Chapter 16. Digital Control using PC with IR

1. Introduction

This chapter is based on a student project, "DigiHouse" by Scotty Mazyck II completed in Spring 2003, which aims to control home appliances using a PC. The essence of "DigiHouse", in PIC 16F877 application point of view, uses two hardware components, one with a 16F877 and other necessary parts and elements, and the other without any microcontroller but with an IR decoder circuit. The one hardware component, IR Master station, contains a 16F877 microcontroller, IR encoder, IR receive, and a serial communication level converter chip RS232 chip. The RS232 chip, when you use a USB based PIC board such as PIC-40-USB from Olimex, is not necessary. The USB control chip in the PIC board handles all serial communication from 16F877. The other component, IR Receiver/Controller, contains an IR receiver and IR decoder circuit. The IR decoder circuit has three LED outputs to simulate the three home appliances this project intends to control. Scotty Mazyck's report on his project DigiHouse contains many interesting aspects: IR transmission, IR encoding, IR reception, and a Visual Basic-based Windows program which, in place of the HyperTerminal, allows a serial communication link between a PC and the IR Master station. This comprehensive work, however, is not properly documented. The project report submitted to me, less than 4 page length, only briefly touches its components and coding.

So I reconstructed his project with much more detailed explanation in words and illustrations. However, I did not change his code except the file register bank changing operations. For example, moving from bank 0 to bank 1 to access `TRISC` register, the original code bothers to set the `RP1` bit of `STATUS` register. This works fine and perfect. However, we can use the MPLAB directive `banksel` to ignore in which bank we are, and to move any bank where the register we try to access is located. With `banksel`, we do not have to frequently look up the file register table to see where a particular register is located.

2. Digital Control using PC - overview

In the Internet age, everybody gets lazy, and our life hinges on network and computer. Now I am very sorry that I provide one more convenience so that you become lazier and more inactive. This example is to control your home appliances like lamps, microwave, heater, or A/C from and using your PC. Of course computer alone cannot do the job. A microcontroller would be just fine to fit in to the case. The term "control" here means a simple on/off control of the appliances.

As we know IR remote control is everywhere and for most of our electronic appliances at home and office. We use IR remote to turn on/off of TV, VCR, CD player, DVD player, etc. But how do we do the same functions using a PC instead of an IR remote? There must be a way to communicate from PC to "IR remote" like device, which can transmit IR information as an IR remote does. This requirement is realized by the IR Master Station. The IR Master station is built around the PIC 16F877 which establishes serial communication with PC and transmits IR data to the IR-ready electronic appliances. In chapter 5, we already discussed about the serial communication, therefore, the communication between PC (using the Hyperterminal in Windows) is not bit a problem. However, if you want to open up your own window in the PC screen with your name or your log, the Hyperterminal cannot be used. Instead, a windows

program based on Visual tool such as Visual Basic is needed. This Visual Basic code is discussed in a separate section.

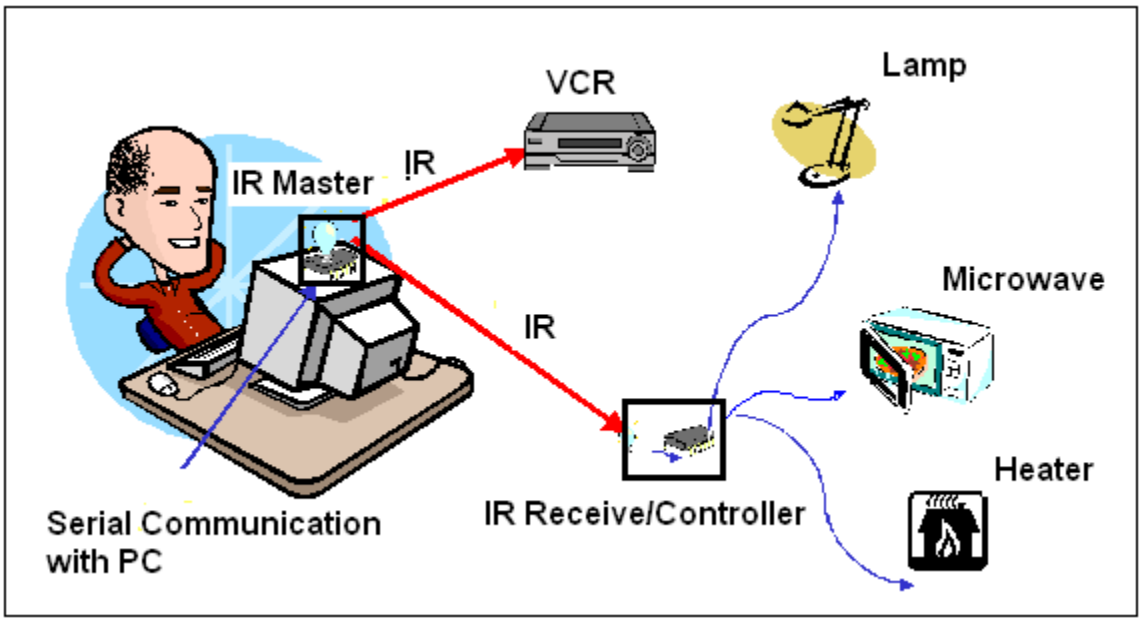

Fig. 101 Control of Home appliances from and using the PC

Then, how do we control those appliances that are not equipped with IR receiver. Most of the home appliances like lamp, microwave, heater, etc are all not IR-ready devices. Here comes the IR Receiver/Controller which is built around an IR receiver and decoder so that a select line could be turned on or off (of course logically). This logical on/off (High or Low, or +5V or 0V) can turn on/off an electronic switch connected to the appliance. In this example, however, we simplified the control part by LED on/off at the IR Receiver/Controller side. In the circuit we installed 3 LEDs, one for each appliance's place, and if an LED is on, for example, the corresponding appliance would be on.

Another function we added in the IR Master is IR learning function. For example, using a real IR remote controller, if you assign the button "2" as the command for the microwave, you aim your remote toward the IR receiver of the IR Master so that it learns the IR command pattern for the appliance. This pattern is stored in the memory, and later, when the microwave control is needed, the stored pattern is used to transmit directly from IR Master (without using any IR remote controller) to the IR Receive/Controller.

3. Hardware Description

IR Master Station: IR Master, implemented on a breadboard as pictured below, consists of 16F877 microcontroller operating at 20MHz, and IR transmitter/receiver. In addition to the essential elements, it has several LEDs as indicators. Also, it has an IR LED to send out pulse IR encoded message generated by an encoding circuit.

Fig. 102 Implementation of IR Master on breadboard

The circuit diagram of the IR Master is shown below.

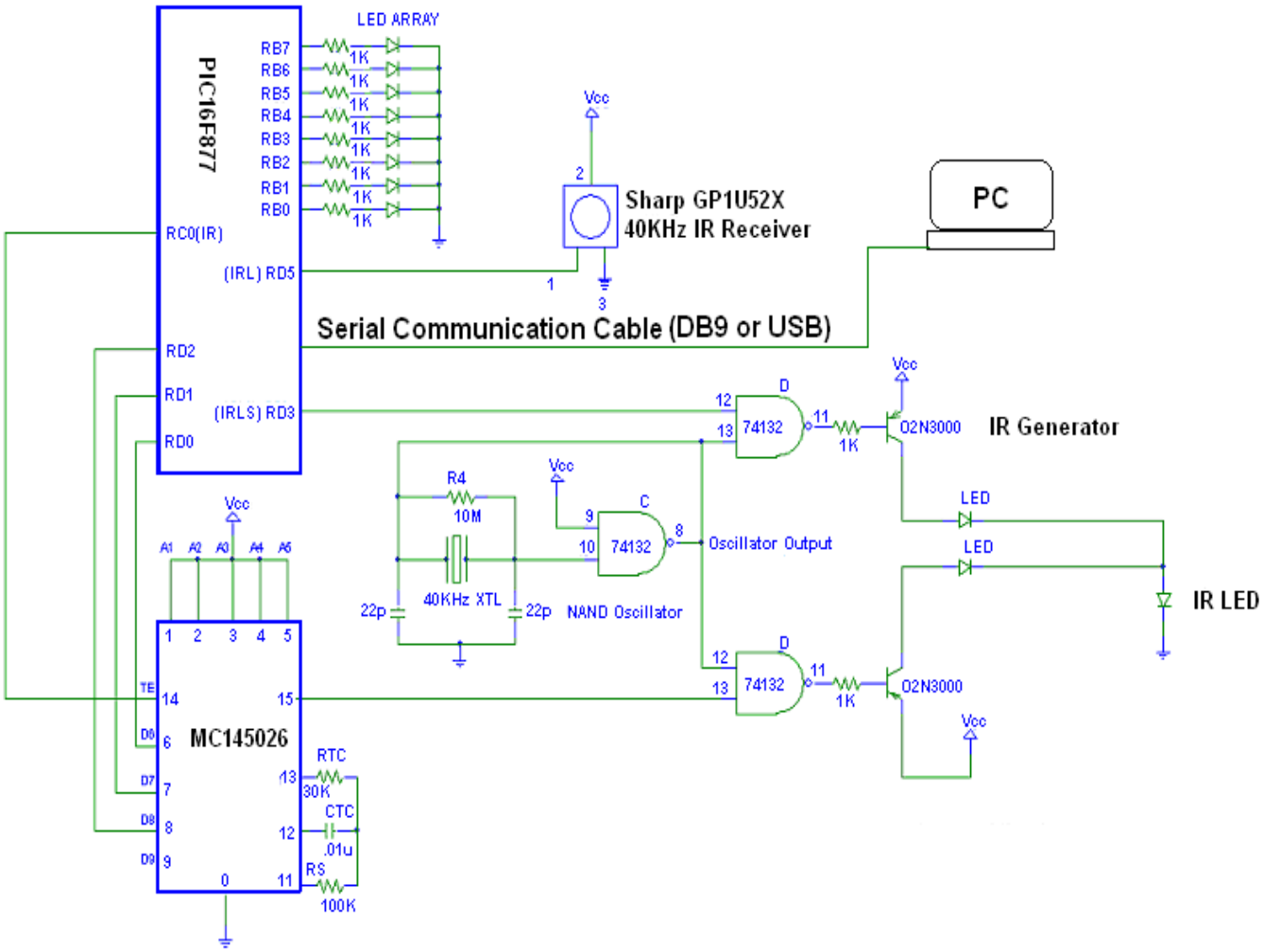

Fig. 103 IR Master circuit diagram

As you see, in addition to the PIC microcontroller and the IR receiver, there is MC145026 Serial Encoder chip. It has a wide bandwidth range which is just low enough for IR communication over a 40KHz carrier. This Serial Encoder is working with MC145027 Decoder chip, which is used in the IR Receiver/Controller board. 40 kHz IR Generator generates a 40 kHz square wave using a 74132 Quad NAND with a Schmitt trigger and a 40.0 KHz crystal. Two LEDs are used as rectifier diodes as well as displays.

