```
1
2   #include <p24FV32KA302.h>
3   #include <stdbool.h>
4   #include <stdlib.h>
5   #include <string.h>
6
7   #include "test_LEDs.h"
8   #include "pic_frequency.h"
9   #include "limits.h"
10  #include <libpic30.h>
11
12  #define CHAR_NULL '\0'
13  #define CHAR_CR '\r'
14  #define CHAR_NEW_LINE '\n'
15
16  #define BUF_LEN 250
17
18  struct struct_buf
19  {
20      char buf[BUF_LEN];
21      unsigned char posWrite;
22      unsigned char posRead;
23  };
24
25  struct struct_buf bufSendWiFi;
26  struct struct_buf bufRecvWiFi;
27
28  struct struct_buf bufSendSPI;
29  struct struct_buf bufRecvSPI;
30
31  extern unsigned long timer_ms;
32
33  void wifi( bool init );
34  void wifiComm( void );
35  void wifiCommandSet( char *command, bool addPrefix );
36  void wifiServer( void );
37  bool wifiResponseCheck( char* response );
38  bool wifiResponseEnd( void );
39  void wifiCommSend( void );
40  bool wifiCommSendChar( char data );
41  bool wifiCommRecv( void );
42  bool wifiCommRecvChar( char *data );
43  void wifiInit( void );
44
45  bool spiCommRecvChar( char *data );
46  bool spiCommRecv( void );
47  bool spiCommSendChar( char data );
48  void spiCommSend( void );
49  void spiComm( void );
50  bool spiResponseEnd( void );
51  void spiCommandSet( char *command );
52  void spiServer( void );
53  void spi( bool init );
54  void spiInit( void );
55
56  void wifiInit( )
57  {
58      ANSBbits.ANSB2 = 0;
59
60      U1MODEbits.UARTEN = 0b0;
61
62  #define UART_BAUD 9600
63
64      U1BRG = (((FCY / 16) / UART_BAUD) - 1);
65
66      U1MODEbits.USIDL = 0b0;
67      U1MODEbits.IREN = 0b0;
68      U1MODEbits.RTSMD = 0b0;
69      U1MODEbits.UEN = 0b00;
```

```
70      U1MODEbits.WAKE = 0b0;
71      U1MODEbits.LPBACK = 0b0;
72      U1MODEbits.ABAUD = 0b0;
73      U1MODEbits.RXINV = 0b0;
74      U1MODEbits.BRGH = 0b0;
75      U1MODEbits.PDSEL = 0b00;
76      U1MODEbits.STSEL = 0b0;
77
78      U1STAbits.URXISEL = 0b00;
79      U1STAbits.UTXINV = 0b0;
80      U1STAbits.UTXBRK = 0b0;
81      U1STAbits.UTXEN = 0b1;
82      U1STAbits.URXISEL = 0b00;
83      U1STAbits.ADDEN = 0b0;
84
85      U1MODEbits.UARTEN = 0b1;
86      U1STAbits.UTXEN = 0b1;
87
88
89
90      ANSBbits.ANSB0 = 0;
91      ANSBbits.ANSB1 = 0;
92
93      U2MODEbits.UARTEN = 0b0;
94
95      U2BRG = (((FCY / 16) / UART_BAUD) - 1);
96
97      U2MODEbits.USIDL = 0b0;
98      U2MODEbits.IREN = 0b0;
99      U2MODEbits.RTSMD = 0b0;
100     U2MODEbits.UEN = 0b00;
101     U2MODEbits.WAKE = 0b0;
102     U2MODEbits.LPBACK = 0b0;
103     U2MODEbits.ABAUD = 0b0;
104     U2MODEbits.RXINV = 0b0;
105     U2MODEbits.BRGH = 0b0;
106     U2MODEbits.PDSEL = 0b00;
107     U2MODEbits.STSEL = 0b0;
108
109     U2STAbits.URXISEL = 0b00;
110     U2STAbits.UTXINV = 0b0;
111     U2STAbits.UTXBRK = 0b0;
112     U2STAbits.UTXEN = 0b1;
113     U2STAbits.URXISEL = 0b00;
114     U2STAbits.ADDEN = 0b0;
115
116     U2MODEbits.UARTEN = 0b1;
117     U2STAbits.UTXEN = 0b1;
118
119     return;
120 }
121
122 void wifi( bool init )
123 {
124     // here we handle all things wifi
125
126     static int stage = 0;
127
128     if( init == true )
129     {
130     bufSendWiFi.posRead = 0;
131     bufSendWiFi.posWrite = 0;
132     bufRecvWiFi.posRead = 0;
133     bufRecvWiFi.posWrite = 0;
134     stage = 0;
135     wifiInit( );
136     spi( true );
137     }
138
```

```
139         wifiComm( );
140         spi( false );
141 
142         static bool runone = false;
143 
144         switch( stage )
145         {
