ECE 362 Lab Verification / Evaluation Form

Experiment 6

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
Prelab 1	Interface Glue Logic/PLD Design	**5**	
Prelab 2	Schematic/Wiring	**3**	
1	Coding (***must be submitted on-line at time of demo***)	**6**	
2	Demonstration (***must function as specified for full credit***)	**8**	
3	Thought Questions	**3**	
	TOTAL	**25**	

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 6: The “B1G > 10” Football Play Clock

Instructional Objectives:
- To learn how an external interrupt device flag can be realized
- To learn how microcontroller port pins can be used as both inputs and outputs
- To learn how to write an interrupt service routine

Parts Required:
- From “DK-2” parts kit (used in ECE 270)
  - ➢ breadboard and wire
  - ➢ DIP switch, pull-up SIP, and LED
  - ➢ ten 150Ω resistors
- From “DK-3” parts kit
  - ➢ two common anode 7-segment displays (one is included in DK-2 parts kit)
  - ➢ one 22V10 PLD (or equivalent) – also included in DK-2 parts kit

Preparation:
- Read this document in its entirety
- Complete the pre-lab steps and have them available for your instructor to check

Introduction

The “B1G > Ten” has a problem. Plagued by an ever-expanding roster of “marginal” football teams whose games seem to drag on forever (especially where one of the teams gets behind by, say, 42 points before the end of the first half), coach Boris R. Badenov of recently added Whatsamatta U has proposed a new rule that will help speed up these adventures in boredom: reduce the play clock time (i.e., the time allotted to line up and call the next play) to the number of teams that will be in the “B1G > Ten” Conference the next time they play *Note ReDaMe* U (in 2020)…which could be as many as 19(!) This change from the current 40/25 second rule has the potential to significantly reduce the boredom quotient (BQ) associated with the current football experience. Before Coach Badenov can get his billion-dollar bonus for coming up with such a brilliant idea, however, he needs to demonstrate a working prototype. Lucky for BRB, there are a bevy of Boilermakers armed with 9S12C32 microcontroller kits (along with a few surviving parts from their earlier digital lives), betting their design will better the BQ of BBFs (Boisterous Boilermaker Fans) cheering for QQs (Questionable Quarterbacks).

Hardware Overview

Your job is to implement a 19-second play clock, i.e., a timer that when reset is loaded with the value 19 (displayed on two 7-segment LEDs) and count down in one second intervals until 1 second remains, at which point it will count down in 0.1 second intervals from .9 to .0 seconds. The function of the play clock timer will be controlled by the two pushbuttons on the docking module: the left pushbutton (“PAD7”) should reset the play clock to 19 seconds (at the end of every play), while the right pushbutton (“PAD6”) should start/stop the play clock (e.g., team calls a time out). The left LED on your docking board (connected to port pin “PT1”) should be on when the play clock is in the “run” state and off when it is in the “stop” state; the right LED (connected to port pin “PT0”) should be turned on when the count reaches .0 seconds. Note that the clock should stop counting down (“freeze”) once it reaches .0 seconds and that the right LED should remain on until the play clock is reset.

A GAL22V10 PLD will be used to realize all the "glue" logic for this experiment. The function of each Port T pin used in this design is listed in Table 1.

Table 1. 9S12C32 Port T Connections to GAL22V10

9S12C32 Port T	*Function*	*GAL22V10 Pin*
PT2	tens ("half") digit*	2 (input)
PT3	middle decimal point	3 (input)
PT4	BCD ones digit, least significant bit (D0)	4 (input)
PT5	BCD ones digit (D1)	5 (input)
PT6	BCD ones digit (D2)	6 (input)
PT7	BCD ones digit, most significant bit (D3)	7 (input)

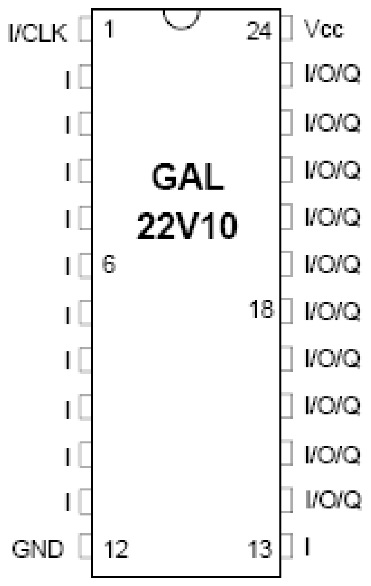
* tens digit is either "1" (segments b and c of 7-segment display *on*) or *blank*, and therefore can be controlled by a single bit

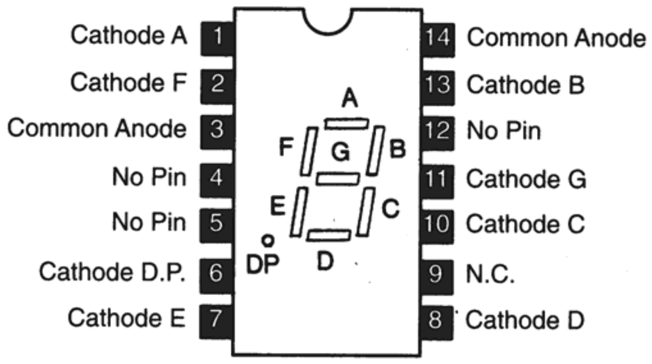
One of the functions the GAL22V10 will be programmed to perform is a BCD-to-display-code conversion for the "ones digit", as illustrated on page 19 of the ECE 270 Module 2 Lecture Summary notes. It will also be used to buffer/drive the middle decimal point (on the two-digit display) and the "tens (half) digit". Since common-anode 7-segment displays will be used, the corresponding output pins on the GAL22V10 will be programmed as *active low*. Note that a 150Ω current limiting resistor should be used between each LED cathode and GAL22V10 output pin (the LED anodes will all be connected to +5 V). Output pins of the GAL22V10 used to drive the display are listed in Table 2.

Table 2. GAL22V10 LED Display Outputs

*GAL22V10 Pin**	*Destination*
14 (com, output)	segments b and c of tens digit
15 (com, output)	middle decimal point
16 (com, output)	segment a of one's digit
17 (com, output)	segment b of one's digit
18 (com, output)	segment c of one's digit
19 (com, output)	segment d of one's digit
20 (com, output)	segment e of one's digit
21 (com, output)	segment f of one's digit
22 (com, output)	segment g of one's digit

* all *active low* – a 150Ω current limiting resistor is required between each LED cathode and PLD pin





The second function the GAL22V10 will be programmed to realize is an interrupt device flag for the timing function. Here, a 100 Hz interrupt source will be provided by the Waveform Generator built in to the MSO at your lab station (reference: *MSO User Guide*, pp. 257-260). Set up the Waveform Generator to produce a 100 Hz square wave with peak-to-peak amplitude of 5.0 volts and an offset of +2.5 volts. This external interrupt request signal will be connected to the CLK input of the GAL22V10 (pin 1), which will then cause a flip-flop within the PLD ("device flag") to *set* on positive (*low-to-high*) edges of this signal. The active low device flag output (pin 23) will be connected to the `IRQ′` input of the 9S12C32 (port PE, pin 1). The device flag flip-flop will be *asynchronously cleared* (by the interrupt service routine) via port M, pin 3 (programmed as an output pin via the DDRM register). These connections are summarized in Table 3.

Table 3. 9S12C32 Port E Connections to GAL22V10

Source	*Function*	*GAL22V10 Pin*
9S12C32 PM3 (output)	asynchronous clear of device flag (active high)	8 (input)
9S12C32 PE1 (input)	CPU IRQ (active low)	23 (output, istype D-reg)
MSO Waveform Generator (output)	100 Hz interrupt request (positive edge) 5 V peak-to-peak square wave with +2.5 V offset	1 (input)

Software Overview

Once the boot-up and initialization sequence is completed, the "main" program should simply "loop" − looking at various "flags" (SRAM locations used as Boolean variables) to see if some action should be taken (this is sometimes called "state machine" code organization). Four such flags will be needed: (a) the "tenth second" flag, (b) the left pushbutton ("reset shot clock") flag, (c) the right pushbutton ("start/stop") flag, and (d) the "run/stop" flag. Note that the timer value should be stored as two "packed" BCD bytes, with an "implied decimal point" between the "ones" and "tenths" digit. For example, the value of the timer when reset should be 019.0 (stored as $0190); after the first interrupt (while the "run" flag is set), this value should be decremented to 018.9 (stored as $0189), i.e., the (4-digit) timer value should be *decremented by one* (in BCD) every time an IRQ interrupt occurs (while the "run" flag is set). Note that the 4-digit BCD value can be decremented by adding $99 to each packed byte (starting with the low order byte), followed by a DAA.

The main program logic should be organized as follows:

If the "tenth second" (IRQ interrupt) flag is set, the following actions should take place:

- clear the "tenth second" (IRQ interrupt) flag
- if the "run/stop" flag is set (i.e., play clock is running), then
    - decrement the play clock by one tenth of a second
    - update the seven-segment display
        - if the time remaining is greater than or equal to10 seconds, display the tens and ones digits
        - if the time remaining is less than 10 seconds but greater than or equal to 1 second, blank the tens display digit and only display the ones digit
        - if the time remaining is less than 1 second, turn on the "middle" decimal point and display the tenths digit instead of the ones digit

If the left pushbutton ("reset play clock") flag is set, the following actions should take place:

- clear the left pushbutton flag
- clear the "run/stop" flag
- turn off both the "run/stop" and "time expired" LEDs (on the docking module)
- reset the display to `19`

If the right pushbutton ("start/stop") flag is set, the following actions should take place:

- clear the right pushbutton flag
- toggle the "run/stop" flag
- toggle the "run/stop" LED (on the docking module)

The main program should also, in its polling loop, check to see if the play clock has reached .0; if it has, the following actions should take place:

- clear the "run/stop" flag
- turn on the "time expired" LED (on the docking module)
- turn off the "run/stop" LED (on the docking module)

In addition to decrementing the timer value stored in memory (while the "run" flag is set), the IRQ interrupt service routine is also used to "sample" the pushbuttons and set the corresponding "pushbutton flags" accordingly (note that this will require an additional variable in memory to track the pushbutton's previous state).

- if the *previous state* of the left pushbutton was high and its *current state* is low (note that the pushbuttons are momentary contact closures to ground, and that their default state is "high"), set the "reset play clock" flag
- if the *previous state* of the right pushbutton was high and its *current state* is low, set the "run/stop" flag

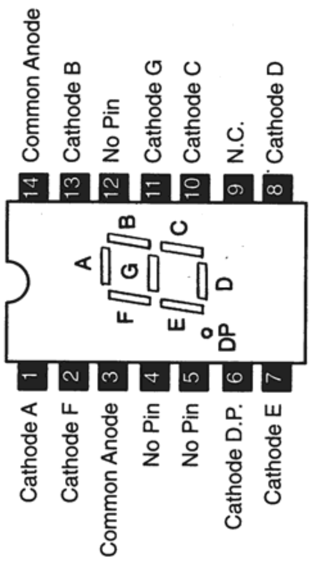
Prelab Step 1. Interface Glue Logic

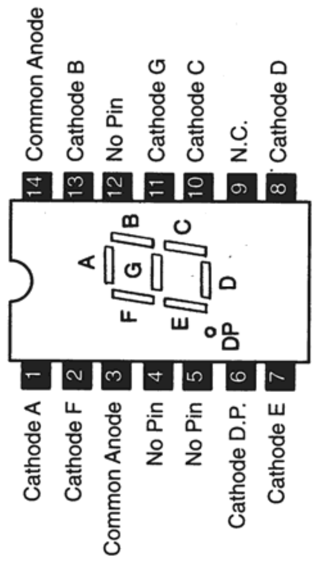
Write an ABEL or Verilog program targeted for a GAL22V10 that realizes the interface glue logic described in the Hardware Overview. Compile your program using ispLever and program your device using the universal programmer at your lab station (review your *ECE 270 Lab Manual* if you have forgotten how to do this).

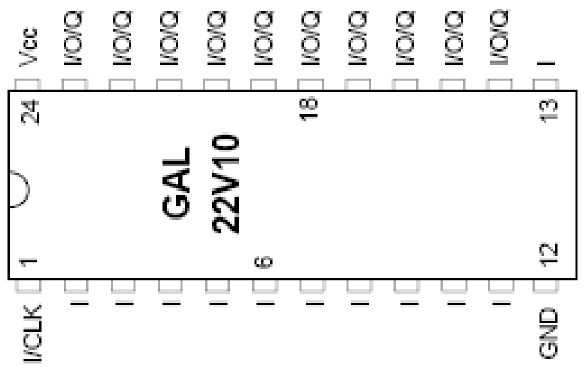
To test your glue logic design, connect the two 7-segment displays to your GAL22V10 as described in Table 2. Use a DIP switch (recalling the need for a pull-up SIP) to provide test inputs to pins 1-8 of the GAL22V10. Finally, connect a "loose" (spare) LED to pin 23 of the GAL22V10 (the active low IRQ signal), which you may want to leave in place for debugging purposes). Now you should be able to test the combinational BCD-to-display-code decoding functionality, the tens (half-digit) display, the middle decimal point, and the operation of the device flag (set by a low-to-high transition of CLK, cleared by an asynchronous reset).

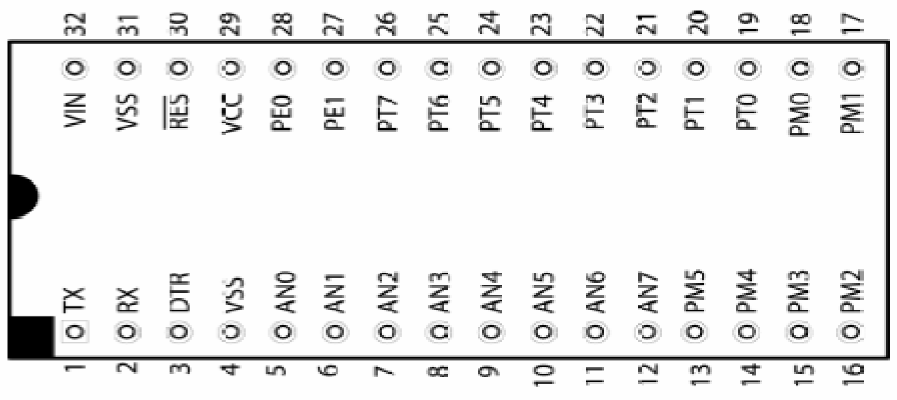
Once you have verified the operation of your interface logic and display circuitry, remove the DIP switch (and its associated pull-up SIP) from your breadboard and install your microcontroller board. *Note that you should only proceed if your glue logic circuitry is functioning correctly – attempting to write code at this point will otherwise prove to be futile!*

Prelab Step 2. Schematic/Wiring
Complete the wiring diagram below, based on information provided in Tables 1-3. Be sure to include the requisite LED current limiting resistors (total of ten 150Ω resistors required). After your TA has checked your schematic, connect your microcontroller module to your GAL22V10 as described in Tables 1 and 3.









Step 1. Coding
Download and unzip the Code Warrior project folder `Lab6` from the course website. Open **main.asm** and note the structure provided, in particular the "vector table" at the end of the file that includes a pointer to the `irq_isr` routine you will need to write. Be sure to review and understand how interrupts work, specifically how control is transferred to/from an interrupt service routine. 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.

Step 2. Demonstration
Demonstrate your working hardware and software to your lab TA, and upload your (zipped) CW project file using the link at the bottom of the Lab Experiments page. ***Project file upload at the time of your code demonstration is required to obtain credit for Steps 1 and 2. Code submitted must be functional to receive full credit. Hardware must work as specified for full credit.***

Step 3. Thought Questions

(a) Why do we need to know the previous state of the pushbuttons, and set "flags" based on their transition from high to low?

____

____

____

(b) Did you notice any "anomalous" behavior as a result of the relatively low (100 Hz) rate at which the pushbuttons are sampled? If so, what do you think might be a more appropriate sampling rate? Be prepared to justify your answer based on an estimate of the mechanical time constant ("bouncing duration") of the pushbuttons on your microcontroller module.

____

____

____

(c) Using the DSO, sample the signal produced by a pushbutton press (contact closure to ground) at either the PAD6 or PAD7 input pin of your microcontroller (set an appropriate trigger condition to accomplish this). Comment on the results obtained. Does the actual pushbutton "bouncing behavior" observed match your prediction in (b)? Based on the actual behavior, what would be an appropriate pushbutton sampling rate?

____

____

____