MIDDLE EAST TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

CENG 491

DETAILED DESIGN REPORT

SERKAN ÇAĞLAR	**1347285**
BURAK CANSIZOĞLU	**1347244**
SERDAR KOÇBEY	**1250471**
HANİFİ ÖZTÜRK	**1298140**

1 INTRODUCTION

1.1 Purpose of the Document

The purpose of this document is to express the detailed design specifications of our project, HSBS_WSN. With the help of resolving software, hardware, functional and non-functional requirements, we have prepared this report. This report will be a guideline for our future studies. In the design process, we intended to design an effective and modular product that will satisfy the needs and constraints of the project. In this document, we tried to explain our design process in an illustrative way with the help of diagrams. These diagrams are specific types of UML diagrams like class, sequence and structural diagrams. Owing to these diagrams, we have been able to explain the functional, structural and behavioral features of our system.

1.2 Detailed Problem Definition

As the technology evolves, usage of wireless networks has increased remarkably. In parallel to this, application area of embedded systems integrated with wireless networks has expanded. As a result of this development, wireless sensor networks emerged in the last decade.

A wireless sensor network (WSN) is a wireless network consisting of spatially distributed autonomous devices using sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants, at different locations.[1]

Wireless sensors are far more efficient and feasible than their wired counterparts with respect to their easiness of use, wider range and application areas and less deployment costs. In our project, we aim to establish a wireless sensor network (HSBS_WSN) that will support IEEE

[1]Wikipedia, Wireless Sensor Network, http://en.wikipedia.org/wiki/Wsn, October 2007

802.11 protocol. Most of the wireless sensors in the market support some protocols like ZigBee and IEEE 802.15; however, unfortunately there are too few sensors that support IEEE 802.11 and these sensors are unaffordable for us. Because of this, we have decided to build our own wireless sensor node (HSBS Sentinel). HSBS Sentinels will have two main advantages. Firstly, they will communicate directly with PCs that support IEEE 802.11. Secondly, they will be more affordable than the other wireless sensors that are available in the market. An HSBS Sentinel is formed of an Airties AP-400, a SP07 (a sensor integrated PIC board) and other essential hardware parts. In the project, there will be a number of HSBS Sentinels; some of them will be used in HSBS Sentinels, some of them will be repeaters and one of them will be the access point. Moreover, there will be a server that will collect data from HSBS Sentinels over the access point and store the data in the database. Furthermore, a user interface that will be used to monitor and process the data will be implemented on this server. HSBS WSNs will be able to be used in many applications in which temperature and humidity are measured.

1.3 Design Constraints

1.3.1 Financial Constraints

In the market, there are a limited number of wireless sensors that support IEEE 802.11 protocol. However, these sensors are not affordable for us. Therefore, we have to design our own wireless sensor nodes, namely HSBS Sentinels.

1.3.2 Manufacturing Constraints

Since we have to build our own wireless node, we are obtaining some hardware equipments and assembling them. However, sometimes we have some problems while combining these different parts and we cannot foresee the difficulties that we may confront. Furthermore, testing the hardware units that we implement takes much time.

1.3.3 Ergonomic Constraints

Since a SP07 and an AirTies AP-400 will be used in HSBS Sentinels, our wireless nodes will be larger and bulkier than other wireless sensors in the market. This makes HSBS Sentinels be less ergonomic.

1.3.4 Power Constraints

Because we are using a SP07 and an AirTies AP-400 in HSBS Sentinels, we are forced to use a power adapter for SP07 and AP-400. For this reason, HSBS Sentinels will need a power socket.

1.3.5 Resource Constraints

Because wireless sensor network area is a new and vast area, this subject is in fact currently being researched by universities and institutions world wide. For this reason, we have difficulties in finding enough resources.

1.3.6 Lack of Experience of Team Members

Since the group members have taken a few hardware courses and have little experience on hardware, sometimes it is difficult for us to visualize the details of the project.

1.3.7 Time Constraints

The schedule of the project is determined by the CENG 491 course syllabus. From now on, we have about four months to finalize the project successfully.

1.4 Design Objectives and Goals

1.4.1 Power

As we stated in the design constraints section, HSBS Sentinels will consist of a SP07 and an AirTies AP-400. In our project, we plan to use only one power supply per an HSBS Sentinel. Only the power adapter of the AP-400 will be used to supply energy to an HSBS Sentinel. The power to the SP07 will be provided from AP-400 with the help of a voltage level converter.

1.4.2 Lifetime

HSBS_WSN should operate properly for a long time. Since, the sensors (SHT15) have CMOSens Technology, they have long-term stability. By the help of this feature of the sensors, the overall product will be able to run for a long time.

1.4.3 Security

Security issue is a big problem for WSNs, however it has been overcome by WPA (802.1x, TKIP, PSK), WPA2 (IEEE802.11i, AES, CCMP), WEP (64/128 bit), MAC filtering and SSID hiding properties of AirTies AP-400.

1.4.4 Accuracy

The HSBS WSN will be able to determine accurate temperature and humidity values by the capability of high-precision measurement of the SHT15s.

1.4.5 Size

In HSBS Sentinels, we could have used a CENG 336 Embedded Board on which a SHT15 is assembled. However, then an HSBS Sentinel would be bulky. For this reason, we have implemented the SP07 to prevent this situation.

1.4.6 Cost

Existing similar products to HSBS Sentinel are expensive to be attained. An HSBS Sentinel costs approximately $150 and this price is about one fourth of the price of the cheapest wireless sensor in the market. Thus, the overall project will be affordable than the existing ones.

1.4.7 Wide Range

The feature of AP-400 that allows it to run as a repeater provides us to implement a mesh network. With mesh networking, an HSBS Sentinel that is not in the range of the access point will send its data over repeaters. Thanks to this property of AP-400, HSBS_WSN will operate on a wide range.

2 ARCHITECTURAL DESIGN

2.1 System Hardware Modules

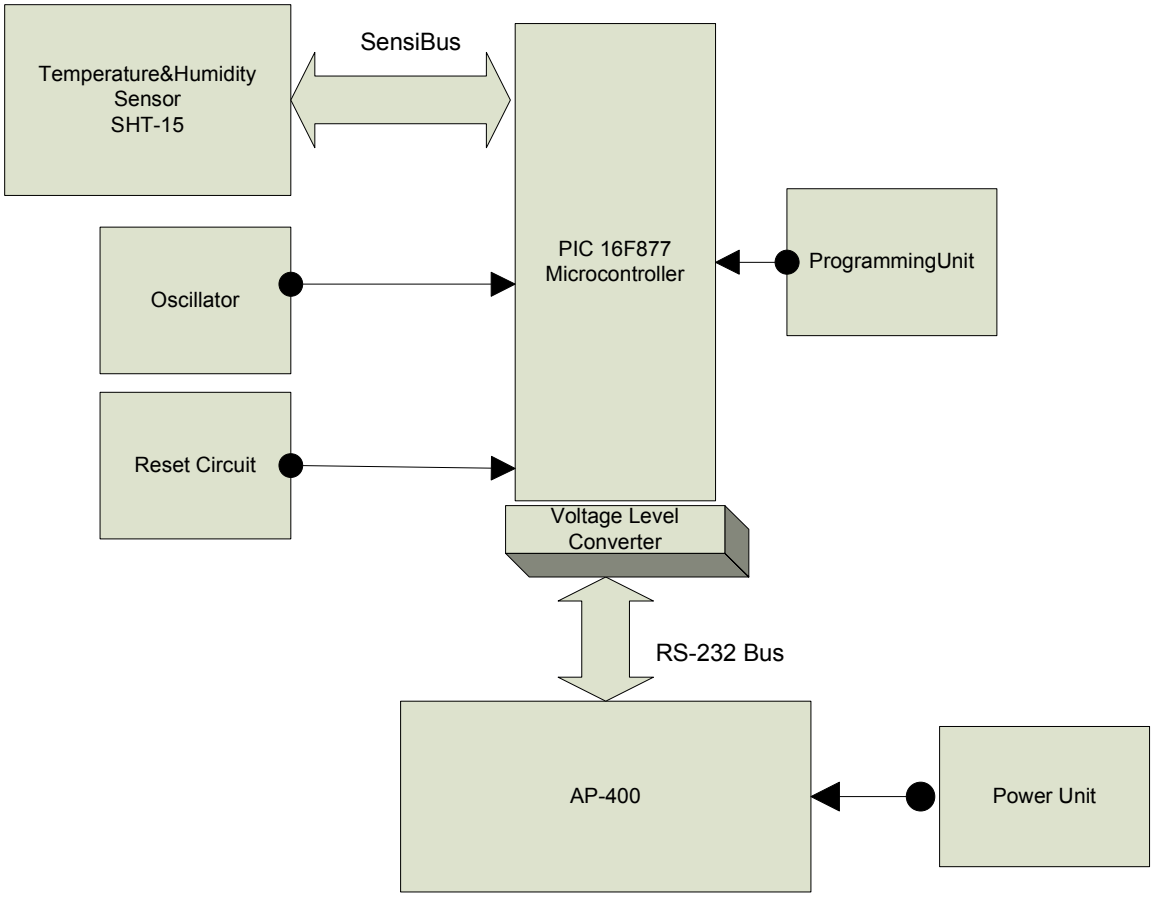


Figure 2-1 Hardware Block Diagram

In Figure 2-1, hardware block diagram is shown. As seen in the figure, the data transfer between SHT15 and PIC 16F877 are provided by SensiBus. PIC 16F877 is programmed by the Programming Unit whenever needed. Moreover, an oscillator and a reset circuit are essential parts for the proper functioning of PIC 16F877 and are connected to it. The data transfer between PIC 16F877 and AP-400 is provided through RS-232 Serial Bus. Furthermore, a power unit is connected to AP-400 and this power unit will supply the power needed SP07 by converting it to 5V via voltage regulator 7805. AP-400 operates via 3.3 V and PIC 16F877 operates via 5V. In order to avoid these power differences in RS232 port, voltage level converter MAX3378 is used between AP-400 and PIC 16F877.

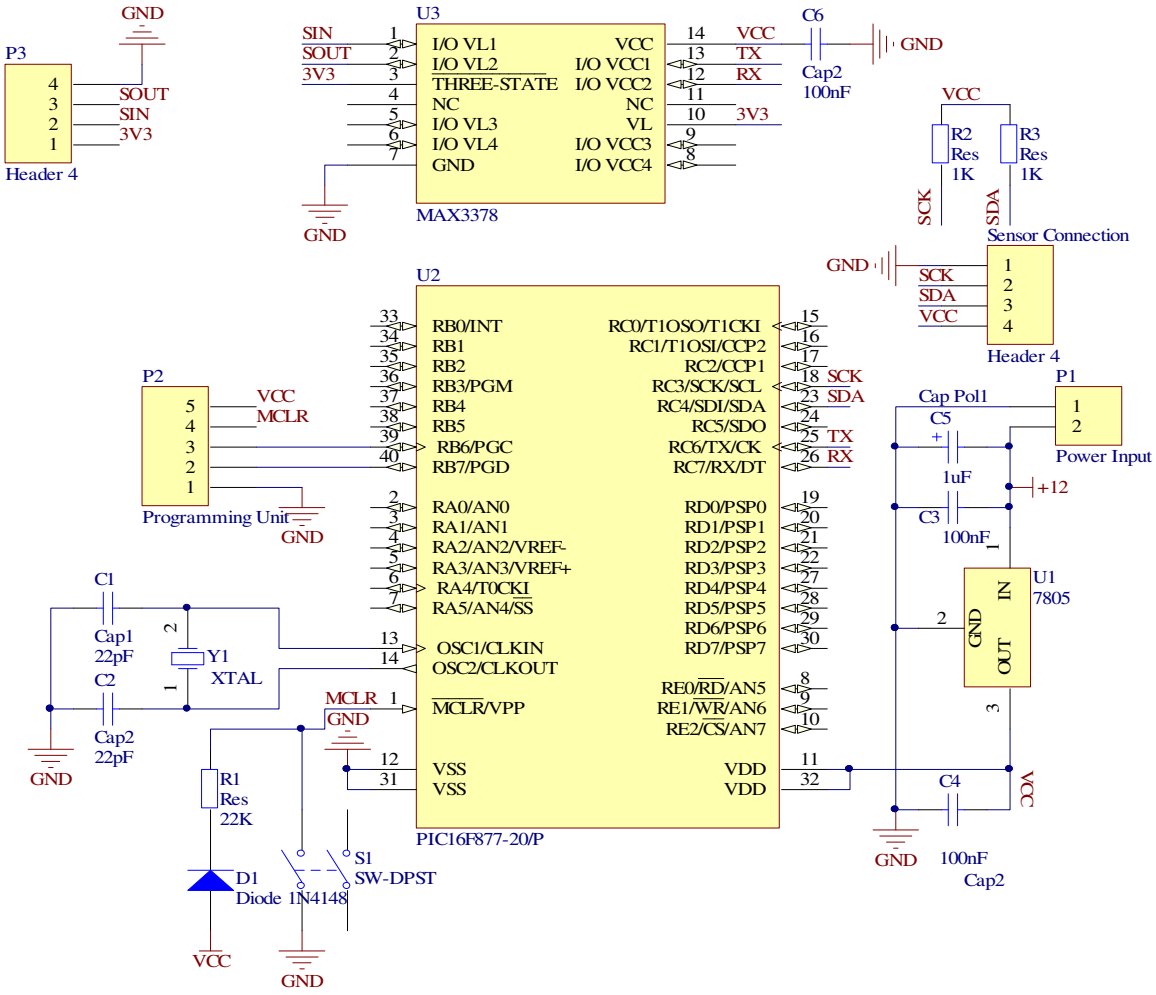


Figure 2-2 General Schematic of SP07 (renewed)

In Figure 2-2, general schematic of SP07 is shown. SP07 is the name for the unit that includes SHT15 and PIC 16F877 for convenience. This schematic shows the connections of the pins in SHT15, Oscillator, Programming Unit, SensiBus, RS-232 Bus and PIC 16F877. We renewed the schematic of SP07 board and eliminated the errors in previous schematic. Designing of schematic was implemented with Altium Designer. This tool also allows us to design printed circuit board (PCB) with this schematic.