146         case 0: // reset and init
147         // set command
148         wifiCommandSet( "RST", true );
149         stage++;
150         break;
151         case 1:
152         //wait for response
153         if( wifiResponseCheck( "ready" ) == true )
154         {
155             stage++;
156         }
157         break;
158         case 2:
159         wifiCommandSet( "RFPOWER=82", true );
160         stage++;
161         break;
162         case 3:
163         if( wifiResponseCheck( "OK" ) == true )
164         {
165             stage++;
166         }
167         break;
168         case 4:
169         wifiCommandSet( "CWMODE_CUR=1", true );
170         stage++;
171         break;
172         case 5:
173         if( wifiResponseCheck( "OK" ) == true )
174         {
175             stage++;
176         }
177         break;
178         case 6:
179         wifiCommandSet( "CWLAP", true );
180         //wifiCommandSet( "ATE0", false );
181         stage++;
182         break;
183         case 7:
184         if( wifiResponseCheck( "OK" ) == true )
185         {
186             stage++;
187         }
188         break;
189         case 8:
190         wifiCommandSet( "CWJAP_CUR=\"mWiFi\",\"mahadaga\"", true );
191         wifiCommandSet( "CWJAP_CUR=\"AW2\",\"*****\"", true );
192         stage++;
193         break;
194         case 9:
195         if( wifiResponseCheck( "OK" ) == true )
196         {
197             stage++;
198         }
199         break;
200         case 10:
201         wifiCommandSet( "CIFSR", true );
202         stage++;
203         break;
204         case 11:
205         if( wifiResponseCheck( "OK" ) == true )
206         {
207             stage++;
```

```
208         }
209         break;
210         case 12:
211         wifiCommandSet( "CIPMUX=1", true );
212         stage++;
213         break;
214         case 13:
215         if( wifiResponseCheck( "OK" ) == true )
216         {
217             stage++;
218         }
219         break;
220         case 14:
221         wifiCommandSet( "CIPSERVER=0", true );
222         stage++;
223         break;
224         case 15:
225         if( wifiResponseCheck( "OK" ) == true )
226         {
227             stage++;
228         }
229         break;
230         case 16:
231         wifiCommandSet( "CIPSERVER=1", true );
232         stage++;
233         break;
234         case 17:
235         if( wifiResponseCheck( "OK" ) == true )
236         {
237             stage++;
238         }
239         break;
240         case 18:
241         if( runone == false )
242         {
243             //      spiCommandSet( "\r\nConnected!*" );
244             for( int inx = 0; inx < 40; inx++ )
245             {
246             if( LED1READ == 0 )
247             {
248                 LED1SET = 1;
249                 LED2SET = 1;
250             }
251             else
252             {
253                 LED1SET = 0;
254                 LED2SET = 0;
255             }
256             __delay_ms( 25 );
257             runone = true;
258             }
259             LED1SET = 0;
260             LED2SET = 0;
261         }
262         wifiServer( );
263         break;
264         }
265 
266         if( wifiResponseEnd( ) == true )
267         {
268         bufRecvWiFi.posRead = 0;
269         bufRecvWiFi.posWrite = 0;
270         }
271 
272     }
273 
274     void wifiServer( void )
275     {
276 
```

```
277       int inPort = 0;
278       int inDataLen = 0;
279
