8bit IR Demo

Engineering Design Forum 2012

Preliminary 1.0.0

Executive Summary

An Infrared (IR) example on the 8bit PIC16LF1939 was constructed for the EDF (Engineering Design Forum) *One PIC Platform* Board. The purpose of the EDF board is to demonstrate seamless integration between the three architectures. This demo complements the theme of integration by showing how a hardware specific demo can still be incorporated into an existing project which is shared between architectures.

A graphical user interface (GUI) is used in conjunction with the EDF board and a PICtail with an IR transceiver that is approved by the Infrared Data Association (IrDA[1]). A connected computer can now send and receive single bytes of data as well as send data to another board via the IR link. With or without the GUI, multiple boards or any other IrDA approved device can communicate to one another.

Table of Contents

[1] http://www.irda.org/

Intro

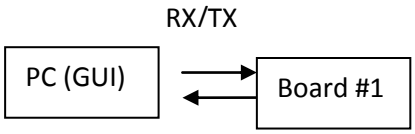
The EDF board consists of 3 PICs (PIC16,PIC24, PIC32) from 3 different architectures (8BIT,16bit, 32bit) respectively. All devices share a portion of code called the *Base Code.* This is to demonstrate the flexibility and cross-platform support of multiple compilers. From this, each platform has its own specific demo that is separate from any other device and will only run on that PIC. This IR demo can only be run by the PIC16.

This document explains the top level functionality of this demo as well as the technical details such as the IrDA and UART protocol. The brief overview of the software routines on the PIC as well as the GUI will be discussed.

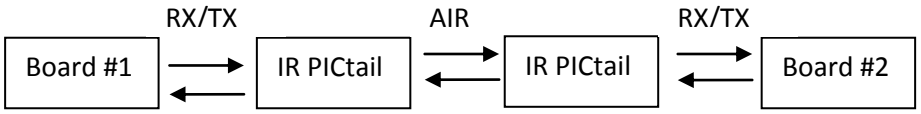
How to use

There are three setups that can communicate with the EDF Board.

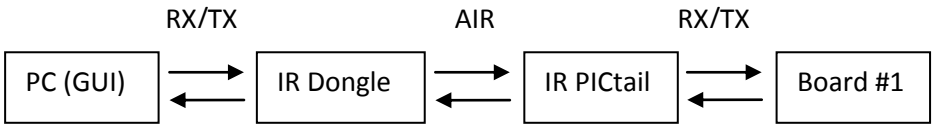
1. PC→ Board



2. Board → Board



3. PC → IR Dongle → Board



The 2nd and 3rd options utilize the associated GUI which was custom made for this demonstration. Please see the 'GUI' section for more detail in setting it up.

For board to board communication, simply attach the IR PICtail on both and press the mTouch buttons. The IR dongle simply supplements the need of another board to communicate via IR. The IR PICtail must be inserted into the EDF board for any IR demo to operate properly. The board must face towards the board with the IR transceiver facing towards the LCD and programmer header!

Please see the 'Getting Started' Guide for information regarding how to select the 8bit IR project and programming.

When the 8bit_IR project is compiled and programmed, the EDF board should display what is shown in Figure 4:

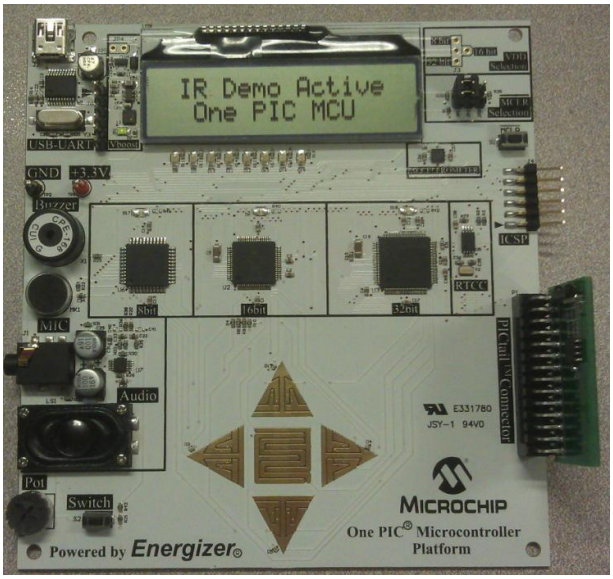


Figure 1: Splash screen of the 8bit_IR demo

This indicates that the IR demo is active and waiting commands. The potentiometer is now disabled. An orange activity LED should flash every few seconds to show that the PIC is still running. Table 1 shows what is displayed on the LCD if any of the following actions are performed.

	Button Press on board	PC TX to board	IR –TX from another board
IR - RX			x
IR - TX	x	x	
PC- RX		x	
PC - TX	x	x	x

Table 1: Each 'x' marks what the board transmits/receives upon each action item.

For every action, an RX or TX LED should flicker. If the same button is held down on the board, then the board will transmit the same byte continuously. The LCD will not refresh unless a different byte is transmitted from its last transmission.

Each button press will send a single byte of either 'U', 'R','L','R', or 'E' for up, right, left, right, and enter respectively. Figure 5 shows what is presented if the left button is pressed.

