Development Kit
For the PIC® MCU
Exercise Book

RFID

September 2006

CCS Inc.

Custom Computer Services, Inc.
Brookfield, Wisconsin, USA
262-522-6500

Recognized Microchip Third-Party Tool Provider

Custom Computer Services, Inc. proudly supports the Microchip brand with highly optimized C compilers and embedded software development tools.

1 UNPACKING AND INSTALLATION

Inventory

❑ Use of this kit requires a PC with Windows 95, 98, ME, NT, 2000 or XP. The PC must have a spare 9-Pin Serial or USB port, a CD-ROM drive and 5 MB of disk space.

❑ The diagram on the following page shows each component in the PIC16F877A kit. Ensure every item is present.

Software

❑ Insert the CD into the computer and wait for the installation program to start. If your computer is not set up to auto-run CDs, then select **Start>Run** and enter **D:\SETUP1.EXE** where D: is the drive letter for your CD drive.

❑ Click on **Install** and use the default settings for all subsequent prompts by clicking NEXT, OK, CONTINUE…as required.

❑ Identify a directory to be used for the programs in this booklet. The install program will have created an empty directory **c:\program files\picc\projects** that may be used for this purpose.

❑ Select the compiler icon on the desktop. In the PCW IDE, click **Help>About** and verify a version number is shown for the IDE and PCM to ensure the software was installed properly. Exit the software.

Hardware

❑ Connect the PC to the ICD(6) using the USB cable.[1] Connect the prototyping board (11) to the ICD using the modular cable. Plug in the DC adaptor (12) to the power socket and plug it into the prototyping board (11). The first time the ICD-U40 is connected to the PC, Windows will detect new hardware. Install the ICD-U40 driver from the CD or website using the new hardware wizard. The driver needs to be installed properly before the device can be used.

❑ The LED should be dimly illuminated on the ICD-U to indicate the unit is connected properly.

❑ Run the following program: **Start>Programs>PIC-C>ICD.** If a communication error occurs, select a different COMM port until the ICD is discovered. See Chapter 3 for assistance.

❑ Select **Check COMM**, then **Test ICD**, then **Test Target**. If all tests pass, the hardware is installed properly.

❑ Disconnect the hardware until you are ready for Chapter 3. Always disconnect the power to the Prototyping board before connecting/disconnecting the ICD or changing the jumper wires to the Prototyping board.

[1] ICS-S40 can also be used in place of ICD-U40. Connect it to an available serial port on the PC using the 9 pin serial cable. There is no driver required for S40.
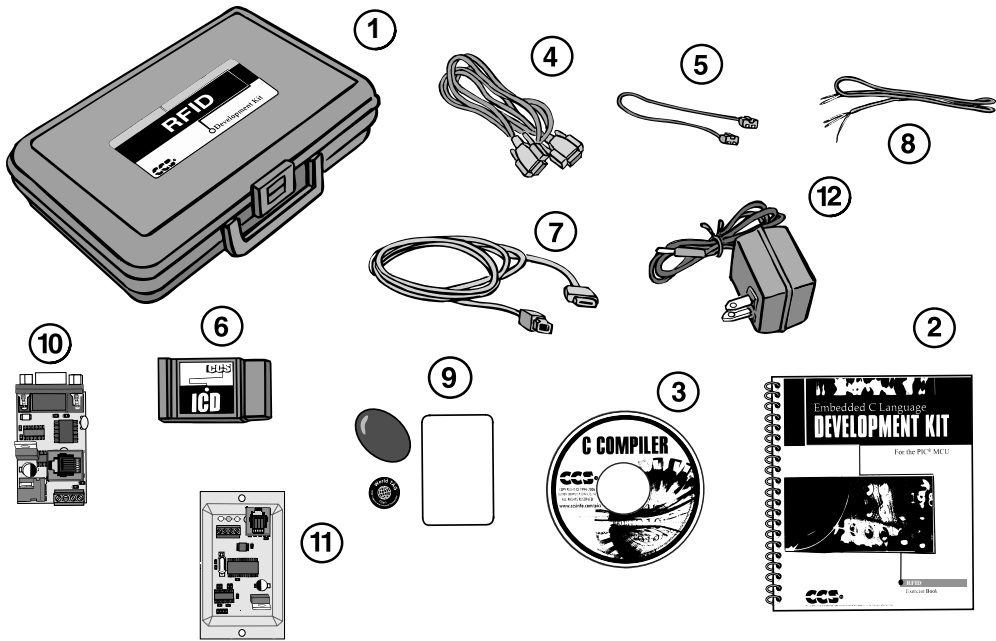
1. Carrying case
2. Exercise booklet
3. CD-ROM of the C compiler (optional)
4. Serial PC to Prototyping board cable
5. Modular ICD to Prototyping board cable
6. ICD unit for programming and debugging
7. USB (or Serial) PC to ICD cable
8. RFID board to adapter board cable
9. Transponders
10. RS-232 to RS-485 adapter board
11. RFID reader board
12. DC Adapter (9VDC)

2 USING THE INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

Editor

☐ Open the PCW IDE. If any files are open, click **File>Close All**

☐ Click **File>Open>Source File**. Select the file: **C:\Program Files\PICC\Examples\Ex_ stwt.c**

☐ Scroll down to the bottom of this file. Notice the editor displays comments, preprocessor directives and C keywords in different colors.

☐ Move the cursor over the **Set_timer0** and click. Press the F12 key. Notice a help file description for set_timer0 appears. The cursor may be placed on any keyword or built-in function and F12 will find help for that item.

☐ Review the editor's special functions by clicking on **Edit**. The IDE allows various standard cut, paste, and copy functions.

☐ Review the editor option settings by clicking on **Options**. The IDE allows selection of the tab size, editor colors, font and more. Click on **Options>Toolbar** to select which icons will appear on the toolbars.

Compiler

☐ Use the drop-down box under Compile to select the compiler. CCS offers different compilers for each family of Microchip parts. All the exercises in this booklet are for the PIC16F876A chip, a 14-bit opcode part. Make sure **PCM 14 bit** is selected in the drop-down box.

☐ The main program compiled is always shown in the bottom of the IDE. If the file you want to compile is not shown, then click on the tab of the file you want to compile. Right click in the editor and select **Make file project.**

☐ Click **Options>Project Options>Include Files…** and review the list of directories the compiler uses to search for included files. The install program should have put two directories in this list: devices and drivers.

☐ Normally the file formats need not be changed and global defines are not used in these exercises. To review these settings, click **Options>Project Options>Output Files** and **Options>Project Options>Global Defines.**

☐ Click **Compile>Compile,** F9, or the compile icon to compile a project. Notice the compilation box shows the files created and the amount of ROM and RAM used by this program. Press any key to remove the compilation box.

Viewer

❑ Click **Compile>Symbol Map**. This file shows how the RAM in the microcontroller is used. Identifiers that start with @ are compiler-generated variables. Notice some locations are used by more than one item. This is because those variables are not active at the same time.

❑ Click **Compile>C/ASM list.** This file shows the original C code and the assembly code generated for the C. Scroll down to the line:
```
int _ count=INTS _ PER _ SECOND;
```

❑ Notice there are two assembly instructions generated. The first loads 4C into the W register. INTS_PER_SECOND is #defined in the file to 76. 4C hex is 76 decimal. The second instruction moves W into a memory. Switch to the Symbol Map to find the memory location where int_count is located.

❑ Click **View>Data Sheet**, then OK. This brings up the Microchip data sheet for the microprocessor being used in the current project.

Click here for the file menu. Files and Projects are created, opened, or closed using this menu.

Place cursor over each icon and press F1 for help.

Place cursor here for slide out boxes. slide out boxes. All of the current project's source and output files can be seen here.

Click the help icon for the help menu. The technical support wizard and download manager are accessed using this menu.

