ENEL4187

MECHATRONICS – ULTRASONIC KEYBOARD MONITOR

ZEDDICIAUS PEARSON (SERJEANT) – 1261476

1 INTRODUCTION

This document outlines the specifications, design and construction of an ultrasonic keyboard monitor. This is a device designed to sit idle next to a keyboard with a "heartbeat" LED flashing slowly to indicate the system is on. When motion occurs upon the keyboard, ultrasonic waves detect this and the LED blinks faster to indicate the system is working.

This process took the following steps:

1. Understand how to program a PIC12F675
2. Enable the flashing of the LED
3. Enable the sending of a signal via the ultrasound
4. Understand what is necessary to process the signal returned to the second transducer
5. Implement capturing of this signal using the ADC
6. Tweak until the system matched the specification

2 SPECIFICATION AND METHODS

A "Heartbeat" was implemented to indicate being on and, separately, detection. "On" but nothing detected is represented by an LED: ON period of 40ms, OFF period of 960ms. This corresponds to 1Hz with a duty cycle of 41%. Detection was represented with an ON period of 40ms, OFF period of 160ms, corresponding to 5Hz, 20%. To implement these timings, along with those necessary for a 40KHz transducer signal, Timer0 was used.

Processing of the ultrasound signal began by taking two ADC points in quadrature (360°+90°) a few milliseconds after a signal is sent from the transducer to obtain the magnitude of the signal echoed back. Another calculation is made, and if the two magnitudes differ by a significant degree, motion has been detected. This set of measurements is then repeated over the distance of the keyboard by increasing the time between the send signal and the processing.

This were the most important processes to implement, *Figure 2.1* presents the entire flow of processing.

