ECE 362 Lab Verification / Evaluation Form

Experiment 8

Evaluation:

IMPORTANT! You must complete this experiment during your scheduled lab period. All work for this experiment must be demonstrated to and verified by your lab instructor *before the end* of your scheduled lab period.

STEP	DESCRIPTION	MAX	SCORE
1	Interfacing (completed prior to your scheduled lab period)	**10**	
2	Software (completed by the end of your scheduled lab period)	**10***	
3	Submission (completed immediately following demonstration)	**5***	
4	Bonus Credit	**5**	
	TOTAL	**25+**	

** code must function as specified to receive full credit for software and submission scores (score for non-functioning code will be 20% of max for both software and submission)*

Signature of Evaluator: _____

Academic Honesty Statement:

IMPORTANT! Please carefully read and sign the Academic Honesty Statement, below. *You will not receive credit for this lab experiment unless this statement is signed in the presence of your lab instructor.*

"In signing this statement, I hereby certify that the work on this experiment is my own and that I have not copied the work of any other student (past or present) while completing this experiment. I understand that if I fail to honor this agreement, I will receive a score of ZERO for this experiment and be subject to possible disciplinary action."

Printed Name: _____ Class No. __ __ __ __ - __

Signature: _____ Date: _____

Experiment 8: Whatsamatta U Reaction Timer

Instructional Objectives:

- To illustrate how the TIM module can be used to implement a precision time base
- To illustrate how the SPI module can be used in conjunction with an external shift register to expand I/O

Parts Required:

- 2 x 16 LCD with 16-pin single-row header (DK-3 parts kit)
- GAL22V10 (DK-3 parts kit) configured as an 8-bit shift register
- RED, YELLOW, and GREEN LEDs plus current limiting resistors (DK-2 parts kit)
- Breadboard and wire
- BONUS CREDIT OPTION: 10K potentiometer (DK-2 or DK-3 parts kit)

Preparation:

- Read this document in its entirety
- Review the material on the TIM and SPI subsystems
- Complete the wiring outlined in the Hardware Overview

Introduction

Midway through the football season, Coach Boris Badenov of Whatsamatta U had an idea. Determined to be as quick as possible to challenge the "bad calls" he's sure will be made at the ***Battle for the Bit Bucket*** game with **Boo-Hoo-U**, he realized that a customized reaction timer could help him test and sharpen the swiftness of his protest skills. The interface, however, would have to be simple enough for a Healthcare.gov website contractor to use, which is why he has enlisted you (along with a bevy of benevolent boisterous Boilermakers) to construct a prototype for him. An illustration of the finished product is shown in Figure 1.

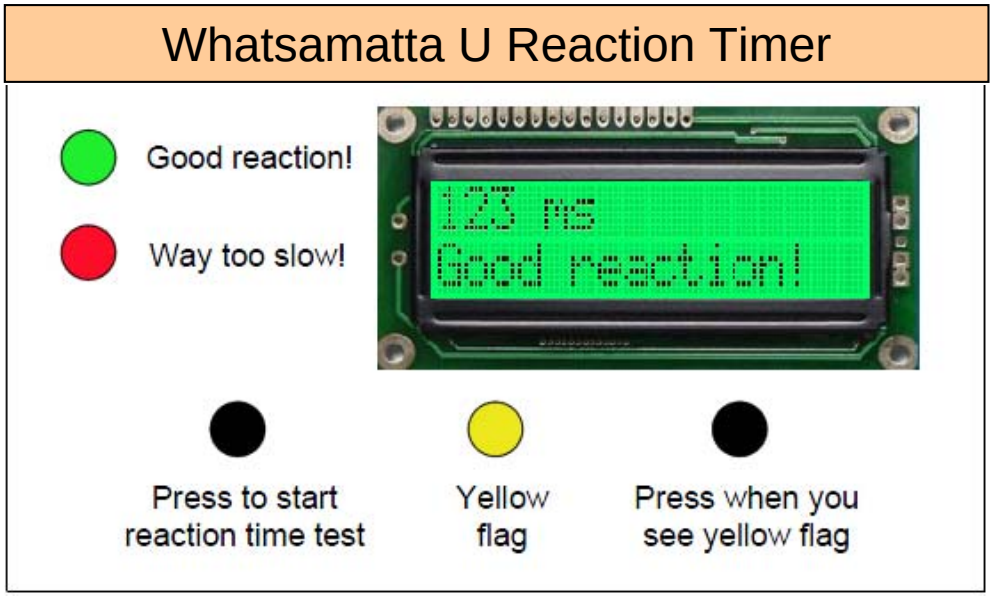


Figure 1. Reaction Timer Control Panel and Display.