Compile ribbon.
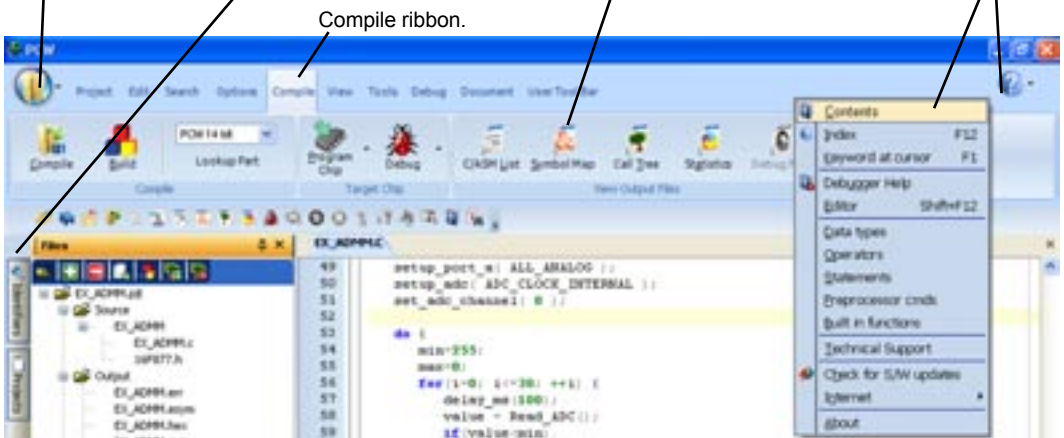
# 3 COMPILING AND RUNNING A PROGRAM

❑ Open the PCW IDE. If there are any files open , **click File>Close All.**

❑ Click **File>New** and enter in the filename **EX3.C.**

❑ Enter in the following source code then **Compile.**

```c
#include <16F876A.h>
#fuses HS,NOWDT,NOPROTECT,NOLVP,NOBROWNOUT,PUT
#use delay(clock=20000000)

#define GREEN_LED    PIN_C3
#define YELLOW_LED   PIN_C4
#define RED_LED      PIN_C5

void main() {
    while(TRUE) {
        output_low(GREEN_LED);
        delay_ms(1000);
        output_high(GREEN_LED);
        delay_ms(1000);
    }
}
```

NOTES

● The first three lines of the source code define the hardware environment. The microcontroller being used is the PIC16F876A running at 20MHz. Click on **View>Valid Fuses** to read about the different fuse settings. Fuses control the microcontroller's configuration word.

● The #define is used to enhance the readability by referring to GREEN_LED in the program instead of PIN_C3.

● The statement while(TRUE) is a simple way to create a loop that never stops. Infinite loops are very common in main for embedded systems.

● The statement delay_ms(1000); is a one second delay (1000 milliseconds).

❑ Connect the ICD to the Prototyping board using the modular cable, and connect the ICD to the PC using the 9-pin serial cable for ICD-S or the USB cable for ICD-U. Power up the Prototyping board. Verify the LED on the ICD is dimly illuminated. If using ICD-S40 and the COMM port box does not list the port, check that no other programs have opened that port.

❑ Click on **Tools>ICD** to download the program to the RFID board. Once completed, the green LED should flash, one second on and one second off.

❑ Highlight everything above void main(). Click **Edit>Paste to file**. Name the file rfid.h. This header file is used in the remaining example programs.
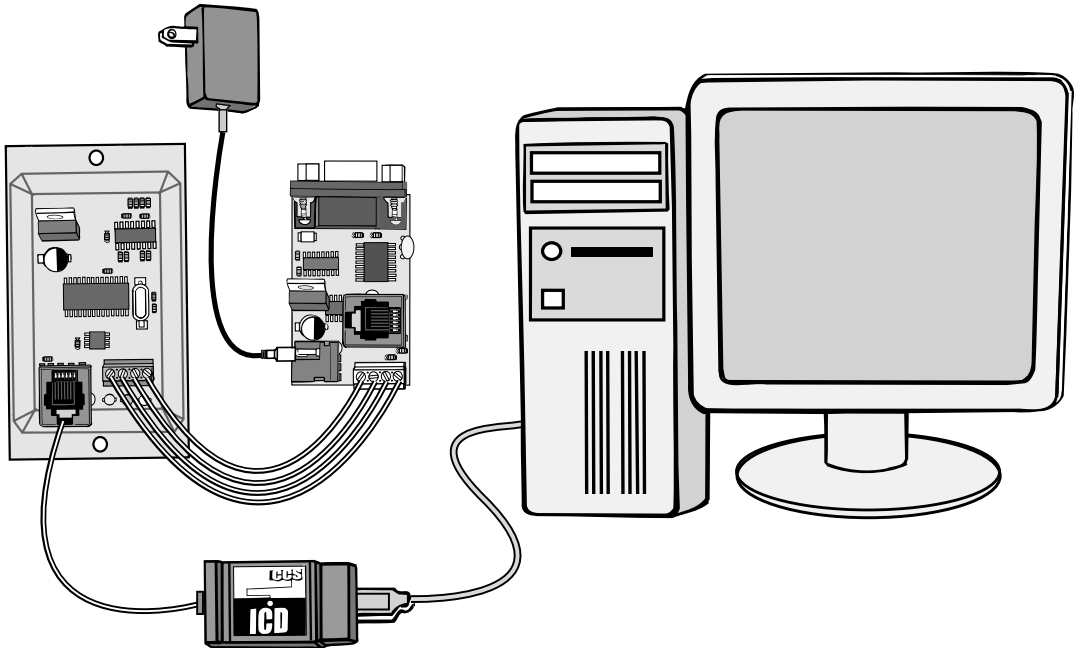
4 DEBUGGING

☐ Open rfid.h and insert **`#device ICD=TRUE`** on the line after #include <16F876A.h> to compile in debug mode.

☐ Create ex4.c, type in the following source code, right click in the editor and select **Make file project**, then compile.

```
#include "rfid.h"

int8 sum(int8 a, int8 b) {
    return a+b;
}

void main() {
    int8 x = 2, y = 3;

    while(TRUE) {
        x = sum(x, y);
    }
}
```

☐ Start the debugger by clicking **Debug>Enable Debugger**. After the program is loaded onto the RFID board, click the step-over icon 🐾 until the yellow arrow passes x = sum(x, y). Each click causes a line of code to be executed. Clicking the step-over icon 🐾 on x = sum(x, y) causes the entire function to be executed in one click.

☐ Click the single step icon 🐾 a few times. The arrow should point at return a+b. The single step icon 🐾 causes the debugger to step into the function. Press the single step icon 🐾 a couple more times to return to main.

☐ Click the **Watches** tab, then click the add icon 🔲 to add a watch. Enter x or choose x from the list of variables and click Add Watch. The current value of x is shown. Continue to press the step-over 🐾 and single step 🐾 icons to see the value of x change. Notice how the value of x is not displayed when it is inside the sum() function because it is not available in the source code at this time.

❑ Click the go icon 🔵 to allow the program to run normally. Click the stop icon 🔴 to halt execution. The debugger arrow will point to where the program was halted.

❑ In the editor, click on return a+b to move the cursor to that line. Click the **Breaks** tab and click the add icon 🔳 to set a breakpoint. The program will be halted every time this line of code is reached. Click the go icon 🔵. The debugger will stop at the breakpoint. Practice setting breakpoints at different locations to learn how they work.

❑ Click **Compile>C/ASM** List. Find the line with the debugger arrow. Notice one assembly instruction was already executed and the arrow has passed the breakpoint. This is a side effect of the debugger. Sometimes breakpoints slip by one ASM instruction.

❑ Click the step-over 🐾 and single step 🐾 icons a few times. Notice that the debugger is stepping through one assembly instruction per click, instead of one entire C line.

