CENG 491

Senior Design Project and Seminar

DIGIPOST

Poster in Blue

Final Design Report

1 Introduction

1 1. Definition and Scope

'PosterInBlue' is a digital poster with interactive Bluetooth. Digital poster enables its users to take the advantage of digital advertising. Besides being more attractive, digital posters will be more useful than any other advertising media. The customers will be able to show their advertisements on LCDs instead of papers. What is more, they will be able to show animating posters, which will capture the attention of their target group. By using Bluetooth technology, all these advertisements will be more permanent, since the advertised information will also be sent to the audience's Bluetooth devices. User friendly GUI will also enable Poster in Blue customers to design their posters easily and effectively.

In Scope:

As we searched the web for possible application areas, we have learned that, due to rapid developments in mobile technology and expanding use of wireless communication there is an increasing demand on Bluetooth based applications. Bluetooth technology is used for many different applications.

Among them we are going to implement the followings:

- Information Sharing (Business Cards, Reminders, Presentations)
- Advertisements (Poster images, Promotion information)

Not In Scope:

Due to hardware restrictions continuous data transfer is not in our scope (ie. Music broadcast, video broadcast) Also broadcasting for large areas like city or country is not in our scope since the range of Bluetooth is max 100mt. Lastly for ethical reasons non on-demand protocols are not in our scope like forced advertisement.

1 2. Design Goals and Objectives

Our design goal is to achieve a well structured system by decomposing the whole system into subsystems. We will try to decompose the system such that subsystems will be highly cohesive and low coupled. This type of design will be helpful both for us to visualize, specify and construct our project. It will also be helpful for the people concerned of our project to understand it more easily. Working on such a design will also enable us to handle unavoidable changes more easily; therefore the software will be maintainable. Besides, we will try to simplify the design of user interface in order to make it more understandable for our potential customers.

1 3. Project Features

'PosterInBlue' project is initially given as a Digital Poster project. We have added new features in order improve flexibility of our design. Core features of our project are:

- Sending image files as advertisements
- Sending vCards[1]
- Sending vCalendars[2]

which are shown in LCD monitors at a public area and broadcasted via Bluetooth.

Extra features that we are planning to implement are:

- Sending Presentations
- Sending Animations

[1] **vCard** is a file format standard for personal data interchange, specifically electronic business cards

[2] **vCalendar** is an electronic calendaring and scheduling exchange format

1 4. Schedule

Here is the job division of our project and its corresponding time schedule. Firstly, we planned to implement the prototype of the system in 20 days. Prototype implementation includes a simple image animation, establishing a basic Bluetooth connection, further graphical interface design and demonstration of these all. We planned to do all these tasks in parallel.

As a second part, implementation of the project will take almost 4 months. Implementation includes two seperate packages named Bluetooth and FPGA packages. All these packages covers subtasks as Blueradios Module Integration, finalizing package and testing package. It is also likewise for the FPGA package which consists of Flash Memory controller, RS232 controller, Main controller designs and finalizing and testing overall FPGA circuit design.

As a last part we planned to integrate seperate packages namely GUI, FPGA design and Bluetooth package in almost 3 weeks. Finally we will make the demonstration in June of 2007.

2 Design Considerations

2 1. Interoperability

Firstly, data packages are to be sent from admin PC to VGA board via standart Bluetooth kit. Bluetooth profiles are used for this communication.

Bluetooth kit and FPGA board are other components that communicate. Serial pins on the support board that we are going to design enable these two devices to communicate.

VGA board and LCD display communicates with the application we implement. According to our research our board is one of the best choices for LCD display.

Furthermore, there are receiver hardware devices which poster data are going to be sent by the BlueRadios kit. Every receiver device having Bluetooth communication will obviously enable us to communicate with them via standard Bluetooth Protocols.

2 2. Stability and Portability

One of the main problems of this design is that our XSA-3S1000 has only 2Mbyte Flash. We have a large SDRAM (32MB) and this is very suitable for getting high performance from our design. However, this RAM is not a stable storage device and in case of any power cut we will loose the data in our SDRAM. Due to our research about how we can solve this problem we have discovered that we may design a new energy supply board or may find a battery which supplies power to both our XSA and BlueRadios. This would not only provide stability but also portability to our system.

However, our further research showed us that none of the above methods totally ensures stability. Because battery may not provide sufficient energy to both boards and it may fail in case of long power cuts. Similarly, a power supply board would also fail.

We decided that to ensure both stability and portability we should design a new board which has both our XESS board and Blueradios kit on it. It should also contain an external flash memory and RS232 port. This external flash solves our problem of limited non-volatile memory. Poster data and bit-stream file of our VHDL design is kept on this larger flash memory and after a power cut system fetches older configuration data from there.

This new board enriches the portability of whole design in a way that we can put our XES board and BlueRadios on this single board then put it into a one closed box. Customers will only see a VGA Port and a plug out of the box.

2 3. Security

Our main concern about security mainly depends on admin actions. Since 3rd party users cannot make any changes in program or program data. We assure this property by setting one way communication between our Bluetooth server and 3rd party users. Our on demand protocol about data communication also makes one way data connection.

