CSCD 462: Embedded Real-Time Systems
Mini Project 1: Score Display Test Routine

Your task is to create a test program on the pinball machine that:

- cycles the score “simultaneously” on all 5 score displays
- from 0 through 999999
- updating once every half second
- with a circular increment of 8353, and an offset between displays of 8353

Note that the score on display 2 should always be 8353 ahead of the score on display 1, etc. It should take about 1 minute to cycle completely through the range and come back *through* zero. Make sure you exercise all digits.

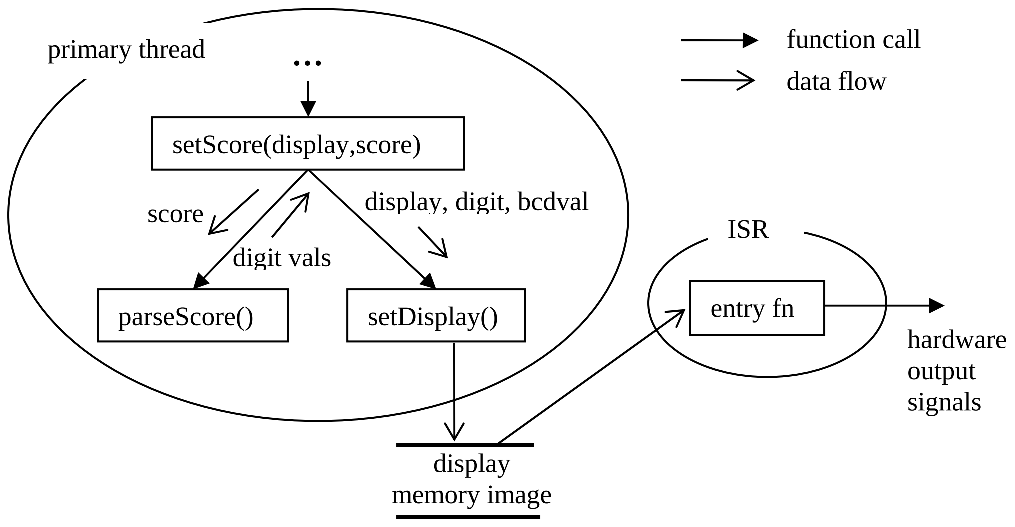
Additional Instructions and Hints:

1. As discussed in class, the hardware on these machines is able to light only a single digit on each display at a time. You’ll therefore need to write an interrupt service routine (ISR) that periodically updates the hardware to display the value of the current digit on each display. At each interrupt it will proceed to the next digit, time multiplexing the displays faster than the human eye can perceive.
2. The current score for each display should be maintained in memory (i.e., in an array). From the main logic, updating the score is then simply a matter of storing values in the array. At some point you will need to parse a score into its individual digits. I highly recommend that you perform that parsing prior to memory storage so that the ISR (which updates the hardware) does not need to spend time performing that parse over and over again. That means array should be 2D (one dimension for the display number and the other for the digit number). Variables that are shared between an ISR and other code should generally be declared volatile. This tells the compiler not to apply optimizations that make assumptions about purely sequential access. This has nothing to do with atomicity of access to shared variables. Atomicity should not be a concern here, but feel free to ask about that in class.
3. The Bally to Arduino interface board will automatically generate Interrupt 2 once every 3.1 msec. I suggest you use a rising edge detection (see the attachInterrupt function).
4. Most Arduino I/O pins are set to input mode by default. There are some pins you’ll need to switch to output mode in your setup() routine:
   a. All 8 bits of PORTA (pins 29, 28, 27, 26, 25, 24, 23, 22, in msb to lsb order). ***A single byte-wide value can be written to these pins by simply assigning to the variable named PORTA (once they have been switched to output mode, of course)***. The entire byte can also be switched to output mode by assigning all ones to the variable DDRA (data direction register A). For this exercise you’ll be writing the nibble representing digit values to the lower 4 bits of PORTA. The upper 4 bits of PORTA contain 4 of the 5 score display strobe bits. Note that you must select the Mega 2560 board in order for PORTA and DDRA to be recognized, because they do not exist on the Uno.
   b. The fifth score display strobe bit, pin 38. This strobe is for the display that shows the credits and the ball-in-play, which you should simply treat like any other score display for this exercise.
   c. The display strobe enable pin, A9. This enables the display strobes that appear on PORTA. Those bits on PORTA have no effect on the score displays unless this line is high.
   d. The display blanking pin, A15. This signal is used to blank the displays while they are being set.
   e. The seven digit enable pins, which are 39, 40, 41, 50, 51, 52, and 53, in LS to MS order. These control which digit the currently captured nibble is setting. You might notice that the Mata Hari

score displays have only 6 digits. The electronics, however, supports up to 7 digits, which some Bally and Stern machines make use of. It won't hurt to write your memory image and ISR to support 7 digits as well, although I won't be checking on that.

Note that when you switch a pin to output mode, its output value goes to a default state of 0 (low voltage), which is fine for all of these pins.

5. The sequence of output signals that must be generated in order to update a given display digit will be covered in class. An outline of the process is:
   a. Set the display blanking bit high (A15) to blank all displays.
   b. Set the enable bit for the previous digit low (one of 39,40,41,50,51,52,53). This turns off that digit.
   c. Update your variable that tracks the current digit.
   d. Enable the display strobes by setting pin A9 high.
   e. for each of the five score displays:
      i. set the lower 4 bits of PORTA to the BCD value to display (>=9 will result in a blank digit).
      ii. toggle the display's strobe line high then low (one of pins 26,27,28,29,38).
   f. Disable the display strobes by setting pin A9 low.
   g. Set the enable bit for the new digit high (39,40,41,50,51,52,53). This turns on the new digit.
   h. Set the display blanking bit low (A15). This turns on all displays.
6. Do a little top-down design thinking on this. You should end up with some functions that have general utility and would be candidates for an eventual library. A hybrid calling tree / data flow diagram for my solution looks like the following. Note also that range checking should occur where appropriate in reusable code. Ask me about this in class if you don't understand what I'm getting at or what this diagram is conveying.


function call
data flow
primary thread
...
setScore(display,score)
score
digit vals
display, digit, bcdval
parseScore()
setDisplay()
ISR
entry fn
hardware output signals
display memory image