TB010

Adding a Simple 4-channel 8-bit A/D to a PIC17C4X

Author: *Stan D'Souza*
Microchip Technology Inc.

INTRODUCTION

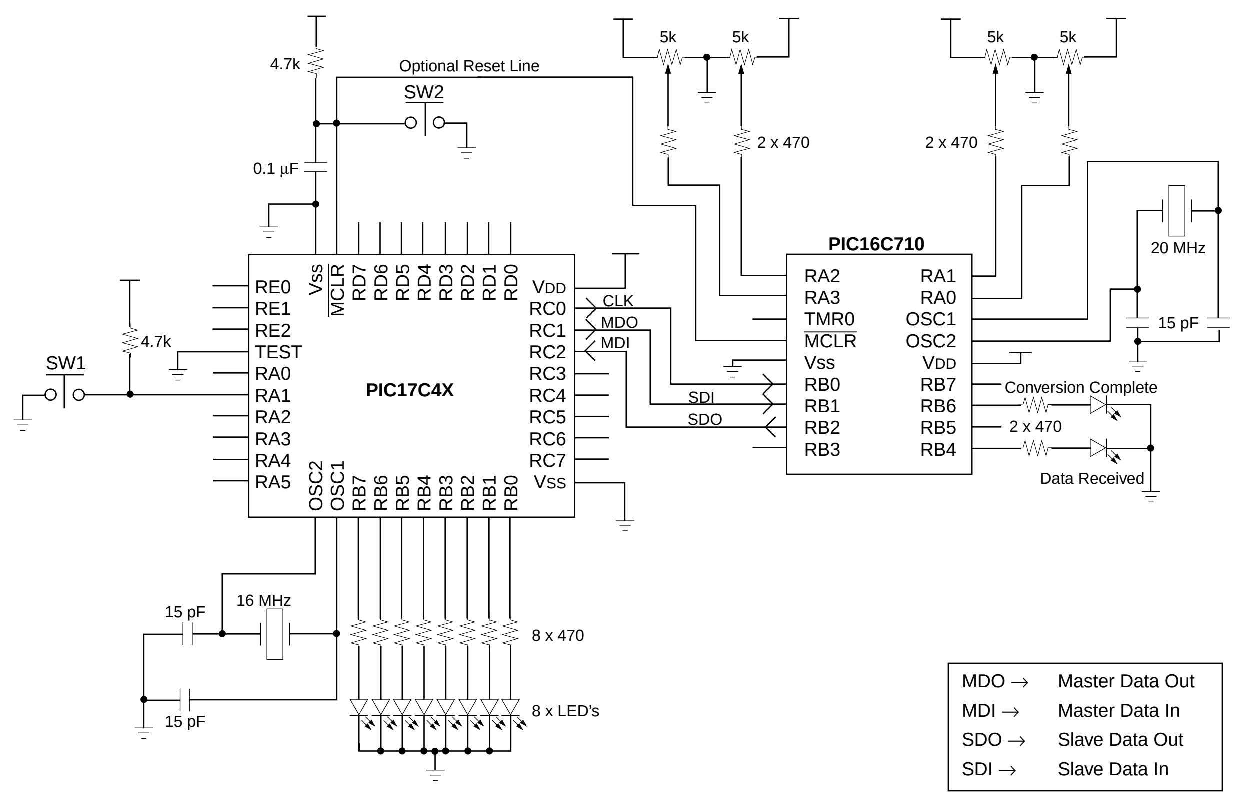
The PIC17C4X high end microcontroller family, offered by Microchip, has many features which make it one of the fastest 8-bit microcontrollers in the industry. However one feature which is missing is an on-chip Analog-to-Digital (A/D) converter. This Technical Brief describes how to connect a PIC16C710 to a PIC17C4X through a simple Serial I/O (SIO) 3 line interface. The result is a quick method by which a user can get a fast (approx. 17 kHz transfer rate) 4-channel A/D converter.

IMPLEMENTATION

The PIC17C4X is the master device in this case, operating at 16 MHz . The master generates the CLK pulse and initiates the conversion. The PIC16C710 is the slave, operating at 20 MHz, and provides the A/D hardware. The two devices are connected using 3 I/O lines, clock, data in, and data out, as shown in Figure 1. The additional connections in Figure 1 are purely for this example and can be omitted if not needed.