In admin connection our focus about security is connection time. Considering safety we avoided on the fly configuration. Instead of keeping the connection alive we first make connection in order to check the Administrator Password. During this connection our Broadcasting service goes offline for a short period of time for full security. This time is approximately 0.1 second which may be seen negligible for 3rd party users. By disabling broadcasting service we are quite sure about who connects our Configuration module. After confirming admin password we process the desired operation. The other advantage

of this method is off-line working. Admin may fetch data in VGA[3] display. After fetching data admin can make necessary changes without VGA shutdown. As we said earlier this improves system in principles of minimum offline time & maximum security.

Below is the FSM[4] of our security system:

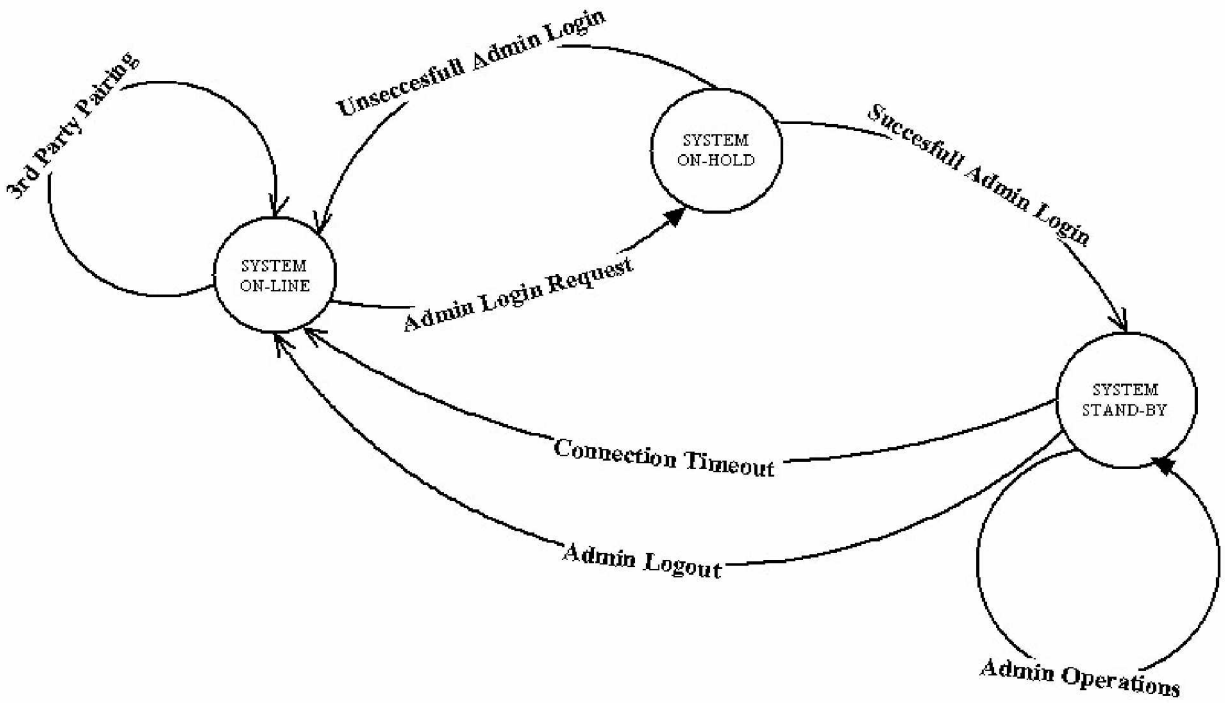


Figure 1: FSM of Security System

As expressed in diagram admin timeout is another property of our program. After certain period of time if no action was taken by admin, system restores itself back to online position.

Authentication protocol between admin and Bluetooth Module is ensured by Bluetooth security protocols. We treat all devices as entrusted devices. Which means in all

[3] **Video Graphics Array** (**VGA**) is an analog computer display standard first marketed in 1987 by IBM

[4] A **finite state machine** (FSM) or **finite state automaton** (plural: *automata*) is a model of behavior composed of a finite number of states, transitions between those states, and actions.

operations password (passkey) is required. Detailed diagram for pairing and authentication **[1]** can be seen below:

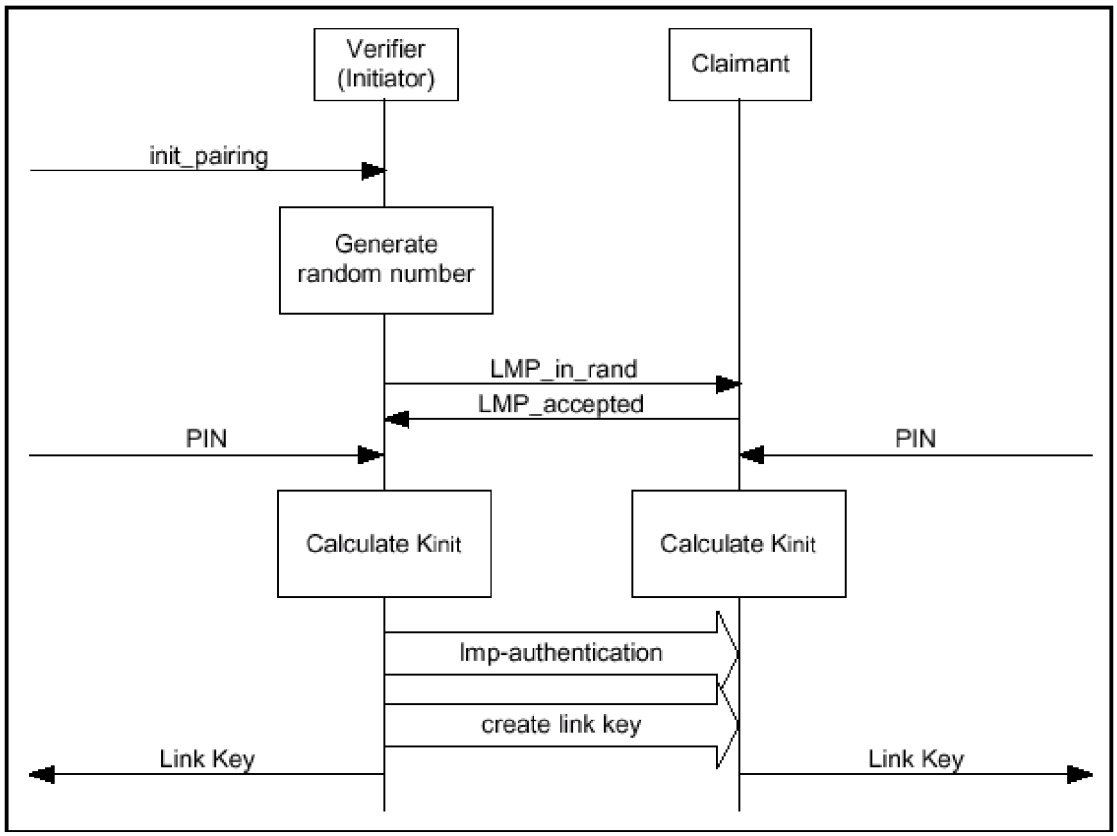


Figure 2: Pairing and Authentication

INFORMATION ABOUT LMP-Pairing: It is a procedure that authenticates two devices, based on a PIN, and subsequently creates a common link key that is used as the basis for a trusted relationship or a secure connection. This procedure consists of the steps:

- Ø creation of an initialization key based on a random number and PIN **[2]**
- Ø LMP-authentication based on the initialization key and creation of the common link key

3 System Architecture

Poster in Blue is a combined hardware/software Bluetooth solution the system of which must be designed to be extendible in the future. Besides, for this system conquering self manageable pieces will be a wise strategy to overcome the complexity of the problem. So, instead of designing the project as an “all of one piece” system which is supposed to be really massive, we decided to divide the system into modules. Designing it in this way will mainly provide us some degree of freedom that

- we will have a reduced and more specific problem space to overcome
- we can further enhance the system by extending each module
- we can even enlarge the system by adding new modules

How we divided the system into modules? We firstly considered possible usage scenarios and in these scenarios we think about internally independent looked parts. For instance, displaying posters onto the VGA, the administration or sending-receiving data over Bluetooth can be thought as modules. Then we further think about these parts and realized some other internal necessities of these parts, for example how to prepare the data to be sent-received over Bluetooth. As a result we decided to construct our system with these 5 modules:

1) Admin Tool (GUI)
2) Configuration Module
3) HCI Module (RS232)
4) VGA Display
5) Broadcast Module

3 1. Admin Tool (GUI)

Admin module is basically the administration tool we will design for configuration of the posters, vCards and calendar events. It will have an authentication system which requires a password to discover PosterInBlue devices. After logging in Bluetooth communicator device starts a search for all BlueRadios devices in the range, more specifically our 'PosterInBlue' devices.

We think that there may be more than one PosterInBlue in an area such as subways and our admin can discover all devices in this area. Then he may click on one of the discovered PosterInBlues and our GUI will request the license key of this device. This is done for two reasons. First one is security concerns; since only the owner of this PosterInBlue can see the contents and configure it. Second one is we decide that while configuring a device nothing will be shown on LCD, in other words LCD is locked during configuration. When configuration is finished and load new configuration command is issued by the admin new configuration is loaded into our FPGA and our LCD starts to display new posters with new time intervals as specified in our configuration.

We may not provide multiple device architecture in our project since we have only one BlueRadios but we think security and handling of multiple devices is a very crucial issue if we can extend this project for commercial use in the future.

After selecting and getting an authorization from a specific PosterInBlue admin can explore the contents of this device. For instance he may see 4 posters are there in this device. Then select one of them to explore its details and attributes. Every poster object should include an image file and it's attributes at least.

