Microprocessor Systems Laboratory (ELCE333)

Laboratory Experiment No.7

MARKING SHEET

Student Names and ID No's: Amal AlQasimi 100036830
Shaikha AlBesher 100037064
Dana Bazazeh 100038654

Laboratory Section: Wednesday **Experiment Date: March 18, 2015**

No.	Criteria	Description	Weight %	Mark	Comments
1	Pre-lab	A mark will be allocated to each student that reflects his preparations for the lab.	20		
2	Performance In the lab	A mark will be allocated to each student individually that reflects his performance in the lab	10		
3	Results and Analysis	Documentation and analysis of the results for each task performed in the lab	30		
4	Summary/ Conclusions	Conclusions for each task performed in the lab	10		
5	Assignment Questions	Answers to assignment questions	20		
6	Report Presentation	Overall presentation of the report including proper layout and clarity of figures, tables, and graphs. Correct use of English language.	10		
	Total		**100**		

KHALIFA UNIVERSITY

ELCE 333: Microprocessor Systems Laboratory

Lab Report- Experiment No.6

Experiment Title: Using HCS12 Timers and Interrupts

Completed by:

Amal Al Qasimi (100036830)

Shaikha Al Besher (100037064)

Dana Bazazeh (100038654)

Lab instructors:

Mahmoud Khonji

Mohammed Al Zaabi

Spring 2015

Table of Contents

Summary

In this lab, students were introduced to the concept of timers and timer interrupts and how they are used in programs. The operation of the timer device is discussed and several registers and functions that are used to manipulate the timer settings are introduced. This lab consists of 3 main tasks that are described and analyzed in detail in this report. The first task talks about the basic operation of a timer and how to enable the timer, timer overflow interrupts and prescaler options. The idea behind this task is to use a delay created by the timer overflow interrupt in order to flash PORT B LEDs, were the prescaler value should be set using the PTH DIP Switches. Task 2 is about how to use the Output Compare function to produce digital wave forms. We analyze and modify a program that uses output compare to produce a 1KHz square pulse with a specified % duty cycle on pin PT2. The 3rd and final task talks about the Input Capture function that uses the timer to count, detect and measure the time of outside events. Here we analyze and modify a program that uses this function to measure the period and frequency of a square wave signal on pin PTT0 and write their values on the LCD. At the end of this report, a detailed analysis and a conclusion are presented to sum up the results obtained.

1. Introduction

The HCS12 microcontroller contains a timer system that consists of a standard timer module (TIM) which contains 8 channels of output compare and input capture functions, a 16 bit timer counter TCNT and a 16 bit pulse accumulator a. Timer functions are an essential part of many programs because they create time delay, measure frequency, pulse and period, count events, track time of day and create waveforms.

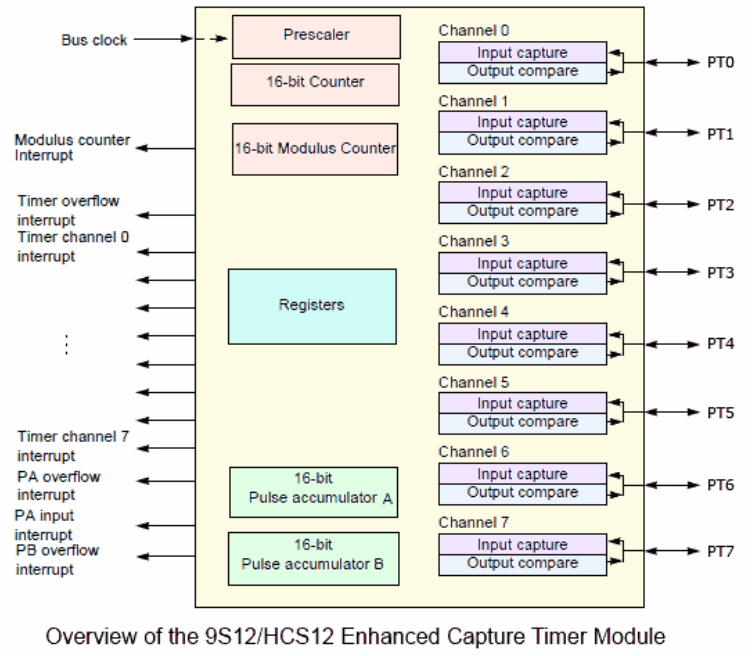

Figure 1: HCS12 Timer System

Timer Counter Register (TCNT)

