Michael Eller - mbe9a
Brendon Koch - blk2jh

Accelerometer

Goal:

Using the MSP430 and a custom header board, the goal of this laboratory was to design and implement an electronic level device. Values gathered from the onboard accelerometer were used to light a ring of 8 LEDs, which also indicated the direction of the tilt. This lab served as a culmination of several previous labs, requiring switch debouncing and a state machine.

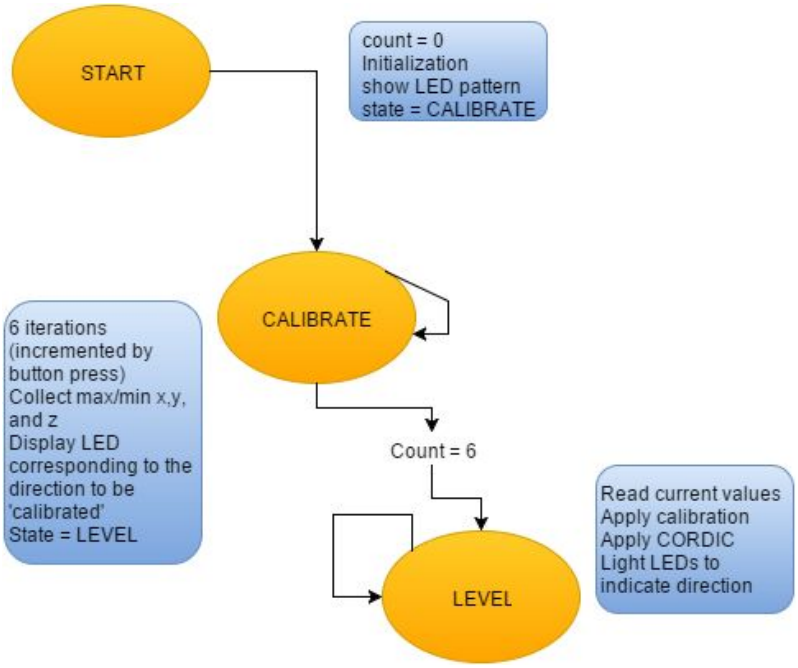
Design Process:

Our process started with designing the electronic level state machine. It was necessary to do this first since the final state machine incorporated all aspects of the project. The project was then divided between us. The data collection was done using the analog to digital converter (ADC) on the MSP430. PWM was used to control the brightness of the LED based on the tilt of the accelerometer and the the position of each LED relative to the center LED. After writing code for the LEDs and ADC, we combined our code and began working on incorporating the CORDIC algorithm to calculate the tilt angle and direction. The accelerometer readings were normalized by subtracting an offset from each axis.

Final Design:

State Machine

The state machine consisted of three states: start, calibrate, and level. Start was used to initialize the calibration counter and show the LED start pattern. The calibration state repeated six times, each waiting for a button press. Depending on the calibration count, the maximum or minimum value for x, y, and z were stored in an array (2 x 3 = 6 iterations). The first calibration iteration also stored the mid x and y values.



Michael Eller - mbe9a
Brendon Koch - blk2jh

ADC

Values from the accelerometer were collected automatically using the data transfer controller (DTC) and the onboard ADC. The values were stored in a cyclic buffer implemented with a two-dimensional array. The values actually used in the level state were an average of the eight values sampled from the ADC and stored in the buffer. The sample function used the DTC to fill the current samples array with new x, y, and z values. The filter function, which was called in main(), used and index variable to determine which element of the buffer to update with new values and calculate the new average.

PWM and LEDs

The LEDs are controlled with three functions: lightLEDs(), setledstates(), and setBrightness(). Each of these takes as an input an LEDinfo struct containing information about the period of the LEDs, the current state the LEDs should be in, and the center LED. It also contains an array, Onticks, that stores how long each LED should be on. The lightLEDs() function has three possible states that it can be in. When in calibrate, it lights the LED corresponding to the center LED to show which direction is being calibrated. The Flat state turns on all LEDs. In the Level state, each of the values in Onticks is decremented and when a value becomes 0 it's corresponding LED is turned off. This is the PWM that controls the LEDs. After the center LED becomes 0, the Onticks array is reset using the setledstates() function. This function sets the duty cycle for each of the LEDs based on the selected center LED. It also generates the byte to send to the LED controller. The setBrightness() function sets the duty cycle for the LEDs based on the tilt of accelerometer. There are three levels of brightness.

Calibration and CORDIC

To calibrate the accelerometer we measured the maximum, minimum, and middle values for the x, y, and z axes. Before sending our current measurements to CORDIC, the middle value for each direction is subtracted from that directions average. This centers the values around 0. CORDIC returns the tilt angle and direction. We tested these values when tilted at different directions to determine when to light each LED. The tolerance for being level was set at 4 degrees as the accelerometer readings are neither stable nor accurate enough to have the tolerance set at 2 degrees.

Problems Encountered

An issue that took us both several hours to fix was a timing issue with the filter() and sample() functions. The set of values in position zero of the buffer was updated with incorrect values and the set in position one was updated with the values that should have been in position zero. All the other values were updated correctly. Fortunately a simple _delay_cycles(100) after sample() solved the issue by giving the buffer time to stabilize.

Work Distribution:

Brendon Koch - PWM
Michael Eller - ADC
The rest of the project was a collaborative effort.