Real Time Control Using Microcontroller Timer Units Plus Interfacing an LCD Display and Keypad

Team Members:

Tatiana Jiselle Ensslin
Gengxiao Li

Table of Contents

Introduction

This lab report presents an application of the Freescale HCS12XS microcontroller, its output compare mode, as well as how to interface the microcontroller with an external keypad and LCD display located on an Altera DE2 board. Using timers from the hardware component of the Freescale HCS12XS, one is able to learn through this lab how to waste an exact amount of time. This practical application carries thorough to the rest of the implementation of the experiment. In the second section of the lab, one will be able to use the 40-pin header in conjunction with the microcontroller's header to program a small piece of software that will implement a Hello World, justified with one's name underneath on the second line. This can be completed through the use of the microcontroller's hardware timer. This use of the timer than continues to traverse through the experiment, as it is needed for implementing a keypad through to the LCD, using the microcontroller to complete its protocols as well. By the end of the experiment, one will be able to understand a 4x4 keypad, how to interface it with the LCD, and how to use the microcontroller to combine all of these things to create a working reaction timing game!

As mentioned, throughout this lab we will explore the abilities of the HCS12XS. This experiment begins by identifying the mode of operation on the microcontroller called Output Compare Mode. This mode is used to connect the Timer Unit Hardware directly to an I/O port. We configure the timer to create square waves with specific frequency, which represents a delay of time as its period. After careful inspection of the microcontroller and its ability to connect the timer hardware to an I/O port, attention is moved to configuring a sentence and name on the Altera DE2 board's LCD display. This connects the 40-pin header to the microcontrollers header, allowing us to learn how to control the LCS using the HCS12. The next component explored using both the LCD display and the microcontroller is learning the construction of a 4x4 keypad, and how to integrate it with the microcontroller to be able to directly show the buttons characters on the LCD display when pressed. After exploring the ability to connect the keypad to the LCD display with the microcontroller, the remainder of the lab uses the ability to work the keypad and LCD display into a software/hardware game, where a user is able to test their reaction time by pressing two given random key pad buttons, and upon a correct input, receiving their reaction time. The code used in both Altera (VHDL) and CodeWarrior (C++) in the experiment can be found in the appendix.

Methods

Microcontroller Timers in Output Compare Mode

For the beginning of this experiment, start by looking up the datasheet for the Freescale HCS12XS microcontroller. Study and analyze the registers in the Timer Hardware Unit. In this section, we will use several timer registers to be able to hook up the timer hardware to the I/O ports. Some of the registers we will be implementing in this section are the TIOS, CFORC, TSCR1, TCTL1/TCTL2, TFLG, as well as the I/O ports, which are needed for implementation. On the HCS12 microcontroller the I/O Port that can be controlled by the timer is Port T, although note that the timer can be used without conjunction to the I/O Ports . Two registers in the Timer Hardware Unit are the timer counter register (TCNT), and the output compare register (TCx). There are 8 output compare registers, TC0-TC7, so that 8 different hardware timers can be functioning at the same time. The basic premise behind the timer in the output compare

mode is based on a hardware compare circuit that implemented a flag when TCNT equals TCx. There are steps that are used in the implementation of the timer to allow one to create enough of a delay in the frequency to create a square wave. They are as follows:

- Set up the timer by writing to the timer control registers at the beginning of the program.
- Read the 16-bit timer counter register TCNT and add a delay.
- Place the result into the output compare register TCx.
- As an example: TC2 = TCNT + 0x3A58, which configured a delay of 14,936 timer counts on timer 2.
- Wait for the flag to be set. While(!TFLG1_C2F).
- Take note that when creating a repeating delay, the delay should be added to the output compare register. TC2=TC2+ 0x3A58. This will eliminate any timing errors that will be associated with overhead due to software delays.

Next, configure the timer to implement a square wave with a frequency in Hertz that is equal to the day of the month you or your partner was born in. Begin by setting the I/O Pin equal to 1 or 0 through a software command (for example: PTT_PTT2=1). Next, configure the timer to automatically flip the I/O Pin by using the hardware toggle option (TCTLx). After that is set, verify that you have the correct frequency using the oscilloscope or logic analyzer. Continue by repeating the experiment, yet this time setting the frequency to your birth month times 10 in kHz. Observe both the software and hardware generated waveforms in comparison through the oscilloscope and see if you can observe any differences (jitter in the wave). Be sure to keep a detailed record of your findings throughout this experiment, including pictures and explanations of your results.

Hello World on the LCD

Next, we begin this section by locating the LCD on the Altera DE2 board. Note that we will be using the microcontroller's pin header to create a small VHDL circuit that connects the LCD pins to the 40-Pin header on the Altera board. This will enable the ability to control the LCD using the microcontroller. IN order to control the LCD through the microcontroller, note that it is important to continue to use the timer component in order to properly complete the LCD display's protocols. Information about the LCD display can be found in the HITACHI HD44780U datasheet on pages 24 and 43. There is both a 4-bit mode and 8-bit mode for the LCD display. We will focus on the 8-bit mode. These two pages on the datasheet will help in interfacing the LCD with CodeWarrior. Using this information, write a small piece of software that will display Hello World on the first line of the LCD. Once you understand how to send data to the LCD display, format the LCD screen to display Hello World left justified on the tip line, followed by you and your partner's name center justified on the second line.