The sequence of timed events are depicted in Figure 1. The Master (PIC17C4X) generates the CLK signal and initiates the conversion by transmitting the ADCON register value to the slave (PIC16C710). The slave (PIC16C710) simultaneously transmits ADRES back to the master (PIC17C4X). The ADCON register value has information requesting a new A/D value. The ADRES is the 8-bit result of the last A/D conversion

FIGURE 1: 4-CHANNEL, 8-BIT A/D INTERFACE



MDO →	Master Data Out
MDI →	Master Data In
SDO →	Slave Data Out
SDI →	Slave Data In

requested. One transmission set consists of 8 clock pulses during which the rising edge of each clock pulse is used to transfer data (ADCON register value) from the PIC17C4X to the PIC16C710 and the falling edge of each clock pulse is used to transfer data (ADRES register value) from the PIC16C710 to the PIC17C4X. This method results in a quick transfer of data back and forth from the two processors. This method of transfer also implies that when the master is requesting for a new A/D value from the slave, it is receiving the A/D result of the last request. The two 8-bit register values (ADCON and ADRES) are maintained in the PIC17C4X and the PIC16C710. The PIC17C4X maintains ADCON and ADRES in general purpose RAM. The PIC16C710 maintains the ADCON in general purpose RAM and the ADRES as a special function register. As seen by its name the ADRES is the register which contains the result of the A/D conversion. ADCON register on the other hand is a merger of various bits from the ADCON0 and ADCON1 special function register defined in the PIC16C710. The definitions for ADCON bits are as below:

ADCON bit #	PIC16C710 definitions	PIC16C710 bit assignments
0	PCFG0	ADCON1<0>
1	PCFG1	ADCON1<1>
2	GO/$\overline{\text{DONE}}$	ADCON0<2>
3	CHS0	ADCON0<3>
4	CHS1	ADCON0<4>
5	Not used	Not Used
6	ADCS0	ADCON0<6>
7	ADCS1	ADCON0<7>

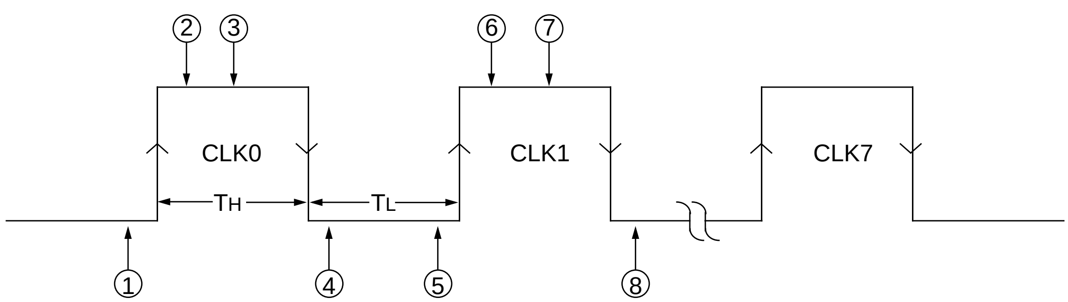
With this setup the user has full control and access to all the A/D configuration features of the PIC16C710. In the example code in Appendix A and Appendix B, the PORTA pins have been all selected as digital I/O and the A/D conversion clock is the internal RC oscillator.

A key feature of this arrangement is the simplicity of the interface. To minimize the number of I/O lines used, the master can reset the slave without having to use the slave's Master Clear ($\overline{\text{MCLR}}$) pin. This can be accomplished by using the PIC16C710's Watchdog Timer (WDT) enabled with a time-out of 18 ms. Once the leading edge of the transfer string is initiated, the master has 8 ms in which to complete the transfer. If the transfer takes longer than the WDT time-out, the WDT will time-out and the PIC16C710 will reset. This feature allows the master to reset the slave, at any time, by initiating a transfer but not completing it before the WDT times out. With a 1:1 prescaler setting, the PIC16C710's WDT will time out between 18 ms and 40 ms.

CONCLUSION

The addition of the PIC16C710 to a PIC17C4X gives the analog input capability to the high-end microcontroller family while only sacrificing 123 words of program memory and 2 bytes of RAM. The simple program in the PIC16C710 uses only 82 words of program memory and 3 bytes of RAM. Since only four pins on the PIC16C710 are used for analog input, the rest of the microcontroller can be used as general purpose I/O if needed.

FIGURE 1: INTERFACE TIMING



1. PIC17C4X sets MDO line = bit0 of ADCON.
2. PIC16C710 gets interrupt and reads bit0 value of ADCON on SDI line.
3. PIC16C710 sets SDO line = bit0 of ADRES.
4. PIC17C4X reads bit0 of ADRES on MDI line.
5. PIC17C4X sets MDO line = bit1 of ADCON.
6. PIC16C710 reads bit1 of ADCON on SDI line.
7. PIC16C710 sets SDO bit = bit1 of ADRES.
8. PIC17C4X reads bit1 of ADRES on MDI line.

Steps 1 through 4 are repeated till all 8-bits are received and sent.

The Clock is generated by the Master.

MDO → Master Data Out
MDI → Master Data In
SDO → Slave Data Out
SDI → Slave Data In
$T_H > 2\ \mu s$ min.
$T_L > 2\ \mu s$ min.

