Department of Electrical and Computer Systems Engineering

Technical Report MECSE-11-2004

Establishing point-to-multipoint operation with multiple Radiometrix SpacePort Modems (SPM2-433-28)

Robert L. Stewart

MONASH UNIVERSITY

Establishing point-to-multipoint operation with multiple Radiometrix SpacePort Modems (SPM2-433-28)

Robert L. Stewart

Intelligent Robotics Research Centre,
Department of Electrical and Computer Systems Engineering,
Monash University, Clayton,
VIC 3800, Australia

Abstract. This report details the steps necessary to establish point-to-multipoint operation of at least two Radiometrix SpacePort Modems (SPM2-433-28). The hardware modifications necessary to allow a master device to address up to 16 slave devices are provided. Along with this, some example Visual C++ code is also provided that establishes software control of the master device. The SpacePort configuration settings used by the master/slave devices are given and an example communications strategy for the entire system is detailed.

1. Introduction

Radiometrix SpacePort Modem modules are able to be configured in different modes of operation. One mode of operation is Point-To-Multipoint. In this mode, one modem is configured as a master device typically under the control of a PC. This device then dynamically changes its unit address to match the address of one of up to 16 slave devices. In a robotic system each slave may be under the control of a microcontroller. It is within this context that the information in this report is to be taken.

The sections in this report detail the different stages that may need to be undertaken to establish point-to-multipoint communication. The sections are as follows:

2. Configuration settings for master and slave devices
3. Hardware modifications for the PCB of the master modem
4. Using software to dynamically address slave devices
5. Communications strategy for controlling multiple robots

It should be noted that the information provided in this report is intended only as a guide and may contain errors. It details the setup of what seems to be a workable system. However, the reader should make changes where appropriate to reflect their intended use for the system.

2. Configuration settings for master and slave devices

The department has designed PCBs and had them manufactured to hold the radiomodem and MAX232 chips. The circuit diagram is shown in Figure 2. It is assumed that the reader is intending to use one of these manufactured PCBs.

Each modem should be configured for use. This can be readily achieved using HyperTerminal. If you are unsure of the current settings of your radiomodem it is best to enter the setup in default mode. To do this, connect jumper 5 (JP5 in Figure 2). Connect the modem to a serial port of a PC and apply power. Open HyperTerminal and 'Connect using:' COM1 or COM2 as appropriate. If using default mode chose: Bits per second=9600, Data bits=8, Parity=None, Stop bits=1, Flow Control=None. Connect jumper 6 (JP6 in Figure 2) to enter the spaceport modem configurator. The '>' character should appear on the screen. Display the current settings by typing 'config' and pressing enter. It is then possible to modify the different variables depending on the intended use. Refer to the SpacePort Modem manual for more details regarding these variables [1].

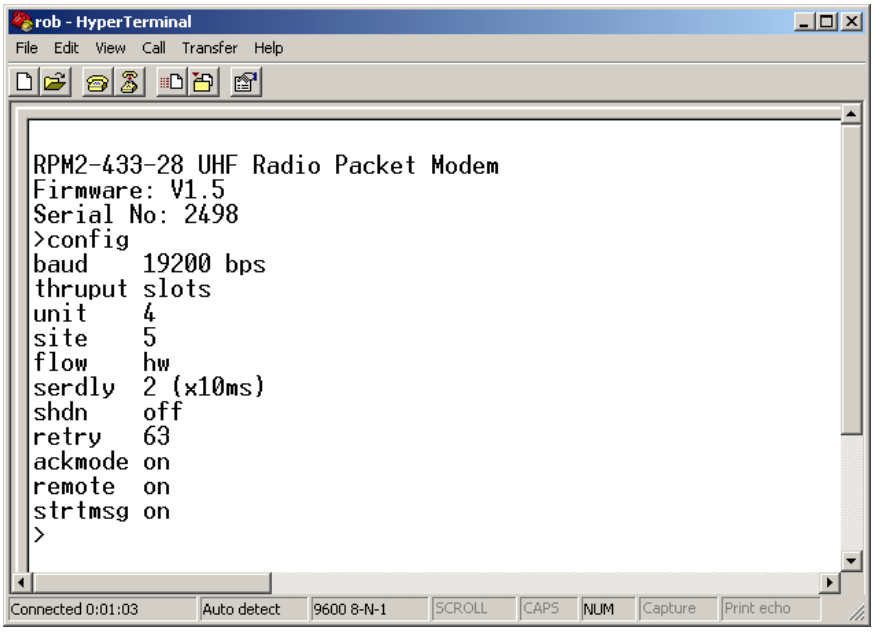

Fig. 1. Typical settings displayed in HyperTerminal using the 'config' command.

