Syracuse University

SURFACE

Syracuse University Honors Program Capstone Projects

Syracuse University Honors Program Capstone Projects

Spring 5-1-2005

Using A PDA to Control Home Appliances

Atif Albraiki

Follow this and additional works at: https://surface.syr.edu/honors_capstone

 Part of the Computer and Systems Architecture Commons, and the Digital Communications and Networking Commons

Recommended Citation

Albraiki, Atif, "Using A PDA to Control Home Appliances" (2005). *Syracuse University Honors Program Capstone Projects*. 679.
https://surface.syr.edu/honors_capstone/679

This Honors Capstone Project is brought to you for free and open access by the Syracuse University Honors Program Capstone Projects at SURFACE. It has been accepted for inclusion in Syracuse University Honors Program Capstone Projects by an authorized administrator of SURFACE. For more information, please contact surface@syr.edu.

Using A PDA to Control Home Appliances

Atif Albraiki

Candidate for B.S. Degree
in Computer Engineering with Honors

May / 2005

APPROVED

Thesis Project Advisor: _____
Dr. Daniel J. Pease

Second Reader: _____
Dr. Frederick W. Phelps

Honors Director: _____

Honors Representative: _____

Date: _____

Abstract

The Personal Digital Assistant (PDA) is a handheld device that gives users different organizational capabilities. Over the years, the functionality of PDAs has expanded tremendously to include internet connectivity and different computer applications. The purpose of my honors thesis, combined with my engineering senior design project, was to include a new functionality to the PDA and use it to control different home appliances. The goal of the project was to develop a PDA program that will communicate with a PC program, and have the PC program communicate with a microcontroller to control home appliances. My part of the project was the development of the PDA and PC programs and the communication between them and the microcontroller.

In this paper I will describe some basics of the PDA, and I will also describe how the two programs that I developed work and communicate together. In addition, I will describe how the PC program communicates with the microcontroller. Most of the microcontroller program and the hardware connection between it and the home appliances were implemented by two members on my senior design team. I will briefly describe their work in this paper to give a clear picture of the project.

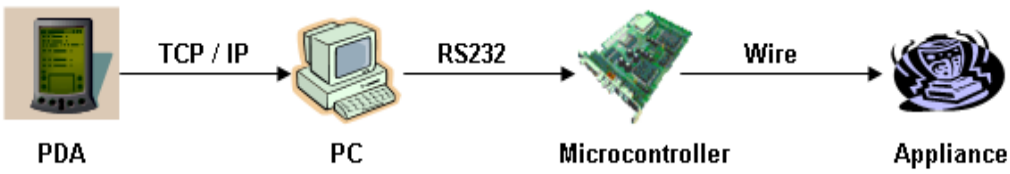

Figure 1: Connection Between Compoments

Table of Contents

Title	Page
Acknowledgements	**i**
Basics About PDAs	**1**
Overview About the PDA Program	**1**
Communication Between PDA and PC	**4**
Functions' Description of the PDA Program	**5**
Installing PDA Program	**10**
Overview About The PC Program	**11**
Communication Between PC and Microcontroller	**12**
Functions' Description of The PC Program	**13**
Microcontroller and Hardware Connection	**17**
How Can This Project Be Taken Further?	**19**
References	**20**
Appendix A: Source Code for the PDA Program	**21-A**
Appendix B: Source Code for the PC Program	**21-B**

Acknowledgments

I would like to thank Dr. Daniel Pease, Dr. Linda Milosky, Dr. Duane Marcy and Dr. Frederick Phelps for supervising me throughout this thesis and senior design project and for their great help. I also would like also to thank my senior design teammates Renaldo Williams, Lindsay Robbins and Matt Candurra for all the work that we did together as a team. And for the biggest part, I would like to thank my parents, without whom I wouldn't have been able to complete this thesis.

Basics About PDAs

The operating systems running on PDAs are different than operating systems running on desktop PCs. While PCs run operating systems such as Windows XP and Linux, PDAs run different operating systems such as Windows Pocket PC and Palm OS. These operating systems run on special CPUs such as the ARM and the MIPS microprocessors. The PDAs don't have hard disks; they store their basic programs in read-only memory (ROM). Additional programs are stored in random-access memory (RAM) which is erased whenever the device completely loses power. This explains why PDAs keep on running on low power even when they are turned off.