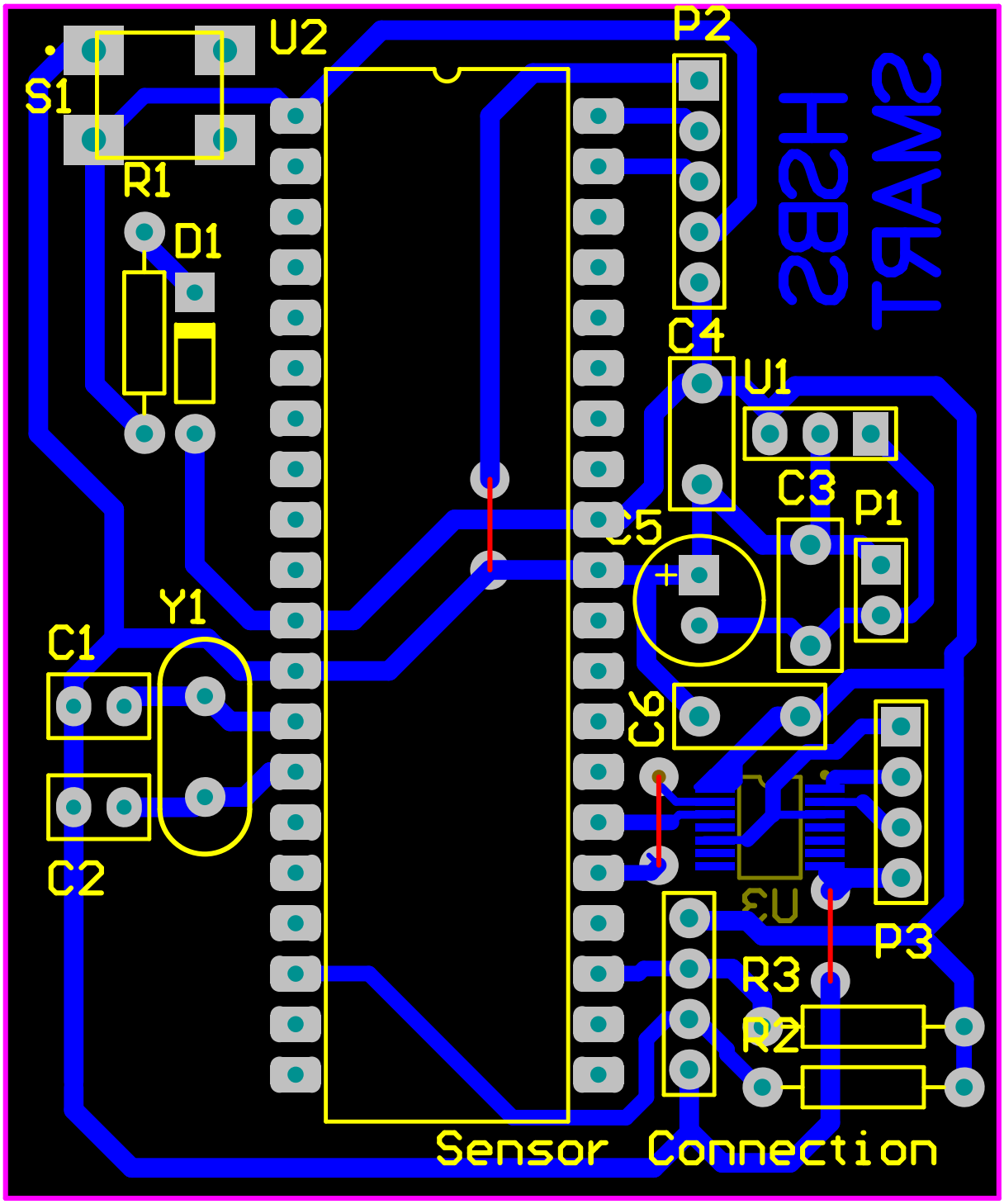


Figure 2-3 Printed Circuit Board (PCB) Design (v1.0)

This PCB is implemented by us manually with the help of our advisor assistant. Process is described as follows:

After designing of PCB we print out the PCB print layout in 1-1 scale onto toner transfer paper with the laser printer. Then we cut the circuit board into dimensions of our PCB design,

we clean the copper plate of circuit board with cleaning powder. We stick the border of printed paper onto cleaned copper plate and we pressed on it with heated iron. When slowly ironing the paper, toners were appeared on copper plate after few minutes. If all wire roads clearly appeared on plate then it can be placed into cool water container. Then we remove the special paper on copper plate. Now it is ready to place into chemical solution for removing unwanted copper plates on circuit board. This solution consists of two chemicals. First one is Hydrochloric Acid (HCL, %38 concentration) with 1.5 measures and the second one is Hydrogen Peroxide (H_2O_2) with 3 measures. After placing circuit board into the solution it was waited about 1 minute. When the circuit wires become clear on board then we placed into water container again for stopping the reaction on board with solution. Then we dried and cleaned the board for drilling pin holes on the board. After drilling operation we placed and soldered the circuit equipments and tested one by one for proper functioning when its work.

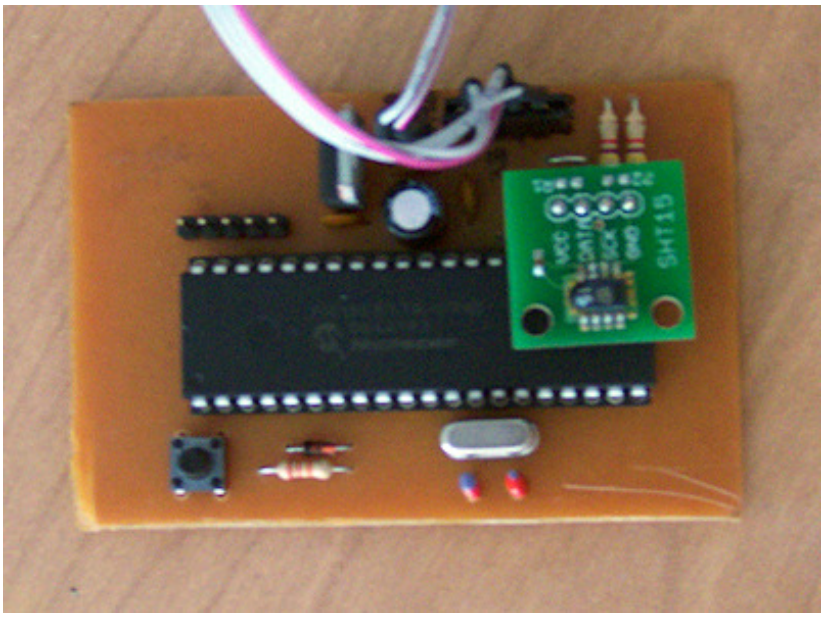


Figure 2-4 SP07 v1.0

2.2 System Software Modules

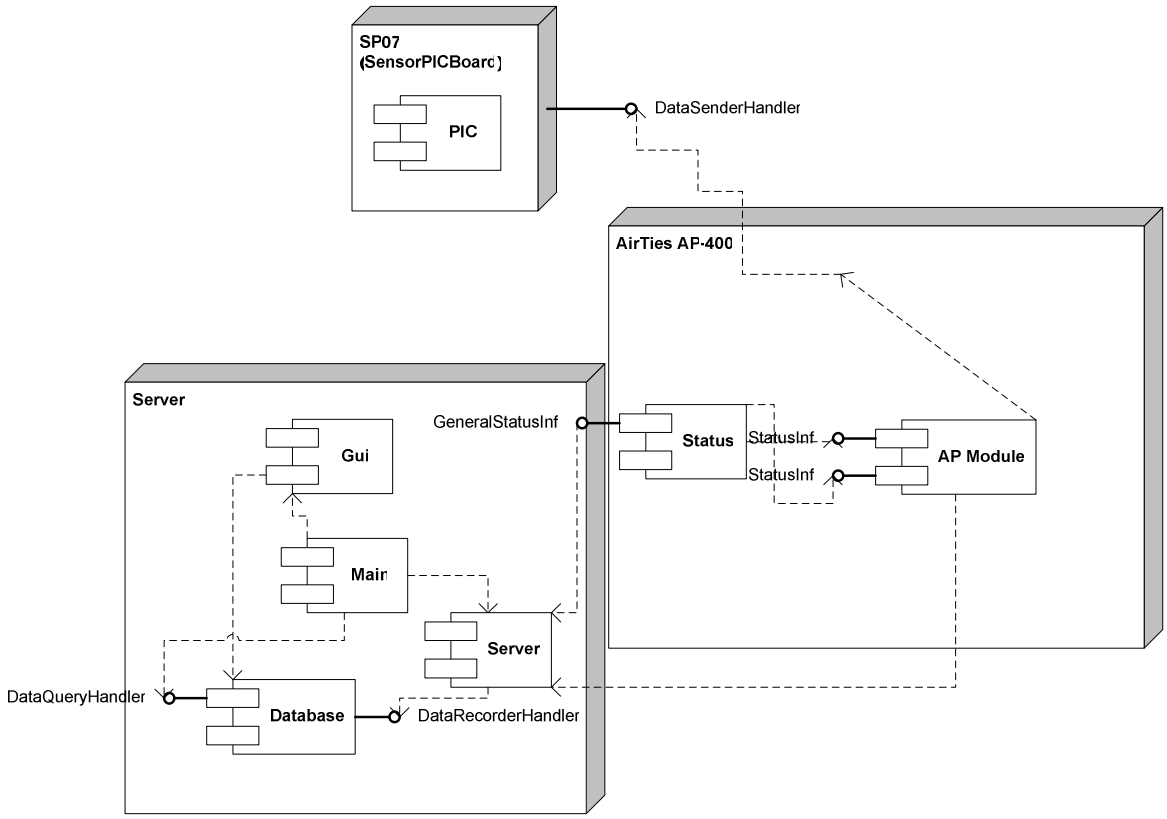


Figure 2-5 Software Component Diagram

The system is composed of three main physical structures; first one is SP07, second AP-400 and third the server. From this point of view, the modules are defined under these three structures hierarchically. The PIC Module which is included in SP07 is responsible for manipulating the sensor read data within the SP07 and provide an interface to AP-400 for data sending. AP Module is placed in AP-400 and requires an interface from SP07 to retrieve data and process this data internally in AP-400 and operates in order to communicate with Server Module and send the data provided. Status Module which is placed in AP-400 also provides an interface for querying, restarting and turning off AP-400. Server Module provides an interface for Main Module for sending requests to nodes and directs the data to Database Module and node messages to Main Module. GUI Module operates on Server side and is responsible for handling the user interactions like receiving user commands, displaying system outputs, recording user settings. Main Module is placed on server and processes the user commands. It requires an interface from Server Module to send requests and GUI Module to return the outputs. Database Module handles the queries generated by sub processes from Main Module and Server Module.

2.2.1 PIC Module

PIC Module reads the sensor data via Data Bus according to its producer protocol. This protocol includes the initialization of sensor and data bus for proper measuring of values.

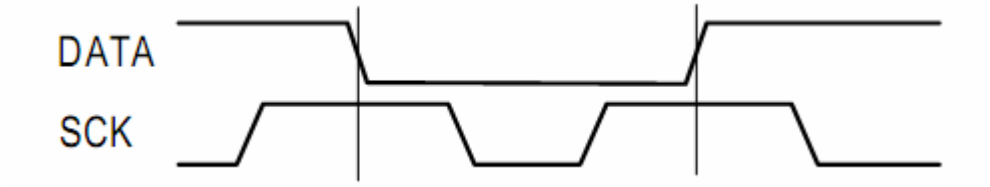


Figure 2-6 Communication Start Sequence [2]

Measured data is calculated and stored in one memory unit in the banks. If interrupt is detected on the RS232 port the PIC Module goes into `send()` routine and sends the stored data to the RS232 stream. The routines for stated functions are as follows:

```
void communicationStart()
// generates a transmission start
// in figure 2-6 we shown the initial configuration of Data and CLK
pins for PIC
   1. set CLK pin high
   2. set DATA pin low
   3. reevaluate step 1&2 with complements of previous pin value
until two cycle


int sendCommandToSHT15(int8 iobyte)
//sending commands to sensor
   1. sends 8 bit
   2. wait ack
   3. return ack

int16 readBytesOnSensiBus ()
//reads a byte (data) from the SensiBus
   1. shift most significant bits
   2. send ack 0 bit
   3. shift least significant bits
   4. send ack 1 bit
   5. return int16 //read data

void waitSHT15Reading ()
//waits for internal functioning of SHT15 - data reading
   1. set DATA pin high
   2. set CLK pin low
   3. wait >1 ms
   4. if DATA pin is low then SHT15 is ready
```

---

[2] Humidity & Temperature SHT15 Sensor Datasheet (p.3)

5. wait 100ms

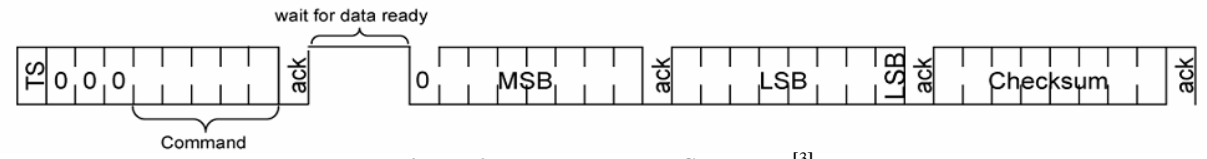


Figure 2-7 Measurement Sequence [3]

```
int16 measuretemp ()
//in figure 2-7 measurement sequence is showed
//measure SHT15 temperature
// #define MEASURE_TEMP 0x03
   1. call communicationStart()
   2. call sendCommandToSHT15(MEASURE_TEMP)
   3. read return ack
   4. call waitSHT15Reading()
   5. call readBytesOnSensiBus()
   6. read return iobyte
   7. return iobyte

int16 measurehum ()
//measure SHT15 humidity
// #define MEASURE_HUM 0x05
   1. call communicationStart()
   2. call sendCommandToSHT15(MEASURE_HUM)
   3. read return ack
   4. call waitSHT15Reading()
   5. call readBytesOnSensiBus()
   6. read return iobyte
   7. return iobyte


void calculate_data (int16 temp, int16 humid)
//required equations and parameters are showed in sensor datasheet
   1. calculate the data according to sensor producers equation

void sht15_initialization ()
//first initialize of sht15 sensor
   1. communicationReset()
   2. delay 20 ms //for powering sensor

void communicationReset ()
//resets the communication and reestablished it
   1. set DATA pin high
   2. set CLK pin low
   3. toggle CLK pin for 9 times
   4.call communicationStart()

void SHT15ReadCalculate()
   1. call measuretemp()
   2. read and store returned iobyte
```

[3] Humidity & Temperature SHT15 Sensor Datasheet (p.4)

```
   3. call measurehum()
   4. read and store returned iobyte
   5. call calculate_data( temp, humid) with stored_parameters

void main(){
   1. call sht15_initialization()
   2. in infinite loop make procedure
   3. call SHT15ReadCalculate()
   4. delay 500 ms //for prevent to self heating of sensor
   5. if interrupts come send measured temp and humid value to rs232
port
   6.return success
```

2.2.2 AP Module

The AP Module opens the serial device 'ttyS0' and polls the PIC Module with a single character string as "T" for temperature and "H" for humidity. Then it waits for the serial input as strings from the PIC Module which is described as "T *val*" and "H *val*" for temperature and humidity accordingly. The first character of the string defines the type of the following floating point data in the "*val*". AP Module parses the string and checks whether there is any threshold violation where the threshold values are defined in "thresholds.txt". If a violation is detected an alert message is sent to Server Module over a TCP socket.

On the other hand AP Module opens a TCP socket for handling Server Module connections, and if a connection exists, AP Module sends the "T *val*", "H *val*" strings over TCP socket. The pseudo code routines of these processes are described as follows:

```
string readSensor(string s)
   1. write(s) to serial port,
   2. read() from serial port to string result,
   3. Return result.


int check(string s)
   1. if s[0]='T' then do
   2.    if not MAX_TEMP > atof(s+2) > MIN_TEMP then do
   3.       Return 1.
   4. else if s[0]='H' then do
   5.    if not MAX_HUMD > atof(s+2) > MIN_HUMD then do
   6.       Return -1.
   7. else do
   8.    Return 0.


void initServerSocket()
   1. Allocate a TCP socket sockserv,
   2. Setup socket structure sockserv for port 2000,
   3. Bind it with the OS,
   4. Start listening,
   5. Return.

Void initClientSocket()
   1. Allocate a TCP socket sockcli,
   2. Setup socket structure sockcli for port 2001,
   3. Connect to sockcli,
```

```
   4. Return.


void initTty(string device)
   1. open device with read/write option,
   2. Setup the terminal I/O structure tio,
   3. Set the control, input, output flags of tio according to
BAUDRATE, LOCAL, NO_PARITY,
   4. Enable canonical inputting,
   5. Clear the device line,
   6. Activate settings for serial port,
   7. Return.

void apServer()
   1. Wait for inbound TCP connection inbound_connection,
   2. While there is inbound_connection and STATUS is ON do
   3.    read() from serial device to string s,
   4.    if (check(s) = -1) then
   5.       sendAlert(HUMD, s).
   6.    else if (check(s) = 1) then
   7.       sendAlert(TEMP, s).
   8.    end.
   9.    write(s) to inbound_connection,
  10. end.
  11. Return.

void sendAlert(int type, string s)
   1. initClientSocket(),
   2. write(type + s) to socket,
   3. close(sockcli),
   4. Return.

Void getThresholds(file f)
   1. open(f) for reading,
   2. read() from f,
   3. Set MAX_TEMP, MIN_TEMP, MAX_HUMD, MIN_HUMD,
   4. close(f),
   5. Return.

void getStatus(file f)
   1. open(f) for reading,
   2. read() from f,
   3. set STATUS,
   4. close(f)
   5. Return.

void statusChange(int sig)
   1. if sig is TURNOFF then
   2.    set STATUS to STANDBY,
   3. else if sig is TURNON then,
   4.    set STATUS to ON,
   5. else if sig is NEWTHRESHOLDS then,
   6.    getStatus("status.txt"),
   7. else do nothing,
   8. Return.

int main()
   1. initTty(TTYS0),
   2. initServerSocket(),
   3. assign signals sig TURNOFF, TURNON, NEWTHRESHOLDS, to
statusChange(sig),
   4. getThresholds("thresholds.txt"),
   5. getStatus("status.txt"),
   6. apServer(),
   7. close(sockserv),
   8. Exit.
```

2.2.3 Status Module

The Status Module provides an interface for Server Module for actions like sending 'reboot', 'turn on', 'turn off' requests, setting node thresholds and retrieving node status information. It opens a TCP socket and listens for the requests from Server Module. Each request consists of a string in the form "*RequestID: [reboot | turnoff | turnon | status | setThreshold(MAX_TEMP, MIN_TEMP, MAX_HUMD, MIN_HUMD)]*". Status Module parses the request message, and takes the according action. The pseudo code routines of these processes are as follows:

```
Void initClientSocket()
   1. Allocate a TCP socket sockcli,
   2. Setup socket structure sockcli for port 2003,
   3. Connect to sockcli,
   4. Return.

void initServerSocket()
   1. Allocate a TCP socket sockserv,
   2. Setup socket structure sockserv for port 2002,
   3. Bind it with the OS,
   4. Start listening,
   5. Return.

void rebootAP()
   1. Wait for 5000ms.
   2. Call system call reboot.

void turn_off()
   1. Send TURNOFF signal to process AP Module,
   2. saveState(STANDBY),
   3. Return.

void turn_on()
   1. Send TURNON signal to process AP Module,
   2. saveState(ON),
   3. Return.

void send_info()
   1. initClientSocket(),
   2. write("state: STATE thresholds: MAX_TEMP, MIN_TEMP, MAX_HUMD,
MIN_HUMD, uptime: UPTIME") to sockcli,
   3. close(sockcli),
   4. Return.

void setThresholds(file f)
   1. open(f) for writing,
   2. write ("MAX_TEMP MIN_TEMP MAX_HUMD MIN_HUMD") to f,
   3. close(f),
   4. Send NEWTHRESHOLDS signal to process AP Module,
   5. Return.

void parseRequest(string s)
   1. set requestID to RequestID,
   2. get request from s,
```

```
   3. if request = REBOOT then
   4.    rebootAP().
   5. else if request = TURNOFF then
   6.    turn_off(),
   7.    Return.
   8. else if request = TURNON then
   9.    turn_on(),
  10.    Return.
  11. else if request = STATUS then
  12.    send_info(),
  13.    Return.
  14. else if request = setThreshold then
  15.    extract MAX_TEMP, MIN_TEMP, MAX_HUMD, MIN_HUMD,
  16.    setThresholds("threshold.txt"),
  17.    Return.
  18. else do nothing.
  19. Return.

Void statusServer()
   1. Wait for inbound TCP connection inbound_connection,
   2. While there is inbound connection do
   3.    read() from inbound_connection to string s,
   4.    parseRequest(s),
   5. end.
   6. Return.

int main()
   1. initServerSocket(),
   2. statusServer(),
   3. close(sockserv),
   4. Exit.
```

2.2.4 Server Module

What Server Module simply does is, establish a connection to database server, setup a TCP socket for collecting the data from nodes in the active nodes list. On the other hand Server Module opens another TCP socket for receiving alert messages, and sending request strings of the type "*RequestID: [reboot | turnoff | turnon | status | setThreshold(MAX_TEMP, MIN_TEMP, MAX_HUMD, MIN_HUMD)]"* and request results. The pseudo code routines of these processes are as follows:

```
void initRequestClientSocket(string ip)
   1. Allocate a TCP socket sockcli,
   2. Setup socket structure sockcli for port 2002,ip,
   3. Connect to sockcli,
   4. Return.

void initRequestServerSocket()
   1. Allocate a TCP socket sockserv,
   2. Setup socket structure sockserv for port 2001,2003
   3. Bind it with the OS,
   4. Start listening,
   5. Return.

void initCollectDataClientSocket(string ip)
```

```
   1. Allocate a TCP socket sockcli,
   2. Setup socket structure sockcli for port 2000,ip,
   3. Connect to sockcli,
   4. Return.


void RequestServer()
   1. Wait for inbound TCP connection inbound_connection,
   2. While there is inbound_connection do
   3.    read() from serial device to string s,
   4.    if s:type is ALERT then
   5.       send signal ALERT to Main Module,
   6.    else if s:type is STATUS then
   7.       send signal STATUS to Main Module,
   8. end.
   9. Return.

void collector()
   1. for each node in the active node list do
   2.    initCollectDataClientSocket(node.ip),
   3.    receive() from sockcli to float temp, humd,
   4.    mysql.insertIntoTable( temp, humd),
   5. end.
   6. Go to step 1.

void sendRequest(sig)
   1. string request buildRequest (sig),
   2. initRequestClientSocket(sig:ip),
   3. write(request) to sockcli,
   4. close(sockcli),
   5. Return.

void main()
   1. mysql.connectDB(HOSTNAME, DB),
   2. initRequestServerSocket(),
   3. assign signals sendRequest(sig) from Main Module,
   4. RequestServer(),
   5. collector(),
   6. Exit.
```

2.2.5 Database Module

The Database Module provides interfaces for Main Module and Server Module for querying and updating the database. Methods as `connectDatabase()`, `createTable()`, `insertIntoTable()`, `dropTable()`, `updateTable()` are offered to system. The database module of the system will be taken from an existing API; therefore the methods are not going to be described in detail here. Class diagrams can be seen for further information.

2.2.6 Main Module

The Main Module requires interfaces from Server Module, Database Module and GUI Module for actions like changing node status, fetching data from database, collecting messages and regulating the data for outputting. The base class of the Module is `Node` class. The methods `getStatus()`, `changeNodeState()`, `rebootNode()`, `acknowledge()` are

offered as interface. The routines of these methods are as follows;

```
void getStatus(){
   1. If status = UNREACHABLE then,
   2.    Return.
   3. Else,
   4.    Generate requestID
   5.    Call server.sendRequest(nodeID, STATUS, requestID)
   6. Return.
}

void changeNodeState(){
   3. If status = ON then,
   4.    Generate requestID
   5.    Call server.sendRequest( nodeID, TURNOFF, requestID)
   6. Else if status = STANDBY then,
   7.    Generate requestID
   8.    Call server.sendRequest( nodeID, TURNON, requestID)
   9. Return.
}

void rebootNode(){
   1. If status is UNREACHABLE then,
   2.    Return.
   3. Else,
   4.    Generate requestID
   5.    Call server.sendRequest(nodeID, REBOOT, requestID)
   6. Return.
}
```

3 CLASS DIAGRAMS

3.1 Diagram

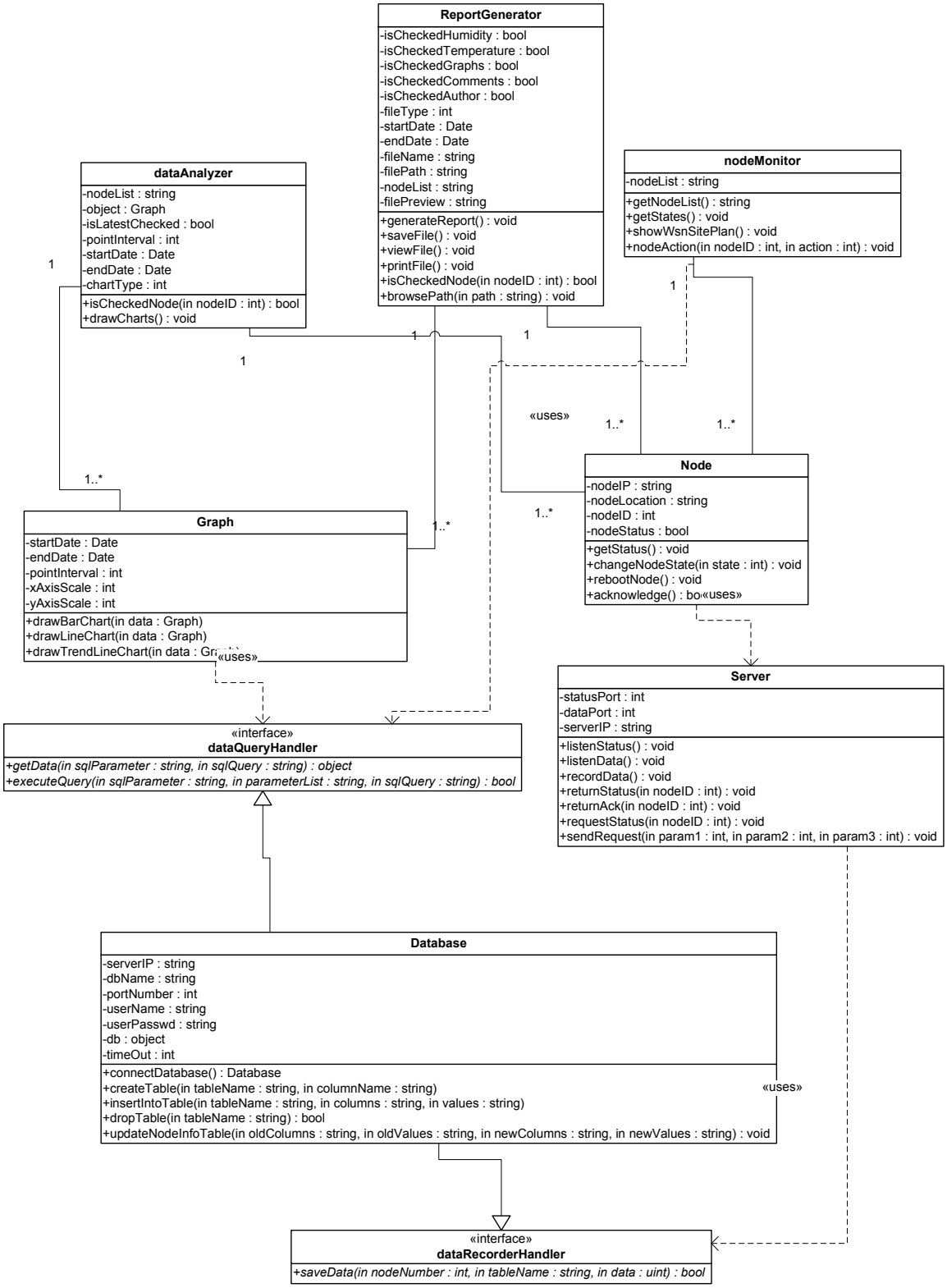


Figure 3-1 Class Diagram

3.2 Class Tables

NodeMonitor				
Attributes	**Attribute Name**	**Type**	**Description**	
	nodeList	Node[ ]	Holds all the information about sensors in the database	
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	getNodeList	Node[ ]	Void	Extracts all the information about sensors in the database
	getStates	Void	Void	Determines the states of all sensors with the help of server module
	showWsnSitePlan	Void	Void	Displays the existing site plan
	nodeAction	Void	Int, Int	Perform node applications(turn on&off, reboot node, ack node)

DataAnalyzer			
Attributes	**Attribute Name**	**Type**	**Description**
	nodeList	Node[ ]	Holds all the information about sensors in the database
	Graph	Object graph	

	isLatestChecked	Bool	Holds the value of the Latest checkbox	
	pointInterval	Int	Sets the x axis interval	
	startDate	Date	Start date of the measured data	
	endDate	Date	End date of the measured data	
	chartType	Int	Holds the chart type	
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	isCheckedNode	Void	Void	Determines whether checkbox of a node is checked or not
	drawCharts	Void	Void	Draws the charts according to user preference

ReportGenerator			
Attributes	**Attribute Name**	**Type**	**Description**
	nodeList	Node[ ]	Holds all the information about sensors in the database
	isCheckedTemperature	Bool	Holds the value of temperature checkbox
	isCheckedHumidity	Bool	Holds the value of humidity checkbox
	isCheckedGraphs	Bool	Holds the value of graph checkbox

	isCheckedComments	Bool	Holds the value of comments checkbox	
	isCheckedAuthor	Bool	Holds the value of author checkbox	
	fileType	Int	Holds the value of file type dropdownlist	
	startDate	Date	Holds the value of start from dropdownlist	
	endDate	Date	Holds the value of end dropdown list	
	fileName	String	Holds the value of file name textbox	
	filePath	String	Holds the value of save to textbox	
	filePreview	String	Holds the value of text version of generated data	
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	generateReport	Void	Void	Shows the data in preview textbox with respect to user preferences
	browsePath	Void	String	Browse the path of the file to be saved
	saveFile	Void	Void	Saves the generated data
	viewFile	Void	Void	Views the print preview of the generated data
	printFile	Void	String	Prints out the generated data

Graph				
Attributes	**Attribute Name**	**Type**	**Description**	
	startDate	Date	Start date of the measured data	
	endDate	Date	End date of the measured data	
	pointInterval	Int	Holds the interval of x axis selected by the user(minutely, hourly, daily)	
	xAxisScale	Int	Holds the interval of x axis scale	
	yAxisScale	Int	Holds the interval of y axis scale	
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	drawBarChart	Void	Void	Draws the bar chart of requested data
	drawLineChart	Void	Void	Draws the line chart of requested data
	drawTrendLine	Void	Void	Draws the line chart of requested data

Node			
Attributes	**Attribute Name**	**Type**	**Description**

	nodeIP	String	Holds IP of the node	
	nodeLocation	String	Holds the location of the node	
	nodeID	Int	Holds the unique id of the node	
	nodeStatus	Bool	Holds the state of the node	
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	getStatus	Void	Void	Sends a request to the server module to get the state of the node
	changeNodeState	Void	Int	Sends a request to the server module to change the state of the node
	rebootNode	Void	Void	Sends a request to the server module to reboot the node
	acknowledge	Bool	Void	Request the acknowledge message of the requested actions

Server			
Attributes	**Attribute Name**	**Type**	**Description**
	statusPort	Int	Holds the port number for the status
	dataPort	Int	Holds the port number for the incoming data

	serverIP	String	Holds the unique id of the server	
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	listenStatus	Void	Void	Listens the status port
	listenData	Void	Int	Listens the data port
	recordData	Void	Void	Records the data which is buffered from the data port
	returnStatus	Bool	Int	Returns the state of the node buffered from the status port to whom it requested it
	returnAck	Void	Int	Returns the acknowledgement of the requests.
	sendRequest	Void	Int, Int, Int	Sends a request to the Status Module in HSBS Sentinel in order to learn the state, turn on, turn off or reboot.

Database			
Attributes	**Attribute Name**	**Type**	**Description**
	serverIP	String	Holds the IP of the database server
	dbName	String	Holds the name of the database
	username	String	Holds the username of the user who will login to the database

	userPasswd	String	Holds the password of the user who will login to the database	
	timeOut	Int	Holds the connection timeout value	
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	connectDatabase	Bool	Void	Creates a connection to the database
	createTable	Void	String, String[]	Creates a table in the database
	insertIntoTable	Void	String, String[],	Inserts a values into a table as a row
	dropTable	Void	String	Deletes the table
	updateTable	Void	String[], String[], String	Updates the table with the new values

dataRecorderHandler <<interface>>				
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	saveData	Void	Int, String, Uint	Saves the data which is coming from the server module

dataQueryHandler <<interface>>

Methods	Method Name	Return	Arguments	Description
	getData	DataTable	String, String	Gets the result of the query from the database
	executeQuery	Bool	String, String, String	Executes the requested query

4 SEQUENCE DIAGRAMS

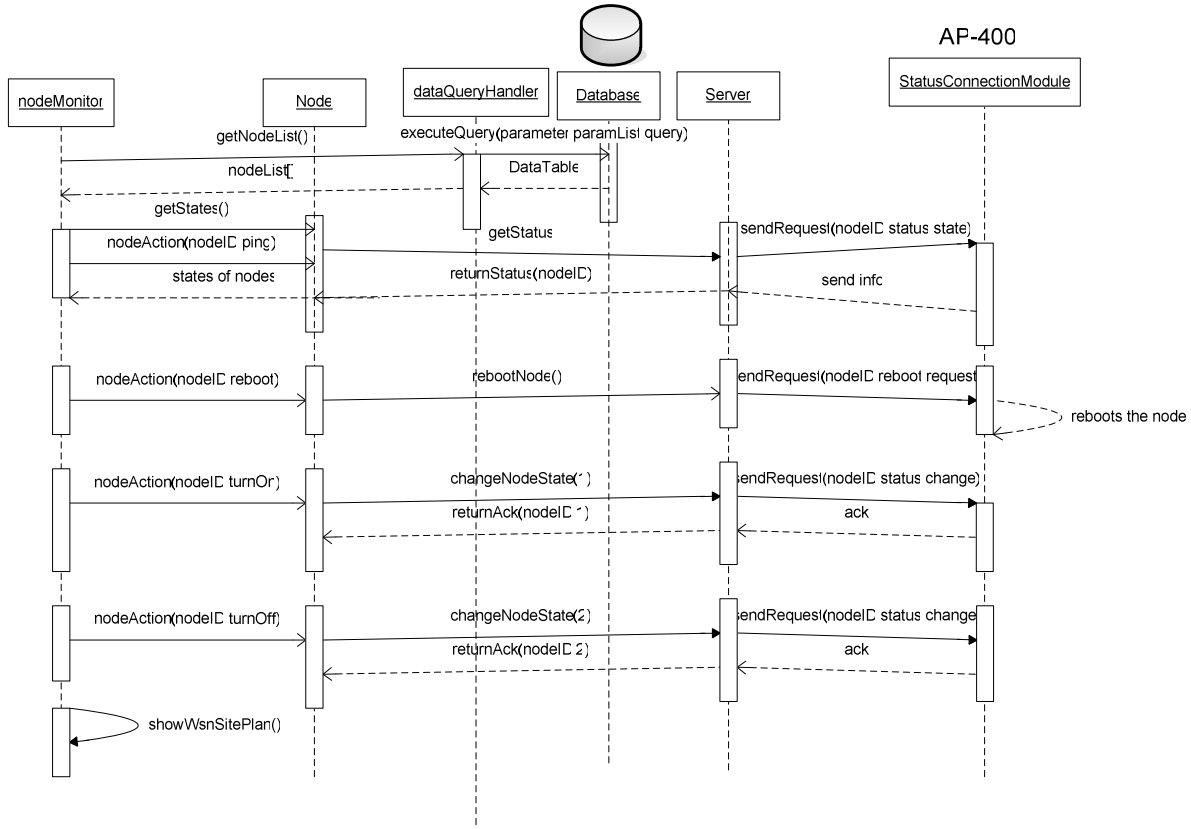


Figure 4-1 Node Monitor

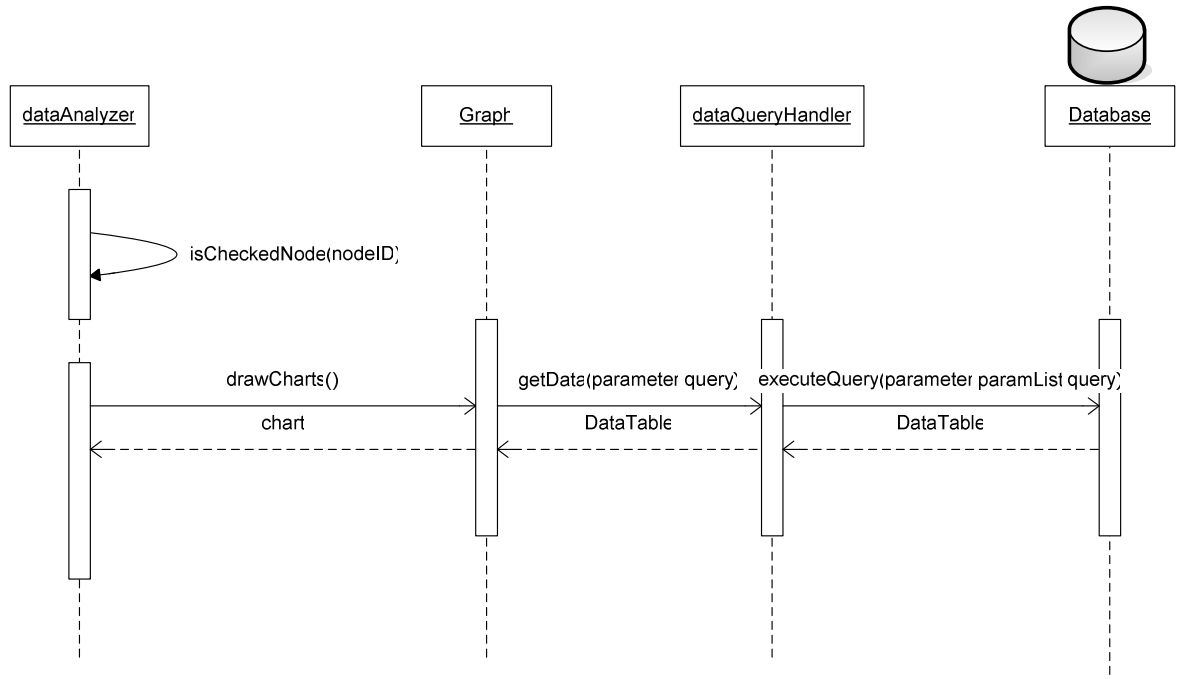


Figure 4-2 Data Analyzer

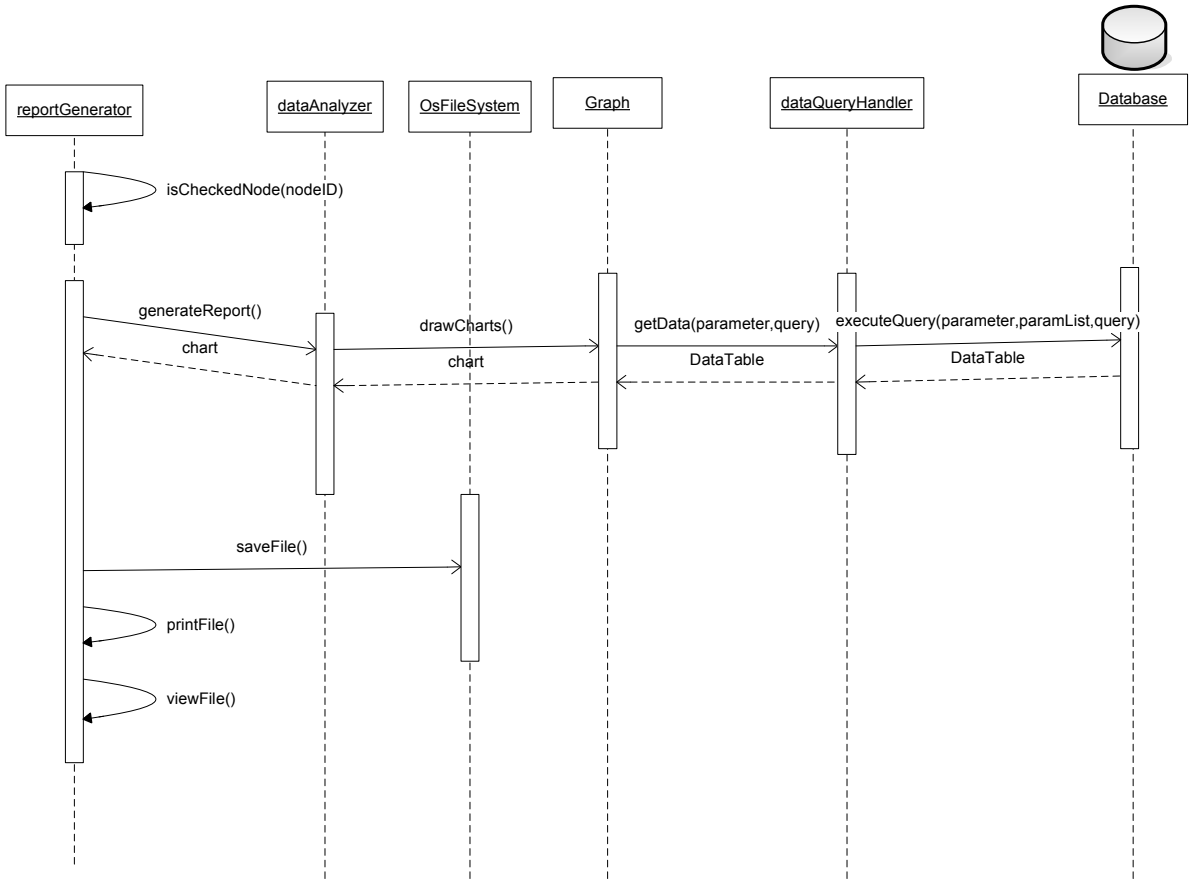


Figure 4-3 Report Generator

5 USER INTERFACE

The HSBS WSN Soft which will run on the server constitutes the user interface of the project. The user interface has three parts namely Node Monitor, Data Analyzer and Report Generator. The following text describes these three features in detail.

5.1 Node Monitor

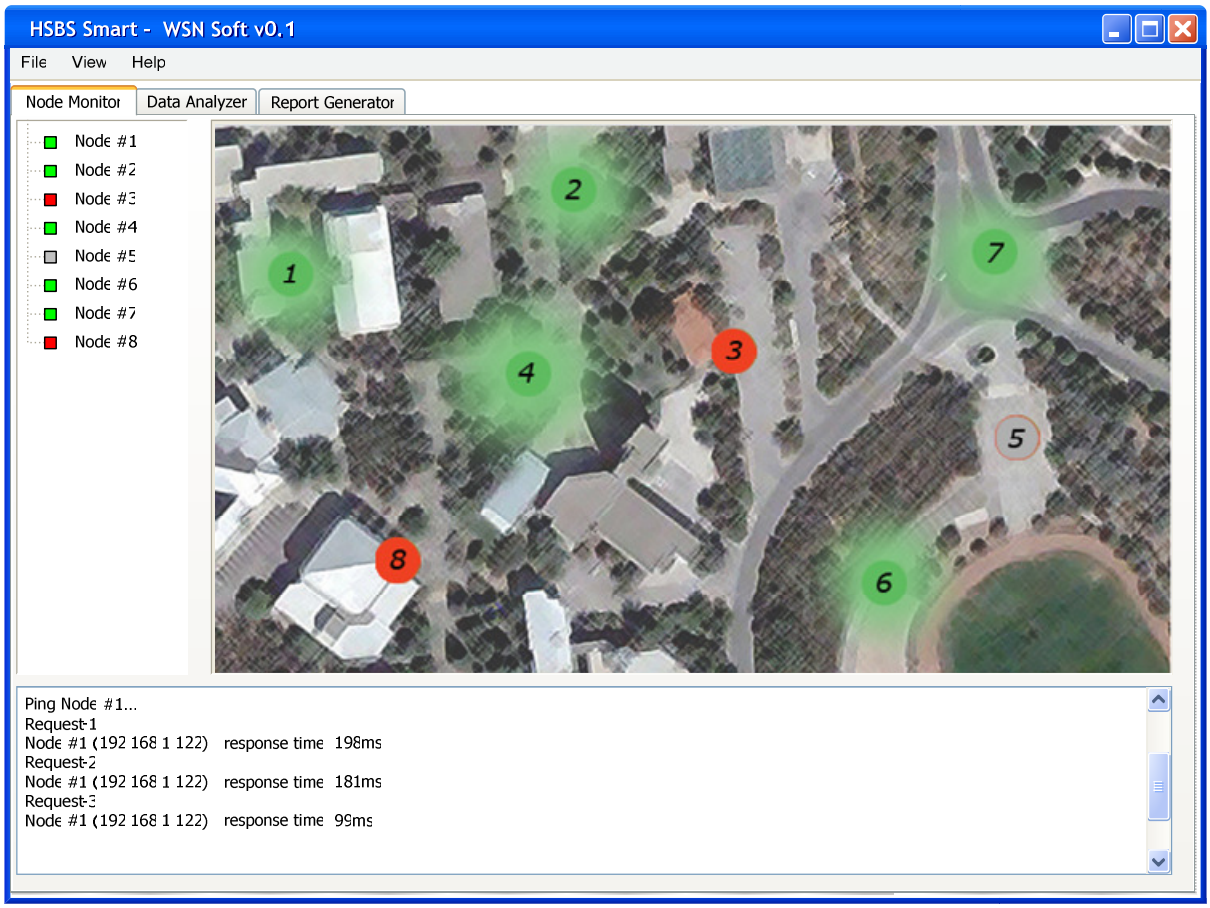


Figure 5-1 Node Monitor

Node Monitor tab has three panels as can bee seen in the Figure-5-1; node list, location map and activation output.

The left side panel lists the nodes with the names user has defined while initiating the system. Little squares before the node names illustrates the status of the node in such a way that "green" square means the node is alive, "red" square means node is turned off, "gray" square means node is unreachable. If a right click mouse event occurs on a node in this panel a menu shows up as shown in the Figure-5-2 allowing user to trigger four actions; ping the node, turn

the node off, turn the node on and reboot the node.

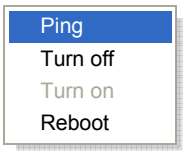


Figure 5-2 Right Click Menu

The location map illustrates the node locations according to their definition to the system. This part is only for only demonstrating the overall view of the system, no interactions are provided to user. The colored circles represent the positions of the nodes. The colors are captured from the left side node list panel.

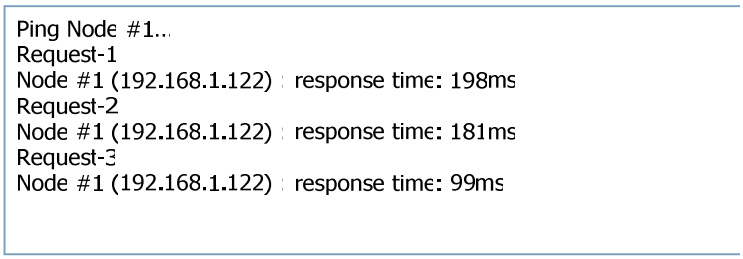


Figure 5-3 Activation Output

The activation output panel in Figure 5-3 displays the messages of system responses the user actions over nodes. The displayed text also recorded into “ActivationLog.txt” in file system.

5.2 Data Analyzer

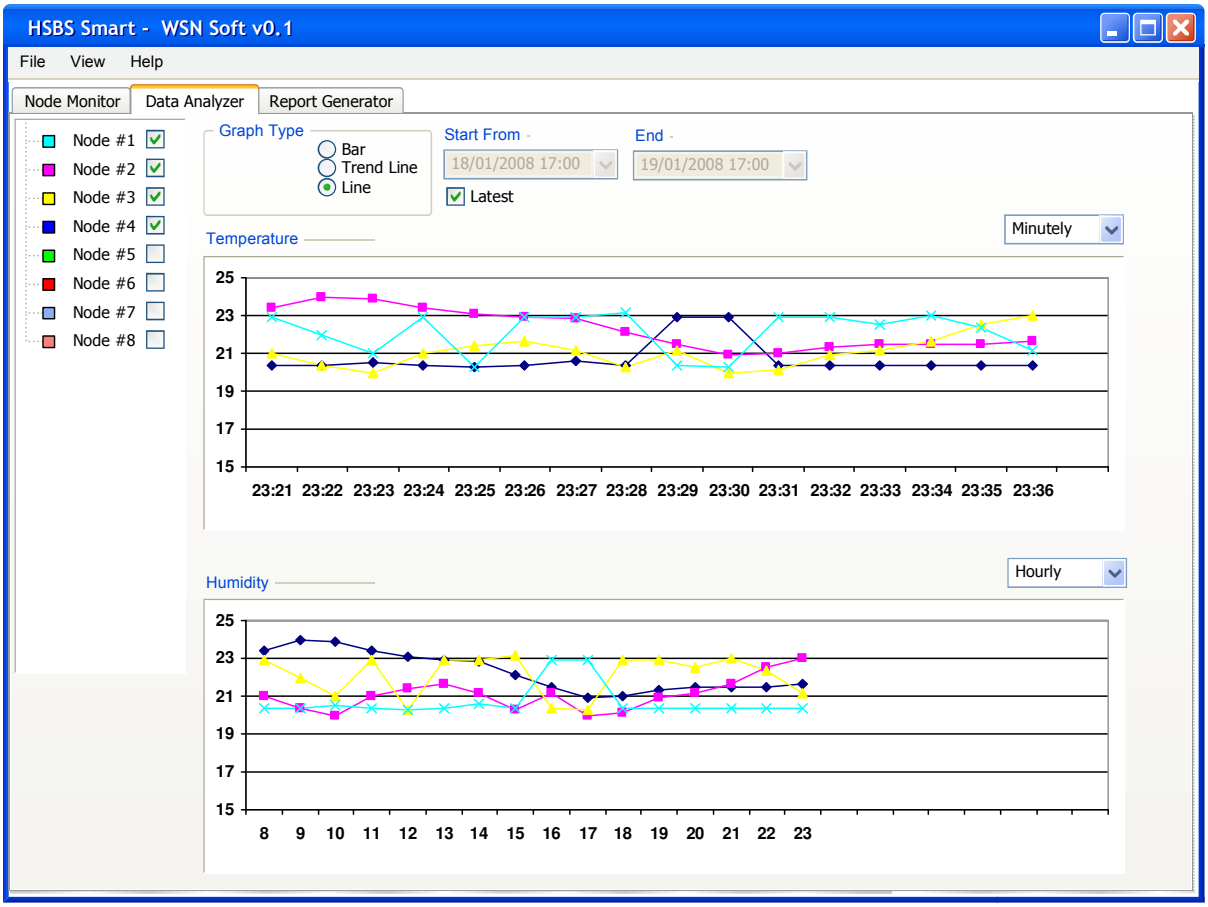


Figure 5-4 Data Analyzer

Data Analyzer tab allows user to see the recorded data on graph illustrations. The nodes defined in the system are listed in the left side panel and specified types of graphs are plotted on the window according to the options specified by user using options panel.

The node list on the left side panel includes checkboxes for each node and user can add or remove the nodes to be displayed in the graphs. Different colors are assigned to each node to make the graphs more understandable.

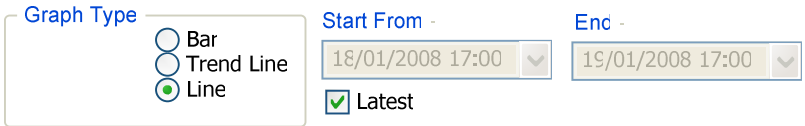


Figure 5-5 Graph Options Panel

The options panel in the Figure 5-5 consists of two parts, first the graph type radio group and second part is time panel to set the interval of the graph to be plotted. In the graph type radio group there are three types of graphs as, bar, trend line and line and if a change event occurs

the graphs are redrawn. The time panel allows user to define the start and end points of the graph in means of time. The user also can select to display latest time interval and let the program select the interval.

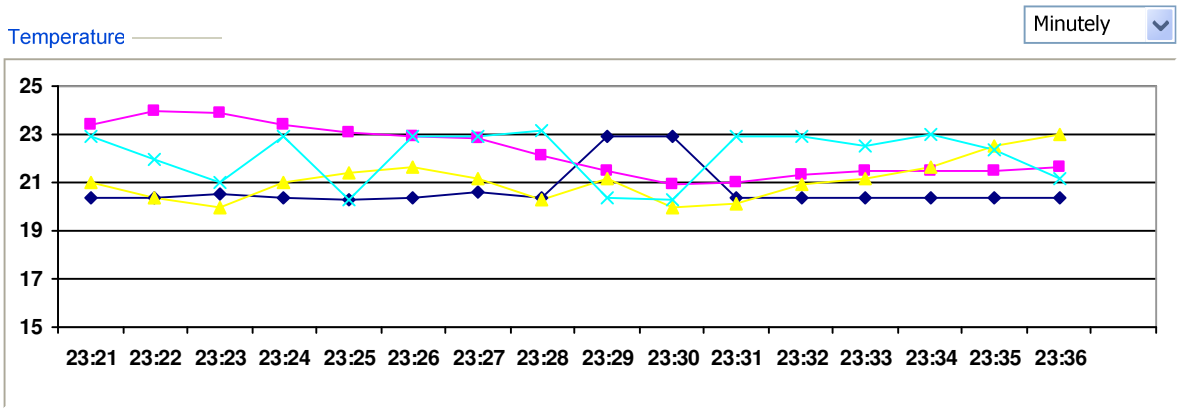


Figure 5-6 Plotted Graphs

There always two graphs are drawn a temperature and a humidity like in Figure 5-6. User can set the frequency of the data shown from the combo box on the right top of each graph. The minutely, hourly and daily options are listed here and if user changes this setting the graph related is redrawn. Each line or bar on the graph is related to a node on the left side node list panel.

5.3 Report Generator

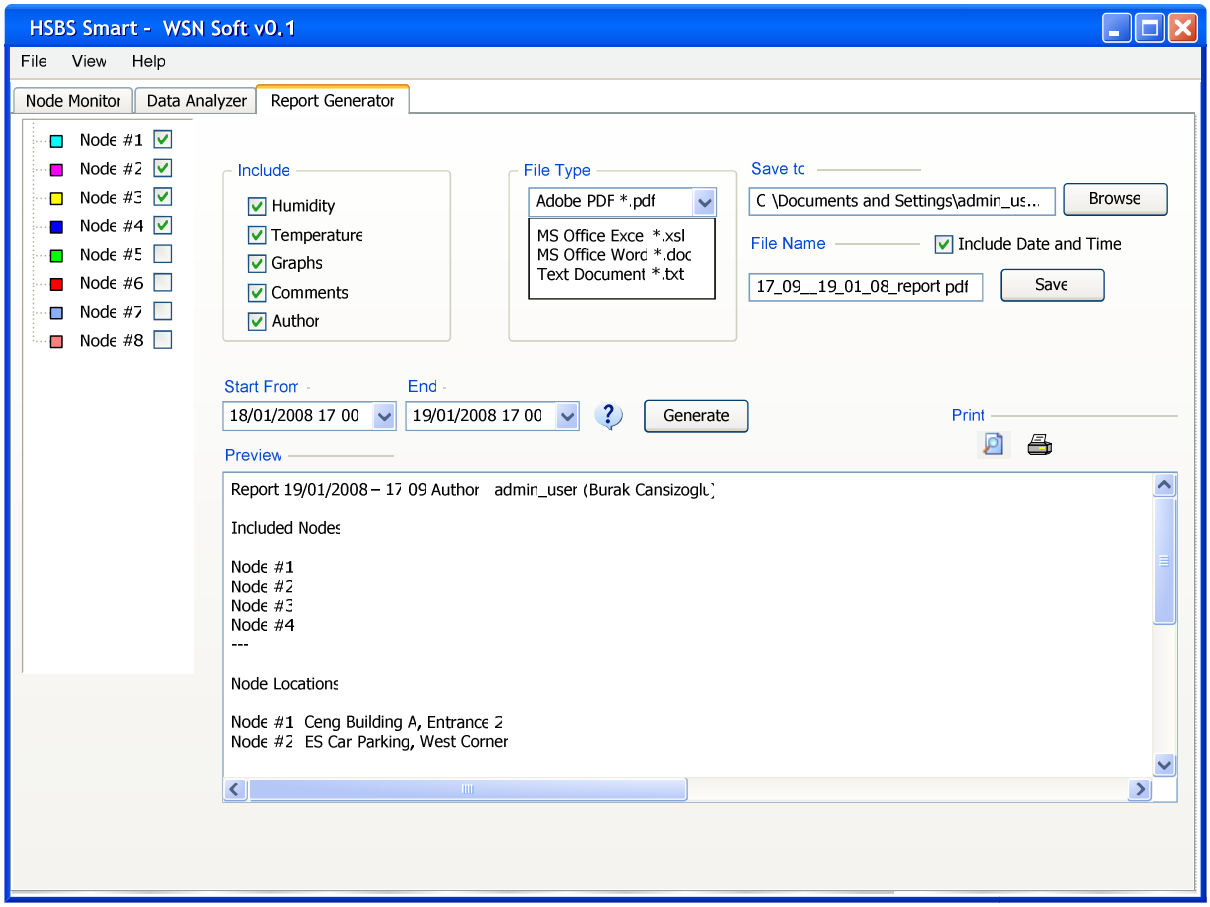


Figure 5-7 Report Generator

Report Generator in tab in Figure 5-7 contains seven parts as left side node list panel, include option panel, file save panel, generate report panel, time panel, print panel and preview panel.

The left side node list panel is same with the panel in Data Analyzer tab and lets the user by check boxes to define to include or exclude the nodes which the report will be created about.

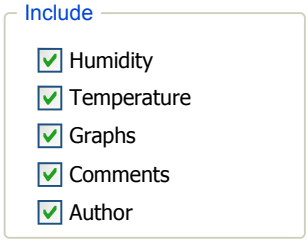


Figure 5-8 Include Panel

Include options panel allows user to select the information which is going to be reported. User can leave out the unwanted info by deselecting the checkboxes on each row. The include options panel can be seen in detail in Figure 5-8.

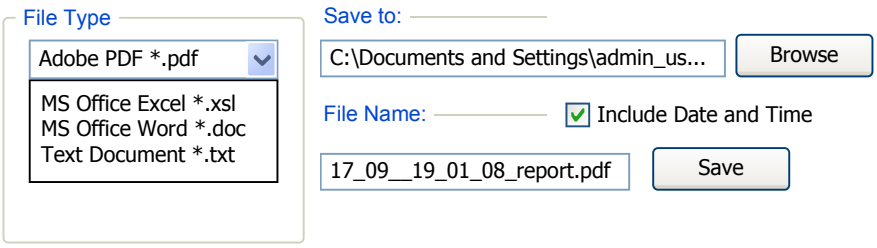


Figure 5-9 File Save Panel

The file save panel in Figure 5-9 has two parts; first one is file type selection combo box and second part is path and file name textbox and buttons. The user can select through four types of file formats which are Adobe PDF, MS Office Excel, MS Office Word and plain text document. Save to text input box displays the path where the file to be saved and file name text input file allows user to type in a custom name. The user can check the "include date and time" check box allowing program to add time and date info to file name.

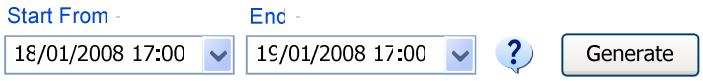


Figure 5-10 Generate Report Panel

Generate report panel consists of a time panel, a help icon and a generate button which are used for defining the time interval that the information in the report will be built on, displaying a calendar for supporting the user while deciding on interval and generating the report accordingly.

Figure 5-11 Print Panel

Print panel has two icons representing the print preview and print functions. User can preview the generated report before printing by clicking on the print preview icon and can print the report by clicking the print icon.

The preview panel is to display the preview of the generated report in a simple text view mode. Objects like graphs or images are included as tags, (<<graph01>>, <<image002>>).

6 PROJECT SCHEDULE

6.1 Finished Work

- The SHT15 Temperature & Humidity Sensor Datasheet has been examined. Furthermore, the schematic of SHT15 has been analyzed.

- RS-232 Serial Communication Bus Protocol which is used in data transferring between AP-400 and PIC 16F877 has been examined.

- We have assembled a SHT15 on a CENG 336 Board in order to try our PIC-sensor protocol.

- We have implemented the PIC Module. This module retrieves the data from a sensor, processes it and sends this data to AP-400 over RS-232.

- After implementing the PIC Module successfully, next mission was to obtain a SP07. In order to obtain a SP07, firstly we have drawn the schematic and PCB layout of SP07. Next, we have printed the board with the help of this schematic and layout.

- We have configured Embedded Linux kernel of AP-400 and reloaded it. We have disabled the serial console of AP-400 which was implemented by default and now, AP Module can use the serial port.

- We have implemented a basic version of AP Module. This module reads the values from the serial port and sends the data to the Server Module on the server.

- We have implemented a basic version of Server Module. This module gets the data from an HSBS Sentinel over TCP/IP protocol, checks the data against an alert condition and prints an alert message on the screen if the data violates the alert condition. Furthermore, this module saves the incoming data into the database.

- We have created the database and necessary tables.

- For the prototype demo, we have implemented a simple interface that is used to show the last 15 temperature and humidity values. The data shown is refreshed every 5

seconds.

- Now, we are able to read data from sensors to PIC, send it to AP-400 over RS-232 port, read data from PIC to AP-400 over serial port, send it to the server over TCP/IP, read data from HSBS Sentinel to server over TCP/IP, save it to the database and show the data via a simple user interface.

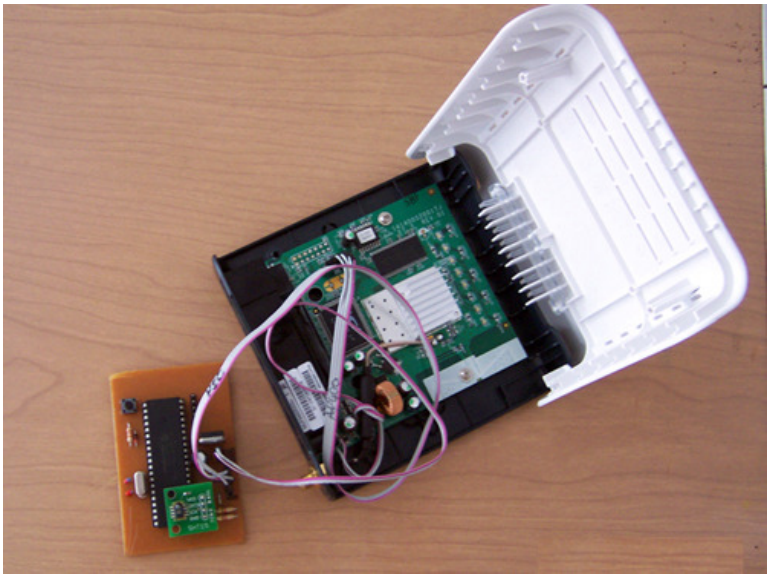


Figure 6-1 HSBS Sentinel v1.0

6.2 Future Work

- We are going to implement two more SP07s. With these two SP07s and two AP-400s, we are going to obtain two more HSBS Sentinels and totally with the existing one, we will have three HSBS Sentinels.

- We are going to upgrade the AP Module. In the demo version we have implemented, AP-400 reads data from the serial port periodically. However, the final AP Module will send a request to the PIC Module and the PIC Module will send a temperature value or a humidity value depending on the request. Furthermore, alert condition testing will be implemented.

- We are going to modify the PIC Module a bit. In the demo version, PIC Module sends temperature and humidity values to the AP Module periodically. In the final version, PIC Module will send a temperature or a humidity value depending on the request from the AP Module.

- We are going to implement the Status Module. This module, will turn off, reboot or turn on the node depending on the request from the Server Module. Moreover, it will get the user defined alert conditions from the Server Module and write them to the flash of AP-400. Furthermore, this module will send data about the state of the node to the Server Module.

- We are going to modify the Server Module significantly. In the demo version, it just gets temperature and humidity values from AP Module and records the data to the database. However, in the final version, in addition to the demo version, the Server Module will send turning on and off and rebooting requests to the Status Module. Moreover, it will send user defined alert conditions to the Status Module. Furthermore, it will get the status of the node from the Status Module. In addition to these, this module get the alert messages from the AP Module.

- We are going to implement the Main Module. Generally, this module is an interface for Server, GUI and Database Modules. In addition, it will handle alert conditions.

- The GUI Module will be implemented. This module will provide a user interface and generally, this interface will be used to monitor temperature and humidity values of the specified nodes and states of the nodes, to turn on, turn off and reboot the nodes and to create reports of the information about the nodes.

- The next step is to build a wireless mesh network via a number of HSBS Sentinels, some repeaters, an access point and a server.

- The last step is to use HSBS_WSN in a scenario. This scenario will be created in the spring semester.

6.3 Gantt Chart

Gantt Charts of first and second semester can be found in Appendix-A and Appendix-B accordingly.

7 REFERENCES

1. Wikipedia, Wireless Sensor Network, October 2007, http://en.wikipedia.org/wiki/Wsn

2. Jason Lester Hill, System Architecture for Wireless Sensor Networks, 2003, http://www.jlhlabs.com/jhill_cs/jhill_thesis.pdf

3. SHT1x Breakout Board, 2007, http://www.sparkfun.com/datasheets/Sensors/ SHT15-Breakout-Schematic.pdf

4. P. Raghavan, Amol Lad, Sriram Neelakandan, "Embedded Linux System Design and Development", Auerbach, (2005).

5. Data_Sheet_humidity_sensor_SHT1x_SHT7x_E.pdf,2007 http://www.sparkfun.com/commerce/product_info.php?products_id=8257/Data_Sheet_humidity_sensor_SHT1x_SHT7x_E.pdf

APPENDICES

Appendix A

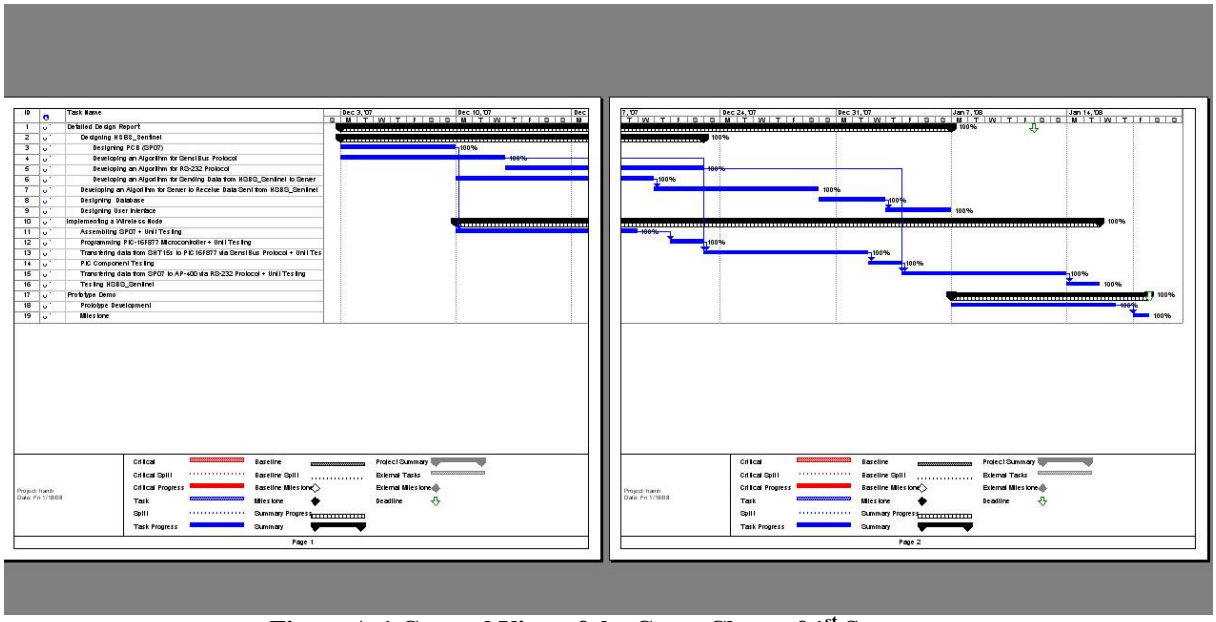
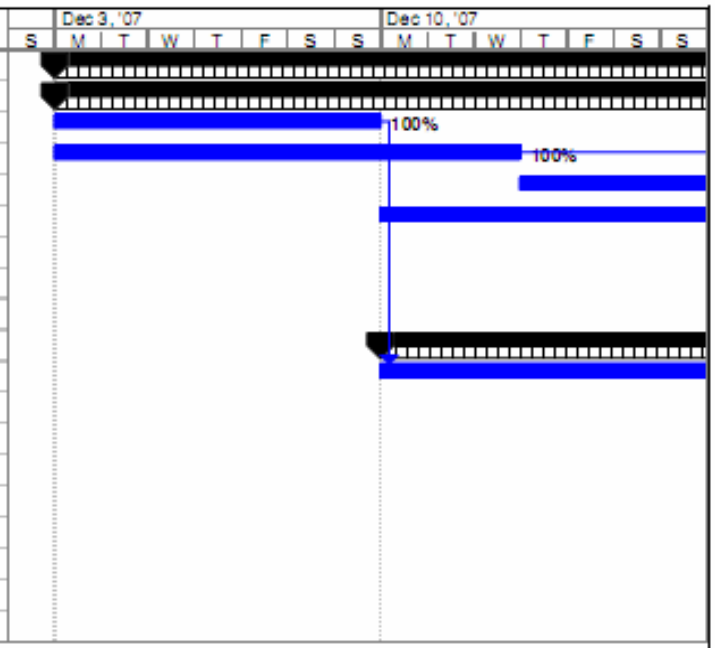


Figure A-1 General View of the Gantt Chart of 1st Semester

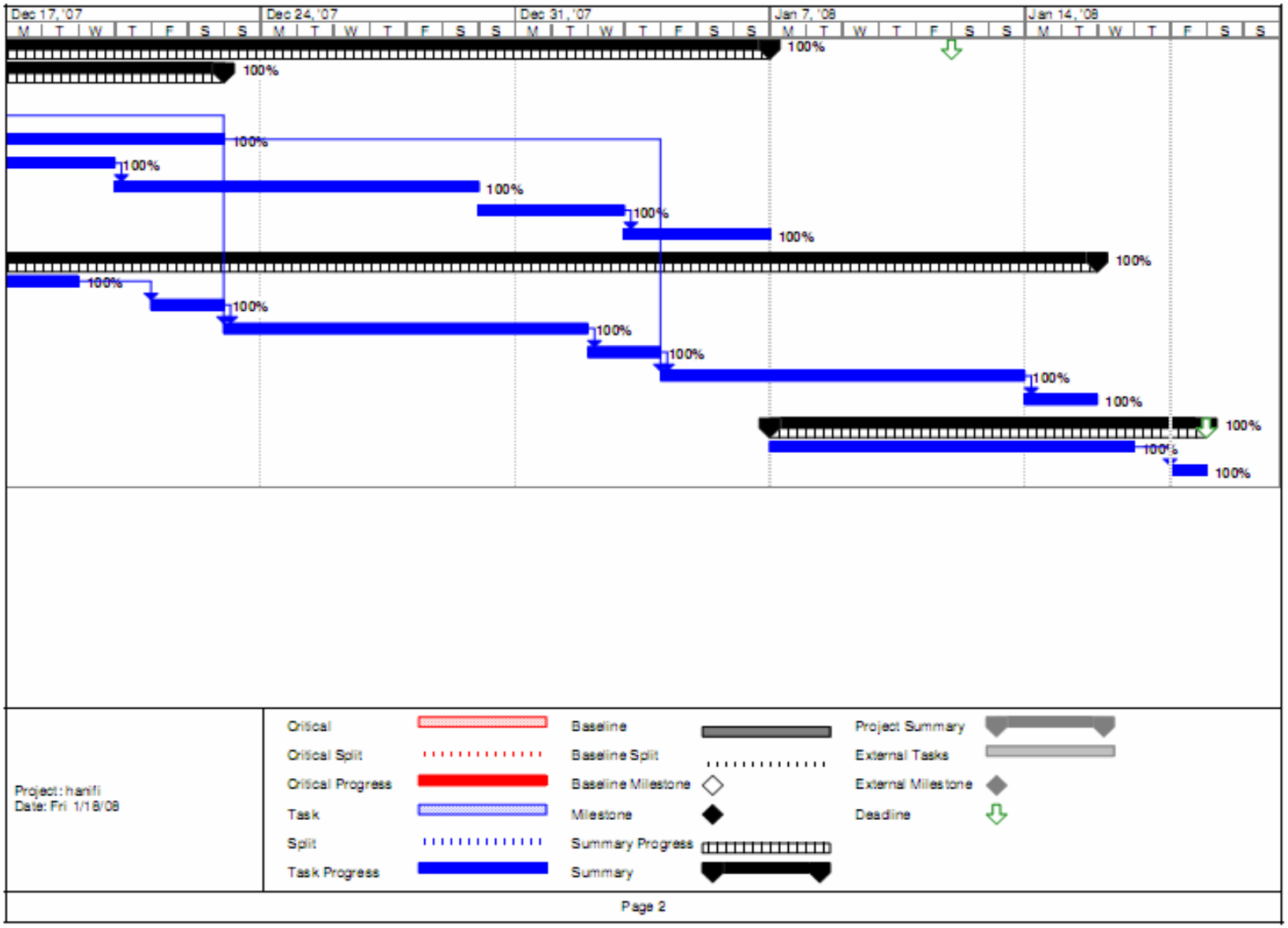
Following 2 images show the Gantt Chart of 1st Semester in detail.

ID	Task Name
1	**Detailed Design Report**
2	**Designing HSBS_Sentinel**
3	Designing PCB (SP07)
4	Developing an Algorithm for SensiBus Protocol
5	Developing an Algorithm for RS-232 Protocol
6	Developing an Algorithm for Sending Data from HSBS_Sentinel to Server
7	Developing an Algorithm for Server to Receive Data Sent from HSBS_Sentinel
8	Designing Database
9	Designing User Interface
10	**Implementing a Wireless Node**
11	Assembling SP07 + Unit Testing
12	Programming PIC-16F877 Microcontroller + Unit Testing
13	Transfering data from SHT15 to PIC16F877 via SensiBus Protocol + Unit Testing
14	PIC Component Testing
15	Transfering data from SP07 to AP-400 via RS-232 Protocol + Unit Testing
16	Testing HSBS_Sentinel
17	**Prototype Demo**
18	Prototype Development
19	Milestone



Project: hanifi
Date: Fri 1/18/08

Critical
Critical Split
Critical Progress
Task
Split
Task Progress
Baseline
Baseline Split
Baseline Milestone
Milestone
Summary Progress
Summary
Project Summary
External Tasks
External Milestone
Deadline


Dec 17, '07
Dec 24, '07
Dec 31, '07
Jan 7, '08
Jan 14, '08
M T W T F S S
100%
Project: hanifi
Date: Fri 1/18/08
Critical
Critical Split
Critical Progress
Task
Split
Task Progress
Baseline
Baseline Split
Baseline Milestone
Milestone
Summary Progress
Summary
Project Summary
External Tasks
External Milestone
Deadline
Page 2

Appendix B

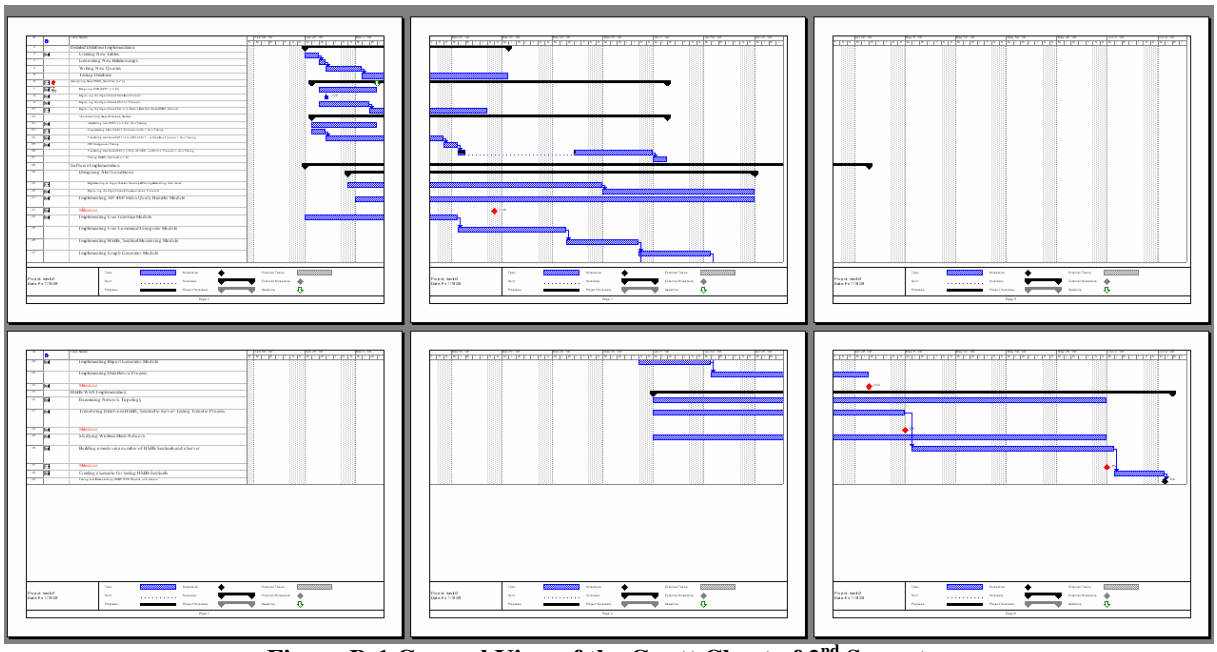
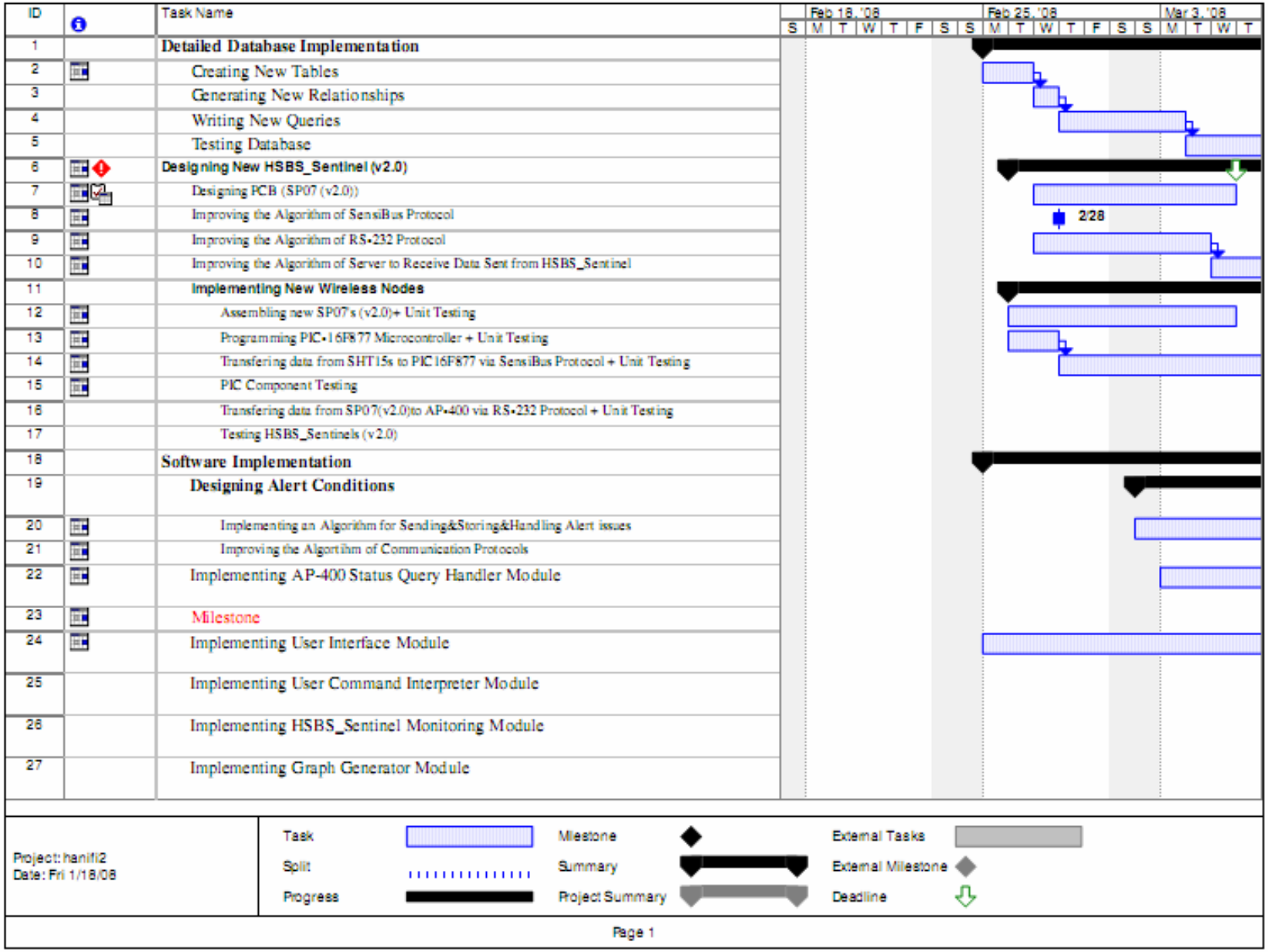
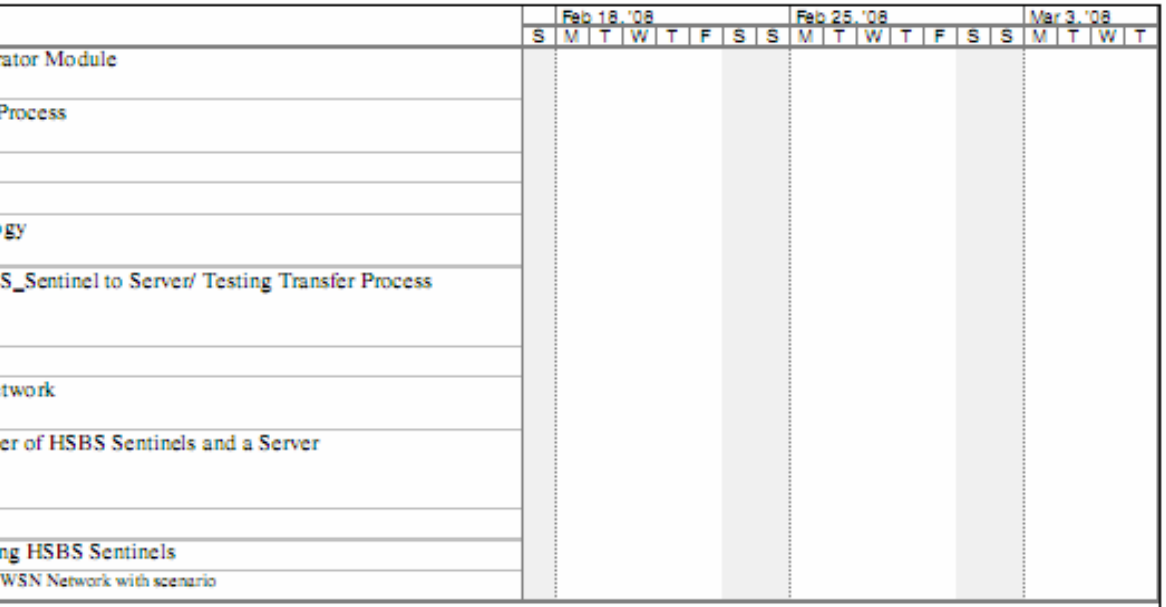


Figure B-1 General View of the Gantt Chart of 2nd Semester

Following 6 images show the Gantt Chart of 2nd Semester in detail.

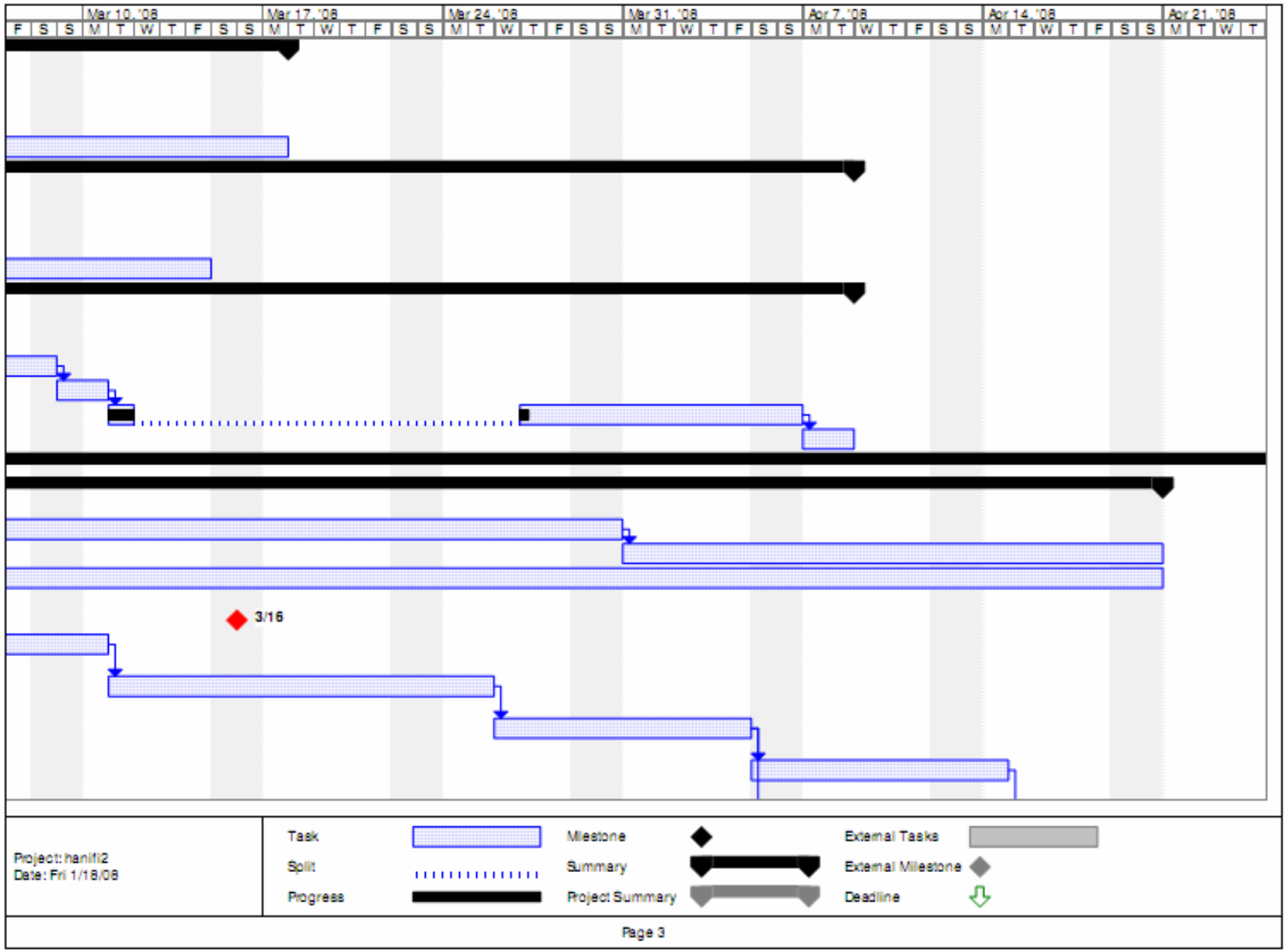

ID
Task Name
Feb 18, '08
Feb 25, '08
Mar 3, '08
S M T W T F S S M T W T F S S M T W T
1 Detailed Database Implementation
2 Creating New Tables
3 Generating New Relationships
4 Writing New Queries
5 Testing Database
6 Designing New HSBS_Sentinel (v2.0)
7 Designing PCB (SP07 (v2.0))
8 Improving the Algorithm of SensiBus Protocol
9 Improving the Algorithm of RS-232 Protocol
10 Improving the Algorithm of Server to Receive Data Sent from HSBS_Sentinel
11 Implementing New Wireless Nodes
12 Assembling new SP07s (v2.0)+ Unit Testing
13 Programming PIC-16F877 Microcontroller + Unit Testing
14 Transfering data from SHT15s to PIC16F877 via SensiBus Protocol + Unit Testing
15 PIC Component Testing
16 Transfering data from SP07(v2.0)to AP-400 via RS-232 Protocol + Unit Testing
17 Testing HSBS_Sentinels (v2.0)
18 Software Implementation
19 Designing Alert Conditions
20 Implementing an Algorithm for Sending&Storing&Handling Alert issues
21 Improving the Algorithm of Communication Protocols
22 Implementing AP-400 Status Query Handler Module
23 Milestone
24 Implementing User Interface Module
25 Implementing User Command Interpreter Module
26 Implementing HSBS_Sentinel Monitoring Module
27 Implementing Graph Generator Module
2/28
Project: hanifi2
Date: Fri 1/18/08
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline
Page 1

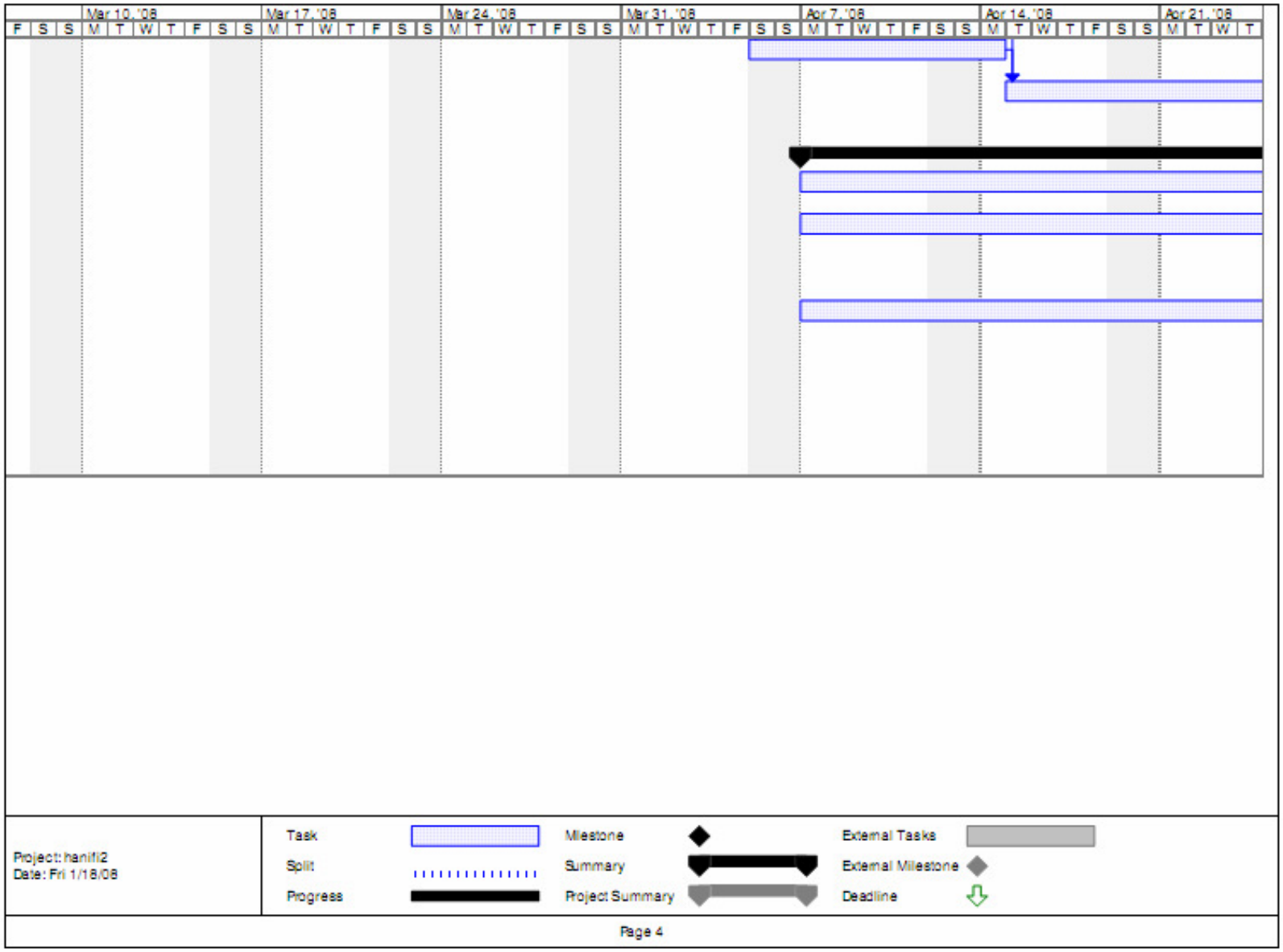
ID	Task Name
28	Implementing Report Generator Module
29	Implementing Data Return Process
30	Milestone
31	**HSBS WSN Implementation**
32	Examining Network Topology
33	Transfering Data from HSBS_Sentinel to Server/ Testing Transfer Process
34	Milestone
35	Studying Wireless Mesh Network
36	Building a mesh via a number of HSBS Sentinels and a Server
37	Milestone
38	Creating a scenario for testing HSBS Sentinels
39	Testing and Demonstrating HSBS WSN Network with scenario



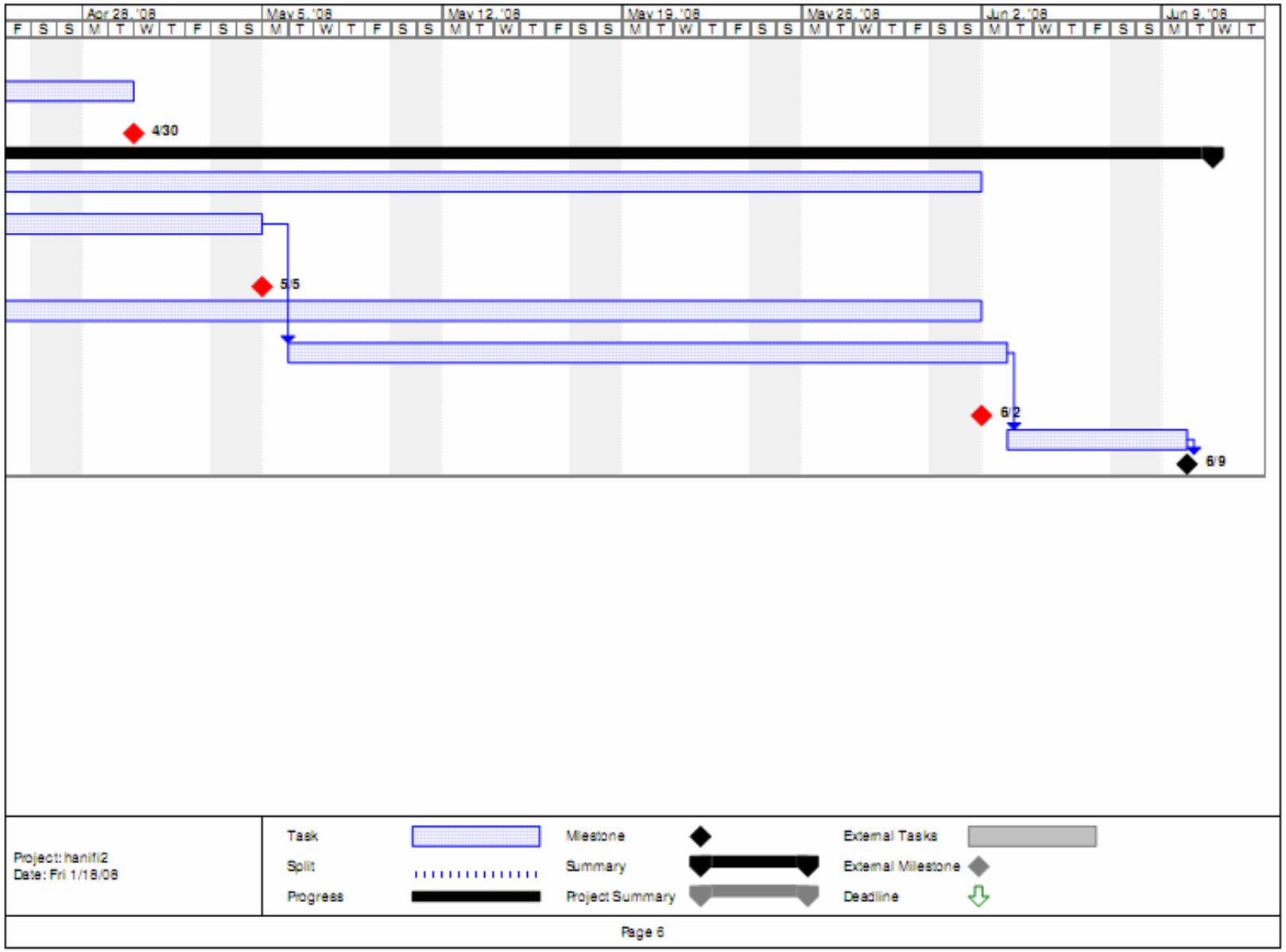
Project: hanifi2
Date: Fri 1/18/08

Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline


Mar 10, '08
Mar 17, '08
Mar 24, '08
Mar 31, '08
Apr 7, '08
Apr 14, '08
Apr 21, '08
F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T
3/16
Project: hanifi2
Date: Fri 1/18/08
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline
Page 3


Mar 10, '08
Mar 17, '08
Mar 24, '08
Mar 31, '08
Apr 7, '08
Apr 14, '08
Apr 21, '08
F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T
Project: hanifi2
Date: Fri 1/18/08
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline
Page 4


Apr 28, '08
May 5, '08
May 12, '08
May 19, '08
May 26, '08
Jun 2, '08
Jun 9, '08
F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T
Project: hanifi2
Date: Fri 1/18/08
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline
Page 5


Apr 28, '08
May 5, '08
May 12, '08
May 19, '08
May 26, '08
Jun 2, '08
Jun 9, '08
F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T
4/30
5/5
6/2
6/9
Project: hanifi2
Date: Fri 1/18/08
Task
Split
Progress
Milestone
Summary
Project Summary
External Tasks
External Milestone
Deadline
Page 6