Figure 1 details typical settings for one of the modem modules. The 'site' number should be the same for all the modem units in the system. "The site number is used to distinguish between groups of operating modems" [1]. Therefore, if other people are using the modems and there is likely to be interference, establish with them which site number they will be using. The unit number should be different for each slave device

in the system. If five slave units are to be used then perhaps configure them with unit numbers 0, 1, 2, 3 and 4. Later, the master unit will be able to talk to a slave unit by dynamically setting its unit number to match the unit number of the slave it wants to communicate with.

Again, if other people are using radiomodems then another arrangement needs to be worked out. The 'thruput' should be set to slots in all systems. This will reduce throughput but will allow "other SPM pairs, operating within close proximity, equal opportunity to transmit data" [1]. If this is not done then it is likely that one or both systems will not function correctly. Also, it is important to enable hardware flow control (i.e. CTS) on each device that is connected to a radiomodem. This will prevent data being lost due to "internal buffer overruns" [1]. Setting retry to the maximum value of 63 will maximize the number of retries should there be some form of radio interference present. By turning on 'ackmode' we assume that error free data is sent and received.

The start message was turned off for the master unit and shutdown was turned off for all units. The variable 'serdly' was set to the value of 2. Finally, the 'remote' configuration option was kept on although this isn't so important.

3. Hardware modifications for the PCB of the master modem

To dynamically change the address of the master spaceport modem, the 'configurator' must be entered and the 'addr' command used to change the unit address. To do this we require the ability to change the state of the $\overline{\text{setup}}$ pin. It was decided that the DTR line of the serial cable would be used to drive the $\overline{\text{setup}}$ pin of the radiomodem chip. This can be achieved by cutting the track between pin 16 ($\overline{\text{WAKE}}$ / DTR) of the radiomodem chip and pin 12 (R1OUT) of the MAX232 chip (in Figure 2 changes are indicated in red), and then by connecting pin 12 of the MAX232 chip to pin 1 of jumper 6 (JP6) which connects to the radiomodem $\overline{\text{setup}}$ pin.

4. Using software to dynamically address slave devices

In the robotic system that was developed a PC was connected to the master radiomodem. Five slave modem units were configured and each connected to a microcontroller onboard a robot. To establish communications between the master unit and a slave unit it is necessary to set the unit number of the master to match the unit number of the slave. As mentioned, this can be done dynamically by entering the configurator and using the 'addr' command. To enter the configurator we need to assert the $\overline{\text{setup}}$ pin. Since this is now connected to the DTR line of the serial port we can readily do this in software.

A project was created under Microsoft Visual C++. Some useful documentation is available to assist in writing a project that uses serial communication in windows [2].

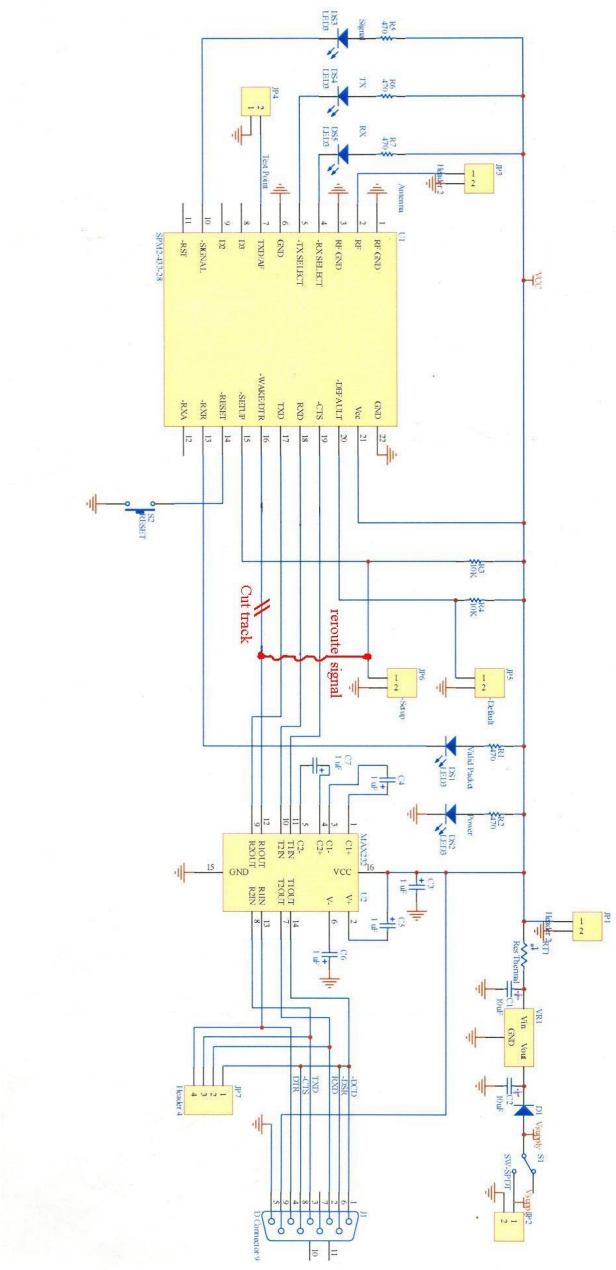

Fig. 2. The circuit diagram for the radiomodem PCB (based on a circuit detailed in [1]). Changes to be made for the master modem are indicated in red.