Interface a Keypad

Next in the experiment, you will be given a 4x4 keypad. This keypad is simply 16 switches controlled on the back with 8 wires. When a button is selected, the two wires create a connection by touching each other. This allows for a completed circuit to occur if power is sent. There are several algorithms that can be used to interface the keypad with the microcontroller. Discuss one with your partner and write a piece of software to detect which key is pressed on the keypad and then display the corresponding character on the LCD display through VHDL and the Altera board. Note that it is important to continue to use the timer component in order to properly complete the LCD display's protocols in conjunction with sending

the character once pressed. For this section, you will be using both full rows of 8 X Ports on the header. Be sure that you can demonstrate your work.

Write a Simple Game

For this section, leave the keypad and Altera board connected to the microcontroller. Start by programming on CodeWarrior a piece of software that will begin the game by showing "Press any key to begin" on the LCD. When a key on the keypad is pressed, the program should clear the screen and then display two of the 16-keypad buttons randomly on the first line of the LCD. The user should then press the two keys. The objective is to be able to see how ling it will take to press the two keys. This is not an Input Capture problem; timing resolution on the sub-microsecond scale is not necessary. Once the user has pressed the two keys in the proper order, the program should be able to display the time between the last character displayed on the LCD and the last character pressed on the keypad. The time should then be properly displayed the second line of the LCD to the nearest millisecond. The second line should be formatted to display "Reaction: xxx ms." The game should then restart after a short 5 to 10 second delay.

Results

Microcontroller Timers in Output Compare Mode

This section will focus on Output Compare Mode—also know as wasting an exact amount of time. This may be used to connect the Timer Unit Hardware directly to an I/O pin. On the HCS12, the I/O Port T can control the timer. The timer can also be implemented apart from I/O. There are two registers in the Timer Hardware Unit. They are the timer counter resiter (TCNT) and the output compare registers (TCx). There are 8 output compare registers (TC0-TC7), meaning 8 timers can be implemented at a time. When TCNT equals TCx, a flag is set. The steps we used in implementing the timer is as follows:

- Set up the timer by writing to the timer control registers at the beginning of the program.
- Read the 16-bit timer counter register TCNT and add a delay.
- Place the result into the output compare register TCx.
- As an example: TC2 = TCNT + 0x3A58, which configured a delay of 14,936 timer counts on timer 2.
- Wait for the flag to be set. While(!TFLG1_C2F).
- Take note that when creating a repeating delay, the delay should be added to the output compare register. TC2=TC2+ 0x3A58. This will eliminate any timing errors that will be associated with overhead due to software delays.

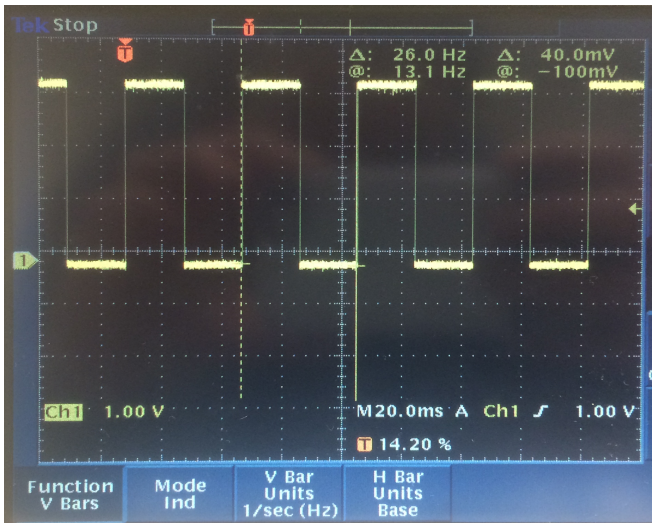
We configured the timer to create a square wave in 26Hz. First, we set the I/O pin to enable the data direction and then we configured the timer to automatically flip the I/O pin using the toggle option. The frequency will be verified via the oscilliscope. To find the delay in time for the 26Hz:

$6x10^6$ / 128 = 46875Hz 46875Hz/26Hz = 1802.9 1802.9/2 = 0x386 (note: 128 is prescale: 111)

SPEEDUP-Note that because its software, when the program is halted, the square wave should stop entirely. In comparison to hardware, where the wave will continue slowed down.

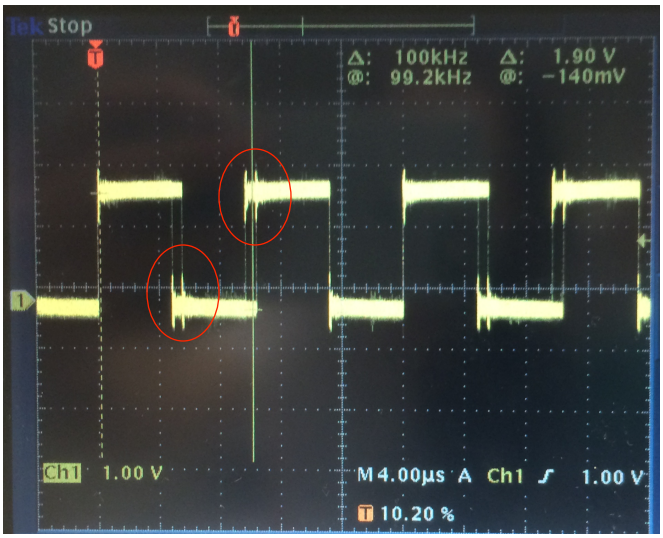
To speed up to 100kHz:

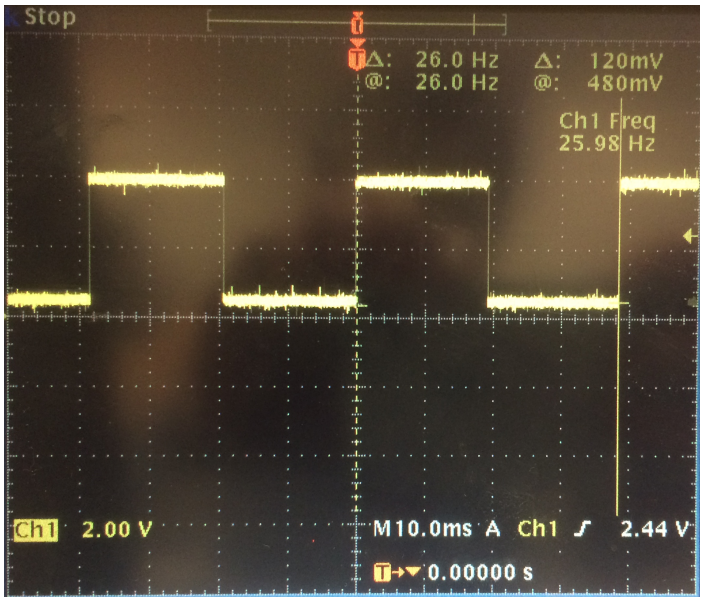
$6x10^6$ / 100kHz = 60 60/2=30 with no prescale. This correlates to about 9.96 micro seconds.



To the left is the software created square wave going at 26Hz.

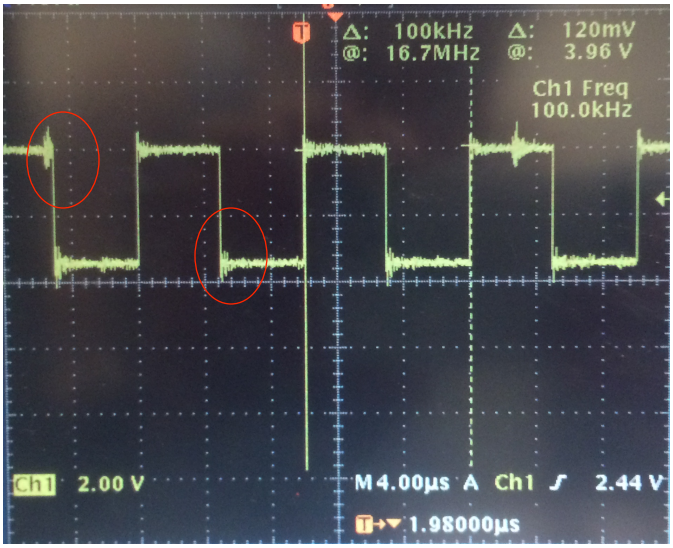
To the right is the software created square wave going at a speed of 100kHz. Note the jitter on the wave.





To the left is the 26Hz swuare wave created through the hardware

To the rigth is the 100kHz square wave created by the hardware. Note the jitter on the wave.



Note the jitter which can be seen on the faster 100kHz square waves. Note that we had to set the prescaler to divide by 128 for our delay time to accurately slow down the frequency on the 26Hz . The prescaler was removed for the 100kHz frequency. The code for this section can be found in the appendix.

Hello World on the LCD

The LCD is located on the Altera DE2 board. We will create a small VHDL circuit that simple connects the LCD pins to the 40-pin header. This will allow us to control the LCd using the HCS12 microcontroller. Using interfacing information about the LCD, we will write a small piece of software that will display "Hello World" on the LCD. In order to make sure we can format the LCD screen, we will display Hello World left justified, along with showing Tatiana and GX centered on the second line. We set two P Ports and 8 AD ports for this section. We will use the timer and set it to 90 for a 49 microsecond delay, and 300 for a 1.65 ms delay. This means we need to keep the prescale set to all 0's. There are two outputs we will be coding on the Altera in VHDL, the LCD_ON and LCD_RW. Table 6 (seen right) on the LCD microcontroller's data sheet contains the list of signals that can be used to send instructions to the LCD. We will use its 8-bit mode. The process of sending our data to the LCD is as

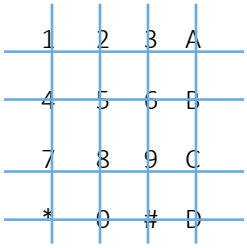
Table 6 Instructions

	Code											
Instruction	**RS**	**R/$\overline{W}$**	**DB7**	**DB6**	**DB5**	**DB4**	**DB3**	**DB2**	**DB1**	**DB0**	**Description**	**Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)**
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs

follows: Connect power. Set function set (tells what bits to set). Display on control (set to enable). Set entry mode to 8-bits. Write data in ASCII. Set DDR address (this shifts the cursor to the second line at address 0x064 (if set to that address)). We use VHDL code with 11 inputs and 11 outputs. This will allow us to connect the 40-pin header to out LCD display. To center the 40-Pin header to out LCD display we sent the function set, display control, entry mode set, then "Hello world", then set the DDR address instruction. We used 0xC4 to center our names (position) them on the second line. The code for this section can be found in the appendix.