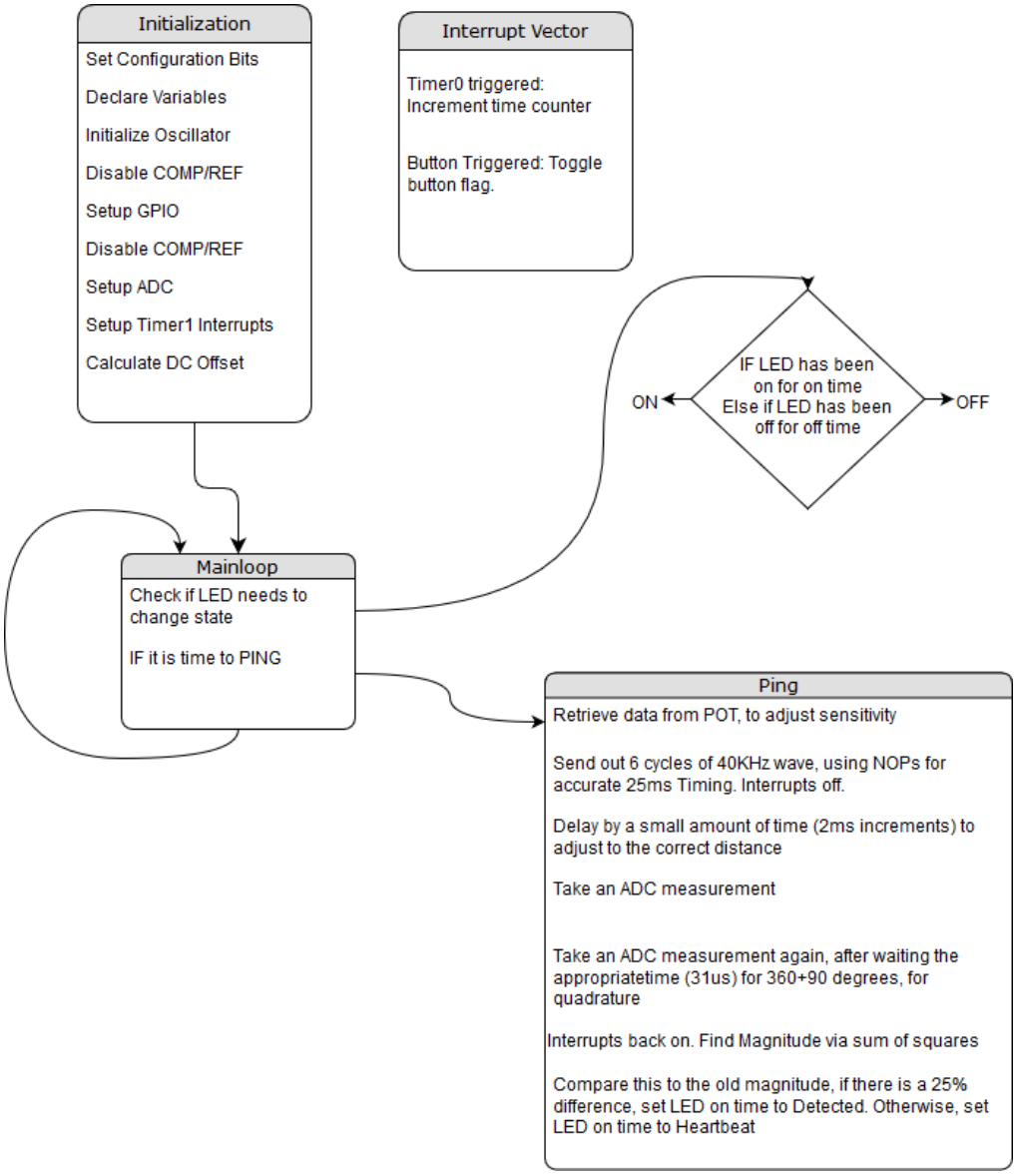

Figure 2.1 - Flowchart of Process

3 RESULTS

The timing of the system was implemented correctly, as can be seen in *Figure 3.1*. This shows the LED in resting state, when a detection hasn't been made.

Figure 3.1 - Resting Heartbeat of LED

Figure 3.2 shows the signal as transmitted from the transducer.

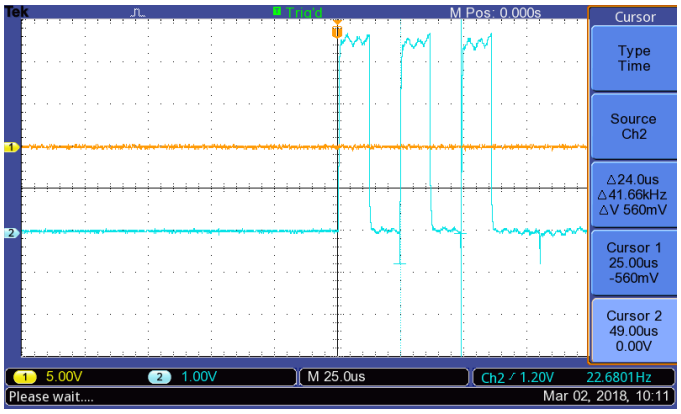

Figure 3.2 - Transmitted Signal

This is not perfectly 40KHz, due to the difficulty of timing, however this was not considered a problem. 40KHz is only the nominal resonant frequency of the transducer; it is likely to vary considerably, though tests to confirm the true resonant frequency were not conducted.

Figure 3.3 shows the initial attempt at measuring the ADC in quadrature, with No NOPs to correct timing.

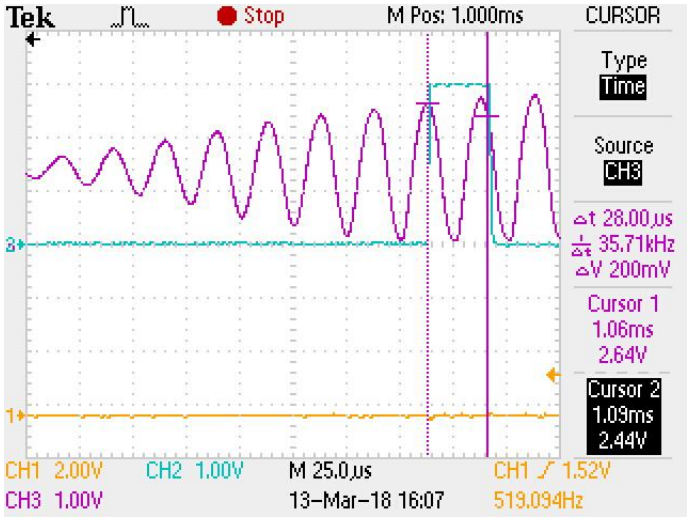

Figure 3.3 - Quadrature Refinement

Finally, to show the complete project, *Figure 3.4* will show the transmitted signal, the received signal, and ADC measure points.

