```
1    // ============================================================================
2    // Vernon Johnson Engineering
3    //
4    // http://www.vernonjohnson.net
5    //
6    // REMS project url = dev.vernonjohnson.net/myPIC32/
7    //
8    // ============================================================================
9    //
10   // Code Project:    Retrofit Engine Management System Diagnostics.
11   //
12   //
13   // Description:     Diagnostic Monitor for REMS myPIC32 Design Challenge:
14   //
15   //                  1.  Inital Template REMS Prototype.
16   //                      First test performed on PIC32 Starter Kit Board.
17   //
18   // ============================================================================
19   // Filename:    REMS_DIAG_a.c   "main.c"
20   // ============================================================================
21   // Author:      Vernon Johnson
22   // Company:     Vernon Johnson Engineering
23   // Revision:    1.0.011609.3
24   // Date:        01/16/2009
25   // ============================================================================
26   // Compiled using MPLAB-C32 V1.03 on MPLAB IDE v 8.14.
27   // for device: PIC32MX360F512L on PIC32 Starter Kit Board / REMS 324.001 rev.A.
28   // ============================================================================
29   // Project History:
30   //
31   //
32   // 01/11/2009   1.  New Project created (REMS_Template_mmddyy).
33   //
34   // 01/12/2009   2.  Configuration for PWM,Timers 1 and 5, app. specific I/O.
35   //
36   // 01/16/2009   3.  Bring up and test serial monitor functions.
37   //                  Initial ADC and Digital Input templates.
38   //
39   // ============================================================================
40   //
41   //  ADDITIONAL NOTES:
42   //
43   //      Configuration is establised in code for this template.
44   //
45   // ============================================================================
46   //
47   //                          Reference Declarations
48   //
49   // ============================================================================
50
51   // ============================================================================
52   //
53   //                          Include Declarations
54   //
55   // ============================================================================
56   #include <plib.h> // PIC32MX Standard Peripheral Library.
57   #include <stdlib.h>
58   #include <stdio.h>
59   #include "REMS_Data.h"
60   #include "typeconvert.h"
61
62
63   // ============================================================================
64   //
65   //                          MCU Device Configuration
66   //
67   // ============================================================================
68   // Device Configuration for PIC32 Starter Kit.
69   // Setting each config parameter on a seperate line makes it convenient to
70   // manipulate the parmaters seperately.
71
```

```
72  #pragma config FPLLODIV = DIV_1 // System PLL Clock Divider
73  //#pragma config FPLLMUL = MUL_18 // PLL Output Multiplier (EX16 was 20 ???).
74  #pragma config FPLLMUL = MUL_20 // PLL Output Multiplier (EX16 was 20 ???).
75  #pragma config FPLLIDIV = DIV_2 // PLL Input Divider.
76  #pragma config FWDTEN = OFF // Wattchdog OFF.
77  #pragma config WDTPS = PS1048576 // WDPS = Max.
78  #pragma config FCKSM = CSDCMD // Clk Sw Disabled, Fail Safe Clk Mon Disabled.
79  #pragma config FPBDIV = DIV_8 // Boot Up PBCLK Divider.
80  #pragma config OSCIOFNC = OFF // CLKO (output on OSC0 pin) Disabled.
81  #pragma config POSCMOD = HS // Primary Oscillator = HS.
82  #pragma config FSOSCEN = OFF // Secondary Oscillator Disabled.
83  #pragma config FNOSC = PRIPLL// Oscillator selection = PRIMARY + PLL.
84  #pragma config CP = OFF // Code Protect Disabled.
85  #pragma config BWP = OFF // Boot Flash write protect OFF.
86  #pragma config PWP = OFF // Program Flash write protect OFF.
87  #pragma config ICESEL = ICS_PGx1 // ICE pins are shared with PGC1, PGD1.
88  #pragma config DEBUG = OFF // ICE Background debugger = OFF.
89  //#pragma config DEBUG = ON // ICE Background debugger = ON.
90
91
92  // ============================================================================
93  //
94  //                              Function Declarations
95  //
96  // ============================================================================
97
98
99  // ============================================================================
100 //
101 //                          ISR Function Declarations
102 //
103 // ============================================================================
104 //
105 //  ISR function Name:  UART 1 interrupt handler
106 //  Description:        Serial Port 1 Receive Interrupt Handler(no int on send).
107 //                      Reads data from UART 1.
108 //                      Priority level 4.
