waveform.