❑ Change return a+b to return a-b and recompile. Step over the call to sum and examine the value of x. The int data type by default is not signed, so x cannot be the expected –1. The modular arithmetic works like a car odometer in reverse, only in binary. For example, 00000001 minus 1 is 00000000; subtract another 1 to get 11111111, or decimal 255.

❑ Press the reset button ⊙ and step up to x = sum(x, y). Click the **Eval** tab. This pane allows a one-time expression evaluation. Type in x+y and click **Eval** to see the debugger calculate the result. The complete expression may also be put in the watches pane. Now enter y=1 and click **Eval**. If the "Keep side effects" checkbox is checked, this expression will change the value of y. Check "Keep side effects" and click **Eval** again. Click the **Watches** tab. Then step over the call to sum to verify that the value of x was calculated with the new value of y.

❑ Set a break point at x = sum(x, y) then click the **Break Log** tab. Check the Log checkbox, make sure break 1 is selected, and enter x in the edit box. Press the go icon 🔵. Each time the breakpoint is reached, the debugger will retrieve the value of x, add it to the log, and continue execution.

❑ Remove #device ICD=TRUE from rfid.h before continuing the remaining exercises.

5 ◆ RS-232 AND RS-485

❑ RS-232 is a popular point-to-point, asynchronous communication protocol used on most PCs and many embedded systems. Two signal wires transmit and receive data while a third ground wire is used for reference voltage. Both microcontollers included in the kit have built-in hardware to buffer serial data. The 16F876A uses C6 for transmitting and C7 for receiving while the 16F627A uses B2 and B1. The compiler is able to use any pin, but will take advantage of the built-in hardware when available.

❑ The following line of code includes RS-232 support into the RS-232 to RS-485 adapter:

```
#use rs232(baud=9600, xmit=PIN _ B2, rcv=PIN _ B1)
```

❑ RS-232 sends a series of bits at the hardware level. The baud= option specifies how many bits are sent per second. The bit stream, as specified above, is a start bit (always 0), 8 data bits (lsb first) and a stop bit (always 1). The data line then remains at the logic 1 level. The number of bits may be changed with a bits= option. A 0 is represented as a positive voltage (+3V to +12V) and a 1 is represented as a negative voltage (-3V to –12V). Since the microcontroller outputs only 0V and 5V, a level converter is required to interface to standard RS-232 devices such as a PC. A popular converter chip is the MAX232. See the schematic on the back cover for details.

❑ RS-485 is also an asynchronous communication protocol, but it differs from RS-232 in many ways. Primarily, it allows for multiple points to be connected to the same signal wires. In half duplex or b-directional mode, one twisted pair is used for both transmitting and receiving. To send a bit over the network, one of the wires is set to a high state and the other is set to a low state over a –7V to +12V range. The receiving device subtracts line voltages. If the difference is greater than 200mV, a bit has been sent. When a device is not transmitting, it set its outputs for high impedance. The voltage difference across the twisted pair is then less than 200mV. RS-485 transmissions are achievable over longer distances than RS-232 due to the twisted pair usage. If noise is incurred, both wires will be affected because the voltage difference is still detectable. The twisted pair also provides noise cancellation to prevent emitting interference to other nearby communication wires.

❑ There is a variety of software methods for handling RS-485 communication. The drivers included with the CCS compiler use carrier detection. Before attempting to send a message, the bus is checked for activity. If signals are present, the device waits until the voltage levels are in the idle state.

❑ Each device on the network is assigned a unique address. When sending a message, the address and message size are included with a parity check to ensure integrity. A device only accepts messages sent to its specific address.

- [ ] The RS-232 to RS-485 adapter board, included in the kit, comes preprogrammed with a conversion program. Any message it receives from RS-485 is sent to the PC's RS-232 port. Characters from the PC are sent to address 0x11. See ex_RS232_485.c in the examples directory for the source code.

- [ ] The following example is for the RFID board. It gets characters over the RS-485 bus from the adapter board to change the color of its bicolor LED. Create a new file called ex5.c and enter the following source code:

```
#include "rfid.h"

#define  RS485_ID             0x11
#define  RS485_USE_EXT_INT    FALSE
#define  ADAPTER_RS485_ID     0x7F
#include <rs485.c>

int8 msg[32];

typedef enum {OFF, GREEN, RED} LEDcolor;

void twoColorLED(LEDcolor color) {
    switch(color) {
        case OFF:
            output_low(PIN_A3);
            output_low(PIN_A5);
            break;
        case GREEN:
            output_high(PIN_A3);
            output_low(PIN_A5);
            break;
        case RED:
            output_low(PIN_A3);
            output_high(PIN_A5);
            break;
    }
}
void RS485send(char* s) {
    int8 size;
    for(size=0; s[size]!='\0'; ++size);      // Find message size
    rs485_wait_for_bus(FALSE);
    while(!rs485_send_message(ADAPTER_RS485_ID, size, s)) {
        delay_ms(RS485_ID);
    }
}
(continued...)
```

```
(continued...)

char RS485getc() {
    rs485_get_message(msg, TRUE);
    return msg[2];
}

void main() {
    output_low(GREEN_LED);    // Show power is on
    rs485_init();

    sprintf(msg, "(O)ff, (G)reen, or (R)ed\n\r");
    RS485send(msg);

    while(TRUE) {
        switch(toupper(RS485getc())) {
            case 'O': twoColorLED(OFF);    break;
            case 'G': twoColorLED(GREEN); break;
            case 'R': twoColorLED(RED);    break;
        }
    }
}
```

❑ Connect the 9-pin serial cable, to RS-232; to the RS-485 adapter board, and to the PC. Click **Tools>Serial Port Monitor** within the PCW IDE. Configure the COMM port by clicking **Configuration>Set port options**. Set the baud rate to 9600, parity to none, data bits to 8, stop bits to 1, and flow control to none. Make sure the correct COMM port is selected.

❑ Compile the program and download it to the RFID board. Once the program is running, , a message will display in SIOW,

❑ Click on SIOW and press O, G, or R to change the color of the LED.

❑ Before moving on to the next exercise, open the rfid.h file and add the three #define statements from the ex5.c program. The enum and functions from ex5.c are likely to be reused in future programs. In order to use this code in other sources, highlight lines from ➡ start point to ➡ endpoint. Click **Edit>Paste to File**. Name the file utilities.c and click **Save**.
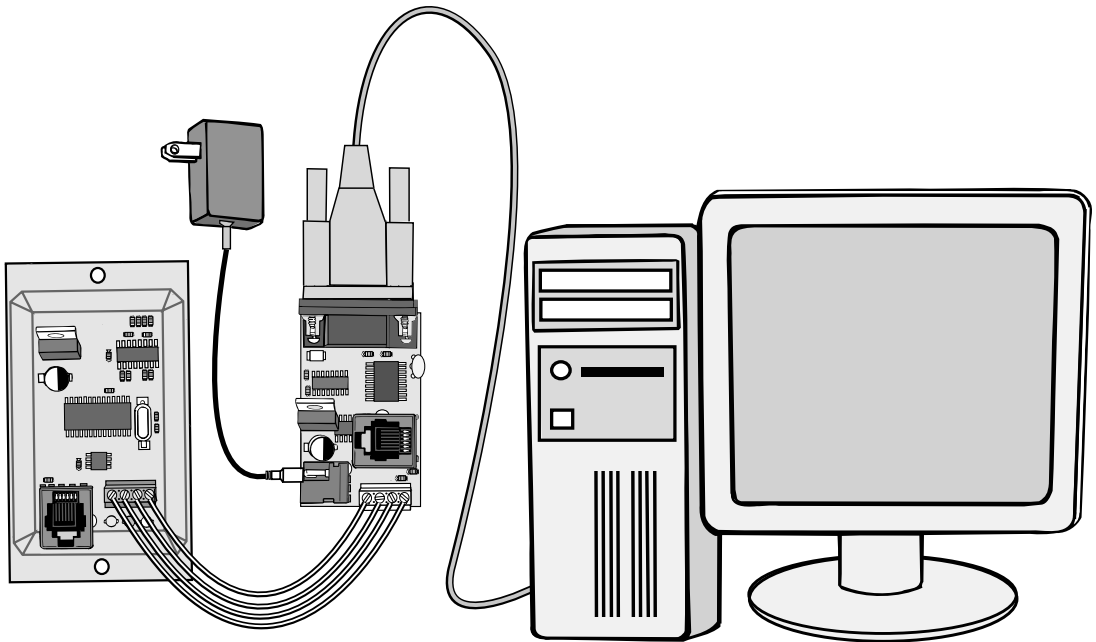
RFID TECHNOLOGY

Although Radio Frequency Identification (RFID) technology is a few decades old, it has become more widely used in recent years due to diminishing cost restraints. Previously, it was not feasible to attach expensive, disposable transponders to consumer products. They were mainly utilized in situations where they could be reused, such as assembly lines. The basic RFID system is composed of three parts: transponders, antennas, and controllers.