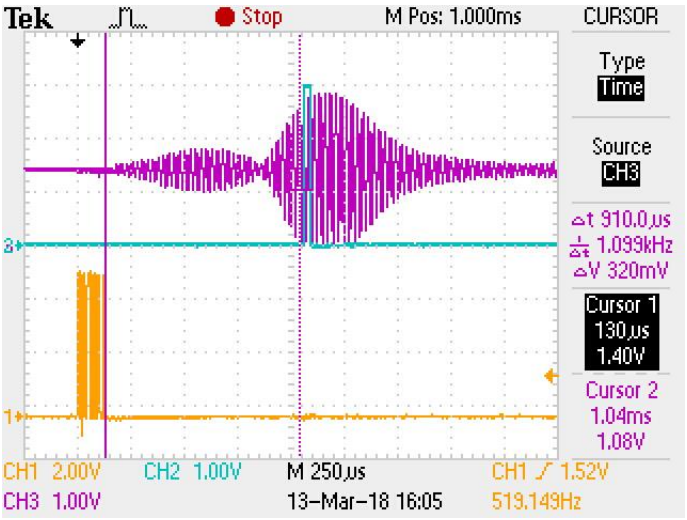

Figure 3.4 - System Functionality

4 PROBLEMS

The graphs shown in S*ection 3* indicate a functional system, however there are goals that were not achieved as, unsurprisingly, the vast amount of time was spent in stage six adjusting variables such as timing and the sensitivity of the magnitude threshold.

Notably, to ensure motion is detected across the entire keyboard, the delay in row three of the flowchart ping block needs to adjust across several values. The best way to display this occurring would be a line spread in the time axis of *Figure 3.4*, however this feature was not implemented correctly at the time of presentation.

Secondly, this device was intended to detect motion, however achieving a sensitivity enough for movement of fingers and not a hand at rest was found to be difficult. The potential reason for this may be an incorrect implementation of the timing such that the sample point was not consistent, leading to motion detected erroneously. This issue may also be at fault for the inability for range detection.