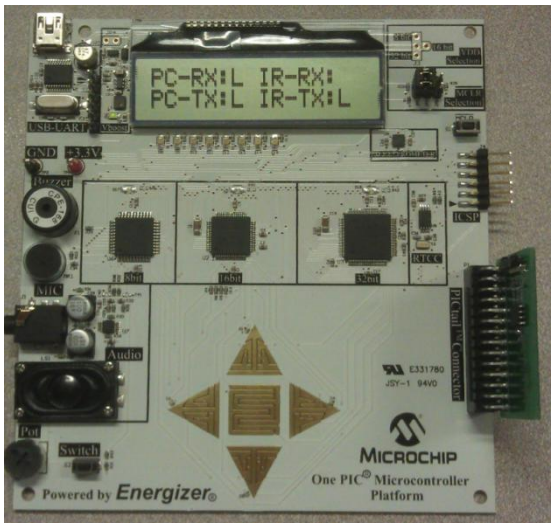


Figure 5: A button press will cause the byte to be transmitted to the IR PICtail and USB

If a byte is sent over the USB cable via the GUI, the program will echo the same character back to the PC as well as transmit it over the IR link. Figure 6 shows what happens if the letter 'k' is sent.

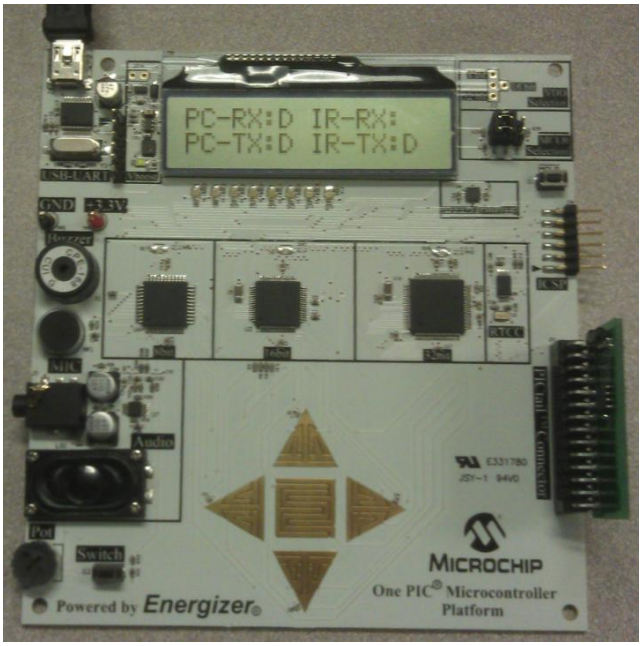


Figure 5: USB reception will cause the same byte to be echoed back and transmitted to IR PICtail

If a byte is received through the IR link, then the board will transmit the byte through USB . The IR transceivers MUST be aimed towards one another for good reception. The maximum range of the transceiver is around 6 feet.

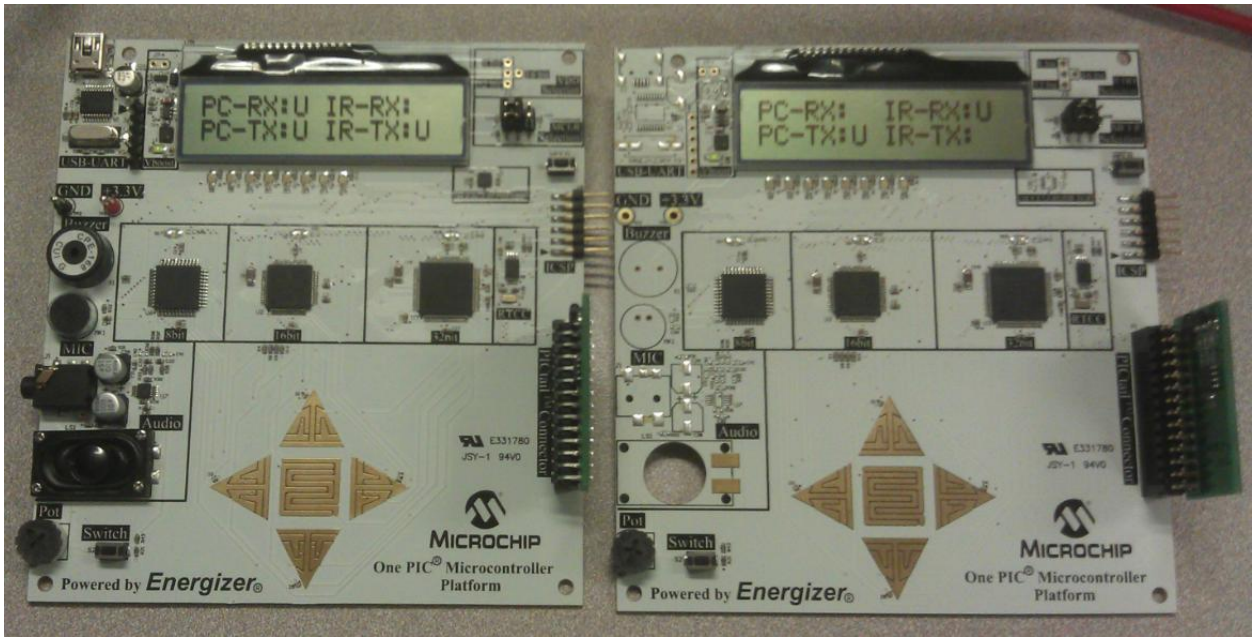


Figure 6: A full example of Board→Board communication (Demo 2). The left button is pressed on the leftmost board and is transmitted to the board on the right. The boards must have their transceiver facing one another for correct reception despite what the picture shows.

A board does not have to be plugged into a PC for this demo to work. Two or more boards can communicate on their own if powered by battery, programmer, or USB.

GUI

A GUI was developed to accommodate demos 1 and 3 above. The MCP2200 driver must be installed as well as the .NET framework 2.0 and above. The SimpleIO-M.dll must also be in the same directory as the executable. This file comes packaged in the same zip file as the driver listed above. Once the driver is installed and a board or dongle (or any MCP2200 device) is plugged in, Windows will create a virtual COM port. Figure 12 shows the splash screen of the GUI.