Transponders

A transponder consists of an Integrated Circuit (IC) and an antenna. The IC can range from the size of a fingernail to a flake of pepper. Antennas are usually made from a coil of wire surrounding the IC.

There are two different approaches to powering transponders. An active transponder contains its own power source for retaining information and communication. A passive transponder relies on being powered through its antenna coil by the carrier frequency of the radio signal. Waves power the transponder either through backscatter or inductive coupling. The system in this kit uses inductive coupling. When the oscillating radio waves match the oscillating frequency of the transponder's antenna coil circuit, enough voltage is induced to power its IC and transmit signals. Passive devices normally contain an EEPROM to preserve their information while not powered. These transponders do not require maintenance because there is no battery to replace.

Transponders come in a few package types. The package included in the kit is called a tag. The other package types are labels and printed circuit boards (PCB). Labels are ideal for mass production, low cost applications in a tame environment. Tags can be used in a wider variety of situations because they are durable against heat, chemicals, dirt, and water. PCBs are normally utilized in permanent installations, such as pallet tracking, where they are reused and more information needs to be stored.
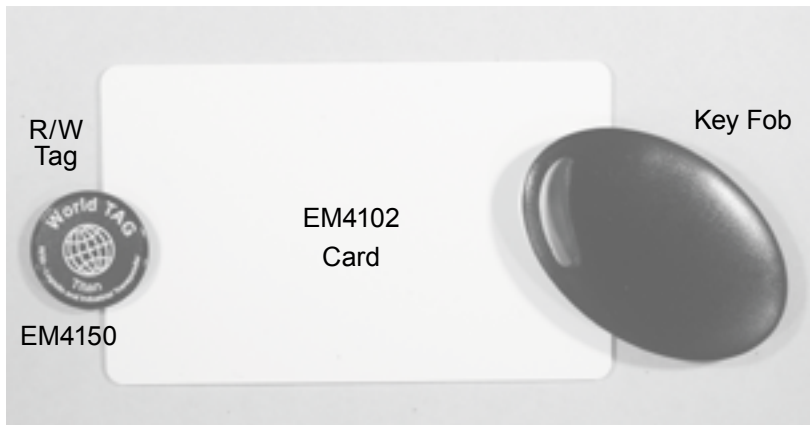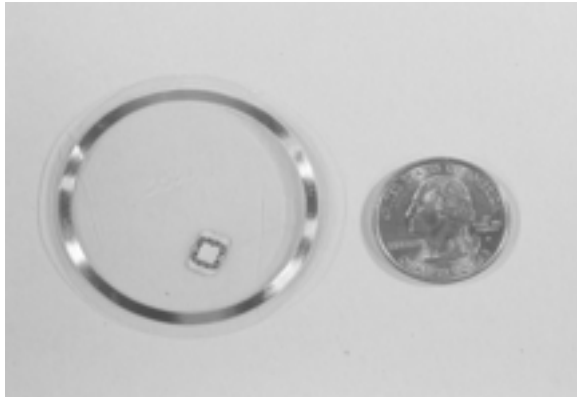
Transponders also come in read-only or read/write models. The read-only type usually has less storage available. They are useful for tracking items or animals. Since there are so many IDs available with a small number of bits, it is easy to uniquely identify every item with its own ID. For example, there are over 4 billion different ID numbers in 32 bits. Read/write transponders are useful for small remote databases and more intelligent tracking systems. Each item can have every location written to its transponder. This allows service technicians and customer support to handle problems more accurately. They also help control manufacturing processes by writing specific assembly instructions and quality control information on each part. Sometimes it may be necessary to protect the data to prevent unwanted reads. Some transponders offer read, write, and password protection, while others encrypt data to prevent a different reader from intercepting a transmission and decoding it.

This kit uses parts from EM Microelectronics. Transponders available include:

Part #	Read-Only	Read/Write	Rectifier	Contactless	Anti-Collision	Encryption
Card EM4102	X		X			
EM4100	X			X		
EM4056		X		X	X	
EM4450/4550		X		X		
EM4025/4125	X				X	
EM4055		X		X	X	
EM4083				X		
EM4469		X				
EM4170		X		X		X

Antennas

The antenna is a very important part of designing an RFID system. The size, shape, and quality of the antenna directly influence communication distance and circuit board design. Generally, as the size of the antenna increases, so does the reading distance. Similarly, a transponder with a larger antenna is readable at a greater distance. The driver circuit designed to operate at the resonant frequency depends on the antenna's attributes.

There are two things to remember when designing an antenna. First, field strength for the small loop antennas decreases proportionately to $1/r^3$ where r is the distance from the coil. This means that the effectiveness of an antenna falls quickly over a greater distance and there will be a point when transponders are suddenly unreadable. Second, high powered antennas are not always a solution to reading distance. The FCC has regulations and ICs can handle only so much antenna voltage.

Controllers

A controller is the device in charge of handling all the communication. It drives the resonating frequency of the antenna and monitors signal fluctuations to detect nearby transponders. Most controllers are also connected to an external device with a different communication protocol, such as RS-232 or RS-485, to pass along information.

When an antenna and a controller are combined into one unit, a reader is created. The RFID board included in the kit is an example of a reader. Readers can operate over multiple frequency spectrums; among the most common are 50 to 500kHz, 13.56mHz, and 0.9 to 2.5GHz. Each spectrum has its benefits and downfalls. The low frequency RFID systems are very reliable and allow for cheap, low speed systems.

They are perfect for access control and similar situations where slow data transfer rates over short ranges are acceptable.

There are a few different ways for an RFID system to modulate radio signals. FM changes the carrier frequency, PM changes the phase, and AM changes the amplitude. Some techniques using these modulations are frequency shift key (FSK), phase shift key (PSK), and amplitude shift key (ASK). The technique used by the RFID board in the kit is a special type of ASK known as on/off key (OOK), which completely modulates the antenna voltage to send a '0'. Data is transmitted to the reader using Manchester encoding. The carrier frequency also acts as a clock for synchronous communication. This protocol is described in detail in Chapter eight, Read/Write Transponder.

Benefits

There are many benefits to using an RFID system over other predominant methods. Barcodes require optical readers, which require maintenance, are more prone to damage, and do not work well in direct sunlight. Barcodes themselves cannot contain nearly as much information as a transponder and require a line of sight to the reader. Magnetic strip cards wear out and need to be replaced and their readers have mechanical parts that also require periodic service. Transponders only need to be within the reading range and are readable even when placed inside packages. They also have a higher reading success rate. For these reasons, many companies are switching to RFID for logistics and inventory control.

