Department of Computer Science and Engineering
The University of Texas at Arlington

Team: Auto-Climatix

Project: Automotive Climate Controller

Team Members:
  Raju Karki
  Robert Lopez
  Bishal Shrestha
  Hai Nguyen
  Anthony Vecera

Last Updated: 7/27/2012 11:21 AM
# Table of Contents

1 Introduction ............................................................................................................................ 5
1.1 Overview .............................................................................................................................. 5
1.2 Project Overview .................................................................................................................. 6

2 Architecture Overview ............................................................................................................. 7
2.1 Overview ............................................................................................................................... 8
2.1.1 Mobile Application Layer .............................................................................................. 8
2.1.2 Microcontroller Logic Layer ......................................................................................... 8
2.1.3 Signal Delivery Layer .................................................................................................... 8
2.2 Module Decomposition ........................................................................................................ 8
2.2.1 Overview ....................................................................................................................... 8
2.2.2 Module Decomposition Chart ........................................................................................ 9
2.2.3 Data Flows within the system ....................................................................................... 10
2.3 Module Descriptions .......................................................................................................... 13

3 Mobile Application Layer ..................................................................................................... 16
3.1 Design Overview .................................................................................................................. 16
3.2 GUI Subsystem ................................................................................................................... 17
3.2.1 GUI Handler .................................................................................................................. 17
3.3 Data Processor Subsystem .................................................................................................. 20
3.3.1 Profile ............................................................................................................................ 20
3.3.2 Profile Writer .................................................................................................................. 20
3.3.3 Profile Parser ................................................................................................................ 21
3.3.4 Climate Control Processor ............................................................................................ 22
3.4 Data Conversion Subsystem ............................................................................................... 26
3.4.1 Configuration file .......................................................................................................... 26
3.4.2 Config Parser .................................................................................................................. 27
3.4.3 Hexlogic Converter ....................................................................................................... 28
3.5 Mobile Communications Subsystem .................................................................................... 29
3.5.1 Hexbin Converter ............................................................................................................ 29
3.5.2 Thread Listener .............................................................................................................. 30

4 Microcontroller Logic Layer .................................................................................................. 31
4.1 Design Overview ................................................................................................................ 31
4.2 Microcontroller communication subsystem ......................................................................... 32
Detailed Design Specification
Automotive Climate Controller

4.2.1 RS232 Port ........................................................................................................................................ 32
4.2.2 MAX232 ......................................................................................................................................... 33
4.3 Signal Processor Subsystem .................................................................................................................. 35
4.3.1 Interfaces......................................................................................................................................... 35

5 Signal Delivery Layer .................................................................................................................................. 37
5.1 Design Overview .................................................................................................................................... 37
5.2 Signal Conditioner Subsystem .............................................................................................................. 37
5.2.1 12 VS ................................................................................................................................................ 38
5.2.2 0-12 Analog In ................................................................................................................................. 39
5.2.3 Ground .............................................................................................................................................. 39
5.3 Power Subsystem .................................................................................................................................... 41
5.3.1 C (Constant) ................................................................................................................................... 41
5.4 S (Switched) .......................................................................................................................................... 42

6 Quality Assurance ..................................................................................................................................... 43
6.1 Test Plans and Procedures .................................................................................................................... 43
6.1.1 Module/unit test ............................................................................................................................... 43
6.1.2 Component test ............................................................................................................................... 45
6.1.3 Integration test ................................................................................................................................. 46
6.1.4 System verification test .................................................................................................................... 47
6.2 Test Cases ............................................................................................................................................ 47

7 Requirements Traceability ....................................................................................................................... 48
7.1 Overview .............................................................................................................................................. 48
7.2 Architecture Requirements Mapping .................................................................................................. 48
7.3 Modules Requirements Mapping ......................................................................................................... 49

8 Acceptance Plan ....................................................................................................................................... 50
8.1 Packaging and Installation ................................................................................................................... 50
8.2 Acceptance Testing ............................................................................................................................... 50
8.3 Acceptance Criteria .............................................................................................................................. 50

9 Appendices ............................................................................................................................................... 51
List of Figures

Figure 1: Product Overview ................................................................. 6
Figure 2: Layer and subsystems decomposition chart ........................................ 7
Figure 3: Module Decomposition Chart .................................................. 9
Figure 4: Mobile Application Layer Components ........................................ 16
Figure 5: GUI subsystem ........................................................................ 17
Figure 6: Layout for home screen of mobile application ............................... 19
Figure 7: Data Processor Subsystem ...................................................... 20
Figure 8: Data Conversion Subsystem .................................................... 26
Figure 9: Mobile Communications Subsystem .......................................... 29
Figure 10: Microcontroller Logic Layer Components ................................... 31
Figure 11: Microcontroller communication Subsystem ............................... 32
Figure 12: RS232 Process ..................................................................... 33
Figure 13: MAX232 Process .................................................................. 34
Figure 14: Signal Processor Subsystem .................................................. 35
Figure 15: Signal Delivery Layer Components ......................................... 37
Figure 16: Signal Conditioner System .................................................... 37
Figure 17: Power Subsystem .................................................................. 41
Figure 18: Constant Module Process ..................................................... 41
Figure 19: Switch Module Process .......................................................... 42

List of Tables

Table 1: Descriptions of the data flowing within the system ......................... 10
Table 2: Producer-Consumer relationship of data elements in the system ....... 12
Table 3: Architecture Requirements Mapping .......................................... 48
Table 4: Module Requirements Mapping ................................................ 49
1 Introduction

1.1 Overview
This document illustrates and specifies the detailed design of the Automotive Climate Controller (ACC) project. The design follows the previously specified architecture, decomposing and refining previously defined subsystem and layers. The three layers specified in the architectures are the Mobile Application Layer, the Microcontroller Logic Layer and the Signal Delivery Layer. The three layers are additionally decomposed into appropriate subsystem. It is at these subsystems that the detailed design decomposes at a more significant level.

The decomposition of the subsystems into specific modules reflects logical groupings of functionality. Additionally, the decomposition explicitly defines data structures and data flows within and without the subsystems and layers. Special focus is made on the dependencies and algorithms associated with each module. Finally, the modules reflect implementation concerns and tradeoffs.

The document includes acceptance criterion concerns and requirements mapping, illustrating how critical requirements were taken into consideration in the decomposition of the subsystems. These acceptance criteria were explicitly defined in the baseline system requirements specification. The document concludes with a discussion of the testing considerations required for system and component verification.
1.2 Project Overview
The Automotive Climate Controller is a system that will act, as the name suggests, as a climate controller for automobiles. It will detect current climate conditions in a vehicle and allow the user to either manually adjust climate conditions or enable an automatic mode, allowing the system to calculate and adjust to current settings based on desired climate conditions. The system will be implemented by end-users with a portable Android 1.5 or newer device on their vehicles. Fundamentally, older analog systems will be replaced with a sleek new 'smart' Automotive Climate Controller.