MC145026 (Decoder) and MC145027 (Encoder) pair are designed to be used as encoder/decoder pairs in remote control applications.

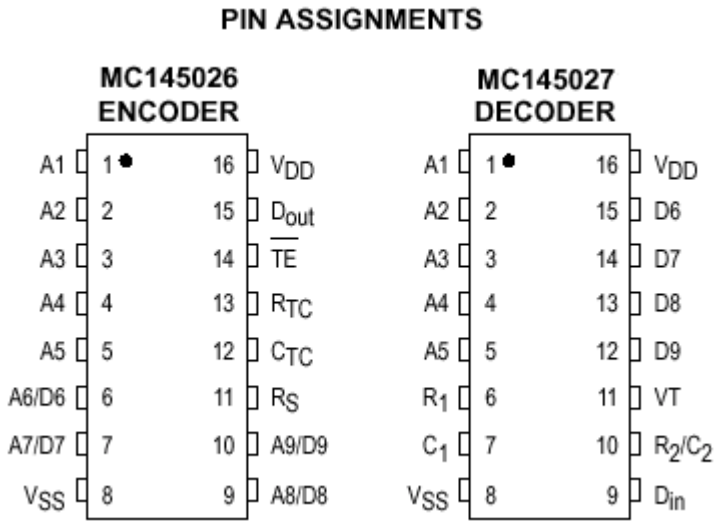

Fig. 104 Pin Assignments for MC145026 and MC145027

The MC145026 encodes nine lines of information and serially sends this information upon receipt of a transmit enable (TE) signal. The nine lines may be encoded with trinary data (low, high, or open) or binary data (low or high). The words are transmitted twice per encoding sequence to increase security.

The MC145027 decoder receives the serial stream and interprets five of the trinary digits as an address code. Thus, 243 addresses are possible. If binary data is used at the encoder, 32 addresses are possible. The remaining serial information is interpreted as four bits of binary data. The valid transmission (VT) output goes high on the MC145027 when two conditions are met. First, two addresses must be consecutively received (in one encoding sequence) which both match the local address. Second, the 4 bits of data must match the last valid data received. The active VT indicates that the information at the Data output pins has been updated.

Details of the application of the encoder/decoder pair can be found in an application note from On Semiconductor.

IR Receiver/Controller: IR Receiver/Controller is to control home appliances by the received command of IR protocol. This board, similarly implemented on a separate breadboard, contains IR receiver and receiver circuit, along with decoder to switch On/Off control of the appliances.

The On/Off control of appliances is indicated by LED On/Off status. The IR Receiver/Controller is pictured below, with 3 colored LEDs representing 3 appliances.

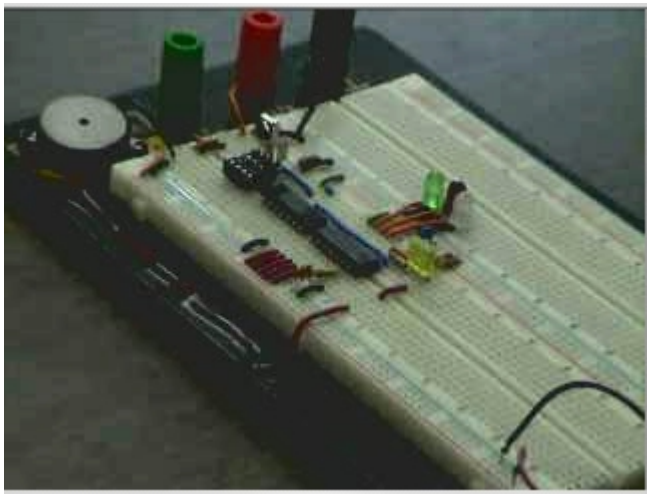

Fig. 105 Implementation of IR Receiver/Controller on breadboard

The schematic of the IR Receiver/Controller is shown below.

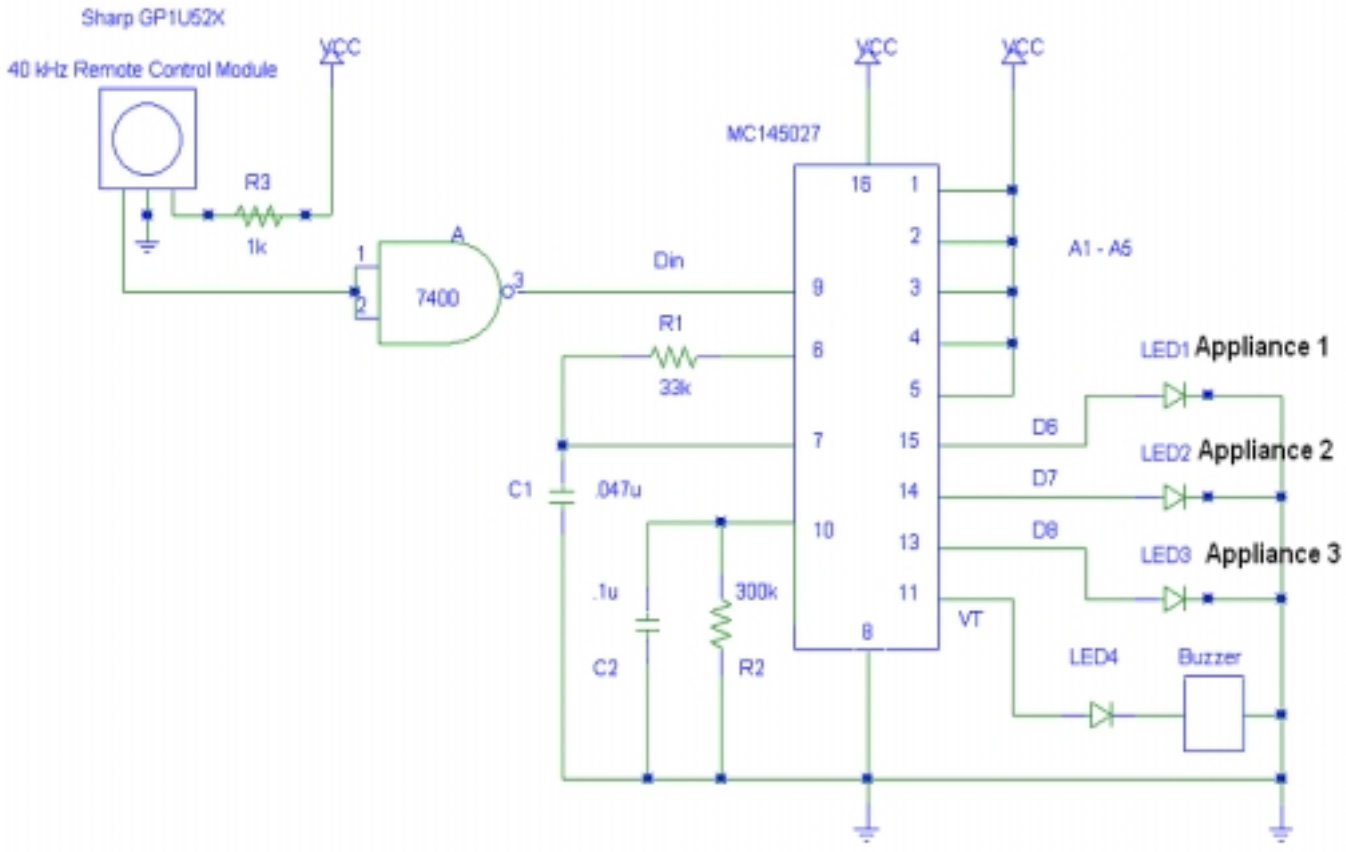

Fig.106 Schematic of IR Receiver/Controller

Parts list is shown below for those who are seriously considering implementing this project.

Part	Quantity	Part	Quantity
Sharp IR 40KHz Receiver	2	Motorola MC145026P	1
PIC16F877 Microcontroller	1	Motorola MC145027P	1
20 MHz Crystal	1	Maxim MAX233ACPE	1
PNP Transistors	2	AlGaAs IR LED	1
Big Bulb LEDs	6	74HC00N Quad NAND	1
Small Bulb LEDs	5	74HC132 Quad NAND	1
Diodes (LED)	2	Resistors, Capacitors	20, 7
40 KHZ Crystal	1	Breadboards	2
5V Power Supply	2		
Buzzer	1		

In addition to the PIC 16F877 assembly language programming for the boards, there is another element of software: a windows programming using Visual Basic to connect a PC to the IR Master station as the controller of the appliances. In all, the overall structure of the system operation is as illustrated below.