Interface a Keypad

We were given a 4x4 keypad, which is simply 16 switches connected by 8 wires. We will create an algorithm that will implement knowing the pressed button and display it to the LCD. The back of the keypad looks like:



We will have for inputs and four outputs. We will also set a pull enable and polarity register on our ports in order for all the ports to be initialized to 0. Our algorithm will register a '1==1,' which will then print that selected character. We will use an 8-pin header to connect the keypad to our controller. To do this, we set OCPD (enable T ports for non-timer use) to 0xFF and use the T ports to connect the keypad. We will set our rows as out and our columns as in. T0-T3 are outputs and T4-T7 are our inputs. We now enable the pull registers, which will set all the T ports to '0.' We also created multiple functions to clean and ease the code, such as an enable function which sets the enable bit to '1' then '0,' a cleardisplay functions, as well as a writedata function which intakes a character to be printed out. These functions and others include a special delay calculated for the keypad between buttons. Our VHDL code remained the same from the last part. Note that in our code, we clear the LCD screen before changing to the next character. When using the keypad, the wires can have bounce, which is due to the 8 wires crossing unintentionally when a button is pressed. This can be resolved through a small delay, which allows for the full '1' of the button press to be registered. The code for this section can be found in the appendix.

To the left are screenshots of two of the characters being displayed properly on the LCD display after being pressed on the keypad.

Write a Simple Game

This game started by displaying "Press any key to begin." When a key on the keypad is pressed the program should clear the screen and then display two of the 16 keypad characters randomly on the first line of the LCD. The user will then press the two characters. Upon correct input, the microcontroller will then capture the keys and display the time between the last character displayed on the LCD and the last character pressed on the keypad. The time will then display on the second line of the LCD to the nearest ms, saying "Reaction xxx ms." The game should reset after a 5-10 second delay. For the counter, we set TCNT to a variable and when the two buttons are finished being properly inputted, saved TCNT to another variable. We then found the difference from those variables. After, we converted this difference to milliseconds and then to seconds (x1000), and the frequency to time/clock cycles. 1/6,000,000 is the time for 1 clock cycle. 2^{16} (size of TCNT) –1/6,000,000. Therefore, with our timer prescaler, we divide the reaction time x1000 by 46875 to get out time to display in ms. After this, we get the time that we would like to display on the LCD, however, in order for it to be displayed, it needs to be modded by 10 in order to be split up by places to be printed because the number is longer than 1 character. Unfortunately, due to a byte compile error restriction, we were unable to debug past this stage in our code and completed the remainder of the section. We tried splitting the code into a .n and .c file, and creating a new project to load them in, however we encountered link errors when trying to do so. Although we were unable to finish, you can see out unfinished code in the appendix.

Discussion

This lab has provided us with a good understanding of what a microcontroller is capable of doing through its ports and with its timer hardware component. The timer Output Compare Mode, is extremely useful in being able to communicate with the LCD display, for it allows us to insert delays between instructions being sent to the LCD microcontroller. Once again, we learn that the microcontroller has multiple capabilities for communication protocols that can be useful when programming a complex device/project. The microcontroller itself contains several programmable registers, which allow the microcontroller to preform different actions on its ports. This principle is apparent in and throughout this lab. We used the Port T in the beginning as a I/O for the timer's output compare, however, we broke it off of the hardware and used it in the third section to program the keypad in conjunction with the microcontroller and the LCD display. We used all 8 AD Ports in the second section to connect the LCD display to the microcontroller, as well as using two M Ports in order to control the instructions being sent over to the LCD display. The most difficult part of this experiment was being able to distinguish the prescaler's size in order to accurately get the precise time needed for each function to allow it to implement to the best performance. Additionally, it was extremely frustrating to experience a delay that was too long on the keypad's algorithm and see how it created a delay on the LCD display. Through trial and error, we were eventually able to figure out the cause of our errors in our timing by using the debugging tool to correctly identify the proper registers and logic to successfully complete the section. Lastly, although we were unable to finish the reaction game in the last section, we were able to complete most of it and realize that section truly tested our knowledge of being able to use the timer hardware, as

well as properly implement the logic to create a working game. Unfortunately because of the byte limit, our design was really pressured to take strange shapes, and then eventually unable to even be completed. Although we couldn't finish, to see how to interface the keypad in terms with both the LCD's microcontroller and the HCS12 microcontroller, this experiment was by far one of the best of the semester.

Although we did encounter a few challenges while working through this lab, such as selecting incorrect registers to program, or picking an un-programmable port (in our Hello World section, we used the T ports originally, which caused an interference with the timer, which was using the port T0), our initial approach to implementing the microcontroller's hardware unit in terms with external hardware was primarily correct. Through use of the datasheet, header file in the CodeWarrior, and knowledge about the communication protocols for both microcontrollers, we found the use of the timer hardware component to be critical to being able to fully implement larger projects and experiments.

Appendix A – Microcontroller Timers in Output Compare Mode

Software SLOW (26Hz):

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

void main(void) {
      TIOS_IOS0 = 1; //enable output compare for timer0
      TSCR1_TEN = 1; //enable main counter/timer
      TSCR1_TFFCA = 1; //enable flag to clear
      TSCR2_PR0=1; // divide the prescaler 128
      TSCR2_PR1=1;
      TSCR2_PR2=1;
      OCPD = 1;