TCNT is a 16 bit register used as a counter that is driven by the pre-scaled bus clock bus clock and is essential for input capture and output compare functions. TCNT starts counting from zero and counts up with each pulse coming from the oscillator until it reaches a maximum value of $FFFF, where it begins again at 0000 and a flag called *timer overflow flag (TOF)* is set to 1. The 3 registers TSCR1, TSCR2, TFLG2 are used for programming the HCS12 timer.

Output Compare Functions

The HCS12 has eight output compare functions used to trigger an action (event) at a specific time in the future. Each output compare channel consists of a 16-bit comparator, 16-bit compare register TCx, an output action pin (PTx, can be pulled high, pulled low, or toggled) and an interrupt request circuit.

Input Capture Functions

Physical time is often represented by the contents of the main timer. The occurrence of an event is represented by a signal edge (rising or falling edge). The time when an event occurs is recorded by latching the count of the main timer when a signal edge arrives. The HCS12 has eight input capture channels. Each channel has a 16-bit capture register, an input pin, edge-detection logic, and interrupt generation logic.

1.1 Aim

The aim of this report is to introduce the students to use of timers and how delays can be implemented using times and interrupts.

1.2 Objectives

On completion of this experiment the student should be able to:

1- Understand the concept of timers.
2- Write C program using timers overflow with interrupts.
3- To program the Output Compare feature of timer in HCS12.
4- To program the Input Capture feature of timer in HCS12.
5- Use the CodeWarrior IDE for the development of HCS12 microcontroller C programs.
6- To compile, download and debug/test a C program using
7- CodeWarrior C compiler and Dragon12 Plus Trainer board.
8-

2. Design and Results

Task 1: Timer Overflow

1.1 This task asks us to execute the program below that was provided in the labscript on the Dragon12 Plus Trainer. Next, using the help of the oscilloscope found in NI ELVIS instruments, we need to monitor the signal at any of the pins of PORTB. The program below flashes the LEDs of PORTB and we must examine the toggling rate for the pin and measure its frequency.

The full program is shown in figure 1 below:

```
#include <hidef.h> /* common defines and macros */
#include "derivative.h" /* derivative-specific definitions */
int flag=0;
void init_timer(void)
{TSCR1 = 0x80; // enable timer counter
TSCR2 = 0x87; // enable timer interrupt, set prescaler to 128
TFLG2 = TFLG2_TOF_MASK; } // Reset timer overflow flag
void main(void)
{ DDRB = 0xFF; //PORTB as output since LEDs are connected to it
DDRJ = 0xFF; //PTJ as output to control Dragon12+ LEDs
DDRP=0xFF;
PTP=0x0F; //Disable 7-seg display
PTJ=0x0; //Allow the LEDs to display data on PORTB pins
PORTB = 0x00;
init_timer();
__asm CLI;
for(;;){
if (flag==1)
{PORTB=PORTB ^ 0xFF;
flag=0;}} }
#pragma CODE_SEG NON_BANKED
void interrupt (((0x10000-Vtimovf)/2)-1) TIMOVF_ISR(void)
{flag=1;
TFLG2 =TFLG2_TOF_MASK; }//Clear Interrupt
```

Figure 1: Program code to flash PORTB LEDs using timer interrupt

We begin by enabling the timer counter by setting bit7 of the TSCR1 register and enabling the timer interrupt by setting the TEN bit (bit 7) of TSCR2 and the prescale value, which is the 4 lower bits if TSCR2 is set to 128. We then set the data direction registers for PORTB as done usually. We initialize a flag integer to indicate the occurrence of a timer interrupt. In an infinite loop, we constantly check the flag, if it's set, it means a timer interrupt has occurred and we flash

all the LEDs by XORing PORTB with 0xFF and then clearing the flag. Also, in the interrupt service function, we must clear the interrupt in order to receive new timer interrupts.

Next, we open the oscilloscope to analyze and measure the toggling frequency, figure 2 below shows that the frequency of PORTB pin when the prescaler value is 128 is 1.43Hz.