Figure 1: Product Overview
2 Architecture Overview

![Layer and subsystems decomposition chart](image)

Figure 2: Layer and subsystems decomposition chart
2.1 Overview
The Automotive Climate Controller is a software and hardware system that is designed to allow users to control the disparate components of their vehicle’s air conditioning system through an application on their mobile device. In order to use this system, the majority of the vehicle’s existing air conditioning control system will be replaced by special hardware that will then be connected to the various components of the car. Once this hardware is properly installed within the vehicle, the mobile device will then be attached to the hardware through a serial cable, and this will allow the application to control the air conditioning control system. The whole system is broken down into three layers, which are described as following:

2.1.1 Mobile Application Layer
The Mobile Layer gets the user’s input via a mobile device, processes the user input and provides the appropriate output to the user after interacting with the microcontroller layer. This layer is in the form of the software in the mobile device of the user that is physically connected to the microcontroller.

2.1.2 Microcontroller Logic Layer
The Microcontroller logic layer acts as a logical translator between the Mobile Application Layer and the Signal Delivery Layer. This layer receives a formatted command signal from the Mobile Application Layer, converts it into a control signal, and delivers it to the Signal Delivery Layer. Similarly, this layer receives a status signal from the Signal Delivery Layer and creates a logical response signal to be relayed to the Mobile Application Layer. It should be noted that this layer is blind to the vehicle hardware, trusting that the commands it receives are accurate with respect to ports and signals.

2.1.3 Signal Delivery Layer
The Signal Delivery Layer interfaces with the Microcontroller Logic Layer via the Signal Processor Subsystem, which interfaces with the car. The Signal Delivery Layer consists of two subsystems: the Signal Conditioner Subsystem and power. The Signal Delivery Layer collects control signals from the Microcontroller Logic Layer and conditions them for use by the vehicle hardware. The Signal Delivery Layer also receives status and power signals from the vehicle’s hardware and prepares them for use in the other layers of the system.

2.2 Module Decomposition

2.2.1 Overview
Modules of Automotive Climate Controller System are created by breaking down layers and subsystem components into their individual functions and parts. Components may include internal hardware and architectural subsystems.
2.2.2 Module Decomposition Chart

Figure 3: Module Decomposition Chart
2.2.3 Data Flows within the system

<table>
<thead>
<tr>
<th>Data Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>User touch input</td>
</tr>
<tr>
<td>U2</td>
<td>Screen display</td>
</tr>
<tr>
<td>MD1</td>
<td>Call from climate control processor to write user profile</td>
</tr>
<tr>
<td>MD2</td>
<td>User requests with different parameters</td>
</tr>
<tr>
<td>MD3</td>
<td>System status information</td>
</tr>
<tr>
<td>MD4</td>
<td>Parsed user profile input</td>
</tr>
<tr>
<td>M3A</td>
<td>Reading of user profile file</td>
</tr>
<tr>
<td>M3B</td>
<td>Command to write user information to the user profile</td>
</tr>
<tr>
<td>M4A</td>
<td>Formatted logical climate data request</td>
</tr>
<tr>
<td>M4B</td>
<td>Logical climate data response</td>
</tr>
<tr>
<td>MDC1</td>
<td>Parsed vehicle hardware configuration and settings</td>
</tr>
<tr>
<td>MOB1</td>
<td>Binary signal listened by thread listener</td>
</tr>
<tr>
<td>M5</td>
<td>Vehicle hardware configuration and settings</td>
</tr>
<tr>
<td>M6</td>
<td>Formatted command and port signal</td>
</tr>
<tr>
<td>M7</td>
<td>Formatted port and status signal</td>
</tr>
<tr>
<td>M8</td>
<td>Formatted binary signal passing via a serial protocol</td>
</tr>
<tr>
<td>L1</td>
<td>Formatted binary status signal to the thread listener</td>
</tr>
<tr>
<td>L2</td>
<td>Data to denote the system status</td>
</tr>
<tr>
<td>L3</td>
<td>Data to denote the user command</td>
</tr>
<tr>
<td>MCC1</td>
<td>Data to denote the user command</td>
</tr>
<tr>
<td>MCC2</td>
<td>Data to denote the system status</td>
</tr>
<tr>
<td>L4, L5</td>
<td>Analog control signal</td>
</tr>
<tr>
<td>L6</td>
<td>Input selector</td>
</tr>
<tr>
<td>S1</td>
<td>Analog status input</td>
</tr>
<tr>
<td>S2A</td>
<td>12V Analog control signal</td>
</tr>
<tr>
<td>S2B</td>
<td>012V analog status signal</td>
</tr>
<tr>
<td>S3, S6</td>
<td>12V Source power</td>
</tr>
<tr>
<td>S4</td>
<td>12VS power</td>
</tr>
<tr>
<td>S5</td>
<td>Ground completion</td>
</tr>
<tr>
<td>S7</td>
<td>Ground</td>
</tr>
<tr>
<td>S8</td>
<td>5V power</td>
</tr>
<tr>
<td>S9</td>
<td>5VC power</td>
</tr>
</tbody>
</table>

Table 1: Descriptions of the data flowing within the system
<table>
<thead>
<tr>
<th>Producer</th>
<th>Consumer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Touch Input</td>
<td>GUI Handler</td>
</tr>
<tr>
<td></td>
<td>Profile Writer</td>
</tr>
<tr>
<td></td>
<td>Profile Parser</td>
</tr>
<tr>
<td></td>
<td>Profile</td>
</tr>
<tr>
<td></td>
<td>Climate Control Processor</td>
</tr>
<tr>
<td>Touch Input</td>
<td>U1</td>
</tr>
<tr>
<td>GUI Handler</td>
<td>U2</td>
</tr>
<tr>
<td>Profile Writer</td>
<td></td>
</tr>
<tr>
<td>Profile Parser</td>
<td></td>
</tr>
<tr>
<td>Profile</td>
<td></td>
</tr>
<tr>
<td>Climate Control Processor</td>
<td>MD2</td>
</tr>
<tr>
<td>Config File</td>
<td></td>
</tr>
<tr>
<td>Config Parser</td>
<td></td>
</tr>
<tr>
<td>Hexlogic Converter</td>
<td>M4B</td>
</tr>
<tr>
<td>Hexbin Converter</td>
<td></td>
</tr>
<tr>
<td>Thread Listener</td>
<td></td>
</tr>
<tr>
<td>Producer</td>
<td>Consumer</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Hexbin Converter</td>
<td></td>
</tr>
<tr>
<td>Thread Listener</td>
<td>MOB1</td>
</tr>
<tr>
<td>RS232 Port</td>
<td>L1</td>
</tr>
<tr>
<td>Max232</td>
<td>MCC2</td>
</tr>
<tr>
<td>Signal Processor</td>
<td></td>
</tr>
<tr>
<td>12VS Power</td>
<td></td>
</tr>
<tr>
<td>0-12V Analog In</td>
<td></td>
</tr>
<tr>
<td>Ground</td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td></td>
</tr>
<tr>
<td>Switch</td>
<td></td>
</tr>
<tr>
<td>Vehicle Hardware</td>
<td></td>
</tr>
<tr>
<td>Battery</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Producer-Consumer relationship of data elements in the system
2.3 Module Descriptions

