

**Complete by:** Wednesday Feb. 29<sup>th</sup> for an A+

**Reference:**

User-Defined LCD character handouts  
Bosch BMA140 accelerometer datasheet, available from the reference section of the piclab website

**Overview:**

For this project, use Timer1 and the 32768 Hz crystal oscillator to sleep except when doing useful stuff. Blink LED1 for 10 ms every two seconds.

Initialize the top row of the LCD with your handle taking up the leftmost five (or so) character positions and with "Sens=0" taking up the rightmost six character positions.

- ✓ Your main loop is to call a Schedule function each time around the main loop. Schedule is to use a scale-of-20 counter, to count loop times, rolling over every fifth of a second.
  - When counter = 5, call the Bosch function described below.
  - When counter = 10, call the ReadScale function described below.
  - When counter = 15, call the UpdateBubble function described below.

The Bosch function checks the output of the left-right axis of the accelerometer, sleeping the Bosch chip when not making a measurement. Initially, read this output with the board horizontal and store the value read into a global signed int variable, LEVEL.

The ReadScale function reads the pot value and scales it to eight values - 0,1,...,7 - each representing one eighth of a turn of the pot and stored in a global unsigned char variable, SCALE. This value will also be used, when changed, to update its value shown in the top right corner of the LCD (i.e. the digit value in the expression set up initially as "Sens=0").

The UpdateBubble function is to update the bottom row of the display if the scaled output of the accelerometer has changed. The update should consist of "erasing" the bubble previously displayed and writing the new one.

**Bosch function:**

To make a left-right accelerometer measurement, make SEL=0b01, wait for two milliseconds using the Delay(200) macro to let the analog output voltage stabilize, read the AN1 channel of the ADC and save the value in a global signed int variable, ACCEL. Then make SEL=0b11 to put the accelerometer back into the standby mode. Form the signed value

BUBBLE\_POSITION = ACCEL - LEVEL

and then scale this value shifting it right by SCALE places. For example, if before scaling,

BUBBLE\_POSITION = 0b0000 0000 0111 0000

then if SCALE = 4, form

BUBBLE\_POSITION = 0b0000 0000 0000 0111

**UpdateBubble function:**

The bubble is to consist of a user-defined 4x4 pixel pattern of dots extending over two character positions of the second row of the LCD. The initial bubble, with the board horizontal, is centered on the two characters in the center of the second row of the display:

```

○○○○○ ○○○○○
○○○○○ ○○○○○
○○●●● ●●○○○
○○●●● ●●○○○
○○●●● ●●○○○
○○●●● ●●○○○
○○○○○ ○○○○○
○○○○○ ○○○○○

```

If BUBBLE\_POSITION has changed, then overwrite the previous two characters holding the bubble with blank characters, i.e., ASCII code = 0x20 = 040 (octal). Then for each increment change in BUBBLE\_POSITION, move one pixel position to the left or right. Do this by creating two new user-defined characters by sending them to the LCD and then using these two characters to update the display.

### BubbleCharStr string:

To facilitate the above process, create a variable string:

```
char BubbleCharStr[18]
```

Initialize the “cursor-position code” with `BubbleCharStr[0] = 0110; // 0x48`

Initialize the null terminator with `BubbleCharStr[17] = 0000; // Null terminator`

Also, create a two-dimensional array of pixel patterns,

```
const char BubbleTbl[5][16] = { ... };
```

Each entry for the first dimension consists of the pixel definitions for two user-defined characters to be used by the Display function to create two characters that will subsequently be accessed with ASCII codes 0x01 and 0x02. For example, the above two characters will have the following entries in the table for BubbleTbl[3][0] to BubbleTbl[3][15]:

```
{0100,0100,0103,0103,0103,0103,0100,0100, 0100,0100,0130,0130,0130,0130,0100,0100}
```

This table entry is copied into BubbleCharStr[1] to BubbleCharStr[16]. The zeroth element of the string, 0110, is the octal code equivalent of 0x48 that will be sent by the Display function as a “Set CG RAM address” command to the LCD controller. The following eight characters generate the ASCII code = 0x01 character and the next eight generate the ASCII code = 0x02 character. The final 0000 in the string is the null terminator.

When the bubble pattern is to be updated, blank the former two characters, translate the signed BUBBLE\_POSITION value into which two characters are to be created and which of the five table entries are to be used. If the bubble is off scale to the left, display < in the leftmost character position. If off scale to the right, display > in the rightmost character position.

### Position and Characters:

Add 58 to the signed number BUBBLE\_POSITION. If the result is zero, the bubble begins at the very left of the display. Divide that result by 5. The remainder designates which of the five strings to use to create the two characters. Adding the octal number 0300 to the quotient gives the cursor position code to which the two user-defined characters are to be written.