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 ADRES register 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
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:

<table>
<thead>
<tr>
<th>ADCON bit #</th>
<th>PIC16C710 definitions</th>
<th>PIC16C710 bit assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PCFG0</td>
<td>ADCON1&lt;0&gt;</td>
</tr>
<tr>
<td>1</td>
<td>PCFG1</td>
<td>ADCON1&lt;1&gt;</td>
</tr>
<tr>
<td>2</td>
<td>GO/DONE</td>
<td>ADCON0&lt;2&gt;</td>
</tr>
<tr>
<td>3</td>
<td>CHS0</td>
<td>ADCON0&lt;3&gt;</td>
</tr>
<tr>
<td>4</td>
<td>CHS1</td>
<td>ADCON0&lt;4&gt;</td>
</tr>
<tr>
<td>5</td>
<td>Not used</td>
<td>Not Used</td>
</tr>
<tr>
<td>6</td>
<td>ADCS0</td>
<td>ADCON0&lt;6&gt;</td>
</tr>
<tr>
<td>7</td>
<td>ADCS1</td>
<td>ADCON0&lt;7&gt;</td>
</tr>
</tbody>
</table>

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 (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.
PERFORMANCE

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

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

FIGURE 2: FLOWCHART FOR MASTER (PIC17C4X)

![Flowchart for Master](image)

FIGURE 3: FLOWCHART FOR SLAVE (PIC16C710)

![Flowchart for Slave](image)
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 ADCON0 and bits 3 to 7 are placed in ADCON1. 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;
#define AdconDefault 0xC7 //
#define CLK 0 // PORTC bit 0 is the CLK
#define DO 1 // bit 1 is Data Out
#define DI 2 // bit 2 is Data in

```
#define Chs0 3
#define Chs1 4

char Count;
char Temp;
char ADRES;
```
bits Channel; // tracks which channel is selected

```
#define SIOClkPulse 2 // CLK pulse in uS
#define MaxWdtTime 40 // WDT timeout in mS
#define AdAcqTime 25 // A/D Acquisition time in uS
#define DebounceTime 100 // Debounce time in mS
```

```
void InitADC(void)
{
    PORTB = 0;
    PORTC = 0;
    DDRB = 0; // set PORTB as outputs
    DDRC = 0xFC; // set PORTC for SPI
    Channel = 0;
    ADCON = AdconDefault;
}
```

```
/* 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)
{
    ADCON = AdconDefault; // set ADCON to default
    if (Channel.0) // select channel
        ADCON.Chs0 = 1;
    if (Channel.1)
        ADCON.Chs1 = 1;
```
TB010

003D 2921   CLR  21         Temp = 0;
003E 8804   BCF  ALUSTA,0   ALUSTA.C = 0;
003F 2920   CLR  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
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
004A B8F1   MOVLB F1h       else PORTC.DO = 0;
004B 8911   BSF PORTC,1
004C B8F1   MOVLB F1h       PORTC.CLK = 1;
004D 8011   BSF PORTC,0
004E B002   MOVLW 02h
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
0055 8804   BCF ALUSTA,0    ALUSTA.C = 0;
0056 1921   RRCF 21         RRCF(Temp);
0057 B8F1   MOVLB F1h       PORTC.CLK = 0;
0058 8811   BCF PORTC,0
0059 B002   MOVLW 02h
005A E09C   CALL 009Ch
005B 8804   BCF ALUSTA,0    ALUSTA.C = 0;
005C 1520   INCF 23         Channel++;
005D 0002   RETURN           return(Temp);  
005E 6A21   MOVFP 21,WREG
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)
{
    if(!PORTA.1)
    {
        Delay_Ms_16MHz(DebounceTime);
        if(!PORTA.1)
        {
            Channel++;
        }
        if (Channel > 3)
    }
void ResetADC(void)
{
    PORTC.CLK = 1;
    BSF PORTC,0
    MOVLW 02h
    CALL 009Ch
    PORTC.CLK = 0;
    BCF PORTC,0
    MOVLW 28h
    CALL 0086h
    RETURN
}

void main(void)
{
    CALL 0028h
    CALL 0070h
    CALL 0035h
    Temp = GetADC();
    MOVWF 21
    while(1)
    {            
    CALL 0079h
    CALL 007Ah
    CALL 007Bh
    MOVLW 19h
    CALL 0035h
    MOVWF 22
    MOVLB 00h
    CALL 0060h
    GOTO 007Dh
    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. */

    #pragma option +l;
    /***************************************************************************/
    void Delay_Ms_16MHz(registerw delay)
    /***************************************************************************/
    {        
        #asm
        0000                              
        0000              NOP
        0001B01A  MOVWF __WImage
        0002   DLMS16M1
        0003    RADIX DEC
        0004    MOVLF 249
        0005    MOVWF FSR0
        0006  DLMS16M2
        0007    DLMS16M__  REPT 13
        0008   NOP
        0009   ENDM

/* 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 async. reset the slave PIC16C710 as shown below. */
void SetADC(void)
{
void Delay_Us_16MHz(registerw delay)
{    
    #asm
    nop
    movwf __WImage
    decfsz __WImage
    goto DLUS16M

    #endasm
    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;

#define CLK 0
#define DI 1
#define DO 2
#define Received 4
#define Complete 6

bits ADCON;
char Count;
bits Flag;

#define NewCommand 0

#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 */

void Interrupt(void)
{
    PORTB.Complete = 0;
    INTCON.INTF = 0;     // clr flag
    while(!PORTB.CLK); // make sure rising edge is present
    if (PORTB.DI)        // Read Data in
        STATUS.C = 1;
    else STATUS.C = 0;
    PORTB.DO = 1;    // set data out
    while(PORTB.CLK); // wait for falling edge
    Count++;   // increment count
    while(Count < 8);
```
void Convert(void)
{
    Flag.NewCommand = 0;
    Count = 0;
    ADCON1 = ADCON & 0x03;
    ADCON0 = ADCON & 0xF8;
    ADCON0.ADON = 1;
    while (ADCON0.ADIF == 1); // wait till conv. is done
    ADCON0.GO = 1;
    while (ADCON0.ADIF == 1); // wait till conv. is done
    ADCON0.GO = 1;
    while (ADCON0.ADIF == 1); // wait till conv. is done
}

void main(void)
{
    PORTB = 0x00;
    TRISB = 0x03;
    OPTION = OPTION & 0xF8; // WDT = 18 mS
    OPTION.INTEDG = 1; // int on rising edge
    INTCON = 0;
    INTCON.INTE = 1; // enable interrupt
    INTCON.GIE = 1;
    Flag.NewCommand = 0;
    while (1)
    {
        if(Flag>NewCommand)
        {
            Convert();
            CLRWDT();
            GOTO 004Dh
        }
        RETURN
    }
}
ROM USAGE MAP

0000 to 0002  0004 to 0052
Total ROM used 0052

Errors : 0
Warnings : 0