

```

1  /*
2  * File:   WiFi_Test_main.c
3  * Author: Austin
4  *
5  * Created on July 11, 2017, 9:42 PM
6  */
7
8  /* Configuration Bits *****/
9  // FBS
10 #pragma config BWRP = OFF           // Boot Segment Write Protect (Disabled)
11 #pragma config BSS = OFF           // Boot segment Protect (No boot program
12 // flash segment)
13
14 // FGS
15 #pragma config GWRP = OFF           // General Segment Write Protect (General
16 // segment may be written)
17 #pragma config GSS0 = OFF           // General Segment Code Protect (No
18 // Protection)
19
20 // FOSCSEL
21 //#pragma config FNOSC = FRCDIV      // Oscillator Select (8MHz FRC oscillator With
22 // Postscaler (FRCDIV))
23 //#pragma config FNOSC = PRI//SOSC
24 #pragma config FNOSC = FRCPLL
25
26 #pragma config SOSCSRC = DIG         // SOSC Source Type (Analog Mode for use with
27 // crystal) (Digital Mode for not crystal)
28 #pragma config LPRCSEL = HP          // LPRC Oscillator Power and Accuracy (High
29 // Power, High Accuracy Mode)
30 #pragma config IESO = ON             // Internal External Switch Over bit (Internal
31 // External Switchover mode enabled (Two-speed
32 // Start-up enabled))
33
34 // FOSC
35 //#pragma config POSCMOD = NONE      // Primary Oscillator Configuration bits
36 #pragma config POSCMOD = HS          // Primary Oscillator Configuration bits
37 // (Primary oscillator disabled)
38 #pragma config OSCIOFNC = OFF        // CLK0 Enable Configuration bit (CLK0 output
39 // signal is active on the OSC0 pin)
40 #pragma config POSCFREQ = HS         // Primary Oscillator Frequency Range
41 // Configuration bits (Primary oscillator/
42 // external clock input frequency greater than
43 // 8MHz)
44 #pragma config SOSCSSEL = SOSCHP     // SOSC Power Selection Configuration bits
45 // (Secondary Oscillator configured for
46 // high-power operation)
47 #pragma config FCKSM = CSDCMD        // Clock Switching and Monitor Selection (Both
48 // Clock Switching and Fail-safe Clock Monitor
49 // are disabled)
50
51 // FWDTP
52 #pragma config WDTPS = PS32768      // Watchdog Timer Postscale Select bits
53 // (1:32768)
54 #pragma config FWPSA = PR128         // WDT Prescaler bit (WDT prescaler ratio of
55 // 1:128)
56 #pragma config FWDTEN = OFF          // Watchdog Timer Enable bits (WDT disabled in
57 // hardware; SWDTEN bit disabled)
58 #pragma config WINDIS = OFF          // Windowed Watchdog Timer Disable bit
59 // (Standard WDT selected(windowed WDT
60 // disabled))
61
62 // FPOR
63 #pragma config BOREN = BOR0          // Brown-out Reset Enable bits (Brown-out
64 // Reset disabled in hardware, SBOREN bit
65 // disabled)
66 #pragma config LVRCFG = OFF          // (Low Voltage regulator is not available)
67 #pragma config PWRTEN = ON           // Power-up Timer Enable bit (PWRT enabled)
68 #pragma config I2C1SEL = PRI         // Alternate I2C1 Pin Mapping bit (Use Default
69 // SCL1/SDA1 Pins For I2C1)