Apart from that some posters may contain vCards or calendar events in it. Clicking on a specific poster our admin will see two rollouts which are attributes of this poster. One of them is vCard and one of them is calendar event associated with this poster. These two may be configured before or not configured. Admin can reconfigure, add or delete them.

We will also provide the user the ability to delete existing poster or adding a new poster. Duration of the poster to be shown in LCD attributes of it and if it will have associated vCard or calendar event should be provided by admin when adding a new poster object.

After completing all configurations admin will press ‘load new configuration’ button and admin module completes its work here.

3 2. Configuration Module

Configuration module basically configures the overall behavior of the system. To some extend it is a circuit logic which resides on the FPGA, keeps the internal structure stable and consistent. Configuration module mostly maintains the poster data, ensures its coherence.

Besides, Configuration module is the central module which provides a data interface to other modules, synchronize and keep all modules consistent by providing necessary address and data information to them.

Since this module is going to be a circuit stored on the XESS board’s programmable array here we need to show how a circuit can be designed on a Spartan 3 FPGA. We will show this design and our pin assignments in our Hardware Design section.

To achieve logic design for the Spartan FPGA of our XESS board, XILINX currently provides the WebPACK tools. To prepare design we use the XILINX ISE 8.2i **[3]** tool.

3 3. HCI Module (RS232)

Bluetooth is not just a radio system but also a software specification to handle devices from different manufacturers. This specification defines header and packet formats, error checking and some other functions. Each layer in the stack is strictly defined and maintained by the Bluetooth Special Interest Group (SIG)[5]. The protocol stack consists of eight layers, *see figure 3*. For HCI[6] module we are not going to give detailed information of each layer and except from HCI layer we briefly mention some layers.

RFCOMM: Provides a RS232 interface
L2CAP: Multiplexes data from higher layers
Link Manager: Controls and manages links to other devices
Radio: The physical communication

HCI module will be the interface between the Bluetooth device and the host device running the software on which the rest of the stack will reside. In our case this host device will be our FPGA board.

Data and commands will be sent by using 4 types of packets: Command packets, event packets and ACL[7] packets. Despite not directly being a part of HCI module ACL is just a link type that HCI module will use to communicate with another device. HCI module will use command packets with which it can control the behavior of local and remote link managers. All results are returned in event packets which return replies and data. We will also use these packets to inform the host, FPGA board, when events occur in the Bluetooth like termination of a connection. When a connection established between HCI module and another device, an ACL link automatically set up and L2CAP on the broadcast module and the configuration module can use HCI for communication.

[5] Bluetooth Special Interest Group is a group of companies who promotes and defines Bluetooth specifications.The use of the Bluetooth is limited to the companies that joined the SIG