The code from two of the more important functions is included in this report. The function *setUpSerialSettings(void)* sets up and opens a serial port for communications. The *setAddress(int n)* function receives the unit number of the slave that we want to communicate with. The command *EscapeCommFunction(hSerial, SETDTR)* is used within this function to enter the configurator of the master unit. The unit number of the master unit is then set to match the unit number of the slave (*n*) by writing the command 'addr' to the serial port followed by the number *n*. The character '.' is then received back from the radiomodem indicating that the address has been changed and that communication between master and slave can commence. Code snippets follow.

Global variables:

```
HANDLE hSerial;
DCB dcbSerialParams;
DCB dcbSerial;
COMMTIMEOUTS commTimeouts;
```

Function to setup the serial communications:

```
void setUpSerialSettings(void)
{

  hSerial=CreateFile("COM1", GENERIC_READ | GENERIC_WRITE, 0,0, OPEN_EXISTING,
                     FILE_ATTRIBUTE_NORMAL, 0);
  if(hSerial= =INVALID_HANDLE_VALUE)
  {printf("ERROR OPENNING PORT\n");}

  //initialise communication parameters for the device
  printf("result of setup is:%d\n", SetupComm(hSerial, 2000, 2000));

  if(!BuildCommDCB("19200,n,8,1", &dcbSerial))
        printf("error setting port conditions\n");

  dcbSerial.fOutxCtsFlow=1;//allow flow control

  commTimeouts.ReadIntervalTimeout=0;
  commTimeouts.ReadTotalTimeoutConstant=1000;
  commTimeouts.ReadTotalTimeoutMultiplier=0;
  commTimeouts.WriteTotalTimeoutConstant=1000;
  commTimeouts.WriteTotalTimeoutMultiplier=0;

  if(!SetCommTimeouts(hSerial, &commTimeouts))
        printf("error setting timeouts conditions\n");

  if(!SetCommState(hSerial, &dcbSerial))
    printf("Error setting Comm state\n");
}
```

Function to establish a link between the master unit and a designated slave unit:

```
void setAddress(int n)
{
  //this function assumes the startup message has been deactivated

  int j;
  const int NUMBYTES=1;
  char szBuff[NUMBYTES+1];
```

```
    char szBuff2[NUMBYTES+1];
    DWORD dwBytesRead;
    DWORD dwBytesWritten;

    debug("set address=%d", n);
    //enter the configurator by asserting the DTR signal – this is rerouted to connect to the setup pin
    EscapeCommFunction(hSerial, SETDTR);

    //keep reading characters until the '>' character is received
    do
    {
            if(!ReadFile(hSerial, szBuff, NUMBYTES, &dwBytesRead, NULL)) printf("error\n");
            printf("%c", szBuff[0]);
    }
    while(szBuff[0]!='>');

    //buffer to hold the information to send to the radiomodem
    char mytext[20];

    //deassert the DTR pin ready for next time
    EscapeCommFunction(hSerial, CLRDTR);

    //formulate a command to issue to the master unit
    sprintf(mytext, "addr %d\r", n); //address the appropriate slave device (given by n)

    //send each character in the string (FIX this loop terminating condition if n>9 – okay for me since only
    //servicing 5 robots)
    for(j=0; j<7; j++)
    {
       szBuff2[0]=mytext[j];
            if(!WriteFile(hSerial, szBuff2, NUMBYTES, &dwBytesWritten, NULL))
            { /*error reading serial port. Process appropriately.*/  }
       if(!ReadFile(hSerial, szBuff, NUMBYTES, &dwBytesRead, NULL))
            { /*error reading serial port. Process appropriately.*/}
       printf("%c", szBuff[0]);
    }

    //read characters until the '.' character is received
    do
    {
      ReadFile(hSerial, szBuff, NUMBYTES, &dwBytesRead, NULL);
      printf("%c", szBuff[0]);
    }
    while(szBuff[0]!='.');

  }//end setAddress(int n)
```

4. Communications strategy for controlling multiple robots

An application was developed that allows multiple robots to be controlled. In this application each robot has a main control loop that resides in a thread on the PC. This control loop contains low-level robot commands. A communications thread relays these commands to the correct robot and also receives back any data. The communications thread is continually servicing the robots according to their need. To determine how a robot should be serviced a query character ('?') is issued to each

robot which replies with its current state. Each robot can be in one of four states. These are:

- **a**: attentive – waiting to receive a new command
- **b**: busy – currently executing a command
- **c**: completed – finished executing command & ready to send back return data
- **d**: data – wanting to send back log data

The serial receive and transmit functions on the robot side are interrupt driven. When the '?' query character is received an interrupt is generated and the microcontroller responds by sending back the character that corresponds to its current state. The communications thread then responds as appropriate. The control thread is outlined in the following pseudocode.

