

DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES

A HOME NETWORKING APPLICATION FOR
CONTROLLING HOUSEHOLD APPLIANCES

by
Olgun ÜNAL

October, 2007
İZMİR

A HOME NETWORKING APPLICATION FOR CONTROLLING HOUSEHOLD APPLIANCES

**A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Master of Science in
Computer Engineering, Computer Engineering Program**

**By
Olgun ÜNAL**

**October, 2007
İZMİR**

M.Sc. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled “**A HOME NETWORKING APPLICATION FOR CONTROLLING HOUSEHOLD APPLIANCES**” completed by **OLGUN ÜNAL** under supervision of **INS. DR. M. KEMALŞİŞ** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

INS. DR. M. KEMAL ŞİŞ

.....
Supervisor

PROF. DR. HALDUN KARACA

(Jury Member)

ASSC. PROF. DR. ADİL ALPKOÇAK

(Jury Member)

Prof.Dr. Cahit HELVACI

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

I would like to thank my supervisor, “Ins. Dr. M. Kemal ŞİŞ” for his support and patience.

My great thanks to my wife for patiently motivating me to finalize this thesis on time and I appreciate her supreme technical help to build up the application of the thesis.

I also would like to thank my parents for supporting me in my whole education life.

A HOME NETWORKING APPLICATION FOR CONTROLLING HOUSEHOLD APPLIANCES

ABSTRACT

In this thesis, a consumer electronics product that is related with Home Networking is considered to be built up with a special software and hardware application combination.

The software application consists of three parts, which are independent each other and communicating via dedicated communication protocols; one is Embedded Control Server's embedded software, which is developed from the scratch with using Keil C compiler and development environment on the 8051 platform in ANSI C programming language. Second part is at the PC side that provides an HTTP server and a Web Based User Interface to the outside world via internet. In addition, it stores all the inputs from the WBUI in to a database. This software application is developed in Java programming language by Eclipse development environment from Apache Software Foundation. In addition, the last part is the port access application at PC side that can send and receive data via the serial port and update the common database with the web server. This application is also developed in Java by Eclipse IDE

The hardware application includes a demo board of the Keil Software that has 8051 microcontroller and the device representative units with port expander integrated circuits that are controlled via I2C protocol.

With using the combination of developed software and hardware applications above I have built up a Home Control Network, which provides ability to user to access a device at home from any internet accessed PC in the world. This application is intentionally built up very flexible to give chance to user to extend the application easily. To do this, user can change parameters provided via web interface. Also the

application can be developed according to the user requests and can be downloaded to ECS easily via same communication interface that port application uses.

Keywords : Home Network, Control, Embedded Systems, 8051, I2C, port expander, UART, protocol definition, Web Server, port control

EV CİHAZLARININ BİR EV AĞI UYGULAMASIYLA YÖNETİLMESİ

ÖZ

Bu tezde özel bir yazılım ve donanım kombinasyonu ile oluşturulan bir ev kontrol ağı uygulaması ele alınmış olup bu uygulama bir tüketici elektroniği ürünü haline getirilmiştir.

Yazılım uygulaması 3 bağımsız bölümün haberleşmesi esasına dayanmaktadır. Birinci yazılım, gömülü kontrol sunucusu (ECS) içerisinde yer alıp, uygulama geliştirme süreci tamamen sıfırdan başlayarak Keil C derleyici ve geliştirme ortamı kullanılarak tamamlanmıştır. Ayrıca bu yazılım daha sonra kullanıcının isteğine göre değiştirilip mevcut haberleşme kanalları üzerinden kolaylıkla yazılım güncellemesi yapılabilmektedir. İkinci yazılım PC tarafında yer alıp bir HTTP sunucu görevi yapmaktadır. Bu sunucu ev ağını dış dünyaya bağlamakta ve ev ağının kontrolü için bir kullanıcı arayüzü sunmaktadır. Aynı zamanda bu kullanıcı arayüzünden girilen veriler bir veritabanında saklanmaktadır. Bu yazılım Java programlama dili ile Eclipse entegre geliştirme ortamında geliştirilmiştir. Üçüncü yazılım yine PC tarafında olup, web server yazılımıyla ortak veritabanındaki değişiklikleri PC'nin seri portu üzerinden ECS'e gönderme ve ECS'den seri port üzerinden gelen verileri ortak databasede güncelleme görevlerini yapmaktadır. Bu yazılım da Java programlama dili kullanılarak Eclipse ortamında geliştirilmiştir.

Donanım uygulaması bir adet 8051 mikroişlemci uygulama kartı ve ev cihazlarını temsil etmek için kullanılan port genişletme entegre devrelerinden kurulmuş 3 adet ünitelerden oluşmaktadır. Bu ünitelerle haberleşme bir IC kontrol standardı olan I2C ile yapılmaktadır.

Yukarıda sayılan yazılım ve donanım bileşenlerinin kombinasyonu yardımıyla bir ev ağı yönetim uygulaması yapılmıştır. Bu uygulama kullanıcıya dünyanın herhangi bir yerinden internet bağlantısı olan bir PC aracılığıyla ev ağını yönetme imkanı vermektedir.

Anahtar kelimeler: Ev ađı, kontrol, gml sistemler, 8051, I2C, port geniřletme, UART, protokol tanımlama, web sunucu, port kontrol

CONTENTS

	Page
THESIS EXAMINATION RESULT FORM	ii
ACKNOWLEDGMENTS	iii
ABSTRACT.....	iv
ÖZ.....	vi
CHAPTER ONE - INTRODUCTION.....	1
1.1 Home Networking Concept.....	1
1.2 Home Networking Systems.....	2
1.2.1 Embedded Controller Based System.....	2
1.2.2 PC Centric Systems.....	2
1.3 Home Networking Applications.....	2
1.3.1 Implemented Applications.....	3
1.3.1.1 Simply Run.....	3
1.3.1.2 Simply Stop.....	3
1.3.1.3 Retrieve Significant Data.....	3
1.3.1.4 Write Significant Data.....	3
1.3.1.5 Time Application.....	3
1.3.1.6 Relative Application.....	3
1.3.2 Embedded Control Protocol (ECP)	4
1.4 General Home Networking Applications	4
1.4.1 Controlling Household Devices.....	4
1.4.2 Computing and Resource Sharing.....	4
1.4.3 Entertainment and Information.....	5

1.4.4 Communications	5
----------------------------	---

CHAPTER TWO - DESIGN BASIS OF THE SOFTWARE APPLICATION..6

2.1 Embedded Control Server (ECS).....	6
2.1.1 User Interface.....	7
2.1.2 Application.....	8
2.1.3 System.....	8
2.1.4 Drivers.....	8
2.1.5 Datalink.....	10
2.1.5.1 I2C.....	10
2.1.5.2 UART.....	11
2.1.6 Physical Layer – Hardware.....	12
2.1.7 Architecture of the 8051 Microcontroller.....	13
2.1.7.1 Memory Organization.....	13
2.1.7.2 Bit processing and Boolean logic.....	14
2.1.7.3 Addressing Modes.....	14
2.1.7.4 Interrupts on the 8051.....	14
2.1.7.5 On-Board Timer/Counters.....	14
2.1.7.6 On-Board UART.....	15
2.1.7.7 I2C.....	15
2.1.7.8 Analog to Digital Converters.....	15
2.1.7.9 Watchdog Timers.....	16
2.1.7.10 Using C with the 8051.....	16
2.2 Embedded Control Client / Web Server.....	17
2.2.1 Web Based Used Interface (WBUI)	18
2.2.2 Web Server Application and database link.....	20
2.2.3 Control Application and database link.....	20
2.2.4 Device Message Database (DMDB).....	21

CHAPTER THREE - HARDWARE ARCHITECTURE.....22

3.1 Embedded Control Server (ECS)..... 22

 3.1.1 Microcontroller..... 22

 3.1.1.1 EEPROM..... 22

 3.1.1.2 Real Time Clock..... 23

 3.1.1.3 General Purpose I/O Port..... 23

 3.1.1.4 UART interface..... 23

 3.1.1.5 I2C interface..... 23

 3.1.1.6 Memory..... 24

 3.1.1.7 Processor..... 24

 3.1.2 Household Interface Devices (Port Expander)..... 24

3.2 Embedded Control Client and Web Server (Computer Workstation)25

 3.2.1 Central Processor Unit (CPU).....

 26

 3.2.2 Ethernet Adapter..... 26

 3.2.3 USB Controller..... 26

CHAPTER FOUR - THE PROTOCOLS USED IN THE APPLICATION..... 28

4.1 Embedded Server Protocol (ECP).....28

 4.1.1 Headers:.....29

 4.1.1.1 Read / Write byte.....29

 4.1.1.2 Address byte.....29

 4.1.1.3 ASD0 (Application Code)..... 30

 4.1.1.4 ASD1 (Second)30

 4.1.1.5 ASD2 (Minute)30

 4.1.1.6 ASD3 (Hour)30

 4.1.1.7 ASD4 (Day)31

4.1.1.8	ASD5 (Month)31
4.1.1.9	ASD6 (Year)31
4.1.1.10	ASD7 and ASD8 (Time Offset and Unit) 31
4.1.1.11	ASD9 (Relative Device) 32
4.1.1.12	ASD10 (Relative Value) 32
4.1.1.13	ASD11 (Relative State)32
4.1.1.14	ASD12 (Device Pin number)33
4.1.1.15	ASD13 (Device Pin state)33
4.1.1.16	ASD14, ASD15 (Reserved bytes) 33
4.1.2	Fetching ECP Messages34
4.1.3	Processing ECP Messages35
4.2	Used Protocols at ECC Side36
4.2.1	ODBC37
4.2.1.1	JDBC-ODBC bridges37
4.2.1.2	ODBC-JDBC bridges38
4.2.2	TCP/IP39
4.2.2.1	Network Interface Layer39
4.2.2.2	Internet Layer40
4.2.2.3	Transport Layer40
4.2.2.4	Application Layer41
4.2.2.5	Apache Tomcat HTTP Server41
CHAPTER FIVE - DEVELOPMENT ENVIRONMENTS	42
5.1	ECS Application Development Environment 43
5.2	ECC Application Development Environment44
CHAPTER SIX - APPLICATION PRODUCT SPECIFICATIONS	 46

6.1	Representating the Product and Units.....	46
6.2	Predefined Applications.....	49
6.2.1	Simply Run.....	49
6.2.2	Simply Stop.....	49
6.2.3	Retrieve Significant Data.....	50
6.2.4	Write Significant Data.....	50
6.2.5	Time Application.....	50
6.2.6	Relative Application.....	51
CHAPTER SEVEN - CONCLUSION.....		52
REFERENCES.....		53
APENDIX -ABBREVIATIONS.....		54

CHAPTER ONE

INTRODUCTION

1.1 Home Networking Concept

Home networking is the collection of elements that process, manage, transport, and store information, enabling the connection and integration of multiple computing, control, monitoring and communication devices in the home.

Until recently, the home network has been largely ignored. However, the rapid proliferation of personal computers (PCs) and the Internet in most of the homes, advancements in telecommunications technology, and progress in the development of smart household devices have increasingly emphasized the last 100 feet of any consumer-related network

Furthermore, as these growth and advancement trends continue, the need for simple, flexible, and reliable home networks will greatly increase.

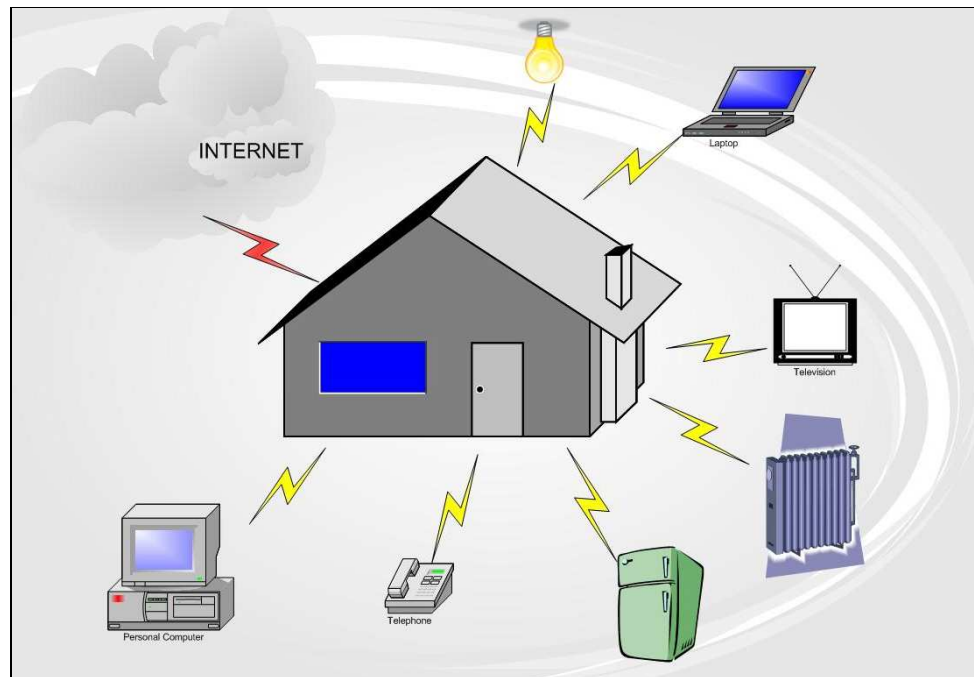


Figure 1.1 A generic home networking concept.

1.2 Home Networking Systems

1.2.1 Embedded Controller Based System

In this system, the microprocessor-based digital switch acts as the communications server, addressing and routing data traffic throughout the home.