❑ The EM4102 transponder included in the kit is a read-only device. It contains 64 bits of information consisting of a 9-bit header, 40 data bits (D), 10 row-parity bits (P), 4 column-parity bits (C), and one stop bit. When the transponder coil is close enough to a 125kHz RF signal to power the IC, it will continuously send the 64 bit stream, one bit every 64 RF periods. It starts by sending nine 1s to create a header, then a pattern of four data bits followed by one row-parity bit. The last five bits are column-parity and a 0 for the stop bit. Of the 40 bits of data, the first eight are version bits or a customer ID number. The remaining 32 bits offer over 4 billion unique tag ID numbers.

1	1	1	1	1	1	1	1	1
				D	D	D	D	P
				D	D	D	D	P
				D	D	D	D	P
				D	D	D	D	P
				D	D	D	D	P
				D	D	D	D	P
				D	D	D	D	P
				D	D	D	D	P
				D	D	D	D	P
				D	D	D	D	P
				C	C	C	C	0

❑ The example program for the RFID board continuously searches for data from an EM4102 transponder and sends it to the adapter board. The function read_4102() expects a pointer to a 5 byte array. The following diagram depicts how read_4102() stores data in the array.

Customer ID	Tag ID	Tag ID	Tag ID	Tag ID

❑ The first byte contains the customer ID, which is followed by the 32 tag ID bits. The most significant tag ID bit is not defined in the data sheet for the EM4102. This means the bytes can be converted to a 32 bit number in any order. Just be sure to convert in the same way every time.

❑ Open the rfid.h file created in Chapter 3 and insert **#device *=16** on the line after #include <16F876A.h>. The compiler will use 16-bit RAM addresses, thereby allowing more memory to be used.

❑ Create a file called ex7.c and type in the following source code:

```
#include "rfid.h"
#include <em4095.c>          // Controls the reader IC
#include <em4102.c>          // Allows reading 4102 transponders
#include <rs485.c>

int8 msg[32];
#include "utilities.c"

void main() {
    int8  customerCode;
    int32 tagNum;

    rf_init();              // Initialize the RF reader
    rf_powerUp();           // Power up the antenna
    rs485_init();           // Initialize RS485 communication
    output_low(GREEN_LED);  // Show the board is powered and ready

    for(;;) {
        if(read_4102(msg)) {
            customerCode = msg[0];
            tagNum = make32(msg[1], msg[2], msg[3], msg[4]);

            sprintf(msg, "Customer Code: %u\n\r", customerCode);
            RS485send(msg);
            sprintf(msg, "Tag Number: %lu\n\n\r", tagNum);
            RS485send(msg);
        }
    }
}
```

❑ Compile the program and download it to the RFID board. Open SIOW and configure it in the same way as in Chapter 5. Take an EM4102 transponder and place it near the RFID board. The customer code and tag ID number should be displayed in the terminal window. Rotate the transponder from parallel to perpendicular relative to the antenna and compare reading distances. The reader should be able to find a parallel transponder about three inches away.

❑ Write down the Customer Code and the Tag Number.

❑ Append the following two functions to utilities.c:

```
int8  RS485getInt() {
    int8 i, s[5];
    for(i=0; (s[i]=RS485getc()) != '\r' && i<5; ++i);
    return atoi(s);
}

int32 RS485getI32() {
    int8 i, s[11];
    for(i=0; (s[i]=RS485getc()) != '\r' && i<11; ++i);
    return atoi32(s);
}
```

❑ Create ex8.c and type in the following source code:

```
#include<..\rfid.h>
#include<em4095.c>          //Controls the reader IC
#include<em4102.c>          //Allows reading 4102 transponders
#include<rs485.c>
#include<stdlib.h>

int8 msg[32];
#include "..\utilities.c"

void main()
{
      int8 customerCode, code;
      int32 tagNum, tag_ID;

      rf_init();               //Initialize the RF reader
      rf_powerUp();            //Power up the antenna
      rs485_init();            //Initialize RS485 commuication
      output_low(YELLOW_LED);  //Show program is running

      sprintf(msg, "Enter the customer code: ");
      RS485send(msg);
      code = RS485getInt();

(Continued...)
```

```
(Continued)
    sprintf(msg, "\n\n\r");
    RS485send(msg);

    sprintf(msg, "Enter the tag ID: ");
    RS485send(msg);
    tag_ID = RS485getI32();

    sprintf(msg, "\n\n\rScanning...");
    RS485send(msg);

    for(;;)
    {
        if(read_4102(msg)) {
            customerCode = msg[0];
            tagNum = make32(msg[1], msg[2], msg[3], msg[4]);

            if(customerCode == code && tagNum == tag_ID) {
                output_high(RED_LED);
                output_low(GREEN_LED); //Light green LED if match made
            } else {
                output_high(GREEN_LED);
                output_low(RED_LED);   //Light red LED if no match made
            }


            delay_ms(2000);            //Wait 2s and then turn off LEDs
            output_high(GREEN_LED);
            output_high(RED_LED);
        }
    }
}
```

❑ Compile the program and download it to the RFID board. When prompted, enter the customer code and tag ID number written down in the last exercise of one of the transponders. Then take a turn holding each EM4102 transponder over the antenna. When the program finds a transponder matching the data entered, it will light the green LED for two seconds. If a non-matching transponder is held over the antenna, the red LED will be lit for two seconds.

❑ This behavior is an example of how RFID technology can be applied. If the system was expanded it could easily be modified to serve as an access control system. Instead of just lighting a LED, the system could also unlock a door for the transponders that it recognizes. RFID can also be used for inventory control systems. These systems use antennas with a long range that activate an alarm when they recognize if a product that is part of the inventory has not been sold or is scheduled to move.

❑ The second type of transponder included in the kit is the EM4150. Through a variety of commands, its internal EEPROM can be read, written, password-protected, read-protected, and write-protected. It holds 1024 bits arranged in 32 words of 32 bits. Three of the words are reserved:
- • The first word contains the login password.
- • The second word controls read/write protection.
- • Broadcasted data and password protection enabling are configured in the third word.

Together, these three words customize the information available to transponder readers. In addition to the 1024 bits of EEPROM, there are two laser burned words. A device serial and identification number are located at addresses 32 and 33, respectively.

❑ Communication packets on the EM4150 are organized into a similar row and column structure. A total of 45 bits are used to send 32 bits of data. Eight data bits are followed by one parity bit, a row of column bits, and a stop bit. Reading and writing use the same structure. The following figure details a packet of data.

| D0	D1	D2	D3	D4	D5	D6	D7	P0
D8	D9	D10	D11	D12	D13	D14	D15	P1
D16	D17	D18	D19	D20	D21	D22	D23	P2
D24	D25	D26	D27	D28	D29	D30	D31	P3
C0	C1	C2	C3	C4	C5	C6	C7	0

❑ When the EM4150 becomes powered by the RF field, it begins to broadcast words in the format described above. Before transmitting each word, it sends two listen windows. The listen window pattern consists of two half bit period pulses, a two- bit period pulse, and two one-bit period pulses. The graph below shows two listen windows. One listen window is between the vertical dashed lines.
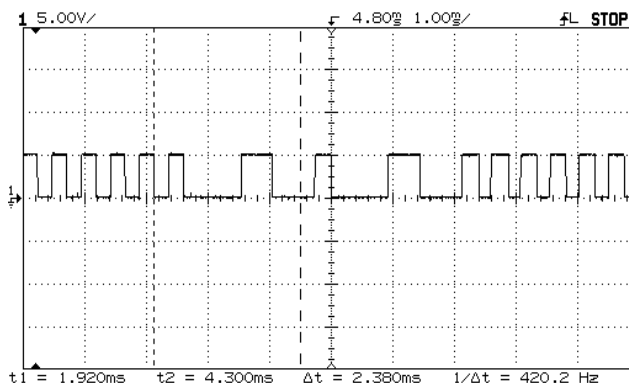
❑ When the microcontroller wants to send a command, it searches for the beginning part of the pattern with the long low pulse. Then it starts transmitting during the following bit period where the demodulated signal is held high. The transponder detects the modulation and stops broadcasting data.