GUI Handler Module

The GUI handler module is what the user interacts with on the touch screen of the mobile device. It displays the GUI components to the user and receives the input from the user. There are three types of touch inputs: pressing a button, pressing and holding a button, and sliding one’s finger across a slider GUI component.

In addition, this module acts as a controller to handle the user events, and passes user input to Climate Control Processor and Profile Writer modules.

Climate Control Processor Module

This module takes the input of system status information from the Hexlogic Controller, which provides the formatted data. It also receives the user commands from the user event controller. It combines those inputs and uses the climate control algorithm to maintain the climate control on the system.

Profile

This module stores the user information that includes the current system settings of the user and the username of the user.

Profile Writer Module

This module takes the command from the user event controller module to write the profile information to the profile. This occurs every time when there is a climate configuration change request from the user.

Profile Parser Module

This module takes the information from the profile module. It parses the information from the profile module to the format that the Climate Control Processor can use. This occurs when the system starts.

Config File

This module stores the configuration of the vehicle hardware and the port connection settings. The information from this file is loaded to the Climate Control Processor via config parser upon the startup of the system.

Config parser Module

This module takes the information from the Config File. It parses the information from the Config File, creates the logical object and passes it to the Climate Control Processor. This occurs for a single time when the system starts.

HexLogic Converter Module
This module converts hexadecimal numbers into logical objects. It can also convert a decimal number into a hexadecimal number.

**HexBin Converter Module**

This module converts hexadecimal numbers into a binary stream. It can also convert a binary stream into a hexadecimal number.

**Thread Listener Module**

This module constantly listens to the signal sent out from the RS232 port and relays it to the Hexlogic Converter module.

**RS232 Port**

The RS232 provides a physical connection to enable communications between the microcontroller logic layer and the signal processor.

**MAX232**

The MAX232 Module ensures that there is a unified serial protocol between the mobile device and the signal processor. It additionally synchronizes communications between the two devices.

**Signal Processor Module**

The Signal Processor Module takes input from the vehicle hardware through the signal delivery layer and the mobile device. It produces analog control PWM signals based on mobile device input and delivers status digital signals to the MAX232 modules.

**12VS Module**

The 12VS Module delivers a 12V output to a connected hardware device dependent on whether or not a 5V input is provided from the Signal Processor Modules.

**0-12V Analog Input Module**

The 0-12V Analog Input Module acts as a voltage divider, downscaling a maximum of 12V to 5V. The maximum 5V input is delivered to the Signal Processor Module.

**Ground Module**

The Ground Module connects specific components to ground depending on whether or not the Signal Processor Module is delivering a 5V activation signal to the circuit.

**Switch Power Module**

The Switch Power Module is a circuit that takes as 12V input from the vehicle battery. When the vehicle's key is turned to the 'power on position' the circuit is completed and provides 5V and 12V normalized voltage.

**Constant Power Module**
The Constant Power Module is a circuit that takes 12V input from the vehicle battery and provides 5V and 12V normalized voltage at all times.
3 Mobile Application Layer

3.1 Design Overview

This layer is the only layer that interfaces with the user. This layer will take input from external sources such as user input in order to control the hardware of the system. This layer will display system status information to the user. This layer also has the climate control algorithm that drives the whole system. It reads the configuration file and the user profile and has the required parser to read the information from the configuration file and the user profile file. It writes the information to the user profile too. It consists of four subsystems that are further divided into 10 modules and 2 files as described below:

**Figure 4: Mobile Application Layer Components**
3.2 GUI Subsystem

![GUI Subsystem Diagram](image)

The GUI subsystem displays the application interface to the end user through the touchscreen display in the mobile device. It displays the current configuration of the system and the readings of the sensors in the appropriate format. This subsystem also displays the different GUI components like buttons, scroll bars text input boxes and other required GUI components in order to take the input from the end users via the touchscreen of the mobile device.

3.2.1 GUI Handler

The GUI handler module handles GUI of the application, the control events, and passes user input to Climate Control Processor.

3.2.1.1 Interfaces

This module interfaces to the end user and gets the user touch input from the end user through the touch screen interface on the mobile device, displays the system information to the user and passes the user input to the Climate Control Processor.

3.2.1.2 Process

The components and the different pages to be displayed on the GUI of the mobile application are described mostly in the xml files in the mobile application. Some interactive components of the GUI like the number picker is drawn and updated from java files of the application. The following are the basic outlines of the code that composes the GUI of the mobile application:

3.2.1.2.1 Pseudocode for the Startup Screen/Automatic/Manual Mode

```xml
<RelativeLayout ... />
<Button android:id="@+id/Auto" />
```

//defines the layout of the application
//creates an actionListener when the user picks
Automatic mode
<Button
android:id="@+id/Manual" //creates an actionListener when the user picks
Manual mode
<Button
android:id="@+id/Debug" //creates an actionListener when the user picks
Debug mode
<TextView
android:id="@+id/Temp" //displays the current temperature of the system
<Button
android:id="@+id/Fd1" //creates an actionListener when the user picks Fan
direction 1
<Button
android:id="@+id/Fd2" //creates an actionListener when the user picks Fan
direction 2
<Button
android:id="@+id/Fd3" //creates an actionListener when the user picks Fan
direction 3
<Button
android:id="@+id/Fd4" //creates an actionListener when the user picks Fan
direction 4
<Button
android:id="@+id/Fd5" //creates an actionListener when the user picks Fan
direction 5
<Button
android:id="@+id/Fd6" //creates an actionListener when the user picks recirculation within the vehicle
<ToggleButton
android:id="@+id/Ac" //creates an actionListener to turn the ac on/off in
the system
<ToggleButton
android:id="@+id/Off" //creates an actionListener to turn the whole system on/off
<AC.Controller.NumberPicker //creates an actionListener to let the user
pick the desired temperature in automatic mode
android:id="@+id/numberPicker"
<SeekBar
android:id="@+id/FanSeekBar" //creates an actionListener to let the user
pick the fan speed
<ImageView
android:id="@+id/Fan" //an icon to denote the fan

