

RF System Evaluation and Integration

Christopher Mischaikow

Advisor: Dr. Ravi Doraiswami

Synopsis: The main project I worked on during these eight weeks dealt with RF transceivers and making them communicate. When I first arrived Greg had just received the WIZ-SML-AI; an RF transceiver made by Abacom Technologies. Because we could not get the two WIZ-SML-AIs to communicate we decided to try and write our own program for it all. A few weeks later we received our next device: The TR-220 transceiver made by Otek Corporation. The serial port that was used by this chip was not very well designed and for that reason we decided to send it back. But before we did send it back we tried to fix the problem. The supposed solution, which did not work for the TR-220, ended up working for the WIZ-SML-AI. A few days later we received the TRF6901 made by Texas Instruments. The TRF6901 is a computer-free RF communications device that we could use to play a simple game between devices. Once we had both the WIZ-SML-AI and the TRF6901 what we then needed to do was to figure out how each chip worked; which is how the rest of the eight-week period was spent.

The Program: We first tried to make the WIZ-SML-AIs communicate we used Hyper Terminal, a piece of software that can send and receive information through the serial port. Unfortunately this didn't work, so we decided to look on the Internet for other pieces of free software that could possibly test if a signal going out one transceiver could be received by the other. We found quite a lot of software, but none of it did quite what we wanted for it to do. This led us to try and write our own serial program in C++ (for a copy of the final program, check appendix A).

What we first did was do research for examples and guides to writing software for serial communications. We ended up trying to decipher about 50 pages of code for serial communications in an attempt to understand how to write a piece of software that could get our WIZ-SML-AIs to talk to each other. The manual told us that the WIZ-SML-AI would be able to use serial information sent straight out of the computer and it could convert that information into a form that can be understood by the receiving chip transceiver.

Given this information we were able to figure out a skeleton for how the program would work. First, a connection between one serial port and the other has to be established. Second, information has to be sent through the port and received by the other end. Third, the other end needs to recognize that information has been received. Fourth, we need to be alerted when information has been successfully sent. And fifth, the connection between the two serial ports has to be broken.

Part One: The establishment of the connection is where we define all of the standards and set up communication between the computer and the RF transceiver. Because the WIZ-SML-AI has very common settings, we can just ask for the computer to collect the information from the device, but because we knew that we were going to use the software for more than just one RF device. We decided to tell the computer all of the transceiver's settings so that if we decided to use a different transceiver we could manually change the settings ourselves without having to worry about whether or not the transceiver could give the information to the computer.

Through out this program we need to have multiple error checks so that if there is ever a problem or a failure for anything we can pull out and break the connection immediately. So throughout

the program we set up spots where a status check was entailed and we told it to jump to step 5: the communication break. This error check made up a significant part of the entire setup since if there is ever an error there is the chance the program will freeze and we will be leaving the comm. link open between the computer and the RF transceiver. If it does happen that the transceiver-computer link is still open after the program has stopped it will significantly slow down the computer and could possibly cause fatal errors with other software, ultimately resulting in the need for the computer to be rebooted for the connection to be closed.

Step Two: The second step, the sending of information, is one of the most complex since the information sent has to be able to be exposed to corruption and noise when being sent from one transceiver to the other so that there can be a minimum change in the message or else the information sent will not be correct. One way of avoiding this is to send several identical pieces of information from one chip to another so that even if one bit gets corrupted, chances are that the others won't and the information can be corrected.

Step Three: The downside to this is that it means that the other end has to not only recognize the information it received, but it also needs to do a comparative check. This is the third step: receiving and recognizing the information received by the transceiver. This step has to make sure that the computer realizes when it has received information and when it is just picking up noise from its surroundings. How this would be done is not what I worked on. It is incredibly difficult to do and for this reason it is the only section that Greg worked on.

Step Four: The fourth section is what tells us when the RF transceiver has successfully received the message from the source. It can be a noise or a window that pops up, but it is more complicated than it seems. We need to first define the noise, or design the window so that the program knows what to put out. We decided to use the window, so we needed to create just a simple alert window that tells us when the information has been successfully sent from one transceiver to the other.

