```
 1 /*****************************************************************************************/
 2 /* BME 363 Biomeasurement Lab - PIC18F4525BT Demo                                        */
 3 /* Laboratories 1-8 from BME 361: A/D/A, LCD display, ECG simulation, filters, QRS detection */
 4 /* Instructors: John DiCecco, Ying Sun                                                   */
 5 /* Update history: 12/01/2015 modified by Cody Goldberg for Bluetooth interface */
 6 /*                 03/12/2016 modified by Ying Sun for supporting both LCD and BT */
 7 /*                 03/24/2016 modified by Ying Sun for XC8 compiler                      */
 8 /*                 02/13/2021 modified by Joseph Reyes for current XC8 compiler          */
 9 /*****************************************************************************************/
10
11 /***************************** Specify the chip that we are using ****************************/
12 #include <p18cxxx.h>
13 #include <math.h>
14 #include <stdlib.h>
15
16 /************************* Configure the Microcontroller PIC18f4525 ************************/
17 #pragma config OSC = XT
18 #pragma config WDT = OFF
19 #pragma config PWRT = OFF
20 #pragma config FCMEN = OFF
21 #pragma config IESO = OFF
22 #pragma config BOREN = ON
23 #pragma config BORV = 2
24 #pragma config WDTPS = 128
25 #pragma config PBADEN = OFF
26 #pragma config DEBUG = OFF
27 #pragma config LVP = OFF
28 #pragma config STVREN = OFF
29 #define _XTAL_FREQ 4000000
30
31 /******************************** Define Prototype Functions *******************************/
32 unsigned char ReadADC();
33 void Delay_ms(unsigned int x);
34 void Transmit(unsigned char value);
35 void TransmitBT(unsigned char value);
36 void PrintNum(unsigned char value1, unsigned char position1);
37 void SetupBluetooth();
38 void SetupSerial();
39 void SetupADC(unsigned char channel);
40 void ClearScreen();
41 void Backlight(unsigned char state);
42 void SetPosition(unsigned char position);
43 void PrintLine(const unsigned char *string, unsigned char numChars);
44 void PrintInt(int value, unsigned char position);
45 void __interrupt () isr(void);
46
47 /*********************************** Global variables ************************************/
48 unsigned char function, functionBT, mode, update, debounce0, debounce1, debounce2;
49 unsigned char LEDcount, output, output1, output2, counter, counter1, skipCount;
50 unsigned char data0, data1, data2, array[9], rank[9], do_MOBD, refractory, display;
51 unsigned char temp, sampling[16], TMRcntH[16], TMRcntL[16], sampling_H, sampling_L;
52 unsigned char enableBT; // BLUETOOTH
53 int i, j, dummy, d0, d1, d2, mobd, threshold, rri_count, hr;
54
```

```
55 unsigned char ReadADC() { /************* start A/D, read from an A/D channel ****************/
56     unsigned char ADC_VALUE;
57     ADCON0bits.GO = 1;// Start the AD conversion
58     while(!PIR1bits.ADIF) continue;// Wait until AD conversion is complete
59     ADC_VALUE = ADRESH;// Return the highest 8 bits of the 10-bit AD conversion
60     return ADC_VALUE;
61 }
62
63 void Delay_ms(unsigned int x){ /****** Generate a delay for x ms, assuming 4 MHz clock ******/
64     unsigned char y;
65     for(;x > 0; x--) for(y=0; y< 82;y++);
66 }
67
68 void Transmit(unsigned char value) {  /********** send an ASCII Character to USART ***********/
69     while(!PIR1bits.TXIF) continue;// Wait until USART is ready
70     TXREG = value;// Send the data
71     while (!PIR1bits.TXIF) continue;// Wait until USART is ready
72     Delay_ms(4); // Give the LCD some time to listen to what we've got to say.
73 }
74
75 void TransmitBT(unsigned char value) {  /********* send an ASCII Character to USART **********/
76     while(!PIR1bits.TXIF) continue;// Wait until USART is ready
77     TXREG = value;// Send the data
78     while (!PIR1bits.TXIF) continue;// Wait until USART is ready
79 }
80
81 void PrintNum(unsigned char value1, unsigned char position1){ /** Print number at position ***/
82     int units, tens, hundreds, thousands;
83     SetPosition(position1);// Set at the present position
84     hundreds = value1 / 100;// Get the hundreds digit, convert to ASCII and send
85     if (hundreds != 0) Transmit(hundreds + 48);
86     else Transmit(20);
87     value1 = value1 - hundreds * 100;
88     tens = value1 / 10;// Get the tens digit, convert to ASCII and send
89     Transmit(tens + 48);
90     units = value1 - tens * 10;
91     Transmit(units + 48);// Convert to ASCII and send
92 }
93
94 void SetupBluetooth() { /*************** set up Bluetooth modem Roving RN-42 ****************/
