

**Khalifa University of Science, Technology and Research**

**Electronic Engineering Department**

**ELCE332 Microprocessor Systems laboratory**

**Laboratory Experiment 7**

**Using HCS12 Timers and Interrupts**

**Lab Partners**

Hamad Eisa Alhazami 100036985

Jasim Alhammadi 100035510

**Date Experiment Performed**: 18-3-2015

**Date Lab Report Submitted**: 25-3-2015

**Lab Instructor: Mr. Mohammed Alzaabi/ Mr. Mahmoud Khounji**

**Spring 2015**

# Summary

The purpose of this report is to discuss the results obtained after conducting two of the three main tasks in the laboratory session. Unfortunately, the team was only able to complete the first two tasks. The first task required observing the frequency of Port B with a fixed prescaler value then using Port H Dip Switches to change the values of the prescaler dynamically. The second task was about measuring the signal period and frequency of the PTT2 pin, after that, the nest step was producing a 1 KHz 30% duty cycle signal.

# 

Contents

[Summary 2](#_Toc415055078)

[1. Introduction 4](#_Toc415055079)

[1.1 Aim 4](#_Toc415055080)

[1.2 Objectives 4](#_Toc415055081)

[2. Design and Results 5](#_Toc415055082)

[***Task 1: Timer Overflow*** 5](#_Toc415055083)

[Task 2: Output Compare 7](#_Toc415055084)

[3. Analysis and Interpretation 9](#_Toc415055085)

[4. Assignment questions 10](#_Toc415055086)

[5. Conclusions and Recommendations 12](#_Toc415055087)

# 1. Introduction

Timers are a very important feature in every microcontroller, for these timers to operate efficiently, a clock pulse is needed. The clock source can be either internal or external. In terms of the internal clock source, oscillator frequency on the XTAL/EXTAL pins is fed into the timer. Timers are used for time measurement and time delay generation. On the other hand, with the external clock option, pulses feed through one of the HCS12’s pins. HCS12 has a single 16-bit counter called *free-running timer*. The 16-bit register for this timer is called **TCNT** (**timer count**). Since this register is used for event counting it is called a counter.

In order to implement an overflow, the following sequential steps must be followed:

1. The desired timer frequency should be set using the prescaler. (TSCR2)
2. The timer should be enabled. (TEN in TSCR1)
3. TOF is then cleared by writing “1” to it. (TFLG2)
4. A loop that counts the number of overflows should be created.

* Wait on TOF.
* Clear TOF by writing “1” to it.
* Repeat until loop is broken.

## 1.1 Aim

To introduce the students to use of timers and how delays can be implemented using timers 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 CodeWarrior C compiler and Dragon12 Plus Trainer board.

# 2. Design and Results

## ***Task 1: Timer Overflow***

In the first task, it is required to develop a mechanism using C language to generate pulses, and with the oscilloscope. After that the results should be monitored from PORTB pins. Then, modifying the program by setting the pre-scalar using PTH DIP switches, and measuring the frequency for different pre-scalar values. The code is written by Code Warrior and executed on Dragon 12 Plus Trainer Board. The code is shown 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**

**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(;;){**

**TSCR2\_PR0 = PTH\_PTH0 ; //**

**TSCR2\_PR1 = PTH\_PTH1 ; //**

**TSCR2\_PR2 = PTH\_PTH2 ; //**

**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**

At first, the provided code is run and the results are observed. Then, the code is modified by setting the prescaler to the first three pins of DIP switches in order to be able to set the prescaler values manually. The modification is set into a loop in order to make the prescaler dynamic and flexible to instant changes. The code is run on the board, and examination is made on the following pre-scalar values:

Table 1: frequency values of PORTB

|  |  |  |
| --- | --- | --- |
| Pre-scalar in Decimal | Pre-scalar | Freq. of PORTB Pin |
| 1 | 000 | 183 Hz |
| 2 | 001 | 91.5 Hz |
| 4 | 010 | 45.8 Hz |
| 8 | 011 | 22.8 Hz |
| 16 | 100 | 11.4 Hz |
| 32 | 101 | 5.7 Hz |
| 64 | 110 | 2.8 Hz |
| 128 | 111 | 1.4 Hz |

## Task 2: Output Compare

In the second task, it is required to create a C code that generates pulses with predefined duty cycle and frequency. Then, modify the code to produce 1 KHz 30% duty cycle pulse signal. The code is written by Code Warrior and executed on Dragon 12 Plus Trainer Board. The code is shown below:

**#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;**

**TCTL2\_OM2=1; } // Toggle OC2 pin**

**void main(void)**

**{ init\_timer();**

**\_\_asm CLI;**

**/\* enable leds \*/**

**DDRJ = 2;**

**DDRB = 0xff;**

**PTJ = 0;**

**PORTB = 0;**

**for(;;){}**

**}**

**#pragma CODE\_SEG NON\_BANKED**

**void interrupt (((0x10000-Vtimch2)/2)-1) TIMCH2\_ISR(void)**

**{**

**if (TCTL2\_OL2==1) {**

**TC2 += 900;**

**TCTL2\_OL2=0;**

**TFLG1 =TFLG1\_C2F\_MASK; } //reset Ch 2 interrupt**

**else if (TCTL2\_OL2==0) {**

**TC2 += 2100;**

**TCTL2\_OL2=1;**

**TFLG1 =TFLG1\_C2F\_MASK; } //reset Ch 2 interrupt**

**//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**

**} //reset Ch 2 interrupt**

At first, the provided code is run and the results are observed. Then, with the help of the oscilloscope, the signal is examined to test the requirements with the output taken at PTT2 pin. The code is then modified by changing the number of cycles to determine the increment values and the toggling mechanism. The desired values to be put in the code are computed as follows:

**24 MHz / pre-scalar (8) = 3 MHz**

**3MHz = 1kHz \* TC TC = 3000**

**To get 30% duty cycle Pulse Signal 3000\*0.3 = 900**

**The remaining is 3000 – 900 = 2100 (%70)**

The code is run on the board, and the frequency is observed with the desired duty cycle.

# 3. Analysis and Interpretation

In the first task, the code of the timer provides a delay in time between each counting number using a set of registers, such as TSCR1. The register enables the timer when it is 1 and TSCR2 register sets the prescaler factor. Also the rate of toggling (1/T) is measured using the oscilloscope when applying a different prescaler value. This step shows that the rate of toggling changes due to the change of the value of the prescaler. In other words, when increasing the prescaler values, the values of the rate toggling decrease.

In the second task, the code of the output compare provides a 1 KHz 50% cycle duty. It is measured by the equation: **(No. of cycles (ON/OFF) = (24MHz \* period [ON/OFF]) /pre-scalar)** to find the period. By substituting the given terms, the 1 KHz 30% duty cycle pulse signal is obtained.

# 4. Assignment questions

**When using two output compare channels we can generate two different pulse waveforms. Write a program that produces two signals with different frequencies. The program should output a 50-Hz square wave on Port T bit 7 with a duty cycle based of 5% and a 300-Hz square wave on Port T bit 0 with a duty cycle of 5%. The duty cycles should be incremented by a 5% each time you press SW5 (use a PTH based interrupt) (Show your register settings and calculations).**

**Calculations for 50-Hz square wave:**

TC= 3MHz/50 Hz= 60 000

When ON:

TC\*0.05 = 3000

When OFF:

TC\*0.95= 57000

**Calculations for 300-Hz square wave:**

TC= 3MHz/300 Hz= 10

When ON:

TC\*0.05= 500

When OFF:

TC\*0.95= 9500

**Using Input Capture on channel PTT2 write a program that will measure the ON time and OFF time of the signals produced in the previous question and displays those on the LCD. Connect the signals to PTT2 and provide snap shots for the LCD screen.**

**#include <hidef.h> /\* common defines and macros \*/**

**#include "derivative.h" /\* derivative-specific definitions \*/**

**void main(void)**

**{**

**unsigned int count\_0;**

**unsigned int count\_7;**

**DDRH=0;**

**TSCR1 = 0x90;**

**TIOS = 0b10000001; //Output Compare at Channel 5**

**TCTL1 = 0b01000100; //Toggle PT7**

**TCTL2 = 0b00000001; //Toggle PT0**

**TSCR2 = 0x07;**

**for(;;)**

**{count\_0 = TCNT;**

**count\_7 = TCNT;**

**count\_0 = count\_0 + 348; //270 Hz square wave**

**count\_7 = count\_7 + 1875; // 50 Hz square wave**

**TC0 = count\_0;**

**TC7 = count\_7;**

**while (!(TFLG1 & TFLG1\_C5F\_MASK));}**

**}**

# 5. Conclusions and Recommendations

To conclude, one of the objectives of this laboratory session was to create an understanding of timer overflows and their implementations. In addition to that, implementing interrupts and time overflows is now clearer thanks to the tasks of this laboratory session.