[6] Host Controller Interface

[7] Asynchronous Connection-Less and transport time insensitive data

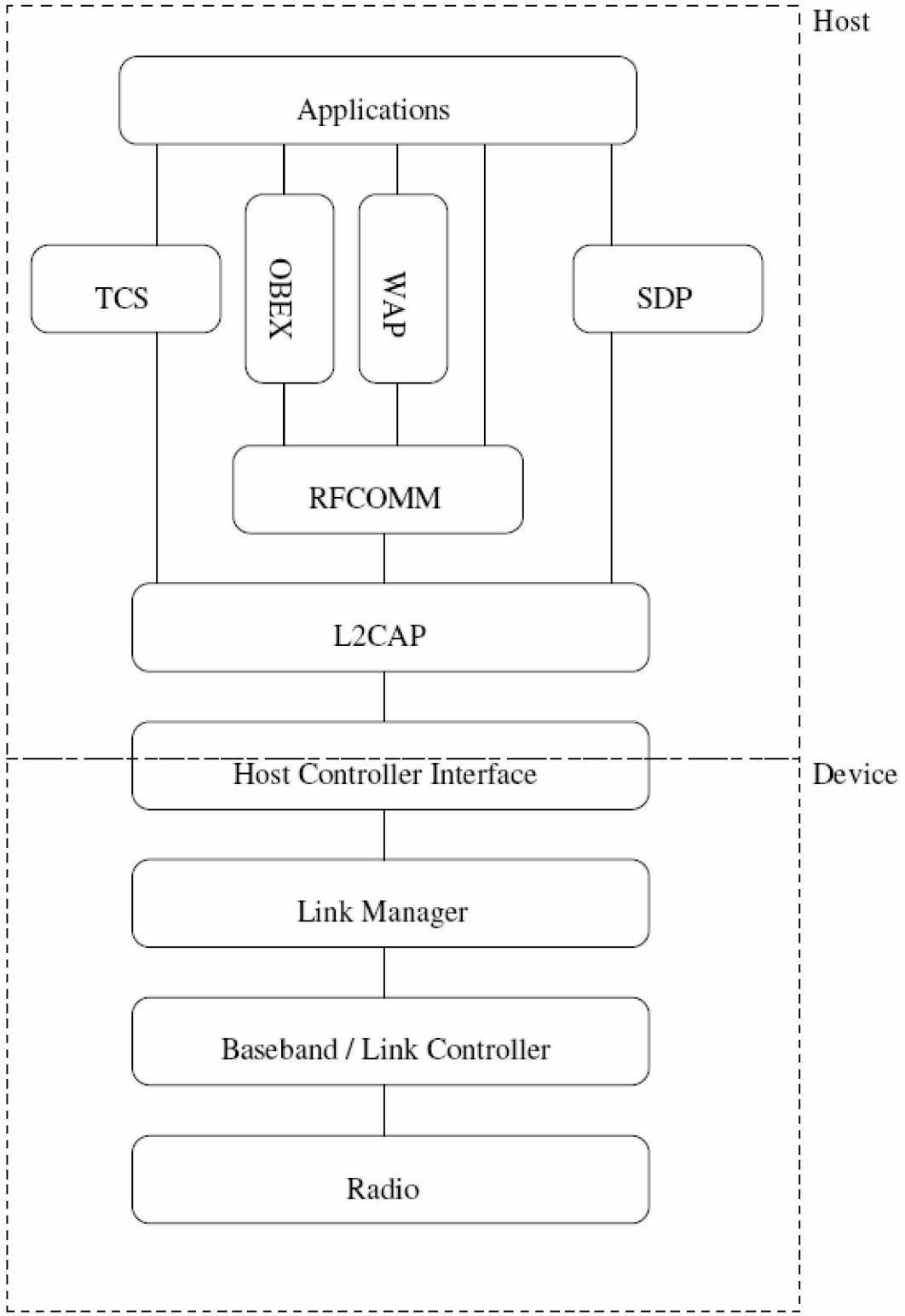


Figure 3: Bluetooth protocol stack

3 4. VGA Display Module

VGA Display module deals with the video subsystem of the XESS Board. The main responsibility of this module is to display the poster image on the screen.

Most of the concerns about display issues should be handled by Configuration module. To clarify, pixel data is fetched from correct SDRAM address by the Configuration module and supplied to VGA Display module. Consistency of pixel data transfer is ensured by the signals between SDRAM and Main Controller and between Main Controller and VGA Display.

3 5. Broadcast Module

This module concerns with providing various services to end users. In other words, this is the module in which communication with customers Bluetooth devices are held and interactively transferring the data on customers' demand.

Broadcast Module should communicate with HCI module which in turn communicates with Configuration Module and VGA Display Module. By this way we will maintain synchronization of posters in LCD and the posters we advertise.

To achieve our goal in this module we should use some of Bluetooth Profiles. Bluetooth Profiles are defined by Bluetooth specification as subsets. Profiles are a way to categorize different types of applications. This simplifies the development of Bluetooth products. For example in our project we are not dealing with audio and we will never have to know what 'Headset profile' is. Profiles make the communication between hardware and the Bluetooth software less complex since they are customized for different types of applications.

According to our previous research on this issue we decided that these are the profiles that we should use: File Transfer Profile (FTP), Generic Access Profile (GAP), Object

Exchange (OBEX), Object Push Profile (OPP) and Synchronization Profile (SYNC)**.** Since our BlueRadios kit supports "SDP, SPP, DUN, LAP, GAP, RFCOMM, L2CAP, Headset, Audio Gateway, FTP Client, OBEX, OPP - Object Push/Pull" we have no concerns on reliability of these profiles.
Below is a short description of each profile and how we will make use of them in our design:

OBEX is a transfer protocol that defines data objects and a communication protocol two devices can use to exchange those objects. OBEX enables applications to work over the *Bluetooth* protocol stack. For *Bluetooth* enabled devices, only connection-oriented OBEX is supported. Three application profiles have been developed using OBEX which include **FTP**, **OPP** and **SYNC**. We will use all these three application profiles.

FTP defines how folders and files on a server device can be browsed by a client device. **FTP** will be useful for us in sending presentations and selection of posters since it provides getting folder listings, changing to different folders and getting files.

OPP focuses on a narrow range of object formats to maximize interoperability. The most common acceptable format is the vCard and since our project scope includes sending vCards we should use this profile.

The **SYNC** profile is used in conjunction with **GOEP** to enable synchronization of PIM[8] items such as calendar and address information. A common application of this profile is the exchange of data between a PDA and computer. This will be useful for us to have synchronization in sending calendar events and personal information to different types of receiver devices.

[8] A **personal information manager** (**PIM**) is a type of application software that functions as a personal organizer.

GAP ensures the interoperability of any two Bluetooth devices, regardless of manufacturer and application. Bluetooth enabled devices not conforming to any other Bluetooth profile must conform to GAP to ensure basic interoperability and co-existence. It means we should use it in our project since all devices conform to at least GAP. We will use this since our aim is to provide a generic application, independent from brand, model and manufacturer.