Coach BB's reaction timer will use the docking module pushbuttons as follows: the left pushbutton starts the reaction test which will display the message "`Ready, Set…`" on the first line of the LCD and initiate a random delay. When this delay expires, "`Go Team!`" will be printed to the second line of the LCD, the yellow LED will be illuminated (simulating that a "yellow flag" has been thrown by the referee) and the reaction timer will start. The right pushbutton stops the test, at which point the reaction time will be displayed on the first line of the LCD in the format "RT = NNN ms" and an *appropriate* custom message will be displayed on the second line. Also, the YELLOW LED should be turned off and, if the resulting reaction time was less than 250 milliseconds, the GREEN LED should be turned on. If the reaction time was greater than 1 second, the RED LED should be turned on. The left LED on the docking module will be used to indicate the reaction time test is "stopped" (i.e., ready to start a new test), while the right LED will be used to indicate a reaction time test is "in progress".

In addition to the reaction time, *appropriate* messages (i.e., messages you wouldn't mind if your Mom saw) should be displayed on the second line of the LCD (e.g., "`Ready to start!`" upon reset, "`Go Whatsamatta U`" if a really fast reaction time is recorded, etc.). It should be noted that LCD messages should be limited to 16 characters due to the size of the screen. If, for some reason, the reaction time exceeds 999 milliseconds, an appropriate suggestion should be displayed on the second line of the LCD (e.g., "`Get HKN coffee!!`").

Lucky for Coach BB, there are plenty of underpaid and overworked ECE 362 students armed with 9S12C32 microcontroller kits willing to work for the opportunity to buy full-price ***Bit Bucket*** game tickets. Time's running out, though, so gather up your parts and get to work!

Figure 2. Coach Boris Badenov of Whatsamatta U and the Coveted Boiler Bit Bucket.

Hardware Overview

On the docking module, the left pushbutton will be used to initiate a reaction time test, while the right pushbutton will be used to stop the test in progress (and display the reaction time results). The left LED will be used to indicate the reaction timer is stopped (ready to start a new test), while the right LED will be used to indicate a reaction time test is in progress.

A GREEN LED on PTT[5] will be used to indicate a reaction time less than 250 milliseconds, a RED LED on PTT[6] will be used to indicate a reaction time greater than 999 milliseconds and a YELLOW LED on PTT[7] will indicate the start of the reaction time test.

An external 8-bit shift register (GAL22V10, programmed using the ABEL code given in the Module 2 homework) will be used to interface LCD to the microcontroller via the SPI module (MOSI, port pin PM[4]; and SCK, port pin PM[5]). The LCD will be interfaced as described to the microcontroller module as described in the table below:

LCD Pin #	LCD Pin Description	Connected to Microcontroller
1	Vss (ground)	Vss (ground)
2	Vcc (+5V)	Vcc (+5V)
3	VEE (contrast adjust)	Vss (ground)
4	R/S (register select)	PTT[2]
5	R/W' (LCD read/write)	PTT[3]
6	LCD Clock	PTT[4]
7	DB[0] (LS bit)	Q[0]
8	DB[1]	Q[1]
9	DB[2]	Q[2]
10	DB[3]	Q[3]
11	DB[4]	Q[4]
12	DB[5]	Q[5]
13	DB[6]	Q[6]
14	DB[7] (MS bit)	Q[7]
15	Not connected	
16	Not connected	

NOTE: DB[#] are the LCD data inputs and Q[#] are the data outputs of the GAL22V10 shift register.

Some of the LCD pins require a bit more explanation:

Mnemonic	Name	Description
RS	Register select	This pin is logic 0 when sending an instruction command over the LCD data bus and logic 1 when sending a character.
R/W'	Read/write	This pin is logic 0 when writing to the LCD, logic 1 when reading from it. For this lab, we will only write to the LCD.
LCDCLK	LCD clock	This pin latches in the data on the data[7:0] bus on the falling edge. Therefore, this line should idle as logic 1.

Software Overview

Timer Channel 7 will be used to provide periodic interrupts at precise 1.0 millisecond intervals for the purpose of gauging the reaction time. The Timer Channel 7 interrupt service routine should keep count of the elapsed time using the `react` variable provided in the skeleton file. This variable should be maintained in BCD format.

The RTI will be used to sample the pushbuttons on the docking board every 2.048 milliseconds as well as increment the `random` counter. Data for the LCD display will be shifted out to an external shift register (GAL22V10) and will be interfaced to the SPI module through Port M.

A key element will be providing a "random" delay between the instant the "start test" (left pushbutton) is pressed and the "`Go Whatsamatta U`" message is displayed on the LCD. Here, the `random` counter maintained by the RTI service routine will be used to trigger the start of a reaction time test.