I used a special protocol that I suggested in the application called Embedded Control Protocol (ECP), which handles the communication between the Embedded Control Server (ECS) and Embedded Control Client (ECC). ECC can be based on a standard PC system.

1.2.2 PC Centric Systems

This is an alternative technology that can be used as well for Home Networking Control. Some software and hardware suppliers provide home-networking solutions via a wireless LAN, using the home's PC as the central control element. Reflects a wireless home LAN configuration in which one PC acts as a master to the network. It provides network addressing and routing between the home and the internet.

This straightforward approach simply marries familiar PC technology to new home-networking technology. It also represents some consumer-marketplace challenges to ultimate mass-market success.

1.3 My Home Networking Application

The implemented applications and protocols in the product of the application of this thesis can be found below. The details of each application will be explained in chapter six.

1.3.1 Implemented Applications

1.3.1.1 Simply Run

When this application runs, the data will be written to device 1 as 0xFF, which enables all 8 of the pins of device 1, and all the LEDs will be switched ON without taking care of the previous state of the pins.

1.3.1.2 Simply Stop

When this application runs, the data will be written to the device as 0x00 and all the LEDs will be switched OFF without taking care of the previous state of the pins.

1.3.1.3 Retrieve Significant Data

When this application runs, ECS responds with the message that contains the current states of the each pin of the device.

1.3.1.4 Write Significant Data

This application is used to set each pin of the specified device with the indicated value in binary form without taking care of the current state of the pins.

1.3.1.5 Time Application

This application is used to control the related device according to any of the specified time and period. After entering value of 3 as application code, the time parameters also need to be entered according to the explanations in product specification chapter.

1.3.1.6 Relative Application

This application is used for establishing a correlation between the devices. Therefore, the state of the devices can be changed according to any of the other device.

1.3.2 *Embedded Control Protocol (ECP)*

ECP is used for the communication between the Embedded Control Server (ECS) and Embedded Control Client (ECC). The signaling is based on the UART standard, which provides the sequence of serial data transfer. The details about the protocol will be explained in chapter four.

1.4 General Home Networking Applications

1.4.1 *Controlling Household Devices*

Home networking can allow controls within the house, such as temperature and lighting, to be managed through the network and even remotely through the Internet. The network can also be used for home security monitoring with network cameras.

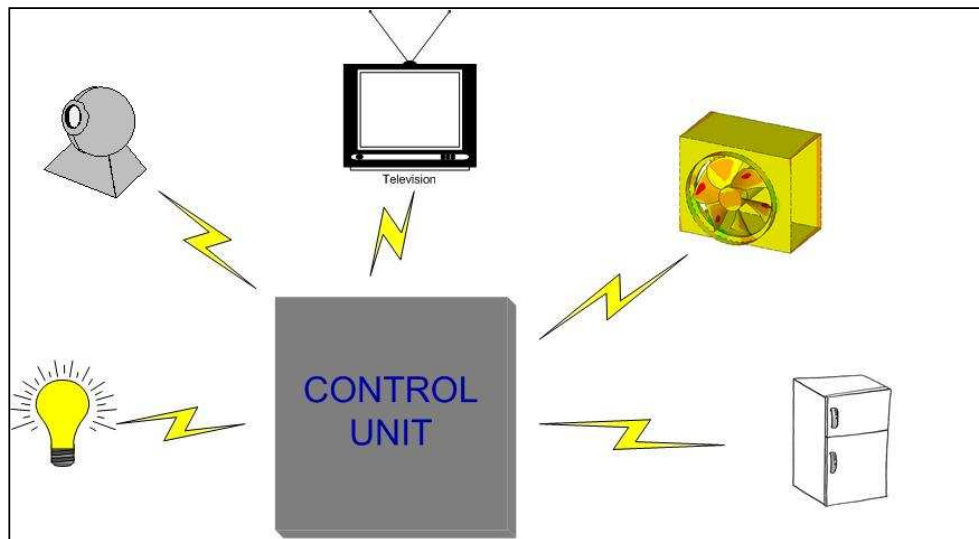


Figure 1.2 A simple representation of a home control network.

1.4.2 *Computing and Resource Sharing*

Home networking allows all users in the household to access the Internet and applications at the same time. In addition, files (not just data, but also audio and video depending on the speed of the network) can be swapped and peripherals such

as printers and scanners can be shared. There is no longer the need to have more than one Internet access point, printer, scanner, or in many cases, software packages.

1.4.3 Entertainment and Information

Home networks support entertainment services like interactive gaming, digital and streaming audio/video and hybrid Internet-TV services like WebTV and DirecTV. Home entertainment networks will connect the following:

- Traditional consumer electronic products such as stereo components, TVs and VCRs
- Convergence based devices such as MP3 players, Internet-enabled gaming consoles and set top boxes.
- Computing products such as PCs, modems and residential gateways.

1.4.4 Communications

Home networking allows easier and more efficient communication between users within the household and better communication management with outside communications. Phone, fax, and e-mail messages can be routed intelligently. Access to the Internet can be attained at multiple places in the home with the use of terminals and Webpads.

CHAPTER TWO

DESIGN BASIS OF THE SOFTWARE APPLICATION

2.1 Embedded Control Server (ECS)

Embedded Control Server is the heart of our Home Control Network application, which coordinates the over all network and runs independently. Software Architecture of Embedded Control Server is divided in to several layers as in the figure below.

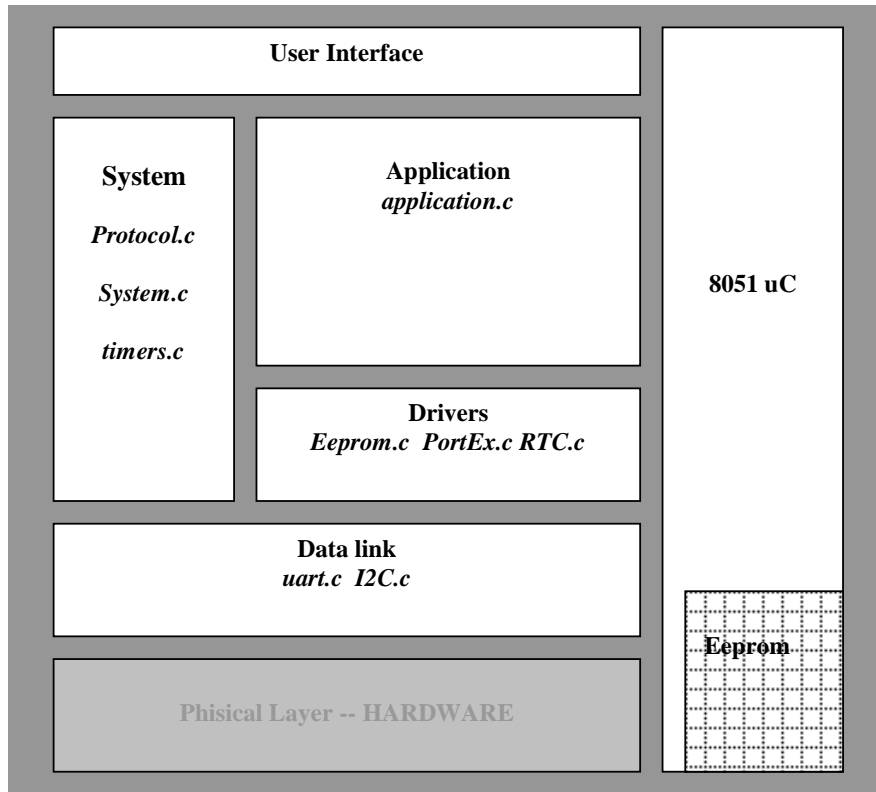


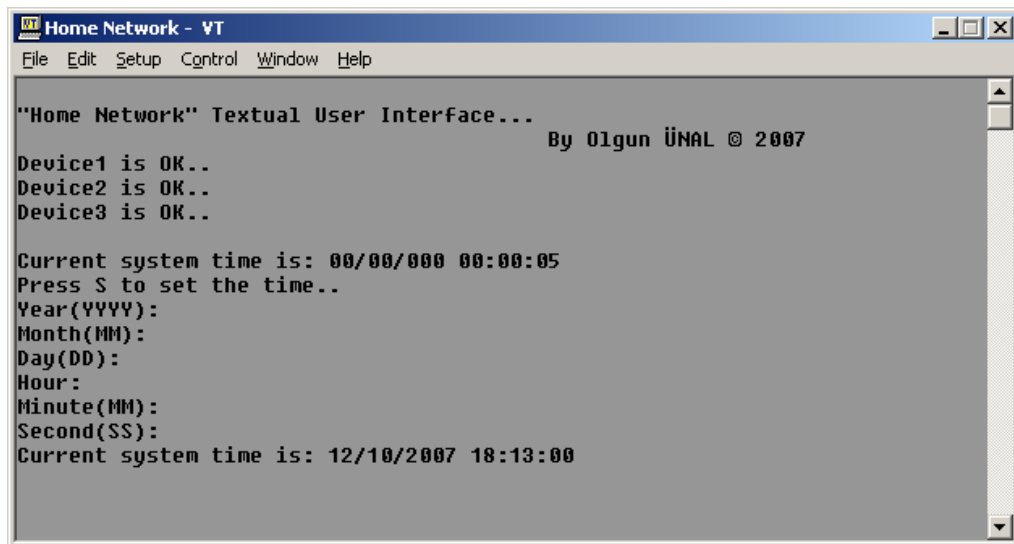
Figure 2.1 The software architecture of Embedded Control Server.

2.1.1 User Interface

To control and re-configure the embedded system, there is an interface needs to be provided to the user, then user can access to the system and manipulate the state of the system. In that sense, the User Interface refers o textual information of the embedded program to the user, and by the control sequences (such as keystrokes with the computer keyboard), the user employs to control the embedded program.

The information received and sent via the user interface is presented via a terminal emulator's program display output. The terminal emulator program can be work like a dummy terminal and uses the serial communication standard of UART.

Here in the Figure 2.2 below there is a snapshot of a simple terminal emulator window.



```
Home Network - VT
File Edit Setup Control Window Help

"Home Network" Textual User Interface...
By Olgun ÜNAL © 2007
Device1 is OK..
Device2 is OK..
Device3 is OK..

Current system time is: 00/00/000 00:00:05
Press S to set the time..
Year(YYYY):
Month(MM):
Day(DD):
Hour:
Minute(MM):
Second(SS):
Current system time is: 12/10/2007 18:13:00
```

Figure 2.2 An example of the textual user interface in the terminal emulator window.

2.1.2 Application

It interfaces directly to and performs common application services for the application processes; it also issues requests to the middleware layer. In this layer, device control requests are processed via a function named “DeviceController” in the embedded code. This function is called if any data is updated in the device register map. Then it retrieves data from all the device address spaces in the memory and checks which parameter is changed regarding each individual devices. If any change occurs for a device, device controller function processes the new data according to the specific device implementation.

2.1.3 System

This layer consists of the system resources, which are available for entire structure such as drivers, application and data link. There are some of system facilities in this layer that can be provided to other layers. The main module of the system is the “protocol” that is responsible of transferring the messages to outside world and processing the messages coming from there. Then the entire embedded system will use the processed messages.

2.1.4 Drivers

A driver typically communicates with the device through a bus or communications subsystem to which the hardware is connected. When a calling program invokes a routine in the driver, the driver issues commands to the device. Once the device sends data back to the driver, the driver may invoke routines in the original calling program. Drivers are hardware-dependent and system-specific. They usually provide the interrupt handling required for any necessary asynchronous time-dependent hardware interface. Device drivers simplify programming by acting like a translator between a device and the applications that use it. The higher-level code can be written independently of whatever specific hardware device it may control. Every version of a device requires its own specialized commands. In contrast, most applications access devices by using high-level, generic commands. The driver

accepts these generic statements and converts them into the low-level commands required by the device . In our application, we have the following device drivers, Non-Volatile Memory (Eeprom), Real Time Clock (RTC) and Household Devices (Port expander).

The base functions in Eeprom are as following.

```
void eeprom_write(u16 adr, u8 dat);
```

Writes data (dat) to the related address (adr) of the NVM

```
u8 eeprom_read(u16 adr);
```

Reads data from the related address (adr) and returns the read value in u8 type

The base functions in RTC are as following.

```
void rtc_config(void);
```

Configures the Real time clock settings of the microcontroller

```
void rtc_CountTime_interrupt(void);
```

Executes the interrupt routine when the timer produces the related interrupt.

```
void rtc_PrintCurrentTime(void);
```

Prints the current time to UART channel

```
void rtc_SetCurrentTime(void);
```

Starts asking user for setting the time via user interface.

The base functions in Port expander are as following.

```
u8 PortEx_getport(PortEx_devID device);
```

Gets the 8-bit data of the specified port (device) and returns the data.

```
void PortEx_setport(PortEx_devID device, u8 value);
```

Sets the 8 bit data to specified port(device)

```
u8 PortEx_getpin (PortEx_devID device, u8 pin);
```

Gets the 1-bit data from the specified port's (PortEx_devID device) specified pin (u8 pin)

```
void PortEx_setpin (PortEx_devID device, u8 pin, u8 value);
```

Sets the 1-bit data of the specified port's (PortEx_devID device) specified pin.

2.1.5 Datalink

Data link is the mean of connecting one location to another for the purpose of transmitting and receiving data. It can also be an assembly, consisting of parts of two data terminal equipments and the interconnecting data circuit that is controlled by a link protocol enabling data to be transferred from a data source to a data sink.