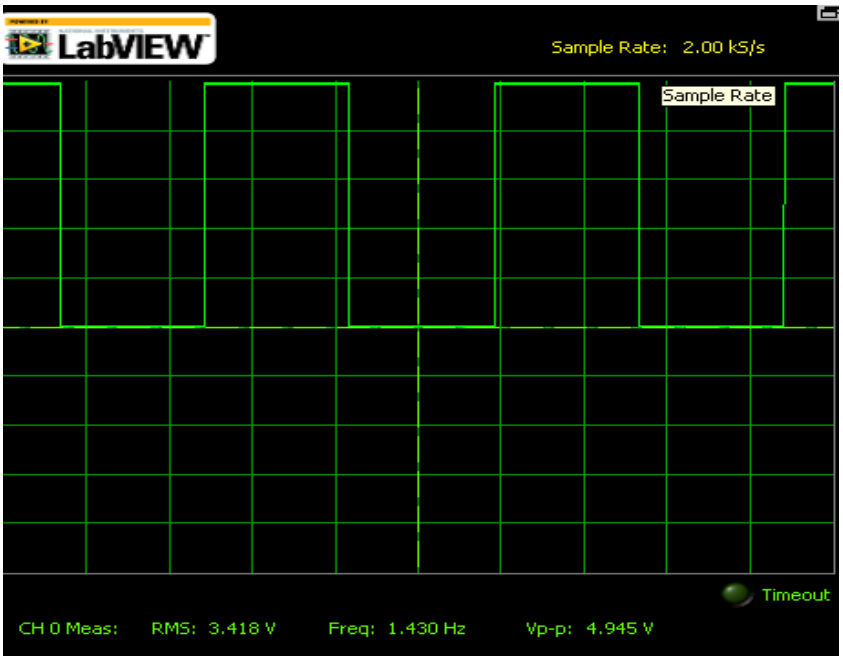

Figure 3: Output of the oscilloscope for prescaler value 8(decimal)

1.2 This section asks us to modify the program in order to set the prescaler value using the PTH DIP Switches and then fill in a table with specific prescaler values.

To do so, all we need to do is modify the below instruction:

TSCR2 = 0x87;

And instead replace it with:

TSCR2 = 0x80|PTH;

This will now OR the values of the 4 lower bits of TSCR2, which represent the prescaler value, with the value of PTH switches.

We analyzed and recorded the frequency for several prescaler values and the results are shown below in Table 1.

\

Table 1

Prescaler	Freq. of PORTB Pin (Hz)
000	183.1
001	91.5
010	45.7
011	22.07
100	10.98
101	5.76
110	2.77
111	1.43

Below are 3 different cases for the prescaler values of 000, 001 and 010 respectively:

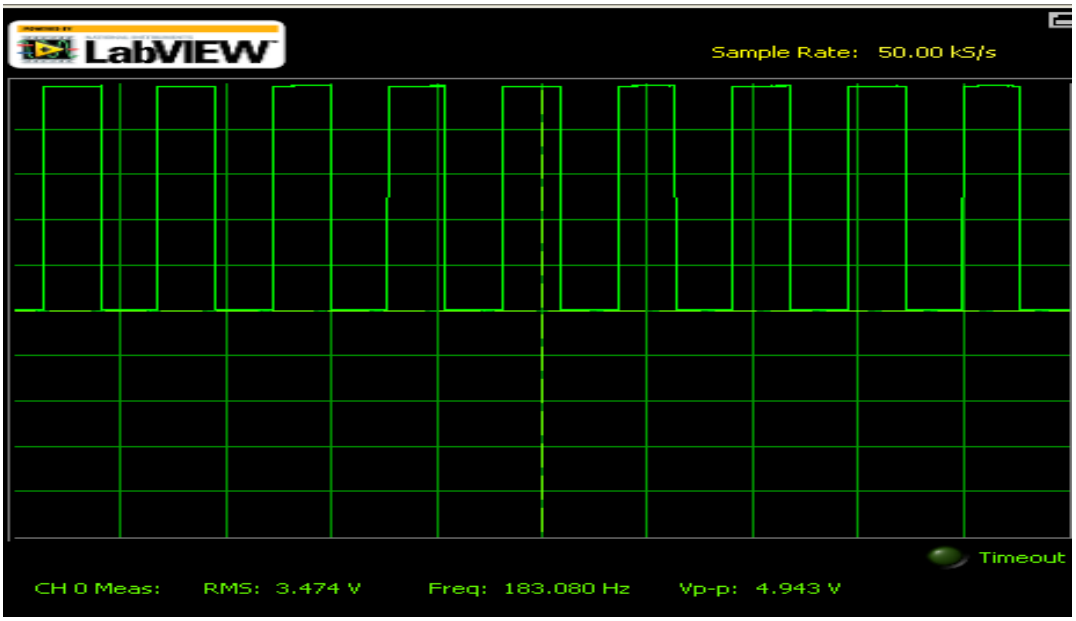

Figure 4: Output of the oscilloscope for prescaler value 000

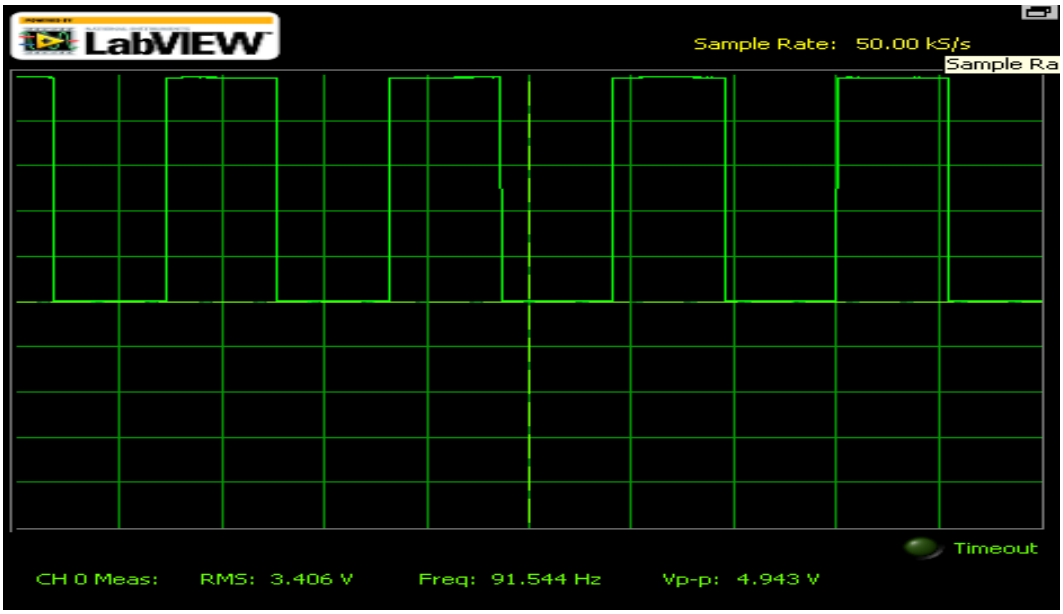

Figure 5: Output of the oscilloscope for prescaler value 001

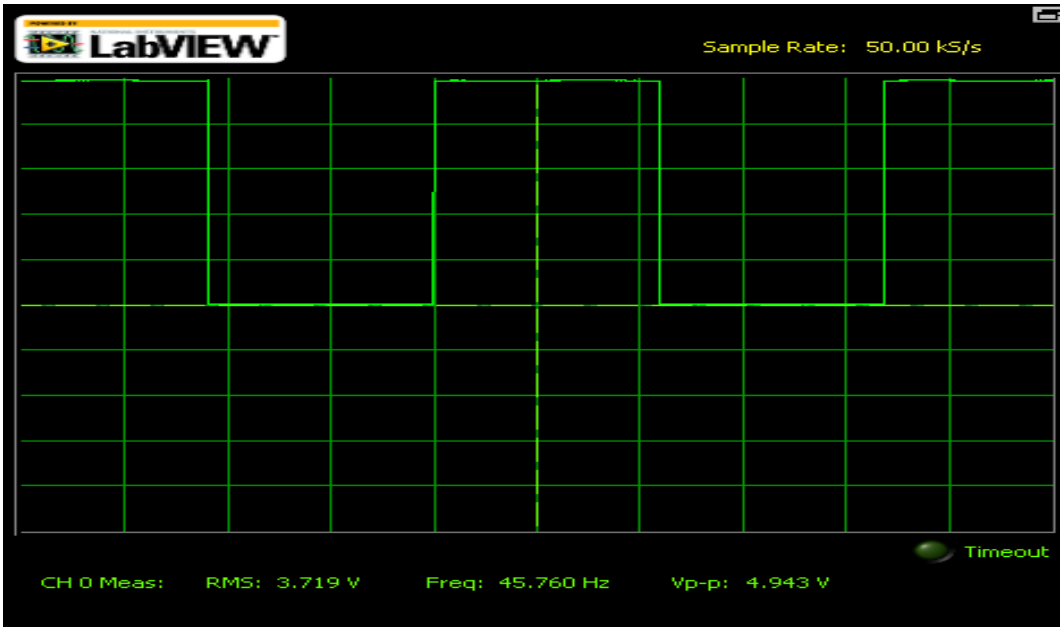

Figure 6: Output of the oscilloscope for prescaler value 010

We noticed from our results that, as the prescaler value increased by 1, the toggling frequency has halved. Therefore there is an inverse relationship between the prescaler value and the toggle frequency.

Task 2: Output Compare

2.1 This task asks us to run the output compare function given in the labscript and with the help of an oscilloscope, monitor the signal at PTT2 pin. Then, we have to measure the signal period and frequency. The full code is provided below which is used to produce a 1KHz square pulse with 50% duty cycle on pin PT2

```
#include <hidef.h>           /* common defines and macros */
#include "derivative.h"      /* derivative-specific definitions */
void init_timer(void)
{ TSCR1 = 0x80;               // enable timer counter
TSCR2 = 0x03;                 // disable timer interrupt, set prescaler to 8
TIE_C2I=1;                    // enable channel 2 interrupt
TFLG1 |= TFLG1_C2F_MASK;     // Clear interrupt flags
TIOS_IOS2=1;                  // enable output compare channel 2