PERFORMANCE

The conversion rate is the addition of the SIO transfer rate as well as acquisition time rate:

The calculation is as follows:

$$[(2 + 2) \bullet 8 + 25]\ \mu s = 57\ \mu s \Rightarrow 17.5\ kHz.$$

FIGURE 2: FLOWCHART FOR MASTER (PIC17C4X)

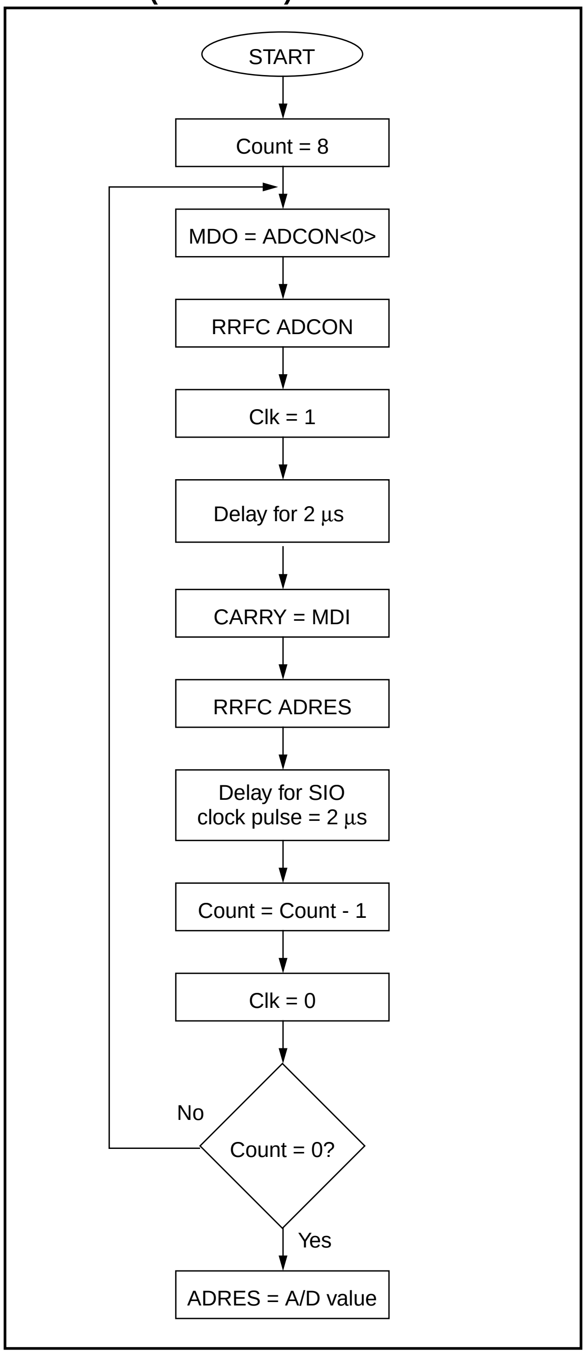
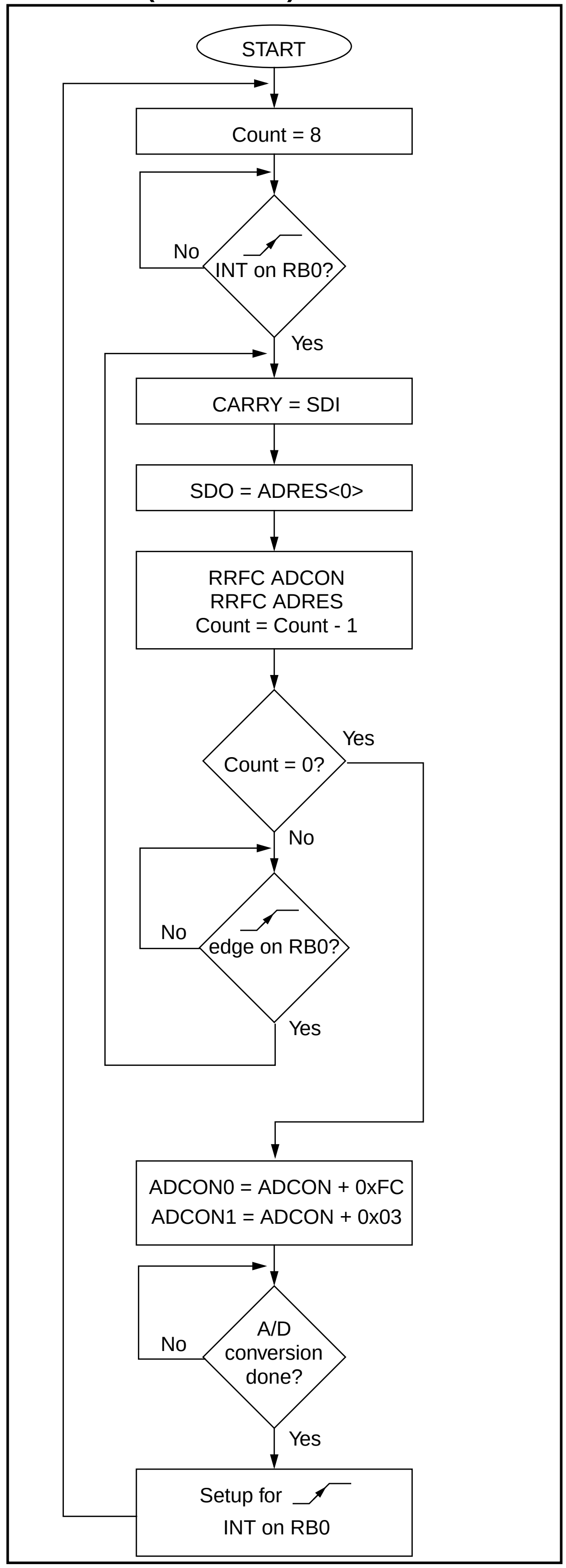