In our application, we have two base communication protocol used in the datalink layer, I2C and UART

2.1.5.1 I2C

I²C is a multi-master serial computer bus invented by Philips that is used to attach low-speed peripherals to an embedded system. The name stands for Inter-Integrated Circuit and is pronounced *I-squared-C* and also, incorrectly, *I-two-C*. I²C uses only two bidirectional open-drain lines, Serial Data (SDA) and Serial Clock (SCL), pulled up with resistors. Typical voltages used are +5 V or +3.3 V although systems with other, higher or lower, voltages are permitted.

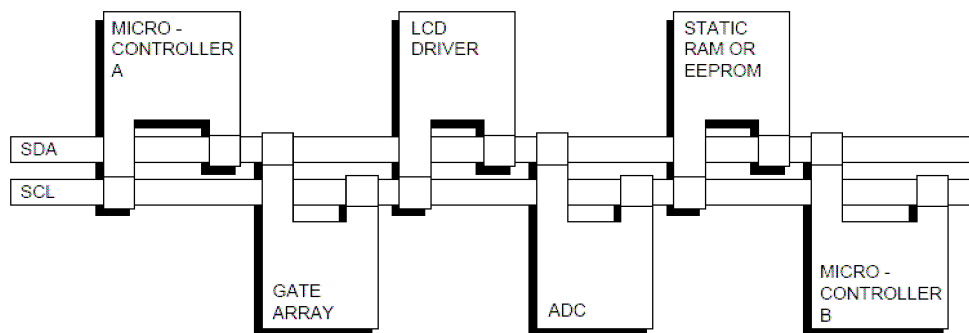


Figure 2.3 An example of I2C bus configuration.

Here are some of the features of the I2C-bus:

- Only two bus lines are required; a serial data line (SDA) and a serial clock line (SCL).
- Each device connected to the bus is software addressable by a unique address and simple master/slave relationships exist at all times; masters can operate as

master-transmitters or as master-receivers.

- It is a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer.
- Serial, 8-bit oriented, bidirectional data transfers can be made at up to 100 kbit/s in the Standard-mode, up to 400 kbit/s in the Fast-mode, up to 1 Mbit/s in Fast-mode plus, or up to 3.4 Mbit/s in the High-speed mode.
- On-chip filtering rejects spikes on the bus data line to preserve data integrity.
- The number of ICs that can be connected to the same bus is limited only by a maximum bus capacitance. More capacitance may be allowed under some conditions.

In our application, we use I2C to communicate with the port expander ICs (PCF8574) that is representing each of the household device.

2.1.5.2 *UART*

Universal Asynchronous Receiver/Transmitter is a type of "asynchronous receiver/transmitter", a piece of computer hardware that translates data between parallel and serial interfaces. Used for serial data telecommunication, a UART converts bytes of data to and from asynchronous start-stop bit streams represented as binary electrical impulses. UARTs are commonly used in conjunction with other communication standards such as RS-232. UARTs are now commonly included in microcontrollers

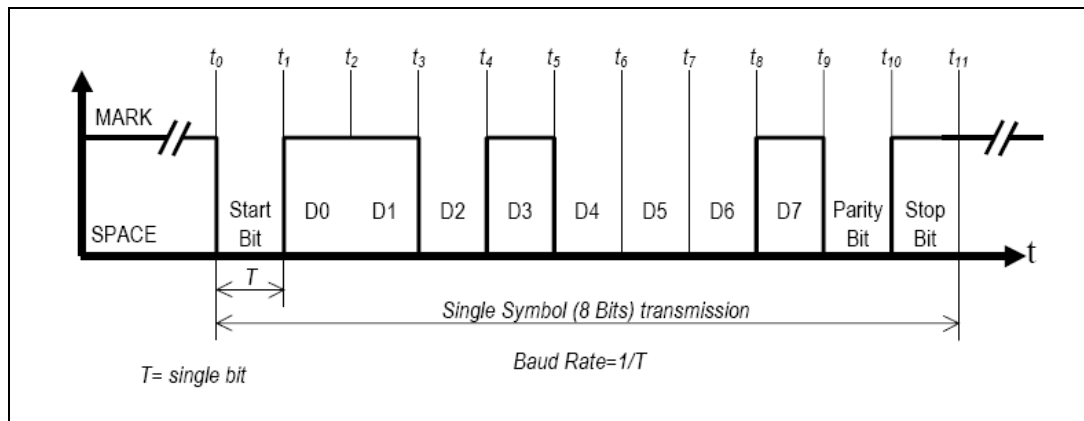


Figure 2.4 A sample data stream of a UART communication.

In asynchronous communication, data is preceded with a start bit which indicates to the receiver that a word (a chunk of data broken up into individual bits) is about to begin. To avoid confusion with other bits, the start bit is twice the size of any other bit in the transmission. The end of a word is followed by a stop bit, which tells the receiver that the word has come to an end, that it should begin looking for the next start bit, and that any bits it receives before getting the start bit should be ignored. To insure data integrity, a parity bit is often added between the last bit of data and the stop bit. The parity bit makes sure that the data received is composed of the same number of bits in the same order in which they were sent. See the diagram in Figure 2.4 for a portrayal of how asynchronous communication works.

2.1.6 Physical Layer – Hardware

Details of this layer will be considered in the Chapter of Hardware Architecture.

2.1.7 Architecture of the 8051 Microcontroller

The 8051 family of microcontrollers is based on an architecture, which is highly optimized for embedded control systems. It is used in a wide variety of applications from military equipment to automobiles to the keyboard on your PC.

The basic architecture consists of the following features:

- 32 discrete I/O pins (4 groups of 8) which can be individually accessed
- Two 16 bit timer/counters
- Full duplex UART
- 6 interrupt sources with 2 priority levels
- 128 bytes of on board RAM
- Separate 64K byte address spaces for DATA and CODE memory

2.1.7.1 Memory Organization

The 8051 architecture provides the user with three physically distinct memory spaces. Each memory space consists of contiguous addresses from zero to the maximum size, in bytes, of the memory space. Address overlaps are resolved by utilizing instructions which refer specifically to a given address space.

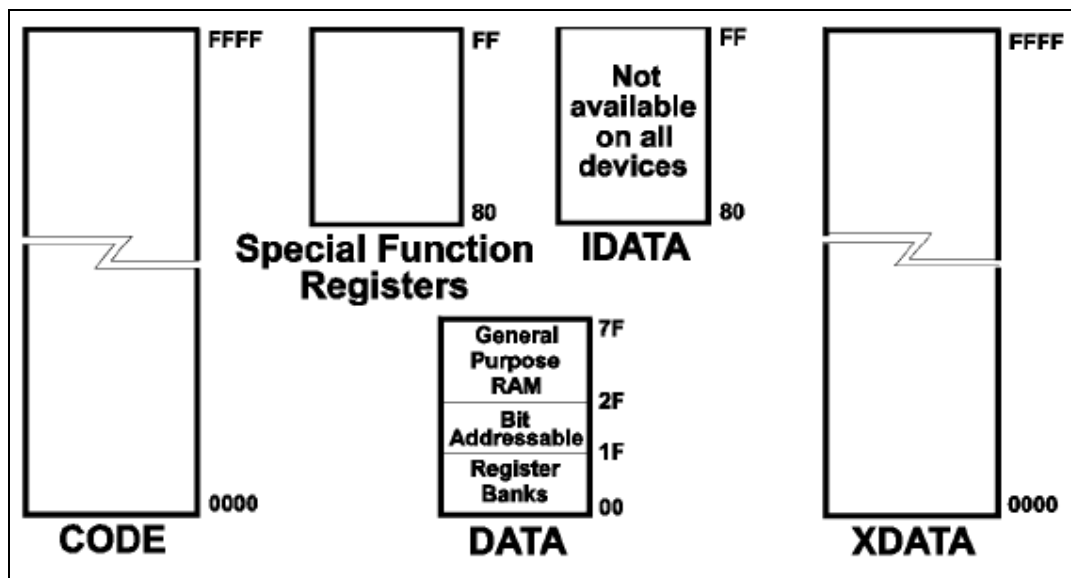


Figure 2.5 8051 memory architecture.

2.1.7.2 *Bit processing and Boolean logic*

The 8051 contains a single bit Boolean processor that can be used to perform logical operations on any of the 128 addressable bits in the BIT segment, the 128 addressable bits in the SFRs, and any of the 32 I/O lines (port 0 through port 3). The 8051 can perform OR, AND, XOR, complement, set, and clear operations on bits as well as moving bit values as one would normally move byte values.

2.1.7.3 *Addressing Modes*

The 8051 is capable of performing direct and indirect memory accesses on its various memory spaces. These are the typical methods through which processor systems access memory. Direct accesses are characterized by presence of the address of the accessed variable in the instruction itself. These accesses can only be performed on the DATA segment and the SFRs.

2.1.7.4 *Interrupts on the 8051*

The basic 8051 supports six interrupt sources: two external interrupts, two timer/counter interrupts, and a serial byte in/out interrupt. These interrupt sources force the processor to vector to one of five locations in the lowest part of the CODE address space (serial input and serial output interrupts share the same vector). The interrupt service routine must either reside there or be branched to from there.

2.1.7.5 *On-Board Timer/Counters*

The standard 8051 has two timer/counters (other 8051 family members have varying amounts), each of which is a full 16 bits. Each timer/counter can be function as a free running timer (in which case they count processor cycles) or can be used to count falling edges on the signal applied to their respective I/O pin (either T0 or T1). When used as a counter, the input signal must have a frequency equal to or lower than the instruction cycle frequency divided by 2 (ie: the oscillator frequency /24) since the incoming signal is sampled every instruction cycle, and the counter is incremented only when a 1 to 0 transition is detected (which will require two

samples). If desired, the timer/counters can force a software interrupt when they overflow.

2.1.7.6 On-Board UART

The 8051 features an on board, full duplex UART that is under software control. The UART is configured via the SCON (Serial Control) SFR. The SCON register allows the user to select the UART mode, enable reception, and check UART status.

2.1.7.7 I2C

A new form of inter-device communication becoming popular is the I2C (inter-integrated circuit) interface created and popularized by Phillips. I2C is a serial format data link, which uses two wires (one for data and one for clock) and can have many drops to varying devices. Each device has its own ID on the link to which it will respond, data transfers are bi-directional, and the bus can have more than one master. Phillips has been a leader in adding I2C capability to the 8051. Hardware wise, two I/O pins are taken from port 1 for the I2C interface and a set of SFRs are added to control the I2C and aid in implementing the protocol of this interface.

2.1.7.8 Analog to Digital Converters

Analog to digital converters are peripherals, which are not available on every 8051 family member, but are common enough that they were worth discussing in this overview. A/D converters are usually controlled via some master register (usually called ADCON) which is given one of the empty locations in the SFR memory segment. The ADCON register allows the user to select the channel to be used for A/D conversion, to start a new conversion and to check the status of a current conversion.

2.1.7.9 *Watchdog Timers*

Watchdog timers are available on an expanding group of 8051 family members. The purpose of a watchdog timer is to reset the controller if the timer is not fed by a specific sequence of operations within a specified amount of time. This prevents coincidental reloading of the watchdog by runaway software.

2.1.7.10 *Using C with the 8051*

When designing software for a smaller embedded system with the 8051, it is very commonplace to develop the entire product using assembly code. With many projects, this is a feasible approach since the amount of code that must be generated is typically less than 8 kilobytes and is relatively simple in nature. If a hardware engineer is tasked with designing both the hardware and the software, he or she will frequently be tempted to write the software in assembly language. The trouble with projects done with assembly code can be that they can be difficult to read and maintain, especially if they are not well commented. Additionally, the amount of code reusable from a typical assembly language project is usually very low. Use of a higher-level language like C can directly address these issues.

A program written in C is easier to read than an assembly program. Since a C program possesses greater structure, it is easier to understand and maintain. Because of its modularity, a C program can better lend itself to reuse of code from project to project. The division of code into functions will force better structure of the software and lead to functions that can be taken from one project and used in another, thus reducing overall development time.

A high order language such as C allows a developer to write code, which resembles a human's, thought process more closely than does the equivalent assembly code. The developer can focus more time on designing the algorithms of the system rather than having to concentrate on their individual implementation. This will greatly reduce development time and lower debugging time since the code is more understandable.

By using a language like C, the programmer does not have to be intimately familiar with the architecture of the processor. This means that a someone new to a given processor can get a project up and running quicker, since the internals and organization of the target processor do not have to be learned. Additionally, code developed in C will be more portable to other systems than code developed in assembly. Many target processors have C compilers available, which support ANSI C.

2.2 Embedded Control Client / Web Server

As in the Figure 2.8 below, Software Architecture of Embedded Control Client and Web Server have split in to several layers.

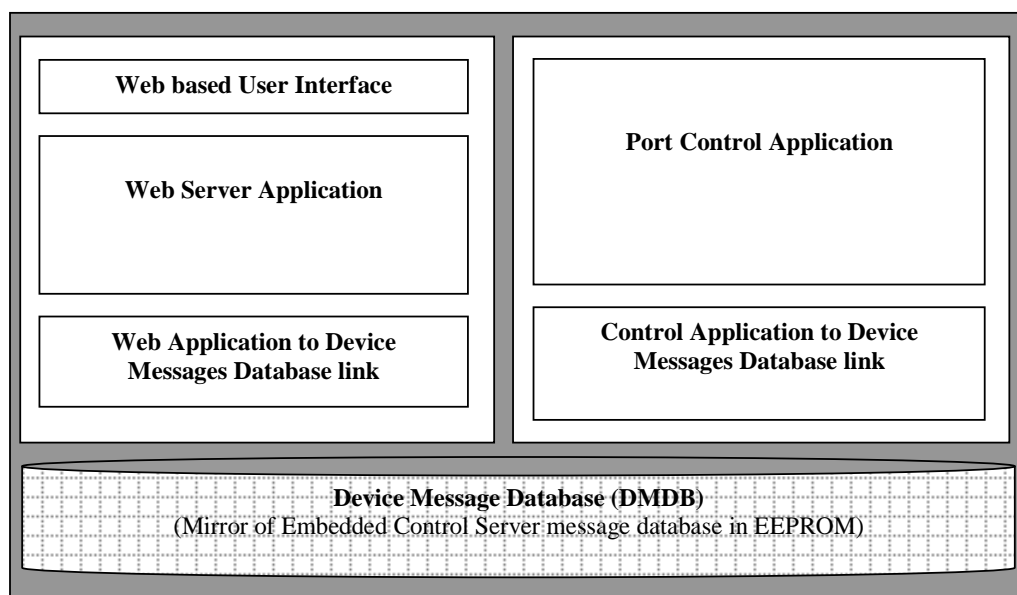


Figure 2.6 Software architecture of Embedded Control Client.