TCTL2_OL2=1; }                // Toggle OC2 pin
void main(void)
{ init_timer();
__asm CLI;
for(;;){}
}
#pragma CODE_SEG NON_BANKED
void interrupt (((0x10000-Vtimch2)/2)-1) TIMCH2_ISR(void)
{TC2 += 1500; //start a new OC2 operation
//(No. of cycles (ON/OFF)= (24MHz*period(ON/OFF))/prescaler)
//period(ON/OFF)=1/(1KHz/2)
//(24MHz*0.0005)/8=1500
TCTL2_OL2=1; //Toggle OC2 pin
TFLG1 =TFLG1_C2F_MASK; } //reset Ch 2 interrupt
```

Figure 7: Program code to produce a 1KHz square pulse with 50% duty cycle on pin PT2

Again, we begin by enabling the timer and disabling the timer interrupt and we set the prescaler value to 8. Since we will be using channel 2, we enable channel 2 interrupt and the channel 2 itself as well as the output compare channel 2. In order to toggle OC2 pin, we reference the appendix at the end of the labscript and find that we can achieve this by setting OL2 pin of the TCTL2 register to 1 while keeping OM2 as 0.

Using the formula to attain a 50% duty cycle:

(24MHz*period(ON/OFF))/prescaler)

(24MHz*0.0005)/8=1500

In the interrupt service routine, we set the compare channel 2 TC2 to this value and toggle the OC2 pin and reset the interrupt flag for channel 2.

The output of the oscilloscope is shown in the following figure:

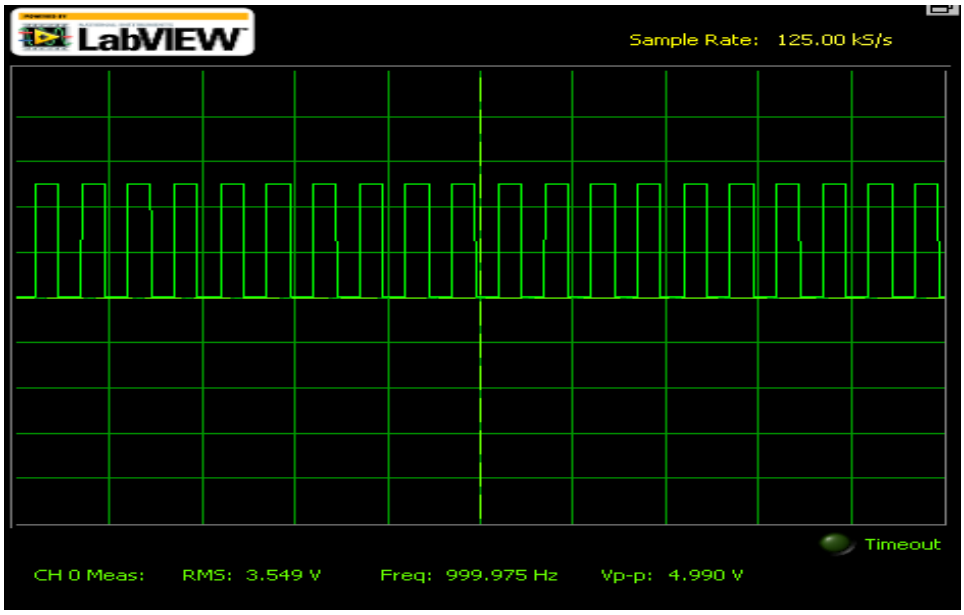

Figure 8: Output of the oscilloscope for task 2.1

We can clearly see that the frequency is around 1 KHz (999.975Hz) and therefore the period is 1/f = 1/1KHz = $10^{-3} secs$

2.2 This section asks us to modify the program in order to produce a 1 KHz 30% duty cycle pulse signal. To do so, we must perform calculations to obtain the value

Pulse Signal = (24MHz*0.0003)/8=900
And the remaining which is 70% = (24MHz*0.0007)/8 = 2100

The modified program is shown below:

```
Clock frequency = 24 MHz
Where 3MHz is the speed of the processor with prescalar 8, while n is the value
that has to be assigned to TC2 (number of cycles on\off).

#include <hidef.h> /* common defines and macros */
#include "derivative.h" /* derivative-specific definitions */

void init_timer(void)
{ TSCR1 = 0x80; // enable timer counter
TSCR2 = 0x03; // disable timer interrupt, set prescaler to 8
TIE_C2I=1; // enable channel 2 interrupt
TFLG1 |= TFLG1_C2F_MASK; // Clear interrupt flags
TIOS_IOS2=1; // enable output compare channel 2
TCTL2_OL2=1; } // Toggle OC2 pin
void main(void)
{ init_timer();
__asm CLI;
for(;;){}
}

#pragma CODE_SEG NON_BANKED
void interrupt (((0x10000-Vtimch2)/2)-1) TIMCH2_ISR(void)
{  if(TCTL2_OL2==1){

  TC2 += 900; //start a new OC2 operation
//(No. of cycles (ON/OFF)= (24MHz*period(ON/OFF))/prescaler)
//period(ON/OFF)=1/(1KHz/2)
//(24MHz*0.0003)/8=900
TCTL2_OL2 =0;
TCTL2_OM2 =1;
TFLG1 = TFLG1_C2F_MASK;
}   else if (TCTL2_OL2==0) //lower edge
{   TC2 += 2100; //start a new OC2 operation
 //(No. of cycles (OFF)= (24MHz*period(OFF))/prescaler)
 //period(OFF)=1/(1KHz/2)
 //(24MHz*0.0007)/8=2100
TCTL2_OL2 =1;
TCTL2_OM2 =1;
TFLG1 = TFLG1_C2F_MASK;
}} //reset Ch 2 interrupt
```

Figure 9: modified program code for task 2.2

What we have modified here is the interrupt service routine. First we used the if condition to check whether the OC2 pin is toggling. where we set TC2 to increase by 900 for a 30% duty cycle. We also clear the OL2 bit and set the OM2 bit of register TCTL2 and this will clear the OC2 pin to 0. The else condition is for the remaining 70% duty cycle where we will increase

TC2 by 2100 and set both OL2 and OM2 bits to 1 and this will cause the OC2 pin to be set to high as shown by the appendix.