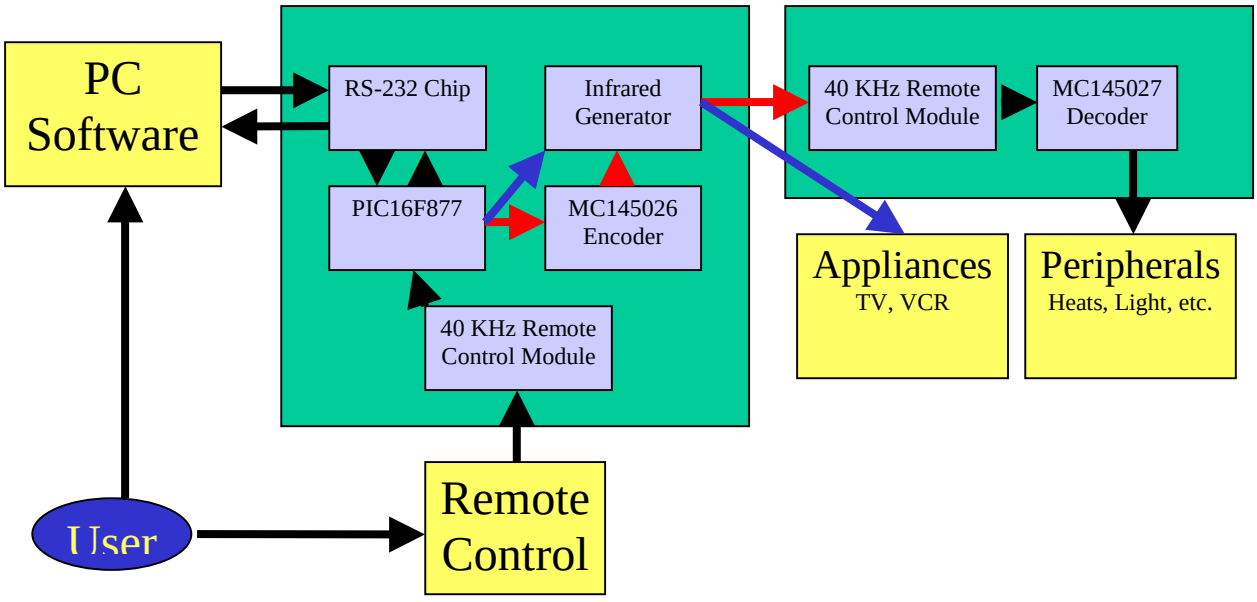

Fig. 107 Overall Structure of System Operation

4. 16F877 Code Segments - General

PORT Set Up for 16F877 in the IR Master Board: As you see in the schematic diagram, 16F877 has numerous outputs: RC0, RD2, RD1, RD0 for MC145026 Encoder, RD5 for IR reception, and RD3 for IR transmission. In addition to these essential connections, 16F877 has additional 8 outputs to LEDs in PORTB. Therefore all PORTB pins are to be assigned as outputs. And except RD5, all PORTD pins are also assigned as outputs. PORTD<5> must be declared as

input, in order to receive IR communication. Since only PORTC<0> is used to initiate transmission of encoded IR, all PORTC pins except RC0 are assigned as inputs. PORTC<0> is designated as output.

```
banksel     TRISB
clrf        TRISB       ;Make PORTB output ports
movlw       0x20        ;Binary = 00100000
movwf       TRISD       ;Make PORTD output ports except the bit 5(IRL)
movlw       0xFE        ;Binary = 11111110
movwf       TRISC       ;Make PORTC input ports except bit 0( SendIR)
banksel     PORTC
bsf         PORTC, IR   ;Turn off IR Transmission (IR=0) or PORTC<0>
clrf        PORTD
bcf         PORTD, IRLS ;IRLS=PORTD<3>
clrf        PORTB       ;Turn off any LEDs
```

Serial Communication Initialization: For the baud rate, we keep our usual 19200 bps. However, instead of low rate we choose to use high rate selection of the value for the SPBRG (see Chapter 5 for details). Anyway, if we briefly review, when TXSTA<2>=0 (low rate selection), the formula for the value of SPBRG, N_{BRG}, is:

$$N_{BRG} = \frac{f_{osc}}{64 \times B_d} - 1$$

where, f_{osc} is crystal oscillation frequency and B is a desired Baud rate.
If TXSTA<2>=1 (high rate selection), the formula for the value of SPBRG, N_{BRG}, is changed to:

$$N_{BRG} = \frac{f_{osc}}{16 \times B_d} - 1$$

Therefore, with high rate selection with TXSTA<2> set, the value for N_{BRG} for 19200 bps for 20MHz crystal oscillation is:

$$N_{BRG} = \frac{f_{osc}}{16 \times 19200} - 1 = \frac{20{,}000{,}000}{16 \times 19200} - 1 = 64.104 \rightarrow 64$$

Therefore the value for SPBRG is 0x40 which is the hexadecimal equivalent value of 64 in decimal.
Other sequences are just routine one we finished in Chapter 5.

```
banksel     SPBRG
movlw       0x40
movwf       SPBRG           ;set baud rate 19200 with high rate
bsf         TXSTA, BRGH     ;Set for High Speed
bcf         TXSTA, SYNC     ;clear for Asynchronous Mode
bcf         TXSTA, TX9      ;Clear for 8-bit
banksel     RCSTA
bsf         RCSTA, SPEN     ;enable serial port
banksel     TXSTA
bsf         TXSTA, TXEN     ;enable transmission
banksel     RCSTA
bsf         RCSTA, CREN     ;Enable Receiver
```

Checking the link between PIC and PC: Checking if there is serial communication link is established between PC and 16F877 is simple. After 16F877 sends an initial code (like 0x09) to the PC via the serial communication link, if 16F877 receives the same code from the PC, then it is considered that communication link between two is established and running fine.

```
PCLoop
      Call        SHDELAY           ;short delay
      movlw       Init              ;a code to be sent to PC
      movwf       OUTCODE           ;buffer for the code INIT
      Call        TRANS             ;Call TRANS Sub to send Init Code to PC


      BTFSS       RCSTA, OERR       ;Check for Rx overrun error.
      goto        ChkIn             ;If none then continue program
      bcf         RCSTA, CREN       ;by clearing CREN and
      bsf         RCSTA, CREN       ;resetting it


ChkIn BTFSS       PIR1, RCIF        ;Check if we received anything from PC
      goto        ContPCL           ;If not then Continue PC Looping
      bsf         PORTB, 6          ;Indicates something has been received
      movf        RCREG, 0          ;If so, then check if its the Init code
      sublw       Init
      BTFSC       STATUS, Z         ;If it is the Init code, then
      goto        PROGRAM           ;Start PROGRAM (main part)
                        ;IF not, then continue waiting for Init code

ContPCL
      Call        SHDELAY
      goto        PCLoop


;TRANS subroutine

TRANS NOP
TRANS1
      movf        OUTCODE, 0        ;Move contents of W code to OUTCODE.
HoldT BTFSS       PIR1, TXIF
      goto        HoldT             ;If TXIF is set (empty TREG)then continue
      movwf       TXREG             ;Move W to Transmit
      Return
; END of TRANS subroutine
```

Main Part of the Code: The main portion of the code is, first, to receive control command sent from PC via serial communication established, then, second, to decode the command and to act accordingly. The commands from PC are generated by clicking the button generated in the screen of the PC using Visual Basic code.

The command code sent to PIC from PC is stored `INCODE` register in RAM area. There are numerous function commands from PC (The command code inside INCODE is indicated inside the parentheses:
(a) PIC ready inquiry: PC checks if PIC is ready for communication (0x10)

(b) Termination of the control session (0x11)
(c) Turn Off Appliance 3 (0x13)
(d) Turn Off Appliance 2 (0x14)
(e) Turn Off Appliance 1 (0x15)
(f) Turn On Appliance 3 (0x16)
(g) Turn on Appliance 2 (0x17)
(h) Turn on Appliance 1 (0x18)
(i) Learn IR (0x19)
(j) Send IR (0x1A)

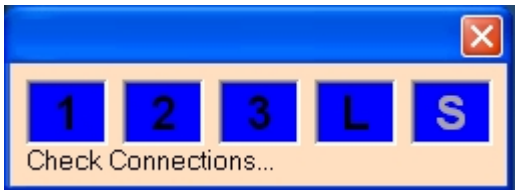

Fig. 108(a) Buttons generated using Visual Basic code

Fig. 108(b) Visual Basic code for Hyperterminal program on PC screen

So when PROGRAM is executed, the content sent from PC is first compared with the 0x10 for readiness of PIC. Once PIC is ready, then reply is made from PIC to PC. And the next PC command is then subtracted by 0x10 for easier comparison of the commands. Then, the content inside the `INCODE` would be the original number minus 0x10. In other words, the code for Turn Off 2 would be now 0x03 inside the `INCODE`. If we decrease it by 1, and check the content every time we check it, we can find the command sent from the PC:

```
PROGRAM

PollRx
      BTFSS       RCSTA, OERR       ;Check for Rx overrun error. It could
happen.
      goto        RxClear           ;If none then continue program
      bcf         RCSTA, CREN             ;by clearing CREN and
      bsf         RCSTA, CREN ;resetting it

RxClear
      BTFSS       PIR1, RCIF        ;Poll Rx for code
      goto        PollRx            ;

GetCode
      movf        RCREG, 0    ;If a code comes through, determine the code.
```

```
      movwf       INCODE            ;Codes are in order by value

      movlw       Ready             ;Is it the PIC Ready code? (ready = 0x10)
      subwf       INCODE            ;To determine, subtract Ready from INCODE
                                    ;[INCODE]=[INCODE]-[ready]
      BTFSC       STATUS, Z         ;and check the Zero flag
      Call        REPLY       ;If so then REPLY to PC and continue Polling Rx
      movf        INCODE
      BTFSC       STATUS, Z         ;The second BitCheck was added to break
      goto        PollRx            ;comparisons if the INCODE was actually
the Ready Code.
;Now ready check is over, and COMMAND check
      decf        INCODE      ;Is User requesting to end of the session?
      BTFSC       STATUS, Z
      goto        EndPrg            ;If so then end the program (bottom)

      decf        INCODE      ;Is user asking to Check IR Link?
      BTFSC       STATUS, Z   ;This is option is currently not available.
      goto        PollRx

      decf        INCODE      ;Turn off Light 3
      BTFSC       STATUS, Z
      goto        OFF3

      decf        INCODE      ;Turn off Light 2
      BTFSC       STATUS, Z
      goto        OFF2

      decf        INCODE      ;Turn off Light 1
      BTFSC       STATUS, Z
      goto        FF1

      decf        INCODE      ;Turn on Light 3
      BTFSC       STATUS, Z
      goto        ON3

      decf        INCODE      ;Turn on Light 2
      BTFSC       STATUS, Z
      goto        ON2

      decf        INCODE      ;Turn on Light 1
      BTFSC       STATUS, Z
      goto        ON1


      decf        INCODE      ;Learn an IR CODE
      BTFSC       STATUS, Z
      goto        LEARN

      decf        INCODE      ;Send learned IR CODE
      BTFSC       STATUS, Z
      goto        SendLIR

      goto        EndPrg      ;If none of these codes were right, then nother
                        ;PC program has control over the COM port. RESET PIC.
```