TC0=TCNT + 0x386;  //26Hz in clock cycles
      DDRT_DDRT0 =1;
      PTT_PTT0 =1;
while(1){
       TC0=TC0 + 0x386;

       while(!TFLG1_C0F){}
          PTT_PTT0 =!PTT_PTT0;
       }
}
```

Software FAST (100kHz):

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

void main(void) {
      TIOS_IOS0 = 1; //enable output compare for timer0
      TSCR1_TEN = 1; //enable main counter/timer
      TSCR1_TFFCA = 1; //enable flag to clear
      TSCR2_PR0=0; // divide the prescaler 128
      TSCR2_PR1=0;
      TSCR2_PR2=0;
      OCPD = 1;

      TC0=TCNT + 30;  //26Hz in clock cycles
      DDRT_DDRT0 =1;

      PTT_PTT0 =1;
```

```
      while(1){
        TC0=TC0 + 30;
        while(!TFLG1_C0F){}
        PTT_PTT0 =!PTT_PTT0;
   }
}
```

Hardware SLOW (26Hz):

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

void main(void) {
      TIOS_IOS0 = 1; //enable output compare for timer0
      TSCR1_TEN = 1; //enable main counter/timer
      TSCR1_TFFCA = 1; //enable flag to clear
      TTOV_TOV0 = 0x01;

      TCTL1_OL4 = 1;
      TCTL1_OL5 = 1;
      TCTL1_OL6 = 1;
      TCTL1_OL7 = 1;
      TCTL2_OL0 = 1;
      TCTL2_OL1 = 1;
      TCTL2_OL2 = 1;
      TCTL2_OL3 = 1;

      TSCR2_PR0=1; // divide the prescaler 128
      TSCR2_PR1=1;
      TSCR2_PR2=1;

      TC0=TCNT + 0x386;  //26Hz in clock cycles
      DDRT_DDRT0 =1;
      PTT_PTT0 =1;

      while(1){
        TC0=TC0 + 0x386;

       while(!TFLG1_C0F){}
   }
}
```

Hardware FAST (100kHz):

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

void main(void) {
      TIOS_IOS0 = 1; //enable output compare for timer0
      TSCR1_TEN = 1; //enable main counter/timer
      TSCR1_TFFCA = 1; //enable flag to clear
      TSCR2_PR0=0; // divide the prescaler 128
      TSCR2_PR1=0;
      TSCR2_PR2=0;
      OCPD = 1;

      TC0=TCNT + 30;  //26Hz in clock cycles
      DDRT_DDRT0 =1;
      PTT_PTT0 =1;

      while(1){
       TC0=TC0 + 30;

       while(!TFLG1_C0F){}
         PTT_PTT0 =!PTT_PTT0;
```

```
    }
}
```

Appendix B – Hello World on the LCD

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

void enable();

void main(void) {
   DDR1AD0=0XFF;
   DDRM_DDRM2=1;
   DDRM_DDRM3=1;

//Timer Initializations
   TIOS=0XFF;
   TCTL2=0X10;
   TSCR1_TEN=1;
   TSCR1_TFFCA=1;
    DDRT=0xFF;
   TSCR2=0x03;
   OCPD_OCPD2=1;

       TC0=TCNT + 300; //49mic delay
      while(!TFLG1_C0F){}

       TC0=TCNT + 300; //49mic delay
      while(!TFLG1_C0F){}

       //clear screen
        PT1AD0=0X01;//CLEAR DISPLAY

   PTM_PTM3=0;  // RS
    PTM_PTM2=1;  // SELECT DATA
    PTM_PTM2=0;

      TC0=TCNT + 300; //49mic delay

      while(!TFLG1_C0F){}
      TC0=TCNT + 300; //49mic delay

      while(!TFLG1_C0F){}
      TC0=TCNT + 300;  //49microsecond delay

      //functon set
          PT1AD0 = 0x3F;
          PTM_PTM3 = 0;
          enable();

          TC0=TCNT + 300; //49mic delay
      while(!TFLG1_C0F){}

                //display on
           PT1AD0 = 0x0E;
           PTM_PTM3 = 0;
           enable();

          TC0=TCNT + 300; //49mic delay
      while(!TFLG1_C0F){}

       //entry mode
         PT1AD0 = 0x06;
         PTM_PTM3 = 0;
         enable();
```

```
    TC0=TCNT + 300; //49mic delay
while(!TFLG1_C0F){}

   TC0=TCNT + 300; //49mic delay
while(!TFLG1_C0F){}

   //write data
   PT1AD0 = 'H';
    PTM_PTM3 = 1;
    enable();

TC0=TCNT + 300; //49mic delay
while(!TFLG1_C0F){}

  //write data
    PTM_PTM3 = 1;
    PT1AD0 = 'E';
    enable();

TC0=TCNT + 300; //49mic delay
while(!TFLG1_C0F){}

  //write data
    PTM_PTM3 = 1;
    PT1AD0 = 'L';
    enable();

TC0=TCNT +300; //49mic delay
while(!TFLG1_C0F){}

//write data
    PTM_PTM3 = 1;
    PT1AD0 = 'L';
    enable();

TC0=TCNT + 300; //49mic delay
while(!TFLG1_C0F){}

  //write data
    PTM_PTM3 = 1;
    PT1AD0 = 'O';
    enable();

TC0=TCNT + 300; //49mic delay
while(!TFLG1_C0F){}

  //write data
    PTM_PTM3 = 1;
    PT1AD0 = ' ';
    enable();

TC0=TCNT + 300; //49mic delay
while(!TFLG1_C0F){}

  //write data
    PTM_PTM3 = 1;
    PT1AD0 = 'W';
    enable();

TC0=TCNT + 300; //49mic delay
while(!TFLG1_C0F){}

  //write data
    PTM_PTM3 = 1;
    PT1AD0 = 'O';
```

```
  enable();