109 //                      SubPriority level = default (none specified).
110 //  Inputs:             None.
111 //  Returns:            None.
112 //  NOTE:               This version = 1.0.011509.1
113 void __ISR(_UART1_VECTOR, ipl4) IntUart1Handler(void)
114 {
115     // Is this an RX interrupt?
116     if(mU1RXGetIntFlag())
117     {
118         // Local variable for the char in U1RXREG.
119         unsigned char RX1_char;
120         RX1_char = U1RXREG;
121
122         // Add the received character to the receive buffer
123         if (rx1_ptr < PACKET_LENGTH)
124         {
125             receive1_buffer[rx1_ptr++] = RX1_char;
126         }
127         else
128         {
129             // indicate an overrun
130             processInput = 10;
131         }
132
133         // Check for CR char and set processInput flag if true
134         if ( RX1_char == 0x0D )
135         {
136             processInput = 1;
137         }
138         else
139         {
140             processInput = 0;
141         }
142
```

```
143         //custom code
144         while(!U1STAbits.TRMT); // Echo Back Received Character.
145         U1TXREG = RX1_char;
146 
147         // Clear the RX interrupt Flag
148         mU1RXClearIntFlag();
149 //
150 //      // Echo what we just received.
151 //      putcUART1(ReadUART1());
152 
153         // Toggle LED to indicate UART1 activity.
154 
155         // PIC32 Starter.
156         mPORTDToggleBits(BIT_1);
157 
158     }
159 
160     // We don't care about TX interrupt
161     if ( mU1TXGetIntFlag() )
162     {
163         mU1TXClearIntFlag();
164     }
165 }
166 
167 //  ISR function Name:  UART 2 interrupt handler
168 //  Description:        Serial Port 2 Receive Interrupt Handler(no int on send).
169 //                      Reads data from UART 2.
170 //                      Priority level 2.
171 //                      SubPriority level = default (none specified).
172 //  Inputs:             None.
173 //  Returns:            None.
174 //  NOTE:               This version = 1.0.011509.1
175 void __ISR(_UART2_VECTOR, ipl2) IntUart2Handler(void)
176 {
177     // Is this an RX interrupt?
178     if(mU2RXGetIntFlag())
179     {
180         // Clear the RX interrupt Flag
181         mU2RXClearIntFlag();
182 
183         // Echo what we just received.
184         putcUART2(ReadUART2());
185 
186 
187         // Toggle LED to indicate UART2 activity
188 
189         // PIC32 Starter.
190         mPORTDToggleBits(BIT_2);
191 
192     }
193 
194     // We don't care about TX interrupt
195     if ( mU2TXGetIntFlag() )
196     {
197         mU2TXClearIntFlag();
198     }
199 }
200 
201 //  ISR function Name:  OC1 interrupt handler
202 //  Description:        Output Compare Module 1 Interrupt Handler.
203 //                      Used for Fuel Injection PWM control.
204 //                      Priority level 4.
205 //                      SubPriority level = default (none specified).
206 //  Inputs:             None.
207 //  Returns:            None.
208 //  NOTE:               This version = 1.0.011509.1
209 void __ISR(_OC1_VECTOR, ipl4) OC1IntHandler(void)
210 {
211     // Perform interrupt handler tasks.
212     IFS0CLR = 0x00000080; // Clear the OC1 Interrupt flag.
213 }
```

```
214
215 // Timer 1 interrupt handler provides a 1 Second interval.
216 void __ISR(_TIMER_1_VECTOR, ipl2) Timer1Handler(void)
217 {
218     // clear the interrupt flag
219     mT1ClearIntFlag();
220
221     // .. things to do
222     // .. in this case, toggle the LED
223     mPORTDToggleBits(BIT_1);
224
225     // Set the GoADC flag.
226     GoADC = 1;
227 }
228
229 // Timer 5 interrupt handler is provided to gaste the Fuel Injectors.