❑ Each bit of data is sent via the OOK antenna modulation. When the antenna is modulated, it stops sending the carrier signal. A bit value of '0' is sent by modulating the antenna for the first half of a bit period. No modulation occurs during the second half, so the signal returns to its normal operating voltages to keep the transponder powered. A '1' is sent with no modulation for either half. The following graph shows the antenna voltages while sending 00100000001, the reset command, by modulating the signal amplitude. The EM4150 transponders included in the kit have a bit period of 64 RF periods. Each modulation pulse consists of half a bit period or 32 RF periods.
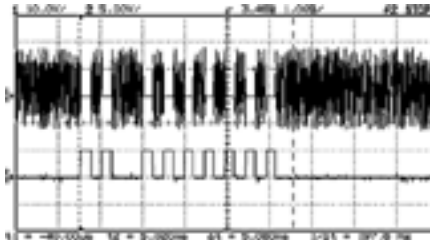
❑ The graph on the next page illustrates a complete communication cycle. The demodulated signal is on top and the antenna modulation control is on the bottom. Appearing on the top left side is the long low pulse of the listen window. After detecting this pulse, two zeros were sent during the high one-bit period to put the transponder into listen mode. The next eight bits are the command, which are followed by a parity bit. The send time is located between the two vertical dashed lines. Notice the extra time at the end when a '1' was sent for parity. To complete the process, the transponder validates the command and sends an acknowledgment (ACK). The reader detects the emitted pattern to confirm a successful transmission. The ACK pattern begins within the second low pulse after the second vertical line and ends with the last high pulse.
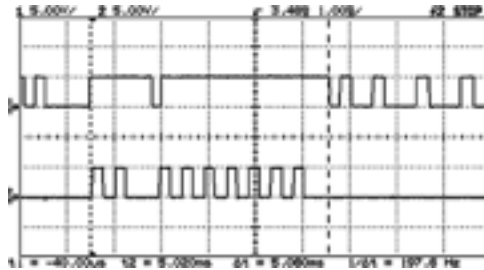
❑ Create ex9.c and type in the following source code:

```
#include <rfid.h>
#include <rs485.c>
#include <em4095.c>
#include <em4150.c>
#include <stdlib.h>

int8 msg[32];
#include "utilities.c"
void main() {
    int8  err;
    int32 temp;
        rs485_init();
    rf_init();                  // Initialize the RF reader
    output_low(GREEN_LED);      // Show the board is powered and ready
for(;;) {
        sprintf(msg, "\n\n\rR,W: ");     // Choose read or write
        RS485send(msg);

(continued...)
```

```
(continued...)
        switch(toupper(RS485getc())) {
        case 'R':
            sprintf(msg, "\n\rAddress (0-33): "); // Get an address
        RS485send(msg);
            if((err = read_4150(msg, RS485getInt())) == ERR_OK) {
                temp = make32(msg[3], msg[2], msg[1], msg[0]);
                sprintf(msg, "\n\rData: %lu", temp);
                RS485send(msg);
        }
            break;
        case 'W':
            sprintf(msg, "\n\rAddress (0-33): "); // Get an address
            RS485send(msg);
            temp = RS485getInt();
            sprintf(msg, "\n\rData: ");      // Get data to write
            RS485send(msg);
            err = write_4150(RS485getI32(), temp);
            break;
        default: continue;
    }
    switch(err) {
        case ERR_OK:        sprintf(msg, "\n\rOK");        break;
        case ERR_LIW:       sprintf(msg, "\n\rLIW");       break;
        case ERR_NAK:       sprintf(msg, "\n\rNAK");       break;
        case ERR_PARITY:    sprintf(msg, "\n\rPARITY");    break;
    }
        RS485send(msg);
    }
}
```

❑ Compile the program and download it to the RFID board. Once running, place an EM4150 transponder near the antenna and test the program. The characters sent to the RFID board will not be displayed without local echo enabled in SIOW. Since the transponder is new, all EEPROM should first contain 0. Try reading addresses 32 and 33 to see the device serial and identification numbers.

❑ The last switch statement examines the error code returned by the transponder functions. Different error codes are available to let software know what is causing a problem and handle it as necessary.

❑ As mentioned in the previous chapter, the read/write transponder included in the kit offers read, write, and password protection. Create a file called **EX10.C**, copy over all the source code from **EX8.C**, and add the following input options:

```
case 'L':
    sprintf(msg, "\n\rPassword: ");      // Get login password
    RS485send(msg);
    err = login_4150(RS485getI32());
    break;
case 'C':
    sprintf(msg, "\n\rOld PW: ");        // Get old password
    RS485send(msg);
    temp = RS485getI32();
    sprintf(msg, "\n\rNew PW: ");        // Get new password
    RS485send(msg);
    err = setPassword_4150(temp, RS485getI32());
    break;
case 'P':
    sprintf(msg, "\n\rFirst: ");         // First read protected word
    RS485send(msg);
    temp = RS485getInt();
    sprintf(msg, "\n\rLast: ");          // Last read protected word
    RS485send(msg);
    err = readProtect_4150(temp, RS485getInt());
    break;
case 'E': err = PWprotect_4150(TRUE);    break;
case 'D': err = PWprotect_4150(FALSE);   break;
case 'T': err = reset_4150();            break;
```

❑ Add the following cases to the error-checking switch statement:

```
case ERR_NAK_OLDPW: sprintf(msg, "\n\rNAK OPW"); break;
case ERR_NAK_NEWPW: sprintf(msg, "\n\rNAK NPW"); break;
case ERR_LIW_NEWPW: sprintf(msg, "\n\rLIW NPW"); break;
```

- ❏ Change sprintf (msg, "\n\n\rR,W: "); to include L,C,P,E,D,T.

- ❏ Compile the program and download it to the controller board. Place an EM4150 transponder near the antenna and press 'L' to login. The default password is zero. Once a successful login has occurred, the read-protected region may be modified. Press 'P' and read protect five through ten. Write some data to address seven and read it back. Move the transponder away from the antenna so it loses power, then return it to the RF field. A read to address seven will return zero. Write some data to address six and read it back. It will also read zero. Log back in, and read addresses six and seven again. This demonstration shows how read protection works. It allows a write to occur anytime, but data can only be read after a successful login. Write-protection works a little differently. In order to change the data in a write-protected word, write-protection must first be removed from that address.

- ❏ While logged in, press 'E' to enable password-protection, then 'T' to reset. A reset is similar to removing and replacing the transponder from the RF field. Now an error will occur when trying to write to any location. Log back in to gain write access and try writing some data once more.

- ❏ Try experimenting on your own. Be careful not to lose the password if it is changed. Open em4150.c and explore some other options as well.

11 ANTI-COLLISION

❏ The transponders provided in the kit only permit one transponder in the RF field at a time. They always broadcast data while powered and respond to every command they receive. When two or more transmit at the same, their signals will interfere with each other, preventing any communication. Try reading the read/write transponder with the read-only transponder in the field at the same time to witness the effect. The single transponder scheme is sufficient for many applications, such as access control; however, sometimes it is necessary to read multiple transponders at once. Some example situations include reading an entire pallet of items or a continuous stream of packages.

❏ There are many algorithms for handling multiple transponders in a single RF field, as long as the transponders support anti-collision. Some algorithms work better for certain situations. Described below is one algorithm for read-only anti-collision and one for read/write anti-collision.

❏ This method is suited for read-only transponders. Each transponder contains a random number clock generator and a random number clock counter. Unlike the transponders in the kit, these do not transmit when powered until they see a gap in the RF field. At this time, they begin their random number generation, wait for a given period, then begin to transmit. Upon seeing the second gap, they stop their random number circuit and save the current number. Each gap in the RF field causes the saved number to be decremented. Once it reaches zero, the transponder transmits its data. Since each transponder has a different random number, there are no collisions. The reader controls communication by providing the RF gaps. If the reader notices a collision while reading, it ignores the data and provides another gap. Otherwise the reader attempts to read the data until successful, and then sends another gap. It would be very difficult to synchronize two-way communication with this method, so a different approach is used for some read/write transponders.