Overview About The PDA Program

PDA programs can be written in different languages such as Visual Basic and C++. For this project I decided to develop the program using C# because it provides powerful features and it simplifies the program development process. C# is a managed code language that gives the programmer the capability of developing applications that target the .NET Compact Framework. There are two advantages for targeting the .NET Compact Framework. The first advantage is the common language runtime, which makes the framework responsible for managing the code at runtime so that the programmer doesn't have to worry about writing code for memory and thread management. The second advantage is the rich class libraries included in the framework. These libraries give the programmer the capability of using prewritten class variables and functions and make the

development process easier. The operating system that I targeted for the PDA program was Windows Mobile 2003, also known as Windows Pocket PC 2003, and it is based on Windows CE .NET 4.2.

The program was built and tested using Microsoft Visual Studio .NET 2003, which was the only option I had for developing mobile device applications using C# .NET. I had to download the Pocket PC 2003 Software Development Kit (SDK) to develop applications for Windows Mobile 2003. I also had to download the emulator for Pocket PC 2003 to be able to test the program on my PC. For the purpose of connecting to the PDA device and downloading the developed program there, I had to download Microsoft ActiveSync 3.7.1.

The first development process for the PDA program was to specify the graphical user interface (GUI). I tried to keep the GUI as simple as possible to avoid confusing the user. All windows forms in the program are displayed in a full screen mode, except in the case of displaying error messages where the form is displayed as a small message box. I created a special integer in my program to keep track of the state of the program. This integer is called *ProgramStatus*. The program GUI is displayed based on the value of *ProgramStatus*, as seen in the next page in figure 2.