3.2.1.2.2 Pseudocode for the Debug Mode
<RelativeLayout … //defines the layout of the application
<Button
android:id="@+id/Auto" //creates an actionListener when the user picks
**Automatic mode**

<Button
android:id="@+id/Manual" //creates an actionListener when the user picks

**Manual mode**

<Button
android:id="@+id/Debug" //creates an actionListener when the user picks

**Debug mode**

<TextView
android:id="@+id/Temp" //displays the current temperature of the system

<ToggleButton
android:id="@+id/Ac" //creates an actionListener to turn the ac on/off in
the system

<ToggleButton
android:id="@+id/Off" //creates an actionListener to turn the whole system
on/off

<TextView
android:id:="@+id/debugInfo" //displays the component information

---

**Figure 6:** Layout for home screen of mobile application
3.3 Data Processor Subsystem

![Diagram of Data Processor Subsystem]

Figure 7: Data Processor Subsystem

3.3.1 Profile
User configurations are stored in a text file, profile.txt. The data processor retrieves each line of the file and by basing on each period in the text, splits it into separated information correspond to intended order of the components. Therefore, the data processor can specify necessary parameters to recover previous status for the components of the air conditioning system.

3.3.1.1 Interfaces
The Profile file stores information under explicit text, which has a fixed and standard format. Information is separated by a period.

3.3.1.2 Physical data structure/data file descriptions
The information is arranged following:


The line below is an example:
“Jane. Automatic. 70. 60%. front”

3.3.2 Profile Writer
The system needs to store the current operating status of entire devices whenever a user changes any climate settings. The Profile Writer module will store the configuration of the air conditioning devices into the profile file, which will be retrieved anytime requested from a user.

3.3.2.1 Interfaces
Whenever user changes value of a device on the GUI, the Climate Control Processor module s a current status for the system and calls function writeProfile() from the Profile
Writer to save parameters of the devices corresponding to the user name. The Profile Writer module then bases on the current status to save the configuration of the devices into Profile file.

### 3.3.2.2 Physical data structure/data file descriptions

*String user name ➔* After updating the new status, the User Event Controller module calls function writeProfile() of the Profile Writer module and passes the user name for this function.

*Current Status ➔* Status class is stored current status of entire devices at runtime, and the User Event Controller subsystem will updates a new current status.

#### 3.3.2.3 Process

Void writeProfile( string username)

Begin

Make information string newline= “username. mode of current status. Temperature of current status. Fan speed of current status. Air flow direction of current status”;

Open User Profile file;

For each line (ex: “Jane. Automatic. 70. 60%. front”)

Begin

ArrayDevices = split line by a period (.);

If username matches ArrayDevices[0]

Begin

Delete this line;

End If

End For

Add newline to the User Profile file as a new line;

Close the User Profile file;

End function

### 3.3.3 Profile Parser

A user can request to set up the system with the previous status configuration of the air conditioning devices. The configuration was stored in the Profile by the user. The Profile Parser retrieves the user profile to load information of the air conditioning devices currently requested by the user.

#### 3.3.3.1 Interfaces

Whenever the User Event Controller subsystem requires a configuration of a particular user, it calls function loadProfile() from the User Profile Parser to load parameters of the devices corresponding to the user name which is supplied. The User Profile Parser then bases on the user profile file to set up and return necessary values of the devices for the User Event Controller.
3.3.3.2 Physical data structure/data file descriptions

String user name → the Climate Control Processor module calls loadProfile(string user name) and passes user name for this function.

Current Status → Status class is stored current status of entire devices, and function loadProfile(string user name) is called to change current status.

3.3.3.3 Process

Void loadProfile(String username)
Begin
Open profile file;
For each line (ex: “Jane. Automatic. 70. 60%. front”)
Begin
ArrayDevices = split line by a period (.);
If username matches ArrayDevices[0]
Begin
Mode of Current Status = ArrayDevices[1];
Temperature of Current Status = ArrayDevices[2];
Fan speed of Current Status = ArrayDevices[3];
Air flow direction of Current Status = ArrayDevices[4];
End If
End For
Close the Profile file;
End function

3.3.4 Climate Control Processor

This module takes the input of system status information from the Hexlogic Controller, which provides the formatted data. It also receives the user commands from the user event controller. It combines those inputs and uses the climate control algorithm to maintain the climate control on the system.

3.3.4.1 Interfaces

This module takes the input from the user event controller and the hexlogic controller module and user them in addition to the already present logical climate objects to make decisions on how to drive different hardware components and what to display to the user.

3.3.4.2 Physical data structure/data file descriptions

Logical Climatic Objects in the form of classes.

Enum mode{
        auto, manual, debug
    }
Enum fanDirection{
        HeadOnly, HeadNFoot, FootOnly, DefrostNFoot, Defrostonly, Recirculation
    }
Class UserDesiredSettings{
    Int desiredTemp;
    Int desiredFanSpeed;
}
mode;
fanDirection;
}
Class CurrentVehicleStatus{
    BlowerMotor;
    ComClutch;
    MonoValve;
    SwitchOverValve;
    Temp;
}
Class BlowerMotor {
    Int PortNo, Int CurrentSpeed
}
Class ComClutch{
    Int PortNo, Boolean status
}
Class MonoValve{
    Int portNo, int currentSpeed
}
Class SwitchOverValve{
    Int portNo, Boolean status
}
Class Temp{
    Int portNo, int currentTemp
}

3.3.4.3 Process
<Pseudocode>
//initialize the currentVehicleStatus class with the GUI user input or update the currentVehicleStatus class with the hexlogic converter
//desired configuration comes in from the user event controller that include the mode Switch (mode)
/
//get the temperature input from the user event controller
//set the fan direction to 2, which is front and bottom
//run the fan algorithm:
//get the temperature difference
int tempDiff= (userDesiredTemp - vehicleTemp)
if (mod(tempDiff) >=10)
    fanSpeed= 5//it is the 100% speed
else if(mod(tempDiff)>=8)
    fanSpeed= 4//it is the 80% speed
else if(mod(tempDiff)>=6)
    fanSpeed= 3//it is the 60% speed
else if(mod(tempDiff)>=4)
    fanSpeed= 2//it is the 40% speed
else if(mod(tempDiff)>0)
    fanSpeed= 1//it is the 20% speed
else
    fanspeed=0;
If(desiredTemp<currentTemp){
    currentVehicleStatus.setCompressorClutch(true);  //turn on the compressor clutch
}
else if(desiredTemp>currentTemp){
    currentVehicleStatus.setCompressorClutch(false);  //turn off the compressor clutch
}