280       // check the buffers and see if we have a command
281       if( wifiResponseEnd( ) == true )
282       {
283       // check if data came in
284       if( strncmp( "+IPD,", bufRecvWiFi.buf, 5 ) == 0 )
285       {
286           // get the port
287   #define ESP_BUF_IPD_START_PORT 5
288   #define ESP_BUF_IPD_START_LEN 7
289           char tmpbuf[3];
290
291           tmpbuf[0] = bufRecvWiFi.buf[ESP_BUF_IPD_START_PORT];
292           tmpbuf[1] = CHAR_NULL;
293
294           inPort = atoi( tmpbuf );
295
296           tmpbuf[0] = bufRecvWiFi.buf[ESP_BUF_IPD_START_LEN];
297           switch( bufRecvWiFi.buf[ESP_BUF_IPD_START_LEN + 1] )
298           {
299           case ':':
300           tmpbuf[1] = CHAR_NULL;
301           bufRecvWiFi.posRead = 9;
302           break;
303           default:
304           tmpbuf[1] = bufRecvWiFi.buf[ESP_BUF_IPD_START_LEN + 1];
305           tmpbuf[2] = CHAR_NULL;
306           bufRecvWiFi.posRead = 10;
307           break;
308           }
309
310           inDataLen = atoi( tmpbuf );
311
312           // we make things here static in case a command spans multiple data packets
          from the esp8266
313           static bool inCommand = false;
314   #define ESP_BUF_IPD_COMMAND_PROCESS_SIZE 200
315           static char bufRecvWiFiCommand[ ESP_BUF_IPD_COMMAND_PROCESS_SIZE ];
316           static int bufRecvWiFiCommandPos = 0;
317
318           bufRecvWiFiCommandPos = 0;
319
320
321           while( bufRecvWiFi.buf[bufRecvWiFi.posRead ] != CHAR_NULL )
322           {
323           if( inCommand == false )
324           {
325               if( bufRecvWiFi.buf[bufRecvWiFi.posRead] == '!' )
326               {
327               inCommand = true;
328
329               //           if( LED1READ == 0 )
330               //           {
331               //               LED1SET = 1;
332               //           }
333               //           else
334               //           {
335               //               LED1SET = 0;
336               //           }
337               }
338           }
339
340           if( inCommand == true )
341           {
342               bufRecvWiFiCommand[bufRecvWiFiCommandPos] =
              bufRecvWiFi.buf[bufRecvWiFi.posRead ];
343               bufRecvWiFiCommandPos++;
```

```
344             if( bufRecvWiFiCommandPos >= ESP_BUF_IPD_COMMAND_PROCESS_SIZE )
345             {
346             bufRecvWiFiCommandPos = (ESP_BUF_IPD_COMMAND_PROCESS_SIZE - 1);
347             }
348
349             if( bufRecvWiFi.buf[bufRecvWiFi.posRead ] == '*' )
350             {
351             if( LED2READ == 0 )
352             {
353                 LED2SET = 1;
354             }
355             else
356             {
357                 LED2SET = 0;
358             }
359
360             bufRecvWiFiCommand[bufRecvWiFiCommandPos] = CHAR_NULL;
361             inCommand = false;
362
363             spiCommandSet( bufRecvWiFiCommand );
364             bufRecvWiFiCommandPos = 0;
365             }
366         }
367         bufRecvWiFi.posRead++;
368
369         }
370     }
371     bufRecvWiFi.posRead = 0;
372     bufRecvWiFi.posWrite = 0;
373
374
375     }
376 }
377
378 void wifiCommandAdd( char *addCommand )
379 {
380     //AT+CIPSENDBUF=<link ID>,<length>
381
382     const char *commandDataPrefix = "CIPSEND=";
383
384     int inx;
385     char temp[BUF_LEN];
386     int inxtemp;
387
388     inx = 0;
389     inxtemp = 0;
390
391     while( commandDataPrefix[inx] != CHAR_NULL )
392     {