TC0=TCNT + 300; //49mic delay
while(!TFLG1_C0F){}

 //write data
   PTM_PTM3 = 1;
   PT1AD0 = 'R';
   enable();

TC0=TCNT + 300; //49mic delay
while(!TFLG1_C0F){}

 //write data
   PTM_PTM3 = 1;
   PT1AD0 = 'L';
   enable();



TC0=TCNT + 300; //49mic delay
while(!TFLG1_C0F){}

   //write data
   PTM_PTM3 = 1;
   PT1AD0 = 'D';
   enable();

 TC0=TCNT + 300; //49mic delay
while(!TFLG1_C0F){}
 TC0=TCNT + 300; //49mic delay
while(!TFLG1_C0F){}

  //set DDRAM
  PT1AD0 = 0xC4;  //set to 1st line far left
   PTM_PTM3 = 0;
   enable();

    TC0=TCNT + 300; //49mic delay
while(!TFLG1_C0F){}

    //write data
    PT1AD0 = 'T';
   PTM_PTM3 = 1;
   enable();

    TC0=TCNT + 300; //49mic delay
  while(!TFLG1_C0F){}

    //write data

   PTM_PTM3 = 1;
   PT1AD0 = 'A';
   enable();

 TC0=TCNT + 300; //49mic delay
 while(!TFLG1_C0F){}

    //write data
   PTM_PTM3 = 1;
   PT1AD0 = 'T';
   enable();

 TC0=TCNT + 300; //49mic delay
while(!TFLG1_C0F){}
```

```
          //write data
         PTM_PTM3 = 1;
         PT1AD0 = ' ';
         enable();

         TC0=TCNT + 300; //49mic delay
        while(!TFLG1_C0F){}

         //write data
         PTM_PTM3 = 1;
         PT1AD0 = 'G';
         enable();

         TC0=TCNT + 300; //49mic delay
        while(!TFLG1_C0F){}
         //write data
         PTM_PTM3 = 1;
         PT1AD0 = 'X';
         enable();

     TC0=TCNT + 300; //49mic delay
      while(!TFLG1_C0F){}

     while(1){
     }
}

void enable(){
    PTM_PTM2 = 1; //this is the enable for the LCD
    PTM_PTM2 = 0; //this is the enable for the LCD
}

================================================
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity LCD is
port (
                        t0 : in std_logic;
                        t1 : in std_logic;
                        t2 : in std_logic;
                        t3 : in std_logic;
                        t4 : in std_logic;
                        t5 : in std_logic;
                        t6 : in std_logic;
                        t7 : in std_logic;
                        p0 : in std_logic;
                        outa : out std_logic;
                        outb : out std_logic;
                        outc : out std_logic;
                        outd : out std_logic;
                        oute : out std_logic;
                        outf : out std_logic;
                        outg : out std_logic;
                        outh : out std_logic;
                        out1 : out std_logic;
                        enable : in std_logic;
                        LCDon : out std_logic;
                        out2 : out std_logic;
                        out3 : out std_logic

            );
end entity;

architecture behave of LCD is
```

```
        BEGIN
        LCDon <= '1';
        out3 <= enable; --P1
        out1 <= p0; --RS
        out2 <= '0'; --rw
        outa <= t0; --hex byte
        outb <= t1;
        outc <= t2;
        outd <= t3;
        oute <= t4;
        outf <= t5;
        outg <= t6;
        outh <= t7;