95     SPBRG = 1;          // Push up to 115200 BAUD to configure the BL module.
96     Delay_ms(500);
97     TransmitBT("$");      // Enter command mode
98     TransmitBT("$");
99     TransmitBT("$");
100    Delay_ms(100);      // Wait for the BL module to respond
101    PrintLine((const unsigned char*)"U,115200,N", 8);   // Tell it that we want 9600 BAUD
102    Delay_ms(100);
103 }
104
105
106 void SetupSerial(){  /*********** Set up the USART Asynchronous Transmit (pin 25) ************/
107 // For LCD - use SPBRG = 25;  9600 BAUD at 4MHz: 4,000,000/(16x9600) - 1 = 25.04
108 // For Bluetooth - use SPBRG = 1; 115200 BAUD at 4MHz: 4,000,000/(16x115200) - 1 = 1
```

```
109     TRISC = 0x80;// Transmit and receive, 0xC0 if transmit only
110     SPBRG = 25;
111     TXSTAbits.TXEN = 1;// Transmit enable
112     TXSTAbits.SYNC = 0;// Asynchronous mode
113     RCSTAbits.CREN = 1;// Continuous receive (receiver enabled)
114     RCSTAbits.SPEN = 1;// Serial Port Enable
115     TXSTAbits.BRGH = 1;// High speed baud rate
116 }
117
118 void SetupADC(unsigned char channel){ /******** Configure A/D and Set the Channel **********/
119     TRISA = 0b11111111;// Set all of Port A as input
120     // ADCON2 Setup
121     // bit 7: Left justify result of AD (Lowest 6bits of ADRESL are 0's) (0)
122     // bit 6: Unimplemented
123     // bit 5-3: AD aquisition time set to 2 TAD (001)
124     // bit 2-0: Conversion clock set to Fosc/8 (001)
125     ADCON2 = 0b00001001;
126     // ADCON1 Setup
127     // bit 7-6: Unimplemented
128     // bit 5: Vref - (VSS) (0)
129     // bit 4: Vref + (VDD) (0)
130     // bit 3-0: Configuration of AD ports (Set A0-A4, others to digital, including the PORTB)
131     ADCON1= 0b00001010;
132     // ADCON0 Setup
133     // bit 7,6 = Unimplemented
134     // bits 5-2 = Channel select
135     // bit 1: GO Bit (Starts Conversion when = 1)
136     // bit 0: AD Power On
137     ADCON0 = (channel << 2) + 0b00000001;
138     PIE1bits.ADIE = 0;// Turn off the AD interrupt
139     PIR1bits.ADIF = 0;// Reset the AD interrupt flag
140 }
141
142 void ClearScreen(){   /************************** Clear LCD Screen ***************************/
143     Transmit(254);// See datasheets for Serial LCD and HD44780
144     Transmit(0x01);// Available on our course webpage
145 }
146
147 void Backlight(unsigned char state){  /************* Turn LCD Backlight on/off ***************/
148     Transmit(124);
149     if (state) Transmit(0x9D);// If state == 1, backlight on; ox96 for 73% on
150     else Transmit(0x81);// otherwise, backlight off
151 }
152
153 void SetPosition(unsigned char position){ /********** Set LCD Cursor Position  *************/
154     Transmit(254);
155     Transmit(128 + position);
156 }
157
158 void PrintLine(const unsigned char *string, unsigned char numChars){ /**** Print characters ****/
159     unsigned char count;
160     for (count=0; count<numChars; count++) Transmit(string[count]);
161 }
162
```

```
163 void PrintInt(int value, unsigned char position){ /******** Print number at position *********/
164     int units, tens, hundreds, thousands;
165     SetPosition(position);// Set at the present position
166     if (value > 9999) {
167         PrintLine((const unsigned char*)"Over", 4);
168         return;
169     }
170     if (value < -9999) {
171         PrintLine((const unsigned char*)"Under", 5);
172         return;
173     }
174     if (value < 0) {
175         value = -value;
176         Transmit(45);
177     }
178     else Transmit(43);
179     thousands = value / 1000;// Get the thousands digit, convert to ASCII and send
180     if (thousands != 0) Transmit(thousands + 48);
181     value = value - thousands * 1000;
182     hundreds = value / 100;// Get the hundreds digit, convert to ASCII and send
183     Transmit(hundreds + 48);
184     value = value - hundreds * 100;
185     tens = value / 10;// Get the tens digit, convert to ASCII and send
186     Transmit(tens + 48);
187     units = value - tens * 10;
188     Transmit(units + 48);// Convert to ASCII and send
189 }
190
191 void __interrupt () isr(void) { /************ high priority interrupt service routine *************/
192     //for new version compiler void __interrupt () isr(void), prev versions omit (), then omit '__'
193     if (INTCONbits.TMR0IF == 1) {// When there is a timer0 overflow, this loop runs