3 6. Module Interactions

We will explain the interactions between modules from the diagram below, *see figure 4.*

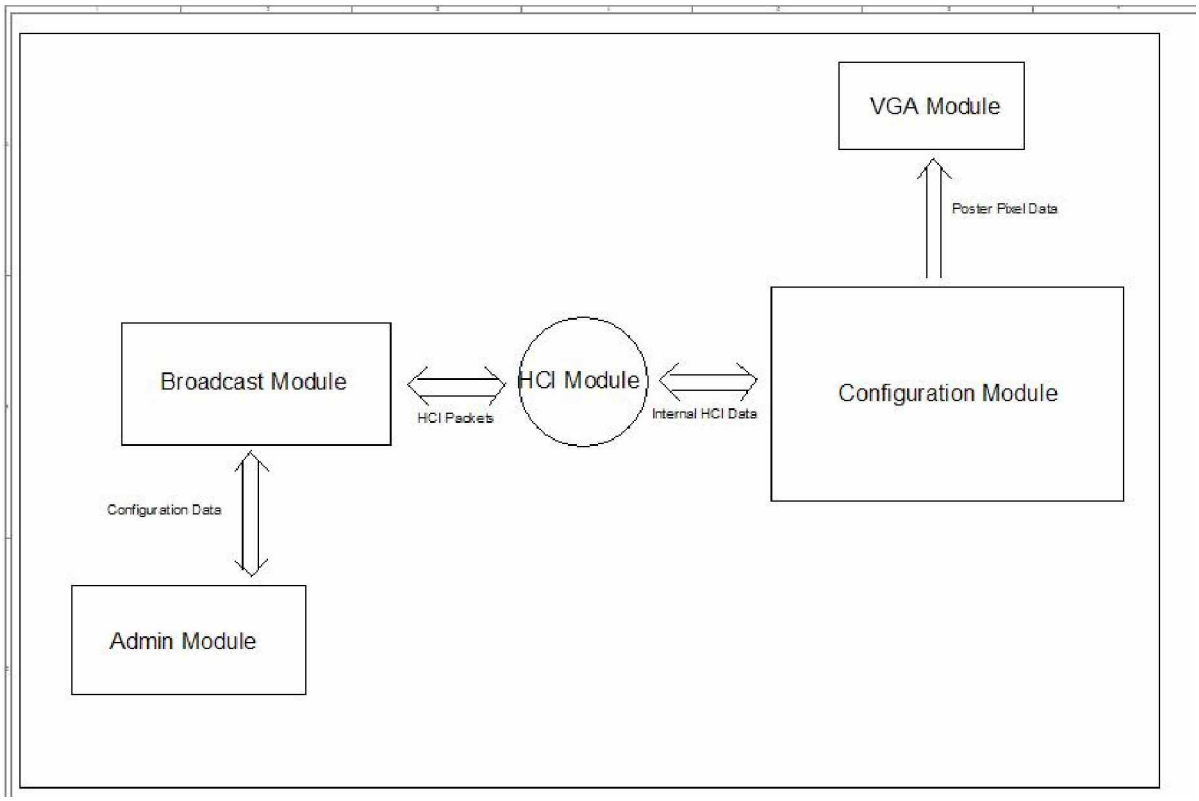


Figure 4: Module Interaction Diagram

Making PIB configuration (Configuration Data):

Configuration module maintains the current PIB settings and posts these settings to the Admin module via the virtual connection supplied by the HCI module. Admin module converts these settings into a representative format. With the help of the graphical user interface (GUI) a supposed administrator makes a new configuration, Bluetooth software on the admin module post the new configuration over the HCI module. Configuration module now gets the configuration over the HCI module and reconstructs the internal configuration.

Displaying posters (Pixel Data):

Admin module is responsible to prepare the poster in a specific format. It also needs to prepare the required software stack to communicate with the HCI module. HCI module is the interface that Admin and Configuration modules communicate on. Configuration module also prepares the stack, forms the internal representation of the poster data coming from the HCI module. Then it posts poster pixel data to VGA module using VGA Port. Now VGA module can handle display operation of images.

Posting poster data to 3rd party users (Broadcasting):

End users' Bluetooth devices communicate with HCI module via Bluetooth. As usual HCI is the interface and Broadcast module use this interface to communicate with the end users. Broadcast module advertises the poster, vCard and Calendar services on this virtual connection with the end users.

4 Detailed Hardware Design

Here we will basicly explain the logic designs and some hardware considerations of our system.
Firstly, we can think about the responsibilities of system on the hardware side. These respnsibilities are

- Send necessary RGB and vertical-horizontal synchronization signals over the VGA port on the hardware
- Store, read and write system data (poster attached data, poster image data or system configuration etc) from the external flash memory.
- Send and get HCI packets over the transmit and receive pins of the RS 232 DB9 of the bluetooth board.
- Store, read and write firsthand data, for instance poster pixel bits, on the SDRAM.

To achieve all these tasks we must design a logic circuit on the FPGA and this circuit should provide the necessary interfaces to all devices mentioned above. To some sense this design can be considered as a micro kernel –not ofcourse the same meaning, resides over the hardware. We call this circuit design “PIBCore”.

For schematic of PIBCore, *see figure 5*.

Supposed pin assignments of PIBCore of our XSA-3S1000 board can be seen in, *figure 6*.

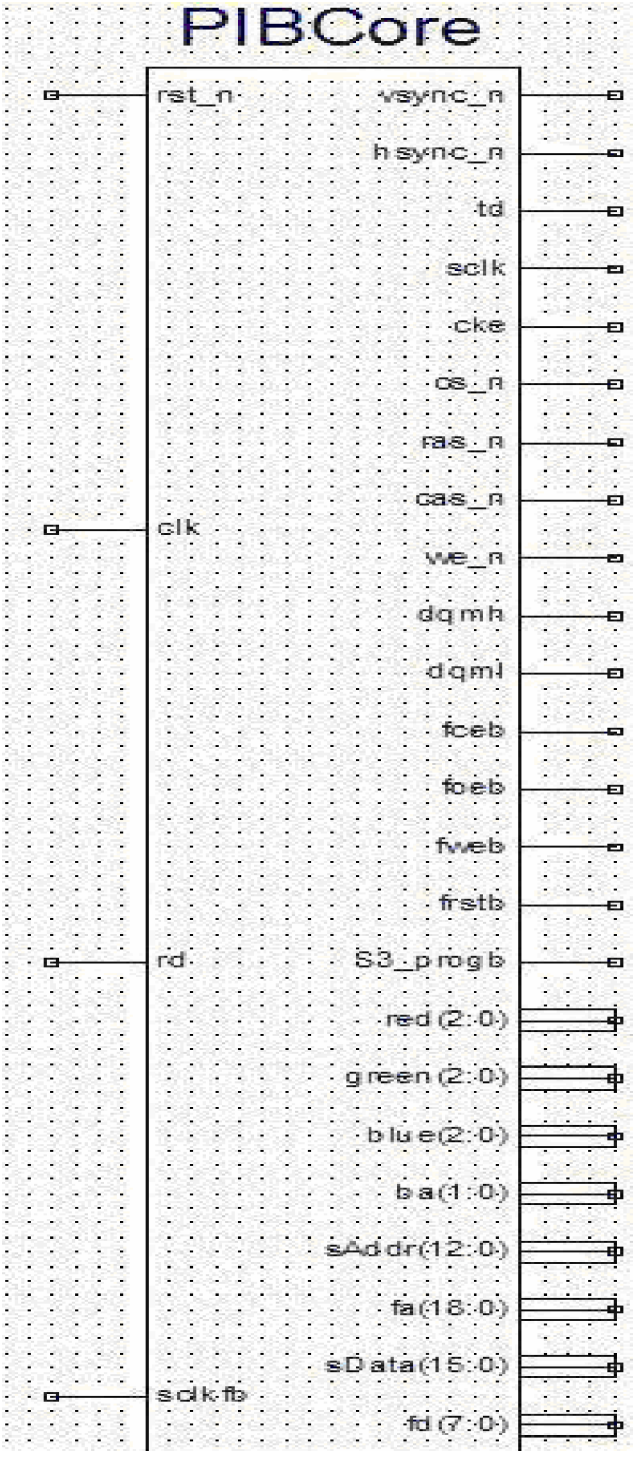


Figure 5: Schematic of PIBCore

I/O Name	I/O Direction	Loc	Bank
ba<0>	Output	A7	BANK0
ba<1>	Output	C7	BANK0
blue<0>	Output	C9	BANK1
blue<1>	Output	E7	BANK0
blue<2>	Output	D5	BANK0
cas_n	Output	A10	BANK1
cke	Output	D7	BANK0
clk	Input	T9	BANK4
cs_n	Output	B8	BANK0
dqmh	Output	D9	BANK1
dqml	Output	C10	BANK1
fa<0>	Output	N5	BANK5
fa<1>	Output	K14	BANK3
fa<2>	Output	K13	BANK3
fa<3>	Output	K12	BANK3
fa<4>	Output	L14	BANK3
fa<5>	Output	M16	BANK3
fa<6>	Output	L13	BANK3
fa<7>	Output	N16	BANK3
fa<8>	Output	N14	BANK3
fa<9>	Output	P15	BANK3
fa<10>	Output	R16	BANK3
fa<11>	Output	P14	BANK3
fa<12>	Output	P13	BANK4
fa<13>	Output	N12	BANK4
fa<14>	Output	T14	BANK4
fa<15>	Output	R13	BANK4
fa<16>	Output	N10	BANK4
fa<17>	Output	M14	BANK3
fa<18>	Output	K3	BANK6
fceb	Output	R4	BANK5
fd<0>	InOut	M11	BANK4
fd<1>	InOut	N11	BANK4
fd<2>	InOut	P10	BANK4
fd<3>	InOut	R10	BANK4
fd<4>	InOut	T7	BANK5
fd<5>	InOut	R7	BANK5
fd<6>	InOut	N6	BANK5
fd<7>	InOut	M6	BANK5
foeb	Output	P5	BANK5
frstb	Output	P16	BANK3
fweb	Output	M13	BANK3
green<0>	Output	A8	BANK0
green<1>	Output	A5	BANK0
green<2>	Output	C3	BANK7
hsync_n	Output	B7	BANK0
ras_n	Output	A9	BANK1
rd	Input	G5	BANK7
red<0>	Output	C8	BANK0
red<1>	Output	D6	BANK0
red<2>	Output	B1	BANK7
rst_n	Input	E11	BANK1
sAddr<0>	Output	B5	BANK0
sAddr<1>	Output	A4	BANK0
sAddr<2>	Output	B4	BANK0
sAddr<3>	Output	E6	BANK0
sAddr<4>	Output	E3	BANK7
sAddr<5>	Output	C1	BANK7
sAddr<6>	Output	E4	BANK7
sAddr<7>	Output	D3	BANK7
sAddr<8>	Output	C2	BANK7
sAddr<9>	Output	A3	BANK0
sAddr<10>	Output	B6	BANK0
sAddr<11>	Output	C5	BANK0
sAddr<12>	Output	C6	BANK0
sclk	Output	E10	BANK1
sclkfb	Input	N8	BANK5
sData<0>	InOut	C15	BANK2
sData<1>	InOut	D12	BANK1
sData<2>	InOut	A14	BANK1
sData<3>	InOut	B13	BANK1
sData<4>	InOut	D11	BANK1
sData<5>	InOut	A12	BANK1
sData<6>	InOut	C11	BANK1
sData<7>	InOut	D10	BANK1
sData<8>	InOut	B11	BANK1
sData<9>	InOut	B12	BANK1
sData<10>	InOut	C12	BANK1
sData<11>	InOut	B14	BANK1
sData<12>	InOut	D14	BANK2
sData<13>	InOut	C16	BANK2
sData<14>	InOut	F12	BANK2
sData<15>	InOut	F13	BANK2
S3_progb	Output	F5	BANK7
td	Output	J2	BANK6
vsync_n	Output	D8	BANK0
we_n	Output	B10	BANK1