230 void __ISR(_TIMER_5_VECTOR, ipl7) Timer5Handler(void)
231 {
232     // clear the interrupt flag
233     mT5ClearIntFlag();
234
235     // .. things to do
236     // .. in this case, toggle the LED
237     mPORTDToggleBits(BIT_2);
238
239     switch(Sequencer)
240     {
241         case 0:
242                 FGATE1 = 1;
243                 FGATE2 = 0;
244                 FGATE3 = 0;
245                 FGATE4 = 0;
246                 Sequencer += 1;
247                 break;
248         case 1:
249                 FGATE1 = 0;
250                 FGATE2 = 1;
251                 FGATE3 = 0;
252                 FGATE4 = 0;
253                 Sequencer += 1;
254                 break;
255         case 2:
256                 FGATE1 = 0;
257                 FGATE2 = 0;
258                 FGATE3 = 1;
259                 FGATE4 = 0;
260                 Sequencer += 1;
261                 break;
262         case 3:
263                 FGATE1 = 0;
264                 FGATE2 = 0;
265                 FGATE3 = 0;
266                 FGATE4 = 1;
267                 Sequencer = 0;
268                 break;
269         default:
270                 Sequencer = 0;
271                 break;
272     }
273 }
274
275
276
277 // ============================================================================
278 //
279 //                          Buffer Function Declarations
280 //
281 // ============================================================================
282
283 HR_Sensor_Buffer Init_HR_SensorBuffer( HR_Sensor_Buffer buffer )
284 {
```

```
285
286     // Initialize buffer
287     buffer.startPointer = 0;
288     buffer.currentItem = 0;
289     buffer.endPointer = 1;
290     buffer.overflow = 0;
291     buffer.overflow = 0;
292     return(buffer);
293 }
294
295 //
296 //  Function Name:  Init_LR_SensorBuffer.
297 //  Description:    Initializes a Low Resolution SensorBuffer.
298 //  Inputs:         HR_Sensor_Buffer type.
299 //  Returns:        HR_Sensor_Buffer type.
300 LR_Sensor_Buffer Init_LR_SensorBuffer( LR_Sensor_Buffer buffer )
301 {
302
303     // Initialize buffer
304     buffer.startPointer = 0;
305     buffer.currentItem = 0;
306     buffer.endPointer = 1;
307     buffer.overflow = 0;
308     buffer.overflow = 0;
309     return(buffer);
310 }
311
312 //
313 //  Function Name:  Service_HR_Buffer.
314 //  Description:    updates the High Resolution sensor Acquisition buffer(s).
315 //  Inputs:         uint_16, HR_Sensor_Buffer type.
316 //  Returns:        HR_Sensor_Buffer type.
317 HR_Sensor_Buffer Service_HR_Buffer( uint_16 input, HR_Sensor_Buffer buffer )
318 {
319     // Always increment the total counter
320     buffer.count++;
321
322     if ( buffer.endPointer <= HiResBufLen )
323         {
324             buffer.item[buffer.currentItem] = input;
325             buffer.currentItem++;
326             buffer.endPointer++;
327             return (buffer);
328         }
329     else
330         {
331             buffer.overflow = 1;
332             buffer.endPointer =  1;
333             buffer.currentItem = 0;
334             buffer.startPointer = 0;
335             return (buffer);
336         }
337 }
338
339 //
340 //  Function Name:  Service_LR_Buffer.
341 //  Description:    updates the Low Resolution sensor Acquisition buffer(s).
342 //  Inputs:         uint_16, LR_Sensor_Buffer type.
343 //  Returns:        HR_Sensor_Buffer type.
344 LR_Sensor_Buffer Service_LR_Buffer( uint_16 input, LR_Sensor_Buffer buffer )
345 {
346     // Always increment the total counter
347     buffer.count++;
348
349     if ( buffer.endPointer <= LoResBufLen )
350         {
351             buffer.item[buffer.currentItem] = input;
352             buffer.currentItem++;
353             buffer.endPointer++;
354             return (buffer);
355         }
```

```
356      else
357          {
358              buffer.overflow = 1;
359              buffer.endPointer =  1;
360              buffer.currentItem = 0;
361              buffer.startPointer = 0;
362              return (buffer);
363          }
364  }
365
366  //
367  //  Function Name:  HR_BufferDump.
368  //  Description:    Processes the input stream.
369  //  Inputs:         HR_Sensor_Buffer.
370  //  Returns:        None.
371  void HR_BufferDump(HR_Sensor_Buffer buffer)
372  {
373      char putCountString[11]; // 11 should be enough for signed long.
374
375      for ( loopIndex = 0; loopIndex <= HiResBufLen - 2; loopIndex++)
376          {
377              putsUART1(uitoa(putCountString,
378                          buffer.item[loopIndex]));
379              putsUART1(",");
380          }
381          putsUART1(uitoa(putCountString,
382                      buffer.item[HiResBufLen - 1]));
383  }
384
385  //
386  //  Function Name:  LR_BufferDump.