//use mono valve algorithm
    if(tempDiff<-10)
        currentVehicleStatus.setMonoValve(100);  //set the mono valve to 100 percent
    else if(tempDiff<-8)
        currentVehicleStatus.setMonoValve(80);  //set the mono valve to 80 percent
    else if(tempDiff<-6)
        currentVehicleStatus.setMonoValve(60);  //set the mono valve to 60 percent
    else if(tempDiff<-4)
        currentVehicleStatus.setMonoValve(40);  //set the mono valve to 40 percent
    else if(tempDiff<-2)
        currentVehicleStatus.setMonoValve(20);  //set the mono valve to 20 percent
    else
        currentVehicleStatus.setMonoValve(0);  //turn off the mono valve

    //disable all GUI components except temperature control
}

Case Manual: {
    //get the temperature input from the user event controller
    //set the fan direction to the user desired direction
    If(desiredTemp<currentTemp){
        currentVehicleStatus.setCompressorClutch(true);  //turn on the compressor clutch
    }
    else if(desiredTemp>currentTemp){
        currentVehicleStatus.setCompressorClutch(false);  //turn off the compressor clutch
    }

    //use mono valve algorithm
    //get the temperature difference
    int tempDiff = (userDesiredTemp – vehicleTemp)

    if(tempDiff<-10)
        currentVehicleStatus.setMonoValve(100);  //set the mono valve to 100 percent
    else if(tempDiff<-8)
        currentVehicleStatus.setMonoValve(80);  //set the mono valve to 80 percent

else if(tempDiff<-6)
    currentVehicleStatus.setMonoValve(60);//set the mono valve to 60 percent
else if(tempDiff<-4)
    currentVehicleStatus.setMonoValve(40);//set the mono valve to 40 percent
else if(tempDiff<-2)
    currentVehicleStatus.setMonoValve(20);//set the mono valve to 20 percent
else
    currentVehicleStatus.setMonoValve(0);      //turn off the mono valve
//enable all GUI components
}
Case Debug: {
    //turn on all connected ports
    //get input from all connected ports
    //display that to the GUI by passing the parameters received for each parts
    //disable all GUI components

}
3.4 Data Conversion Subsystem

This subsystem converts the data coming in from the mobile communications subsystems and converts them to the appropriate format that can be used in the mobile data processor layer. Also, this converts the data coming from the data processor, so the mobile communications subsystem can use that. The information from the configuration file is loaded into it at the startup of the system so that it has the information of the port to hardware connection settings information that is passed to it via the configuration parser that only takes place during the beginning of the application.

![Data Conversion Subsystem Diagram]

Figure 8: Data Conversion Subsystem

3.4.1 Configuration file

Configuration file stores information of air conditioning devices in config.xml file. The data conversion subsystem will build a configuration of each air conditioning device by loading fundamental parameters of the device, and then assign necessary values for these.

3.4.1.1 Interfaces

Configuration file is a file enclosed with the application. It is stored under normal text format, which is compatible and recognized by Android operating system. The data conversion subsystem bases on the I/O library of the Android OS to access the configuration file.

3.4.1.2 Physical data structure/data file descriptions

The configuration file stores following information of air conditioning devices:
• Type of each component: information about what a component is, such as blower, switchover, clutch, etc.
• Ports of each component.
• Default values that are used when the program does not have explicitly a value for the device.
• Signal type and magnitude.

Configuration file is formatted xml. Elements of information thereby are formatted in xml tags. The data conversion subsystem accesses config.xml and analyze the xml tags to get information of the components.

3.4.1.3 Process
<Component>
  <Identifier code >08</>
  <Chip Port to control the component >RN6</>
  <Default value of the component >0101</>
  <Signal type to manage the component>PWM</>
</Component>

3.4.2 Config Parser
The Config Parser module is to build up information of the component classes from their fundamental parameters stored in the configuration file. The Hexlogic Converter module will pass information of a particular component to perform its functionality.

3.4.2.1 Interfaces
The Config Parser subsystem includes loadConfig() that allows other subsystem to get information of a component by passing an identifier code, and then the Config Parser retrieves the configuration file to load the parameters based on the identifier code. The parameters will be assigned to properties of the component classes used by the caller subsystem.

3.4.2.2 Physical data structure/data file descriptions

Integer identifier code → the Hexlogic Converter module calls loadConfig(int identifier code) and passes identifier code to this function.

Component class → Component class is created and returned to the caller, and function loadConfig is called to build this component class and its parameters.

3.4.2.3 Process
Component loadConfig (String identifier code)
Begin
  Component result = NULL;
  Array component = Load from configuration XML file;
  For each component in array component
  Begin
    If identifier code = identifier code of the component
Begin
    result = component;
    exit for loop;
End If
End For
Return result;

End function

3.4.3 Hexlogic Converter
The Hexlogic Converter converts logical data, such as temperature, into hexadecimal. It can also convert hexadecimal back into logical data.

3.4.3.1 Interfaces
The Hexlogic Converter receives input from the config parser and the climate control processor. The input from the config parser is a sequence of values that are intended to initialize the current state of the vehicle's hardware. The input from the climate control processor is a sequence of values that are intended to change the current state of the vehicle's hardware at the request of the user. The Hexlogic Converter sends hexadecimal to the Hexbin Converter, and it receives hexadecimal from the Hexbin Converter, as well. The Hexlogic Converter will convert any hexadecimal it receives from the Hexbin Converter into logical data.

3.4.3.2 Physical data structure/data file descriptions
3 Hexadecimal digits (2 bytes) as input from HexBin Converter

Boolean signalStatus, int tempValue

3.4.3.3 Process
<pseudocode>

//update the values in the currentVehicleStatus class by supplying the Boolean value for the signal status and the int value for the temperature value

// take in the command generated in the climate control process module and converts it into the hexadecimal form
3.5 Mobile Communications Subsystem

This subsystem communicates with the microcontroller via a serial connection that is connected from mini-USB port of the mobile device to RS232 port of the microcontroller. This receives the hardware status signals from the microcontroller and also sends the climate commands to the microcontroller from where they are relayed to the vehicle hardware. It also sends the signal received from the microcontroller to the signal conversion subsystem after converting it into the hexadecimal format that is further processed in the data conversion subsystem.

![Diagram of Mobile Communications Subsystem]

Figure 9: Mobile Communications Subsystem

3.5.1 Hexbin Converter

The Hexbin Converter reads the signal from the thread listener module. It converts hexadecimal into a binary stream. It can also convert binary streams back into hexadecimal.

3.5.1.1 Interfaces

The Hexbin Converter converts hexadecimal input into a binary stream. It then sends this binary stream to the RS232 module of the Microcontroller Logic Layer. The Hexbin Converter also receives binary streams from the RS232 module. It converts these streams back into hexadecimal and sends that hexadecimal to the Hexlogic Converter.