2.2.1 Web Based User Interface (WBUI)

Web Based User Interface allows user to write data in to DMDB. Each parameter for the each device can be individually set or get via this User Interface. Here below in the figure, there are snapshots from how to enter values for a selected device from the beginning.

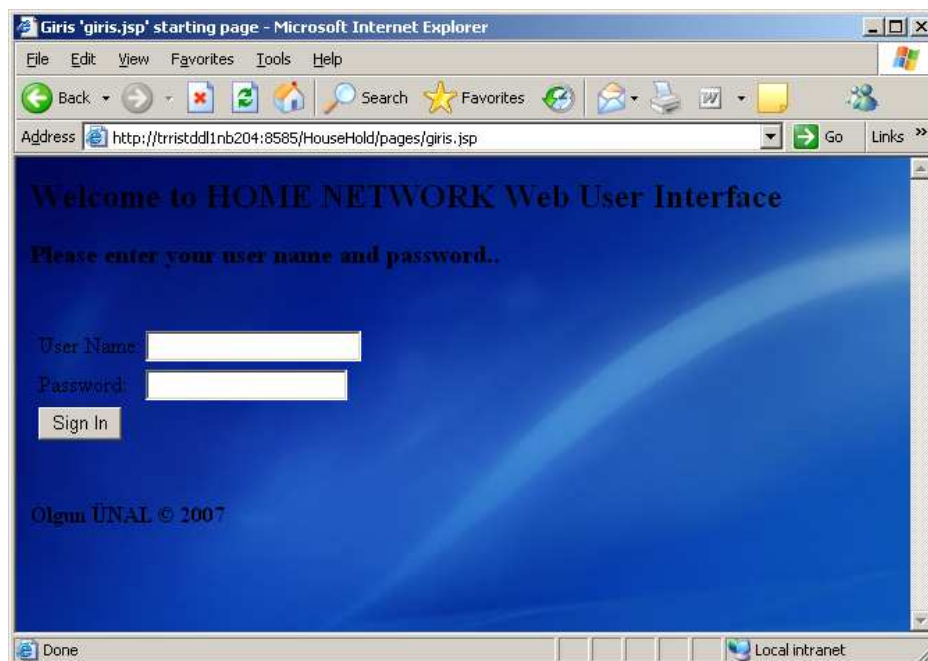


Figure 2.7 Snapshot for the login and welcome page.

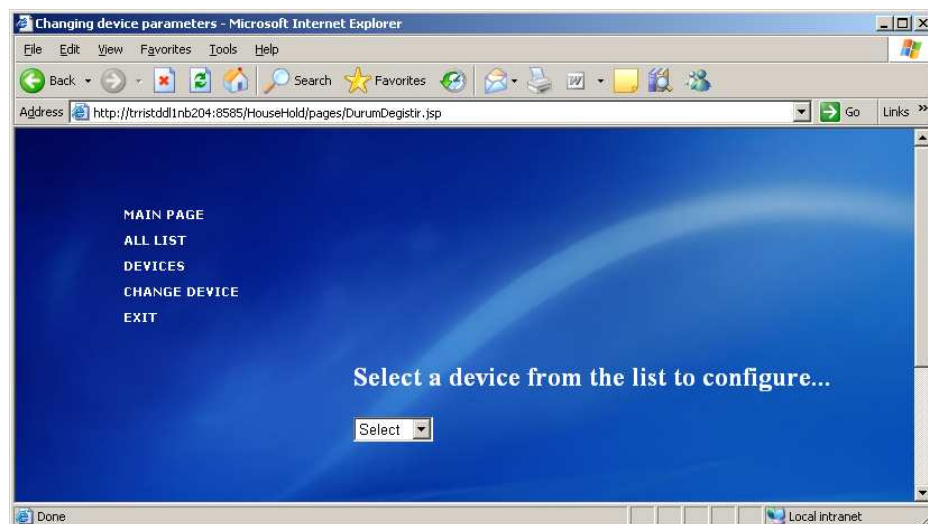


Figure 2.8 Snapshot for the "CHANGE DEVICE" submenu's device selection phase.

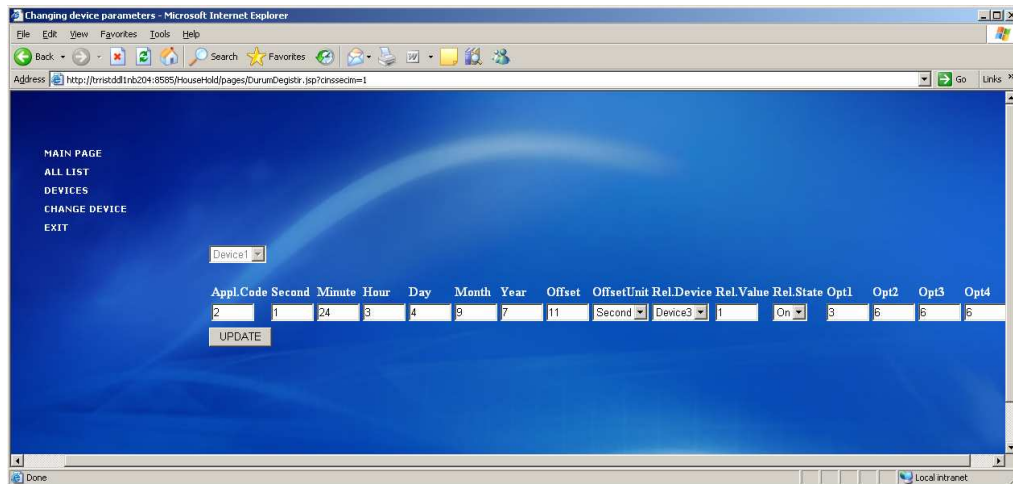


Figure 2.9 Snapshot when changing the parameters of device1.

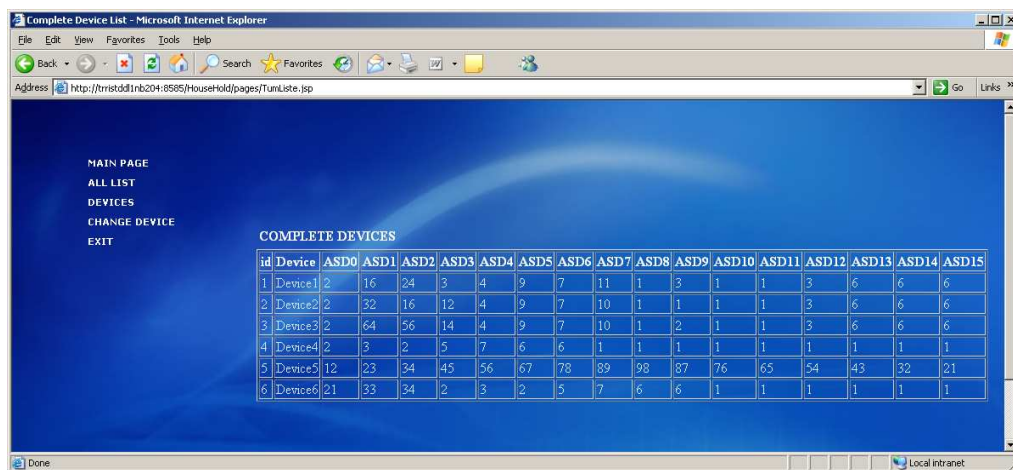


Figure 2.10 Snapshot when displaying the contents of the all devices.

2.2.2 Web Server Application and database link

Web Server Application provides to establish an HTTP server. This server publishes the Web Based User Interface home page to access to DMDB. HTTP server is based on the “Tomcat” from the Apache Software Foundation.

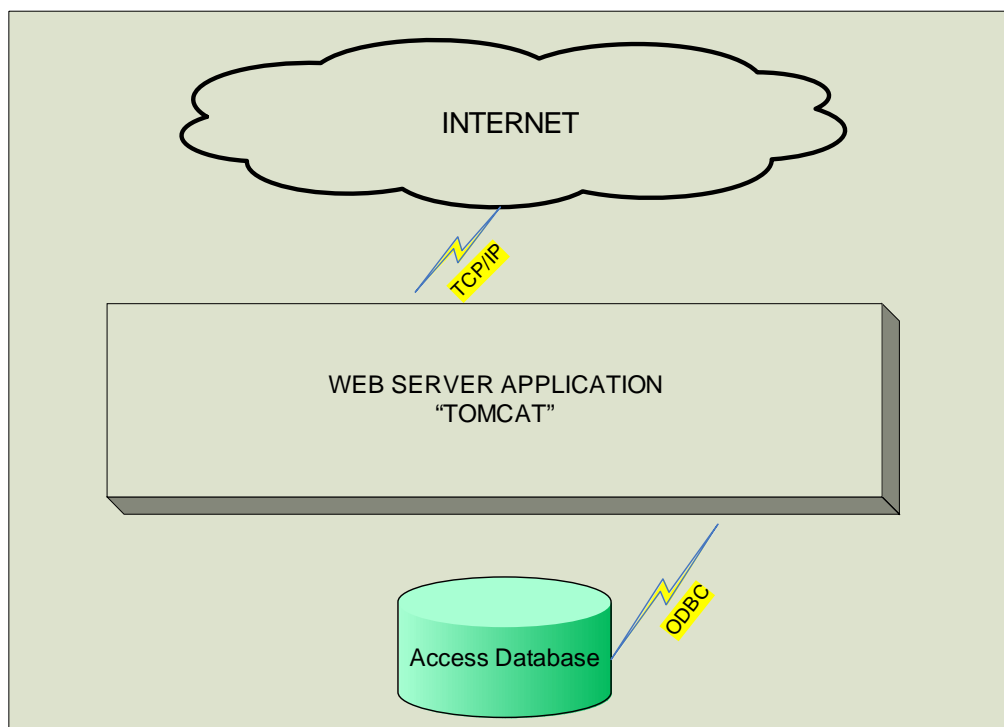


Figure 2.11 Scheme of the web server application.

2.2.3 Control Application and database link

The port control application, which is written in java, checks the Device Message Database and writes to the port in case any device information was updated in the database. In addition, it listens the port in case any message was sent by the host and gets the valid messages and updates the DMDB according to the valid message.

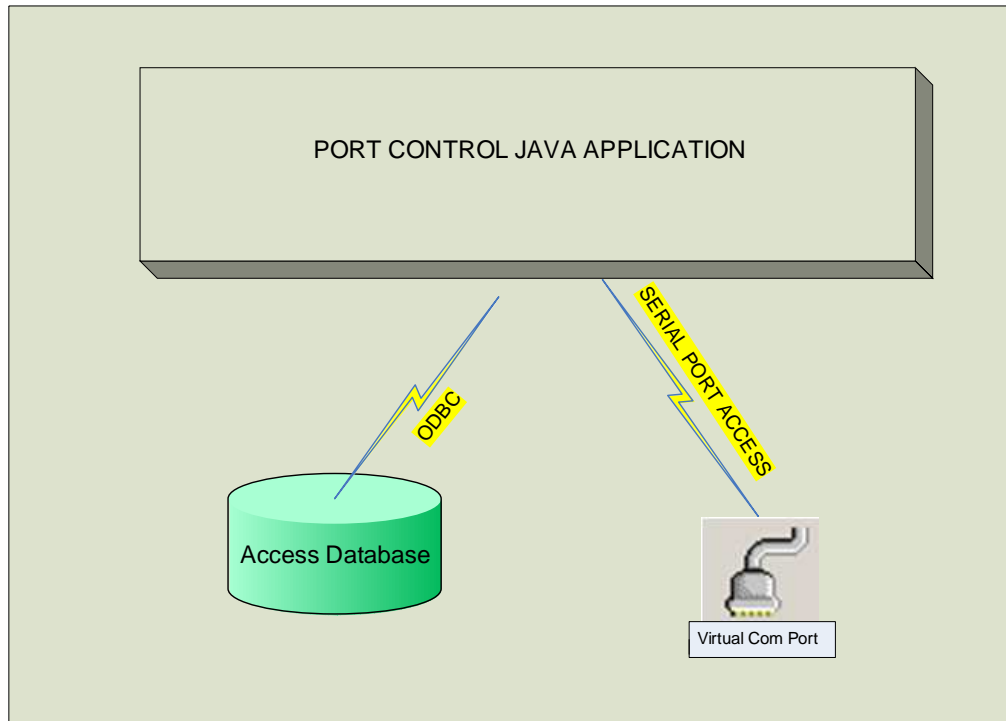


Figure 2.12 Scheme of the port control application.

2.2.4 Device Message Database (DMDB)

DMDB is an Access database filed as name of Home.mdb. Both Web server application and Port Control application access to that database and update it according to the requests comes from the user via Web UI or from the Embedded Control Server.

DeviceId	DeviceName	State	AddressId	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8	Byte9	Byte10	Byte11	Byte12	Byte13	Byte14	Byte15	Byte16
1	Device1	1	0	2	1	24	3	4	9	7	11	1	3	1	1	3	6	6	6
2	Device2	0	1	2	0	16	12	4	9	7	10	1	1	1	1	3	6	6	6
3	Device3	5	2	1	1	56	14	4	9	7	10	1	2	1	1	3	6	6	6
4	Device4	6	3	2	3	2	5	7	6	6	1	1	1	1	1	1	1	1	1
5	Device5	0	4	12	23	34	45	56	67	78	89	98	87	76	65	54	43	32	21
6	Device6	1	5	21	33	34	2	3	2	5	7	6	6	1	1	1	1	1	1
oNumber		0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 2.13 Contents of the device message database.

CHAPTER THREE

HARDWARE ARCHITECTURE

3.1 Embedded Control Server (ECS)

Embedded Control Server consists of the following HW parts, which are shown, in the Figure 3.1

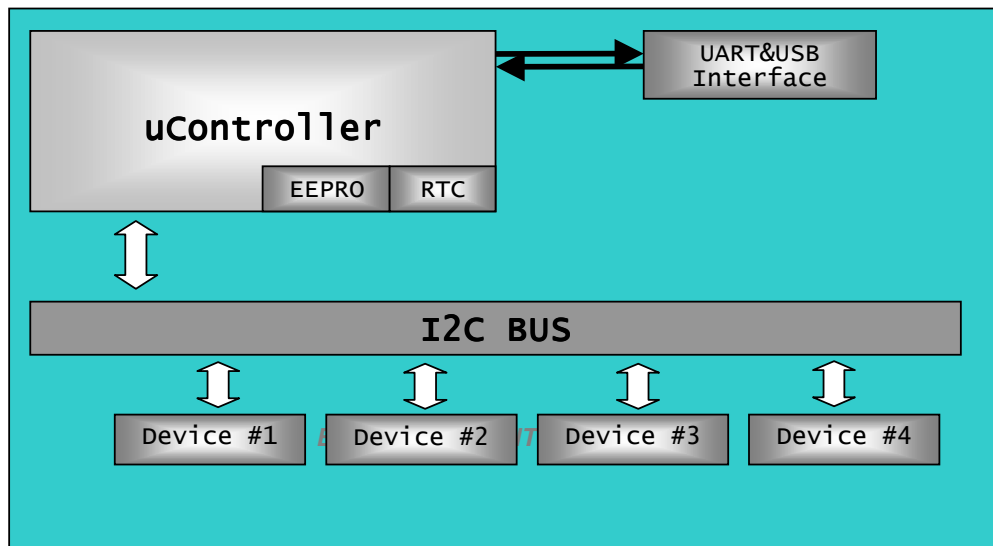


Figure 3.1 Hardware architecture of Embedded Control Server.

3.1.1 Microcontroller

Microcontroller has the following peripherals and units,

3.1.1.1 EEPROM

The EEPROM (also called as E²PROM) or Electrically Erasable Programmable Read-Only Memory is a non-volatile storage chip used in computers and other devices to store small amounts of volatile (configuration) data.

The size of the eeprom used in the reference application is 512 KB. This eeprom is integrated in the same chip with the main microprocessor.

3.1.1.2 *Real Time Clock*

The real-time clock (RTC) is an internal clock (most often in the form of an integrated circuit) that keeps track of the current time. Although the term often refers to the devices in personal computers, servers and embedded systems, RTCs are present in most any electronic device, which needs to keep accurate time.

Virtually all RTCs use a crystal oscillator. In many cases, the oscillator's frequency is 32.768 kHz. This is the same frequency used in quartz clocks and watches, and for the same reasons, namely that the frequency is exactly 2^{15} cycles per second, which is a convenient rate to use with simple binary counter circuits.

3.1.1.3 *General Purpose I/O Port*

The general-purpose input/output (GPIO) peripheral provides dedicated general-purpose pins that can be configured as either inputs or outputs. When configured as an output, you can write to an internal register to control the state driven on the output pin. When configured as an input, you can detect the state of the input by reading the state of an internal register

3.1.1.4 *UART interface*

Hardware of the UART is integrated in the same chip as microcontroller. However, converter hardware converts the uart signals to USB signals. Then the ECC only sees the USB signals via this converter. USB to UART converter hardware acts as a translator between the ECC and the UART module of ECS.

3.1.1.5 *I2C interface*

8051 processor has built in I2C interface and registers which are linked to the dedicated GPIO pins. Therefore, whenever an I2C message is sent by the microcontroller the integrated I2C interface delivers the message in a proper format in to the dedicated pins.

3.1.1.6 Memory

8051 controller has its own built in memory structure. There are 3 main memory organization we can talk about, one is the Random Access Memory of the controller which the runtime code runs on, the second is the flash memory that contains the read only code data and the last is the non volatile memory that contain Read/write data to be read or written in the run time by the current application.

3.1.1.7 Processor

8051 processor has a processor core that executes the instructions in the order that program data provides in the proper way. One 8051 processor cycle consists of twelve oscillator periods. Each of the twelve oscillator periods is used for a special function by the 8051 core such as op code fetches and samples of the interrupt daisy chain for pending interrupts. The time required for any 8051 instruction can be computed by dividing the clock frequency by 12, inverting that result and multiplying it by the number of processor cycles required by the instruction in question. Therefore, if you have a system, which is using an 11.059MHz clock, you can compute the number of instructions per second by dividing this value by 12. This gives an instruction frequency of 921583 instructions per second. Inverting this will provide the amount of time taken by each instruction cycle (1.085 microseconds).

3.1.2 Household Interface Devices (Port Expander)

The name of the chip that is used for interfacing the household device is PCF8574. The PCF8574 is a silicon CMOS circuit. It provides general-purpose remote I/O expansion for most microcontroller families via the two-line bidirectional bus (I2C). The device consists of an 8-bit quasi-bidirectional port and an I2C-bus interface. The PCF8574 has a low current consumption and includes latched outputs with high current drive capability for directly driving LEDs. It also possesses an interrupt line (INT) which can be connected to the interrupt logic of the microcontroller. By sending an interrupt signal on this line, the remote I/O can

inform the microcontroller if there is incoming data on its ports without having to communicate via the I2C-bus. This means that the PCF8574 can remain a simple slave device.

Basic Features of the chip is as following;

- Operating supply voltage 2.5 to 6 V
- Low standby current consumption of 10 mA maximum
- I2C to parallel port expander
- Open-drain interrupt output
- 8-bit remote I/O port for the I2C-bus
- Compatible with most microcontrollers
- Latched outputs with high current drive capability for directly driving LEDs
- Address by 3 hardware address pins for use of up to 8 devices
- DIP16, or space-saving SO16 or SSOP20 packages.

3.2 Embedded Control Client and Web Server (Computer Workstation)

Embedded Control Client and Web Server consists of following hardware sub parts

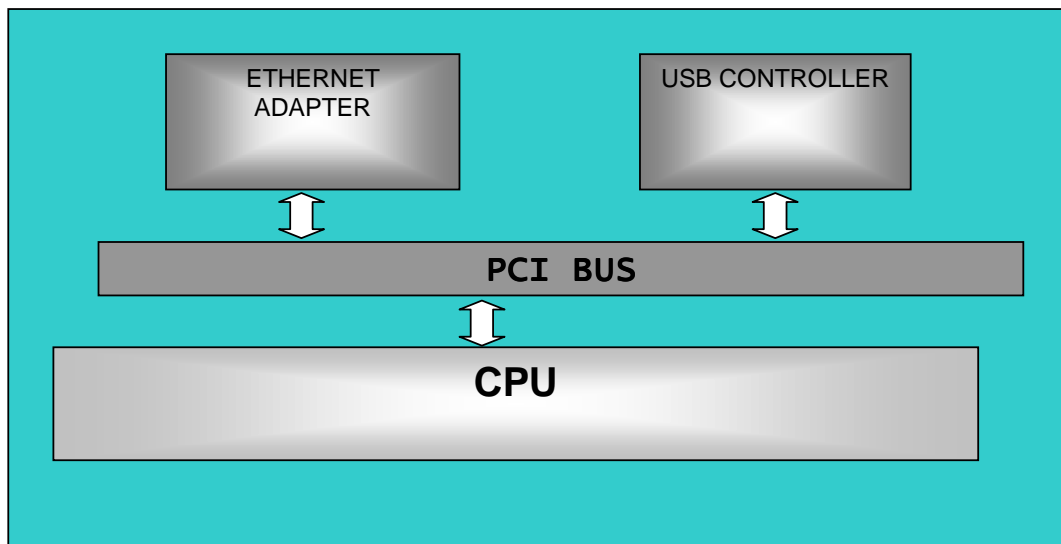


Figure 3.2 Hardware architecture of Embedded Control Client.

3.2.1 Central Processor Unit (CPU)

CPUs provide the fundamental digital computer trait of programmability, and are one of the necessary components found in computers of any era, along with primary storage and input/output facilities. A CPU that is manufactured as a single integrated circuit is usually known as a microprocessor. Beginning in the mid-1970s, microprocessors of ever-increasing complexity and power gradually supplanted other designs, and today the term "CPU" is usually applied to some type of microprocessor.

3.2.2 Ethernet Adapter

A network card, network adapter or NIC (network interface card) is a piece of computer hardware designed to allow computers to communicate over a computer network. It is both an OSI layer 1 (physical layer) and layer 2 (data link layer) device, as it provides physical access to a networking medium and provides a low-level addressing system through the use of MAC addresses. It allows users to connect to each other either by using cables or wirelessly.

A network card typically has a twisted pair, BNC, or AUI socket where the network cable is connected, and a few LEDs to inform the user of whether the network is active, and whether or not there is data being transmitted on it. The Network Cards are typically available in 10/100/1000 Mbit/s (Mbit/s). This means they can support a transfer rate of 10 or 100 or 1000 Megabits per second

3.2.3 USB Controller

The USB controller is a universal serial bus (USB) peripheral interface device designed specifically for applications that require isochronous data streaming. The device is fully compatible with the USB Specifications. This kind of devices mostly use a standard 8052 microcontroller unit (MCU) core with on-chip memory. The

MCU memory includes 4K bytes of program memory ROM that contains a boot loader program. At initialization, the boot loader program downloads the application program code to an 8K RAM from a nonvolatile memory on the printed-circuit board (PCB). The MCU handles all USB control, interrupt and bulk endpoint transactions. In addition, the MCU can handle USB isochronous endpoint transactions.

The USB interface includes an integrated transceiver that supports 12 Mb/s (full speed) data transfers. In addition to the USB control endpoint, support is provided for up to seven in endpoints and seven out endpoints. The USB endpoints are fully configurable by the MCU application code using a set of endpoint configuration blocks that reside in on-chip RAM. All USB data transfer types are supported.

CHAPTER FOUR

THE PROTOCOLS USED IN THE APPLICATION

4.1 Embedded Server Protocol (ECP)

This protocol is used for the communication between the Embedded Control Server (ECS) and Embedded Control Client (ECC). The signaling is based on the UART standard, which provides the sequence of serial data transfer.

ECS Protocol is based on 20 bytes of message stream at a time for each message. The message format can be seen as in the Table 4.1 below.

Table 4.1 ECP message format.

0	Header		Start	1 byte
1	read / write		R/W	1 byte
2	Address		Device Code	2 bytes
3				
4	Application Specific Data (ASD)	0	Application Code	2 bytes
5		1	Second(*)	2 bytes
6		2	Minute(*)	2 bytes
7		3	Hour(*)	2 bytes
8		4	Day(*)	2 bytes
9		5	Month(*)	2 bytes
10		6	Year(*)	2 bytes
11		7	Time Offset(*)	2 bytes
12		8	Time Unit(*)	1 byte
13		9	Rel.Device(**)	1 byte
14		10	Rel.Value(**)	1 byte
15		11	Rel.State(**)	1 byte
16		12	Device Value	1 byte
17		13	Reserved	1 byte
18		14		1 byte
19	15	1 byte		
20	Header		Stop	1 byte
* : Time application parameter				
** : Relative application parameter				

The message can be split in to four main parts, which are Headers, Read/Write, Address and Application Specific Data.

4.1.1 Headers:

These parts indicate the start and stop of the message. Protocol decoder sections of ECS and ECC take the message in to account according to this start and stop bytes. In our application, application Start byte is defined as 49 and Stop byte is defined as 56 ASCII. So in a message stream like below;

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.1 Any dummy message stream to show how to detect a valid message.

After detecting the Start byte, the protocol application at the destination side counts the data until the total number of bytes in a message is reached, and then the last item is checked whether it is matching with the defined Stop byte.

4.1.1.1 Read / Write byte

This byte indicates that the message is sent to write a data to the destination or a data to be read from the destination.

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.2 Any dummy message stream to that includes read/write byte.

4.1.1.2 Address byte

Every start address corresponds to a command or event. Commands are sent by the client side, which composes the use cases. According to the use cases, an order of commands can be sent to the ECS and ECS's software will process the command and its contents. Events are created in the ECS according to the processed commands. The results of the events are stored in to memory in a same structure of the contents of the commands.

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.3 Address byte.

Start addresses must be aligned for the size of the contents. So the addresses can be used as 0, 0+<contents size>, 0+2x<content size>, ... so on.. if this structure changed, all the nvm content must be erased and re-written.

4.1.1.3 ASD0 (Application Code)

Includes 2 byte of data that represent the Application Code of the specified device

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.4 ASD0.

4.1.1.4 ASD1 (Second)

ASD1 is a 2 byte of data that represent the seconds (in Time Application) of the time to be set for the command sent from client or the seconds of the event time.

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.5 ASD1.