393     temp[inxtemp] = commandDataPrefix[inx];
394     inxtemp++;
395     if( inxtemp >= BUF_LEN )
396     {
397         inxtemp = (BUF_LEN - 1);
398     }
399     inx++;
400     }
401
402     temp[inxtemp] = '0';
403     inxtemp++;
404     temp[inxtemp] = ',';
405     inxtemp++;
406
407
408     // need the length of addCommand
409     int dataLen = 0;
410
411     while( addCommand[dataLen] != '*' )
412     {
```

```
413         dataLen++;
414         }
415         dataLen++; // we need to add one since this is a count, not index
416 
417         char dataLenChar[5];
418 
419         itoa( dataLenChar, dataLen, 10 );
420 
421         inx = 0;
422         while( dataLenChar[inx] != CHAR_NULL )
423         {
424         temp[inxtemp] = dataLenChar[inx];
425         inxtemp++;
426         inx++;
427         }
428 
429         temp[inxtemp] = CHAR_NULL;
430         wifiCommandSet( temp, true );
431 
432         bufSendWiFi.buf[ bufSendWiFi.posWrite] = CHAR_NULL;
433         bufSendWiFi.posWrite++;
434         if( bufSendWiFi.posWrite >= BUF_LEN )
435         {
436         bufSendWiFi.posWrite = (BUF_LEN - 1);
437         }
438 
439         inx = 0;
440         while( addCommand[inx] != '*' )
441         {
442         bufSendWiFi.buf[ bufSendWiFi.posWrite] = addCommand[inx];
443         bufSendWiFi.posWrite++;
444         if( bufSendWiFi.posWrite >= BUF_LEN )
445         {
446             bufSendWiFi.posWrite = (BUF_LEN - 1);
447         }
448         inx++;
449         }
450 
451         bufSendWiFi.buf[ bufSendWiFi.posWrite] = '*';
452         bufSendWiFi.posWrite++;
453         if( bufSendWiFi.posWrite >= BUF_LEN )
454         {
455         bufSendWiFi.posWrite = (BUF_LEN - 1);
456         }
457 
458         return;
459 
460     }
461 
462     void wifiCommandSet( char *command, bool addPrefix )
463     {
464         int inx;
465 
466         const char *commandPrefix = "AT+";
467         const char *commandPostfix = "\r\n";
468 
469         if( addPrefix == true )
470         {
471         inx = 0;
472         while( commandPrefix [inx] != CHAR_NULL )
473         {
474             bufSendWiFi.buf[ bufSendWiFi.posWrite] = commandPrefix[inx];
475             bufSendWiFi.posWrite++;
476             if( bufSendWiFi.posWrite >= BUF_LEN )
477             {
478             bufSendWiFi.posWrite = (BUF_LEN - 1);
479             }
480             inx++;
481         }
```

```
482       }
483
484       inx = 0;
485       while( command[inx] != CHAR_NULL )
486       {
487       bufSendWiFi.buf[bufSendWiFi.posWrite] = command[inx];
488       bufSendWiFi.posWrite++;
489       if( bufSendWiFi.posWrite >= BUF_LEN )
490       {
491           bufSendWiFi.posWrite = (BUF_LEN - 1);
492       }
493       inx++;
494       }
495
496       inx = 0;
497       while( commandPostfix [inx] != CHAR_NULL )
498       {
499       bufSendWiFi.buf[ bufSendWiFi.posWrite] = commandPostfix[inx];
500       bufSendWiFi.posWrite++;
501       if( bufSendWiFi.posWrite >= BUF_LEN )
502       {
503           bufSendWiFi.posWrite = (BUF_LEN - 1);
504       }
505       inx++;
506       }
507
508       return;
509
510   }
511
512   bool wifiResponseCheck( char* response )
513   {
514       bool match = false;
515
516       if( wifiResponseEnd( ) == true )
517       {
518       match = true;
519       int inx = 0;
520
521       while( (response[inx] != CHAR_NULL) && (match == true) )
522       {
523           if( response[inx] != bufRecvWiFi.buf[inx] )
524           {