3.5.1.2 Physical data structure/data file descriptions

3.5.1.3 Process

```pseudocode
// convert the hexadecimal value to binary

//send the binary value to the RS232 port and hexlogic converter
```
3.5.2 Thread Listener
This takes responsibility for listening to the mini USB port of the mobile device. It creates a thread over the port and waits for any binary signal sent from the RS232, and then bases on the operating mode of the ACC application to decide whether or not it should pass the binary signal to the Hexbin Converter module.

3.5.2.1 Interfaces
The Thread Listener offers function startListen() for initializing a listening thread. The thread is created to listen to the binary signal as soon as the ACC application is started up by calling startListen(). Function startListen() sets a connection by requesting permission over the USB port, thereby the thread stands there to wait for the coming signal sent from the RS232.

To make the connection, the Thread Listener establishes a connection in USB Accessory mode, mAccessory, requests permission over mAccessory mode, and then creates the thread to listen in the mAccessory mode. In the thread’s run method, the Thread Listener can receive the input signal stream over the buffer of FileInputStream supplied by the I/O java.

3.5.2.2 Physical data structure/data file descriptions

UsbAccessory mAccessory; → android.hardware.usb.UsbAccessory, to establish connection

UsbManager mUsbManager; → android.hardware.usb.UsbManager, to manage and offer permission for the connection

3.5.2.3 Thread thread = new Thread ("UBS Host"); → creating a thread to listen to the binary signal in UBS Host mode.

3.5.2.4 thread.start(); → starting the thread.

FileInputStream mInputStream; → java.io.FileInputStream, the thread reads data into buffer of mInputStream

3.5.2.5 Process

Void startListen()
Begin
   Establish USB Accessory Connection: mAccessory;
   Register mAccessory to get permission;
   Create the thread over mAccessory;
   Open Accessory connection with FileInputStream mInputStream;
   Start the thread;
End

Void thread. Run ()
Begin
   While (true)
   Begin
      If (mInputStream is not empty)
4 Microcontroller Logic Layer

4.1 Design Overview

![Microcontroller Logic Layer Components Diagram]

This layer will take input from external sources such as user input in order to control the hardware of the system. This layer will also display system status information to the user.
4.2 Microcontroller communication subsystem

![Microcontroller communication subsystem diagram]

**Figure 11: Microcontroller communication Subsystem**

4.2.1 RS232 Port
The RS232 port provides a physical connection to enable communications between the microcontroller logic layer and mobile device. It transfers a single byte of data over a serial transmission method where bytes of data are output one bit at a time onto a single wire.
4.2.1.1 Interfaces

The RS232 Port receive two Tx signals, digital binary streams, from the MAX232 Module and mobile device and transmits them to the respective RX ports of the MAX232 and mobile device. The RS232 Port additionally takes power input directly from the Constant power module and delivers the power to the mobile device. The RS232 port defines the voltage levels that correspond to logical 0 and logical 1 level for the data transmission. So, the valid signal are + or – 3 to 15 volts.

4.2.1.2 Physical data structure/data file descriptions

Data Transmission (Rx/Tx) logic 0 -> +3 V to +15 V

Data Transmission (Rx/Tx) logic 1 -> -3 V to -15 V

4.2.1.3 Process

The RS232 port receive signal from MAX232 and send it mobile system. At the same it also receive signal from mobile system and send it to MAX232.

Figure 12: RS232 Process

4.2.2 MAX232

The MAX232 port enables communication between RS232 and signal processor subsystem by providing RS232 voltage level within the rage from 0 V to 5 V.
4.2.2.1 Interfaces

The MAX232 Port receive two Tx signals, digital binary streams, from the RS232 Module and signal processor subsystem and transmits them to the respective RX ports of the RS232 and signal processor subsystem. Basically, It receive 0 V to 5 V logic bit stream from the output pin of the microcontroller or Universal Asynchronous Receiver/Transmitter (UART) and boost it to the required voltage of the RS232 by using 5V power supply. At the same time it also receives RS232 input and reduces that input to standard 5V levels.

4.2.2.2 Physical data structure/data file descriptions

Data Transmission (Rx/Tx) logic 0 -> +3 V to +15 V – 0 V

Data Transmission (Rx/Tx) logic 1 -> -3 V to -15 V – 5 V

4.2.2.3 Process

The MAX232 module receive logic bit stream signal from the output pin of the microcontroller or UART and transmitted to the RS232 module and vice-versa.

![Diagram of MAX232 Process]

Figure 13: MAX232 Process
4.3 **Signal Processor Subsystem**

The Signal Processor Module takes input from the vehicle hardware through the signal delivery layer and the mobile device. It produces analog control PWM signals based on mobile device input and delivers status digital signals to the MAX232 modules.

![Signal Processor Subsystem Diagram](image)

**Figure 14: Signal Processor Subsystem**

### 4.3.1 Interfaces

The Signal Processor interfaces with the MAX232 port and individual signal conditioning modules. It receives binary encoded control signals from the MAX232 port, decodes them and transmits them as analog PWM signals to the signal conditioning modules. It receives analog input from the 0-12V Analog Input module and encodes them into a 4 bit binary value and transmits them to the MAX232 port.

#### 4.3.1.1 Physical data structure/data file descriptions

Digital Control Input and Status Output \(\rightarrow\) 8-bit value. Low bits (first four) identify port input or port output. High bits (last four) identify PWM control signal or Status signal. Only six possible outputs exist, 0, 20%, 40%, 60%, 80% and 1.

Status Input \(\rightarrow\) 0-5V Analog Input

Input Selector Output \(\rightarrow\) 4-bit value identifying which port on 16-bit analog multiplexer to take input from.
Control Output → 0.5V Digital Output.

4.3.1.2 4.1.4.3 Process
Void signalProcessing()
Begin

Initialize all input and output ports to 0, RA1 = 0, RA2 = 0... RD12 = 0.
Initialize selector to zero
Initialize counter i = 0
Set digital input wait to 8 // for one byte

While(true)
Begin

If digital input.low bits = 0
RA1 = digital input.high bits
Else if digital input.low bits = 1
RA2 = digital input.high bits
...
Else
RD12 = digital input.high bits

If i >= RA1
RA1.output = 0
...
If i >= RD12
RD12.output = 0
if i < 5 then i++
else i = 0

Selector.output
Analog input = MUX input
Digital Status.low bits = Selector
Digital Status.high bits = Analog input

Selector++;
If Selector = 16 then Selector = 0;
End While
End Function
5 Signal Delivery Layer

5.1 Design Overview

This layer will take input from external sources such as user input in order to control the hardware of the system. This layer will also display system status information to the user.