end architecture;
```

Appendix C – Interfacing a Keypad

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

void enable();
void writeData(unsigned char letter);
void clearDisplay();
void Delay();

void main(void) {

  // port initializations
   DDR1AD0=0xFF;
   DDRM_DDRM2=1;
   DDRM_DDRM3=1;
   DDRT=0xF0; //set t0-t3 as inputs and t4-t7 as outputs
   PERT = 0x0F; //set the pull enable all to 0
   PPST = 0x0F; //polarity
   OCPD=0xFF;   //enable T ports as i/o

   //Timer Initializations
   TIOS=0xFF;
   TCTL2=0x10;
   TSCR1_TEN=1;
   TSCR1_TFFCA=1;
   TSCR2=0x03;

  //send beginning instructions
      clearDisplay();

      while(!TFLG1_C0F){}
      TC0=TCNT + 300;  //49microsecond delay
      //functon set
          PT1AD0 = 0x3F;
          PTM_PTM3 = 0;

         enable();
          TC0=TCNT + 300; //49mic delay
      while(!TFLG1_C0F){}


       //display on
           PT1AD0 = 0x0E;PTM_PTM3 = 0;

         enable();
```

```
      TC0=TCNT + 300; //49mic delay
    while(!TFLG1_C0F){}



     //entry mode
       PT1AD0 = 0x06;
       PTM_PTM3 = 0;
       enable();

    TC0=TCNT + 300; //49mic delay
    while(!TFLG1_C0F){}

     TC0=TCNT + 300; //49mic delay
    while(!TFLG1_C0F){}


  // end beginning instructions

//=================================================================
   //algorithm to recognize letter
  int i =0;
   for(;;){

   for(int i = 0; i <4; i++){


    if(i==0){
      PTT_PTT4=1;
      PTT_PTT5=0;
      PTT_PTT6=0;
      PTT_PTT7=0;

      Delay();

        if(PTT_PTT0==1) {
         clearDisplay();
           writeData(0x44); //1
      }

       Delay();

        if(PTT_PTT1==1) {
         clearDisplay();
           writeData(0x23); //4
      }

       Delay();

        if(PTT_PTT2==1) {
         clearDisplay();
           writeData(0x30); //7
      }

       Delay();

        if(PTT_PTT3==1) {
         clearDisplay();
           writeData(0x2A); //x
        }
       Delay();
    }


    if(i==1){
      PTT_PTT4=0;
```

```
  PTT_PTT5=1;
  PTT_PTT6=0;
  PTT_PTT7=0;
    if(PTT_PTT0==1) {
     clearDisplay();
       writeData(0x43);//send 2
   }

    if(PTT_PTT1==1) {
     clearDisplay();
       writeData(0x39);//send5
   }

    if(PTT_PTT2==1) {
     clearDisplay();
       writeData(0x38);//send8
   }

    if(PTT_PTT3==1) {
     clearDisplay();
      writeData(0x37); //send 0
   }
 }


if(i==2){
  PTT_PTT4=0;
  PTT_PTT5=0;
  PTT_PTT6=1;
  PTT_PTT7=0;

    if(PTT_PTT0==1) {
     clearDisplay();
      writeData(0x42);//send 3
   }

    if(PTT_PTT1==1) {
     clearDisplay();
      writeData(0x36); //send6
   }

    if(PTT_PTT2==1) {
     clearDisplay();
       writeData(0x35);//send9
   }

    if(PTT_PTT3==1) {
     clearDisplay();
       writeData(0x34);//send #
   }
  }

if(i==3){
  PTT_PTT4=0;
  PTT_PTT5=0;
  PTT_PTT6=0;
  PTT_PTT7=1;
          Delay();
    if(PTT_PTT0==1) {
     clearDisplay();
       writeData(0x41);//send A
   }
         Delay();
    if(PTT_PTT1==1) {
     clearDisplay();
       writeData(0x33);//sendB
```

```
        }

         if(PTT_PTT2==1) {
          clearDisplay();
            writeData(0x32);//sendC
        }

         if(PTT_PTT3==1) {
          clearDisplay();
            writeData(0x31);//send D
        }

     }
     }
    }
}

void clearDisplay(){
    TC0=TCNT + 300; //49mic delay
       while(!TFLG1_C0F){}


        //clear screen
         PT1AD0=0X01;//CLEAR DISPLAY
   PTM_PTM3=0;  // RS
    PTM_PTM2=1;  // SELECT DATA
    PTM_PTM2=0;

}

void enable(){
      PTM_PTM2 = 1; //this is the enable for the LCD
      PTM_PTM2 = 0; //this is the enable for the LCD
}

void writeData(unsigned char letter){
      TSCR2_PR0=0;
      TSCR2_PR1=1;
      TSCR2_PR2=0;

       TC0=TCNT + 6760; //49mic delay
       while(!TFLG1_C0F){}

           //write data
          PTM_PTM3 = 1;
          PT1AD0 = letter;
          enable();

}

void Delay(){
      TSCR2_PR0=0;
      TSCR2_PR1=1;
      TSCR2_PR2=1;
 TC0=TCNT + 0x2497;  //delay
 while(!TFLG1_C0F){}
}
```

Appendix D – Reaction Game

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

void enable();
void writeData(unsigned char letter);
```

```
void clearDisplay();
void Delay();
void reposition(unsigned char position);  //set DDgram
void writeData2(unsigned char letter);
void run();
void welcome();
void sendInitInstructions();
void generateSecretNumbers();
//void pressAnyButton(); //begins game
void findButton(); //finds and prints when a button is pressed

int num;
int num2;
unsigned char symbols[16]={0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x30,0x2A,0x23,0x41,0x42,0x43,0x44};
unsigned char answer1;
unsigned long int time1;
unsigned long int time2;
unsigned long int reactionTime;
unsigned long int a,b,c,d;

void main(void) {

  // port initializations
   DDR1AD0=0xFF;
   DDRM_DDRM2=1;
   DDRM_DDRM3=1;
   DDRT=0xF0; //set t0-t3 as inputs and t4-t7 as outputs
   PERT = 0x0F; //set the pull enable all to 0
   PPST = 0x0F; //polarity
   OCPD=0xFF;   //enable T ports as i/o

   //Timer Initializations
   TIOS=0xFF;
   TCTL2=0x10;
   TSCR1_TEN=1;
   TSCR1_TFFCA=1;
   TSCR2=0x03;



  //send beginning instructions
  sendInitInstructions();

 /* generate secret number between 1 and 10: */
 generateSecretNumbers();

 //Welcome the player to the game
 welcome();
 //pressAnyButton();

  for (int k; k<5000; k++){

  PTT_PTT4=1;
 if (PTT_PTT0 == 1 && PTT_PTT4 == 1)
        run();
  }

 //start timer here
  time1= TCNT;
    findButton();
}
```

```
//============================end of main=====================================//

void run(){
  clearDisplay();
  writeData(symbols[num]);
  writeData(symbols[num2]);
}

void clearDisplay(){
    TC0=TCNT + 300; //49mic delay
      while(!TFLG1_C0F){}


       //clear screen
        PT1AD0=0X01;//CLEAR DISPLAY
   PTM_PTM3=0;  // RS
    PTM_PTM2=1;  // SELECT DATA
    PTM_PTM2=0;

}

void enable(){
     PTM_PTM2 = 1; //this is the enable for the LCD
     PTM_PTM2 = 0; //this is the enable for the LCD
}

void writeData(unsigned char letter){
     TSCR2_PR0=0;
     TSCR2_PR1=1;
     TSCR2_PR2=0;