By the way the subroutine REPLY is for PIC to reply the READY code received from PC.

```
;Subroutine REPLY-------------------------------------------------------------
;Replies to the READY code sent from PC. If the PIC does not reply with the
;READY code within 750ms, the Software will ask you to check connections.
REPLY movlw       READY
      movwf       OUTCODE     ;Prepare to transmit READY code to PC
      call        TRANS       ;OUTCODE holds whatever to be sent to PC
      return
```

Appliance On/Off Control: To describe this part, we have to look at the encoder/decoder chip more carefully. First, the decoder is the exact mirror image of the encoder. In other words, if the encoder input is, for example, 000 at the D8, D7, and D6 inputs, then the decoder's outputs D8, D7, and D6 are 000. This allows us to easily control any device attached at the outputs of decoder, by setting/clearing the inputs of the encoder. However, there is another thing to consider in the encoder/decoder. The inputs to the encoder are not just 0 or 1: it has hi-Z state (more like a disconnected state) if not specified. In other words, an input to D8 of the encoder can be 0, 1, or hi-Z. Therefore, if I choose the three inputs D8, D7, and D6 are our three inputs to the encoder, and if I set D8=0, then inputs to D7 and D6 are interpreted as hi-Z. And the outputs at the decoder will have, D8=0, and D7& D6 would be nothing with disconnection.

In the code of the project, this is exactly what happens to control the three appliances. Only thee inputs (D8, D7, and D6) are used for the control, and the thee pins of PORTD are connected as follows: D8 to PORTD<2>, D7 to PORTD<1>, and D6 to PORTD<0>. At the decoder side, D8 - D6 are used and connected to three LEDs, mimicking three appliances, respectively. The control table is shown below. Blank spaces, unspecified, are hi-Z logic.

D8	**D7**	**D6**	**Appliance Control Logic**
PORTD<2>	**PORTD<1>**	**PORTD<1>**	
0			Turn off 1
	0		Turn off 2
		0	Turn off 1
1			Turn on 3
	1		Turn on 2
		1	Turn on 1

Therefore, coding of the appliance control consists of two parts: control the logic for the encoder and IR signal transmission based of the select logic. The following code part shows how easy the control is.

```
OFF3  bcf   PORTD, 2    ;Whatever appears at PORTD
      goto  SEND_IR

OFF2  bcf   PORTD, 1
      goto  SEND_IR

OFF1  bcf   PORTD, 0
      goto  SEND_IR
```

```
ON3   bsf   PORTD, 2
      goto  SEND_IR

ON2   bsf   PORTD, 1
      goto  SEND_IR

ON1   bsf   PORTD, 0
      goto  SEND_IR

SEND_IR     bcf   PORTC, IR
      call  SHDELAY           ;Send short IR burst
      call  SHDELAY
      Call  REPLY       ;Let the PC know that the PIC received the code.
      bsf   PORTC, IR
      goto  PollRx            ;go back to polling for incoming code
```