```

```

70 #pragma config BORV = V20           // Brown-out Reset Voltage bits (Brown-out
71 // Reset set to lowest voltage (2.0V))
72 #pragma config MCLRE = ON           // MCLR Pin Enable bit (RA5 input pin
73 //disabled,MCLR pin enabled)
74
75 // FICD
76 #pragma config ICS = PGx3           // ICD Pin Placement Select bits (EMUC/EMUD
77 // share PGC2/PGD2)
78
79 // FDS
80 #pragma config DSWDTPS = DSWDTPSF // Deep Sleep Watchdog Timer Postscale Select
81 // bits (1:2,147,483,648 (25.7 Days))
82 #pragma config DSWDTOSC = LPRC      // DSWDT Reference Clock Select bit (DSWDT
83 // uses Low Power RC Oscillator (LPRC))
84 #pragma config DSBOREN = OFF        // Deep Sleep Zero-Power BOR Enable bit (Deep
85 // Sleep BOR enabled in Deep Sleep)
86 #pragma config DSWDTEN = ON         // Deep Sleep Watchdog Timer Enable bit (DSWDT
87 // enabled)
88
89
90 #include <p24FV32KA302.h>
91 #include <stdio.h>
92 #include <stdlib.h>
93 #include <stdbool.h>
94
95 #include "test_LEDs.h"
96 #include "pic_frequency.h"
97 #include "limits.h"
98 #include <libpic30.h>
99
100 #include "WiFi.h"
101
102
103 unsigned long timer_ms = 0;
104 unsigned long oldTimer = 0;
105
106 void init( void );
107 void initTimer( void );
108 void initUART1( void );
109
110 /*
111  *
112  */
113 int main( int argc, char** argv )
114 {
115     ANSBbits.ANSB14 = 0;
116     ANSBbits.ANSB15 = 0;
117
118     LED1DIR = 0;
119     LED2DIR = 0;
120
121     LED1SET = 0;
122     LED2SET = 0;
123
124
125     __delay_ms( 500 );
126     for( int inx = 0; inx < 3; inx++ )
127     {
128         if( LED1READ == 0 )
129         {
130             LED1SET = 1;
131         }
132         else
133         {
134             LED1SET = 0;
135         }
136     }
137     __delay_ms( 500 );
138 }

```

```

139     init( );
140
141     wifi( true );
142
143
144     for( int inx = 0; inx < 10; inx++ )
145     {
146         if( LED1READ == 0 )
147         {
148             LED1SET = 1;
149         }
150         else
151         {
152             LED1SET = 0;
153         }
154         __delay_ms( 100 );
155     }
156
157     LED1SET = 0;
158     LED2SET = 0;
159
160
161     LED1SET = 1;
162     __delay_ms( 500 );
163     LED2SET = 1;
164     __delay_ms( 500 );
165     LED1SET = 0;
166     __delay_ms( 500 );
167     LED2SET = 0;
168
169
170     __delay_ms( 1000 );
171
172     //     unsigned long oldTimer = 0;
173
174     U2TXREG = '\r';
175     __delay_ms( 50 );
176     U2TXREG = '\n';
177     __delay_ms( 50 );
178     U2TXREG = 's';
179     __delay_ms( 50 );
180     U2TXREG = 't';
181     __delay_ms( 50 );
182     U2TXREG = 'a';
183     __delay_ms( 50 );
184     U2TXREG = 'r';
185     __delay_ms( 50 );
186     U2TXREG = 't';
187     __delay_ms( 50 );
188     U2TXREG = '\r';
189     __delay_ms( 50 );
190     U2TXREG = '\n';
191     __delay_ms( 50 );
192
193
194     while( 1 )
195     {
196         #define TIMER_MS_COUNT 1000
197         wifi( false );
198         if( TMR1 > TIMER_MS_COUNT )
199         {
200             if( timer_ms == ULONG_MAX )
201             {
202                 timer_ms = 0;
203             }
204             else
205             {
206                 timer_ms++;
207             }

```

```

208         TMR1 = 0x0000;
209     }
210
211     // if( timer_ms > (oldTimer + 1000) )
212     // {
213     //     oldTimer = timer_ms;
214     //     if( LED2READ == 0 )
215     //     {
216     //         LED2SET = 1;
217     //         U2TXREG = 't';
218     //     }
219     //     else
220     //     {
221     //         LED2SET = 0;
222     //     }
223     // }
224
225
226     }
227
228     return (EXIT_SUCCESS);
229 }
230
231 void init( void )
232 {
233     ANSBbits.ANSB14 = 0;
234     ANSBbits.ANSB15 = 0;
235
236     TRISBbits.TRISB14 = 0;
237     TRISBbits.TRISB15 = 0;
238
239     initTimer( );
240
241     //set timer and interrupt to run a ms timer
242     // no more __delay_ms if at all possible
243
244 }
245
246 void initTimer( void )
247 {
248     // set timer up here
249     T1CONbits.TSIDL = 0b1; //Discontinue module operation when device enters idle mode
250     T1CONbits.T1ECS = 0b00; // doesn't matter because we use internal FOSC
251     T1CONbits.TGATE = 0b0; // Gated time accumulation is disabled
252     T1CONbits.TSYNC = 0b0; // Do not synchronize external clock input (asynchronous)
253     T1CONbits.TCS = 0b0; //use internal clock
254
255
256     T1CONbits.TCKPS = 0b01; // Timer 1 Input Clock Prescale (11-256) (10-64) (01-8) (00-1)
257     TMR1 = 0x0000; // start timer at 0
258
259     timer_ms = 0;
260
261     T1CONbits.TON = 0b1; //turn on timer
262
263 }

```