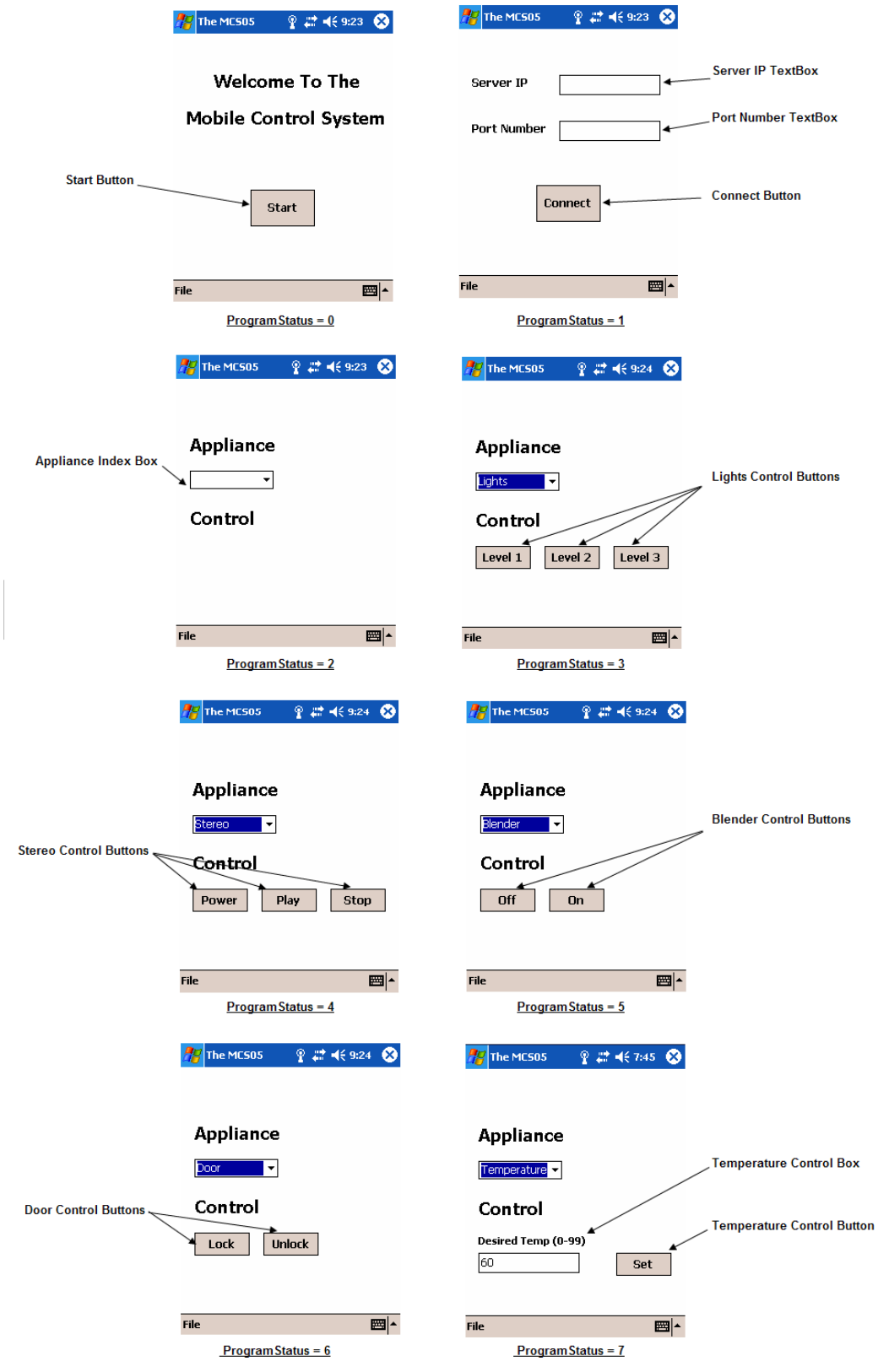

Figure 2: PDA program snapshots based on the value of ProgramStatus

Communication Between PDA and PC

There are different ways that programs can communicate together. For example, they can communicate using Bluetooth, Infrared or Internet. I used the Internet as a communication method between the PDA program and the PC program. The Internet actually uses TCP/IP, which stands for Transmission Control Protocol / Internet Protocol. TCP is a connection-oriented protocol, meaning that it provides reliable connection and data delivery between two computers, usually a client and a server. IP is a protocol that manages the routing of the message from the sender to the receiver.

Each computer on the Internet has a unique IP address that is provided by the Internet Service Provider. It is divided into four octets, separated by a dot, in the form of n.n.n.n, with each octet holding a value between 0-255. In order for a client to connect to a server, it needs to know the server's IP address. In this project, the PDA program acts as a client program that needs to communicate with the server PC program. The PDA program needs to know the IP address of the PC that it wants to communicate with. However, when the data, also known as packet, reaches the PC, it needs to know to which program it must go to. Therefore, the PC program must be listening into a specific port number for incoming messages or connection requests.

A port number is a logical 16-bit number that works as a channel to a specific application in a computer. Specific service applications such as FTP and HTTP have special port addresses called well-known port addresses and they range from 0-1024. The user must enter the port number that he or she wants the PC program

to be listening at. It is always advisable to enter a port number that is not used by another application to avoid having the error of two programs listening at the same port. The combination of the server IP address and a port number is known as socket and it defines a unique application to connect to in the Internet.

Functions' Description of the PDA Program

a. Main()

This is the first function called when the user requests to open the PDA program. It starts running the application by creating a new instance of PDAProgram class, which is the main form in this program. Creating an instance of the class leads to calling the class constructor.

b. PDAProgram()

This is the PDAProgram class constructor. It performs two actions. First it specifies the value of *ProgramStatus* to 0. Second it calls the *InitializeComponent()* function, which is described below.

c. InitializeComponent()

This function is called from the class constructor. It initializes all the windows controls in the program, such as buttons and textboxes, into their default value. It specifies their sizes, locations and it also specifies the function that should be called when a specific button is clicked. Initializing the values doesn't mean that the control will appear in the GUI. The control needs to be added to the program's control collection to be activated. Therefore, the only control added in this initial stage is the start button.

d. OnPaint(PaintEventArgs)

This overridden function is called whenever the program is refreshed. The program can be refreshed at any point by calling the .NET library function *Refresh()*, which forces the form to redraw itself based on the content of the control collection. The *OnPaint(PaintEventArgs)* function calls the *DoPage(Graphics)* function which is described below.

e. DoPage(Graphics)

This function is called to redraw the screen whenever the program is refreshed. It draws the text that appears in the program based on the value of *ProgramStatus*. I always use a white background and black text with bold "Brush Script MT" font.

f. StartButton_Click(object , System.EventArgs)

This function is called when the start button is clicked. It changes the value of *ProgramStatus* from 0 to 1. Then it removes the start button from the control collection and adds the connect button, the server IP textbox and the port number textbox into the control collection. The user gets to write the server IP address and the port number to connect to inside the two textboxes. Then this function calls the *Refresh()* function to redraw the screen.

g. ConnectButton_Click(object , System.EventArgs)