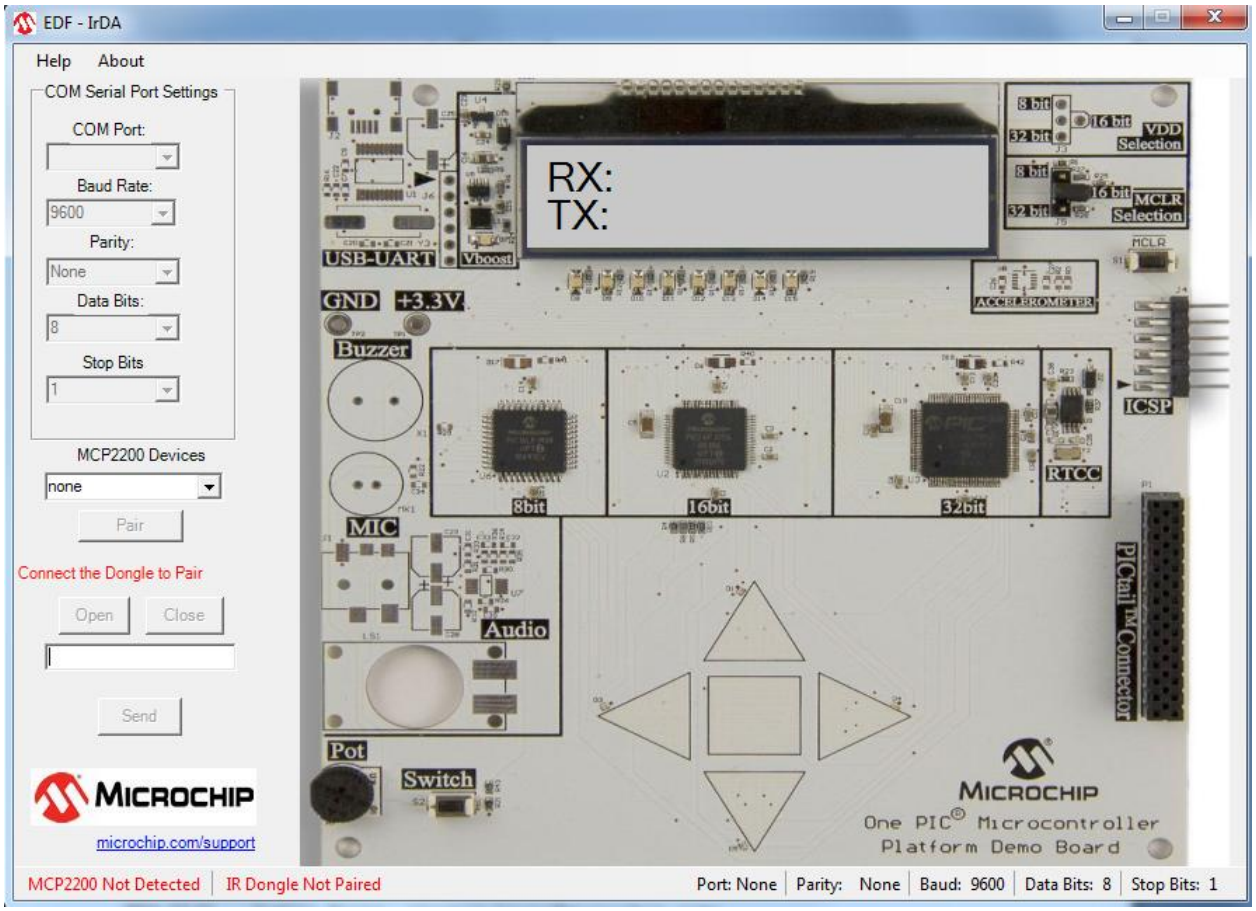


Figure 12: Main appearance of the GUI

This is the screen that will be presented when the GUI is first started. The Dongle must be paired first before any communication is to occur. Only plug in one MCP2200 device (the dongle) to pair. Once paired successfully, you should see something similar to Figure 13.