194         INTCONbits.TMR0IE = 0;// Disable TMR0 interrupt
195         INTCONbits.TMR0IF = 0;// Reset timer 0 interrupt flag to 0
196         switch (function) {
197             case 0:
198                 TMR0H = 0xEF;       // Reload TMR0 for 4.167 ms count, Sampling rate = 240 Hz
199                 TMR0L = 0xEA;       // 0xFFFF-0xEFB8 = 0x1047 = 4167, adjust for delay by 50 us
200                 break;
201             case 1:// Function 1: ECG simulation
202                 TMR0H = 0xFC;// Reload TMR0 for 1 ms count, sampling rate = 1KHz
203                 TMR0L = 0x4D;// 0xFFFF-0xFC17 = 0x3b2 = 946,adust for delay by 54 us
204                 switch (mode) {
205                     case 0:// P wave up
206                         counter++;output++;if (counter == 30) mode++;
207                         break;
208                     case 1:// P wave flat
209                         counter--;if (counter == 0) mode++;
210                         break;
211                     case 2:// P wave down
212                         counter++;output--;if (counter == 30) mode++;
213                         break;
214                     case 3:// PR segment
215                         counter++;if (counter == 100) mode++;
216                         break;
```

```
217                     case 4:// QRS complex - Q
218                         counter++;output -= 3;
219                         if (counter == 105){
220                             counter = 0;
221                             mode++;
222                         }
223                         break;
224                     case 5:// QRS complex - R up
225                         counter++;output += 6;if (counter == 30) mode++;
226                         break;
227                     case 6:// QRS complex - R down
228                         counter++;output -= 6;if (counter == 62) mode++;
229                         break;
230                     case 7://QRS complex - S
231                         counter++;output += 3;
232                         if (counter == 71) {
233                             mode++;
234                             counter = 0;
235                         }
236                         break;
237                     case 8:// ST segment
238                         counter++;
239                         if (counter == 89){
240                             counter = 0;
241                             mode++;
242                         }
243                         break;
244                     case 9:// T wave up
245                         counter++;output++;if (counter == 55)mode++;
246                         break;
247                     case 10:// T wave flat
248                         counter++;if (counter == 110) mode++;
249                         break;
250                     case 11:// T wave down
251                         counter++;output--;if (counter == 165) mode++;
252                         break;
253                     case 12:// End ECG
254                         counter--;if (counter == 0) mode++;
255                         break;
256                     case 13:// Reset ECG
257                         counter++;
258                         if (counter == 202){
259                             counter = mode = 0;
260                             output = 50;
261                         }
262                         break;
263                 }
264                 PORTD = output;
265                 if (enableBT) {
266                     skipCount++;
267                     if (skipCount == 8) {
268                         TransmitBT(functionBT);
269                         TransmitBT(output);
270                         TransmitBT(128);
```

```
                    skipCount = 0;
                }
            }
            break;
        case 2:// Function 2: Echo
            TMR0H = 0xEF;           // Reload TMR0 for 4.167 ms count, Sampling rate = 240 Hz
            TMR0L = 0xEF;           // 0xFFFF-0xEFB8 = 0x1047 = 4167, adjust for delay by 55 us
            data0 = ReadADC();// Read A/D and save the present sample in data0
            output = data0;
            PORTD = output;// Echo back
            if (enableBT) {
                skipCount++;
                if (skipCount == 2) {
                    TransmitBT(functionBT);
                    TransmitBT(data0);