This function is called when the connect button is clicked. It tries to connect to the server based on the IP address and the port number entered in the two textboxes. The function performs that by calling the *ConnectToServer()* function described below.

h. ConnectToServer()

This function tries to connect to the IP address and the port number requested by the user. The PC program must be listening at the requested port number to ensure the success of the connection. The function first initializes the socket that will be used in connecting to the PC program. I chose the TCP protocol for this connection. Next the function converts the inputted server IP address from string into a special *IPAddress* format that the socket can understand. It also converts the port number from string into integer. After that it creates a variable of type *IPEndPoint*, which combines the IP address and the port number to specify the end point of the connection. Then the socket tries to connect to the desired end point. If the connection succeeds, *ProgramStatus* value is incremented from 1 to 2, the current control collection is removed and the combo box that displays the appliances' names is added into the control collection. At the end of this function, the screen is refreshed.

i. ApplianceBox_SelectIndexChange(object , EventArgs)

This function is called whenever the user chooses one of the options in the appliance index box seen in figure 2. First it gets the index value of the appliance that the user chooses. It then removes the control buttons of the previously chosen appliance. It does this by calling *RemovePreviousControls()* function described below. Then based on the index value of the appliance currently chosen from the list, *ProgramStatus* value is set and the desired appliance control buttons are added into the control collection. At the end of this function, the screen is refreshed.

j. RemovePreviousControls()

This function is called from the *ApplianceBox_SelectIndexChange(object , EventArgs)* function. It removes the control buttons based on the value of *ProgramStatus*. Note that this function is called before the *ProgramStatus* is changed in the *ApplianceBox_SelectIndexChange(object, EventArgs)* function.

k. Lights1_Click(object , System.EventArgs)

This function is called when the user presses the Lights Level 1 button. It sends special bytes to the PC program informing it to turn on lights to level 1.

l. Lights2_Click(object , System.EventArgs)

This function is called when the user presses the Lights Level 2 button. It sends special bytes to the PC program informing it to turn on lights to level 2.

m. Lights3_Click(object , System.EventArgs)

This function is called when the user presses the Lights Level 3 button. It sends special bytes to the PC program informing it to turn on lights to level 3.

n. BlenderOn_Click(object , System.EventArgs)

This function is called when the user presses the blender on button. It sends special bytes to the PC program informing it to turn on the blender.

o. DoorUnLock_Click(object , System.EventArgs)

This function is called when the user presses the door unlock button. It sends special bytes to the PC program informing it to unlock the door.

p. DoorLock_Click(object , System.EventArgs)

This function is called when the user presses the door lock button. It sends special bytes to the PC program informing it to lock the door.

q. StereoPlay_Click(object , System.EventArgs)

This function is called when the user presses the stereo play button. It sends special bytes to the PC program informing it to play a song on the stereo.

r. StereoNext_Click(object , System.EventArgs)

This function is called when the user presses the stereo next button. It sends special bytes to the PC program informing it to play the next song on stereo.

s. StereoStop_Click(object , System.EventArgs)

This function is called when the user presses the stereo stop button. It sends special bytes to the PC program informing it to stop playing the stereo.

t. TempButton_Click(object , System.EventArgs)

This function is called when the user presses the temperature set button. Based on the value inside the temperature control box, it sends special bytes to the PC program informing it to set the temperature to the value in control box. If the value in the control box is not between 0-99, an error message is displaying asking the user to input a valid temperature number.

u. ExitMenu_Click(object , System.EventArgs)

This function is called when the user chooses from the menu file → Exit. If there is connection established with the PC program, the connection is terminated by sending specific bytes to end the connection. Then the socket is shutdown and closed in the PDA program. At the end, the Close() function within the .NET library is called to close the application. I added this menu to the program because when the user presses the X button at the top right of the

form, the application doesn't close completely. Instead it keeps on running behind the scene.

v. RestartMenu_Click(object , System.EventArgs)

This function is called when the user chooses from the menu file → Restart. It removes the current control collection based on the value of *ProgramStatus*. If there is a connection established with the PC program, it terminates it. Then it adds the start button to the control collection, it changes the value of *ProgramStatus* to 0 and it refreshes the screen.