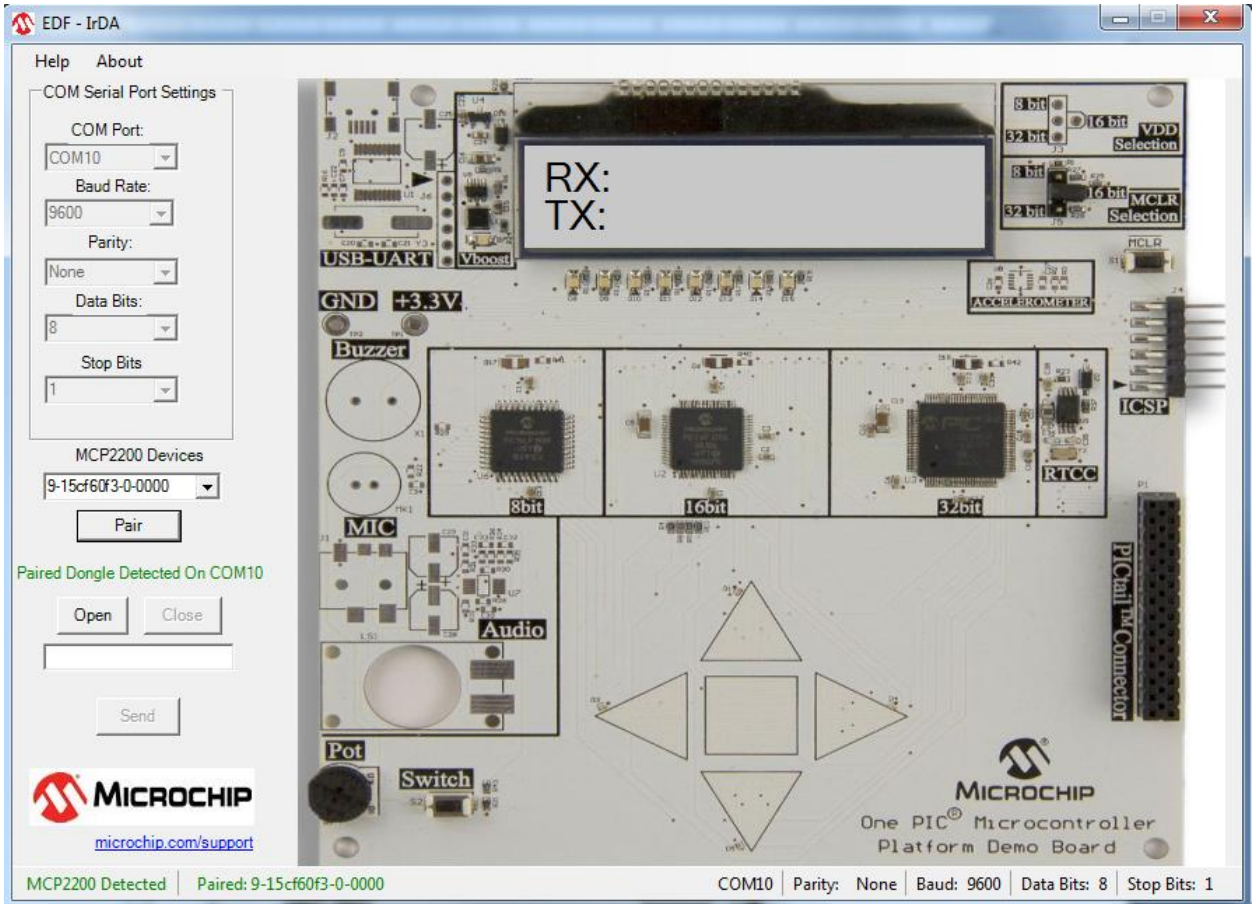


Figure 13: Display after the dongle is connected and paired.

The MCP2200 serial number as well as the COM port in which it is connected to will now be saved and automatically loaded on each time the GUI is run. These settings are saved in on the PC in the user's local AppData folder.

Press 'Open' to open the port and start communicating. When this is pressed, the Serial Port will be assigned the settings that are currently selected in their respective drop box. This must be performed! A "SUCCESS" will appear if windows could open the port. *This does not test if the board is successfully communicating with the GUI.*

Only the default settings work as of this writing, which is why some boxes are grayed out. If any of these settings are changed from their default, a: "Warning. Unsupported Settings Used" will be issued.

The EDF board should now be programmed with the 8bit_IR project. It can be powered by either a battery or USB. The GUI should now display what button was pressed as well as TX/RX LEDs. The RX/TX status labels are with respect to the Dongle as seen in Figure 14.

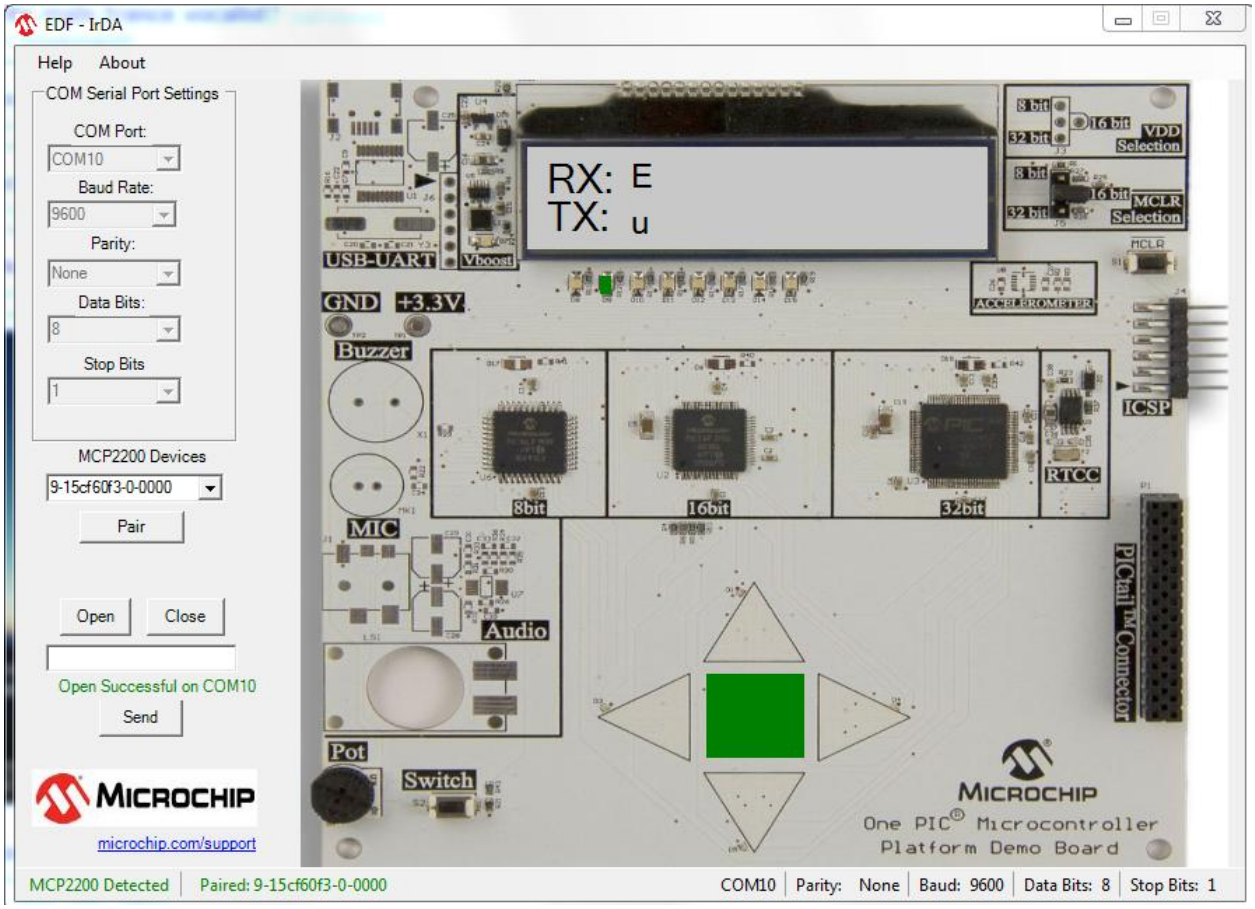


Figure 14: This is what is displayed if the middle button is pressed on the attached board.