5.2 Signal Conditioner Subsystem

Figure 15: Signal Delivery Layer Components

Figure 16: Signal Conditioner System
5.2.1  12 VS

The 12VS Module delivers a 12V output to a connected hardware device dependent on whether or not a 5V input is provided from the Signal Processor Modules.

5.2.1.1 Interfaces

The 12VS Module interfaces with the Switch Power module, the Signal Processor module and the actual vehicle hardware. It takes 0 or 5V input and outputs 0 or 12V to the vehicle. Additionally, it takes a 12V input to ensure that 12Vs can be provided to the vehicle. Finally, 5V input is provided to drive the opto-coupler to isolate the Signal Processor from high voltages.

5.2.1.2 Physical data structure/data file descriptions

Digital Control Input → 0,5V Digital Input

12V Switch Input → 12V Normalized Power input

5V Switch Input → 5V Normalized Power input

Digital Output → 0,12V Digital Output

5.2.1.3 Process
5.2.2 0-12 Analog In

The 0-12V Analog Input Module acts as a voltage divider, downscaling a maximum of 12V to 5V. The maximum 5V input is delivered to the Signal Processor Module.

5.2.2.1 Interfaces

As the module's name suggests, the 0-12V Analog Input Module accepts 0-12V analog input from the vehicle hardware. It scales the input to a 0-5V output to the Signal Processor. It additionally receives input from the signal processor, determining which of the 16 input ports on the 16 channel analog multiplexer to take input from.

5.2.2.2 Physical data structure/data file descriptions

0-12V Device Input → 0-12V Analog Power

Multiplexer Select → 4-Bit Digital Value distributed across four MUX selector channels

0-5V Status Output → 0-5V Analog Power

5.2.2.3 Process

5.2.3 Ground

The Ground Module connects specific components to ground depending on whether or not the Signal Processor Module is delivering a 5V activation signal to the circuit.

5.2.3.1 Interfaces

The Ground Module interfaces with the Switch Power module, the Signal Processor module and the actual vehicle hardware. It takes 0 or 5V input and completes the vehicle hardware's circuit to the ground depending on whether or not 5V input is received from the Signal Processor module. Additionally, it is connected to battery ground to complete the grounding module. Finally, it takes 5V input to drive an opto-coupler in order isolate the Signal Processor module from high voltages.
5.2.3.2 Physical data structure/data file descriptions

Digital Control Input $\rightarrow$ 0,5V Digital Input

5V Switch Input $\rightarrow$ 5V Normalized Power input

5.2.3.3 Process
5.3 Power Subsystem

5.3.1 C (Constant)

The constant module communicates with MAX232 port by providing 5V constant from the voltage source of the vehicle. This module also prevents the circuit from high voltage, negative voltage and short circuit.

5.3.1.1 Interfaces

The constant module receive 12 V static from the voltage source of the vehicle and convert it to 5 V constant and send it to MAX232 port.

5.3.1.2 Physical data structure/data file descriptions

12 V static -> 5V constant

5.3.1.3 Process

Figure 17: Power Subsystem

Figure 18: Constant Module Process
5.4 S (Switched)

The Switch communicate with 12VS of signal conditioner subsystem and signal processor subsystem by providing 12V constant and 5 V constant respectively from the voltage source of the vehicle. This module also prevents the circuit from high voltage, negative voltage and short circuit.

5.4.1.1 Interfaces

The switch receives 12 V static voltage from the voltage source of the vehicle and convert it to constant voltage of 5 V and 12 V. The 12 VS of signal conditioner subsystem receive constant 12 V and signal processor of microcontroller logical layer receives constant 5 V from the constant module.

5.4.1.2 Physical data structure/data file descriptions

12 V static -> 5 V constant

12 V static -> 12 V constant

5.4.1.3 Process

![Figure 19: Switch Module Process](image-url)
6 Quality Assurance

6.1 Test Plans and Procedures
This section will cover the test plans and procedures required to ensure the quality of the Automotive Climate Controller. This plan is established to outline all architectural testing at a high level.

6.1.1 Module/unit test

6.1.1.1 Mobile GUI Handler Test
- Use each GUI component to make sure they perform the processes that is supposed to be performed.
  - Ensure the navigation between different modes is obtained by manual, auto and debug buttons.
  - Ensure the buttons: AC on/off and system on/standby activate appropriate methods in climate control processor.
  - Ensure the temperature input text box takes in the input and calls the appropriate method in climate control processor while in automatic and manual mode.
  - Ensure the temperature text view, debug text view are continuously refreshed with the update from the climate control processor.

6.1.1.2 Profile Writer
- Call function writeProfile(user name) with passing a user name
- Check file profile.txt existence of the user name and compare other parameters with values supposed to be written.

6.1.1.3 Profile Parser
- Call function loadProfile(user name) with passing a user name existed in profile.txt
- Verify the values loaded into parameters with values stored in profile.txt

6.1.1.4 Climate Control Algorithm
- Call the appropriate functions to ensure that the various processes of the Automotive Climate Controller are called by the proper user commands.
• Call the algorithm method to ensure that appropriate action is taken in different modes of operation of the system.

• Verify the profile writer is called whenever there is change in the current vehicle status.

• Verify that the update from the hexlogic converter is being received in regular interval of time.

6.1.1.5 Config Parser
• Pass a code of a component into function loadConfig
• Check the returned component class if it is initialized and it includes values corresponding to information of the component supplied by the configuration file.

6.1.1.6 Hexlogic Converter
• Convert a logical data object, such as temperature, into a hexadecimal number.
• Convert a hexadecimal number back into a logical data object.

6.1.1.7 Hexbin Converter
• Convert a hexadecimal number to a binary stream.
• Convert a binary stream back into a hexadecimal number.

6.1.1.8 Thread Listener
• Randomly send a signal from the Microcontroller to the Mobile device.
• Check the input stream received by the thread in the Thread Listener, and verify the data of input stream to ensure that the transmitted signal will be received precisely.