Installing The PDA Program

In order to install the PDA program into the device I had to connect the device to the PC using Microsoft ActiveSync 3.7.1. The first step I had to do was to generate the CAB file by clicking on the Build Cab File in the Build menu in Visual Studio. This generated an .inf file and a batch file in the obj\Debug folder of the project. In the .inf file, I had to specify the name that would appear in the Program section of the device. After that I had to run the batch file with the changes and this generates different CAB files for different processors such as ARM and SH3 processors. Since I was targeting the ARM processor, I had to copy the generated CAB for ARM processors to the PDA device. Then when I run the file in the PDA device the program gets installed.

Overview About The PC Program

The PC Program was developed using C# and it targeted the .NET Framework. It gives the programmer the same capabilities as the ones described in the .NET Compact Framework. In fact .NET Framework even includes more libraries than the compact version. The PC program is not displayed in full screen mode. Instead the form is displayed in fixed size, 640x480. Similar to the PDA program, I created an integer called *ScreenState* to keep track of the state of the program. I also tried to keep the GUI as simple as possible, and the screen is drawn based on the value of *ScreenState* as seen in figure 3.

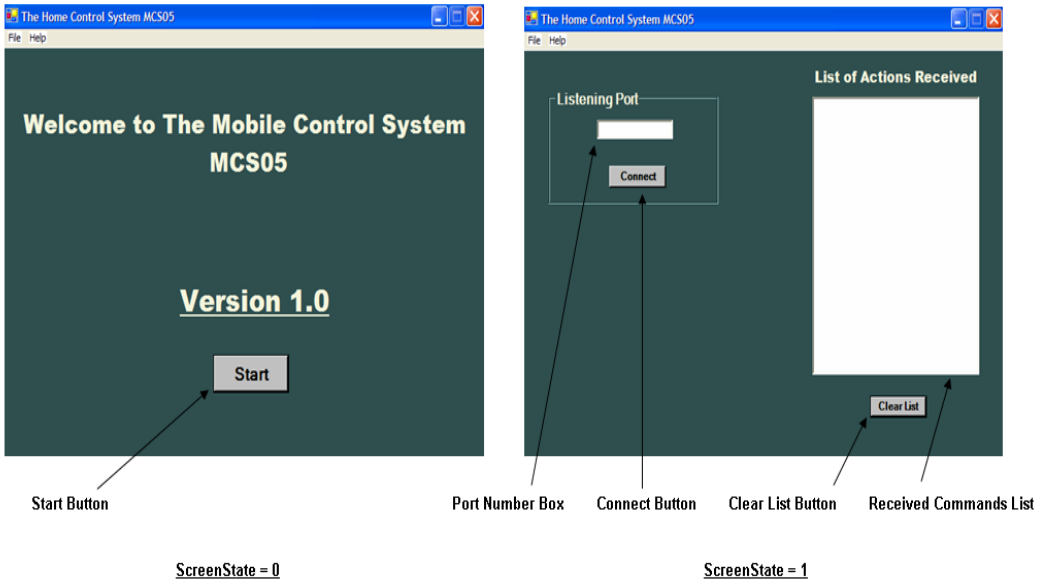

Figure 3: PC Program snapshots based on the value of ScreenState

As mentioned in the previous sections, the PC program must be listening into a specific port number for incoming connection requests. The user gets to enter the port number in the port number box seen in the figure above.

Communication Between PC and Microcontroller

The Microcontroller we used for this project is the Motorola MC9S12C32. We found that a PC program can communicate reliably with the Microcontroller through the RS232 serial interface. This protocol is used to communicate within the serial port. It is the communication method used to load a developed program from the PC to the microcontroller. As I mentioned before, I have been using Visual Studio .NET 2003 for developing the PC program. It works with .NET Framework 1.1, and unfortunately this framework does not contain a library for serial communication. This library is only included with .NET Framework 2.0, which works with Visual Studio .NET 2005. Unfortunately, up to today, this version of Visual Studio is only available for MSDN subscribers. The actual release is expected to be available sometime this summer.

To solve this problem, I had to include a free library called OpenNETCF.IO.Ports into the PC program project. This library is available through the OpenNETCF.org, which is a shared open-source site for .NET Compact Framework development. This library was written by Yuri Astrakhan, with the original idea implemented in Visual Basic .NET by Daniel Moth, a software developer and a member of the OpenCF.org. This library gives me the same capabilities found in the .NET Framework 2.0 serial port library.

Functions' Description of the PC Program

a. Main()

This is the first function called when the user requests to open the PC program. It starts running the application by creating a new instance of PCProgram class, which is the main form in this program. Creating an instance of the class leads to calling the class constructor.

b. PCProgram()

This is the class constructor that is called by the *Main()* function. It mainly does three actions. It calls the *InitializeComponent()* function to initialize the variables of this class, it sets *ScreenState* value to 0 and it calls the *CreateMainMenu()* function to create the main menu displayed in the top of the form as seen in figure 3.

c. InitializeComponent()

This function does the same task as in the PDA program. It only activates the start button that appears in the beginning of the program.

d. CreateMainMenu()

This function adds a menu at the top of the application. There are basically two items in the menu. The first is File → Exit, which closes the application whenever pressed. And the second is Help → About MCS05, which just shows a small message box describing the program.

e. OnPaint(PaintEventArgs)

This function is called whenever the screen is refreshed, and as in the PDA program, it calls the *DoPage(Graphics)* function.

f. DoPage(Graphics)

This function displays the text on the program based on the value of *ScreenState*. I specified the background of the program as a DarkSlateGray color and text as Beige Arial Black font.

g. MenuExitOnClick(object , EventArgs)

This function is called whenever the user presses File → Exit. Basically it closes the application.

h. MenuAboutOnClick(object , EventArgs)

This function is called whenever the user presses Help → About MCS05. Basically it displayes a message box describing the application.

i. StartButton_Click(object , EventArgs)

This function is called when the user pressed the start button at the beginning of the application. It removes all the items in control collection and adds the next GUI controls into the collection.

j. ConnectButton_Click(object , EventArgs)

The function is called when the user presses the connect button seen in figure 3. First it creates a listening socket that will be used to listen for incoming connection request at the specified port number entered in the text box. This socket must be bind with a specific IP address and port number. This is done by creating *IPEndPoint* variable. Since we want the application to accept connection request from any IP address, we specify the *IPEndPoint* value to accept connection from any IP address on the port number requested by the user. The *IPEndPoint* variable is then bound with the socket by calling

the socket function *Bind(IPEndPoint)*. After that the listening socket function *BeginAccept(System.AsyncCallback , object)* is called to specify which function to be called when a connection is requested. I specified the *ClientRequestConnect(IAsyncResult)* function to be called whenever a connection is requested.

This function also does two additional tasks. It disables the connect button so that the user can't have control over it, and it also opens a serial port connection, if not yet opened, to the microcontroller by calling the serial port function *Open()*. If an error occurs at anytime inside this function, an error message box is displayed that explains the error.

k. ClearList_Click(object , EventArgs)

This function is called when the user presses the clear list button at the bottom of the received command list. Basically it removes all the items from the command list.

l. ClientRequestConnect(IAsyncResult)

This function is called when the listening socket receives a connection request. It initializes a new socket to work as a connection socket. This socket starts listening for incoming data by calling the *WaitForData()* function, which is described below.

m. WaitForData()

This function makes the connection socket start asynchronously listening for incoming data. The received data is stored in a special buffer. Whenever data

is received, a special function *ServerReceiveData(IAsyncResult)* is called. The function is described below.

n. ServerReceiveData(IAsyncResult)

This function is called when the PC program receives data from the PDA program. First it checks to see if the received data in "END", which is sent by the PDA before it closes the connection. If the received data equals "END", it closes the listening socket as well as the connection socket, then it starts to listen again for incoming connection requests. If the received data does not equal "END", then it calls *SendDataToMicrocontroller()* function, which is described below. After that it clears the receiving buffer so that it can hold the next command that will be received and calls the *WaitForData()* function.

o. ProgramClosing(object , CancelEventArgs)

This function is called right before the program terminates. Basically it closes the serial port that was opened for communicating with the microcontroller.

p. SendDataToMicrocontroller()

This function is called whenever data is received from the PDA program. Basically it sends a special byte to the microcontroller through the serial port. The value of the byte depends on the data received from the PDA program, which is stored in the receiving buffer. If the request is to control any of the appliances other than the temperature, then a byte is sent to the microcontroller with the 4 least significant bits set to 1. However, if the received request is to control temperature, another function is called to handle that as explained below.

q. SendTempData()

This function is called when the PC has to send a byte representing a temperature. Since a temperature can be between 0-99, one byte can be efficient to represent the value of the temperature. And instead of having 100 different byte representations for each possible temperature value, I decided to represent the first digit (the tens digit) using the 4 most significant bits of the byte and to represent the second digit (the ones digit) using the 4 least significant bits of the byte. The byte is then sent to the microcontroller using the serial port connection.