To send a single character, type the character in the textbox and then either press enter or click ‘send’. The character will now be erased and displayed on the LCD next to ‘TX: ‘. Only the first character can be sent at a time. If a string is entered into the box, it will be discarded with the exception of the first byte.

Board-to-Board communication is also possible without the use of the IR dongle. Simply connect the EDF board to the PC and pair it. You can now communicate to the EDF board directly.

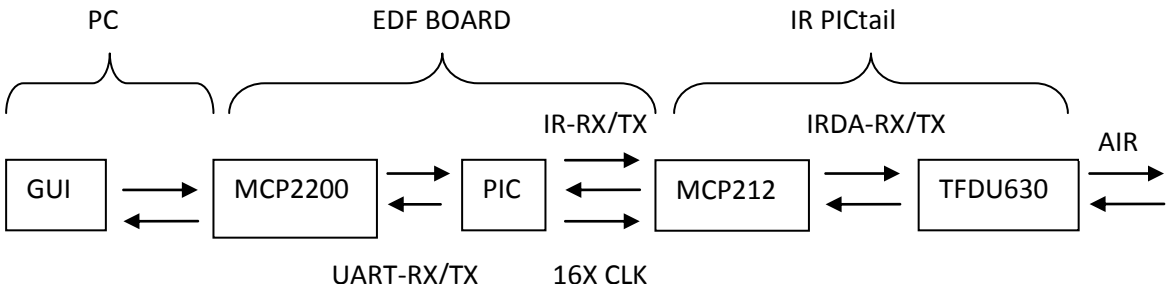
Note:

Please do not disconnect any MCP2200 devices while the GUI is running. The virtual serial driver and protocol are antiquated and will break if this happens. To be safe, if any error messages occur such as “COMX does not exist”, simply unplug the device and restart the GUI.

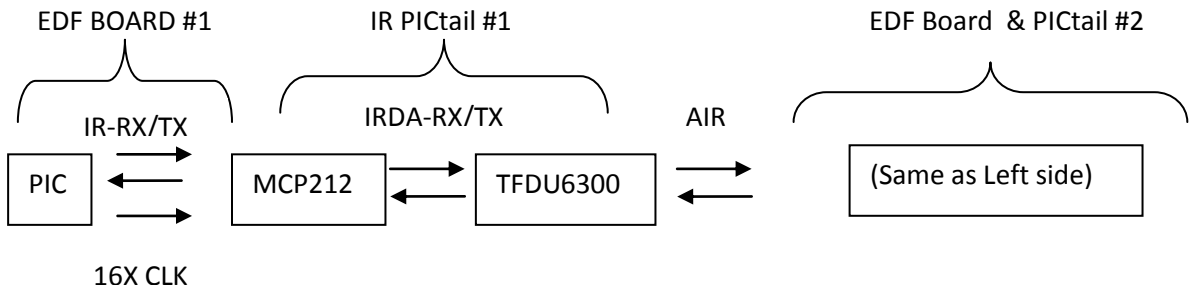
Technical

Details over the software routines and hardware add-ons such as the IR Dongle and IR PICtail will now be discussed in full detail. A more in-depth block diagram than the one shown in Figure 1 can be seen below in Figure 5.

1. PC→Board



2. Board→Board



3. PC→IR Dongle→Board

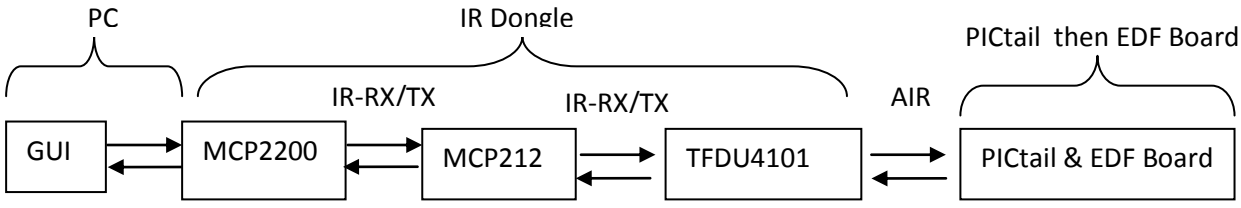


Figure 7: Enhanced Block diagram of all communications

PIC

All of the code is written in C in MPLABX Beta 7.12 with Doxygen style comments across multiple compilers. Please read the Doxygen html files to get a good grasp on the flow of the code. Table 2 lists some characteristics of each device.

	8bit	16bit	32bit
Device	PIC16LF1939	PIC24FJ256GA106	PIC32MX795F512L
Complier	HI-TECH	C30	C32
Demo	IR	RF	WIFI
Frequency	32MHz	16MHz	20MHz
mTouch	2.00 Beta3 Framework	?	?