      TC0=TCNT + 6760; //49mic delay
      while(!TFLG1_C0F){}

         //write data
        PTM_PTM3 = 1;
        PT1AD0 = letter;
        enable();

}

void Delay(){
     TSCR2_PR0=0;
     TSCR2_PR1=1;
     TSCR2_PR2=1;
 TC0=TCNT + 0x2497;  //delay
 while(!TFLG1_C0F){}
}

void reposition(unsigned char position){
     TSCR2_PR0=0;
     TSCR2_PR1=1;
     TSCR2_PR2=0;

      TC0=TCNT + 6760; //49mic delay
      while(!TFLG1_C0F){}
        PT1AD0 = position;  //set to 1st line far left
        PTM_PTM3 = 0;
        enable();

}

void writeData2(unsigned char letter){
  if(letter == symbols[num]){

       answer1 = letter;
```

```
        return;

  }
  if(answer1 == symbols[num]){
    if (letter == symbols[num2]){

      //end counter
      time2=TCNT;
      reactionTime = time2-time1;
      reactionTime = reactionTime * 1000;  // convert clock cycles to seconds
      reactionTime = reactionTime/46875;
      a = reactionTime % 10;        // convert number to single byte
      reactionTime/=10;
      b = reactionTime % 10;
      reactionTime/=10;
      c = reactionTime % 10;
      reactionTime/=10;
      d = reactionTime % 10;

       reposition(0xC2);
       writeData(0x52);//R
       writeData(0x65);//e
       writeData(0x61);//a
       writeData(0x63); //c
       writeData(0x74);//t
       writeData(0x69);//i
       writeData(0x6F);//o
       writeData(0x6E);//n
       writeData(0x3A);//:
       writeData(0x20);// #+0x30 convert to ascii
       writeData(d);      //send single byte
       writeData(c);
       writeData(b);
       writeData(a);
    }

}
}

void welcome(){

  writeData(0x50);//P
  writeData(0x72);//r
  writeData(0x65);//e
  writeData(0x73);//s
  writeData(0x73);//s
  writeData(0x20);//
  writeData(0x41);//A
  writeData(0x6E);//n
  writeData(0x79);//y
  writeData(0x20);//
  writeData(0x4B);//K
  writeData(0x65);//e
  writeData(0x79);//y
  writeData(0x20);//
  writeData(0x54);//T
  writeData(0x6F);//o
  reposition(0xC6);//next line
  writeData(0x42);//B
  writeData(0x65);//e
  writeData(0x67);//g
  writeData(0x69);//i
  writeData(0x6E);//n

}
```

```
void sendInitInstructions(){
      TSCR2=0x03;

     clearDisplay();

       while(!TFLG1_C0F){}
       TC0=TCNT + 300;  //49microsecond delay
       //functon set
            PT1AD0 = 0x3F;
            PTM_PTM3 = 0;

           enable();
            TC0=TCNT + 300; //49mic delay
       while(!TFLG1_C0F){}


        //display on
             PT1AD0 = 0x0E;PTM_PTM3 = 0;

           enable();
            TC0=TCNT + 300; //49mic delay
       while(!TFLG1_C0F){}



        //entry mode
           PT1AD0 = 0x06;
           PTM_PTM3 = 0;
           enable();

       TC0=TCNT + 300; //49mic delay
       while(!TFLG1_C0F){}

        TC0=TCNT + 300; //49mic delay
       while(!TFLG1_C0F){}
}

void generateSecretNumbers(){

  num = TC0 % 16;
  while(!TFLG1_C0F){}
  TC0=TCNT + 300; //49mic delay
  num2 = (TC0 +30) %16;

}

void findButton(){
        TSCR2=0x07;

  for(;;)  {

     for(int i = 0; i <4; i++){
       PTT_PTT4=0;
       PTT_PTT5=0;
       PTT_PTT6=0;
       PTT_PTT7=0;
     if(i==0){
       PTT_PTT4=1;

       Delay();

if(PTT_PTT0==1) {
            writeData2(0x44);
        }

        Delay();
```

```
    if(PTT_PTT1==1) {
          writeData2(0x23);
   }

   Delay();

    if(PTT_PTT2==1) {
          writeData2(0x30);
   }

   Delay();

    if(PTT_PTT3==1) {
          writeData2(0x2A);
    }
    Delay();
}


if(i==1){

  PTT_PTT5=1;

    if(PTT_PTT0==1) {
          writeData2(0x43);
   }

    if(PTT_PTT1==1) {
          writeData2(0x39);
   }

    if(PTT_PTT2==1) {
          writeData2(0x38);
   }

    if(PTT_PTT3==1) {
          writeData2(0x37);
   }
}


if(i==2){
  PTT_PTT6=1;
    if(PTT_PTT0==1) {
          writeData2(0x42);
   }

    if(PTT_PTT1==1) {
          writeData2(0x36);
   }

    if(PTT_PTT2==1) {
          writeData2(0x35);
   }

    if(PTT_PTT3==1) {
      writeData2(0x34);
   }
  }

if(i==3){

  PTT_PTT7=1;
          Delay();
    if(PTT_PTT0==1) {
```

```
                writeData2(0x41);
        }
                Delay();
         if(PTT_PTT1==1) {
                writeData2(0x33);
        }

         if(PTT_PTT2==1) {
                writeData2(0x32);
        }

         if(PTT_PTT3==1) {
                writeData2(0x31);
        }



     }
    }
  }
}
```