4.1.1.5 ASD2 (Minute)

ASD2 is a 2 byte of data that represent the minutes (in Time Application) of the time to be set for the command sent from client or the minutes of the event time.

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.6 ASD2.

4.1.1.6 ASD3 (Hour)

Hour is a 1 byte of data that represent the hours (in Time Application) of the time to be set for the command sent from client or the hours of the event time.

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.7 ASD3.

4.1.1.7 ASD4 (Day)

Day is a 1 byte of data that represent the days (in Time Application) of the time to be set for the command sent from client or the days of the event time.

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.8 ASD4.

4.1.1.8 ASD5 (Month)

Month is a 1 byte of data that represent the months (in Time Application) of the time to be set for the command sent from client or the months of the event time.

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.9 ASD5.

4.1.1.9 ASD6 (Year)

Year is a 1 byte of data that represent the years of the time (in Time Application) to be set for the command sent from client or the years of the event time.

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.10 ASD6.

4.1.1.10 ASD7 and ASD8 (Time Offset and Unit)

Consist of 3 bytes of the data. First 2 bytes represent (in Time Application) how much offset will be add for the selected time part in unit. Next 1 byte represents (in Time Application) which unit of the time data to be offset. For example, if it is six the “Seconds” of the time data will be offset. So if the offset is 15 and the unit are six that mean the 15 minutes offset of the current time to process this offset later on the application.

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.11 ASD7 and ASD8.

4.1.1.11 ASD9 (Relative Device)

This byte of data is used for indicating the relative device id in Relative Application, which is one of the predefined applications with application code of four.

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.12 ASD9.

4.1.1.12 ASD10 (Relative Value)

This byte of data is used for indicating the relative value for the Relative Application, which is one of the predefined applications with application code of four.

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.13 ASD10.

4.1.1.13 ASD11 (Relative State)

This byte of data is used for indicating the relative state for the Relative Application, which is one of the predefined applications with the application code of four. The value can be only set to On or Off via WBUI

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.14 ASD11.

4.1.1.14 ASD12 (Device Pin number)

This byte of data is used to indicate which pin of the specified device to be controlled. The value range of this byte can be 0 to 7.

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.15 ASD12

4.1.1.15 ASD13 (Device Pin state)

This byte of data is used to indicate the state of the selected pin number in previous byte. The state can be only 1 or 0.

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.16 ASD13

4.1.1.16 ASD14, ASD15 (Reserved bytes)

There are 2 bytes of space is reserved for any of the specific purpose to be used later. In addition, the reason why it is 2 bytes is to align the memory address to 16 bytes for the convenient usage of the limited amount of the non-volatile memory in the embedded system.

43	56	91	51	00	01	00	00	26	20	09	07	15	2	2	1	1	2	0	0	0	93	32
----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	----	----

Figure 4.17 ASD14 and ASD15

4.1.2 Fetching ECP Messages

According to the algorithm below, the ECP messages are detected and sent to NVM to store.

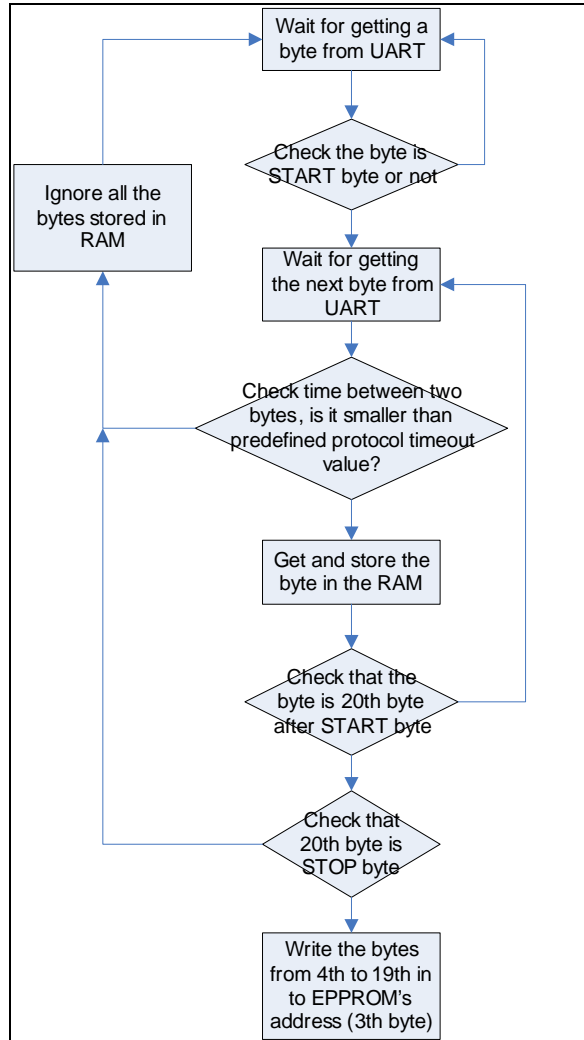


Figure 4.18 Flowchart of how to get the protocol messages

4.1.3 Processing ECP Messages

After getting the ECP messages, they need to be processed. Processing procedure is completely depends on which application will be run. Therefore, the application code byte in the memory for each device is always checked whenever the register map in the nvm is updated. In case any update occurs in the register map, ECS own main application checks the application code of the updated device. If the code is a valid code and was defined in the ECS before then the application starts running according to its related parameters in the memory.

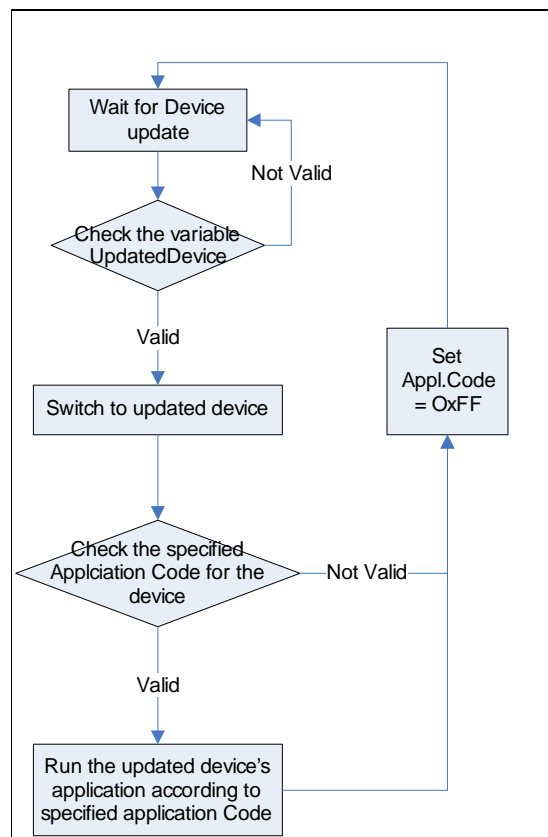


Figure 4.19 Algorithm of how to process ECS messages

4.2 Used Protocols at ECC Side

Embedded Control Client side software applications are communication each other by using a protocol and the web server application are using another protocol to communicate with outside world. The figure below explains how the applications are connected and communicating with other applications.

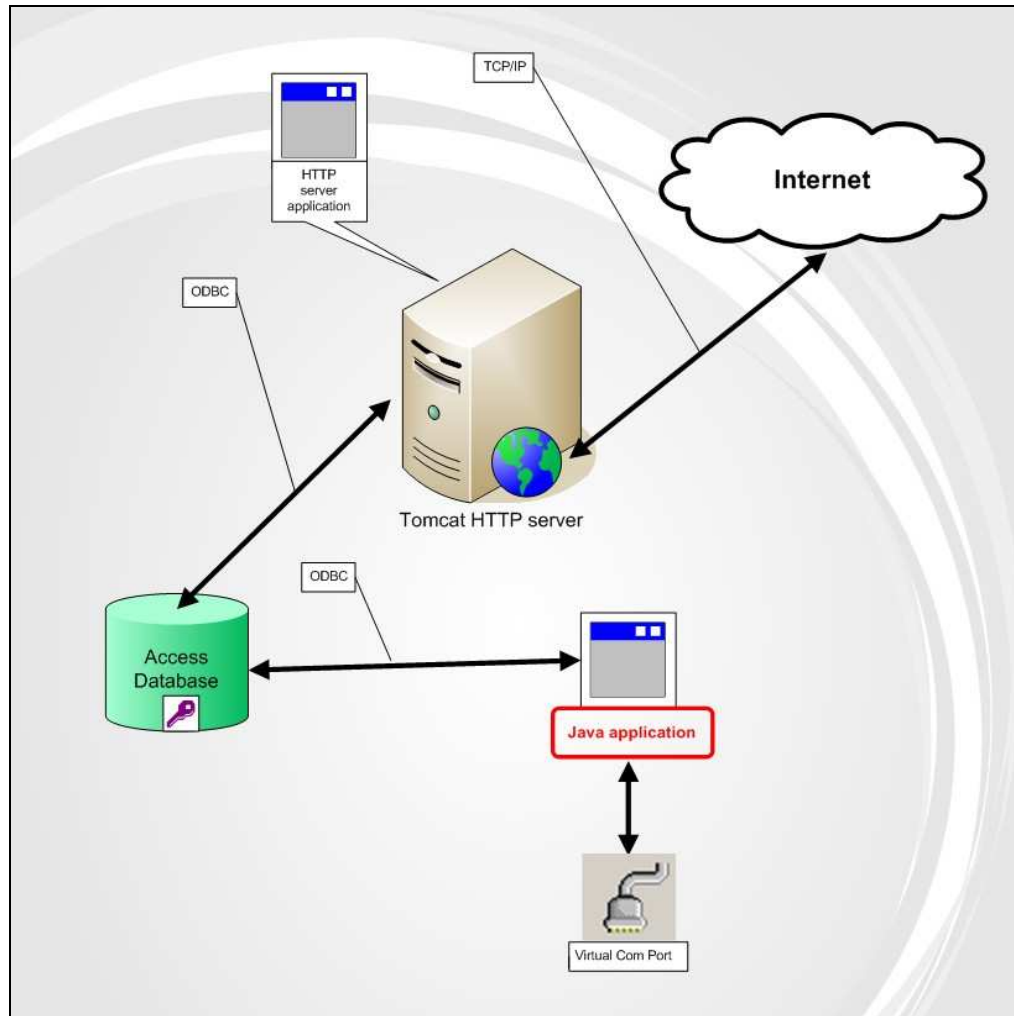


Figure 4.20 Diagram of the connections from top to bottom (from internet to serial port)

4.2.1 ODBC

Open Database Connectivity (ODBC) provides a standard software API method for using database management systems (DBMS). The ODBC specification offers a procedural API for using SQL queries to access data. An implementation of ODBC will contain one or more applications, a core ODBC library, and one or more "database drivers". The core library, independent of the applications and DBMS, acts as an "interpreter" between the applications and the database drivers, whereas the database drivers contain the DBMS-specific details. Thus, a programmer can write applications that use standard types and features without concern for the specifics of each DBMS that the applications may encounter. Likewise, database driver implementers need only know how to attach to the core library. This makes ODBC modular.

To write ODBC code that exploits DBMS-specific features requires more advanced programming. An application must use introspection, calling ODBC metadata functions that return information about supported features, available types, syntax, limits, isolation levels, driver capabilities and more. Even when programmers use adaptive techniques, however, ODBC may not provide some advanced DBMS features. The ODBC 3.x API operates well with traditional SQL applications such as OLTP, but it has not evolved to support richer types introduced by SQL:1999 and SQL:2003.

4.2.1.1 JDBC-ODBC bridges

A JDBC-ODBC bridge consists of a JDBC driver, which employs the ODBC driver to connect to the database. This driver translates JDBC method calls into ODBC function calls. Programmers usually use such a bridge when a particular database lacks a JDBC driver. Sun Microsystems included one such bridge in the JVM, but viewed it as a stopgap measure while few JDBC drivers existed. Sun never intended its bridge for production environments, and generally recommends against its use. Independent data-access vendors now deliver JDBC-ODBC bridges which support current standards for both mechanisms, and which far outperform the JVM built-in.

4.2.1.2 ODBC-JDBC bridges

An ODBC-JDBC bridge consists of an ODBC driver, which uses the services of a JDBC driver to connect to a database. This driver translates ODBC function calls into JDBC method calls. Programmers usually use such a bridge when they lack an ODBC driver for a particular database but have access to a JDBC driver.

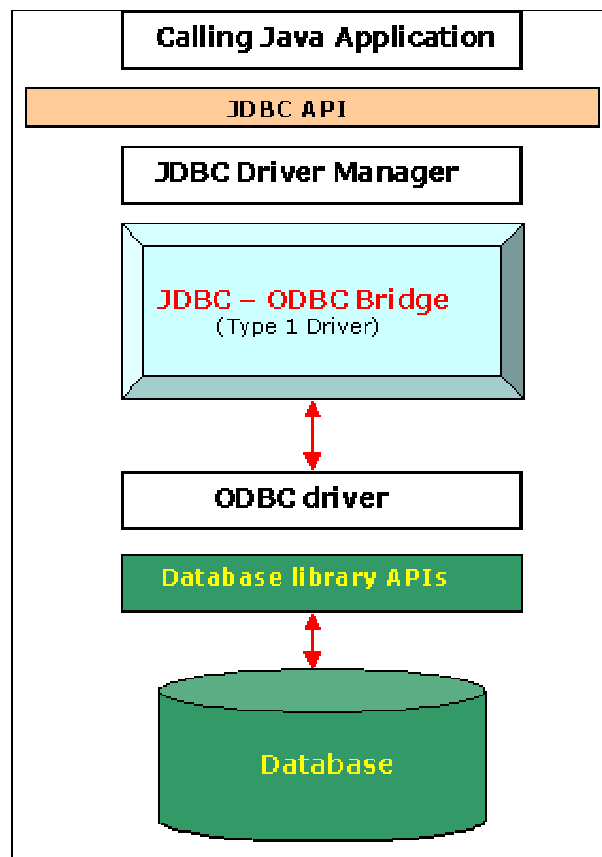


Figure 4.21 Schematic of the JDBC-ODBC bridge

The JDBC type 1 driver, also known as the JDBC-ODBC bridge is a database driver implementation that employs the ODBC driver to connect to the database. The driver converts JDBC method calls into ODBC function calls. The bridge is usually used when there is no pure-Java driver available for a particular database.