In order to help with LCD interfacing, the "skeletons" of several functions outlining a simple device driver have been provided for you in the Code Warrior project folder:

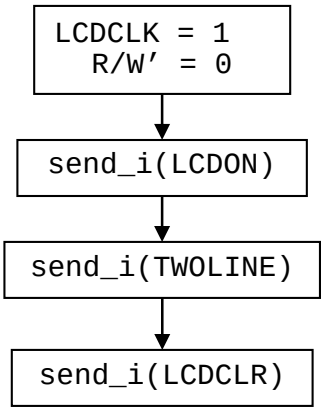
Function	**Operation Performed**
`shiftout(char)`	`Transmit data to shift register using SPI`
`lcdwait(void)`	`Delay for 2 milliseconds`
`send_byte(char)`	`Write data to the LCD`
`send_i(char)`	`Send instruction byte to LCD`
`chgline(char)`	`Move LCD cursor to the position indicated`
`print_c(char)`	`Print character on the LCD`
`pmsglcd(char[])`	`Print character string on the LCD`

You will be required to write all of these functions (or equivalent) to interface with the LCD. Additionally, many of these routines depend on others (for instance, `pmsglcd` requires `print_c` be written). You should build the driver from the ground up starting at `shiftout` and designing toward `pmsglcd`. The `pmsglcd` routine should print the string passed to it on the LCD. Your program should make multiple calls to `print_c` to print individual characters to the LCD screen.

Before printing characters to the LCD, it must first be initialized by writing instruction bytes. The relevant LCD instructions are given in the skeleton file and are replicated below:

Instruction	**Byte**	**Operation Performed**
`LCDON`	`$0F`	`Turn on the LCD driver chip`
`LCDCLR`	`$01`	`Clear the LCD`
`TWOLINE`	`$38`	`Enable two line display mode`
`CURMOV`	`$FE`	`Cursor move command (requires second cursor position byte)`

To initialize the LCD, you must send the LCD commands in the following order. NOTE: LCDCLK and R/W′ are names of signals, not instructions.



NOTE: *The LCD initializations call functions that you must write. It is recommended that you validate these functions before attempting initialization.*

The function `chgline` moves the cursor on the LCD to the cursor position passed to it. To accomplish this, you must first send the CURMOV instruction, followed by a second instruction byte telling the LCD which position you want the cursor to move. These cursor position bytes are given in the table below.

	Cursor Position						
	0	**1**	**2**	**…**	**15**	**…**	**40**
Line 1	$00	$01	$02		$0F		$26
Line 2	$40	$41	$42		$4F		$66
Line 3	**$80**	**$81**	**$82**		**$8F**		$A6
Line 4	**$C0**	**$C1**	**$C2**		**$CF**		$E6

This is the cursor position byte map for our LCD driver (Hitachi HD44780). The area shaded in green corresponds to the valid cursor positions for our 2 x 16 character LCD. For example, the first character of the first line corresponds to cursor position $80 and first character of the second line corresponds to cursor position $C0. It should be noted that both of these cursor positions are given as variables in your skeleton file (`LINE1`/`LINE2`).

Keep in mind that the LCD driver chip is designed to accommodate up to a 4x40 character display (ours is only 2 x 16). This means that you will be able to write to cursor positions that are not within the viewable area of your LCD.

Step 1. Interfacing
Interface the LCD and LEDs to your microcontroller kit as described in the Hardware Overview section. This will require programming your GAL22V10 to function as an 8-bit shift register, as described in the Module 2 homework. Complete all the interface wiring on your breadboard as part of your pre-lab preparation.

Step 2. Software
Write the LCD device driver routines and test them with the interface circuitry completed for Step 1. Once you have verified the LCD driver routines, complete the remainder of the "C" skeleton file provided in the Code Warrior project folder. Note that the "finished product" should work in a "turn key" fashion, i.e., your application code should be stored in flash memory and begin running upon power-on or reset. Demonstrate the completed reaction timer system to your Lab T.A.

Step 3. Submission
Zip the completed Code Warrior project folder and submit it on-line (using the link at the bottom of the Lab Experiments page) *immediately after* demonstrating it to your Lab T.A. ***Be sure identifying information*** (i.e., name, class number, and lab division) ***is included in the*** `main.asm` ***file you submit – credit will not be awarded if identifying information is omitted.***

Bonus Credit
The "stock" reaction timer described above has a "fixed" 250 ms threshold which it uses to distinguish between "fast" and "slow" reaction times. A simple extension would be to "dial in" the desired threshold using a potentiometer, and to display how much faster/slower the user was relative to that threshold.

Use a potentiometer (referenced to 5 VDC) connected to PAD0 as the means for entering the variable "slow/fast" threshold – a suitable range might be 0 to 500 ms. This value should be read as part of the program initialization (after reset) and displayed on the LCD before the first reaction test is initiated: "`Thresh = NNN ms`". Then, when a reaction test is run to completion, the first line of the LCD should display the reaction time the same way as the "stock" version, but the second line should display the *difference* between the measured reaction time and the threshold: e.g., "`100 ms slower`" or "`50 ms faster`".