The oscilloscope output is shown in the figure below:

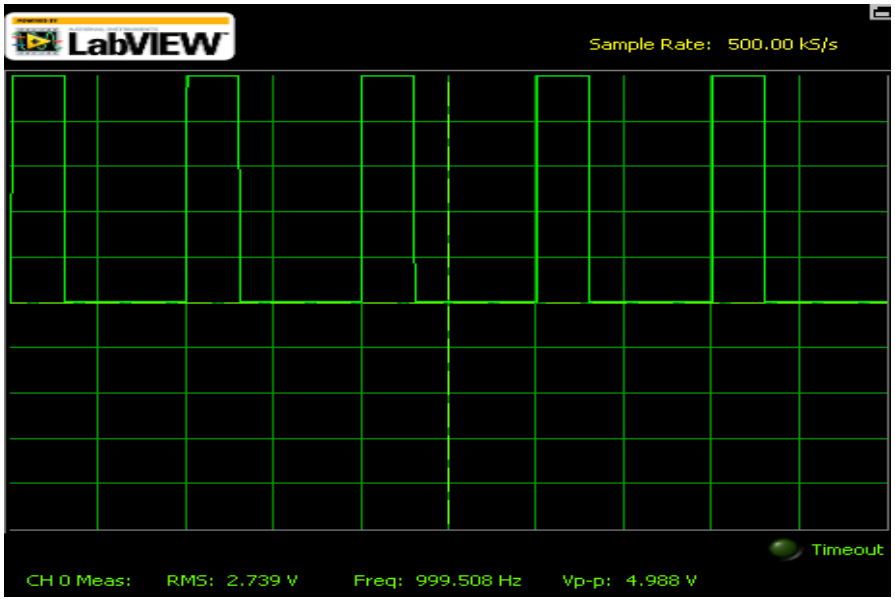

Figure 10: 2 The recorded frequency from the input compare function displayed in the

Task 3: Input Capture

In this task the HCS12 timer is used to count, detect and measure the time of outside events. The registers TCTL3 and TCTL4 are dedicated to Input Capture function; they are used to select the channel and the type of edge detection. If an edge is detected the TCNT will be loaded into the TCx register and raise it for the channel. The second register used for the function is is TIOS (Timer Input/Output Select). Zero has been set to select the input capture for the desired channel. The function generator is used to apply a 1 KHz 50% duty cycle signal on PTT0 and check the period and frequency values on the LCD, as shown in the figure on the right:

Figure 11: Output of the oscilloscope for task 2.2

Cearly from the figure, the period is 919µs and the frequency is 1.0881 kHz.

In the second part, we combined the previous code with the second part code of task 2 in order to measure the frequency and the duty cycle of the signal after connecting PTT0 with PTT2.

The combined code is:

```
#include <hidef.h>      /* common defines and macros */
#include "derivative.h"      /* derivative-specific definitions */
#include "lcd.h"

int edge1,flag; // this has to be int not float
float freq, period;

void init_timer(void)
{ TSCR1 = 0x80; // enable timer counter
TSCR2 = 0x03; // disable timer interrupt, set prescaler to 8
TIE_C0I=1; // enable channel 0 interrupt
TFLG1 |= TFLG1_C0F_MASK; // Clear interrupt flags
TIOS_IOS0=0; // enable input capture channel 0
TCTL4 = 0x01; // capture the rising edge of the PT0 pin
TIE_C2I=1; // enable channel 2 interrupt
TIOS_IOS2=1; // enable output compare channel 2
TCTL2_OL2=1; } // Toggle OC2 pin

void main(void)
{ init_timer();
LCD_Init();
__asm CLI;
flag=0;
for(;;){
LCDWriteLine(2, "period= ");
LCDWriteInt(period);
LCDWriteChar(' ');LCDWriteChar('u'); LCDWriteChar('s');
LCDWriteLine(1, "freq= ");
LCDWriteFloat(freq);
LCDWriteChar(' ');LCDWriteChar('K');LCDWriteChar('H');LCDWriteChar('z');
delay(100);
LCD_clear_disp();}}

#pragma CODE_SEG NON_BANKED

void interrupt (((0x10000-Vtimch0)/2)-1) TIMCH0_ISR(void)

{if(flag==0) { edge1= TC0; // save the first captured edge
flag=1;}
else{period=TC0 - edge1; // calculates period in cycles (second captured edge - first captured edge))
period = period * 0.33; // calculates period in us (period in cycles*(1/(24/prescaler)))
freq=(1/period)*1000; // calculates frequency in KHz
flag=0;}
TFLG1 =TFLG1_C0F_MASK; } //reset Ch 0 interrupt
 void interrupt (((0x10000-Vtimch2)/2)-1) TIMCH2_ISR(void) {
   if(TCTL2_OL2==1)
```

```
 {TC2 += 900;
 TCTL2_OL2 =0;
 TCTL2_OM2 =1;
 TFLG1 = TFLG1_C2F_MASK;
} else if(TCTL2_OL2==0)
 {

TC2 += 2100;
TCTL2_OL2 =1;
TCTL2_OM2 =1;
TFLG1 = TFLG1_C2F_MASK;
 }
 } //reset Ch 2 interrupt
```