6.1.1.9 Signal Processor Module
• decodeSignal(byte Signal) properly decodes and assigns signal strength to appropriate port
• deliverSignal() properly delivers all signals on each port
• chooseInput(char Selector) properly activates the correct channel of input from the multiplexer
6.1.1.10 12 VS Module

- Activates upon receiving a 5V control signal
- Protects the microcontroller from >5V fly-back voltage
- Delivers 12V to the desired components

6.1.1.11 0-12V Analog Input Module

- Accepts selector input and properly chooses an input attached to the multiplexer
- Divides input voltage from 12V max to 5V max
- Protects the multiplexers from >5V input

6.1.1.12 Ground Module

- Activates upon receiving a 5V control signal
- Protects the microcontroller from >5V fly-back voltage
- Completes desired component circuits with ground

6.1.2 Component test

- **Microcontroller**
  - Verify Programmability and Communicability
  - Verify PWM signal generation
  - Verify MUX input selection
  - Verify control signal decoding and status signal encoding

- **Mobile Device**
  - Verify compatibility with microcontroller
  - Verify communications with microcontroller
  - Verify proper control signal encoding and status signal decoding
  - Verify user input acceptance

- **Switch Power**
  - Verify 12V and 5V normalized power
  - Verify Startup triggered on car switch settings

- **12V Constant Power**
  - Verify 12V and 5V normalized power

- **Battery**
  - Verify 12V power supply

*The following items are for demonstration purposes only.*

- **Switchover Valve**
  - Verify activation and deactivation from '100%' and '0%' PWM signals
  - Verify input to MUX
• **Monovalve**  
  o Verify proper movement from PWM signal  
  o Verify input to MUX  
• **Blower Motor**  
  o Verify proper adjustment from PWM signal  
  o Verify input to MUX  
• **Compressor Clutch**  
  o Verify activation and deactivation from '100%' and '0%' PWM signals  
  o Verify input to MUX  
• **Temperature Sensor**  
  o Verify input to MUX

6.1.3 **Integration test**

6.1.3.1 **Mobile Application Layer**  
  • The layer shall take user touch input and process the input correctly.  
  • The layer shall send hexadecimal signals to the microcontroller logic layer.  
  • The layer shall decode hexadecimal signals that were sent from the microcontroller logic layer back into logical signals.  
  • The layer shall take input from a configuration file correctly.  
  • The layer shall write to a Profile correctly.  
  • The layer shall take input from a Profile correctly.

6.1.3.2 **Microcontroller Logic Layer**  
  • The layer shall send digital signals to the mobile application layer.  
  • The layer shall send analog signals to the signal delivery layer.  
  • The layer shall send binary status signal to the mobile application layer.  
  • The layer shall convert binary status signal to appropriate voltage.  
  • The layer shall send logical bit signal to microcontroller.

6.1.3.3 **Signal Delivery Layer**  
  • The layer shall send power to the mobile device in the appropriate amounts.  
  • The layer shall send power to the microcontroller logic layer in the appropriate amounts.  
  • The layer shall send analog signals to the vehicle's hardware.  
  • The layer shall send condition analog signals correctly.
The layer shall protect the system from high voltage, negative voltage or short circuit.

The layer shall convert static voltage to constant voltage.

6.1.4 System verification test

The System Verification test will verify that the Automotive Climate Controller has carried out and met all of the high priority requirements as stated in the SRS.

6.2 Test Cases

- Loading a Configuration file
- Debug Mode
- Manual Mode
- Automatic Mode
- Loading a Profile
7 Requirements Traceability

7.1 Overview

One or more modules from the Detailed Design along with our Architectural Design Specification cover requirements detailed in the System Requirements Document. The tables below show how each of the requirements is traced to its relevant modules and layer as divide in this document and the Architecture Design Document.

7.2 Architecture Requirements Mapping

<table>
<thead>
<tr>
<th>Acceptance Criterion</th>
<th>Mobile Application Layer</th>
<th>Microcontroller Logic Layer</th>
<th>Signal Delivery Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile Device Implemented</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microcontroller Connected to Mobile Device</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Automatic/Manual/Debugging Mode</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programmable, Real-time Responding Microcontroller</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Controls the vehicle hardware interface with the microcontroller</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Fly Back Circuits Implemented</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 3: Architecture Requirements Mapping
## 7.3 Modules Requirements Mapping

<table>
<thead>
<tr>
<th>Modules</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mobile Device Implemented</td>
</tr>
<tr>
<td>GUI Handler</td>
<td>X</td>
</tr>
<tr>
<td>GUI Listener</td>
<td>X</td>
</tr>
<tr>
<td>User Event Controller</td>
<td>X</td>
</tr>
<tr>
<td>Profile Writer</td>
<td>X</td>
</tr>
<tr>
<td>Profile Parser</td>
<td>X</td>
</tr>
<tr>
<td>Profile</td>
<td>X</td>
</tr>
<tr>
<td>Climate Control Processor</td>
<td>X</td>
</tr>
<tr>
<td>Configuration File</td>
<td>X</td>
</tr>
<tr>
<td>Configuration Parser</td>
<td>X</td>
</tr>
<tr>
<td>Hexlogic Converter</td>
<td>X</td>
</tr>
<tr>
<td>Hexbin Converter</td>
<td>X</td>
</tr>
<tr>
<td>Thread Listener</td>
<td>X</td>
</tr>
<tr>
<td>RS232 Port</td>
<td>X</td>
</tr>
<tr>
<td>Max232</td>
<td>X</td>
</tr>
<tr>
<td>Signal Processor</td>
<td>X</td>
</tr>
<tr>
<td>12VS Power</td>
<td>X</td>
</tr>
<tr>
<td>0-12V Analog In</td>
<td>X</td>
</tr>
<tr>
<td>Ground</td>
<td>X</td>
</tr>
<tr>
<td>Constant</td>
<td>X</td>
</tr>
<tr>
<td>Switch</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 4: Module Requirements Mapping
8 Acceptance Plan

8.1 Packaging and Installation
The final Auto-Climatix Package shall consist of the following items:

- Hardware:
  - A microcontroller Board.
  - A wire to connect between the microcontroller boards to the Android device.
  - A CD includes the application.

- Software:
  - Android Application for the Mobile device Device.

- Others:
  - The user guideline book to support installation and maintenance the product.

8.2 Acceptance Testing
The Auto-Climatix system acceptance testing shall be priority to ensure that product meets all the requirements to be accepted. The acceptance criteria for the product shall go by a critical testing process from the unit testing to the system testing which will be run successfully over the sophisticated test cases. However, the Black box testing shall be held to ensure productivity and effectiveness after the product passes over entire test cases.

8.3 Acceptance Criteria
ACC System must meet the following requirements agreed upon by all the stakeholders involved in the project:

- The user shall have a mobile device.
- The mobile device shall have touch screen capability
- The ACC application shall be compatible with the user’s mobile device running Android OS version 3.1 or later
- The ACC application shall have the sleep mode
- The ACC application shall have a friendly user interface
- The mobile application shall have a GUI for the user to interact with, thereby allowing them to view the information of the AC devices.
- The MC shall connect to a vehicle reliable
- The ACC ensures a reliable connection between the ACC application and the MC
9 Appendices
UTA – The University of Texas at Arlington
ACC – Automotive Climate Controller
GUI – Graphical User Interface
OS – Operating System
AC – Air Conditioning
Auto-Climatix – Name of the development team