387  //  Description:    Processes the input stream.
388  //  Inputs:         LR_Sensor_Buffer.
389  //  Returns:        None.
390  void LR_BufferDump(LR_Sensor_Buffer buffer)
391  {
392      char putCountString[11]; // 11 should be enough for signed long.
393
394      for ( loopIndex = 0; loopIndex <= LoResBufLen - 2; loopIndex++)
395          {
396              putsUART1(uitoa(putCountString,
397                          buffer.item[loopIndex]));
398              putsUART1(",");
399          }
400          putsUART1(uitoa(putCountString,
401                      buffer.item[LoResBufLen - 1]));
402  }
403
404  // ============================================================================
405  //
406  //                          END - Buffer Function Declarations
407  //
408  // ============================================================================
409
410
411  // ============================================================================
412  //
413  //                          Terminal I/O Function Declarations
414  //
415  // ============================================================================
416
417  //
418  //  Function Name: U1CRLF
419  //  Description:   low level CRLF for UART1
420  //  Inputs:        None
421  //  Returns:       None
422
423  void U1CRLF( void )
424  {
425
426      asm("nop");
```

```
427     asm("nop");
428     while(!U1STAbits.TRMT);     // Wait for TXREG to clear.
429     U1TXREG = 13;               // Send a CR
430     asm("nop");
431     asm("nop");
432     while(!U1STAbits.TRMT);     // Wait for TXREG to clear.
433     U1TXREG = 10;               // Send a LF
434
435 }
436
437
438 //
439 //  Function Name: U2CRLF
440 //  Description:   low level CRLF for UART2
441 //  Inputs:        None
442 //  Returns:       None
443
444 void U2CRLF( void )
445 {
446
447     asm("nop");
448     asm("nop");
449     while(!U2STAbits.TRMT);     // Wait for TXREG to clear.
450     U2TXREG = 13;               // Send a CR
451     asm("nop");
452     asm("nop");
453     while(!U2STAbits.TRMT);     // Wait for TXREG to clear.
454     U2TXREG = 10;               // Send a LF
455
456 }
457
458 //
459 //  Function Name:  InitRX1.
460 //  Description:    Initializes the Receive buffer pointers.
461 //  Inputs:         None.
462 //  Returns:        None.
463 void InitRX1(void)
464 {
465     rx1_ptr = 0;
466     rx1_buf = 0;
467 }
468
469 //
470 //  Function Name:  InitRX2.
471 //  Description:    Initializes the Receive buffer pointers.
472 //  Inputs:         None.
473 //  Returns:        None.
474 void InitRX2(void)
475 {
476     rx2_ptr = 0;
477     rx2_buf = 0;
478 }
479
480 //  Function Name:  ProcessInput.
481 //  Description:    Processes the input stream.
482 //  Inputs:         None.
483 //  Returns:        None.
484 void ProcessInput(void)
485 {
486     // use ASCII to integer conversion on first char in line (command).
487     char cmdChar;
488
489     // test strings for valid commands.
490     const char CT_cmdString[] = "d CT";
491     const char TPS_cmdString[] = "d TPS";
492     const char RPM_cmdString[] = "d RPM";
493     const char BV_cmdString[] = "d BV";
494
495     cmdChar = receive1_buffer[0];
496
497     U1CRLF();
```

```
498      U1CRLF();
499
500      if ( (cmdChar == 0x3F) | (cmdChar == 0x48) | (cmdChar == 0x68) )
501          {
502              // If "?"... display help message.
503              putsUART1(helpString1);
504              U1CRLF();
505              U1CRLF();
506              putsUART1(helpString2);
507              U1CRLF();
508              putsUART1(helpString3);
509              U1CRLF();
510              U1CRLF();
511              putsUART1(helpString4);
512              U1CRLF();
513              U1CRLF();
514              putsUART1(helpString5);
515          }
516      else if ( (cmdChar == 0x53) | (cmdChar == 0x73) )
517          {
518              // If "S" or "s"... Return Status.
519              putsUART1(statusString1);
520              U1CRLF();
521              putsUART1(statusString2);
522          }
523      else if ( (cmdChar == 0x52) | (cmdChar == 0x72) )
524          {
525              // If "R" or "r"... Reset me.
