Lab 3 – Part 3
HCS12 Timer Interrupt 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. Prelab

For the prelab write the interrupt service routines TIC0_ISR and TIC2_ISR.

2. 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 ( )
{
        whi l e (TRUE)
        {
                _asm(wai);
        }
}
```

3. Add to your program a global 16-bit variable called *reaction_time*. 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 *reaction_time* to a known value (e.g., 0x1234), and verify that this is correctly displayed on the seven segment LEDs.

Program 1 Program required to display characters on 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 [ ( reaction_time >>12)&0x0F ];
                 break;
        case 1 : PTP = 0x0D;
                 PORTB = c2seve_seg [ ( reaction_time >>8)&0x0F ];
                 break;
        case 2 : PTP = 0x0B;
                 PORTB = c2seven_seg [ ( reaction_time >>4)&0x0F ];
                 break;
        case 3 : PTP = 0x07;
                 PORTB = c2seven_seg [ ( reaction_time )&0x0F ];
                 break;
        }
        digit = (digit + 1) % 4;
        CRGFLG = 0x80;
}
```

---

4. Now measure your reaction time and display it on the 7-seg display. You will set up the hardware so that when Key 1 (of the 4 x 4 keypad on the lower right of the Dragon12 board) is pressed, the LED labeled RGB will turn on, and the time of the Key 1 press will be captured. In a recent board edition you need to use the RGB LED. To use this LED you need to include the following code in your program:

```
DDRP = DDRP | 0x7F;    // Use P0-P3 for 7-seg, and P4-P6 for RGB LED
PTP    = PTP | 0x0F;
DDRJ = DDRJ | 0x02;       // Make J1 output
PTJ    = PTJ & ~0x02;
DDRB = DDRB | 0xFF;  //  Activate control lines for 7-seg display
```

To turn off all colors on the RGB LED use:

```
PTP = PTP&~0x70;
```

To turn on all colors on the RGB LED use:

```
PTP = PTP | 0x70;
```

When SW2 of the push-button switches (below the DIP switches on the lower left side of the Dragon12 board) is pushed, the time of that push will be captured. You will have a lab partner press Key 1 to turn on the RGB LED. When you see the RGB LED turn on, you will press SW2 as quickly as possible. By subtracting the two times (SW2 press minus Key 1 press), you can measure your reaction time, then display it (in milliseconds) on the seven-segment LEDs. After completing this once, pressing SW5 will reset the system so you can measure your reaction time again. To set up the hardware, you need to make a Key 1 press generate a falling edge on PT0 and make an SW2 press generate a falling edge on PT2. You will use interrupts to capture the times of the falling edges of PT0 and PT2. After the falling edge of PT2, you will display the time on the seven-segment LEDs. After pushing SW5 (which is connected to PTH0) you will start the process over again.

(a) Run a wire from PA4 to PT0. Connect a 10 K resistor from PT0 to VCC. In your program, make Bit 0 of Port A an output port, and write a 0 to it. Make Bit 4 of Port A an input port. Make Bit 0 of Port T an input port. When this is done, PT0 will be high when Key 1 is not pressed. When Key 1 is pressed on the keypad, PT0 will go low.

(b) Make sure all DIP switches are on. Run a wire from PH3 to PT2. In your program make Bit 3 of Port H an input port. Make Bit 2 of Port T an input port.

When this is done, PT2 will be high when SW2 is not pressed. When SW2 is pressed, PT2 will go low.

(c) Set up PT0 as input capture, to capture the time of the falling edge. In the setup part of your program, clear the TC0 flag and enable the PT0 interrupt. Write an ISR (called something like TIC0_ISR) to capture the time of the falling edge on PT0, and save the value in a global variable (`time_1`). In the ISR, turn on the RGB LED, disable the PT0 interrupt, and enable the PT2 interrupt. Be sure to clear both the TC0 and TC2 flags in the ISR.

(d) Set up PT2 as input capture, to capture the time of the falling edge. Clear the TC2 flag and disable the PT2 interrupt. Write an ISR (called something like TIC2_ISR) to capture the time of the falling edge on PT2, and save the value in a global variable (`time_2`), and set the global variable `done` to 1. In the ISR, turn off the RGB LED, and disable the PT2 interrupt. Be sure to clear the TC2 flag in the ISR.

(e) In the setup part of your program, set the global variable `done` to 0. Also, enable the timer subsystem. Set the timer prescaler so the overflow rate is greater than 100 ms.

(f) When your lab partner presses Key 1, the RGB LED will turn on. Press SW2 as quickly as you can. In the infinite loop part of your program, wait until SW2 is pressed by monitoring the global variable `done`.

After your program captures the time of the PT2 falling edge, calculate the time between the edge on PT2 (captured as `time_2` in the TIC2_ISR) and the edge on PT0 (captured as `time_1` in the TIC0_ISR). Convert this from clock cycles to milliseconds, and then convert to BCD to display on the seven-segment LEDs. (How to do this is discussed below.)

To reset the system, press SW5, which makes PH0 go low. In your program, when PH0 goes low, clear the TC0 and TC2 flags, and re-enable the TC0 interrupt. Also, set the global variables `reaction_time` and `done` to 0.

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

• Convert the number of timer cycles from PT0 edge to PT2 edge to a floating point number. Divide it by (24,000,000/prescaler).

• Convert the time in milliseconds back to the fixed-point number. Convert the hexadecimal fixed-point number to a BCD number so the time in decimal milliseconds will be displayed.

• Here is some code which will convert the time difference in timer clock cycles to BCD milliseconds:

```
#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 ) / clockfreq*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;
}
```

5. Write an output compare ISR to generate a 10 Hz square wave on PT4. Use a logic probe to verify that it works.