❏ One read/write algorithm begins with all transponders enabled at power on. When transponders are enabled they will respond to commands, but not continuously broadcast. The process starts with the reader sending a read command. If the reader notices a collision, it starts the arbitration process by sending the arbitration command. Each tag responds with its least significant address bit. The tags with a '0' respond immediately and tags with a '1' wait a moment before sending; this avoids collisions. The reader checks if it received a 0, 1, or 0 and 1. If the reader reads a 0, it responds with a 0. If it received only a 1, it responds with a 1. The transponders that sent a matching address bit continue the process by sending their next address bit. The cycle continues until the most significant address bit is read. The entire process identifies the address of one transponder. The reader disables this transponder and continues to find addresses until no more collisions occur. Now the reader can enable a transponder and communicate without interference.

12 RFID RESOURCES

There are many websites with great information about RFID technology, products, and design. Many manufacturers provide product information along with technology descriptions.

The RFID Handbook is located at *www.rfid-handbook.com*. This is an in-depth book discussing everything about RFID. The site offers a forum and a free preview of one chapter in the book.

The RFID Journal is located at *www.rfidjournal.com*. This site contains RFID news, case studies, ideas, and opinions. Some articles are free, while others require a paid subscription.

EM Microelectronic is located at *www.emmicroelectronics.com*. Manufacturers of transponders in this kit. See their website for datasheets, application guides, different transponders, and other information.

Escort Memory Systems is located at *www.ems-rfid.com*. They are another company that offers information about RFID and a variety of transponders and readers.

RFID Updates are located at *www.rfidupdate.com*. This site contains daily news posting related to RFID.

RFID Talk is located at *www.rfidtalk.com*. This site is a forum discussing design advice, news, jobs, and general information about RFID.

Links to RFID privacy articles are located at *www.rfidprivacy.org*. This site provides articles and information that explore the issue of privacy as RFID becomes more integrated into our society.

On The Web

Comprehensive list of PICmicro® Development tools and information	www.pic-c.com/links
Microchip Home Page	www.microchip.com
CCS Compiler/Tools Home Page	www.ccsinfo.com
CCS Compiler/Tools Software Update Page	www.ccsinfo.com click: Support → Downloads
C Compiler User Message Exchange	www.ccsinfo.com/forum
Device Datasheets List	www.ccsinfo.com click: Support → Device Datasheets
C Compiler Technical Support	support@ccsinfo.com