5 APPENDIX I - CODE

```
1.  /* PIC functionality Attempts*/
2.
3.  //#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT enabled)
4.
5.
6.  // #include <pic.h>
7.  #include <xc.h>
8.  // #include <stdlib.h>
9.
10. // Config
11. #pragma config FOSC = INTRCIO   // Oscillator Selection bits (INTOSC oscillator: I/O function on
    GP4/OSC2/CLKOUT pin, I/O function on GP5/OSC1/CLKIN)
12. #pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled)
13. #pragma config PWRTE = OFF      // Power-Up Timer Enable bit (PWRT disabled)
14. #pragma config MCLRE = OFF      // GP3/MCLR pin function select (GP3/MCLR pin function is digital I/O,
    MCLR internally tied to VDD)
15. #pragma config BOREN = OFF      // Brown-out Detect Enable bit (BOD disabled)
16. #pragma config CP = OFF         // Code Protection bit (Program Memory code protection is disabled)
17. #pragma config CPD = OFF        // Data Code Protection bit (Data memory code protection is disabled)
18.
19. #define LED GPIO5
20. #define POT GPIO4 // the potentiometer. This is purely to get a value into the system. //AN3?
21. #define SWITCH GPIO3 // on-off switch
22.
23. // time defines
24. #define n_time_on 40 // time for led to spend on
25. #define n_time_heartbeat 960 // time for led rest
26. #define n_time_detected 160 // time for led detected
27. #define ping_time_top 100 // time to take for each ping
28.
29. // unsigned volatile short int rtmr1 @0x0E;
30.
31. bit flag_button = 0; // represents a flag set by the button
32.
33. bit led_on = 0;
```

```
34. char led_counter = 0;
35.
36. // time counting variables 1tick = 1/(1MHz/4/256)~=1ms
37.
38. int n_time_off = n_time_heartbeat; // time for led to spend off. This changes depending on signal detection to one of the above 2 variables
39. volatile int time_count = 0; // the counter that is compared to the above two variables
40.
41. signed short int dc_offset = 0;
42.
43. volatile char ping_time_count = 0; // counting to the top
44.
45. char n_pings = 6; // the number of cycles of a ping to broadcast at once
46.
47. char delay_time = 0; // allows a delay to occur, by setting this higher than 0
48.
49. char tmp_GPIO = 0; // Used to save the GPIO State between Ultrasound pulses, otherwise wrong pins are changed
50.
51. unsigned short long int temp = 0; // used for temporary calculations
52. unsigned short long int old_mag = 0;
53. unsigned short long int mag = 0;
54. short int pulse_sensitivity = 0; // difference in phase before something is considered movement. we don't need degrees, so absolute divisions work
55.
56. const char n_bins = 4;
57. // char measurement_delay[n_bins] = {2, 2, 3, 3}; // XXX to modify
58. // char measurement_index = 0;
59. char new_measured = 0;
60. bit measured = 0;
61.
62.
63.
64. // union ADC_STORE // allows chars to overlap in memory into a 16bit int
65. // {
66. //     unsigned short int value; // final 16bit value that combines the two bytes
67. //     char bytes[2]; // left and right bytes
68. // } q1, q2; // two points minimum for data capture
```

```
69.
70.  char h1, h2;
71.  signed short int a1, a2;
72.
73.  void main()
74.  {
75.      char i;
76.      // Initialize XXX convert to C eventually
77.      asm("call 0x3FF");
78.      asm("bsf STATUS,5");
79.      //RP0=1; // use Bank 1 for registers // asm("bsf STATUS,5");
80.      asm("movwf OSCCAL");
81.
82.      VRCON = 0X00; // turn off voltage reference
83.      CMCON = 0X07; // disable comparator
84.
85.      //interrupts
86.      INTCON = 0x00; // disable all interrupts
87.      T0IE = 1; // enable timer0 interrupt
88.      //PEIE = 1; // enable peripheral interrupts (ADC is a peripheral)
89.      //ADIE = 1; // enables ADC interruptrs
90.
91.      TRISIO = 0b00011100; // Set all Set all except GP4(AN3), GP3(Switch), GP2(Transducer) to be outputs
92.      GPIO = 0x00; // set all GPIO to low
93.      IOC3 = 1; // enable interrupts on change for GPIO3
94.
95.      //ADC
96.      ADFM = 1; // RIGHT justified registers (2MSB in ADRESH, 8LSB in ADRESL)
97.      VCFG = 0; // Use Vdd as reference
98.      CHS1 = 1; CHS0 = 0;// (AN3,POT)CHS=0b11; (AN2,Receive)CHS=0b10;
99.      //ADCS = 0b001; // Set ADC to Fosc/16 1us (XXX Check this)
100.     ADCS2 = 0;
101.     ADCS1 = 0;
102.     ADCS0 = 1;
103.     ANS3 = 1; // Set AN3 as an analogue input
104.     ANS2 = 1; // Set AN2 as an analogue input
```

```
105.    ADON = 1; // enable the ADC, GO=1 starts a conversion
106.
107.
108.    //timer0
109.    T0CS = 0; // Use internal clock for the timer
110.    PSA = 0; // prescaler is used by Timer0
111.    //set the prescaler to be 1:4
112.    PS2 = 0;
113.    PS1 = 0;
114.    PS0 = 1;
115.
116.    GIE = 1; // enable all interrupts
117.
118.    //sample to find DC offset
119.    GO = 1; // fire ADC
120.    while (GO); // wait until the ADC has a value