526              U1CRLF();
527              U1CRLF();
528  //          CloseUART2();
529  //          asm ( "GOTO 0x00000" );
530  //          exit(0);
531  //          //  Soft Reset needs some work (VJ 01/16/2009).
532              U1CRLF();
533              U1CRLF();
534              U1CRLF();
535              U1CRLF();
536  //          Initialize();
537          }
538      else if ( ! strncmp((char *)receive1_buffer, CT_cmdString, 4) )
539          {
540              putsUART1("Coolant Temperature Buffer dump... ");
541              HR_BufferDump(CT_Buffer);
542          }
543      else if ( ! strncmp((char *)receive1_buffer, TPS_cmdString, 5) )
544          {
545              putsUART1("Throttle Position Sensor Buffer dump... ");
546              HR_BufferDump(TPS_Buffer);
547          }
548      else if ( ! strncmp((char *)receive1_buffer, RPM_cmdString, 5) )
549          {
550              putsUART1("Engine Speed Buffer dump... ");
551              HR_BufferDump(RPM_Buffer);
552          }
553      else if ( ! strncmp((char *)receive1_buffer, BV_cmdString, 4) )
554          {
555              putsUART1("Battery Voltage Buffer dump... ");
556              HR_BufferDump(BV_Buffer);
557          }
558      else
559          {
560              putsUART1(errorString);
561          }
562      U1CRLF();
563      U1CRLF();
564      putsUART1(titleString1);
565      U1CRLF();
566      U1CRLF();
567      putsUART1(titleString5);
568      U1CRLF();
```

```
569      U1CRLF();
570      InitRX1();
571  }
572
573  // ============================================================================
574  //
575  //                      END - Terminal I/O Function Declarations
576  //
577  // ============================================================================
578
579  // ============================================================================
580  //
581  //                           Begin: ADC functions section
582  //
583  // ============================================================================
584
585  //  Function Name:  InitADC.
586  //  Description:    Initialize the ADC Module.
587  //  Inputs:         int amask provides the configuration of the ADC input pins.
588  //  Returns:        None.
589  void InitADC(int amask)
590  {
591      AD1PCFG = amask;        // Select analog input pins.
592      AD1CON1 = 0x00E0;       // Automatic Converssion after sampling.
593      AD1CSSL = 0;            // No scanning.
594      AD1CON2 = 0;            // Use MUXa, AVdd & AVss as Vref +/-.
595      AD1CON3 = 0x1F3F;       // Tsamp = 32 Tad.
596      AD1CON1bits.ADON = 1;   // Turn on the ADC.
597  }
598
599  //  Function Name:  ReadADC.
600  //  Description:    Read an ADC channel.
601  //  Inputs:         int ch = the channel to sample.
602  //  Returns:        int ADC1BUF0 = the result of the conversion.
603  int ReadADC(int ch)
604  {
605      AD1CHSbits.CH0SA = ch;      // Select a channel to sample.
606      AD1CON1bits.SAMP = 1;       // Start the sampleing.
607      while(!AD1CON1bits.DONE);   // Wait until conversion is complete.
608      return ADC1BUF0;            // Return conversion result.
609  }
610
611
612  //  Function Name:  GetAllADC.
613  //  Description:    Rotates through all ADC channels.
614  //  Inputs:         none.
615  //  Returns:        none.
616  void GetAllADC(void)
617  {
618      switch (ADCRevolver)
619      {
620          case 0:
621          case 1:
622                  ADCRevolver++;
623                  break;
624          case 2:
625                  ADC_2 = ReadADC(2);
626                  ADCRevolver++;
627                  break;
628          case 3:
629                  ADC_3 = ReadADC(3);
630                  ADCRevolver++;
631                  break;
632          case 4:
633                  ADC_4 = ReadADC(4);
634                  ADCRevolver++;
635                  break;
636          case 5:
637                  ADC_5 = ReadADC(5);
638                  ADCRevolver++;
639                  break;
```

```
640         case 6:
641                 ADC_6 = ReadADC(6);
642                 ADCRevolver++;
643                 break;
644         case 7:
645                 ADC_7 = ReadADC(7);
646                 ADCRevolver++;
647                 break;
648         case 8:
649                 ADC_8 = ReadADC(8);
650                 ADCRevolver++;
651                 break;
652         case 9:
653                 ADC_9 = ReadADC(9);
654                 ADCRevolver = 2;
655                 break;
656         default:
657                 ADCRevolver = 2;
658                 break;
659     }
660 }
661 // ============================================================================
662 //
663 //                           End: ADC functions section
664 //
665 // ============================================================================
666
667 // ============================================================================
668 //
669 //                      Begin: Digital Input functions section
670 //
671 // ============================================================================
672
673 //  Function Name:  ReadDigitalInputs.