525           match = false;
526           }
527           inx++;
528       }
529       }
530
531       if( match == true )
532       {
533       if( LED1READ == 0 )
534       {
535           LED1SET = 1;
536       }
537       else
538       {
539           LED1SET = 0;
540       }
541       }
542
543       return match;
544   }
545
546   bool wifiResponseEnd( void )
547   {
548       bool responseEnd = false;
549       if( bufRecvWiFi.posWrite >= 2 )
550       {
```

```
551         if( bufRecvWiFi.buf[bufRecvWiFi.posWrite - 2] == CHAR_CR )
552         {
553             if( bufRecvWiFi.buf[bufRecvWiFi.posWrite - 1] == CHAR_NEW_LINE )
554             {
555             responseEnd = true;
556             }
557
558
559         }
560         }
561
562         return responseEnd;
563     }
564
565     void wifiComm( )
566     {
567         wifiCommSend( );
568         wifiCommRecv( );
569
570         return;
571     }
572
573     void wifiCommSend( void )
574     {
575         if( bufSendWiFi.posRead != bufSendWiFi.posWrite )
576         {
577         if( U1STAbits.UTXBF == 0 )
578         {
579             if( wifiCommSendChar( bufSendWiFi.buf[bufSendWiFi.posRead] ) == true )
580             {
581             bufSendWiFi.posRead++;
582             if( bufSendWiFi.posRead >= BUF_LEN )
583             {
584                 bufSendWiFi.posRead = 0;
585             }
586             }
587         }
588         }
589         else
590         {
591         bufSendWiFi.posRead = 0;
592         bufSendWiFi.posWrite = 0;
593         }
594         return;
595     }
596
597     bool wifiCommSendChar( char data )
598     {
599         bool dataSent = false;
600         static bool pause = false;
601         static unsigned long pauseTimer;
602         static bool pauseTimerOverflowWait;
603     #define PAUSE_TIME_MS 100
604
605         if( pause == false )
606         {
607         if( data == CHAR_NULL )
608         {
609             pause = true;
610             if( (ULONG_MAX - timer_ms) > PAUSE_TIME_MS )
611             {
612             pauseTimer = timer_ms + PAUSE_TIME_MS;
613             }
614             else
615             {
616             pauseTimer = PAUSE_TIME_MS - (ULONG_MAX - timer_ms);
617             pauseTimerOverflowWait = true;
618             }
619
```

```
620             dataSent = true;
621         }
622         else
623         {
624             if( U1STAbits.UTXBF == 0 )
625             {
626             U1TXREG = data;
627             dataSent = true;
628             //      U2TXREG = data;
629             }
630         }
631         }
632         else
633         {
634         if( pauseTimerOverflowWait == true )
635         {
636             if( timer_ms < pauseTimer )
637             {
638             pauseTimerOverflowWait = false;
639             }
640         }
641
642         if( pauseTimerOverflowWait == false )
643         {
644             if( timer_ms >= pauseTimer )
645             {
646             pause = false;
647             }
648         }
649         }
650
651         return dataSent;
652     }
653
654     bool wifiCommRecv( void )
655     {
656         bool dataReceived = false;
657         char data;
658
659         if( wifiCommRecvChar( &data ) == true )
660         {
661         bufRecvWiFi.buf[bufRecvWiFi.posWrite] = data;
662         dataReceived = true;
663         bufRecvWiFi.posWrite++;
664         if( bufRecvWiFi.posWrite >= BUF_LEN )
665         {
666             bufRecvWiFi.posWrite = 0;