121.    dc_offset = (ADRESH<<8) | ADRESL;
122.
123.    while (1) // flash LED
124.    {
125.        if (led_on) // && (led_counter < start_counter)
126.        {
127.            LED = 1; // turn on the LED
128.            if (time_count > n_time_on) // check to see if we are finished being on
129.            {
130.                time_count = 0;
131.                led_on = 0;
132.            }
133.        }
134.        else
135.        {
136.            LED = 0;
137.            if (time_count > n_time_off)
138.            {
139.                time_count = 0;
140.                led_on = 1;
```

```
141.         }
142.     }
143.
144.
145.
146.     if (ping_time_count > ping_time_top)
147.     {
148.         ping_time_count = 0;
149.
150.         // if (measurement_index == n_bins)
151.         // {
152.             // measurement_index = 0;
153.         CHS1 = 1; CHS0 = 1;// (AN3,POT)CHS=0b11; (AN2,Receive)CHS=0b10;
154.         asm("NOP");
155.         asm("NOP");
156.         asm("NOP");
157.         asm("NOP");
158.         GO = 1;
159.         while (GO);
160.         // if ((temp>>3) > ((ADRESH<<8)+ADRESL))
161.         // {
162.         //     LED = 1;
163.         // }
164.         // else
165.         // {
166.         //     LED = 0;
167.         // }
168.         //Change ADC back
169.         pulse_sensitivity = (ADRESH<<8)+ADRESL;
170.         CHS1 = 1; CHS0 = 0;// (AN3,POT)CHS=0b11; (AN2,Receive)CHS=0b10;
171.         // }
172.         GIE = 0; // disable interrupts for this
173.         tmp_GPIO = GPIO; // save GPIO
174.
175.
176.         // Loop to get x cycles off the transducer
```

```
177.        for (i=0; i<n_pings; i++)
178.        {
179.            // asm("NOP");
180.            // asm("NOP");
181.            // asm("NOP");
182.            // asm("NOP");
183.            //asm("NOP");
184.
185.            GPIO = 0x01;
186.
187.            asm("NOP");
188.            asm("NOP");
189.            asm("NOP");
190.            asm("NOP");
191.            asm("NOP");
192.            asm("NOP");
193.            asm("NOP");
194.            asm("NOP");
195.            asm("NOP");
196.
197.            GPIO = 0x02;
198.        }
199.        GIE = 1;
200.
201.        delay_time = 2;
202.        // measurement_index++;
203.
204.        while (delay_time); // delay for 3 ticks to give time for the ping to reflect back, ignoring transmit echo
205.        GIE = 0;
206.
207.        GO = 1; // fire ADC
208.        while (GO); // wait until the ADC has a value
209.        // LED = 1;
210.        asm("NOP");
211.        asm("NOP");
212.        // LED = 0;
```

```
213.        GO = 1; // Once it has a value, tell it to fetch another value. This doesn't overwrite current value, and this is done for speed
214.        h1 = ADRESH;
215.        h2 = ADRESL;
216.        while (GO); // wait for current conversion to finish now.
217.
218.        GPIO = tmp_GPIO; // restore GPIO
219.        GIE = 1; // reenable interrupts
220.
221.        a1 = ((signed short int)(((h1<<8)+h2) - dc_offset));
222.        a2 = ((signed short int)(((ADRESH<<8)+ADRESL) - dc_offset));
223.
224.        old_mag = mag;
225.        mag = ((a1*a1) + (a2*a2));
226.        // if (measurement_index == 0)
227.        // {
228.        //     continue;
229.        // }
230.
231.        if (old_mag > mag)
232.        {
233.            temp = old_mag - mag;
234.        }
235.        else if (old_mag < mag)
236.        {
237.            temp = mag - old_mag;
238.        }
239.        else
240.        {
241.            temp = 0;
242.        }
243.
244.        if ((temp>>4) > (pulse_sensitivity)) // if the magnitude with sqrt is larger than 70mV, this is a point worth considering, and for now this will just active
245.        {
246.            n_time_off = n_time_detected;
247.        }
```

```
248.            else
249.            {
250.                n_time_off = n_time_heartbeat;
251.            }
252.        }
253.    }
254. }
255.
256. void interrupt ISR()
257. {
258.    if (T0IF) // timer0 triggered
259.    {
260.        T0IF = 0; //clear the overflow bit
261.        TMR0 = 0; // set the counter back to 0 (XXX maybe not strictly necessary?)
262.        time_count++;
263.        ping_time_count++;
264.        if (delay_time > 0)
265.        {
266.            delay_time--;
267.        }
268.    }
269.    // if (ADIF) // adc triggered
270.    // {
271.    //     ADIF = 0; // clear the interrupt flag
272.    //     if (ADRESH > 125)
273.    //     {
274.    //         LED = 1;
275.    //     }
276.    //     else
277.    //     {
278.    //         LED = 0;
279.    //     }
280.    // }
281.    if (GPIF) // pin change
282.    {
283.        if (GPIO3)
```

```
284.     {
285.         flag_button ^= 1;
286.     }
287.     GPIF = 0;
288. }
289. // return;
290. }
```

```
291.
```