4.2.2 TCP/IP

The TCP/IP model uses four layers that logically span the equivalent of the top six layers of the OSI reference model. The following are the TCP/IP model layers, starting from the bottom.

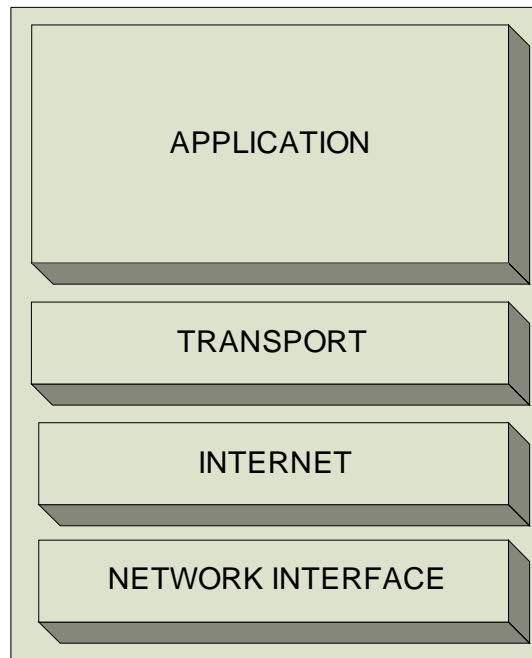


Figure 4.22 TCP/IP layers

4.2.2.1 Network Interface Layer

As its name suggests, this layer represents the place where the actual TCP/IP protocols running at higher layers interface to the local network. This layer is somewhat “controversial” in that some people do not even consider it a “legitimate” part of TCP/IP. This is usually because none of the core IP protocols runs at this layer. Despite this, the network interface layer is part of the architecture. It is equivalent to the data link layer (layer two) in the OSI Reference Model and is sometimes called the link layer.

On many TCP/IP networks, there is no TCP/IP protocol running at all on this layer, because it is simply not needed. For example, if you run TCP/IP over an

Ethernet, then Ethernet handles layer two (and layer one) functions. However, the TCP/IP standards do define protocols for TCP/IP networks that do not have their own layer two implementation. These protocols, the Serial Line Internet Protocol (SLIP) and the Point-to-Point Protocol (PPP), serve to fill the gap between the network layer and the physical layer. They are commonly used to facilitate TCP/IP over direct serial line connections (such as dial-up telephone networking) and other technologies that operate directly at the physical layer.

4.2.2.2 *Internet Layer*

This layer corresponds to the network layer in the OSI Reference Model (and for that reason is sometimes called the network layer even in TCP/IP model discussions). It is responsible for typical layer three jobs, such as logical device addressing, data packaging, manipulation and delivery, and last but not least, routing. At this layer, we find the Internet Protocol (IP), arguably the heart of TCP/IP, as well as support protocols such as ICMP and the routing protocols (RIP, OSPF, BGP, etc.) The new version of IP, called IP version 6, will be used for the Internet of the future and is of course at this layer.

4.2.2.3 *Transport Layer*

This primary job of this layer is to facilitate end-to-end communication over an internetwork. It is in charge of allowing logical connections to be made between devices to allow data to be sent either unreliably (with no guarantee that it gets there) or reliably (where the protocol keeps track of the data sent and received to make sure it arrives, and re-sends it if necessary). It is also here that identification of the specific source and destination application process is accomplished.

The formal name of this layer is often shortened to just the transport layer; the key TCP/IP protocols at this layer are the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). The TCP/IP transport layer corresponds to the layer of the same name in the OSI model (layer four) but includes certain elements that are arguably part of the OSI session layer. For example, TCP establishes a connection

that can persist for a long period of time, which some people say makes a TCP connection more like a session.

4.2.2.4 *Application Layer*

This is the highest layer in the TCP/IP model. It is a rather broad layer, encompassing layers five through seven in the OSI model. While this seems to represent a loss of detail compared to the OSI model. The TCP/IP model better reflects the “blurry” nature of the divisions between the functions of the higher layers in the OSI model, which in practical terms often seem rather arbitrary. It really is hard to separate some protocols in terms of which of layers five, six or seven they encompass.

4.2.2.5 *Apache Tomcat HTTP Server*

Tomcat 3.x is the reference implementation of Sun's Java Server Pages 1.1 and the Java Servlet 2.2 Specifications. Tomcat is a Servlet container, which means that Java Servlets can be used within it to enable programmers to use the Servlet in such things as web pages. Whereas a JSP web page has both HTML and Java JSP code within the same document, a Servlet is a block of code that can be reused. These can then be compiled once and distributed to run on any standards compliant Servlet Container. A Servlet can be called directly to return HTML markup to a browser, but it is more usual that a JSP page uses the abilities of the Servlet (By calling it methods) and returns a custom representation of the returned data.

Servlets can do much more than return useful information for web pages, however. For example, it is possible to create a Servlet that listens for remote connections from an applet, say, and sends live stock information to the client application. In this manner, a Servlet can be much more useful than a JSP/ASP page or other usual web server extension. This technology allows programmers to develop platform independent extensions that can be integrated simply with an existing web server.

CHAPTER FIVE

DEVELOPMENT ENVIRONMENTS

5.1 ECS Application Development Environment

Software Development of the Embedded Control Server is based on the programming language ANSI C. In ECS, there is a microcontroller from the NXP (founded by Philips) part number of LPC935 and core with Intel 8051 processor. To run the instructions in 8051 normally the application should be written in the 8051's instruction codes (assembly). However, there is a possibility to convert the C code to the assembly language, there is a compiler used from the Keil Software Inc. In addition, the complete development environment is provided by Keil that is called Micro Vision that includes compiler, debugger, editor and all libraries for LPC935 microcontroller of NXP.

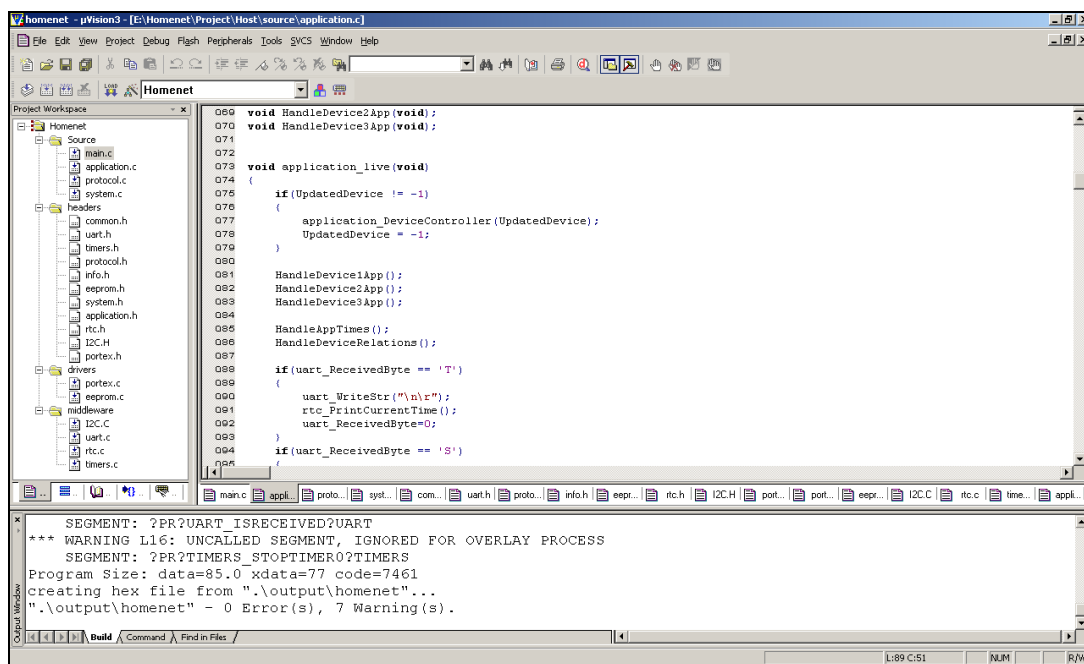


Figure 5.1 Keil Micro Vision editor window.

Keil's 8051 compiler produces an HEX file as a downloadable image to the LPC935 however, there is no programmer tool provides to download the image in to the flash memory of the microcontroller. To do this job there is another tool provided by Embedded Systems Academy, which is called FlashMagic.

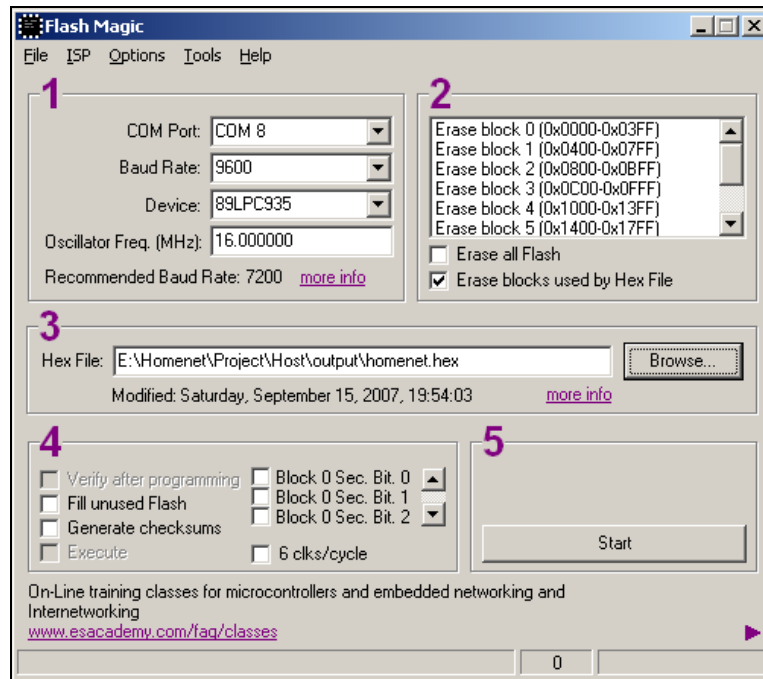


Figure 5.2 Graphical user interface of Flash Magic tool.

On the other hand, to make the downloading more automatized, there is a facility to run the Flash Magic from command prompt that gives ability to write a batch file, which can be run via Keil Micro Vision editor window.

5.2 ECC Application Development Environment

Software Development of the Embedded Control Server is based on the programming language of JAVA from Sun Microsystems. To build and run the written java code there is development environment used that is called Eclipse from Apache Software Foundation.

Both the Port Control Application and Web Server Application are developed in the same platform, which is Eclipse.

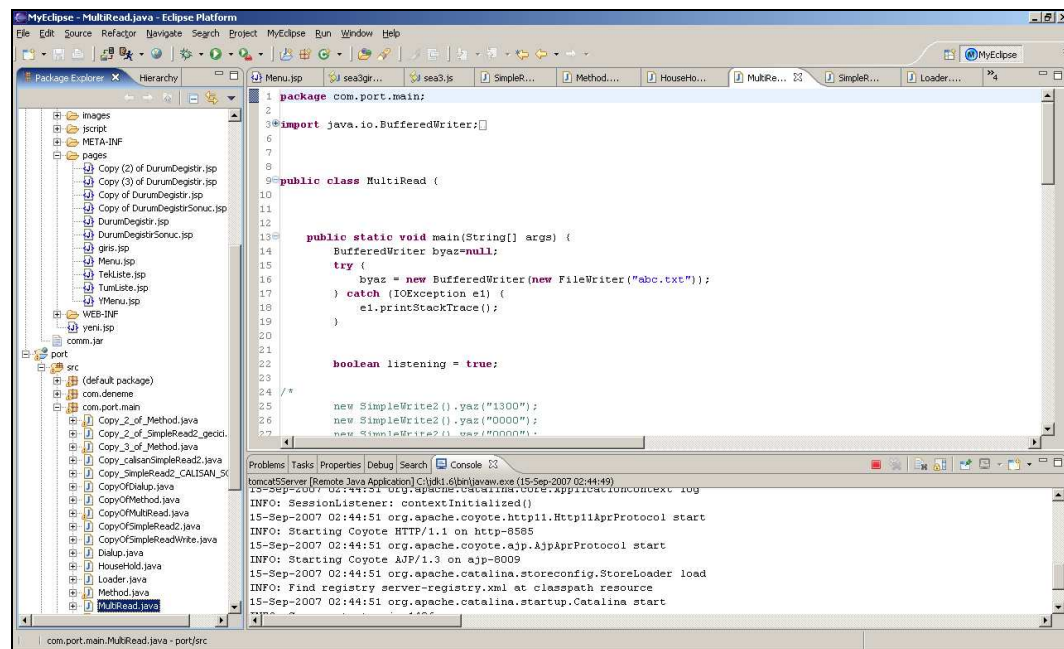


Figure 5.3 Eclipse editor window.

Eclipse consists of java builder, debugger and libraries. It provides to run also Tomcat web server.

To run the Web Server application, also another platform is needed which is called Tomcat. Tomcat is released by Apache Software Foundation.