Table 2: List of characteristics for each architecture

While the Base code is shared and contains several ISP look-a-like functions, the demo code is completely separate and has its own initialization. Each demo has a '*demo_main_loop'* function which delegates tasks. The following tasks in table 2 are used during this 8bit IR demo:

Task	Comment	Speed	ISR?	peripheral
mTouch	Touch buttons polling	Every 100us	X	ADC & TMR2
16X mcp2122 clock	CLK for IR PICtail	153.6kHz		CCP2
IR start bit poll	Bit-bang IR UART link	Every 20us	x	TMR4
Status LED	"I'm Alive" blinking LED	Every 4s		Main loop hardcoded
MCP2200 RX	Hardware UART RX	Anytime RX	x	TXREG

Table 3: IR Demo tasks that affect otherwise normal operation

With no RX/TX activity, the PIC is constantly polling the mTouch pins. When a byte is received, it is printed to the LCD and saved to ensure that the LCD does not constantly update unless a new byte is received/transmitted. This is because if a button is held down, the board will transmit a byte every time the mtouch interrupt occurs and will cause the LCD to refresh constantly. It is in this ISR that each flag will be set as seen in Figure 8.

```
extern IR_DEMO_RX rx_byte;
extern IR_DEMO_FLAG rx_tx_flag;

void interrupt ISR(void) {
    //mTouch Interrupt
    if (EDF_INT_SourceFlagGet(INT_TMR_2) && EDF_INT_SourceEnableGet(INT_TMR_2)) {
        mTouch_Scan();
        EDF_TMR_CounterSet(TMR_2, 255);
        EDF_INT_SourceFlagClear(INT_TMR_2);
    }

    //I2C interrupt
    if (EDF_INT_SourceFlagGet(INT_SSP)) {
        ISR_i2c();
        EDF_INT_SourceFlagClear(INT_SSP);
    }
    //IrDA Demo - Poll RX pin
    if (EDF_INT_SourceFlagGet(INT_TMR_4)) {
        EDF_INT_SourceFlagClear(INT_TMR_4);
        if (!IRDA_RX) { //UART Start Bit is active LOW
            rx_byte.irda_rx = irda_getch(); //get the byte NOW!
            rx_tx_flag.IRDA_RX_FLAG = 1; //print to the LCD out of the ISR
        }
    }
    //IrDA Demo - MCP2200 RX flag
    if (RCIF) {
        RCIF = 0;
        rx_byte.mcp2200_rx = RCREG; //must do this to clear interrupt (empty RCREG)
        rx_tx_flag.MCP2200_RX_FLAG = 1;
    }
}
```

Figure 8: ISR used in the Demo. The i2c is not used.

The individual flags and bytes are in structures that have a GLOBAL scope. Any variables in an interrupt must be made global to avoid losing the data if they are to be used elsewhere. The TOUCH_RX_FLAG is set inside of the mTouch_Scan() routine.

The main loop constantly checks these flags and clears them when finished transmitting/receiving so as to not continuously send if no action is demanded from the user. Figure 9 shows the demo's main loop.

```
while (1) {
    if (counter++ == 300) { //hardcoded delay loop to blink TX/RX status LEDs
        counter = 0;
        if (blink_tx != 0) {
            gpLED_1 ^= 1; //blink RX LED
            blink_tx--;
        }
        if (blink_rx != 0) {
            gpLED_2 ^= 1; //blink TX LED
            blink_rx--;
        }
        if (blink_tx == 0)
            gpLED_1 = 0; //Reset
        if (blink_rx == 0)
            gpLED_2 = 0; //Reset

        if (status_led++ == 30) {
            gpLED_6 = 1; //set the "I'm Alive" LED indicator
        }
    }
    if (status_led == 33) { //only allow this LED to be on for a short while
        status_led = 0; //clear the "I'm Alive" LED indicator
        gpLED_6 = 0;
    }

    if (rx_tx_flag.TOUCH_TX_FLAG) { //was a button pressed?
        touch_service();
        blink_tx = 5;

    }
    if (rx_tx_flag.MCP2200_RX_FLAG) { //was something received from USB?
        mcp2200_service();
        blink_tx = 5;
        blink_rx = 5;
    }
    if (rx_tx_flag.IRDA_RX_FLAG) { //was someting received from the PICtail?
        ir_service();
        blink_tx = 5;
        blink_rx = 5;
    }
    EDF_mTouchPoll(); //keep on polling the buttons
    switchTasks(); //Are we to hop out yet?
}
```

Figure 9: demo_main_loop code which checks each source of activity flag. In the meantime, poll the touch buttons and SW1 if loop is to be left.

If a flag is set, then the program counter will enter one of the service routines (touch, USB, or IR). They are all very similar and only differ in what is printed to the LCD and transmitted. Figure 10 shows the touch service routine.

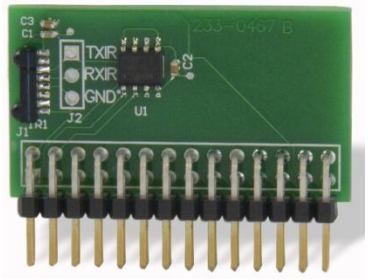
```
void touch_service() {

    mcp2200_tx(rx_byte.touch_tx); //send the button press byte to the PC
    irda_tx(rx_byte.touch_tx);   //send the button press byte to the IR link
    if (rx_byte.rx_tx_old_byte != rx_byte.touch_tx) { //Only enter here to refresh the LCD
        sprintf(LCDBuffer[0], "PC-RX:  IR-RX: ");
        sprintf(LCDBuffer[1], "PC-TX:%c IR-TX:%c", rx_byte.touch_tx, rx_byte.touch_tx);
        rx_byte.rx_tx_old_byte = rx_byte.touch_tx; //save the current byte for checking next time through
        EDF_INT_Disable(); //disable interrupts before updating or else risk failing the write
        LCDCommand(CLEAR_DISPLAY);
        LCDUpdate(); //write both LCD buffers
        EDF_INT_Enable(); //Enable again now that it is safe
    }
    rx_tx_flag.TOUCH_TX_FLAG = false; //Don't enter here again unless a touch button is pressed again
}
```

Figure 10: Enter here if touch button is pressed. Will always transmit every button press, but only update LCD if a new button is pressed

Because of the frequent interrupts, the code must disable interrupts when writing to the LCD or else it may fail to update or make the writes painfully slow to accomplish. This is also why 32MHz is chosen and it should not be changed without a great deal of tweaking for the bit-bang routines to work as well as other timing related events.