5. 16F877 Source Code Details

The whole code is listed here. As I mentioned above, I did not change the code made by Scotty Mazyck except a few MPLAB directives. The .INC file in the second line includes all declaration of file registers and bit information, and an .INC file for 16F877A is included at the end of the chapter. 16F877A.INC can replace 16F877.INC.

```
list p = PIC16F877
#include <P16F877.INC>

;-----------Codes Used by PIC and Program-----------------------------------
;These characters not available on a standard keyboard. This prevents
;unintended interfacing between a PC user and the DigiHouse.

Init  EQU   0x09        ;Used on power-up to initialize PIC
Ready EQU   0x10        ;Used to check Serial Link and to confirm ready to PC
EndP  EQU   0x11        ;Ends Program
IR_OK EQU   0x12        ;Not Used. Checks IR Link between PIC and other
Board.
OFF_3 EQU   0x13        ;Turn Off Light 3
OFF_2 EQU   0x14        ;Turn Off Light 2
OFF_1 EQU   0x15        ;Turn Off Light 1
ON_3  EQU   0x16        ;Turn on Light 3
ON_2  EQU   0x17        ;Turn on Light 2
ON_1  EQU   0x18        ;Turn on Light 1
LearnIR     EQU   0x19        ;Learn a IR Code
SendIR      EQU   0x1A        ;Sends a learned IR Code
IR_BAD      EQU   0x1B        ;Tells the PC that It Couldn't Learn IR CODE
IR_Wait     EQU   0X1C        ;Tells PC to wait while the PIC is learning IR
Code

;-----------CONSTANTS USED IN IR SENDING, RECEIVING, AND SAMPLING-----------
IR    EQU   0           ;PORTC bit used to enable IR Transmission
IRL   EQU   5           ;PORTD bit used for LEARNING IR CODE
IRLS  EQU   3           ;PORTD bit used to Send Learned IR CODE
```

```
MSB   EQU   7           ;Stores voltage level of incoming IR pulse <See
LEARN>

;-------------------------------------------------------------------------

      CBLOCK      0x20        ;RAM AREA for USE at address 20h
      ;Variables sed for RS-232 Communication
            StrtReg           ;20 - Used for transmitting multiple register
            EndReg            ;21

      ;Variables used for program/PIC communication
            INCODE            ;22 - The code entered from the PC to PIC
            OUTCODE           ;23 - The code sent from the PIC to PC

      ;Variables used for Loops and Delays
            first       ;24 - Used for delay loops
            second            ;25
            third       ;26
            DelVal            ;27 - Delay Value for Programmable Delay and
SendLIR
            Temp_Loop   ;28 - Temporary Loop variable

            Num_Dbl_O   ;29 - Counts double-overflows of Timer0 module
                        ;A double overflow is 128 cycles of Timer0 overflow.
                        ;Determines when to quit learning IR Code.
                        ;<See LEARN>

      ;Variables used for Learning and Sending an IR CODE
            IR_Learned  ;2A - Tells an IR Code been learned (Boolean: 0 or 1)
            IR_Reg_Max  ;2B - Maximum number of recording spaces is 80 (0x50)
            IR_Reg_Count      ;2C - Counts registers that recorded IR Pulse
Lengths
            IR_Reg_Start      ;2D - First register for Recording IR Pulse
Length
      ;To tranmit learned IR code, a loop length = 80 (0x50)

      ENDC


;-------CONFIGURE I/O PORTS-----------------------------------------------

      ORG   0x0000
      GOTO  START
      ORG   0x0005
START
      banksel     TRISB
      clrf  TRISB       ;Make PORTB output ports except bit<0>

      movlw 0x20        ;Binary = 00100000
      movwf TRISD       ;Make PORTD output ports except the bit 5(IRL)

      movlw 0xFE        ;Binary = 11111110
      movwf TRISC       ;Make PORTC input ports except bit 0 used for SendIR


      banksel     PORTC
      bsf   PORTC, IR   ;Turn off IR Transmission
```

```
      clrf  PORTD
      bcf   PORTD, IRLS
      clrf  PORTB       ;Turn off any LEDs


;-------CONFIGURE SERIAL PORT-----------------------------------------------
      banksel     SPBRG
      movlw 0x40
      movwf SPBRG       ;set baud rate
      bsf   TXSTA, BRGH ;Set for High Speed
      bcf   TXSTA, SYNC ;clear for Asynchronous Mode
      bcf   TXSTA, TX9  ;Clear for 8-bit

      banksel     RCSTA
      bsf   RCSTA, SPEN ;enable serial port

      banksel     TXSTA
      bsf   TXSTA, TXEN ;enable transmission

      banksel     RCSTA
      bsf   RCSTA, CREN ;Enable Receiver


;-------INITIATE VARIABLES--------------------------------------------------

      bcf   IR_Learned, 0     ;On reset, there is no code stored in PIC

;-------CHECK THE PC LINK---------------------------------------------------
;Initial Auto Detection Subroutine. The PIC continuously sends the Init code
;until the PC responds with the Init code.

ChkPC
      banksel     PORTB
      clrf  PORTB

PCLoop      bsf   PORTB, 7    ;LED display while waiting for PC to send codes
      bcf   PORTB, 5
      Call  SHDELAY

      movlw Init        ;Waiting for PC to reply to send the Init Code
      movwf OUTCODE           ;Init code for Xmission to PC
      Call  TRANS       ;Call TRANS Sub to send Init Code to PC


      BTFSS RCSTA, OERR ;Check for Rx overrun error.
      goto  ChkIn       ;If none then continue program
      bsf   PORTB, 5    ;If so then clear it
      bcf   RCSTA, CREN ;by clearing CREN and
      bsf   RCSTA, CREN ;resetting it


ChkIn BTFSS PIR1, RCIF  ;Check if we have received anything from PC
      goto  ContPCL           ;If not then Continue PC Looping
      bsf   PORTB, 6    ;Indicates something has been received


      movf  RCREG, 0    ;If so, then check to see if its the Init code
```

```
      sublw Init
      BTFSC STATUS, Z   ;If it is the Init code, then
      goto  PROGRAM           ;Start PROGRAM
                        ;IF not, then continue waiting for Init code

ContPCL     bcf   PORTB, 7
      Call  SHDELAY
      bcf   PORTB, 6
      goto  PCLoop


;-------PROGRAM: PROCESSES CODES, RETURNS READY CODE TO PC, ENABLES IR
TRANSMISSION

PROGRAM     bsf   PORTB, 7
      bcf   PORTB, 6    ;Set LEDs for display purposes. LED7 should hold on

PollRx      BTFSS RCSTA, OERR ;Check for Rx overrun error. It could happen.
      goto  RxClear           ;If none then continue program
      bsf   PORTB, 5    ;If so then clear it
      bcf   RCSTA, CREN ;by clearing CREN and
      bsf   RCSTA, CREN ;resetting it

RxClear     BTFSS PIR1, RCIF  ;Poll Rx for code
      goto  PollRx            ;

GetCode     movf  RCREG, 0    ;If a code comes through, determine the code.
      movwf INCODE            ;Codes are in order by value

      movlw Ready       ;Is it the PIC Ready code?
      subwf INCODE, 1   ;To determine, subtract Ready from INCODE
      BTFSC STATUS, Z   ;and check the Zero flag
      Call  REPLY       ;If so then REPLY to PC and continue Polling Rx
      movf  INCODE, 1
      BTFSC STATUS, Z   ;The second BitCheck was added to break the
      goto  PollRx            ;comparisons if the INCODE was actually the
Ready Code.

      decf  INCODE, 1   ;Is User requesting to end DigiHouse session?
      BTFSC STATUS, Z
      goto  EndPrg            ;If so then end the program (bottom)

      decf  INCODE, 1   ;Is user asking to Check IR Link?
      BTFSC STATUS, Z   ;This is option is currently not available.
      goto  PollRx

      decf  INCODE, 1   ;Turn off Light 3
      BTFSC STATUS, Z
      goto  OFF3

      decf  INCODE, 1   ;Turn off Light 2
      BTFSC STATUS, Z
      goto  OFF2

      decf  INCODE, 1   ;Turn off Light 1
      BTFSC STATUS, Z
      goto  OFF1
```

```
      decf  INCODE, 1   ;Turn on Light 3
      BTFSC STATUS, Z
      goto  ON3

      decf  INCODE, 1   ;Turn on Light 2
      BTFSC STATUS, Z
      goto  ON2

      decf  INCODE, 1   ;Turn on Light 1
      BTFSC STATUS, Z
      goto  ON1


      decf  INCODE, 1   ;Learn an IR CODE
      BTFSC STATUS, Z
      goto  LEARN

      decf  INCODE, 1   ;Send learned IR CODE
      BTFSC STATUS, Z
      goto  SendLIR

      goto  EndPrg            ;If none of these codes were right, then
another
                        ;PC program has control over the COM port. RESET PIC.


;-------SUB ROUTINES--------------------------------------------------------

;-------TRANS---------------------------------------------------------------
;Transmits Register Contents to PC. whenever a subroutine calls TRANS, it
;places a value in the W register.

TRANS NOP

TRANS1      movf  OUTCODE, 0  ;Move contents of W code to OUTCODE.
HoldT BTFSS PIR1, TXIF
      goto  HoldT       ;If TXIF is set (empty TREG) then continue

      movwf TXREG       ;Move W to Transmit
      Return



;-------REPLY---------------------------------------------------------------
;Replies to the READY code sent from PC. If the PIC does not reply with the
READY code
;within 750ms, the Software will ask you to check connections.

REPLY movlw READY
      movwf OUTCODE           ;Prepare to transmit READY code to PC
      call  TRANS
      return



;-------SET AND SEND IR-----------------------------------------------------
```

```
;Set/Clear the PORTD bit that corresponds to the correct LED, then clear
PORTC, IR to send

OFF3  bcf   PORTD, 2    ;PORTD conencts to IR encoder. Whatever appears at
PORTD
      bcf   PORTB, 2    ;is what's transmitted. PORTB is used as local
display.
      goto  SEND_IR

OFF2  bcf   PORTD, 1
      bcf   PORTB, 1
      goto  SEND_IR

OFF1  bcf   PORTD, 0
      bcf   PORTB, 0
      goto  SEND_IR

ON3   bsf   PORTD, 2
      bsf   PORTB, 2
      goto  SEND_IR

ON2   bsf   PORTD, 1
      bsf   PORTB, 1
      goto  SEND_IR

ON1   bsf   PORTD, 0
      bsf   PORTB, 0
      goto  SEND_IR

SEND_IR     bcf   PORTC, IR
      call  SHDELAY           ;Send short IR burst
      call  SHDELAY
      Call  REPLY       ;Let the PC know that the PIC received the code.
      bsf   PORTC, IR
      goto  PollRx            ;go back to polling for incoming code



;-------LEARN---------------------------------------------------------------
;This routine RECORDS by counting the number of times TMR0<7> overruns for
each voltage
;level L,H. The values are stored in order starting at RAM area 0x2D. With a
2:1
;prescaler, we can sample pulse lengths in 52.2us intervals (19.9KHz) by
polling TMRO<7>.
;Each register stores the voltage level in the MSB( 7) and the number of
TMR0<7> overflows
;in bits<6:0> The maximum value storable in each register is 52.2us x 127 =
6.5ms

;If a High pulse lasts more than 6ms, I call it a Double Overflow. therefore