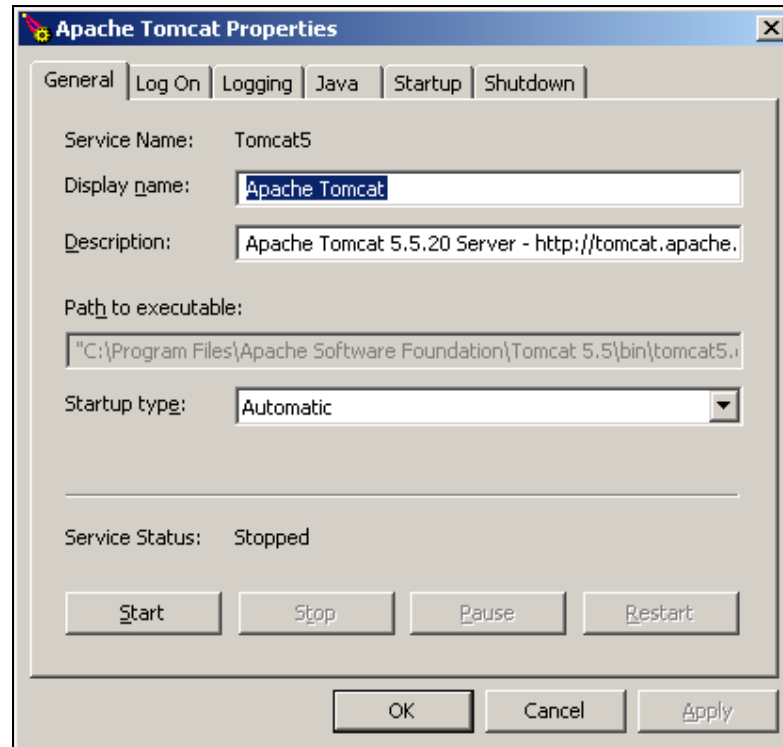


Figure 5.4 Apache Tomcat configuration window.

CHAPTER SIX

APPLICATION PRODUCT SPECIFICATIONS

6.1 Representing the Product and Units

Application product consists of four hardware units. One central control unit, which is called ECS (Embedded Control Server), and three device representative units (represents any of a household appliance) which are including I2C controlled port expanders.

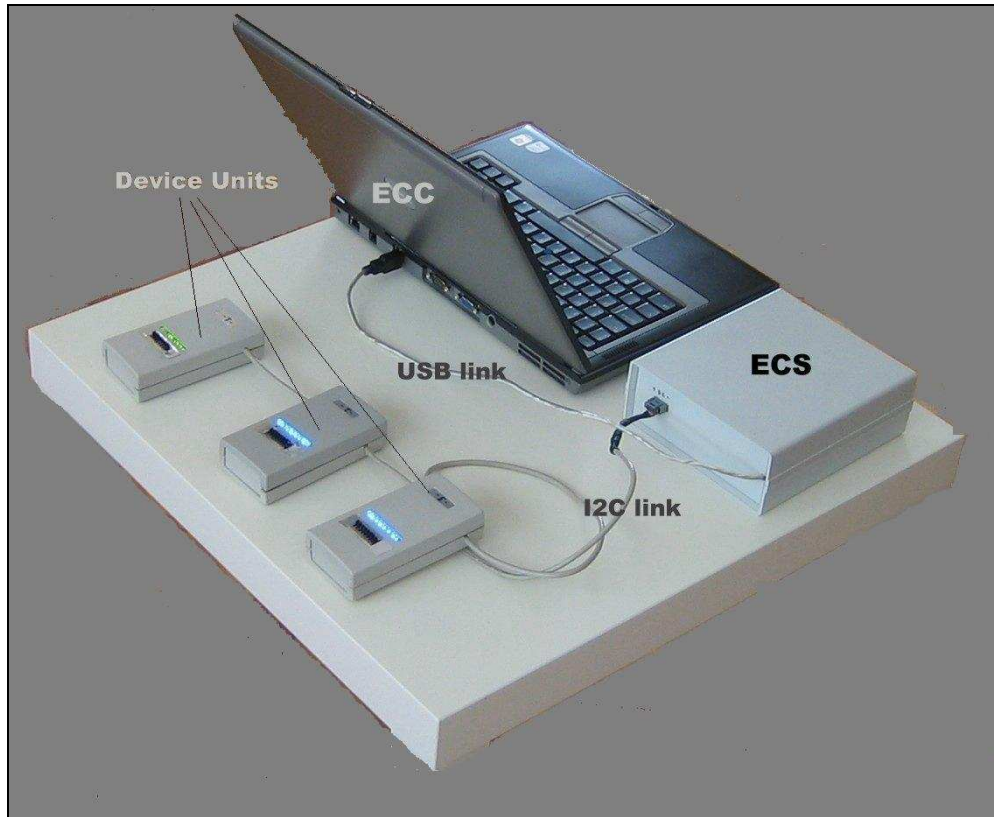


Figure 6.1 Product of the home network application.

Central control unit has onboard microcontroller and its environmental components. It can write 8-bit data to each device and can read 8-bit data from each device.



Figure 6.2 Embedded Control Server (ECS).

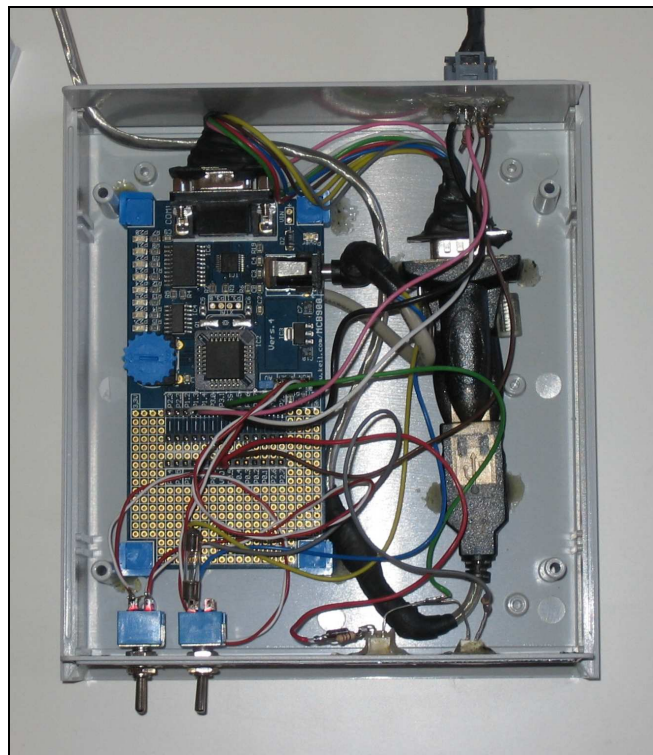


Figure 6.3 Inside of ECS.

There are eight Leds on each of the device unit to show user the current data on the device. In addition, to change the current data on the devices manually, there are eight manual switches to set the each of the bit's state to one or zero.



Figure 6.4 Device Representative Units (DRU).

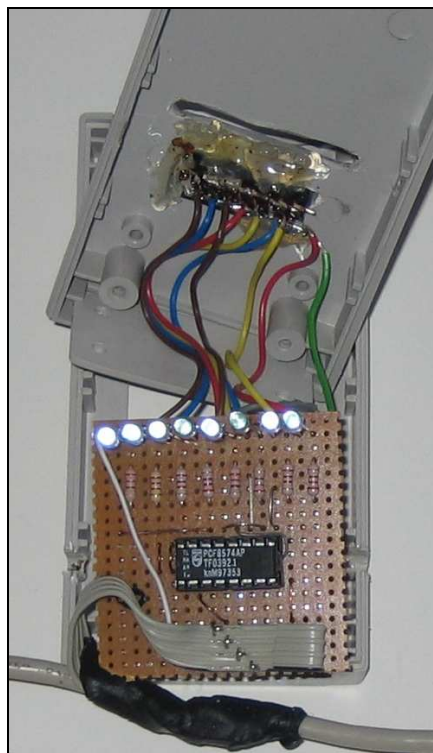


Figure 6.5 Inner side of a DRU.

6.2 Predefined Applications

There are many application can be defined in the ECS and each of the application can be configured via the web interface. Each application has its own defined application code, according to that code ECS decides which application will be configured via web interface. However the applications can be extended freely by easily, write the implementation in the ECS, we have now six of the predefined applications.

6.2.1 *Simply Run*

Application Code of this application is one. After selecting the related device when application code is submitted as one in the WBUI, the data will be written to device 1 as 0xFF, which enables all 8 of the pins of device 1, and all the LEDs will be switched ON without taking care of the previous state of the pins.

After entering one for the related device, rest of the messages to be entered in the WBUI become useless and will not be taken care by the application in the ECS.

6.2.2 *Simply Stop*

Application Code of this application is two. After selecting the related device when application code is submitted as two in the WBUI, the data will be written to the device as 0x00, which enables all eight of the pins of the device, and all the LEDs will be switched OFF without taking care of the previous state of the pins.

After entering two for the related device, rest of the messages to be entered in the WBUI become useless and will not be taken care by the application in the ECS.

6.2.3 Retrieve Significant Data

Application code of this application is 28. After 28 is submitted via WBUI, ECS responds with the message that contains the current states of the each pin of the device. For example, if ECS returns 66, it means that eight of the pins are in order of 01000010.

6.2.4 Write Significant Data

Application code of this application is 29. After 29 is submitted via WBUI, ECS will set the related device's pins to that values in binary form without taking care of the current state of the pins. For example if the data is entered as 66, ECS will set the pins in order of 00111101.

6.2.5 Time Application

Application code of this application is three. This application is used to control the related device according to any of the specified time and period. After entering three as application code, the time parameters also need to be entered. For example, if the user wants to set all the pins of device 1 as ON in the time of 20.09.2007 at 16:49:00 and will keep in this state for 15 minutes. In this scenario the following values should be entered in WBUI;

Appl.Code : 3
 Second : 00
 Minute : 49
 Hour : 16
 Day : 20
 Month : 09
 Year : 07
 Offset : 15
 Offset Unit : Second

Rest of the parameters are not needed to be set

After submitting these values, ECS will start the time application and waits until this entered time. When the system time reaches to the time entered as above, the device is set as the binary value of 00111101 and the related LEDs are switched ON for 15 minutes. After 15 minutes, the value of the device switch back to the pervious value before the time application sets.

6.2.6 Relative Application

Application code of this application is 4. This application is used for establishing a correlation between the devices. Therefore, the state of the devices can be changed according to any of the other device. The value of the device and relative device can be all configured via the WBUI.

For example, if the user wants to set the all pins of Device1 as 1 when the second pin of the Device 2 become 1. In this scenario user should enter the following values for the Device 1 via WBUI;

Appl.Code : 4
Rel.Device : Device 2
Rel.Value : 2
Rel.State : On

Rest of the parameters is not needed to be set.

After submitting these values, ECS will start checking the value of the pin 2 of the Device 2. Whenever it becomes 1, all pins of the device 1 will be set to ON.

CHAPTER SEVEN

CONCLUSION

Today, with decreasing the costs and increasing the expectations of the people on high-tech consumer electronics products, Home Networking concept becomes more popular. Because the people want to be more comfortable and safe at their homes and now they have more chance to find the related products, which cover the expectations of the people at home.

The starting point of this thesis is to show what kind of possibilities can be implemented as a Home Networking sense. Our application can be considered as a Home Control Network by using an embedded system as host unit and a PC application as a client unit.

I choose to use an embedded system in my Home Control Network application because the embedded systems are now more flexible and more powerful to implement especially for control applications at home. Another key point is the robust feasibility and low power consumption rather than PC based systems. I choose the Web Based User Interface as the client of the embedded host, because Internet usage and web based applications are now very common, easy to use for the user, and easy to port to every generic PC systems.

Combination of an embedded system with internet becomes also an attractive application and interested me and others who deal with consumer electronics. The other important point is the possibilities to extend the application according to the type of the home appliance. The physical medium and the protocol to be used for communicating with the devices can be also extended on the platform of the application of this thesis. Then the relation between the devices and ECS can be as client-server model rather than master-client.

From those all aspects, this reference home networking application that I have built up is a good reference for the future home networking applications.

REFERENCES

Hitch Hiker (2001). Tomcat guide. *The "Hitch-Hiker's guide to Tomcat" book*

JDBC driver explanation. Retrieved August 10, 2007 from

http://en.wikipedia.org/wiki/JDBC-ODBC_Bridge

NXP Semiconductors (Founded by Philips) (2007). I2C-bus specifications. *UM10204*.

Matthew Chapman (1994). Programming basis in 8051. *The Final Word On The 8051*.

Philips Semiconductors (1995). Architecture of 8051. *80C51 family programmer's guide*.

Philips Semiconductors (1997). Product specifications of PCF8574. *Datasheet of PCF8574*.

TCP/IP protocol suite. Retrieved August 10, 2007 from

http://www.tcpipguide.com/free/t_TCPIPInternetArchitectureandProtocolSuite.htm

APPENDIX

ABBREVIATIONS

ASD	: Application Specific Data
DMDB	: Device Message Database
DRU	: Device Representative Unit
ECC	: Embedded Control Client
ECP	: Embedded Control Protocol
ECS	: Embedded Control Server
EEPROM	: Electrically Erasable Programmable Read Only Memory
I2C	: Inter ICs Communication
IC	: Integrated Circuit
IDE	: Integrated Development Environment
NVM	: Non-Volatile Memory
ODBC	: Open Data Base Connectivity
PCI	: Peripheral Component Interconnect
SCL	: Serial Clock
SDA	: Serial Data
UART	: Universal Asynchronous Receiver Transmitter
USB	: Universal Serial Bus