FIGURE 3: FLOWCHART FOR SLAVE (PIC16C710)



APPENDIX A: CODE FOR PIC17C4X (MASTER)

```
/* This program implements the master control of the ADC interface
   with a PIC16C710 as a simple serial ADC device for use with the
   PIC17C4X.
   The 3 wire interface is as follows:
    CLK --> RC0 output
    DO  --> RC1 output
    DI  --> RC2 input
   The PIC17C4X will transmit an 8-bit byte to the PIC16C710
   in order for it to configure registers ADCON1 and ADCON0 as follows:
    Bit 0 --> PCFG0
    Bit 1 --> PCFG1
    BIT 2 --> Start
    BIT 3 --> CHS0
    BIT 4 --> CHS1
    BIT 5 --> not used
    BIT 6 --> ADCS0
    BIT 7 --> ADCS1
   The ADCON0 and ADCON1 registers are merged into one register
   to help reduce the interface overhead. After all the 8 bits are
   received, bits 0 and 1 are placed in ADCON1 and bits 3 to 7
   are placed in ADCON0.  Depending on the state of bit 2,
   the A/D conversion is then started.

   The protocol basics are as follows:
    Two registers are maintained in the PIC17C4X which are
    used in the transfer:
    ADCON --> ADCON1 merged with ADCON0
    ADRES --> same as the AD result register
    8 clock signals are used to transmit the ADRES value
    and receive the next ADCON value. On the rising edge of
    the clock the serial ADCON value is sent whilst on the
    falling edge of the same clock the serial ADRES value
    is read.  In this manner an efficient transfer
    rate is maintained with very little overhead to software
    and more importantly rate of A/D conversion.
    The PIC16C710 also maintains a replica of ADCON and ADRES
    and does just the opposite of what the PIC17C4X does i.e
    ADCON is received and ADRES is transmitted.  The clock
    master is always the PIC17C4X, the PIC16C710 is always the
    slave.  The transfer occurs as below:
    Clock   Data IN(DI) Data OUT(DO)
    ----------------------------------------------------
    Rising edge    N/A       RRF ADCON; C --> DO
    Falling edge   read DI     N/A

    This is repeated till all 8 bits are transmitted/received.

    On receiving the 8-bit value, the ADCON1 and ADCON0 registers
    configured and the A/D conversion started.  The result of the
    conversion is then stored in the ADRES register.  The next
    transfer then sends the ADRES result out.

    NOTE: The PIC17C4X has to provide enough time for sampling
          and for the conversion to complete.

    The WDT on the PIC16C710 is enabled and used in the following manner:

    The entire 8 clock and A/D conversion should be done in a
    max of 8 mS, which is the min. timeout period for the WDT.
    If this is not done the PIC16C710 will reset and wait for
    another 8 clocks. In this manner the master can reset the
    slave by toggling the clock once and waiting for 40 mS,
    which is the max. time for WDT with no postscaler.
```



```
        OSC = 16Mhz; XT mode; WDT disabled; Microcontroller Mode;
                                                              */

                                  #include <c:\mplabc\17C44.h>
                                  #ifndef 17C44_H
                                  /*
                      PIC17C44 Standard Header File, Version 1.00
                      (c) Copyright 1996 Microchip Technology, Inc., Byte Craft Limited

                                  RAM locations reserved for temporary variables: 0x1A - 0x1E
                                  */

                                  #pragma option +l;
                                  #endif

                                  #include <c:\mplabc\delay16.h>
                                  #pragma option +l;

00C7                              #define AdconDefault 0xC7 //
0000                              #define CLK      0        // PORTC bit 0 is the CLK
0001                              #define DO       1        // bit 1 is Data Out
0002                              #define DI       2        // bit 2 is data in
001F                              bits    ADCON;
0003                              #define Chs0     3
0004                              #define Chs1     4
0020                              char    Count;
0021                              char    Temp;
0022                              char    ADRES;
0023                              bits    Channel;        // tracks which channel is selected

0002                              #define SIOClkPulse   2     // CLK pulse in uS
0028                              #define MaxWdtTime    40    // WDT timeout in mS
0019                              #define AdAcqTime     25    // A/D Acquisition time in uS
0064                              #define DebounceTime  100   // Debounce time in mS
                                  void InitADC(void)
                                  {
0028 B800    MOVLB  00h               PORTB = 0;
0029 2912    CLRF   PORTB
002A B801    MOVLB  01h               PORTC = 0;
002B 2911    CLRF   PORTC
002C B800    MOVLB  00h               DDRB = 0;          // set PORTB as outputs
002D 2911    CLRF   DDRB
002E B0FC    MOVLW  FCh               DDRC = 0xFC;       // set PORTC for SPI
002F B801    MOVLB  01h
0030 0110    MOVWF  DDRC
0031 2923    CLRF   23                Channel = 0;
0032 B0C7    MOVLW  C7h               ADCON = AdconDefault;
0033 011F    MOVWF  1F
0034 0002    RETURN               }

                      /* The GetADC routine transmits the information on the channel to the
                         PIC16C710 while receiving the result of the last conversion.
                         The channel is selected by the value in Channel which goes from
                         0 to 3.                                                          */

                                  char GetADC(void)
                                  {
0035 B0C7    MOVLW  C7h               ADCON = AdconDefault;  // set ADCON to default
0036 011F    MOVWF  1F
0037 9023    BTFSS  23,0              if (Channel.0)         // select channel
0038 C03A    GOTO   003Ah
0039 831F    BSF    1F,3                 ADCON.Chs0 = 1;
003A 9123    BTFSS  23,1              if (Channel.1)
003B C03D    GOTO   003Dh
003C 841F    BSF    1F,4                 ADCON.Chs1 = 1;
```

```
003D 2921    CLRF   21                 Temp = 0;
003E 8804    BCF    ALUSTA,0           ALUSTA.C = 0;
003F 2920    CLRF   20                 for(Count = 0;Count < 8;Count++)
0040 B008    MOVLW  08h
0041 0420    SUBWF  20,W
0042 9804    BTFSC  ALUSTA,0
0043 C05E    GOTO   005Eh
                                          {     // send 8 bit data
0044 191F    RRCF   1F                    RRCF(ADCON);
0045 9004    BTFSS  ALUSTA,0              if (ALUSTA.C)
0046 C04A    GOTO   004Ah
0047 B8F1    MOVLB  F1h                     PORTC.DO = 1;
0048 8111    BSF    PORTC,1