674 //  Description:    Scans and returns the state of the digital inputs.
675 //  Inputs:         None.
676 //  Returns:        The 8 bit current state of the digital inputs.
677 unsigned char ReadDigitalInputs(void)
678 {
679     return (PORTD>>6);
680 }
681 // ============================================================================
682 //
683 //                        End: Digital Input functions section
684 //
685 // ============================================================================
686
687 // ============================================================================
688 //
689 //                          Begin: Initialization section
690 //
691 // ============================================================================
692
693 void Initialize(void)
694 {
695     // Turn on LED2 indicating entry into Intialize function.
696     mPORTDSetBits(BIT_2);
697
698     // Initialize PORTA.
699     LATA  = 0x0000; // Clear all LATA bits
700     TRISA = 0xFFCF; // Set Pins 4&5 as output, all others input.
701
702     // Ports B&C not used in this program.
703
704     // Initialize PORTD.
705     LATD  = 0x0000; // Clear all LATD bits
706     TRISD = 0xFFC0; // Set Pins 0-5 as output, all others input.
707
708     // Initialize PORTE.
709     LATE  = 0x0000; // Clear all LATE bits
710     TRISE = 0x0FF0; // Set Pins 0-4 as output, all others input/don't care.
```

```
711
712     // Initialize PORTF.
713     LATF  = 0x0000; // Clear all LATF bits
714     TRISF = 0xFFFC; // Set Pins 0&1 as output, all others input.
715
716     // Initialize PORTG.
717     LATG  = 0x0000; // Clear all LATG bits
718     TRISG = 0x0FF0; // Set Pins 0-3,12-15 as output, all others input.
719
720     // Configure the device for maximum performance but do not change the PBDIV
721     // Given the options, this function will change the flash wait states, RAM
722     // wait state and enable prefetch cache but will not change the PBDIV.
723     // The PBDIV value is already set via the pragma FPBDIV option above..
724     pbClk=SYSTEMConfig(SYS_FREQ, SYS_CFG_WAIT_STATES | SYS_CFG_PCACHE);
725
726     // Configure Timer 1 using internal clock, 1:256 prescale
727     OpenTimer1(T1_ON | T1_SOURCE_INT | T1_PS_1_256, T1_TICK);
728
729     // set up the Timer 1 interrupt with a priority of 2
730     ConfigIntTimer1(T1_INT_ON | T1_INT_PRIOR_2);
731
732     // Configure Timer 5 using internal clock, 1:256 prescale
733     OpenTimer5(T5_ON | T5_SOURCE_INT | T5_PS_1_256, T5_TICK);
734
735     // set up the Timer 5 interrupt with a priority of 2
736     ConfigIntTimer5(T5_INT_ON | T5_INT_PRIOR_7);
737
738     // Initialize the ADC module for channels 2-9.
739     InitADC(0xFC03);
740
741     // UART1 Configuration.
742     // This initialization assumes 36MHz Fpb clock. If it changes,
743     // you will have to modify baud rate initializer.
744     OpenUART1(UART_EN,      // Module is ON
745               UART_RX_ENABLE | UART_TX_ENABLE,      // Enable TX & RX
746               pbClk/16/DESIRED_BAUDRATE-1); // 9600 bps, 8-N-1
747
748     // UART2 Configuration.
749     // This initialization assumes 36MHz Fpb clock. If it changes,
750     // you will have to modify baud rate initializer.
751     OpenUART2(UART_EN,      // Module is ON
752               UART_RX_ENABLE | UART_TX_ENABLE,      // Enable TX & RX
753               pbClk/16/DESIRED_BAUDRATE-1); // 9600 bps, 8-N-1
754
755     // Configure UART1 RX Interrupt
756     ConfigIntUART1(UART_INT_PR4 | UART_RX_INT_EN);
757
758     // Configure UART2 RX Interrupt
759     ConfigIntUART2(UART_INT_PR2 | UART_RX_INT_EN);
760
761     // Must enable glocal interrupts - in this case, we are using multi-vector mode
762     INTEnableSystemMultiVectoredInt();
763
764     // Output startup message on UART1.