the next
;register will have the same MSB as the previous register and continues
sampling.

;The maximum number of registers allowable is 80 (0x50). The maximum time
storable is
```

```
;6.5ms/reg x 80reg = 520ms, but it lessens depending on the number of pulses.

;PORTB<4> displays the sampled demodulated IR pulses.

LEARN Call  Reply             ;Let PC know we received LearnIR Code
      movlw IR_Wait                ;Tell the PC to wait while PIC samples IR
Code
      movwf OUTCODE
      Call  TRANS
      bsf   PORTB, 3          ;Show Ready Light

      movlw 0x50              ;Initiation
      movwf IR_Reg_Max        ;Set maximum Recording time to 80 Registers
      clrf  IR_Reg_Start           ;Empty the starting register
      movlw IR_Reg_Start
      movwf FSR               ;Make INDF point to starting register
      clrf  IR_Reg_Count
      clrf  TMR0              ;Reset the Timer0 Register


WAIT_IR     movlw       0xAA        ;This is a delay Loop. In the loop we
      movwf       first             ;are waiting for bit PORTD<IRL> to go
low.
W_IR_L      movwf       second                ;PORTD<IRL> is where the
Remote Receiver
      decfsz      first             ;is connected. <See constants>
      goto  IR_n1
      goto  EndWIRL                 ;<See EndWIRL below>
IR_n1       movwf       third
      decfsz      second
      goto  IR_n2
      goto  W_IR_L
IR_n2       BTFSS PORTD, IRL        ;Check if the user started sending signal

      goto  RECORD                  ;If so then break this loop and RECORD...
      decfsz      third             ;If not, keep waiting.
      goto  IR_n2
      goto  IR_n1             ;End of Waiting for IR Loop

EndWIRL     movlw IR_BAD            ;If the Loop finishes before a low value
      movwf OUTCODE           ;is detected, then send IR_BAD CODE to PC.
      Call  TRANS             ;Return to Waiting For Next Code.
      bcf   PORTB, 3          ;Turn off LED
      bcf   IR_Learned, 0     ;Remember that we didn't learned an IR Code.
      goto  PollRx            ;Go back and wait for next code.

;-------RECORD-----------------------------------------------------------------
------------------

RECORD
      banksel     OPTION_REG  ;Initiate Timer0
      movlw 0x60              ;Binary = (01100000).
      andwf OPTION_REG        ;Prescaler = 2:1, Rising Edge
      bcf   OPTION_REG, T0CS  ;Start Timer0
      banksel     TMR0
      clrf  TMR0
```

```
      goto  LowSamp                ;It's not necessary to Poll TMR0 on first
sample.



;-  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
Availbl     decfsz      IR_Reg_Max
      goto  NewReg                 ;If there are more available registers
use them
      goto  IRGOOD                 ;If not, then we are finished Recording


NewReg      incf  FSR   ;Make a new blank register if double overflows occur.
      clrf  INDF              ;Let the recording subroutines determine what
      incf  IR_Reg_Count            ;to do with it. Keep count of registers
used.

Poll_T0     BTFSS TMR0, 7                ;Poll TMR0. After 128
increments(52.2us) bit<7> goes Hi.
      goto  Poll_T0
      clrf  TMR0              ;Immediately clear TMR0 register
      bcf   INTCON, T0IF

      BTFSC PORTD, IRL        ;Check Status of IRL and the INDF<MSB> bits.
      goto  IRL_Hi            ;Remember, MSB stores the voltage level of the
      goto  IRL_Low           ;current pulse after sampling has begun.
IRL_Low     BTFSC INDF, MSB
      goto  LowRec  ;If IRL is Low and MSB is High, start new Low recording
      goto  LowSamp           ;If IRL is Low and MSB is Low, continue
sampling Low
IRL_Hi      BTFSC INDF, MSB
      goto  HiSamp      ;If IRL is Hi and MSB is Hi, continue sampling Hi
       ;goto      HiRec ;If IRL is Hi and MSB is Low, start new Hi recording

;- - - - - - - - - - - - - This is where actual recording takes place. - - -

HiRec movf  INDF, 0     ;If this register is already empty, there is no need
      BTFSC STATUS, Z         ;to start a new one. It only needs its MSB
changed
      goto  SetHi             ;to Hi. But if it's not new...

      incf  FSR         ;Move to and prepare a clean register for Hi pulse
      clrf  INDF
      incf  IR_Reg_Count
      decf  IR_Reg_Max
      BTFSC STATUS, Z         ;Check if we have any more available registers
      goto  IRGOOD            ;If not then we're finished recording.

SetHi bsf   INDF, MSB   ;Set MSB. IF IRL pulse remains high after next TMR0
HiSamp      bcf   PORTB, 4  ;overflow, then continue sampling with register.
      incf  INDF
      movf  INDF, 0     ;Check if this register has reached its capacity.
      sublw 0xFF              ;(1xxxxxxx - 11111111)
      BTFSS STATUS, Z
      goto  Poll_T0           ;No, keep filling it
      goto  Availbl     ;Yes, Check if we have any more available registers.
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
LowRec      movf  INDF, 0     ;If this register is empty, there is no need
      BTFSC STATUS, Z   ;to start a new one. There is also no need to change
      goto  LowSamp  ;its MSB because its already Lo. But, if it's not new...

      incf  FSR               ;Move to and prepare a clean register for Low
pulse.
      clrf  INDF
      incf  IR_Reg_Count            ;(LowRec not necessary on first sample)
      decf  IR_Reg_Max
      BTFSC STATUS, Z         ;Check if we have any more available registers
      goto  IRGOOD                  ;If not then we're finished recording.

SetLo bcf   INDF, MSB         ;Clear MSB. IF IRL pulse remains low after next
TMR0
LowSamp     bsf   PORTB, 4          ;overflow, then continue sampling with
this register.
      incf  INDF
      movf  INDF, 0                 ;Check if this register has reached its
capacity.
      sublw 0x7F              ;(0xxxxxxx - 01111111)
      BTFSS STATUS, Z
      goto  Poll_T0                 ;No, keep filling it
      goto  Availbl                 ;Yes, Check if we have any more available
registers.




;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;When finished, we let the Program Know.
IRGOOD      bcf   INTCON, T0IF
      banksel OPTION_REG            ;When all 80 Register are filled...
      bsf   OPTION_REG, T0CS  ;Stop Timer0
      banksel PORTB
      bcf   PORTB, 4          ;Turn off LEDs
      bcf   PORTB, 3


bsf   IR_Learned, 0           ;Remember that we learned an IR Code.
      movlw IR_OK
      movwf OUTCODE                 ;then send IR_OK CODE to PC.
      Call  TRANS             ;Return to Waiting For Next Code.


;***********Insert Testing Code Here

;*********************************

Finish      bsf   PORTB, 7
      goto  PollRx



;-------SendLIR-----------------------------------------------------------
;Send a Learned IR Code
;The SendLIR algorithm is structured around DelVal, a temporary varaible that
stores
```

```
;only the timing information of each IR Register.The MSB of INDF tells the
pulse value
;whether it's Lo or Hi. This way a TMR0 loop can be created without change to
registers.

SendLIR     BTFSC IR_Learned, 0     ;Check if PIC has learned a IR code.
      goto  SndLoop                 ;Yes. Decode IR Registers to transmit it.
       ;goto      NoCode                  ;No...

NoCode      call  REPLY             ;REPLY to the Program.

      movlw IR_BAD            ;Let the Program know that PIC has no code.
      movwf OUTCODE                 ;It will reset the button panel.
      call  Trans
      goto  PollRx                  ;Wait for next code.


SndLoop     movlw IR_Reg_Start  ;Point to the 1st Register w/ IR information
      movwf FSR
      movf  INDF, 0           ;DelVal temporarily stores TMRO loop-counts
      movwf DelVal            ;so that actual registers remain unchanged.
      movlw 0x50              ;Set Loop for 80 IR registers
      movwf Temp_Loop

      banksel OPTION_REG            ;Initiate Timer0
      movlw 0x60              ;Binary = (01100000).
      andwf OPTION_REG        ;Prescaler = 2:1, Rising Edge
      bcf   OPTION_REG, T0CS  ;Start Timer0
      banksel TMR0

      clrf  TMR0
PlsInit     bcf   PORTD, IRLS       ;Initiate first IR pulse.
Cycl_T0     BTFSS TMR0, 7                 ;Cycle TMR0 for 128 increments.
      goto  Cycl_T0
      clrf  TMR0              ;Immediately clear TMR0 register
      bcf   INTCON, T0IF

      BTFSS INDF, MSB         ;Check the MSB of the current IR Register
      bsf   PORTD, IRLS       ;Lo - Send Hi pulse to PORTD<IRLS>
      BTFSC INDF, MSB
      bcf   PORTD, IRLS       ;Hi - Send Lo pulse to PORDT<IRLS>

      decfsz      DelVal
      goto  Cycl_T0
       ;goto      SndNexR

SndNexR     incf  FSR               ;Go to the next register
      movf  INDF, 0               ;Temporarily store its value in DelVal
      movwf DelVal                ;Clear DelVal's MSB -- we don't need it.
      bcf   DelVal, MSB

      decfsz      Temp_Loop         ;Have we cycled all 80 registers.
      goto  Cycl_T0               ;No...Continue
       ;goto      NoMoreR               ;Yes...


NoMoreR     call  REPLY
```

```
      bcf   PORTD, IRLS       ;Sometimes pulse must be turned off.
      bcf   PORTB, 3
      goto  PollRx



;-------SHDELAY-------------------------------------------------------------
;Loop generates a short delay

SHDELAY     movlw       0x40
      movwf       first
DLoop       movwf       second
      decfsz      first
      goto  nest1
      goto  theend
nest1       movwf       third
      decfsz      second
      goto  nest2
      goto  DLoop
nest2       decfsz      third
      goto  nest2
      goto  nest1
theend      RETURN

;-------PDELAY--------------------------------------------------------------
-
;Programmable Delay.

PDELAY      movf  DelVal, 0   ;Move the Delay Value to the W Register.
      movwf       first
PLoop       movwf       second
      decfsz      first
      goto  Pnest1
      goto  EndIt
Pnest1      movwf       third
      decfsz      second
      goto  Pnest2
      goto  PLoop
Pnest2      decfsz      third
      goto  Pnest2
      goto  Pnest1
EndIt       RETURN



;-------EndPrg--------------------------------------------------------------
;End Prgram w/ LED Display, and go back to waiting for the Init Code (ChkPC)

EndPrg      bcf   STATUS, C
      clrf  PORTB       ;LED Display
      bsf   PORTB, 0
      movlw 0x6
      movwf Temp_Loop

Display     Call  SHDELAY
      rlf   PORTB
      decfsz      Temp_Loop
```

```
      goto  Display
      goto  ChkPC
END
```