667         }
668         bufRecvWiFi.buf[bufRecvWiFi.posWrite] = CHAR_NULL;
669         }
670
671         return dataReceived;
672     }
673
674     bool wifiCommRecvChar( char *data )
675     {
676         bool dataReceived = false;
677
678         if( U1STAbits.URXDA == 1 )
679         {
680         *data = U1RXREG;
681         dataReceived = true;
682         U2TXREG = *data;
683         }
684
685         return dataReceived;
686     }
687
688     void spi( bool init )
```

```
{
    if( init == true )
    {
    bufSendSPI.posRead = 0;
    bufSendSPI.posWrite = 0;
    bufRecvSPI.posRead = 0;
    bufRecvSPI.posWrite = 0;
    spiInit( );
    U2TXREG = '\r';
    __delay_ms( 100 );
    U2TXREG = '\n';

    }

    static bool spiInReset = false;
    if( PORTBbits.RB12 == 1 )
    {
    if( spiInReset == false )
    {
        spiInReset = true;
        SPI2STATbits.SPIEN = 0; //disable SPI
    }
    }
    else
    {
    if( spiInReset == true )
    {

        spiInReset = false;
        bufRecvSPI.posRead = 0;
        bufRecvSPI.posWrite = 0;
        //      spiCommandSet( "!Set;Watts;55*" );
        SPI2STATbits.SPIEN = 1; //enable SPI
    }

    spiComm( );
    spiServer( );

    // here we need to check if we pass on a command

    if( spiResponseEnd( ) == true )
    {
        bufRecvSPI.posRead = 0;
        bufRecvSPI.posWrite = 0;
    }
    }
}

void spiServer( )
{

    // check the buffers and see if we have a command
    if( spiResponseEnd( ) == true )
    {
    //  LED1SET = 1;

    if( bufRecvSPI.buf[0] == '!' )
    {
        // LED1SET = 1;
        wifiCommandAdd( bufRecvSPI.buf );
    }
    }
}

void spiCommandSet( char *command )
{
    int inx;

    inx = 0;
```

```
758         while( command[inx] != '*' )
759         {
760         //LED1SET = 1;
761         bufSendSPI.buf[bufSendSPI.posWrite] = command[inx];
762         bufSendSPI.posWrite++;
763         if( bufSendSPI.posWrite >= BUF_LEN )
764         {
765             bufSendSPI.posWrite = (BUF_LEN - 1);
766         }
767         inx++;
768         }
769         bufSendSPI.buf[bufSendSPI.posWrite] = command[inx];
770         bufSendSPI.posWrite++;
771         if( bufSendSPI.posWrite >= BUF_LEN )
772         {
773         bufSendSPI.posWrite = (BUF_LEN - 1);
774         }
775
776         return;
777     }
778
779     bool spiResponseEnd( void )
780     {
781         bool responseEnd = false;
782
783         if( bufRecvSPI.buf[ bufRecvSPI.posWrite - 1 ] == '*' )
784         {
785         //  if( LED1READ == 0 )
786         //  {
787         //      LED1SET = 1;
788         //  }
789         //  else
790         //  {
791         //      LED1SET = 0;
792         //  }
793
794         responseEnd = true;
795         }
796
797         return responseEnd;
798     }
799
800     void spiComm( void )
801     {
802         spiCommSend( );
803         spiCommRecv( );
804
805         return;
806     }
807
808     void spiCommSend( void )
809     {
810         if( bufSendSPI.posRead != bufSendSPI.posWrite )
811         {
812         //LED1SET = 1;
813         //  if( U2STAbits.UTXBF == 0 )
814         //  {
815         if( spiCommSendChar( bufSendSPI.buf[bufSendSPI.posRead] ) == true )
816         {
817             bufSendSPI.posRead++;
818             if( bufSendSPI.posRead >= BUF_LEN )
819             {
820             bufSendSPI.posRead = 0;
821             }
822         }
823         //  }
824         }
825         else
826         {
```

```