IR PICtail



The IrDA PICtail contains an MCP2122 which is a decoder/encoder for the TFDU6300 IR transceiver. The transceiver is classified as a fast infrared receiver (FIR) at 4Mbits/s with a tested range of about 6 feet. Its data quality relies heavily on the line of sight to another transceiver.

The PIC provides a clock to the MCP2122 which must be 16X the baud rate. For this example, the baud rate is fixed at 9600 and the CCP generates a 153.6KHz (9600*16) square wave with 50% duty cycle to its clock pin.

The PIC communicates to the MCP2122 via a 2 wire UART interface. Since the PIC16LF1939 only has one hardware UART that is being used by the MCP2200, it had to be bit-banged. According to the UART protocol , the start bit is active LOW, so TMR4 is used to poll the RX pin to check if it is LOW or not. If so, then a custom delay routine is entered to sample the pin 8 times for 8 bits to make a single byte. This routine is hardcoded for 32MHz at 9600 baud rate!

Because there is no hardware UART module connected to the IR link, the baud rate must be significantly reduced. 9600 is the highest tested, although it may perform at even higher. 9600 is a good default baud rate which handles the demo communication just fine.

Line code

The IrDA specification requires that a short, ~3us pulse be transmitted anytime the TX UART pin is LOW. When receiving, the IRDA-RX pin will go LOW for this same pulse period. Because of this, it is vital that the IR-TX pin is kept HIGH when not transmitting to avoid continually sending data. The complete data exchange can be explained by looking at Figure 11.

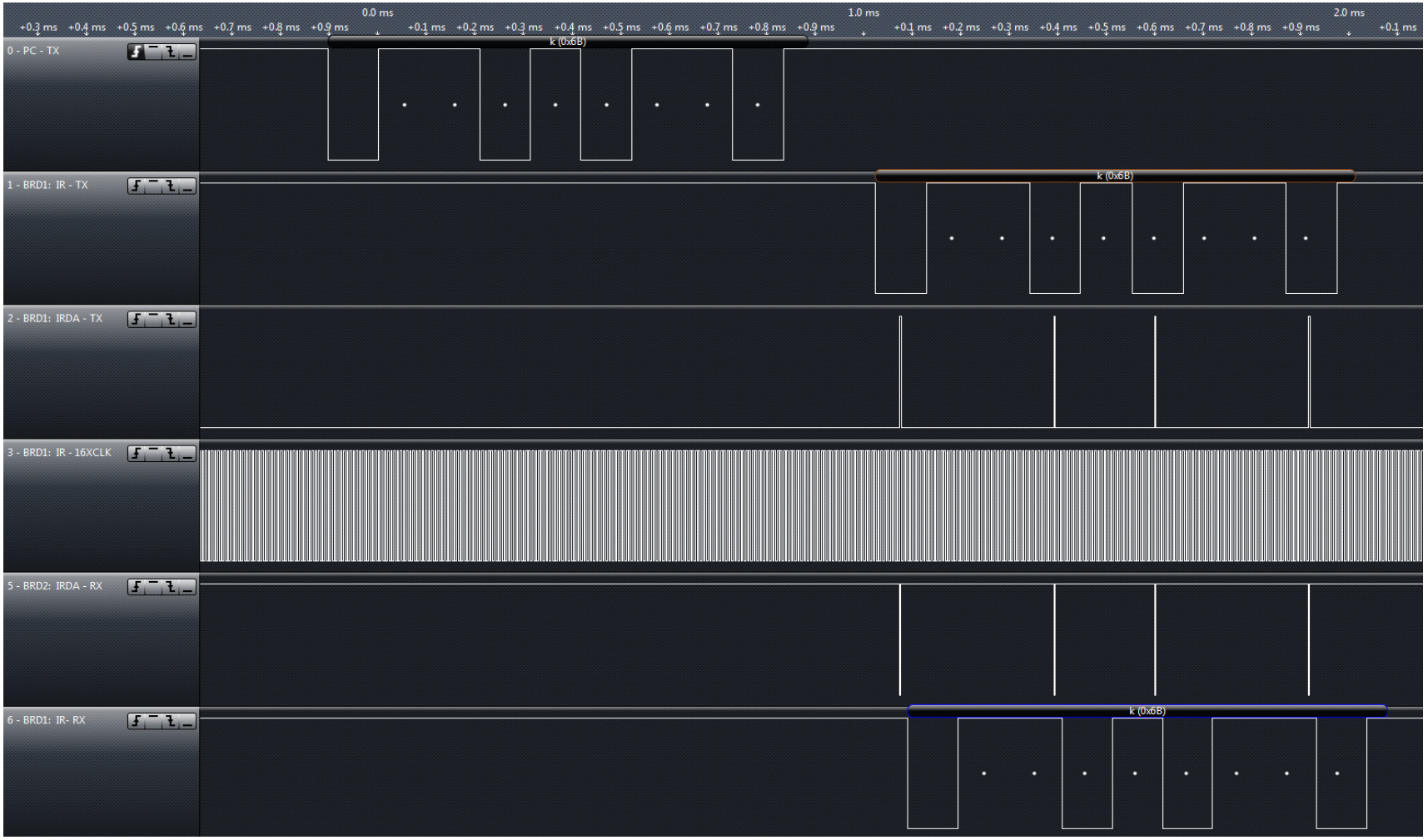
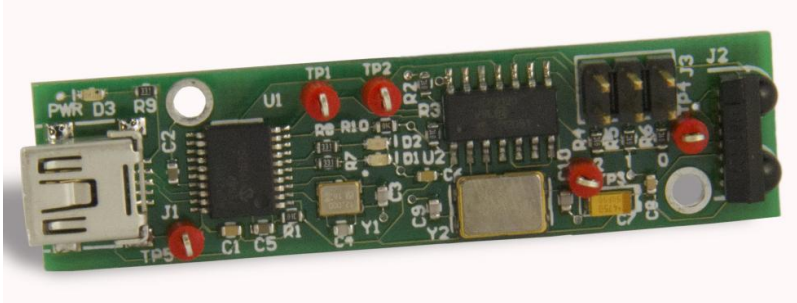


Figure 11: An IR exchange between two boards as seen in Figure 6. BRD1 is connected to the PC and BRD2 is powered by battery and receiving data from BRD1. Refer to Figure 7 for name notation

IR Dongle



This small board is used in conjunction with the EDF GUI. It consists of an MCP2200 as well as the MCP2120 encoder/decoder and TFDU4101 IR transceiver. The MCP2120 behaves like the MCP2122 with the exception that it has a crystal for the clock. There are three jumpers that can be used to change the baud rate. With none, it will default to 9600 which is what the GUI also defaults to.

Any received/transmitted byte will blink an LED. A green LED will indicate that power is applied. The baud rate can be changed in software through one of the GPIOs on the MCP2200, although this is not currently supported in the GUI for simplicity.

Appendix

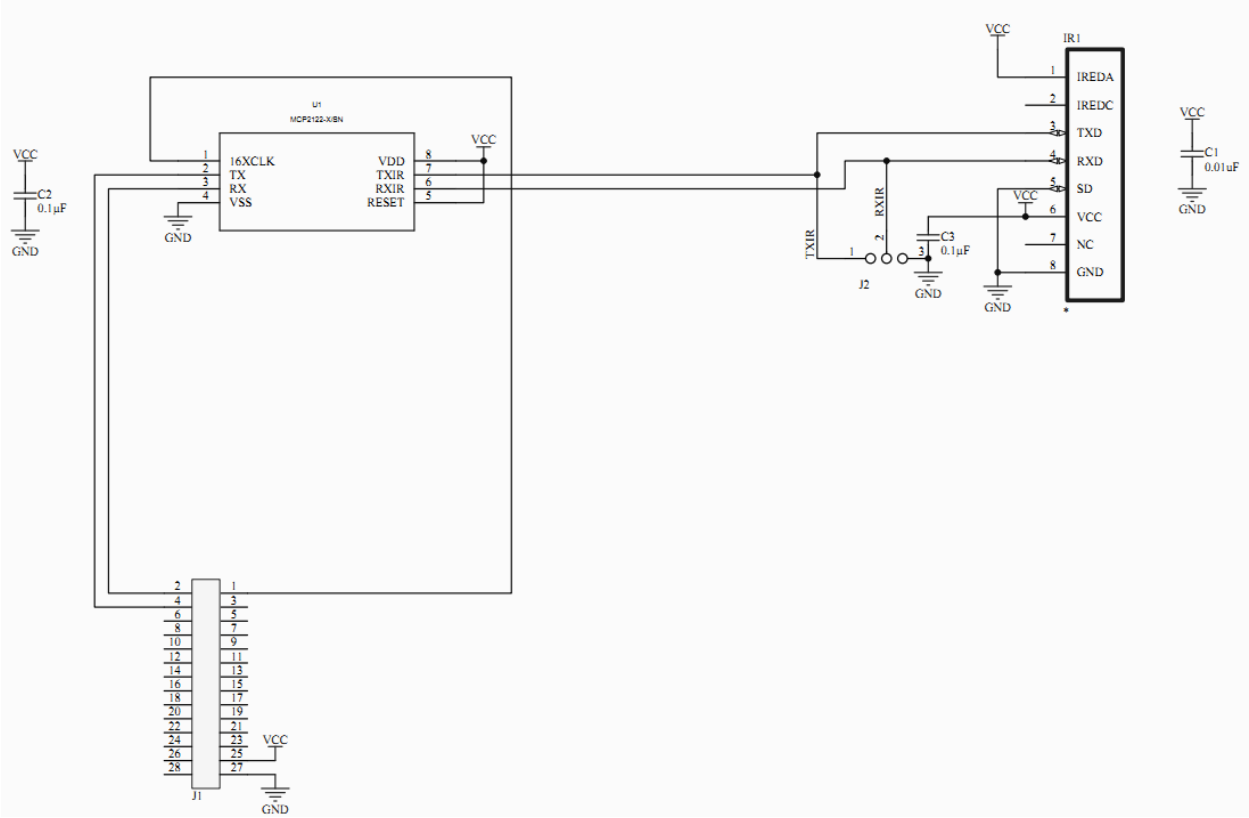
Change log:

```
2011-12-11    Justin Bauer  <justin.bauer@microchip.com>
*Windows GUI
      -updated to the released version
2011-11-17    Justin Bauer  <justin.bauer@microchip.com>
*Windows GUI
      -compatible with .NET Framework 2.0 and above
      -auto-populate dropdown COM port selection
      -Disabled changing default settings for sake of simplicity
*Doxygen
      -Added comments about existences of doxygen

2011-11-15  Justin Bauer  <justin.bauer@microchip.com>

    Initial revision
```

IR PICtail Schematic



IR Dongle Schematic