Microcontroller and Hardware Connection

The microcontroller program and the hardware connection were mainly implemented by other members of the team. I will briefly describe the way the microcontroller and the hardware connection work. The PC program sends different commands to the microcontroller program based on the action of the PDA user. These commands are sent in the form of 1 byte. The microcontroller program has a special subroutine that keeps on checking if data has been received. When the data is received, the microcontroller performs different actions based on the value of the received byte. For the purpose of this project, our team decided to have different options to control.

The first option is setting the light brightness using three different frequencies generated from the microcontroller. These frequencies will be applied to a steady-state relay inserted in the extension cord that connections the appliance to the

power source. This will allow the microcontroller to have light be either off, low or high. The second action is to turn on the blender using a relay that completes the circuit connection between a blender and the power source. The third option is to control two LEDs, displaying the red one to represent that a door is locked and displaying a green one to represent that a door is unlocked. The forth option is to start playing a CD in a stereo, go to next track or stop playing the stereo. This is done through duplicating the stereo's remote control signal using the microcontroller. The last option is to display different temperature using two seven segment displays. The figure below shows the hardware connection between the microcontroller and the appliances.

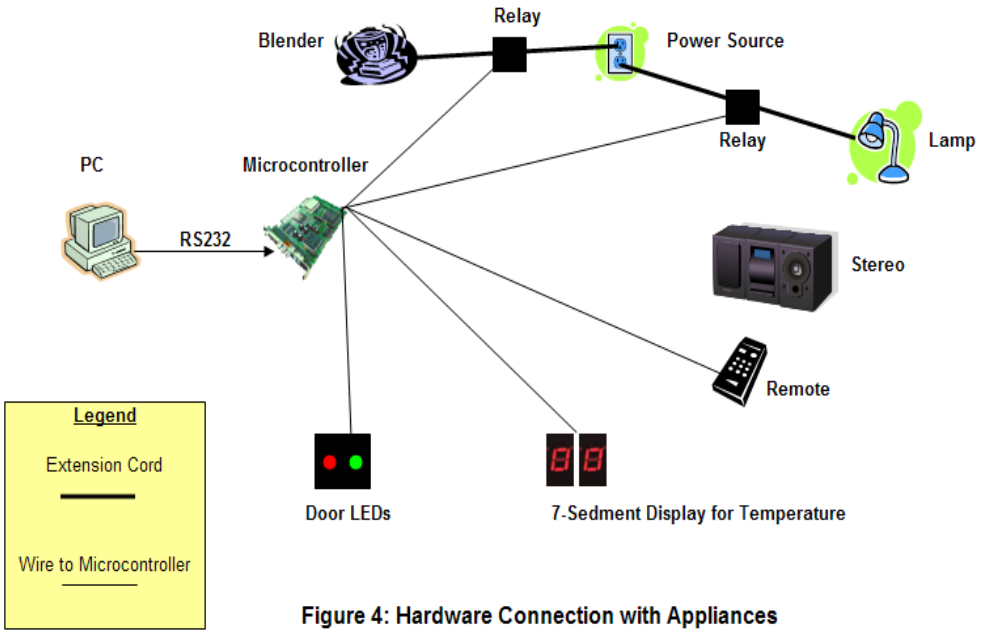

Figure 4: Hardware Connection with Appliances

How Can This Project Be Taken Further?

A reader might ask, how can this project be taken further? There are many answers to this question. One of the main features that can be added into this project is security features. In reality, a home control program will need authentication process that makes sure that only permitted users control the appliances at home. Another feature that can also be added is the two way communication capability between the devices. A user might be interested to know the status of an appliance before performing an action on it.

Another big feature that can be considered is to remove the PC program from the communication channel and make direct communication between the PDA and the microcontroller. Or even a better feature to consider is to have direct communication between the PDA and each of the appliances, considering the fact that there are many computerized appliances in market today. I'm not claiming that any of these features is feasible; I'm just giving ideas on how this system can be improved.

References

1. http://www.msdn.com
2. http://www.opennetcf.org
3. http://www.codeproject.com
4. Programming Microsoft Windows CE .NET by Douglas Boling
5. Pocket PC Network Programming By Steve Makofsky

Appendix A

Appendix B