0049 C04C    GOTO   004Ch                 else PORTC.DO = 0;
004A B8F1    MOVLB  F1h
004B 8911    BCF    PORTC,1
004C B8F1    MOVLB  F1h                   PORTC.CLK = 1;
004D 8011    BSF    PORTC,0
004E B002    MOVLW  02h                   Delay_Us_16MHz(SIOClkPulse);
004F E09C    CALL   009Ch
0050 B8F1    MOVLB  F1h                   if (PORTC.DI)
0051 9211    BTFSS  PORTC,2
0052 C055    GOTO   0055h
0053 8004    BSF    ALUSTA,0                ALUSTA.C = 1;
0054 C056    GOTO   0056h                 else ALUSTA.C = 0;
0055 8804    BCF    ALUSTA,0
0056 1921    RRCF   21                    RRCF(Temp);
0057 B8F1    MOVLB  F1h                   PORTC.CLK = 0;
0058 8811    BCF    PORTC,0
0059 B002    MOVLW  02h                   Delay_Us_16MHz(SIOClkPulse);
005A E09C    CALL   009Ch
005B 8804    BCF    ALUSTA,0              ALUSTA.C = 0;
                                          }
005C 1520    INCF   20
005D C040    GOTO   0040h
005E 6A21    MOVFP  21,WREG            return(Temp);
005F 0002    RETURN
                                   }

                  /* The PIC16C710 has 4 channels each of which can be selected
                     by simply pressing the switch connected to RA1. This increments
                     the value of the Channel register from 0 to 3 and then back to 0.
                     The 100 mS delay is meant as a debounce delay.                 */

                                   void SampleRA1Key(void)
                                   {
0060 B800    MOVLB  00h                if(!PORTA.1)
0061 9910    BTFSC  PORTA,1
0062 C069    GOTO   0069h
0063                                      {
0063 B064    MOVLW  64h                   Delay_Ms_16MHz(DebounceTime);
0064 E086    CALL   0086h
0065 B800    MOVLB  00h                   if(!PORTA.1)
0066 9910    BTFSC  PORTA,1
0067 C069    GOTO   0069h
0068 1523    INCF   23                      Channel++;
                                          }
0069 B003    MOVLW  03h                if (Channel > 3)
006A 0423    SUBWF  23,W
006B 9204    BTFSS  ALUSTA,2
006C 9004    BTFSS  ALUSTA,0
006D C06F    GOTO   006Fh
006E 2923    CLRF   23                    Channel = 0;
006F 0002    RETURN                }
```

```
                 /* The entire SIO transfer from the PIC17C4X and the PIC16C710
                    should be done in less time than the max. WDT timeout time. If
                    in any case this does not occur, the PIC16C710 will experience a
                    WDT reset.  This "feature" can also be used to asyn. reset the
                    slave PIC16C710 as shown below.                                */

                                  void ResetADC(void)
                                  {
0070 B801    MOVLB  01h               PORTC.CLK = 1;
0071 8011    BSF    PORTC,0
0072 B002    MOVLW  02h               Delay_Us_16MHz(SIOClkPulse);
0073 E09C    CALL   009Ch
0074 B801    MOVLB  01h               PORTC.CLK = 0;
0075 8811    BCF    PORTC,0
0076 B028    MOVLW  28h               Delay_Ms_16MHz(MaxWdtTime);
0077 E086    CALL   0086h
0078 0002    RETURN               }

                /* On startup, the PIC17C4X resets the PIC16C710 by using the WDT
                   feature mentioned above.  It then requests an A/D value.
                   The A/D value of the channel selected by the "Channel" register.
                   It is then sampled by the PIC16C710 and the result is sent back
                   to the PIC17C4X and saved in ADRES.  The 8-bit value is displayed
                   on 8 LEDs connected to PORTB.  To select another channel, simply
                   press the switch connected to RA1 till the correct channel is
                   selected.                                                     */

                                  void main(void)
                                  {

0079 E028    CALL   0028h             InitADC();
007A E070    CALL   0070h             ResetADC();
007B E035    CALL   0035h             Temp = GetADC();
007C 0121    MOVWF  21
                                      while(1)
                                         {
007D B019    MOVLW  19h
007E E09C    CALL   009Ch
007F                                     Delay_Us_16MHz(AdAcqTime);
007F E035    CALL   0035h                ADRES = GetADC();
0080 0122    MOVWF  22
0081 B800    MOVLB  00h                  PORTB = ADRES;
0082 7222    MOVFP  22,PORTB
0083 E060    CALL   0060h                SampleRA1Key();
0084 C07D    GOTO   007Dh                }
0085 0002    RETURN               }
                                  #pragma option +l;
                                  /*
/***************************************************************************/
                                  void Delay_Ms_16MHz(registerw delay)
                                  /*
                                     Clock Speed = 16MHz
                                     Inst. Clock =  4MHz
                                     Inst. dur.  = 0.25us  */
0000                              {
                                  #asm
0086 0000                       NOP
0087 011A                       MOVWF   __WImage
                        DLMS16M1
                                RADIX   DEC
0088 B0F9                       MOVLW   249
0089 0101                       MOVWF   FSR0
                        DLMS16M2
                        DLMS16M__  REPT  13
                                NOP
                        ENDM
```

```
008A 0000
008B 0000
008C 0000
008D 0000
008E 0000
008F 0000
0090 0000
0091 0000
0092 0000
0093 0000
0094 0000
0095 0000
0096 0000
0097 0117                    DECFSZ  FSR0
0098 8AC0                    GOTO    DLMS16M2

0099 1A17                    DECFSZ  __WImage
009A 88C0                    GOTO    DLMS16M1
                                #endasm
009B 0002    RETURN               }
/**************************************************************************/
                                  void Delay_Us_16MHz(registerw delay)
                                  /*
                                     Clock Freq. = 16MHz
                                     Inst. Clock = 4MHz
                                     Inst. dur.  = 250ns  */
0000                              {
                                  #asm
009C 0000                       NOP
009D 011A                       MOVWF   __WImage
                        DLUS16M
009E 0000                       NOP
009F 171A                       DECFSZ  __WImage
00A0 C09E                       GOTO    DLUS16M
                                  #endasm
00A1 0002    RETURN               }
/**************************************************************************/
ROM USAGE MAP

    0000 to 0000    0028 to 00A1
    Total ROM used 007B


Errors            :    0
Warnings          :    0
```