Step Five: The importance of the fifth step is explained above, and we made sure that everything written would lead to the last step so that the connection could be broken. While we were programming we discovered that the nature of our program was not that HyperTerminal was not the software we wanted to use, but that the plug needed to be inverted so that it could work. What was happening is that sometimes the RX and TX signals (the ports that send information in and out) need to be inversed because the computer's serial port uses one plug for both, and the wire therefore splits. This means that sometimes the signal will go down the wrong path, and reversing the plug on the chip will fix the problem.

For this reason we never got past step two on the program although we had created a very detailed outline for the entire program.

The Transceivers: A few days after we discovered what was wrong with the plug for the WIZ-SML-AI we received the TRF6901. Both of these are very similar RF transceivers, and what we tried to do is understand how they work so that we could then set up a plan for creating our own. One, the WIZ-SML-AI, sent and received information and relayed to the computer; the other, the TRF6901, used its own I/O system so that there was not need for a computer.

Greg was able to define quite a few sections out of the schematic given to us by the manufacturer for the TRF6901 (see appendix B), but since I didn't know much about the subject. I decided to

read on it and do some research about the work instead from which I was able to learn a little about how the transceivers worked.

Appendix A: Comm. Port program

```
#include <resource.h>
#include <studio.h>
#include <stdlib.h>
#include <process.h>
#include <conio.h>
#include <windows.h>
#include <winbase.h>

#define baud_rate 115200

/*****
**/
// Open Port function
/*****
**/
HANDLE open_port (port_num, int baud_rate)
{
    // port_num == the port number
    // baud_rate == the baud rate
    // these values will be provided from the main loop

    char port_name [8];
    HANDLE port_handle;

    // Define the two separate ports

    // opens the port:
    port_handle = CreateFile(port_name,
                                GENERIC_READ|GENERIC_WRITE,
                                0,
                                NULL,
                                OPEN_EXISTING,
                                0,
                                NULL);

    // ERROR CHECK
    if (port_handle == INVALID_HANDLE_VALUE) // This is the error checker
                                                // It checks if
the port has been opened or not
    {
        port_handle = (HANDLE) -1;
    }
else // This is if
there is no error
    {
        DCB dcb;
```

```

    int status;          // status is just an error-checking variable
    COMMTIMEOUTS ct

    status = GetCommState(port_handle, &dcb); // this just double-
checks the connection

    // if GetCommState fails, status=0
    if (status!=0) //error check, when status has not failed, the
program continues
    {
        dcb.BaudRate = baud_rate;
        dcb.ByteSize = (unsigned char) 8;
        dcb.Parity = 0;
        dcb.StopBits = ONESTOPBIT;
        dcb.fBinary = TRUE;
        dcb.fDtrControl = DTR_CONTROL_DISABLE;
        dcb.fRtsControl = RTS_CONTROL_DISABLE; // not used here
        dcb.fAbortOnError = 0;
        dcb.fOutxCtsFlow = 0; // not used
here

        status = GetCommState(port_handle, &dcb);
            // this just double-checks the connection
            // if GetCommState fails, status=0

        if (status=0) //error check
        {
            ClearCommError (port_handle, CE_IOE, NULL);
        }
    }
    else // error check, when status has failed, the
program cuts here.
    {
        ClearCommError (port_handle, CE_IOE, NULL);
    }

        // Setting up the timeouts.
    ct.ReadIntervalTimeout = MAXWORD;
    ct.ReadTotalTimeoutMultiplier = 0;
    ct.ReadTotalTimeoutConstant = 0;
    ct.WriteTotalTimeoutMultiplier = 0.156;
    ct.WriteTotalTimeoutConstant = 4;
    SetCommTimeouts (port_handle, &ct);
    status = SetupComm (port_handle, 96, 96);
}
if (status=0) // error check
// if the timeouts have failed, the program
goes through this
{
    ClearCommError (port_handle, CE_IOE, NULL);
}

return port_handle;
// the serial port is now connected with the computer.
}

```

```
/**
**/
//  MAIN
/**
**/
for(;;)
{
    int bytesread
    WaitCommEvent(Com2
    //this begins 2 cycles to open ports one and two
    for(int port_num = 1; port_num <= 2; port_num++)
        open_port (port_num, baud_rate);
}
```

Appendix B: Schematics

TRF6901:

