FINAL DESIGN REPORT

by

ROYALFLUSH

nothing beats it!

Table of Contents

1. Introduction

This final design report is prepared for the senior design course of 2007-2008 fall semester by the team **RoyalFlush** to present and describe the general design concepts of our project. This report should be considered as an intermediate outcome of the design process. The work done by our team since the beginning of the project and achievements are included throughout the document in a formal way. Since design process consists of modeling the system, the report contains diagrams and models of the current system. As design process is a continuous task. all the diagrams and models are subject to change.

1.1 Project Title

Wireless Network Traffic Monitoring System

1.2 Project Definition

Our project is featuring a wireless network traffic monitoring system. This system examines local area network usage on the AP-400 Wireless Access Point Repeater / Bridge / 4-Port Switch[1] and shows these data such as upload and download statistics on a graphical LCD display. It will be a real-time embedded system which enables every new wireless connection to be registered by the users' MAC address and monitors its effects on the LCD display.

More specifically, the number of computers engaged to this access point, MAC addresses of those machines with logged sessions, their connection speeds, general upload/download statistics, and the last time they were logged in (in date format) will be shown. The usage of the access point in past 24 hours will also be displayed. At the progressing stages of the project, we will connect the device to a computer by means of a serial port and using the embedded Linux system inside the device, we will conduct the necessary operations. These operations will be monitored on the LCD making use of PIC16F877 and other modules. The input on the LCD display will be handled using a four-button controlling system.

1.3 Project Scope

As soon as the project is accomplished, the end-product can be used in a wide scope. Possible application areas of this system are

- WLAN administrators.
- Security professionals.
- Home users who are interested in monitoring their WLAN traffic.
- Programmers developing software for wireless networks.

As an administrator, it is important for you to keep track of the traffic flowing across your network. This does not necessarily mean that you need to be intimately familiar with every single packet sent or received, but you need to know what types of protocols are flowing across your network. Monitoring the network allows you to have a better understanding of how bandwidth is being used. It also helps you to find out if users are running file sharing programs, or if some kind of evil trojan is silently transmitting information in the background. You can see and estimate these facts from upload/download statistics and other information that will be shown on out LCD display. As we mentioned before, security professionals can use this device to be able to follow the users who are logged into the network. Other professional programmers can use this device to be able to get displayed information. Besides, home users can use this device to create networks and display the device information. Overall, the device serves as a helpful system for almost any wireless network context.

1.4 Recent Works

In this section of our report, we would like to give general information about what has been done so far by us about the design concepts of the project.

After the technical meeting conducted with Air-Ties Company, we agreed on mainly dealing with the integration of the compatible GNU C Compiler version (3.4.6) and compiling the kernel of the device so that we could continue our study on the development and testing phases of applications using the tools provided earlier. To add, we paid attention to fully understand the concepts closely related to hardware design during the meeting and be

prepared for further investigations as to how the internal representation of the device can be interacted with the system software (kernel) and man-made applications installed on the device later on.

We also dealt with more specific issues like 802.11 b/g wireless network standards, possible improvements to the current standard and their impact on the wireless technology, optimization of user-level applications so that power and space consumption is minimized. Our main objective in examining all these was to try to come up with a feasible mechanism in which we can continue working with no waste of effort and time, which are vital phenomena in today's world of innovation. Furthermore, we tried to analyze the cost of reducing system requirements of the applications to be run on the device versus eliminating extra cost of space (in terms of memory or external flash media) wasted. Of course, such an analysis cannot be foreseen to produce reasonable outcomes, but we, as prospect engineers, feel ourselves responsible for such sophisticated analyses both before and during the implementation phases of the project.

We have tried to compile the documents from Air-Ties in the Pardus Linux environment and we were successful on this process. Having finished this, we purchased a USB-to-serial port converter to be able to connect the device to computers without serial ports on them. In Linux environment, we managed to boot the device and installed CuteCom[2] program to our computer. To be able to upload and download files with AP-400's system memory, we need a TFTP[3] server. Trivial File Transfer Protocol (TFTP) is a very simple file transfer protocol, with the functionality of a very basic form of FTP; it was first defined in 1980. Since it is so simple, it is easy to implement in a very small amount of memory, an important consideration at that time. TFTP was, therefore, useful for booting computers such as routers which does not have any permanent data storage devices. It is still used to transfer small files between hosts on a network, such as when a remote X-Window System terminal or any other thin client boots from a network host or server. TFTP is based in part on the earlier protocol EFTP, which was part of the PUP protocol suite. In the early days of work on the TCP/IP protocol suite, TFTP was often the first protocol implemented on a new host type, because it was so simple. Using this TFTP server, we uploaded the already compiled files to test the device. In the future, when we make changes at the coding side of the device, we will upload the updated files in this way.

At the first stage, we tried to see the connected users' MAC addresses in CuteCom terminal along with other necessary information like connection speeds and upload/download quotas.. When a user connects to this device via wireless connection, we can see his/her computer's MAC address using the command: *wlanconfig ath0 list*. This command lists the users' MAC addresses. These commands are MadWifi[4] commands. MadWifi is one of the most advanced WLAN drivers available for Linux today. It is stable and has an established knowledgebase. The driver itself is licensed under open source but depends on the proprietary Hardware Abstraction Layer (HAL) that is available in binary form only. If we want to save this information, we can write it under "tmp" folder of the device (flash memory of the device) by writing *wlanconfig ath0 list >>mac.txt*. This command saves this **.txt* file in the *tmp* folder. The contents of this file are as follows.

```
DDR               AID CHAN RATE RSSI  DBM IDLE  TXSEQ  RXSEQ CAPS ACAPS ERP
S00:15:00:19:67:99    1   11   1M   66  -29  135     21   9040 ESs         0\0x1b[21;1H
```

After doing this, we decided to work on C environment rather than writing a shell script. Our first aim was to get the MAC addresses of the users from this complicated file. For instance, in the above file, the MAC address of the user is :"00:15:00:19:67:99". To be able to do this, we wrote a C code like the one below.

```
#include <stdio.h>
#define COMMAND "mac"
#define MAC_LENGTH 17
int main(int argc, char* argv[])
{
   FILE *fp;
   char *line, **mac;
   int i = 0, total = 0;
   if (argc != 2)
   {
      printf("Usage: %s <input_file>\n", COMMAND);
      exit(0);
   }
   fp = fopen(argv[1], "r");
   line = (char*) malloc(sizeof(char) * 81);
```

```
    while (fgets(line, 81, fp) != NULL)
    {
        mac = (char**) malloc(sizeof(char*));
        *mac = (char*) malloc(sizeof(char)*MAC_LENGTH);
        for (i = 63; i < 80; i++)
        {
            printf("%c", *(line + i));
        }
        printf("\n");
        total++;
    }
    fclose(fp);
    printf("\n%d user(s) online", total);
    return 0;
}
```

To be able to compile this code, we needed the compilers from the tool chain. These compilers were cross-platform compilers so when we compile our code in these compilers, we easily ran our code in other environments like in the embedded Linux environment of AP-400.

Other than these works, we worked on the CEng336 board and PIC16F877. We will use this board to simulate some applications before the main phases of the project and we will also use PIC16F877 to be able to communicate with our graphical LCD display.

2. Data Design

2.1 Embedded File Systems

2.1.1 JFFS

JFFS, the Journaling Flash File System[1], is an efficient and easy to manage file system designed mainly for embedded Linux systems with Flash memory devices. Unlike its

non-Flash counterparts, JFFS employs wear-leveling[2] that accounts for an update mechanism for altered records by means of nodes, which are usually stored in the format described below.

A header section containing metadata (values or indicators that the information stored in this part will not be recognized as valid data, hence should not be used) is written at the beginning of each record. This is mainly due to the necessity of identifying records of data in a qucik and efficinet way. Following this is the data part, where actual data is stored. This part should be taken care of, since any damage to this section of the record may corrupt the record fully, regardless of the fact that the record may be recovered by means of the metadata at the beginning of it..

At the beginning of the update process, all nodes are marked as *valid*, and those changing in time become *obsolete*. As the number of obsolete records gets larger, the efficiency of the entire structure decays due to the nature of the file system pretending to be a circular log. The mounting process requires all node chains to be read from the beginning to the end, which underrates JFFS as compared to classical UNIX file systems like ext2.

2.1.2 JFFS2

JFFS2[3], the successor of the Journaling Flash File System, is an enhanced version of Linux file systems supported by Linux kernel versions not older than 2.4.10. The new system is also available for RedBoot bootloader and three efficient compression algorithms as well as a garbage collection routine compensating for space losses in JFFS.

JFFS2 treats the disk as a collection of blocks rather than a circular log, which brings about a substantial change in the representation of data; namely, *dirty* and *free* blocks (those with all valid nodes, and some obsolete nodes) constitute the mainframe of the whole system. The garbage collector is responsible for making a copy of valid nodes in a new block and mark the block as a free block, saving empty space after all.

2.2 Data Structures

The project employs an enhanced data structures mechanism in order to sustain efficiency and performance during the execution of the modules. For this reason, we found it helpful to examine the general aspects of our design considerations regarding the most important part of the system: Data.

2.2.1 Temporary Data Handling

User data constituted by MAC address, connection speed, upload/download bit rate, and the latest time the user is logged into the system will be integrated in the main application to be executed on AP-400. This way, temporary memory allocation is to be taken as serious with regard to the amount of data stored in device memory. It has been observed many times that the maximum amount of memory that can be accessed by an application is about 2 Megabytes. However, due to the innerworkings of the machine and the embedded system on top of it, we are capable of reaching only a small portion of this amount, namely 50 Kilobytes. It is also noteworthy that the embedded circuitry we will be using will possess no more than 368 bytes of nonvolatile memory (as in PIC16F877 series standard). For this reason, we are tempted to decide on the modules the main application is going to call during lifetime of the program. The main routines can be summarized as follows.

Routine Name	**Expected Number of Calls**	**Relative Complexity(*)**
Startup screen	1 (**)	1
Menu 1	100 – 1000	2
Menu 2	100 – 1000	3
Menu 3	100 – 1000	2
Menu 4	100 - 1000	2

(*) This term is used for comparison purposes only. It accounts for the relative number of operations executed by a module.

(**) For this splash screen, expected number of calls is equal to the number of maximum number of calls, so no additional complexity is of interest.

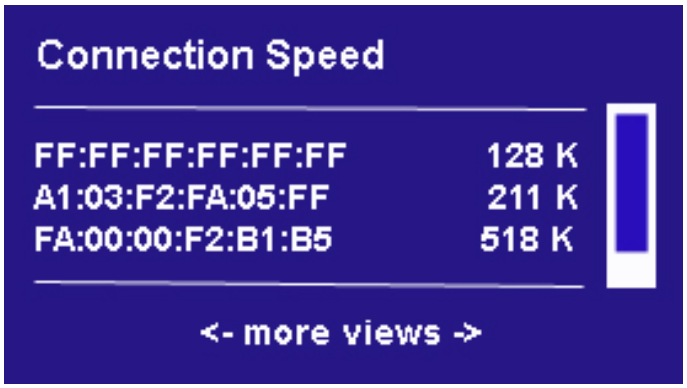
In the table above, startup screen contains a fixed array of characters to be displayed on the LCD as soon as the program starts to run. Regarding that this screen will be displayed

only once, memory consumption for this module is minimal. The extra amount consumed for the time is out of interest since this is the only screen this module exhausts system resources.

As soon as the user switches to a different screen, the memory allocated for the welcome string and the date will be freed. Thus, the system will be able to use all of its available memory for the next operation. A sample startup screen is illustrated below.

Startup screen

Menu 1, as shown below, displays MAC addresses along with connection speeds of devices connected to AP-400 at run-time. The scrolling bar on the right hand side of the screen is designed to navigate in vertical directions, up and bottom. In case the number of devices exceed the maximum number of data lines displayed on the LCD screen (this figure is assumed to be exactly 3 for all the screens in the project), a *bar scrolling algorithm* which takes linear time helps us to control the behavior of the items displayed. A brief description of this algorithm is given below.



Menu 1

At the very first execution of the algorithm, top of the scrolling bar is shown at the topmost position of the right section of the screen. As soon as the user presses the *Down* button, the bar scrolling algorithm is employed and the new position of upper and lower parts of the scrolling bar are computed according to the following formulas.

$P'_{top} = P_{top} - (W / N)$ ...(*)
$P'_{bottom} = P_{bottom} - (W / N)$

where
N is the number of devices logged into the system,
W is the length of the white rectangle encapsulating the scrolling bar (in pixels),
P_{top} is the vertical position of the upper part of the scrolling bar just before the user presses the DOWN button,
P_{bottom} is the vertical position of the lower part of the scrolling bar just before the user presses the DOWN button,
P'_{top} is the vertical position of the upper part of the scrolling bar just after the user presses the DOWN button,
P'_{bottom} is the vertical position of the lower part of the scrolling bar just after the user presses the DOWN button.

Horizontal position of the scrolling bar is fixed; thus, it does not need recomputing upon user input.

(*) Reasoning that W and N are both integers, the result is always an integer, which avoids several incompatibilities such as floating point arithmetic, division by zero flaws corrupting the current routine. It must be also noted that a trade-off between computational complexity and accuracy takes place. We are biased with computational complexity since for larger values of N, the accuracy of the position of the scrolling bar does almost give exact values, especially for those N's exceeding 100.

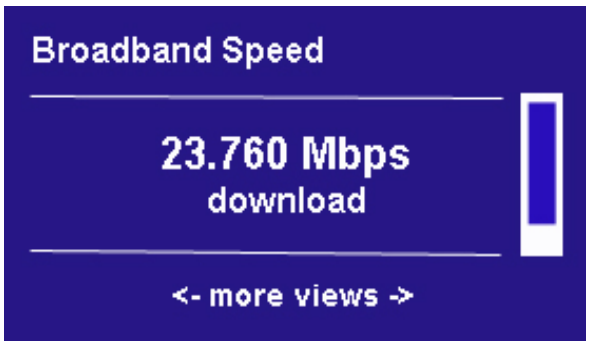
An investigation about the size of data records of interest (MAC address and connection speed of a user) reveals that 21-byte storage (17 bytes for IEEE 802 MAC address, and 4 bytes for a signed integer value on 32-bit devices) is needed for each user. Considering that a user displays this screen five times in a session and the number of devices connected to AP-400 at the same time does not exceed 50, this screen takes up approximately 1 KB of

system memory excluding the fixed amount of characters to be printed on the screen for readability. The main reason why this module has a degree-2 complexity is that the screen refreshes itself whenever a new connection is detected and users whose sessions have already expired are discarded at the same time.

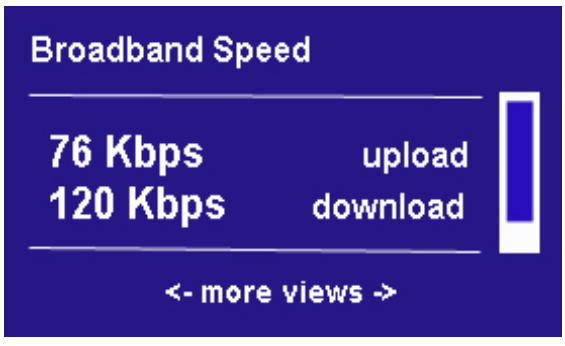
Menu 2 has even a higher relative complexity now that it requires the recent records to be updated using file I/O operations. A user already logged in a session may want to see his/her up-to-date records dating back no more than the last 24 hours. Then, the relevant information is sought in recent records and fetched from the disk, merged with user's current data and displayed. The file I/O being much slower as compared to operations taking place in main memory, this process is likely to be the most costly part of the entire system.

Menu 2

Menu 3 and Menu 4 both display upload/download statistics, the former showing the total traffic flowing through the system, and the latter yielding a specific data for each user in terms of upload and download bandwidths in Kilobits per second (Kbps). Basic layouts for these menus are shown below.



Menu 3



Menu 4

2.2.2 Permanent Data Storage

2.2.2.1 TXT File Format

Plain text files with .txt extension will be used extensively throughout the project, both for reading and writing purposes. The data of interest constitute a proper subset of ASCII encoding format, which allows us to refrain from using sophisticated encoding change algorithms mainly used for Unicode UTF-8 and UTF-16 encoding formats.

We are planning to use text files as an intermediate step during run-time process as can be seen in the following scenario.

1. The user connects to AP-400
2. As soon as the user logs into the system, recent records (not earlier than past 24 hours) of the records in the database are checked for equality.
3. In case a previous record of the user is found, the relevant data is fetched from the database record file (CSV) and read into a dynamically allocated space in system memory.
4. If no recent record is found for the user, a new database record is created and a dynamically allocated space in memory is marked as a new record and saved to a plain text file (TXT).
5. As soon as the user logs off, his/her database record is updated using the data record in system memory. If no user data is found previously, data record in system memory is written to the database file for the upcoming 24 hour interval.

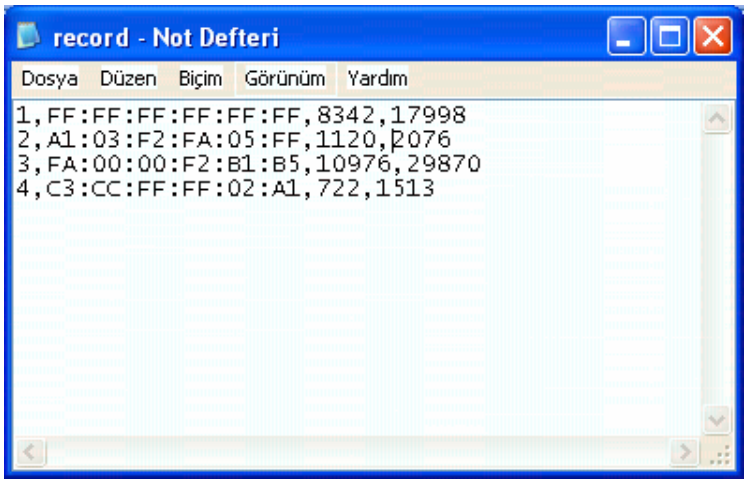
2.2.2.2 CSV File Format

2.2.2.2.1 General Specifications

CSV file format can be regarded as a primitive database structure which is a comma delimited plain text file making I/O operations much simpler than a unformatted text file (TXT) does. For instance, the following database structure shown as a table can be illustrated as a CSV file.

ID	MAC Address	Upload Quota(KB)	Download Quota(KB)
1	FF:FF:FF:FF:FF:FF	8342	17998
2	A1:03:F2:FA:05:FF	1120	2076
3	FA:00:00:F2:B1:B5	10976	29870
4	C3:CC:FF:FF:02:A1	722	1513

Data Records as a Table



As can be seen in the above figures, space consumption in Flash disk of the device is 29 bytes. To account for this, it should be helpful to note that ID (identification number of a record) occupies as many as 4 bytes even though it is likely to be within the range [0, 65536], which can be stored using only two bytes. However, the cross-compiling software mips-linux-gcc[4] behaves the data type *short int* as a default *int* value, so we are to use 4 bytes for this field as we do for upload quota and download quota fields.

IEEE 802 MAC address format can be represented as a 17-byte array of characters; thus, a total of 29 bytes are used.

2.2.2.2.2 Record Insertion

Record insertion routine is designed to be used when a user is yet to have a recent data in the database (stored in CSV files). User data to be entered into the main record file is gathered from three system functions; getMac(), getUploadQuota(), and getDownloadQuota(). Prototypes of these functions are listed in the table below.

Function Name	Return Value	Parameters
getMac()	char*	Void
getUploadQuota()	int	int ID = 0
getDownloadQuota()	int	int ID = 0

The following code snippet will be used to insert records into CSV files.

```
#include <stdio.h>
#include "customfunc.h" /*the functions getMac(), getUploadQuota(),
                          and getDownloadQuota() are defined here*/
#define MAC_SIZE 17

int main(void)
{
   char ch, *mac_address, *id;
   int line_count = 0, id_size = 1, ID, upload_quota, download_quota;
   FILE *fp;
   mac_address = (char*) malloc(sizeof(char) * MAC_SIZE);
   mac_address = getMac();
   upload_quota = getUploadQuota(0); /*no ID passed since recent record
                                       search is out of question*/
   download_quota = getDownloadQuota(0);
   fp = fopen("record.csv", "r");
   fseek(fp, -35, SEEK_END);
   do
      ch = fgetc(fp);
   while (ch != '\n' && ch != '\r');
   id = (char*) malloc(sizeof(char));
   while (1)
   {
      ch = fgetc(fp);
      if (ch == ',')
         break;
      else
```

```
        {
            id = (char*) realloc(id, sizeof(char)*id_size);
            *(id + id_size - 1) = ch;
            id_size += 1;
        }
    }
    fclose(fp);
    ID = atoi(id);
    free(id);
    ID += 1;
    fp = fopen("record.csv", "a");
    fprintf(fp, "%d,%s,%d,$d\n", ID, mac_address, upload_quota, download_quota);
    free(mac_address);
    fclose(fp);
    return 0;
}
```

2.2.2.2.3 Record Deletion

Record deletion routine is employed in case no recent record is found in the most recent version of the main database file. The records to be deleted are identified via their unique ID number, and marked as deleted. Then, the following code portion is used to copy the records not marked to a backup file.

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    char ch, *id, *line;
    int id_size = 1, i, del_mark;
    FILE *ip, *op;
    ip = fopen("record.csv", "r");
    line = (char*) malloc(sizeof(char)*35);
```

```
while (fgets(line, 35, ip) != NULL)
{
   id = (char*) malloc(sizeof(char));
   while (1)
   {
      ch = line[id_size - 1]; /*read the current line one character at a time*/
      if (ch == ',')
         break;
      else
      {
         id = (char*) realloc(id, sizeof(char)*id_size);
         *(id + id_size - 1) = ch;
         id_size += 1;
      }
   };
   del_mark = 0;
   for (i = argc; i > 1; i--)
   {
      if (strcmp(argv[i - 1], id) == 0) /*equality test for ID values*/
      {
         del_mark = 1;
         break;
      }
      else
         continue;
   }
   if (del_mark == 0) /*if no matching record is found,
                  copy the current line into a fresh file*/
   {
      op = fopen("safe_record.csv", "a");
      fputs(line, op);
      fclose(op);
   }
```

```
        free(id);
    }
    fclose(ip);
    return 0;
}
```

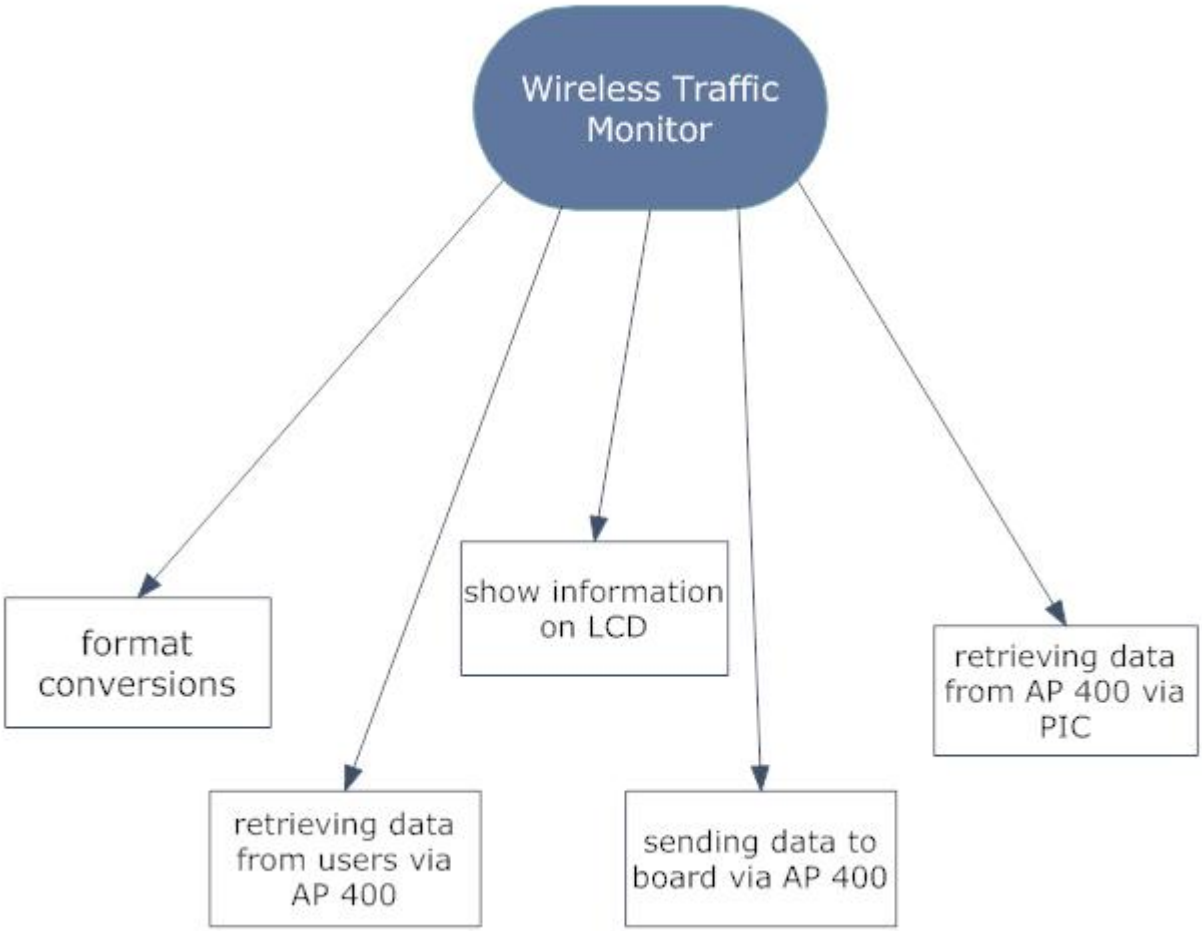
2.2.2.2.4 Record Update

Record update routine is called if and only if a recent record of a user is also existent in the most recent version of the database file. In this case, the file "record.csv" is searched for the ID number of the record and update process is accomplished. As shown in the above code snippet, records marked as updated are copied to a safe file and then written back to the original database file after resetting.

3. Data Modeling

3.1 Structural Design

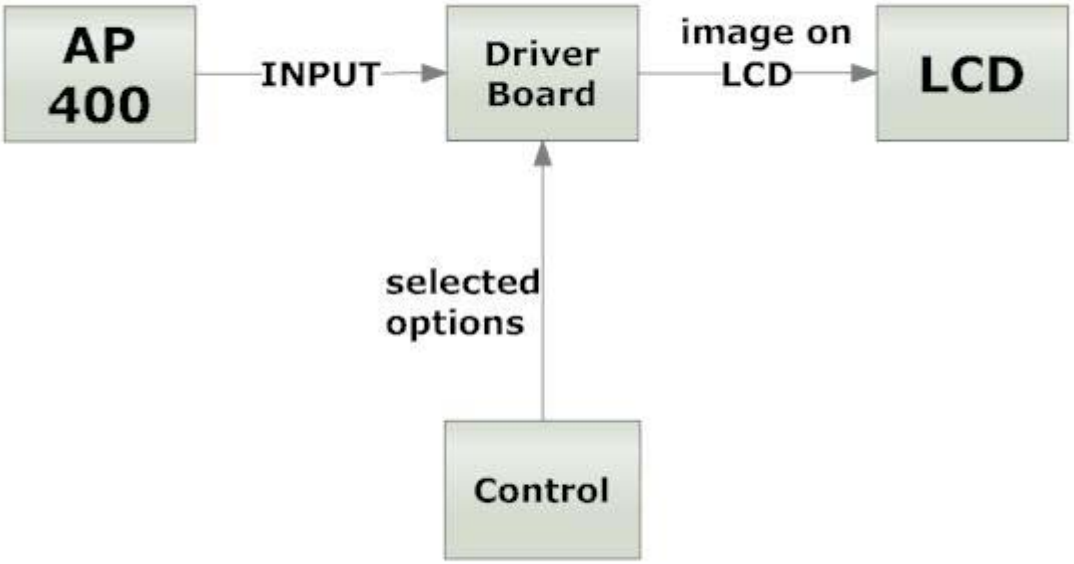
As we mentioned above, our project is about a traffic monitoring system. As you can see on the diagram, our system firstly retrieve data from users via AP400. This process is conducted by the button configuration. It sends data from AP400 to board via RS232. Final messages are displayed on the graphical LCD display. It retrieves data from AP400 via PIC. While these processes, our system makes necessary format conversions.



3.2 Dataflow Diagrams

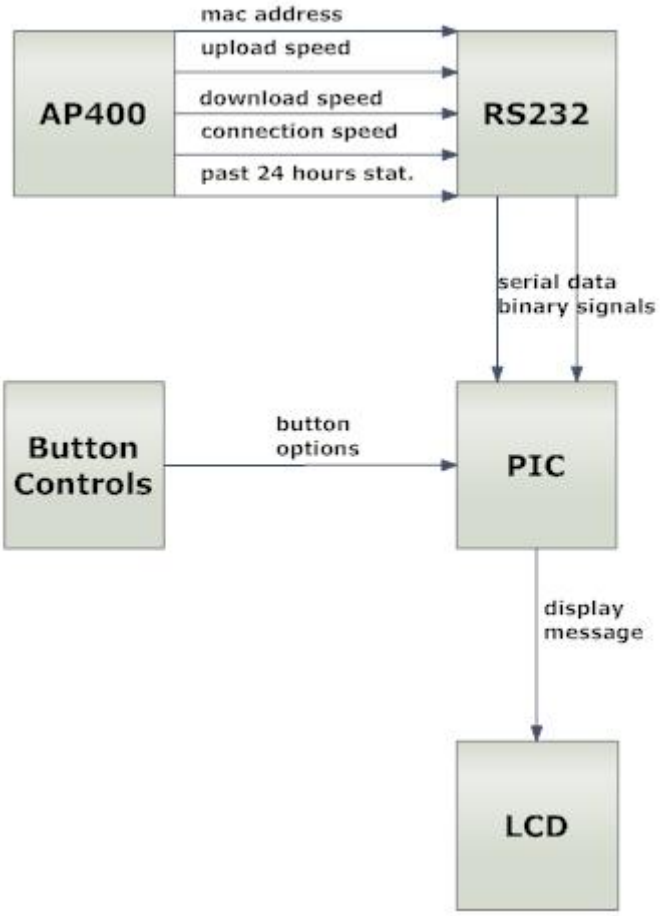
3.2.1 DFD level 0

In this first level data flow diagram, AP400 gives input to the driver board. Driver board is controlled by button configuration system. This driver board displays data on the graphical LCD display.



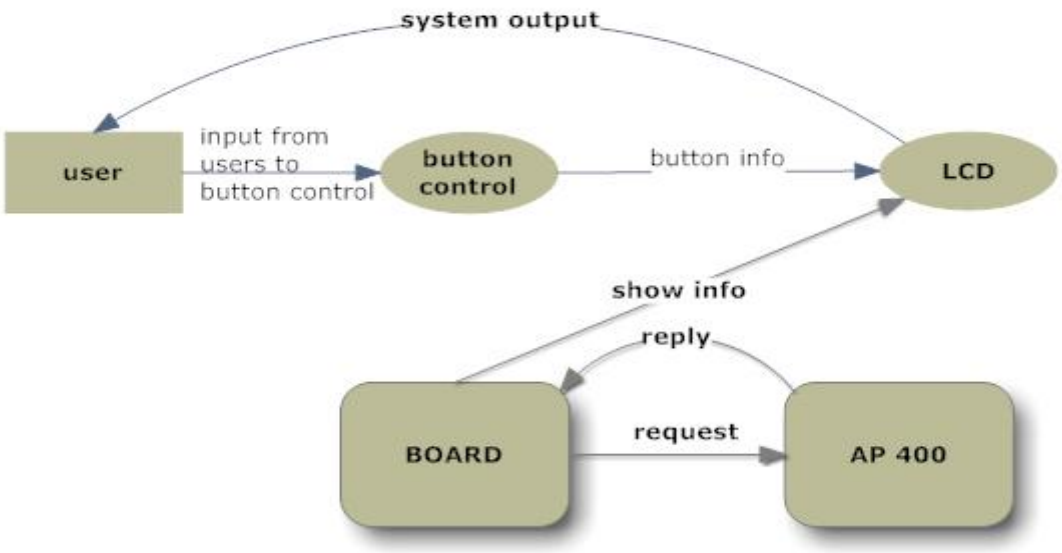
3.2.2 DFD level 1

At this level, we illustrated the connections between components. Firstly, AP400 gives necessary information to RS232, such as MAC addresses of the users, upload and download quota of the users, connection speed of the users and past 24 hours statistics. After this process, RS232 gives serial data binary signals to PIC processor. This PIC processor gets control signals from user via buttons. After this, this PIC processor displays necessary information on graphical LCD display.



3.2.3 DFD level 2

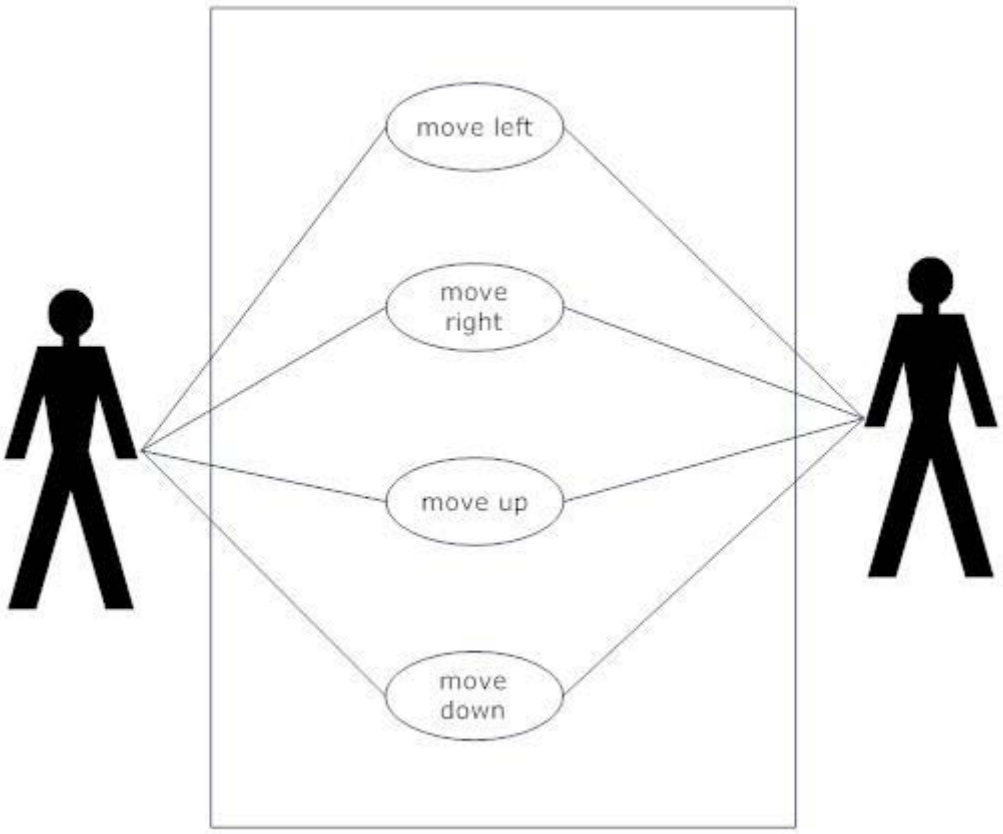
In this diagram, you can see the detailed structure of the system.



3.3 Use-Case Diagrams

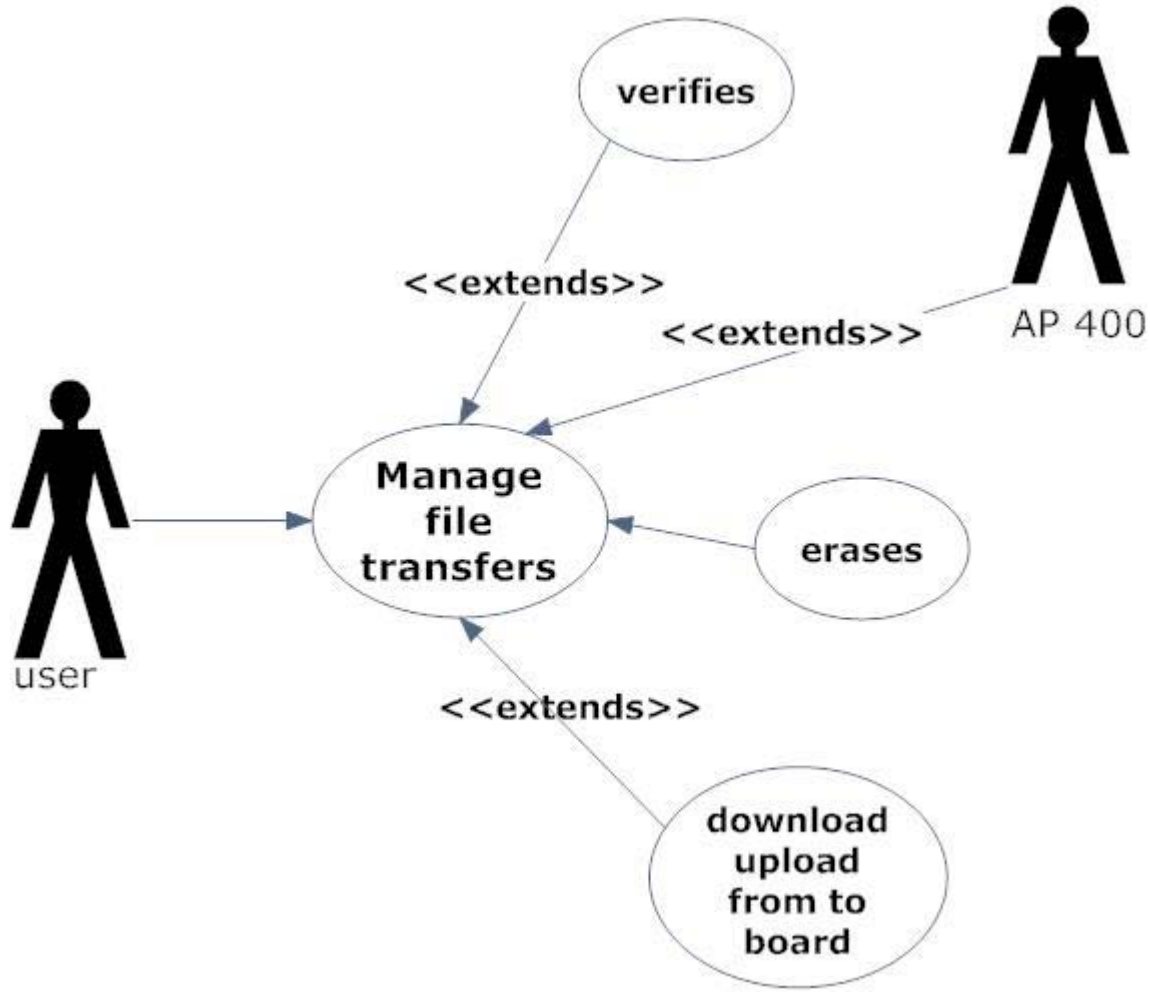
3.3.1 Level 0

In this use-case diagram, you can see the interaction between the user and the LCD display. The user can control the LCD display via buttons like move_right (RIGHT), move_left (LEFT), move_down (DOWN), and move_up (UP).

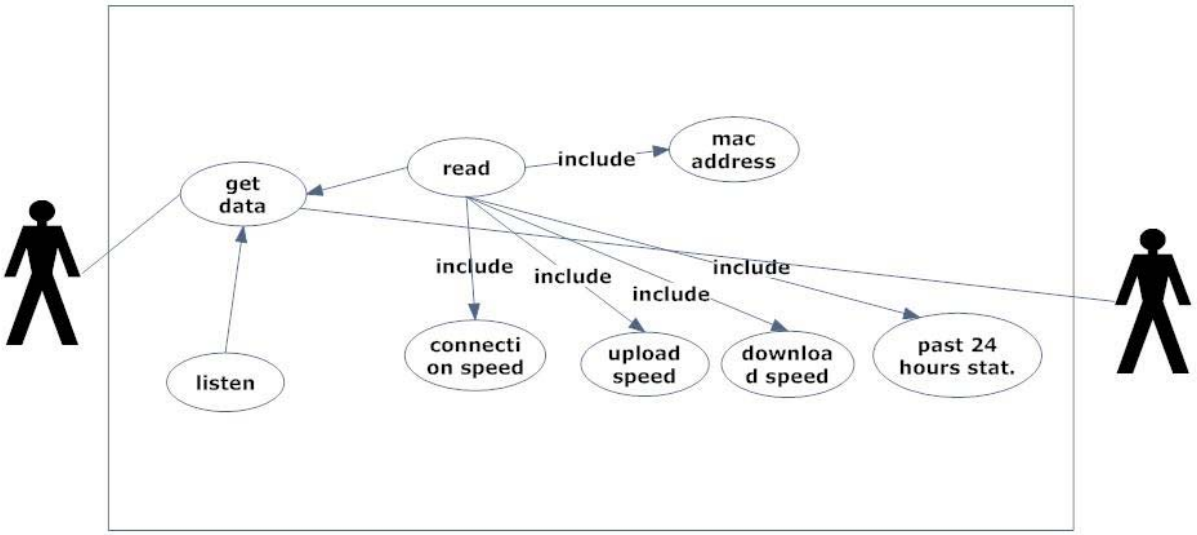


3.3.2 Level 1

At this level of the diagram, you can see the detailed outline of the file transfer management. In this system, file verification, file removal and download/upload processes from/to board take place.

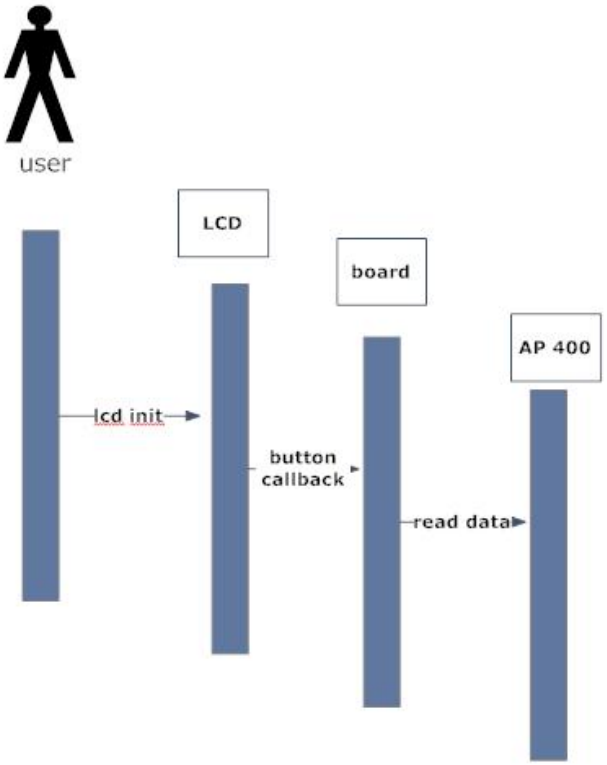

verifies
<<extends>>
<<extends>>
AP 400
user
Manage file transfers
erases
<<extends>>
download upload from to board

3.3.3 Level 2

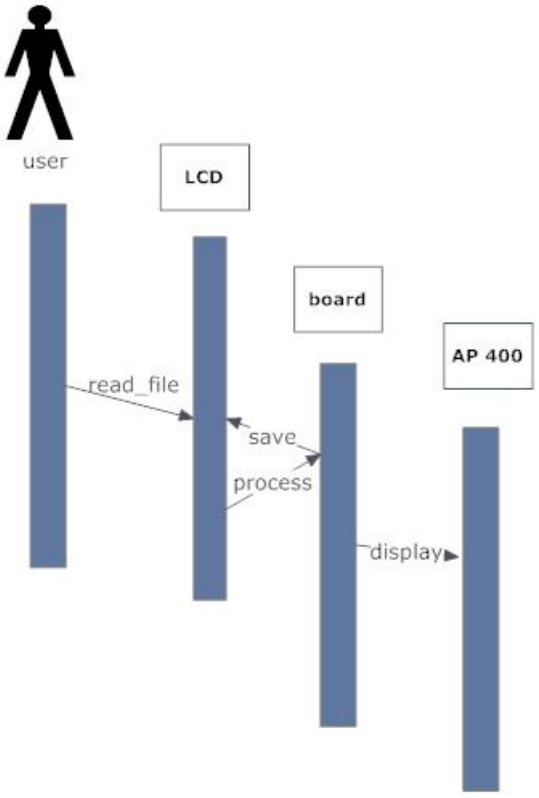

read
include
mac address
get data
include
include
include
include
connecti on speed
upload speed
downloa d speed
past 24 hours stat.
listen

3.4 Sequence Diagrams

3.4.1 Level 0



3.4.2 Level 1



4. Hardware Components

4.1 PIC16F877

We firstly give some information about microcontrollers than pass to the our main topic that is PIC16F877. Since PIC16F877 is very essential part of our Project we analyze the structure in detail and the development tools of it in order to make clear some points. According to information which we taken from the websites of microchip, a microcontroller is a compact standalone computer, optimized for control applications. Entire processor, memory and the I/O interfaces are located on a single piece of silicon so, it takes less time to read and write to extrernal devices.

The elements below are what for microcontrollers are incorporated in control systems:

a) Cost: Microcontrollers with the supplementary circuit components are much cheaper than a computer with an analog and digital I/O.

b) Size and Weight: Microcontrollers are compact and light compared to computers.

c) Simple applications: If the application requires very few number of I/O and the code is relatively small, which do not require extended amount of memory and a simple LCD display is sufficient as a user interface, a microcontroller would be suitable for this application.

d) Reliability: Since the architecture is much simpler than a computer it is less likely to fail.

e) Speed: All the components on the microcontroller are located on a singe piece of silicon. Hence, the applications run much faster than it does on a computer.

Now we will give some properties of PIC16F877 that is one of the most commonly used microcontroller and we use it in our Project in order to monitor gathered statistics from AP-400.

4.2 PIC16F877 Core Features

⇒ Accumulator Based Machine

⇒ High performance RISC CPU

⇒ 14 bit cores with 35 instructions. .

⇒ 3 Addressing Modes (direct, indirect, relative)

⇒ Harvard Architecture Memory (separate program and data memory)

- 8Kx14 Flash Based Instruction Memory
- 368x8 Static Ram Based Data Memory (File Registers)

⇒ 8x13 Hardware Stack (8 levels - not visible from program code)

⇒ Operating speed: DC - 20 MHz clock input

DC - 200 ns instruction cycle

⇒ Up to 14 interrupt capability

⇒ Programmable code protection

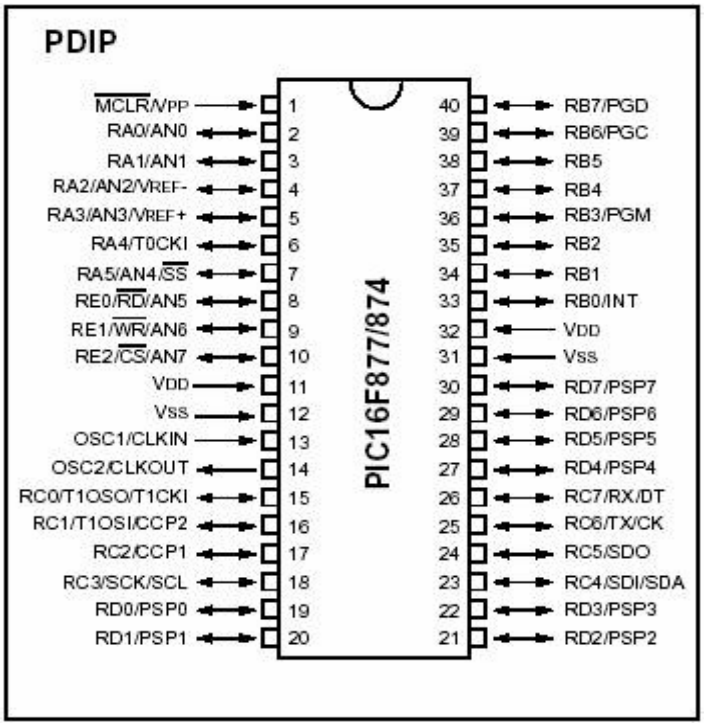
⇒ Single 5V In-Circuit Serial Programming capability

⇒ Wide operating voltage range: 2.0V to 5.5V

⇒ Low power, high speed CMOS FLASH/EEPROM technology

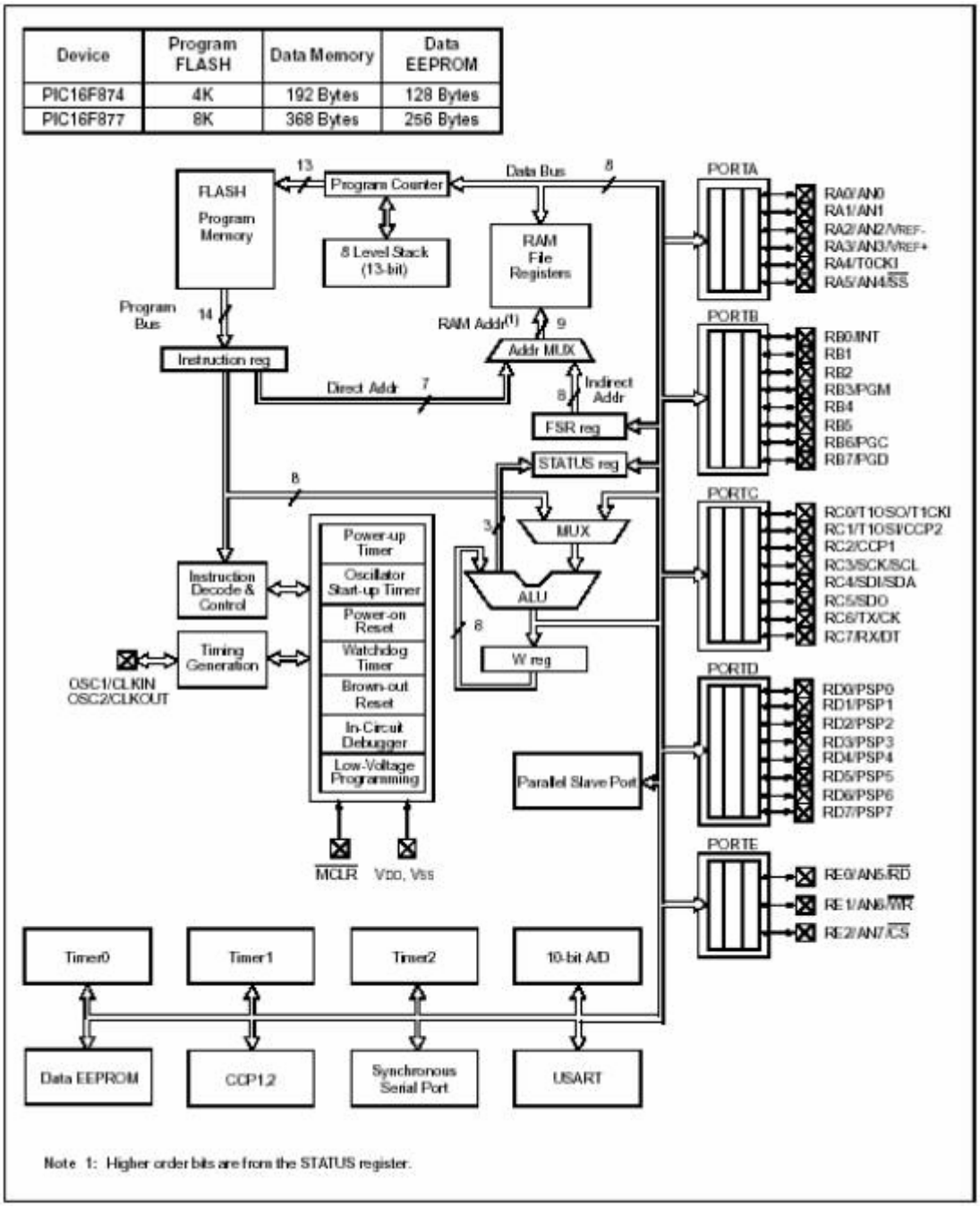
4.3 PIC16F877 Peripheral Features

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler
- Timer2: 8-bit timer/counter with 8-bit period, register, prescaler and postscaler
- Transmitter (USART/SCI) with 9-bit address
- Parallel Slave Port (PSP) 8-bits wide, with external RD, WR and CS controls (40/44-pin only)
- 10-bit multi-channel Analog-to-Digital converter
- Synchronous Serial Port (SSP) with SPI. (Master mode) and I2C. (Master/Slave)
- 256 bytes of EEPROM Memory



PIN DIAGRAM OF PIC16F877

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874	4K	192 Bytes	128 Bytes
PIC16F877	8K	368 Bytes	256 Bytes



BLOCK DIAGRAM OF PIC16F877

4.4 PIC16F877 Development Tools

- **CCS PIC-C COMPİLER**

The CCS compiler is specially developed to meet the special requirements of the microchip PIC. These tools allow developers to quickly design application software for these controllers in a highly readable, high-level language.

The compilers has some limitations when compared to a more traditional C compiler. The hardware limitations make many traditional C compilers ineffective. As an example of the limitations, the compilers will not permit pointers to constant arrays. This is due to the separate code/data segments in the PICmicro MCU hardware and the inability to treat ROM areas as data. On the other hand, the compilers have knowledge about the hardware limitations and do the work of deciding how to best implement your algorithms. The compilers can efficiently implement normal C constructs, input/output operations and bit twiddling operations.

The compiler can output 8 bit hex, 16 bit hex, and binary files. Two listing formats are available. Standard format resembles the Microchip tools and may be required by some third-party tools. The simple format is easier to read. The debug file may either be a Microchip .COD file or Advanced Transdata .MAP file. All file formats and extensions are selected via the Options|File Formats menu option in the Windows IDE.

- **MPLAB**

An Integrated Development (IDE) which is a free integrated toolset for the development of embedded applications.

– Editor

– Build Tools (Assembler and Linker)

– Simulator

4.5 CENG Embedded Board

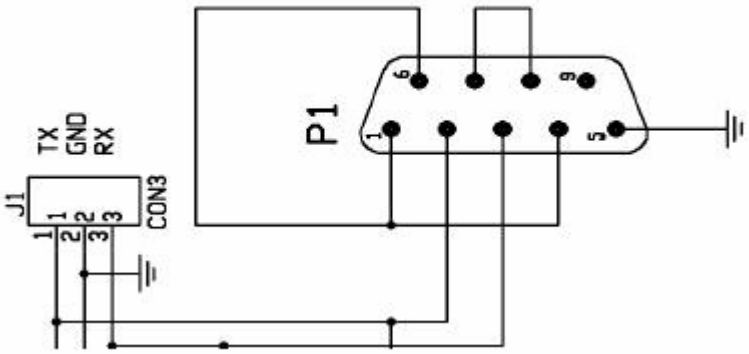
CEng embedded system board is the board that is used in our course CENG 336 embedded system. This board includes two PIC processors and various interfaces like LCD, Parallel, Serial, USB ports, smartcard reader, LED's etc.

The board has the below properties;

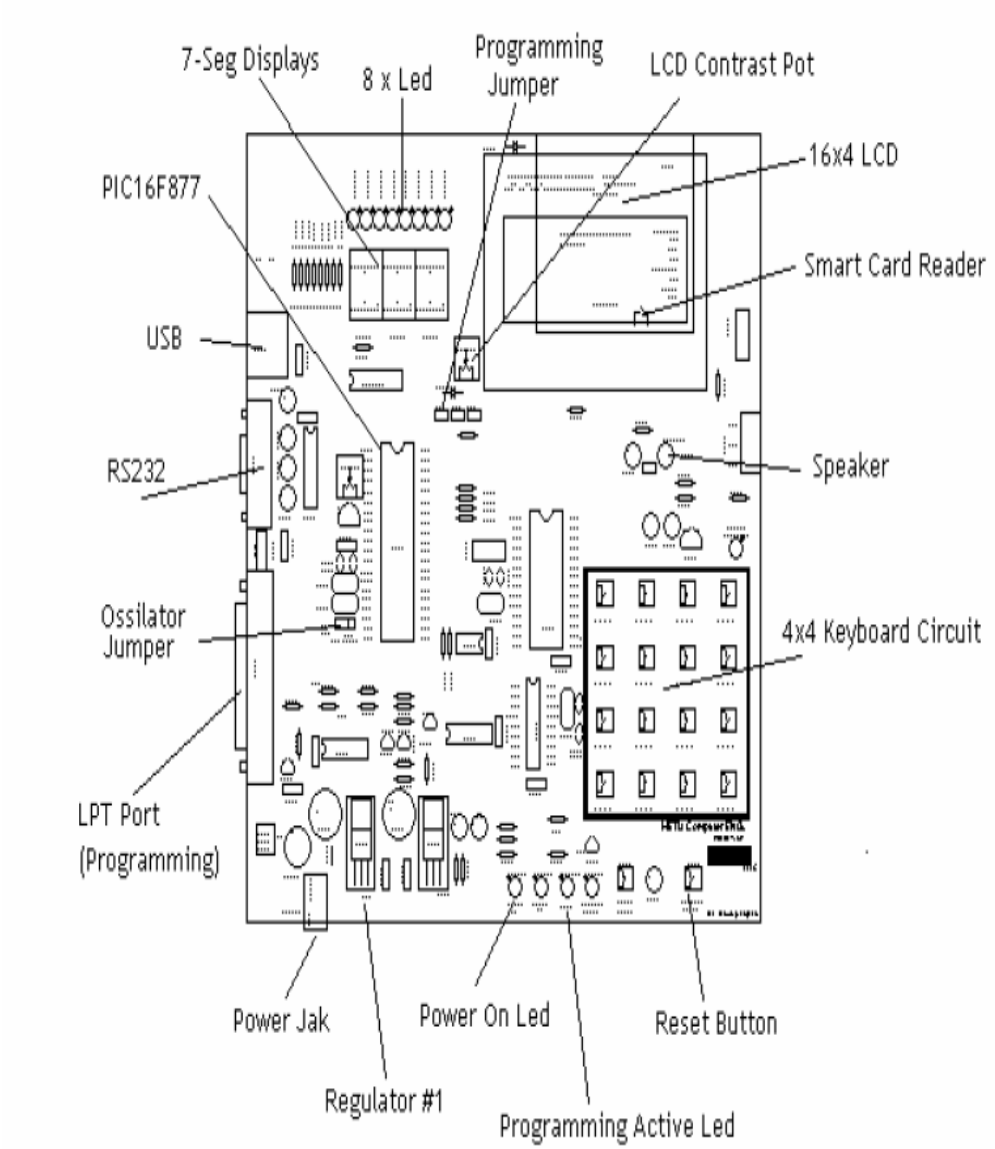
- It has 7-seg displays and 8×Led.
- It has a 4×4 keyboard circuit and smart card reader.
- It has 16×4 characters to lcd. Thus we will not consume time to display characters and lines.
- We do not need additional devices for programming PIC16F877. The board has a parallel port and programmer kid for processors.
- There is no need for additional configuration of buttons on board. They are connected to B-port of PIC16F877.
- It also has the inputs and outputs RS232.

\

An important issue that we want to make clear is the board gather information from the environment and sent it back over the RS232 cable. RS-232 communications will be used as the data interface between the AP-400 and the ceng embedded board. The information must be decoded such that both chips can understand which information is wanted or sent. And also a RS232 serial cable will do if we have a serial port on the PC; otherwise, we will have to use a serial-to-USB converter to establish a connection between the working station (the device itself) and the development environment,.

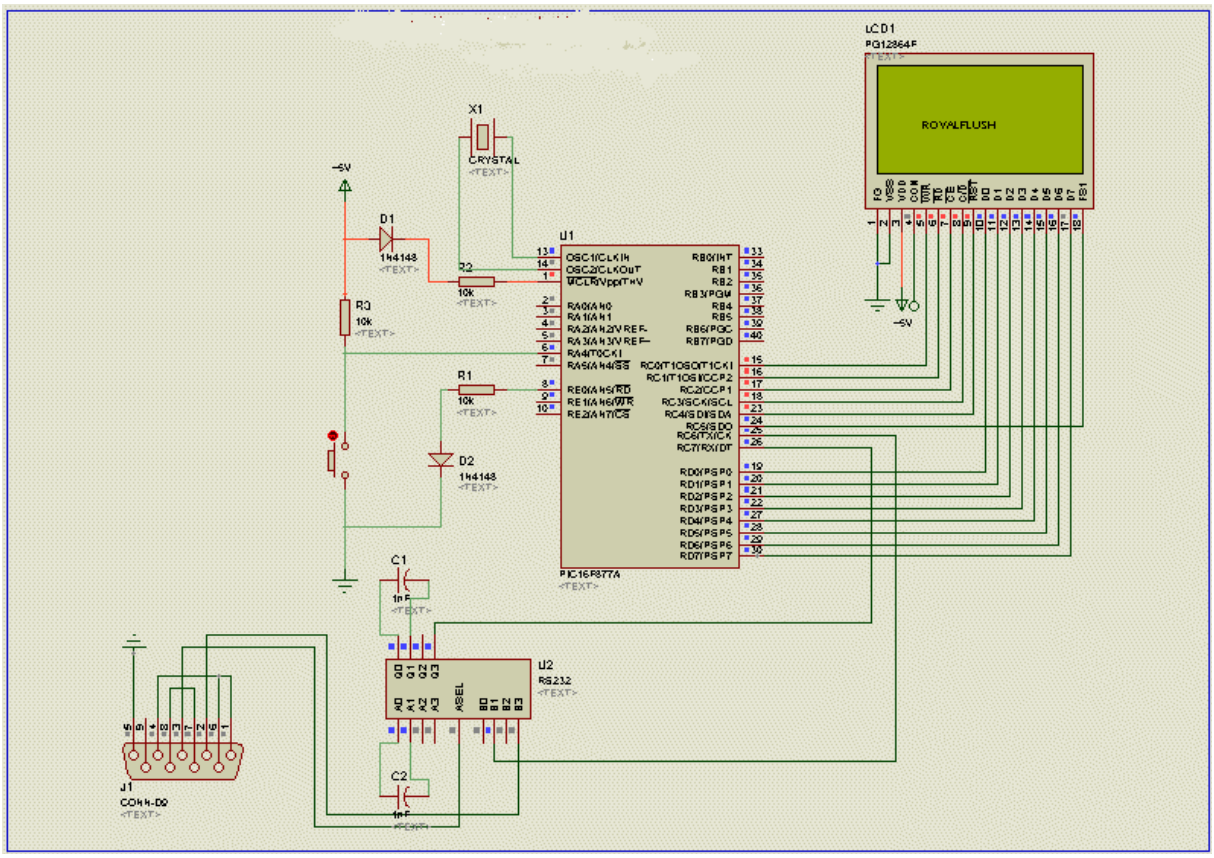


WIRING SCHEMATICS of THE RS-232 COMMUNICATIONS



CENG EMBEDDED BOARD

We completed giving information about PIC16F877 and ceng embedded board. Now we can monitor the statistical data which was gathered from AP-400 over RS232 in our lcd display. But we must write a protocol between PIC16F877 and lcd in order to show what we want . and we must also use the CCS PIC-C compiler. We give a small example of the connection of lcd and PIC16F877.



5. Project Requirements

5.1 Hardware Requirements

Indicated by the nature of the project, all group members are going to work on the sample AP-400 device provided for us by Air-Ties company. In conjunction with this, we will be in need of personal computers employing several distributions of Linux Operating System such as Debian, Ubuntu, and Pardus in order to comply with the development criteria of the company during the project.

The main objective to have different distributions of operating systems is to ensure the highest compatibility so that end-users feel comfortable with purchasing the product. Had we not planned to pay attention to such crucial details, even the end-product could not have been obtained, mainly due to the variations in software standard taken into consideration in the next section.

We will also require necessary equipment to plug the device (AP-400) to a desktop computer. An RS-232 serial cable will do if we have a serial port on the PC; otherwise, we will have to use a serial-to-USB converter to establish a connection between the working station (the device itself) and the development environment, which is the computer.

Depending on the development tool to be used during the software design process, developers may face memory requirements as high as 1 GB since tools with dense graphical user interfaces usually tend to use high amounts of memory during execution. On the contrary, no extra memory will be needed for AP-400.

Targeted users are expected to have an electronic device to which AP-400 and its counterpart, the LCD monitor, are connected. The end-product will be packaged in such a compact way that, users will not have to employ any extra effort to install the product.

5.2 Software Requirements

5.2.1 Software Requirements for RoyalFlush

- **Programming Language**

The C Programming Language will be used in software development tasks throughout the project. The device incorporating Linux Kernel version 2.4 is subject to a number of speed and space limitations that make developers consider trade-offs between the above mentioned phenomena. Our investigation indicates that the maximum size of software that can be run on the target device could not exceed several megabytes, and no complex system such as mathematical analysis tools or programs that employ graphical components is to be designed. Since no object-oriented approach is needed, procedural paradigm is the one to employ.

Yet another factor to be considered prior to making a decision must be the ease of debugging the code produced by a specific programming language. This approach brings about the analysis of compilers since the closer source code is to machine code (in terms

of semantic readability), the more debugging power can be used; and taking into account that compilers used for C are perhaps the most fruitful of all (GNU C Compiler and Bloodshed Dev C++ Compiler on 32-bit Win32/UNIX systems, and mips-linux-gcc, the cross-compiling version of the first one mentioned on embedded Linux systems) clears our mind about the programming language to choose: C.

- **Compilers**

Two of the most powerful C compilers extensively used today are GNU C Compiler[5] (aka. GCC) and Bloodshed Dev C++ Compiler. Currently in its second beta release, the latter uses the same engine (MinGW) as the former and is easily customizable for ranging needs of developers. Its main windows being user friendly, this compiler will be of much help for us during the software implementation phase of the project.

GNU C Compiler is an open-source, easy to use, platform independent, and a very efficient C Compiler that is strongly recommended for low level developers thanks to its facility to generate readable code that can easily be debugged using GNU Debugger, which will be explained in the next section.

The most-widely used PIC compiler, CCS C Compiler[6], will be helpful for us to convert source code into native code complying with PIC standards. Incorporating drivers for real-time clocks and LCD displays, it is likely to save us time letting us deal with more crucial aspects of the system instead of concentrating on time-consuming details resulting from the hardware representation of it.

This project requires us to decide on the compiler version to use among a number of choices ranging from gcc 3.4.6 to gcc 4.1.2. Our examination has so far indicated that gcc 3.x versions are more compatible with source codes written for Linux kernel version 2.6 (desktop computers) since we observed some linkage errors related to .h header files which incorporate all definitions of external functions in newer versions of gcc. Thus, we are most likely to work with gcc 3.4.6 and its predecessors when working on UNIX systems, and Bloodshed Dev C++ Compiler v4.9.9.2 Beta[7] when we switch to 32-bit Win32 systems. Embedded Linux counterpart of gcc, mips-linux-gcc, has stable versions as well. 3.3.3 version of this cross-compiler is already built in Linux kernel version 2.4,

and it will be extensively used when applications created on desktop computers are to be transported to AP-400.

- **Debuggers**

As briefly noted above, we are planning to work with GNU Debugger[8] to test, analyze, and optimize the source code we will be writing throughout the project. This choice is mainly due to the software's great abilities ranging from listing the symbol table for the program being debugged to set breakpoints wherever we would like to during execution of it. Other than these, we are also concerned with the amount of time to locate and correct the flaws/errors in the source code and GNU Debugger is one of the best software to help us locate the correct instruction being executed at a specific time.

- **Real-time Simulators**

Proteus Simulator[9] is a high-performance parallel-architecture simulator that will be of help for us especially when group members work on their own, without having to connect A[P-400 to their computer. This software supports LCD I/O; so, testing the programs we will be creating is not a concern. We believe it will be a facilitator for us during the project thanks to its high performance and real-time-aware facilities.

- **Terminal Operators**

The scope of the project seems to set some limits for developers in terms of complexity of the programs running on AP-400, so we are in a position to employ some additional tools that will help us a lot to clearly observe and analyze the flow of the entire structure (internal relationships between the hardware and software layers). Terminal operators are those software that enable users to execute particular commands that are not available in an alternative environment like a GUI (graphical user interface) component. To illustrate, we can give command-line arguments to a program in case such parameters affect the outcome of the process (output of the program in this scope). It is also noteworthy that we would not be able to do the same thing using a window-driven program with no such abilities.

TeraTerm[10] is a developer-friendly, interactive, and customizable software that is capable of fetching the Flash content of the device and listing it on a terminal window on 32-bit Win32 systems. Even though there are similar programs in the market, we decided to use this one because we will have to deal with command-line arguments and environment variables that are easily managed by the program of interest.

CuteCom[11] can be regarded as a UNIX counterpart of TeraTerm since they function exactly the same way as far as program deliverables are concerned. Advanced USB-port support puts this one step ahead of its rivals in the market. Since we will be working mainly on UNIX systems, this tool will be extensively used and taken help from.

- **GUI Development Tools**

Even though we estimate that we will mainly be working on console programs, some simple functions may be much easier to implement using a graphical user interface environment in terms of comprehensibility. For this reason, we agreed on an open-source GUI library that fits our needs in terms of simplicity and efficiency when we are working on 32-bit systems. The one we are going to use is called GTK+[12] and it employs the C programming language as its default language. GNOME binding of the library increases our hopes about compatibility with our main development environment, the Linux kernel v2.4.

5.2.2 Software Requirements for the End-User

Our current estimates indicate that AP-400 is going to have a compact software package, so end-users will not have to own specific software in order to launch the system. On the other hand, we will be providing some helpful tools to clearly explain the details of the entire system to targeted audience via a number of simulation techniques. Clearly, the users will feel comfortable with the ultimate package provided upon purchase of the product.

6. Risk Management

Risk management plan is necessary for the project because of the possibility of the future problems that may occur. It is always probable that there can be unexpected difficulties during the project. These risks can set borders in front of us to continue working on the project and our work may be wasted. In order not to fail the project, preparing a risk plan is very essential. We can figure out the following risks as our main concerns.

- **Problems due to lack of knowledge and/or experience**

  Since we are not experienced enough in developing such combinatorial systems depending mainly on hardware related issues, unexpected delays could occur. In order to work out this problem, we will continue step by step by searching everything that could lead to a problem. Moreover we are always going to try to keep in contact with our teaching assistant who possesses deep knowledge and experience about the context.

- **Problems resulting from misunderstandings / misconceptions**

  Possible disagreements between group members can affect our usual work flow, and also withdrawal of a group member does have a fatal impact on the project. Similarly, our group members may not be able to accomplish the duties from time to time because of time conflicts and the complexity of the assignment. In order to handle this problem, we always arrange meetings and communicate to better analyze the situation and adjust our schedule accordingly. Since relative difficulty of each assignment is different, sharing duties between team members is an efficient solution.

- **Problems arising from system incompatibilities (failures)**

  We may at times face an unexpected problem in the hardware of the system, and permanent data loss is perhaps the worst outcome of such a nuisance. In order to be prepared for such cases, we will always back up our works and be more careful prior to taking action especially when dealing with the internal structure of the system.

- **Post-production problems**

It may always be the case that customers are not satisfied with a product due to several reasons. In order not to face such difficulties, we are periodically meeting with the project

coordinator of the Air-Ties Company and trying to get all necessary information that can help us about the specifications we are to meet. Above all, our end-product should be of flaws and we will be trying our best to come up with a complete package at the end.

7. Test Plan

Our project, a wireless network traffic monitoring system, examines local area network usage on the AP-400 Wireless Access Point Repeater / Bridge / 4-Port Switch and shows these data such as upload and download statistics on a graphical LCD display by using PIC16F877 in our ceng embedded board. It will be a real-time embedded system which enables every new wireless connection to be registered by the users' MAC address and monitors on the LCD display. On this display, number of computers connected to the access point will be shown. The usage of the access point in past 24 hours will also be displayed. At the progressing stages of the project, we will connect the device to a computer via an RS232 serial port and using the embedded Linux system inside the device, we will conduct the necessary operations.

In each step of this project, we must test and see which stages we are done with because we must make sure that the system works perfectly. Otherwise, we would have to face fatal errors in upcoming steps of the project which would possibly result in irregularities requiring much more work and engineering-time to compensate for. In order not to have an error in the entire system, we prefer dividing the project into small parts each of which are taken care by a group member, and as soon as we have finished these parts an overall testing of the system is employed to see if everything goes well. If our system is perfect, we advance to the next step, but if it fails then we go back and try to locate where the problems arise and correct them as soon as possible to meet deadlines.

8. Conclusion

Our project, a wireless network traffic monitoring system, examines local area network usage on the AP-400 Wireless Access Point Repeater / Bridge / 4-Port Switch and shows these data such as upload and download statistics on a graphical LCD display by using PIC16F877 . And in this initial design report including the formal specification of our system

solution as stated in the Initial Design Report explanation of the ceng490 course. We chose our intended tools and built our guidelines and we made throughout research about our hardware and project requirements and chose our hardware which was a great experience in our project.

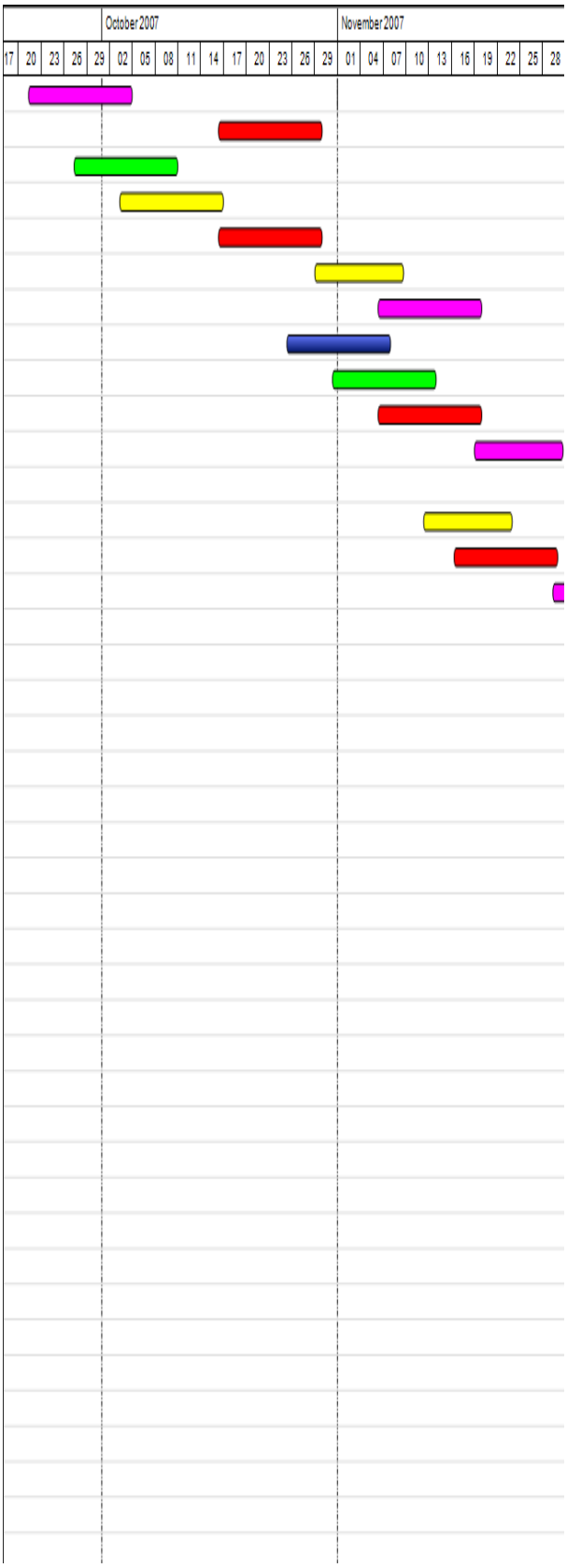
Our initial design report is prepared to establish a connection between our design and implementation. The information given here such as diagrams and other design products are produced in order to guide us through our way in the implementation of our project. Despite being an initial design, this document is a milestone that will help us make our prototype and real design report. We believe that this report will contribute to our project in a quite useful way.

9. Gannt Chart

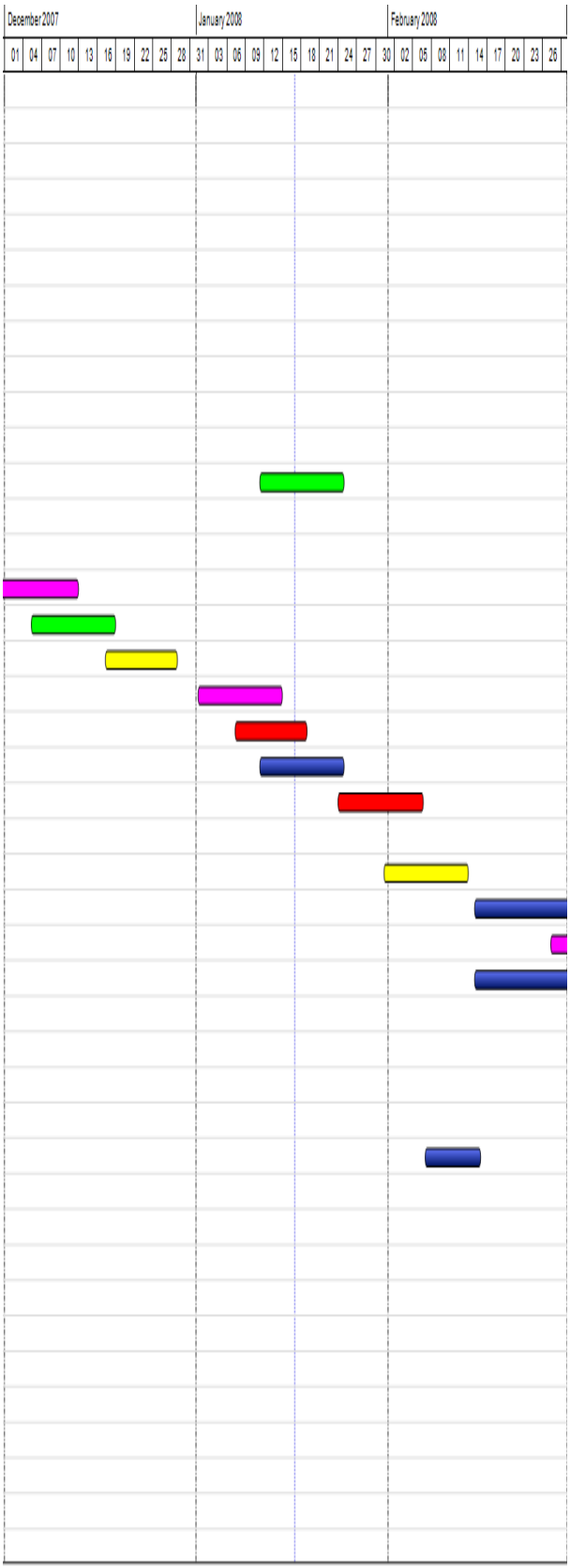
The gannt chart for our project is as follows. The filled rectangles and group members associated with each are summarized below.

Group Member	**Colors**
Hasan Tahsin KILIÇ	yellow, green
Yetkin SAKAL	blue, green
Hasari YAPICI	pink, green
Bahattin YALÇIN	red, green

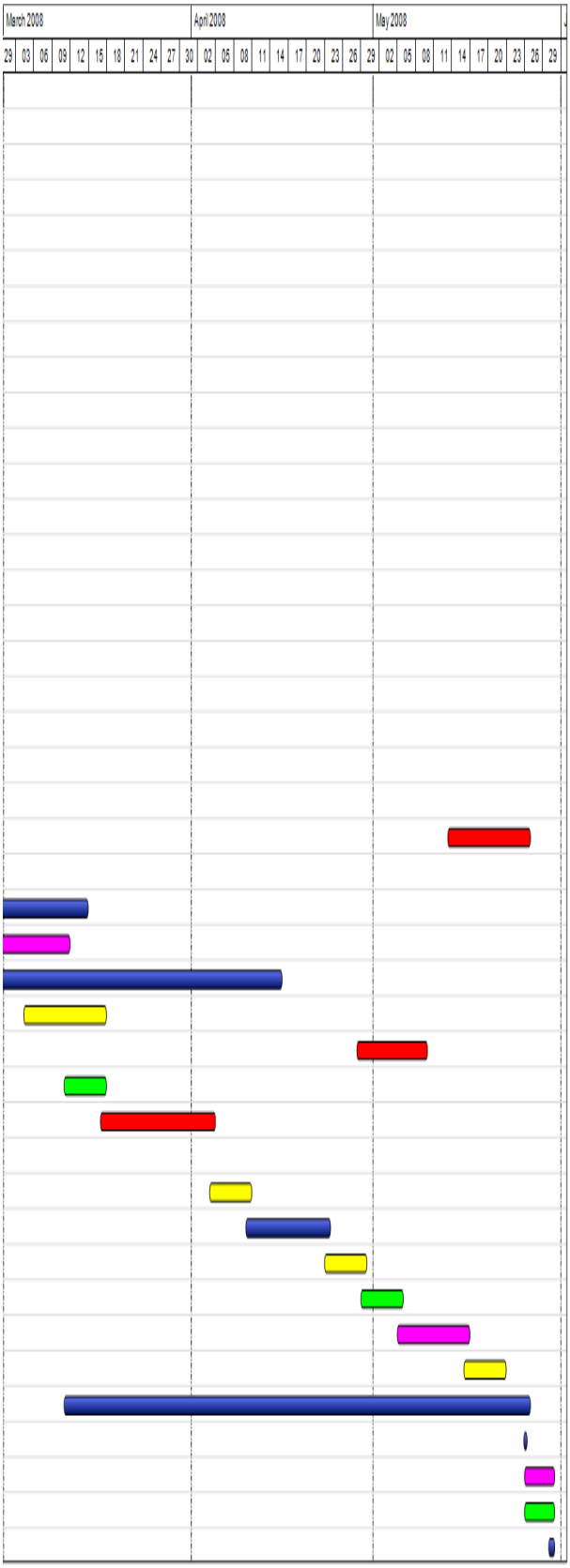
#	Name	Start	Finish
1	PLANNING	9/21/2007	10/4/2007
2	REQUIREMENT ANALYSIS	10/16/2007	10/29/2007
3	Researches on Specifications	9/27/2007	10/10/2007
4	Meeting and Analysis of inspenctions	10/3/2007	10/16/2007
5	Requirement Specifications	10/16/2007	10/29/2007
6	Milestone	10/29/2007	11/9/2007
7	INITIAL DESİGN	11/6/2007	11/19/2007
8	Working on drafts	10/25/2007	11/7/2007
9	Working on specifications	10/31/2007	11/13/2007
10	Initial design specifications	11/6/2007	11/19/2007
11	Milestone	11/19/2007	11/30/2007
12	DETAILED DESIGN	1/11/2008	1/24/2008
13	Detailed design specifications	11/12/2007	11/23/2007
14	Complete class diagram	11/16/2007	11/29/2007
15	Hardware design	11/29/2007	12/12/2007
16	Software design	12/5/2007	12/18/2007
17	User interface design	12/17/2007	12/28/2007
18	Library design	1/1/2008	1/14/2008
19	Tests	1/7/2008	1/18/2008
20	Design specifications	1/11/2008	1/24/2008
21	Milestone	1/24/2008	2/6/2008
22	IMPLEMENTATION	5/13/2008	5/26/2008
23	Working on pre implementations	1/31/2008	2/13/2008
24	Board design	2/15/2008	3/14/2008
25	Working on card	2/27/2008	3/11/2008
26	Serial port communication	2/15/2008	4/15/2008