                    TransmitBT(output);
                    skipCount = 0;
                }
            }
            break;
        case 3:// Function 3: Echo (vary rate), not required during 361
            TMR0H = sampling_H;// Reload TMR0 high-order byte
            TMR0L = sampling_L;// Reload TMR0 low-order byte
            SetupADC(2);// Switch to A/D channel AN2
            counter = ReadADC();// Read potentiometer setting from AN2
            SetupADC(0);// Switch to A/D channel AN0
            counter = counter >> 4;// Scale it to 0-15
            sampling_L = TMRcntL[counter];// Load TMR0 low-order byte
            sampling_H = TMRcntH[counter];// Load TMR0 high-order byte
            data0 = ReadADC();// Read A/D and save the present sample in data0
            output = data0;
            PORTD = data0;// Echo back
            if (enableBT) {
                skipCount++;
                if (skipCount == 2) {
                    TransmitBT(functionBT);
                    TransmitBT(data0);
                    TransmitBT(output);
                    skipCount = 0;
                }
            }
            break;
        case 4:// Function 4: Derivative
            TMR0H = 0xEF;           // Reload TMR0 for 4.167 ms count, Sampling rate = 240 Hz
            TMR0L = 0xF6;           // 0xFFFF-0xEFB8 = 0x1047 = 4167, adjust for delay by 62 us
            data1 = data0;// Store previous data points
            data0 = ReadADC();// Read A/D and save the present sample in data0
            dummy = (int)data0 - data1 + 128;// Take derivative & shift to middle
            if (dummy < 0) dummy = 0;       // Chop off if outside the range of 0 - 255
            if (dummy > 255) dummy = 255;
            output2 = output1;
            output1 = output;
            output = (unsigned char)dummy;
            PORTD = output;
```

```
                if (enableBT) {
                    d2 = d1;
                    d1 = d0;
                    d0 = (int)data0 - data1;
                    if (d0 < 0) d0 = -d0;
                    skipCount++;
                    if (skipCount == 2) {
                        if (d2 > 40) output = output2;
                        if (d1 > 40) output = output1;
                        TransmitBT(functionBT);
                        TransmitBT(data0);
                        TransmitBT(output);
                        skipCount = 0;
                    }
                }
                break;
            case 5:// Function 5: Low-pass filter
                TMR0H = 0xEF;           // Reload TMR0 for 4.167 ms count, Sampling rate = 240 Hz
                TMR0L = 0xF6;           // 0xFFFF-0xEFB8 = 0x1047 = 4167, adjust for delay by 62 us
                data2 = data1;// Store previous data points
                data1 = data0;
                data0 = ReadADC();// Read A/D and save the present sample in data0
                dummy = ((int)data0 + data1 + data1 + data2) / 4;// smoother
                output = (unsigned char)dummy;
                PORTD = output;         // Output to D/A
                if (enableBT) {
                    skipCount++;
                    if (skipCount == 2) {
                        TransmitBT(functionBT);
                        TransmitBT(data0);
                        TransmitBT(output);
                        skipCount = 0;
                    }
                }
                break;
            case 6:// Function 6: High-frequency enhancement filter
                TMR0H = 0xEF;           // Reload TMR0 for 4.167 ms count, Sampling rate = 240 Hz
                TMR0L = 0xF6;           // 0xFFFF-0xEFB8 = 0x1047 = 4167, adjust for delay by 62 us
                data2 = data1;// Store previous data points
                data1 = data0;
                data0 = ReadADC();// Read A/D and save the present sample in data0
                dummy = ((int)data0 + data1 + data1 + data2) / 4;// smoother
                dummy = data0 + data0 - dummy;
                if (dummy < 0) dummy = 0;       // Chop off if outside the range of 0 - 255
                if (dummy > 255) dummy = 255;
                output2 = output1;
                output1 = output;
                output = (unsigned char)dummy;
                PORTD = output;
                if (enableBT) {
                    d2 = d1;