765     U1CRLF();
766     U1CRLF();
767     putsUART1(titleString1);
768     U1CRLF();
769     putsUART1(titleString2);
770     U1CRLF();
771     putsUART1(titleString3);
772     U1CRLF();
773     putsUART1(titleString4);
774     U1CRLF();
775     putsUART1(titleString5);
776     U1CRLF();
777     U1CRLF();
778
779     // Output startup message on UART2.
780     U2CRLF();
781     U2CRLF();
```

```
782      putsUART2(titleString1);
783      U2CRLF();
784      putsUART2(titleString2);
785      U2CRLF();
786      putsUART2(titleString3);
787      U2CRLF();
788      putsUART2(titleString4);
789      U2CRLF();
790      putsUART2(titleString5);
791      U2CRLF();
792      U2CRLF();
793
794      // wait for the first conversion to complete
795      // so there will be vaild data in ADC result registers.
796      while ( ! mAD1GetIntFlag() );// { }
797
798      // PWM example code uses OC1 module
799      // For PWM w FAULT disabled, 50% DC, f=52.08KHz @fPB=40MHz
800      // TMR2 as PWM Timebase and OC1 Interrupt enabled
801
802      // Configure PWM prameters.
803      OC1CON  =   0x0000;         // Turn off OC1 during setup.
804      OC1R    =   0x00600000;     // Initiate primary Compare register.
805      OC1RS   =   0x00007000;     // Initiate secondary Compare register.
806      OC1CON  =   0x0006;         // Configure for PWM mode, FAULT pin disabled.
807      PR2     =   0x00600000;     // Load period register.
808
809      // Configure PWM Interrupt.
810      IFS0    &=  ~0x00000080;    // Clear the OC1 Interrupt flag.
811      IEC0    |=  0x00000080;     // Enable OC1 Interrupt.
812      IFS0    |=  0x001C0000;     // Set OC1 Interrupt priority = 7 (highest).
813      IPC1    |=  0x00000003;     // Set su priority = 3 (highest).
814
815      // Set timer and OC to run
816      T2CON   |=  0x8000;         // Enable TMR2.
817      OC1CON  |=  0x8000;         // Enable OC1.
818
819      for(i=0; i<1600000; i++);   // perform a delay.
820      mPORTDClearBits(BIT_2);         // Turn off LED2 after startup delay.
821
822  } // END function Initialize()
823
824  // ============================================================================
825  //
826  //                          End: Initialization section
827  //
828  // ============================================================================
829
830  // ============================================================================
831  //
832  //                          Begin: main Loop section
833  //
834  // ============================================================================
835
836  int main(void)
837  {
838
839      // Perform the Initialization function before entering the main loop.
840      Initialize();
841
842      // Let interrupt handler do the work
843      while (1)
844      {
845  //      //LED chase demo for Starter Kit board...
846  //      for(i=0; i<1600000; i++);   // perform a delay.
847  //      mPORTDToggleBits(BIT_2);    // toggele LED2.
848  //      mPORTDToggleBits(BIT_0);    // toggele LED0.
849  //      for(i=0; i<1600000; i++);   // perform a delay.
850  //      mPORTDToggleBits(BIT_1);    // toggele LED1.
851  //      mPORTDToggleBits(BIT_0);    // toggele LED0.
852  //      for(i=0; i<1600000; i++);   // perform a delay.
```

```
853  //      mPORTDToggleBits(BIT_2);    // toggele LED2.
854  //      mPORTDToggleBits(BIT_1);    // toggele LED1.
855
856  //      // Output debug pattern on UART1.
857  //      WriteUART1(0xA5);
858
859          // Service the UART 1 RX buffer on a low priority basis.
860          // only when processInput flag is set
861          switch  (processInput)
862              {
863                  case 1:
864                          ProcessInput();
865                          processInput = 0;
866                          break;
867                  case 10:
868                          InitRX2();
869                          processInput = 0;
870                          break;
871                  default:
872                          break;
873              }
874          // GoADC flag is set by the 1 second interval Timer 1 ISR.
875          if (GoADC) GetAllADC();
876
877          DigIn = ReadDigitalInputs();
878
879      };
880      return 0;
881  } // END function main(). End of program.
882
883  // ===============================================================================
884  //
885  //                          End: main Loop section
886  //
887  // ===============================================================================
888
889  // ===============================================================================
890  //
891  //                          END: (REMS_Template_011109.c).
892  //
893  // ===============================================================================
894
895
```