827         bufSendSPI.posRead = 0;
828         bufSendSPI.posWrite = 0;
829         }
830         return;
831     }
832 
833     //bool spiCommSendCharUART( char data )
834     //{
835     //    bool dataSent = false;
836     //
837     //    if( U2STAbits.UTXBF == 0 )
838     //    {
839     //  U2TXREG = data;
840     //  dataSent = true;
841     //    }
842     //
843     //    return dataSent;
844     //}
845 
846     bool spiCommSendChar( char data )
847     {
848         bool sendGood = false;
849 
850 
851         if( SPI2STATbits.SPITBF == 0b0 ) //if in enhance mode use SPI1STATbits.SR1MPT
852         {
853         SPI2BUF = data;
854         sendGood = true;
855         //  U2TXREG = data;
856         }
857 
858         return sendGood;
859     }
860 
861     bool spiCommRecv( void )
862     {
863         bool dataReceived = false;
864         char data;
865 
866         if( spiCommRecvChar( &data ) == true )
867         {
868         bufRecvSPI.buf[bufRecvSPI.posWrite] = data;
869         dataReceived = true;
870         bufRecvSPI.posWrite++;
871         if( bufRecvSPI.posWrite >= BUF_LEN )
872         {
873             bufRecvSPI.posWrite = 0;
874         }
875         bufRecvSPI.buf[bufRecvSPI.posWrite] = CHAR_NULL;
876         }
877 
878         return dataReceived;
879     }
880 
881     //bool spiCommRecvCharUART( char *data )
882     //{
883     //    bool dataReceived = false;
884     //
885     //    if( U2STAbits.URXDA == 1 )
886     //    {
887     //  *data = U2RXREG;
888     //  dataReceived = true;
889     //    }
890     //
891     //    return dataReceived;
892     //}
893 
894     bool spiCommRecvChar( char *data )
895     {
```

```
896         bool recvGood = false;
897 
898         if( SPI2STATbits.SPIRBF == 0b1 )
899         {
900         *data = SPI2BUF;
901         recvGood = true;
902         if( *data != '@' )
903         {
904             U2TXREG = *data;
905         }
906         }
907 
908         return recvGood;
909     }
910 
911     void spiInit( void )
912     {
913         // make sure analog is turned off - it messes with the pins
914         ANSA = 0;
915         ANSB = 0;
916         ANSBbits.ANSB12 = 0;
917 
918         TRISBbits.TRISB5 = 1;
919         TRISBbits.TRISB6 = 0;
920 
921         TRISBbits.TRISB12 = 1;
922         TRISAbits.TRISA7 = 1;
923 
924         //SPI2 Initialize as Slave
925         SPI2CON1bits.MSTEN = 0;
926 
927         SPI2CON1bits.DISSCK = 0b1; // SPI clock disabled
928         SPI2CON1bits.DISSDO = 0b0; // SDO used
929         SPI2CON1bits.MODE16 = 0b0; // 8 bit mode
930         SPI2CON1bits.SMP = 0b0; // sample phase mode middle
931         SPI2CON1bits.CKE = 0b1; // serial data changes on active to idle clock state
932         SPI2CON1bits.SSEN = 0b1; // yes a slave
933         SPI2CON1bits.CKP = 0b1; // clock idle is high
934         SPI2CON1bits.SPRE = 0b000; // secondary prescale 8:1, not used - no clock is run
935 
936         SPI2CON2bits.FRMEN = 0b0; // frame mode, unused
937         SPI2CON2bits.SPIFSD = 0b0; // frame mode, unused
938         SPI2CON2bits.SPIFPOL = 0b0; // frame mode, unused
939         SPI2CON2bits.SPIFE = 0b0; // frame mode, unused
940 
941         SPI2CON2bits.SPIBEN = 0b0; // 1=enhanced buffer mode
942 
943         SPI2STATbits.SPIROV = 0; //clear flag for overflow data
944 
945         SPI2BUF = SPI2BUF; //clear the buffer
946         SPI2STATbits.SPIEN = 1; //enable SPI
947 
948         return;
949     }
950 
```