Communications thread pseudocode:

```
char robotState;
int j;
while(1)   //continuously service each robot in turn
{
  for(j=0; j<NUM_ROBOTS; j++)
  {
    setAddress(j);  //establish communications channel to robot j

    robotState=getRobotState();   //this function sends a query character to robot j and
                                  //receives back the robot's  current state

    if (robotState= ='a')  //a=attentive: the robot is waiting for a new command
    {
      wait for robot main control loop (thread on PC) to issue a command
      relay this command to the robot
    }
    elseif (robotState= ='b') //b=busy: the robot is busy so do nothing
    {}
    elseif (robotState= ='c') //c=completed: the robot is ready to send back data
    {
      receive return data from robot
      signal to robot main control loop that the issued command is now complete
    }
    elseif (robotState= ='d') //d=data: the robot has some log data to send back
      receive log data, time stamp and save it to file
  }
}
```

This technique is quite effective since the communications device is always occupied, checking the state of a robot and sending/receiving data where necessary. The *setAddress(j)* function detailed earlier is used to establish communications between the master and a selected robot slave.

The core components of the complete system that was developed are represented diagrammatically in Figure 3. Additional features also implemented but not discussed here include: the facility for a user to issue a selected command to a chosen robot, a graphical user interface and user feedback such as battery monitoring.

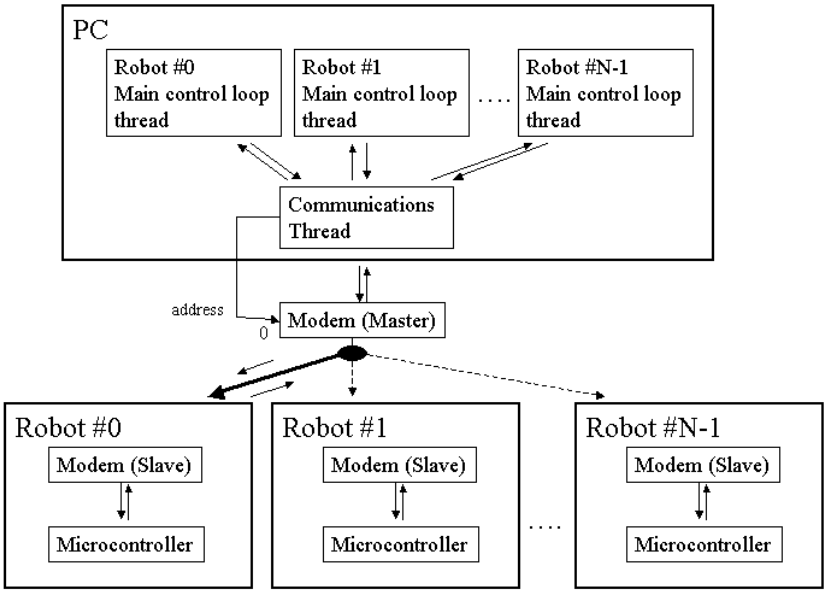

Fig. 3. Core components in the complete system design. Here, N is the number of robots.

5. Conclusion

This report has detailed a method for establishing point-to-multipoint communications within the context of a robotic system. Hardware and software configurations were explained and the core components of a complete system outlined. Note that this document is intended as a guide only.

6. Acknowledgements

The work outlined in this paper is being undertaken as part of a postgraduate research project. The project is supported by the Australian Research Council funded Centre for Perceptive and Intelligent Machines in Complex Environments. The work is also supported by a grant from Monash University's Faculty of Engineering Small Grants Scheme. Robert Stewart would like to acknowledge the support of an Australian Postgraduate Award. The help and advice of academic and support staff within the Department of ECSE, Monash University, has also been appreciated.

6. References

[1]. Radiometrix, “UHF SpacePort Modem,” *Technical Document*, *a*vailable online at http://www.radiometrix.co.uk/dsheets/spm2.pdf, England, 2004.

[2]. Allen Denver, “Serial Communications in Win32,” *Technical Article*, available online at
http://msdn.microsoft.com/library/en-us/dnfiles/html/msdn_serial.asp, 1995.