                    d1 = d0;
                    d0 = ((int)data0 + data1 + data1 + data2) / 4;
                    d0 = data0 - d0;
                    if (d0 < 0) d0 = -d0;
```

```
                    if (d0 < 0) d0 = -d0;
                    skipCount++;
                    if (skipCount == 2) {
                        if (d2 > 40) output = output2;
                        if (d1 > 40) output = output1;
                        TransmitBT(functionBT);
                        TransmitBT(data0);
                        TransmitBT(output);
                        skipCount = 0;
                    }
                }
                break;
            case 7:// Function 7: 60Hz notch filter
                TMR0H = 0xEF;           // Reload TMR0 for 4.167 ms count, Sampling rate = 240 Hz
                TMR0L = 0xFC;           // 0xFFFF-0xEFB8 = 0x1047 = 4167, adjust for delay by 68 us
                data2 = data1;// Store previous data points
                data1 = data0;
                data0 = ReadADC();// Read A/D and save the present sample in data0
                dummy = ((int)data0 + data2) / 2;// 60 Hz notch
                output = (unsigned char)dummy;
                PORTD = output;         // Output to D/A
                if (enableBT) {
                    skipCount++;
                    if (skipCount == 2) {
                        TransmitBT(functionBT);
                        TransmitBT(data0);
                        TransmitBT(output);
                        skipCount = 0;
                    }
                }
                break;
            case 8:// Function 8: Median filter
                TMR0H = 0xEF;           // Reload TMR0 for 4.167 ms count, Sampling rate = 240 Hz
                TMR0L = 0xFF;           // 0xFFFF-0xEFB8 = 0x1047 = 4167, adjust for delay by 71 us
                data0 = ReadADC();// Read A/D and save the present sample in data0
                for (i=8; i>0; i--) array[i] = array[i-1];// Store the previous 8 points
                array[0] = data0;// Get new data point from A/D
                for (i=0; i<9; i++) rank[i] = array[i];// Make a copy of data array
                for (i=0; i<5; i++) {// Perform a bubble sort
                    for (j=i+1; j<9; j++) {
                        if (rank[i] < rank[j]) {
                            temp = rank[i];// Swap
                            rank[i] = rank[j];
                            rank[j] = temp;
                        }
                    }
                }
                output = rank[4];
                PORTD = output;// Median is at rank[4] of rank[0-8]
                if (enableBT) {
                    skipCount++;
                    if (skipCount == 2) {
                        TransmitBT(functionBT);
                        TransmitBT(data0);
                        TransmitBT(output);
```

```
                TransmitBT(output);
                skipCount = 0;
            }
        }
        break;
    case 9:// Function 9: Heart rate meter
        TMR0H = 0xEC;// Reload TMR0 for 5 ms count, sampling rate = 200 Hz
        TMR0L = 0xC3;// 0xFFFF-0xEC77 = 0x1388 = 5000, adjust for delay by 76 us
        data1 = data0;// Move old ECG sample to data1
        data0 =  ReadADC();// Store new ECG sample from ADC to data0
        if (enableBT) {
            skipCount++;
            if (skipCount == 2) {
                TransmitBT(functionBT);
                TransmitBT(data0);
                TransmitBT(output);
                skipCount = 0;
            }
        }                       // MOBD = Multiplication of Backwards Differences
        d2 = d1;// Move oldest difference to d2
        d1 = d0;// Move older difference to d1
        d0 = (int)data0 - data1;// Store new difference in d0, (int) casting important
        rri_count++;// Increment RR-interval
        mobd = 0;// mobd = 0, unless sign consistency is met:
        if (d0 > 0 && d1 > 0 && d2 > 0){// (1) If 3 consecutive positive differences
            mobd = d0 * d1;// Multiply first two differences
            mobd = mobd * d2;// Multiply the oldest difference
        }
        if (d0 < 0 && d1 < 0 && d2 < 0){// (2) If 3 consecutive negative differences
            d0 = -d0;// Take absolute value of differences
            d1 = -d1;
            d2 = -d2;
            mobd = d0 * d1;// Multiply first two differences
            mobd = mobd * d2;// Multiply the oldest difference
        }
        if (refractory){// Avoid detecting extraneous peaks after QRS
            refractory++;
            if (refractory == 40){// Delay for 200 ms
                refractory = 0;// Reset refractory flag to 0
                PORTBbits.RB3 = 0;// Turn buzzer/LED off (Pin 36)
            }
        }
        else if (mobd > threshold){// If a peak is detected,
            refractory = 1;// Set refractory flag
            PORTBbits.RB3 = 1;// Turn buzzer/LED on (Pin 36)
            display = 1;// Set display flag
        }
        if (mobd > 255) output = 255;
        else output = (unsigned char)mobd;
        PORTD = output;                // Output mobd value to Port D
        break;

        case 10: // Function 10: Photoplethysmogram
        TMR0H = 0xFE; // Reload TMR0 high-order byte
        TMR0L = sampling_L; // Reload TMR0 low-order byte
```

```
                TMR0L = sampling_L; // Reload TMR0 low order byte
                PORTCbits.RC3 = !PORTCbits.RC3; // Toggle RC3 (pin 18) @ 1 KHz for PPG
                skipCount++;
                if (skipCount == 5) {
                SetupADC(2); // Switch to A/D channel AN2
                sampling_L = ReadADC(); // Read potentiometer setting from AN2
                SetupADC(3); // Switch to A/D channel AN2
                }
                output = ReadADC(); // Read PPG from AN3
                d0 += output;
                if (skipCount == 8) { // 1 KHz / 8 = 125 Hz, check pin 18 for the actual frequency
                skipCount = 0;
                d0 = d0 >> 3; // d0 contains the sum of 8 points, then / 8 (shift 3 bits)
                output = (unsigned char) d0;
                d0 = 0;
                if (enableBT) {
                TransmitBT(functionBT);
                TransmitBT(output);
                TransmitBT(128);
                             }
                                }
                break;
        }
        if (debounce0) debounce0--;// switch debounce delay counter for INT0
        if (debounce1) debounce1--;// switch debounce delay counter for INT1
        if (debounce2) debounce2--;// switch debounce delay counter for INT2
        PORTCbits.RC2 = !PORTCbits.RC2;// Toggle RC2 (pin 17) for sampling frequency check
        INTCONbits.TMR0IE = 1;// Enable TMR0 interrupt
    }
    if (INTCONbits.INT0IF == 1) {// INT0 (pin 33) negative edge - Function down
        INTCONbits.INT0IE = 0;// Disable interrupt
        INTCONbits.INT0IF = 0;// Reset interrupt flag
        if (debounce0 == 0) {
            if (function <= 0) function = 10;// Set function range 0-9
            else function--;
            if (function == 9) SetupADC(1);// ECG comes from AN1 channel
            else SetupADC(0);// Others come from AN0 channel
            functionBT = function | 0xF0;       // function code for Android
            update = 1;// Signal main() to update LCD display
            debounce0 = 10;// Set switch debounce delay counter decremented by TMR0
        }
        INTCONbits.INT0IE = 1;// Enable interrupt
    }
    if (INTCON3bits.INT1IF == 1) {// INT1 (pin 34) negative edge - Function up
        INTCON3bits.INT1IE = 0;// Disable interrupt
        INTCON3bits.INT1IF = 0;// Reset interrupt flag
        if (debounce1 == 0) {
            if (function >= 10) function = 0;// Set function range 0-9
            else function++;
            if (function == 9) SetupADC(1);// ECG comes from AN1 channel
            else SetupADC(0);// Others come from AN0 channel
            functionBT = function | 0xF0;       // function code for Android
            update = 1;// Signal main() to update LCD display
            debounce1 = 10;// Set switch debounce delay counter decremented by TMR0
        }
```

```
        }
        INTCON3bits.INT1IE = 1;// Enable interrupt
    }
    if (INTCON3bits.INT2IF == 1) {// INT1 (pin 35) either edge - enableBT
        INTCON3bits.INT2IE = 0;// Disable interrupt
        INTCON3bits.INT2IF = 0;// Reset interrupt flag
        if (debounce2 == 0) {
            if (enableBT) {
                enableBT = 0;       // Switching back to LCD display
                SetupSerial();
                INTCON2bits.INTEDG2 = 1;// Set pin 35 (RB2/INT2) for positive edge
                Delay_ms(3000);// Wait until the LCD display is ready