6. Visual Basic Code for Windows Programming of Serial Communication

The following complete code is for making a serial communication windows programming instead of using Hyperterminal program.

Fig. 109 Visual Basic code for serial communication windows programming on PC screen

Visual Basic Form Code: 2 Timers (Timer1, Timer2); 1 Label (Label2), 1 Shape (Circle), 5-Button Array (Send (1 to 5)), Frame containing Buttons (Frame1)

```
Option Explicit         'This expression helps prevent coding errors.

Const Init = &H9        'These codes are sent to the PIC from the PC.
Const Ready = &H10      'They are the same values as in the PIC's code.
Const End_Prg = &H11
Const IR_OK = &H12      'The Init code is used on startup
Const OFF_3 = &H13      'The Ready code is used to check the serial link
Const OFF_2 = &H14      'prior to each transmission. If the PIC received it,
Const OFF_1 = &H15      'then it will REPLY with the Ready code.
Const ON_3 = &H16       'Using a timer, we can detect whether the PIC
Const ON_2 = &H17       'actually received the Ready code or not.
Const ON_1 = &H18
Const LearnIR = &H19    'Code used to Learn a IR Code
Const SendIR = &H1A     'Sends a learned IR Code
Const IR_BAD = &H1B     'PIC was not able to learn IR code
Const IR_Wait = &H1C    'Wait while PIC learns IR code

Dim SEND_CODE           'These variables are used to store the
Dim REC_CODE            'ASCII codes that are being sent and received.

Dim PIC_Ready As Boolean            'PIC_Ready is used for link testing
purposes.
```

```
    'Whenever PIC responds with the Ready or Init code, this value is set
True.
Dim IR_Learned As Boolean           'Program knows if an IR code was Learned.

Dim btn_Index As Integer            'Tells which button is clicked
Dim btn_State(1 To 5) As Boolean    'The Current State of each button
Dim btn_PrevColor(1 To 5)           'Stores the previous color of the button
pressed
Dim btn_Color(1 To 5)
Const btn_Yellow = &HFFFF&
Const btn_Blue = &HFF0000
Const btn_Red = &HFF&



Private Sub C_OnComm()
    'The COMM control is the heart of this project on the PC side.
    'It handles all events that take place during serial operation.

    Dim Comm_Event As Integer   'Know the event that is taking place.
    Comm_Event = C.CommEvent

    Const Send_Event = 1
    Const Rec_Event = 2

    Select Case Comm_Event
    Case Rec_Event
            REC_CODE = Asc(C.Input)
            Select Case REC_CODE
                Case Init
                        'check to see if PIC has already been initiated
                        'if not then initiate it. If it has, then the Init
                        'is being sent because the PIC has been reset
manually
                        'therefore causing it to start Init sending again.
                        'Determining the state of PIC_Ready keeps the system
from
                        'endlessly restarting itself and locking up the PC.

                    If PIC_Ready = True Then
                        PIC_Ready = False
                    ElseIf PIC_Ready = False Then
                        C.Output = Chr(Init)
                        PIC_Ready = True
                    End If

                        'enable button controls
                    Frame1.Enabled = True
                    Timer1.Enabled = False
                    Label2.Caption = "Ready..."
                    Circle1.Visible = False

                Case Ready
                        'The PIC REPLYs with the Ready code after each Tx.
                        'This is used to ensure that serial link is still
good.
                        'The timer will disable the buttons if the Ready code
```

```
                    'is not received from the PIC within 500ms.
                Timer1.Enabled = False
                Frame1.Enabled = True
                Label2.Caption = "Ready..."
                Circle1.Visible = False
                PIC_Ready = True

                    'Controlling the Buttons' Function and Color
                    'The state of the buttons change only if the
                    'Ready code was received after the button was presed.
                If (0 < btn_Index) And (btn_Index < 4) Then
                    If btn_State(btn_Index) = False Then
                        Send(btn_Index).BackColor = btn_Yellow
                        Else: Send(btn_Index).BackColor = btn_Blue
                    End If

                    'Toggle the button state and clear index
                    btn_State(btn_Index) = Not btn_State(btn_Index)
                End If
                btn_Index = 0

            Case IR_Wait
                    'The PIC is telling the Program to wait while it
                    'learns an IR code. The program halts for 20 seconds
                    'waiting on PIC. The PIC requires less than 20
seconds.
                Timer2.Enabled = True
                Frame1.Enabled = False
                Label2.Caption = "Learning your code..."
                Circle1.Visible = True

            Case IR_BAD
                Dim errPrompt As String
                Dim errTitle As String
                Dim Response
                errPrompt = "Digihouse did not read remote control"
                errTitle = "DigiHouse"
                Beep
                Response = MsgBox(errPrompt, vbInformation, errTitle)
                Timer2.Enabled = False
                Frame1.Enabled = True
                Send(5).Enabled = False
                Send(5).BackColor = btn_Blue
                Label2.Caption = "Ready..."
                Circle1.Visible = False

            Case IR_OK
                Timer2.Enabled = False
                Frame1.Enabled = True
                Send(5).Enabled = True
                Send(5).BackColor = btn_Yellow
                Label2.Caption = "Ready..."
                Circle1.Visible = False
                Response = MsgBox("Learned Code", , "DigiHouse")

        End Select
```

```
            'Text2 = Text2 & REC_CODE
            'Text1.Text = Text1.Text & Comm_Event

    Case Send_Event

            'Text1.Text = Text1.Text & Comm_Event
    End Select


End Sub

Private Sub Form_Load()
    On Error GoTo CheckError

    If C.PortOpen = False Then C.PortOpen = True
    C.Output = Chr(Ready)
    Exit Sub

CheckError:
    Dim errPrompt As String
    Dim errTitle As String
    Dim Response
    errPrompt = "Another Program is already using the Serial Port."
    errTitle = "DigiHouse"
    Beep
    Response = MsgBox(errPrompt, vbInformation, errTitle)
    Unload Me
End Sub

Private Sub Form_Unload(Cancel As Integer)
    On Error GoTo CheckError

    C.Output = Chr(End_Prg)

    C.PortOpen = False
CheckError:
    Exit Sub
End Sub


Private Sub Label2_Click()
        If PIC_Ready = False Then C.Output = Chr(Ready)
End Sub

Private Sub Send_Click(Index As Integer)
        'Remember which button was clicked
    btn_Index = Index

        'Determine whether to turn light on or off
    Select Case Index
        Case 1
            If btn_State(Index) = False Then
                SEND_CODE = Chr(ON_1)
                Else: SEND_CODE = Chr(OFF_1)
            End If

        Case 2
```

```
            If btn_State(Index) = False Then
                SEND_CODE = Chr(ON_2)
                Else: SEND_CODE = Chr(OFF_2)
            End If
        Case 3
            If btn_State(Index) = False Then
                SEND_CODE = Chr(ON_3)
                Else: SEND_CODE = Chr(OFF_3)
            End If
        Case 4
            SEND_CODE = Chr(LearnIR)
        Case 5
            SEND_CODE = Chr(SendIR)
    End Select

    Timer1.Enabled = True
    Frame1.Enabled = False

        'Send the code the COMM object.
    C.Output = SEND_CODE
        'After the COMM object sends the code to the PIC, it will wait
        'for the PIC to reply with the Ready code. If there is no response
        'from PIC, then the state of the buttons remain unchanged.
End Sub

Private Sub Send_MouseDown(Index As Integer, Button As Integer, Shift As
Integer, X As Single, Y As Single)

    If Button = 1 Then
        btn_PrevColor(Index) = Send(Index).BackColor
        Send(Index).BackColor = btn_Red
    End If
End Sub

Private Sub Send_MouseUp(Index As Integer, Button As Integer, Shift As
Integer, X As Single, Y As Single)
    If Button = 1 Then Send(Index).BackColor = btn_PrevColor(Index)
End Sub

Private Sub Text1_DblClick()
    C.PortOpen = True
End Sub

Private Sub Text2_DblClick()
    C.PortOpen = False
End Sub

Private Sub Timer1_Timer()
        'If a Ready signal is not sent within 500ms, then
        'the the timer completes its cycle. It clears all present values
        'as well as disables the buttons, and flags a message
    PIC_Ready = False
    btn_Index = 0

    Frame1.Enabled = False

    Circle1.Visible = Not Circle1.Visible     'blink the Circle
```

```
    Label2.Caption = "Check Connections..." 'flag a warning message

End Sub

Private Sub Timer2_Timer()
        'If the PIC doesn't learn IR Code within 20 sec, then
        'the program assumes that it lost connection. It disables
        'itself and enables the Reply error timer, Timer1.
    Timer2.Enabled = False

    Dim errPrompt As String
    Dim errTitle As String
    Dim Response
    errPrompt = "DigiHouse lost connection System"
    errTitle = "DigiHouse"
    Beep
    Response = MsgBox(errPrompt, vbInformation, errTitle)
    Timer1.Enabled = True
End Sub
```