The results are quite similar to the previous part, shown in Figure 12:

Figure 12: The results of combining the input and the output compare shown in the LCD screen

Frequency =1.0101 kHz, and the period=990μs, the results are approximately same as the previous part.

1. Analysis and Interpretation

In this laboratory experiment three tasks was accomplished in order to understand the value and functionality of timers. In task one a C code for timer overflow was designed. In task two the counter was modified to work with a timer, and the timer decides for the counter where to start its counting. In task three, the timer was designed to change the buzzer sound.

2. Conclusion

To sum up, the first task mainly showed how the bus clock can be changed by changing the value of the prescalar TSCR2, however, this can only decrease the frequency (increase the time) but is not applicable to increase the frequency (bus clock). The second task showed the importance of using interrupt services routine to enhance real time application. In addition, the timer interrupts allow the program to do other things while waiting for a timing event to occur. In case of setting a certain timing it is important to be careful in which mode the program would operate (whether it's in serial monitor or in standalone mode) because the frequency is different for each one which would change the value. From the third task we understood how the output compare function can be used to do the same thing as the previous two tasks in a more efficient and practical way. Using the output compare function, the programmer can either decrease or increase the frequency unlike setting the prescalar alone, and as understood through experiment, this function can be used for developing different application such as generating square waves with determined period.

Assignment Questions

When using two outputs compare channels we can generate two different frequency waveforms. Write a program that produces two signals with different frequencies. It should output a 50-Hz square wave on Port T bit 7 and a 270-Hz square wave on Port T bit0.

```
#include <hidef.h> /* common defines and macros */
#include "derivative.h" /* derivative-specific definitions */
void main(void)
{
unsigned int Tcount;
TSCR1 = 0x80;       //enable the timer
TIOS = 0b00100000; //Output Compare option for Channel 5
TCTL1 = 0b01000000; //Toggle PT7 pin option
TCTL2 = 0b00000001; //Toggle PT0 pin option
TIE= 0b10000001;      //enable interrupt for bit 7 and bit0

TC7 = TCNT;      // channel seven have the same value of the free running counter
TC7 = TC7 + 40000; //set the frequency of toggling channel 7 to 50 hz
                   //  (4Mhz)/(2*50hz) = 40000



TC0 = TC7 + 7407; //set the frequency of toggling channel 0 to 270 hz
                   //(4Mhz)/(2*270) = 7407

 TFLG1 = TFLG1_C0F_MASK;//rest interrupt flag of channel 0
 TFLG1 = TFLG1_C7F_MASK; //reset interrupt flag of channel 7
  asm(CLi);  //enable interrupt
for(;;)
{


}
}
void interrupt 15 channel_7 (void) {

   TFLG1 = TFLG1_C7F_MASK;   // reset interrupt flag of channel 7
   TC7= TC7 + 40000;         //reset the counter for 50 hz


}
void interrupt 8 channel_0 (void) {

       TFLG1 = TFLG1_C7F_MASK;  //reset interrupt flag of channel 0
       TC0 = TC0 + 7407;       //reset the counter for 270hz

}
```

References

1. CourseTextBook
2. http://www.evbplus.com/
3. MC9S12DP256User'sManual