Other Development Tools

EMULATORS
The ICD used in this booklet uses two I/O pins on the chip to communicate with a small debug program in the chip. This is a basic debug tool that takes up some of the chip's resources (I/O pins and memory). An emulator replaces the chip with a special connector that connects to a unit that emulates the chip. The debugging works in a simulator manner except that the chip has all of its normal resources, the debugger runs faster and there are more debug features. For example an emulator typically will allow any number of breakpoints. Some of the emulators can break on an external event like some signal on the target board changing. Some emulators can break on an external event like some that were executed before a breakpoint was reached. Emulators cost between $500 and $3000 depending on the chips they cover and the features.
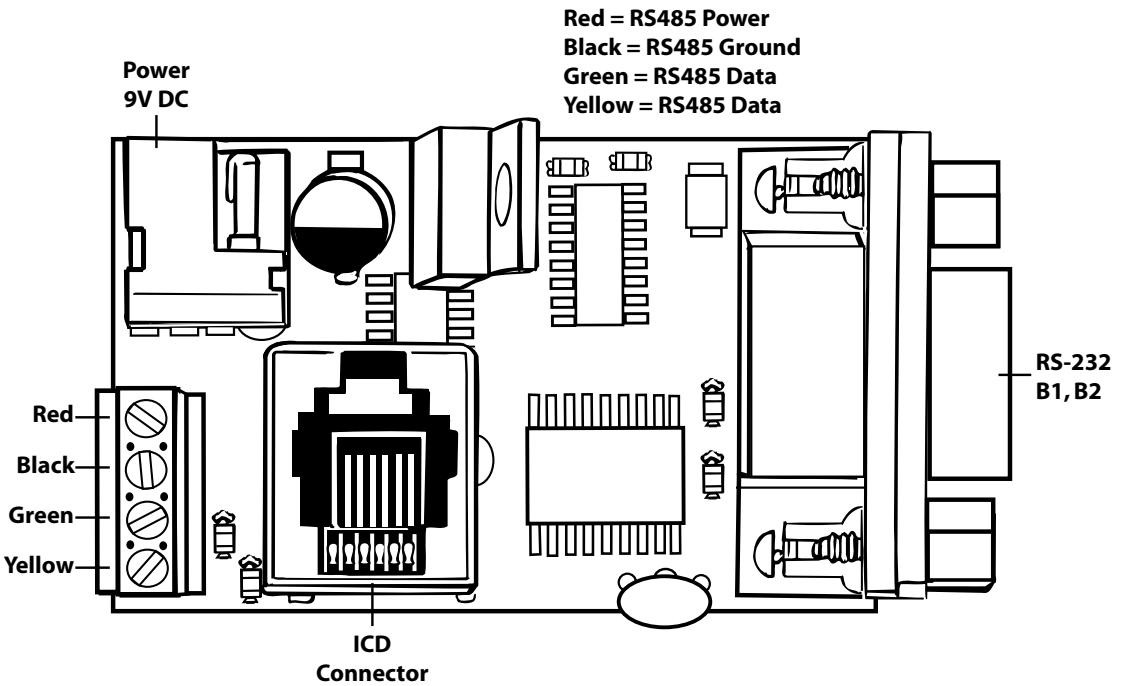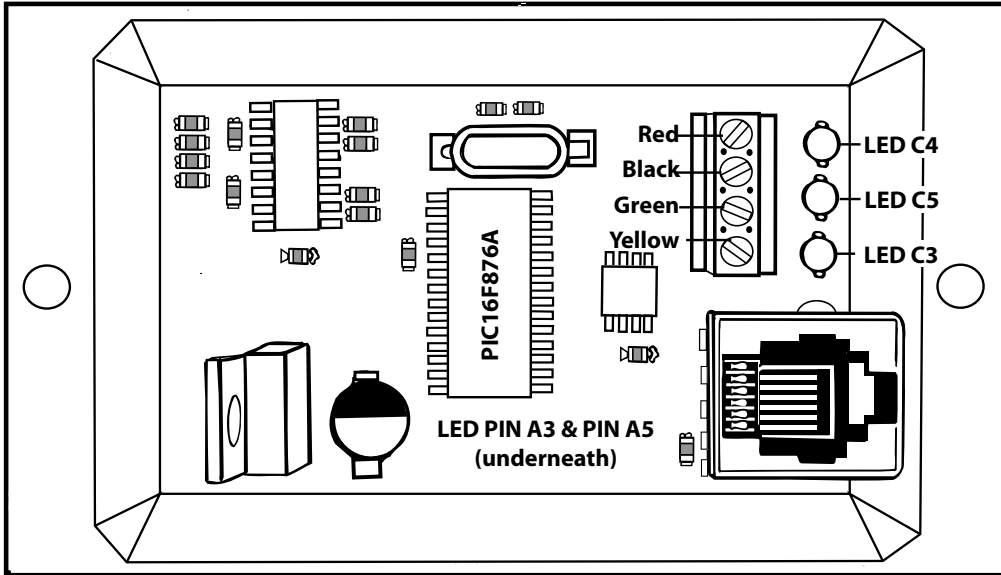
DEVICE PROGRAMMERS
The ICD can be used to program FLASH chips as was done in these exercises. A stand alone device programmer may be used to program all the chips. These programmers will use the .HEX file output from the compiler to do the programming. Many standard EEPROM programmers do know how to program the Microchip parts. There are a large number of Microchip only device programmers in the $100-$200 price range. Note that some chips can be programmed once (OTP) and some parts need to be erased under a UV light before they can be re-programmed (Windowed). CCS offers the Mach X which is a stand-alone programmer and can be used as an in-circuit debugger.
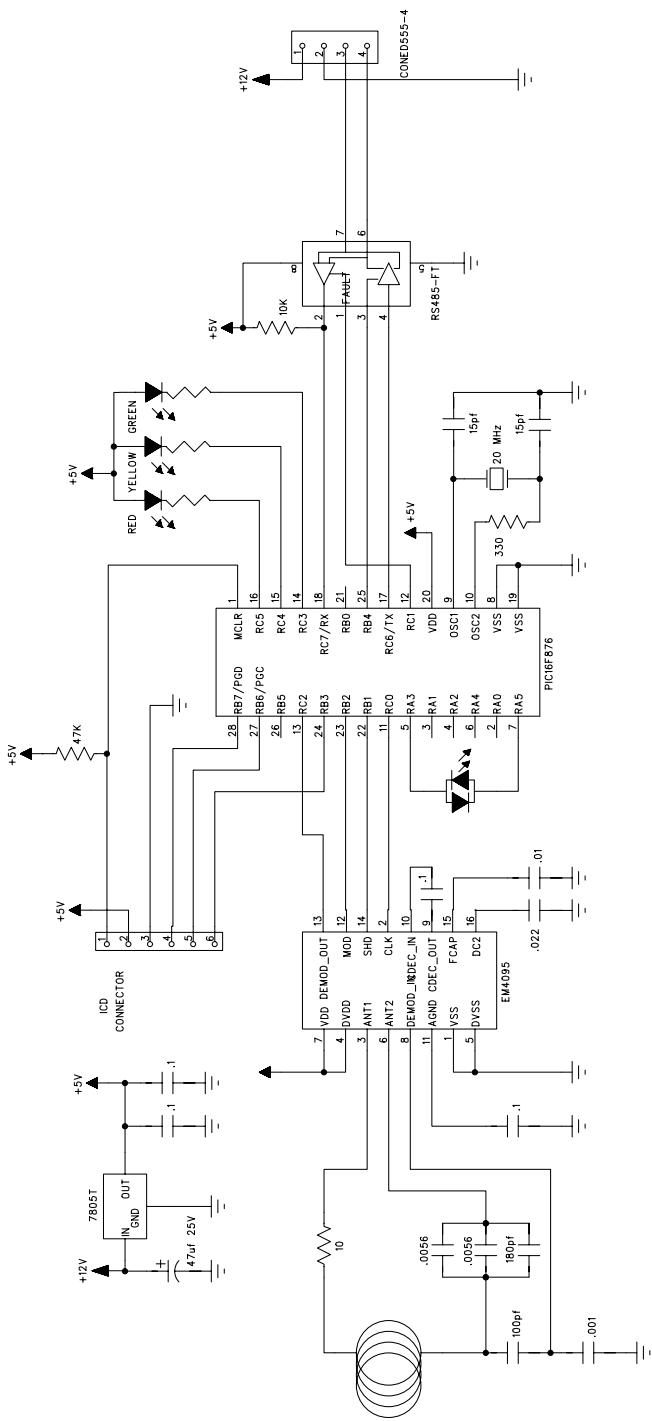
PROTOTYPING BOARDS
There are a large number of Prototyping boards available from a number of sources. Some have an ICD interface and others simply have a socket for a chip that is externally programmed. Some boards have some advanced functionality on the board to help design complex software. For example, CCS has a Prototyping board with a full 56K modem on board and a TCP/IP stack chip ready to run internet applications such as an e-mail sending program or a mini web server. Another Prototyping board from CCS has a USB interface chip, making it easy to start developing USB application programs.
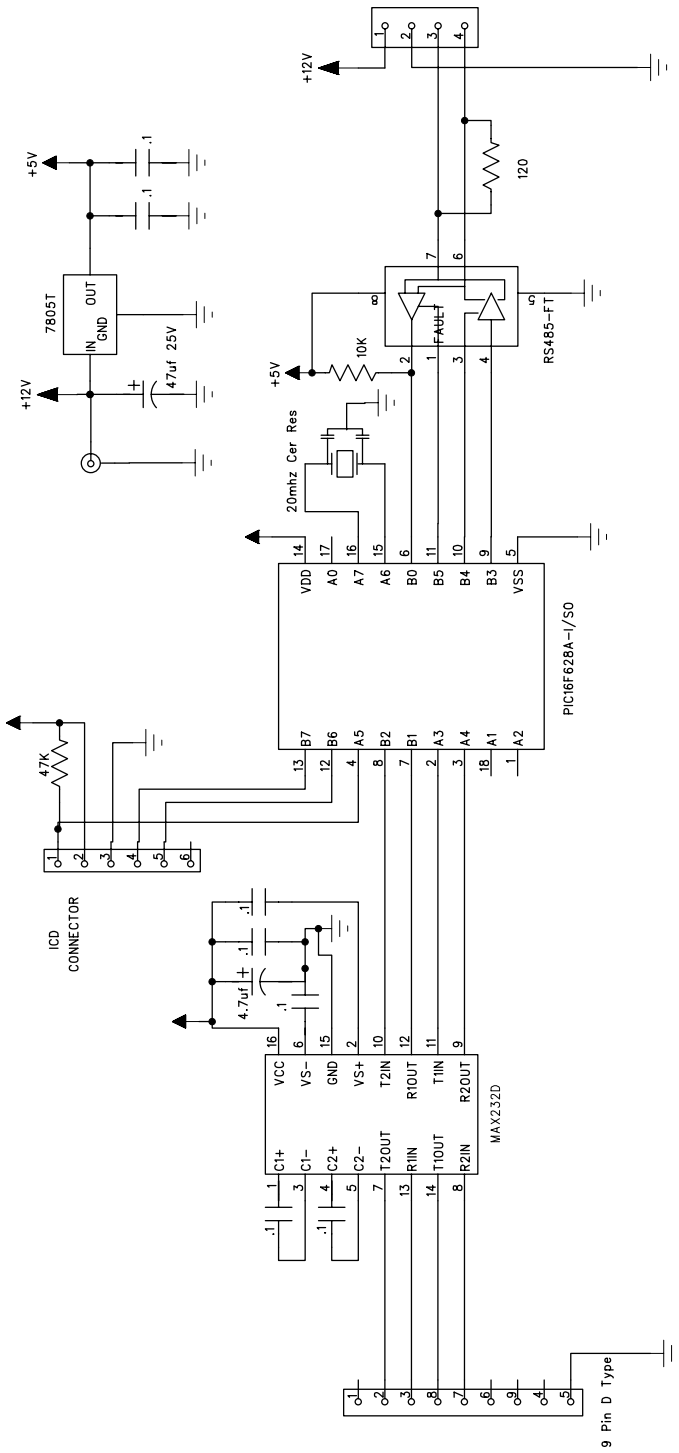
SIMULATORS
A simulator is a program that runs on the PC and pretends to be a microcontroller chip. A simulator offers all the normal debug capability such as single stepping and looking at variables, however there is no interaction with real hardware. This works well if you want to test a math function but not so good if you want to test an interface to another chip. With the availability of low cost tools, such as the ICD in this kit, there is less interest in simulators. Microchip offers a free simulator that can be downloaded from their web site. Some other vendors offer simulators as a part of their development packages.

Red — LED C4
Black — LED C5
Green — LED C3
Yellow

PIC16F876A

LED PIN A3 & PIN A5
(underneath)

Red = RS485 Power
Black = RS485 Ground
Green = RS485 Data
Yellow = RS485 Data

Power
9V DC

RS-232
B1, B2

Red
Black
Green
Yellow

ICD
Connector