7. 17F877A.INC file

```
        LIST
; P16F877A.INC  Standard Header File, Version 1.00    Microchip Technology,
Inc.
        NOLIST

; This header file defines configurations, registers, and other useful bits
of
; information for the PIC16F877A microcontroller.  These names are taken to
match
; the data sheets as closely as possible.

; Note that the processor must be selected before this file is
; included.  The processor may be selected the following ways:

;       1. Command line switch:
;               C:\ MPASM MYFILE.ASM /PIC16F877A
;       2. LIST directive in the source file
;               LIST   P=PIC16F877A
;       3. Processor Type entry in the MPASM full-screen interface

;==========================================================================
;
;       Revision History
;
;==========================================================================

;Rev:   Date:    Reason:
;1.01   09/13/01 Added the PIR2 bit CMIF and the PIE2 bit CMIE
;1.00   04/19/01 Initial Release (BD - generated from PIC16F877.inc)

;==========================================================================
;
;       Verify Processor
```

```
;
;==========================================================================

        IFNDEF __16F877A
           MESSG "Processor-header file mismatch.  Verify selected
processor."
        ENDIF

;==========================================================================
;
;       Register Definitions
;
;==========================================================================

W                           EQU     H'0000'
F                           EQU     H'0001'

;----- Register Files------------------------------------------------------

INDF                        EQU     H'0000'
TMR0                        EQU     H'0001'
PCL                         EQU     H'0002'
STATUS                      EQU     H'0003'
FSR                         EQU     H'0004'
PORTA                       EQU     H'0005'
PORTB                       EQU     H'0006'
PORTC                       EQU     H'0007'
PORTD                       EQU     H'0008'
PORTE                       EQU     H'0009'
PCLATH                      EQU     H'000A'
INTCON                      EQU     H'000B'
PIR1                        EQU     H'000C'
PIR2                        EQU     H'000D'
TMR1L                       EQU     H'000E'
TMR1H                       EQU     H'000F'
T1CON                       EQU     H'0010'
TMR2                        EQU     H'0011'
T2CON                       EQU     H'0012'
SSPBUF                      EQU     H'0013'
SSPCON                      EQU     H'0014'
CCPR1L                      EQU     H'0015'
CCPR1H                      EQU     H'0016'
CCP1CON                     EQU     H'0017'
RCSTA                       EQU     H'0018'
TXREG                       EQU     H'0019'
RCREG                       EQU     H'001A'
CCPR2L                      EQU     H'001B'
CCPR2H                      EQU     H'001C'
CCP2CON                     EQU     H'001D'
ADRESH                      EQU     H'001E'
ADCON0                      EQU     H'001F'

OPTION_REG                  EQU     H'0081'
TRISA                       EQU     H'0085'
TRISB                       EQU     H'0086'
TRISC                       EQU     H'0087'
TRISD                       EQU     H'0088'
```

```
TRISE                           EQU     H'0089'
PIE1                            EQU     H'008C'
PIE2                            EQU     H'008D'
PCON                            EQU     H'008E'
SSPCON2                         EQU     H'0091'
PR2                             EQU     H'0092'
SSPADD                          EQU     H'0093'
SSPSTAT                         EQU     H'0094'
TXSTA                           EQU     H'0098'
SPBRG                           EQU     H'0099'
CMCON                           EQU     H'009C'
CVRCON                          EQU     H'009D'
ADRESL                          EQU     H'009E'
ADCON1                          EQU     H'009F'

EEDATA                          EQU     H'010C'
EEADR                           EQU     H'010D'
EEDATH                          EQU     H'010E'
EEADRH                          EQU     H'010F'

EECON1                          EQU     H'018C'
EECON2                          EQU     H'018D'

;----- STATUS Bits --------------------------------------------------------

IRP                             EQU     H'0007'
RP1                             EQU     H'0006'
RP0                             EQU     H'0005'
NOT_TO                          EQU     H'0004'
NOT_PD                          EQU     H'0003'
Z                               EQU     H'0002'
DC                              EQU     H'0001'
C                               EQU     H'0000'

;----- INTCON Bits --------------------------------------------------------

GIE                             EQU     H'0007'
PEIE                            EQU     H'0006'
T0IE                            EQU     H'0005'
INTE                            EQU     H'0004'
RBIE                            EQU     H'0003'
T0IF                            EQU     H'0002'
INTF                            EQU     H'0001'
RBIF                            EQU     H'0000'

;----- PIR1 Bits ----------------------------------------------------------

PSPIF                           EQU     H'0007'
ADIF                            EQU     H'0006'
RCIF                            EQU     H'0005'
TXIF                            EQU     H'0004'
SSPIF                           EQU     H'0003'
CCP1IF                          EQU     H'0002'
TMR2IF                          EQU     H'0001'
TMR1IF                          EQU     H'0000'

;----- PIR2 Bits ----------------------------------------------------------
```

```
CMIF                    EQU     H'0006'
EEIF                    EQU     H'0004'
BCLIF                   EQU     H'0003'
CCP2IF                  EQU     H'0000'

;----- T1CON Bits --------------------------------------------------------

T1CKPS1                 EQU     H'0005'
T1CKPS0                 EQU     H'0004'
T1OSCEN                 EQU     H'0003'
NOT_T1SYNC              EQU     H'0002'
T1INSYNC                EQU     H'0002'    ; Backward compatibility only
T1SYNC                  EQU     H'0002'
TMR1CS                  EQU     H'0001'
TMR1ON                  EQU     H'0000'

;----- T2CON Bits --------------------------------------------------------

TOUTPS3                 EQU     H'0006'
TOUTPS2                 EQU     H'0005'
TOUTPS1                 EQU     H'0004'
TOUTPS0                 EQU     H'0003'
TMR2ON                  EQU     H'0002'
T2CKPS1                 EQU     H'0001'
T2CKPS0                 EQU     H'0000'

;----- SSPCON Bits -------------------------------------------------------

WCOL                    EQU     H'0007'
SSPOV                   EQU     H'0006'
SSPEN                   EQU     H'0005'
CKP                     EQU     H'0004'
SSPM3                   EQU     H'0003'
SSPM2                   EQU     H'0002'
SSPM1                   EQU     H'0001'
SSPM0                   EQU     H'0000'

;----- CCP1CON Bits ------------------------------------------------------

CCP1X                   EQU     H'0005'
CCP1Y                   EQU     H'0004'
CCP1M3                  EQU     H'0003'
CCP1M2                  EQU     H'0002'
CCP1M1                  EQU     H'0001'
CCP1M0                  EQU     H'0000'

;----- RCSTA Bits --------------------------------------------------------

SPEN                    EQU     H'0007'
RX9                     EQU     H'0006'
RC9                     EQU     H'0006'    ; Backward compatibility only
NOT_RC8                 EQU     H'0006'    ; Backward compatibility only
RC8_9                   EQU     H'0006'    ; Backward compatibility only
SREN                    EQU     H'0005'
CREN                    EQU     H'0004'
ADDEN                   EQU     H'0003'
```

```
FERR                        EQU     H'0002'
OERR                        EQU     H'0001'
RX9D                        EQU     H'0000'
RCD8                        EQU     H'0000'    ; Backward compatibility only

;----- CCP2CON Bits --------------------------------------------------------

CCP2X                       EQU     H'0005'
CCP2Y                       EQU     H'0004'
CCP2M3                      EQU     H'0003'
CCP2M2                      EQU     H'0002'
CCP2M1                      EQU     H'0001'
CCP2M0                      EQU     H'0000'

;----- ADCON0 Bits ---------------------------------------------------------

ADCS1                       EQU     H'0007'
ADCS0                       EQU     H'0006'
CHS2                        EQU     H'0005'
CHS1                        EQU     H'0004'
CHS0                        EQU     H'0003'
GO                          EQU     H'0002'
NOT_DONE                    EQU     H'0002'
GO_DONE                     EQU     H'0002'
ADON                        EQU     H'0000'

;----- OPTION_REG Bits -----------------------------------------------------

NOT_RBPU                    EQU     H'0007'
INTEDG                      EQU     H'0006'
T0CS                        EQU     H'0005'
T0SE                        EQU     H'0004'
PSA                         EQU     H'0003'
PS2                         EQU     H'0002'
PS1                         EQU     H'0001'
PS0                         EQU     H'0000'

;----- TRISE Bits ----------------------------------------------------------

IBF                         EQU     H'0007'
OBF                         EQU     H'0006'
IBOV                        EQU     H'0005'
PSPMODE                     EQU     H'0004'
TRISE2                      EQU     H'0002'
TRISE1                      EQU     H'0001'
TRISE0                      EQU     H'0000'

;----- PIE1 Bits -----------------------------------------------------------

PSPIE                       EQU     H'0007'
ADIE                        EQU     H'0006'
RCIE                        EQU     H'0005'
TXIE                        EQU     H'0004'
SSPIE                       EQU     H'0003'
CCP1IE                      EQU     H'0002'
TMR2IE                      EQU     H'0001'
TMR1IE                      EQU     H'0000'
```

```
;----- PIE2 Bits ----------------------------------------------------------

CMIE                    EQU     H'0006'
EEIE                    EQU     H'0004'
BCLIE                   EQU     H'0003'
CCP2IE                  EQU     H'0000'

;----- PCON Bits ----------------------------------------------------------

NOT_POR                 EQU     H'0001'
NOT_BO                  EQU     H'0000'
NOT_BOR                 EQU     H'0000'

;----- SSPCON2 Bits -------------------------------------------------------

GCEN                    EQU     H'0007'
ACKSTAT                 EQU     H'0006'
ACKDT                   EQU     H'0005'
ACKEN                   EQU     H'0004'
RCEN                    EQU     H'0003'
PEN                     EQU     H'0002'
RSEN                    EQU     H'0001'
SEN                     EQU     H'0000'

;----- SSPSTAT Bits -------------------------------------------------------

SMP                     EQU     H'0007'
CKE                     EQU     H'0006'
D                       EQU     H'0005'
I2C_DATA                EQU     H'0005'
NOT_A                   EQU     H'0005'
NOT_ADDRESS             EQU     H'0005'
D_A                     EQU     H'0005'
DATA_ADDRESS            EQU     H'0005'
P                       EQU     H'0004'
I2C_STOP                EQU     H'0004'
S                       EQU     H'0003'
I2C_START               EQU     H'0003'
R                       EQU     H'0002'
I2C_READ                EQU     H'0002'
NOT_W                   EQU     H'0002'
NOT_WRITE               EQU     H'0002'
R_W                     EQU     H'0002'
READ_WRITE              EQU     H'0002'
UA                      EQU     H'0001'
BF                      EQU     H'0000'

;----- TXSTA Bits ---------------------------------------------------------

CSRC                    EQU     H'0007'
TX9                     EQU     H'0006'
NOT_TX8                 EQU     H'0006'    ; Backward compatibility only
TX8_9                   EQU     H'0006'    ; Backward compatibility only
TXEN                    EQU     H'0005'
SYNC                    EQU     H'0004'
BRGH                    EQU     H'0002'
```

```
TRMT                            EQU     H'0001'
TX9D                            EQU     H'0000'
TXD8                            EQU     H'0000'    ; Backward compatibility only


;----- CMCON Bits --------------------------------------------------------
C2OUT                           EQU     H'0007'
C1OUT                           EQU     H'0006'
C2INV                           EQU     H'0005'
C1INV                           EQU     H'0004'
CIS                             EQU     H'0003'
CM2                             EQU     H'0002'
CM1                             EQU     H'0001'
CM0                             EQU     H'0000'

;----- CVRCON Bits -------------------------------------------------------
CVREN                           EQU     H'0007'
CVROE                           EQU     H'0006'
CVRR                            EQU     H'0005'
CVR3                            EQU     H'0003'
CVR2                            EQU     H'0002'
CVR1                            EQU     H'0001'
CVR0                            EQU     H'0000'

;----- ADCON1 Bits -------------------------------------------------------

ADFM                            EQU     H'0007'
PCFG3                           EQU     H'0003'
PCFG2                           EQU     H'0002'
PCFG1                           EQU     H'0001'
PCFG0                           EQU     H'0000'

;----- EECON1 Bits -------------------------------------------------------

EEPGD                           EQU     H'0007'
WRERR                           EQU     H'0003'
WREN                            EQU     H'0002'
WR                              EQU     H'0001'
RD                              EQU     H'0000'

;=========================================================================
;
;       RAM Definition
;
;=========================================================================

        __MAXRAM H'1FF'
        __BADRAM H'8F'-H'90', H'95'-H'97', H'9A'-H'9B'
        __BADRAM H'105', H'107'-H'109'
        __BADRAM H'185', H'187'-H'189', H'18E'-H'18F'

;=========================================================================
;
;       Configuration Bits
;
;=========================================================================
```

```
_CP_ALL                         EQU     H'3FFF'
_CP_OFF                         EQU     H'1FFF'
_DEBUG_OFF                      EQU     H'3FFF'
_DEBUG_ON                       EQU     H'37FF'
_WRT_OFF                        EQU     H'3FFF'    ; No prog memmory write
protection
_WRT_256                        EQU     H'3DFF'    ; First 256 prog memmory
write protected
_WRT_1FOURTH                    EQU     H'3BFF'    ; First quarter prog memmory
write protected
_WRT_HALF                       EQU     H'39FF'    ; First half memmory write
protected
_CPD_OFF                        EQU     H'3FFF'
_CPD_ON                         EQU     H'3EFF'
_LVP_ON                         EQU     H'3FFF'
_LVP_OFF                        EQU     H'3F7F'
_BODEN_ON                       EQU     H'3FFF'
_BODEN_OFF                      EQU     H'3FBF'
_PWRTE_OFF                      EQU     H'3FFF'
_PWRTE_ON                       EQU     H'3FF7'
_WDT_ON                         EQU     H'3FFF'
_WDT_OFF                        EQU     H'3FFB'
_RC_OSC                         EQU     H'3FFF'
_HS_OSC                         EQU     H'3FFE'
_XT_OSC                         EQU     H'3FFD'
_LP_OSC                         EQU     H'3FFC'

        LIST
```