27	Review	3/4/2008	3/17/2008
28	Implementation details	4/28/2008	5/9/2008
29	Software implementation on card	3/11/2008	3/17/2008
30	Library implementation	3/17/2008	4/4/2008
31	Working on script	2/7/2008	2/15/2008
32	Code implementation for smart card	4/4/2008	4/10/2008
33	Code implementation for ports	4/10/2008	4/23/2008
34	Code implementation for other parts of	4/23/2008	4/29/2008
35	Performance working	4/29/2008	5/5/2008
36	Tests and feedbacks	5/5/2008	5/16/2008
37	Final implementations	5/16/2008	5/22/2008
38	Documentation	3/11/2008	5/26/2008
39	Milestone	5/26/2008	5/26/2008
40	USER EVALUATİON	5/26/2008	5/30/2008
41	Feedback	5/26/2008	5/30/2008
42	Milestone	5/30/2008	5/30/2008



#	Name	Start	Finish
1	PLANNING	9/21/2007	10/4/2007
2	REQUIREMENT ANALYSIS	10/16/2007	10/29/2007
3	Researches on Specifications	9/27/2007	10/10/2007
4	Meeting and Analysis of inspenctions	10/3/2007	10/16/2007
5	Requirement Specifications	10/16/2007	10/29/2007
6	Milestone	10/29/2007	11/9/2007
7	INITIAL DESİGN	11/6/2007	11/19/2007
8	Working on drafts	10/25/2007	11/7/2007
9	Working on specifications	10/31/2007	11/13/2007
10	Initial design specifications	11/6/2007	11/19/2007
11	Milestone	11/19/2007	11/30/2007
12	DETAILED DESIGN	1/11/2008	1/24/2008
13	Detailed design specifications	11/12/2007	11/23/2007
14	Complete class diagram	11/16/2007	11/29/2007
15	Hardware design	11/29/2007	12/12/2007
16	Software design	12/5/2007	12/18/2007
17	User interface design	12/17/2007	12/28/2007
18	Library design	1/1/2008	1/14/2008
19	Tests	1/7/2008	1/18/2008
20	Design specifications	1/11/2008	1/24/2008
21	Milestone	1/24/2008	2/6/2008
22	IMPLEMENTATION	5/13/2008	5/26/2008
23	Working on pre implementations	1/31/2008	2/13/2008
24	Board design	2/15/2008	3/14/2008
25	Working on card	2/27/2008	3/11/2008
26	Serial port communication	2/15/2008	4/15/2008
27	Review	3/4/2008	3/17/2008
28	Implementation details	4/28/2008	5/9/2008
29	Software implementation on card	3/11/2008	3/17/2008
30	Library implementation	3/17/2008	4/4/2008
31	Working on script	2/7/2008	2/15/2008
32	Code implementation for smart card	4/4/2008	4/10/2008
33	Code implementation for ports	4/10/2008	4/23/2008
34	Code implementation for other parts of	4/23/2008	4/29/2008
35	Performance working	4/29/2008	5/5/2008
36	Tests and feedbacks	5/5/2008	5/16/2008
37	Final implementations	5/16/2008	5/22/2008
38	Documentation	3/11/2008	5/26/2008
39	Milestone	5/26/2008	5/26/2008
40	USER EVALUATİON	5/26/2008	5/30/2008
41	Feedback	5/26/2008	5/30/2008
42	Milestone	5/30/2008	5/30/2008



#	Name	Start	Finish
1	PLANNING	9/21/2007	10/4/2007
2	REQUIREMENT ANALYSIS	10/16/2007	10/29/2007
3	Researches on Specifications	9/27/2007	10/10/2007
4	Meeting and Analysis of inspenctions	10/3/2007	10/16/2007
5	Requirement Specifications	10/16/2007	10/29/2007
6	Milestone	10/29/2007	11/9/2007
7	INITIAL DESİGN	11/6/2007	11/19/2007
8	Working on drafts	10/25/2007	11/7/2007
9	Working on specifications	10/31/2007	11/13/2007
10	Initial design specifications	11/6/2007	11/19/2007
11	Milestone	11/19/2007	11/30/2007
12	DETAILED DESIGN	1/11/2008	1/24/2008
13	Detailed design specifications	11/12/2007	11/23/2007
14	Complete class diagram	11/16/2007	11/29/2007
15	Hardware design	11/29/2007	12/12/2007
16	Software design	12/5/2007	12/18/2007
17	User interface design	12/17/2007	12/28/2007
18	Library design	1/1/2008	1/14/2008
19	Tests	1/7/2008	1/18/2008
20	Design specifications	1/11/2008	1/24/2008
21	Milestone	1/24/2008	2/6/2008
22	IMPLEMENTATION	5/13/2008	5/26/2008
23	Working on pre implementations	1/31/2008	2/13/2008
24	Board design	2/15/2008	3/14/2008
25	Working on card	2/27/2008	3/11/2008
26	Serial port communication	2/15/2008	4/15/2008
27	Review	3/4/2008	3/17/2008
28	Implementation details	4/28/2008	5/9/2008
29	Software implementation on card	3/11/2008	3/17/2008
30	Library implementation	3/17/2008	4/4/2008
31	Working on script	2/7/2008	2/15/2008
32	Code implementation for smart card	4/4/2008	4/10/2008
33	Code implementation for ports	4/10/2008	4/23/2008
34	Code implementation for other parts of	4/23/2008	4/29/2008
35	Performance working	4/29/2008	5/5/2008
36	Tests and feedbacks	5/5/2008	5/16/2008
37	Final implementations	5/16/2008	5/22/2008
38	Documentation	3/11/2008	5/26/2008
39	Milestone	5/26/2008	5/26/2008
40	USER EVALUATİON	5/26/2008	5/30/2008
41	Feedback	5/26/2008	5/30/2008
42	Milestone	5/30/2008	5/30/2008



10. References

[1] AP-400 Wireless Access Point Repeater / Bridge / 4-Port Switch

http://www.airties.com/index.asp?page=prdetails&id=23&dil=eng

[2] CuteCom

http://cutecom.sourceforge.net/

[3] TFTP

http://en.wikipedia.org/wiki/Tftp/

[4] MadWifi

http://madwifi.org/

[5] JFFS

http://en.wikipedia.org/wiki/JFFS/

[6] Wear-leveling

http://en.wikipedia.org/wiki/Wear_levelling/

[7] JFFS2

http://en.wikipedia.org/wiki/JFFS2/

[8] Cross Compiler

http://en.wikipedia.org/wiki/Cross_compiler/

[9] GNU C Compiler

http://gcc.gnu.org/

[10] CCS C Compiler

http://www.ccsinfo.com/content.php?page=compilers/

[11] Bloodshed DevC++ Compiler IDE

http://www.bloodshed.net/devcpp.html/

[12] GNU Debugger

http://www.gnu.org/software/gdb/

[13] Proteus Simulator

http://www.labcenter.co.uk/

[14] TeraTerm

http://www.ayera.com/teraterm/

[15] GTK+

http://en.wikipedia.org/wiki/Gtk%2B/