APPENDIX B: CODE FOR PIC16C710 (SLAVE)

```
/* This program creates a serial I/O interface with a PIC17C4X
   device.  The basic intent of this interface is to use the
   PIC16C710 as a simple serial ADC device for use with the PIC17C4X.
   The 3 wire interface is as follows:
    CLK --> RB0/INT
    DI  --> RB1
    DO  --> RB2
   The PIC17C4X device will transmit an 8-bit word to the PIC16C710
   in order for it to configure the following:
    Bit 0 --> PCFG0
    Bit 1 --> PCFG1
    BIT 2 --> Start
    BIT 3 --> CHS0
    BIT 4 --> CHS1
    BIT 5 --> not used
    BIT 6 --> ADCS0
    BIT 7 --> ADCS1
   The ADCON0 and ADCON1 registers are merged into one register
   to help reduce the interface overhead. After all the 8 bits are
   received, bits 0 and 1 are placed in ADCON1 and bits 3 to 7
   are placed in ADCON0.  Depending on the state of the bit 2,
   the A/D conversion is then started.

   The protocol basics are as follows:
    Two registers are maintained in the PIC16C710 which are
    used in the transfer:
    ADCON --> ADCON1 merged with ADCON0
    ADRES --> same as the AD result register
    8 clock signals are used to transmit the ADRES value
    and receive the next ADCON value. On the rising edge of
    the clock the serial ADCON value is read whilst on the
    falling edge of the same clock the serial ADRES value
    is transmitted.  In this manner an efficient transfer
    rate is maintained with very little overhead to software
    and more importantly rate of A/D conversion.
    The PIC17C4X also maintains a replica of ADCON and ADRES
    and does just the opposite of what the PIC16C710 does i.e
    ADRES is received and ADCON is transmitted.  The clock
    master is always the PIC17C4X, the PIC16C710 is always the
    slave.  The transfer occurs as below:
    Clock   Data IN(DI) Data OUT(DO)
    ----------------------------------------------------
    Falling edge   N/A     RRF ADRES; C --> DO
    Rising edge    read DI       N/A

    This is repeated till all 8 bits are transmitted/received.

    On receiving the 8-bit value, the ADCON1 and ADCON0 registers
    configured and the A/D conversion started.  The result of the
    conversion is then stored in the ADRES register.  The next
    transfer then sends the ADRES result out.

    NOTE: The PIC17C4X has to provide for enough time for sampling
    and for the conversion to complete.

   The WDT is enabled and used in the following manner:
    The entire 8 clocks and A/D conversion should be done in a
    max. of 8 mS, which is the min. timeout period for the WDT.
    If this is not done the PIC16C710 will reset and wait for
    another 8 clocks. In this manner the master can reset the
    slave by toggling the clock once and waiting for 40 mS,
    which is the max. time for WDT.
```

```
        OSC = 20Mhz; HS mode; WDT enabled; PWRT timer disabled;
        */

                                  #include <c:\mplabc\16c710.h>
                                  #ifndef 16C710_H
                                  /*
                   PIC16C710 Standard Header File, Version 1.00
                   (c) Copyright 1996 Microchip Technology, Inc., Byte Craft Limited

                   RAM locations reserved for temporary variables: 0x0C - 0x10
        */

                                  #pragma option +l;

                                  #endif

                                  #include <c:\mplabc\delay14.h>
                                  #pragma option +l;

0000                              #define CLK 0
0001                              #define DI  1
0002                              #define DO  2
0004                              #define Received 4
0006                              #define Complete 6
0011                              bits    ADCON;
0012                              char    Count;
0013                              bits    Flag;
0000                              #define NewCommand  0

0004                              #pragma vector Interrupt @ 0x0004;

         /* The whole process is driven by the completion of the interrupt routine
            and once the 1st clock is received, the interrupts are disabled. Because of this
            there is no reason to save the W and STATUS registers during interrupts */

0004 2805    GOTO   0005h         void Interrupt(void)
0005                              {