                Backlight(1);       // turn LCD display backlight on
                ClearScreen();      // Clear screen and set cursor to first position
                PrintLine((const unsigned char*)"Function", 8);
                update = 1;
             }
            else {                  // Switching back to Bluetooth
                enableBT = 1;
                SetupBluetooth();
                INTCON2bits.INTEDG2 = 0;// Set pin 35 (RB2/INT2) for negative edge
                update = 1;
            }
            debounce2 = 10;// Set switch debounce delay counter decremented by TMR0
        }
        INTCON3bits.INT2IE = 1;// Enable interrupt
    }
}

void main(){   /****************************** Main program ********************************/
    function = mode = LEDcount = skipCount = counter = debounce0 = debounce1 = 0; // Initialize
    functionBT = function | 0xF0;
    display = do_MOBD = rri_count = 0;
    threshold = 128;// Threshold for the MOBD QRS-detection algorithm
    update = 1;// Flag to signal LCD update
    output = 50;// Baseline for ECG simulation
    sampling[0] = 16;sampling[1] = 17;sampling[2] = 18;sampling[3] = 19;
    sampling[4] = 20;sampling[5] = 25;sampling[6] = 30;sampling[7] = 50;
    sampling[8] = 70;sampling[9] = 88;sampling[10] = 108;sampling[11] = 126;
    sampling[12] = 145;sampling[13] = 173;sampling[14] = 192;sampling[15] = 228;
    TMRcntH[0] = 11;TMRcntH[1] = 26;TMRcntH[2] = 39;TMRcntH[3] = 50;
    TMRcntH[4] = 60;TMRcntH[5] = 99;TMRcntH[6] = 125;TMRcntH[7] = 177;
    TMRcntH[8] = 200;TMRcntH[9] = 211;TMRcntH[10] = 219;TMRcntH[11] = 225;
    TMRcntH[12] = 229;TMRcntH[13] = 233;TMRcntH[14] = 235;TMRcntH[15] = 238;
    TMRcntL[0] = 229;TMRcntL[1] = 66;TMRcntL[2] = 6;     TMRcntL[3] = 114;
    TMRcntL[4] = 185;TMRcntL[5] = 201;TMRcntL[6] = 212;TMRcntL[7] = 233;
    TMRcntL[8] = 60;TMRcntL[9] = 166;TMRcntL[10] = 222;TMRcntL[11] = 9;
    TMRcntL[12] = 25;TMRcntL[13] = 117;TMRcntL[14] = 177;TMRcntL[15] = 232;
    sampling_H = 0xF0;// initialize for 4.167 ms count, Sampling rate = 240 Hz
    sampling_L = 0x7C;// 0xFFFF-0xEFB8 = 0x1047 = 4167, adjust for delay by 68 us
    TRISB = 0b00000111;// RB0-2 as inputs, others outputs, RB3 drives buzzer
    TRISC = 0b11110011;// RC2 as output, 1 KHz to drive LED of PPG
    TRISD = 0b00000000;// Set all port D pins as outputs
    PORTD = 0;// Set port D to 0's
    PORTCbits.RC3 = 0;          // Turn off PPG LED
```

```
596     SetupADC(0);// Call SetupADC() to set up channel 0, AN0 (pin 2)
597     enableBT = PORTBbits.RB2;   // Check for BLUETOOTH enabled
598     SetupSerial();// Set up USART Asynchronous Transmit for LCD display
599     Delay_ms(100);
600     Transmit(18);               // Ctl R to reset BAUD rate to 9600
601     Delay_ms(2500);// Wait until the LCD display is ready
602     if (enableBT == 0) {
603         Backlight(1);           // turn LCD display backlight on
604         ClearScreen();// Clear screen and set cursor to first position
605         PrintLine((const unsigned char*)"  BME 363 Demo", 14);
606         SetPosition(64);// Go to beginning of Line 2;
607         PrintLine((const unsigned char*)" Biomeasurement ",16);// Put your trademark here
608         Delay_ms(3000);
609         ClearScreen();// Clear screen and set cursor to first position
610         PrintLine((const unsigned char*)"Function", 8);
611     }
612     T0CON = 0b10001000;// Turn on TMR0 and use the prescaler 000 (1:2))
613     INTCON = 0b10110000;// GIE(7) = TMR0IE = INT0IE = 1
614     INTCONbits.TMR0IF = PIR1bits.TMR1IF = 0;
615     INTCONbits.TMR0IE = 1;// Enable TMR0 interrupt
616     INTCON2bits.INTEDG0 = 0;// Set pin 33 (RB0/INT0) for negative edge trigger
617     INTCON2bits.INTEDG1 = 0;// Set pin 34 (RB1/INT1) for negative edge trigger
618     if (enableBT) INTCON2bits.INTEDG2 = 0;// Set pin 35 (RB2/INT2) for negative edge
619     else INTCON2bits.INTEDG2 = 1;// Set pin 35 (RB2/INT2) for positive edge
620     INTCONbits.INT0IF = INTCON3bits.INT1IF = INTCON3bits.INT2IF = 0;// Reset interrupt flags
621     INTCONbits.INT0IE = 1;// Enable INT0 interrupt (function down)
622     INTCON3bits.INT1IE = 1;// Enable INT1 interrupt (function up)
623     INTCON3bits.INT2IE = 1;// Enable INT2 interrupt (enableBT)
624     function = 0;
625     while (1) {
626          if (enableBT && PIR1bits.RCIF) { // Wait until USART got data
627                temp = RCREG; // Read received data
628                PIR1bits.RCIF = 0; // Reset RC flag
629                if (temp == 1) { // 1 for increment
630                if (function >= 10) function = 0; // Set function range 0-10
631               else function++;
632                }
633                if (temp == 2) { // 2 for decrement
634                    if (function <= 0) function = 10; // Set function range 0-10
635                   else function--;
636                    }
637                if (function == 9) SetupADC(1); // ECG comes from AN1 channel
638               else SetupADC(0); // Others come from AN0 channel
639                functionBT = function | 0xF0; // function code for Android
640                update = 1; // Signal main() to update LCD display
641           }
642
643         if (update) {// The update flag is set by INT0 or INT1
644             INTCONbits.TMR0IE = 0;// Disable TMR0 interrupt
645             update = 0;// Reset update flag
646             if (!enableBT) {
647                 PrintNum(function, 8);// Update the function number on LCD display
648                 SetPosition(64);// Go to beginning of Line 2;
649                 switch (function) {
```

```
650                     case 0:  PrintLine((const unsigned char*)"Binary counter  ",16); break;
651                     case 1:  PrintLine((const unsigned char*)"ECG simulation  ",16); break;
652                     case 2:  PrintLine((const unsigned char*)"Echo (A/D - D/A)",16); break;
653                     case 3:  PrintLine((const unsigned char*)"Echo @ fs     Hz",16); break;
654                     case 4:  PrintLine((const unsigned char*)"Derivative      ",16); break;
655                     case 5:  PrintLine((const unsigned char*)"Low-pass filter ",16); break;
656                     case 6:  PrintLine((const unsigned char*)"Hi-freq enhance ",16); break;
657                     case 7:  PrintLine((const unsigned char*)"60Hz notch filtr",16); break;
658                     case 8:  PrintLine((const unsigned char*)"Median filter   ",16); break;
659                     case 9:  PrintLine((const unsigned char*)"HR =       bpm  ",16); break;
660                     case 10: PrintLine((const unsigned char*)"PhotoplethysomoG",16); break;
661                 }
662                 enableBT = PORTBbits.RB2; // Check again for BLUETOOTH enabled
663             }
664             if (function) {
665                 LEDcount = function << 4;   // display "function" at the LEDs
666                 PORTB = LEDcount;
667             }
668             INTCONbits.TMR0IE = 1;// Enable TMR0 interrupt
669         }
670         switch (function) {
671         case 0:             // Function 0: Binary counter
672             LEDcount++;// Upcounter
673             PORTB = LEDcount & 0b11110000;// Mask out the lower 4 bits or 0xF0, this output for LEDs
674             PORTD = LEDcount;// Output ramp to verify linearity of the D/A, signal output of D/A
675             Delay_ms(10);// Delay to slow down the counting
676             if (enableBT) {
677                 skipCount++;
678                 if (skipCount == 2) {
679                     TransmitBT(functionBT);
680                     TransmitBT(LEDcount);
681                     TransmitBT(128);
682                     skipCount = 0;
683                 }
684             }
685             break;
686         case 3:// Function 3: Echo (vary rate)
687             INTCONbits.TMR0IE = 0;// Disable TMR0 interrupt
688             if (counter1 != counter && !enableBT) {
689                 temp = sampling[counter];
690                 PrintNum(temp,74);
691                 counter1 = counter;
692             }
693             INTCONbits.TMR0IE = 1;// Enable TMR0 interrupt
694             break;
695         case 9:// Function 9: Multiplication of Backward Differences (MOBD)
696             if (display && !enableBT){// Display Heart Rate in 3 digits
697                 hr = 12000/rri_count;// 60/0.005 = 12000
698                 rri_count = 0;// Reset RRI counter
699                 PrintNum(hr, 71);// Isolates each digit and displays
700                 display = 0;// Reset display flag
701             }
702             break;
703         }
```

```
704     }
705 }
706 // end of file
```