0005 1283    BCF    03,5              PORTB.Complete = 0;
0006 1306    BCF    06,6
0007 108B    BCF    0B,1              INTCON.INTF = 0;     // clr flag
                                      do          // first int is on rising edge so read data
                                         {
0008 1283    BCF    03,5                 while(!PORTB.CLK); //make sure rising edge is present
0009 1C06    BTFSS  06,0
000A 2808    GOTO   0008h
000B 1283    BCF    03,5                 if (PORTB.DI)        // Read Data in
000C 1C86    BTFSS  06,1
000D 2810    GOTO   0010h

000E 1403    BSF    03,0                   STATUS.C = 1;
000F 2811    GOTO   0011h                else STATUS.C = 0;
0010 1003    BCF    03,0
0011 0C91    RRF    11                   RRCF(ADCON);   // get data in ADCON
0012 0C89    RRF    09                   RRCF(ADRES);   // prepare data out from ADRES
0013 1C03    BTFSS  03,0                 if (STATUS.C)
0014 2817    GOTO   0017h

0015 1506    BSF    06,2                   PORTB.DO = 1;    //set data out
0016 2818    GOTO   0018h                else PORTB.DO = 0;
0017 1106    BCF    06,2
0018 1283    BCF    03,5                 while(PORTB.CLK); // wait for falling edge
0019 1806    BTFSC  06,0
001A 2818    GOTO   0018h

001B 1283    BCF    03,5                 Count++;   // increment count
001C 0A92    INCF   12
001D 3008    MOVLW  08h                  }while(Count < 8);
001E 0212    SUBWF  12,W
001F 1C03    BTFSS  03,0
```



```
0020 2808    GOTO   0008h
0021 1283    BCF    03,5             Flag.NewCommand = 1;      //set flag
0022 1413    BSF    13,0
0023 120B    BCF    0B,4             INTCON.INTE = 0;
0024 1606    BSF    06,4             PORTB.Received = 1;       // set "Received" LED
0025 0009    RETFIE              }

                                 void Convert(void)
                                 {
0026 1283    BCF    03,5             Flag.NewCommand = 0;
0027 1013    BCF    13,0
0028 0192    CLRF   12               Count = 0;
0029 3003    MOVLW  03h              ADCON1 = ADCON & 0x03;
002A 0511    ANDWF  11,W
002B 1683    BSF    03,5
002C 0088    MOVWF  08
002D 30F8    MOVLW  F8h              ADCON0 = ADCON & 0xF8;
002E 1283    BCF    03,5
002F 0511    ANDWF  11,W
0030 0088    MOVWF  08
0031 1408    BSF    08,0             ADCON0.ADON = 1;
0032 1D11    BTFSS  11,2             if(ADCON.GO)
0033 2838    GOTO   0038h
0034                                    {
0034 1508    BSF    08,2                ADCON0.GO = 1;
0035 1283    BCF    03,5                while (ADCON0.ADIF == 1); // wait till conv. is done
0036 1888    BTFSC  08,1
0037 2835    GOTO   0035h
                                        }
0038 160B    BSF    0B,4             INTCON.INTE = 1;            // enable interrupts on edge
0039 1283    BCF    03,5             PORTB.Complete = 1;         // set completion LED
003A 1706    BSF    06,6
003B 1206    BCF    06,4             PORTB.Received = 0;
003C 0008    RETURN              }

                                 void main(void)
                                 {
003D 1283    BCF    03,5             PORTB = 0x00;
003E 0186    CLRF   06
003F 3003    MOVLW  03h              TRISB = 0x03;
0040 1683    BSF    03,5
0041 0086    MOVWF  06
0042 1283    BCF    03,5             Count = 0;
0043 0192    CLRF   12
0044 30F8    MOVLW  F8h              OPTION = OPTION & 0xF8;       // WDT = 18 mS
0045 1683    BSF    03,5
0046 0581    ANDWF  01
0047 1701    BSF    01,6             OPTION.INTEDG = 1;            // int on rising edge
0048 018B    CLRF   0B               INTCON = 0;
0049 160B    BSF    0B,4             INTCON.INTE = 1;              // enable interrupt
004A 178B    BSF    0B,7             INTCON.GIE = 1;
004B 1283    BCF    03,5             Flag.NewCommand = 0;
004C 1013    BCF    13,0
                                     while (1)
                                        {
004D 1283    BCF    03,5                if(Flag.NewCommand)
004E 1813    BTFSC  13,0
004F 2026    CALL   0026h                 Convert();
0050 0064    CLRWDT                     CLRWDT();
0051 284D    GOTO   004Dh               }
0052 0008    RETURN              }

0000 3000    MOVLW  00h
0001 008A    MOVWF  0A
0002 283D    GOTO   003Dh
```

```
ROM USAGE MAP

    0000 to 0002    0004 to 0052
    Total ROM used 0052


Errors             :    0
Warnings           :    0
```


WORLDWIDE SALES & SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602-786-7200 Fax: 602-786-7277
Technical Support: 602 786-7627
Web: http://www.microchip.com

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508-480-9990 Fax: 508-480-8575

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
Microchip Technology Inc.
14651 Dallas Parkway, Suite 816
Dallas, TX 75240-8809
Tel: 972-991-7177 Fax: 972-991-8588

Dayton
Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 714-263-1888 Fax: 714-263-1338

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 416
Hauppauge, NY 11788
Tel: 516-273-5305 Fax: 516-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

Hong Kong
Microchip Asia Pacific
RM 3801B, Tower Two
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

India
Microchip Technology India
No. 6, Legacy, Convent Road
Bangalore 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Shanghai
Microchip Technology
RM 406 Shanghai Golden Bridge Bldg.
2077 Yan'an Road West, Hongiao District
Shanghai, PRC 200335
Tel: 86-21-6275-5700
Fax: 86 21-6275-5060

Singapore
Microchip Technology Taiwan
Singapore Branch
200 Middle Road
#10-03 Prime Centre
Singapore 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan, R.O.C
Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886 2-717-7175 Fax: 886-2-545-0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
Unit 6, The Courtyard
Meadow Bank, Furlong Road
Bourne End, Buckinghamshire SL8 5AJ
Tel: 44-1628-851077 Fax: 44-1628-850259

France
Arizona Microchip Technology SARL
Zone Industrielle de la Bonde
2 Rue du Buisson aux Fraises
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 Müchen, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleone
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-39-6899939 Fax: 39-39-6899883

JAPAN

Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shin Yokohama
Kohoku-Ku, Yokohama
Kanagawa 222 Japan
Tel: 81-4-5471- 6166 Fax: 81-4-5471-6122

5/8/97

MICROCHIP