Figure 6: PIN Assignments of PIBCore

4 1. Interface Explanation

rst_n: reset line
clk: master clock of the board. Our XSA-3S1000 boards frequency is 100000 MHZ

VGA connection
vsync_n: VGA vertical synchronization signal
hsync_n: VGA horizantal synchronization signal
red: Red output signals
green: Green output signals
blue: Blue output signals

RS232 connection
td: RS232 transmitter line
rd: RS232 receiver line

SDRAM connection
sclkfb: clock from SDRAM after PCB delays
sclk: SDRAM clock sync'ed to master clock
cke: clock-enable to SDRAM
cs_n: chip-select to SDRAM
ras_n: SDRAM row address strobe
cas_n: SDRAM column address strobe
we_n: SDRAM write enable
ba: SDRAM bank address bits
sAddr: SDRAM row/column address
sData: SDRAM in/out databus
dqmh: high databits I/O mask
dqml: low databits I/O mask

Flash connection
fd: data bus to flash
fa: address bus to flash
fceb: chip-enable for flash
foeb: output-enable for flash
fweb: write-enable for flash
frstb: reset for flash

We think that FPGAs parallel operation capability is really sufficient for this design. 32 MB XSA SDRAM is also adequate for the firsthand data. However 2MB XSA Flash memory is not sufficient to store poster data, program data and system configuration. Also RS232 connection also makes us to think about another board we need to design for these devices.

Of course this board is not a full featured board like XST 3.0 of XESS. We name it ‘Support Board’. This circuitry has a limited extend of support on which there is only an integrated 256 MB flash memory and an RS232 interface like the one on XST 3.0 board.

4 2. Design Details of PIBCore

PIBCore is basically composed of the device controllers and the main controller logic designs.

4 2 1. SDRAM Controller

This controller deals with the XSA SDRAM. Generally controller has address, data input/output and w/r connections with the host controller logic (in this case Main Controller Logic) and real hardware pin assigments for SDRAM’s address, data input/output signals. SDRAM controller accepts simple read and write requests on the Main Controller side and generates the timed waveforms required to perform these operations on the SDRAM. The controller also manages the refresh operations needed to keep the SDRAM data valid, and will place the SDRAM in a self-refresh mode so data is retained.

4 2 2. Flash Controller

This controller deals with the flash memory on the Support Board. Generally controller has address, data input/output and w/r connections with the host controller logic (in this case Main Controller Logic) and real hardware pin assigments for flash memory's address, data input/output signals. Flash controller accepts simple read and write requests on the Main Controller side and generates waveforms required to perform these operations on the flash memory.

4 2 3. VGA Controller

This controller is the VGA generator of PIBCore. Generally controller collects pixel data from the Main controller in a pixel buffer and generates vsync-hsync signals and red-green-blue output signals on VGA OUT.

4 2 4. RS232 Controller

This controller deals with the RS232 low speed serial communication with the Bluetooth kit. Generally controller has transmitter and receiver data lines for the DB9 interface and HCI data bus for the Main controller. Main capability of this controller is, by the help of the state machine circuit design implemented inside the controller, to generate transmit signals from HCI data received from main controller and generate HCI data from the received data signals.

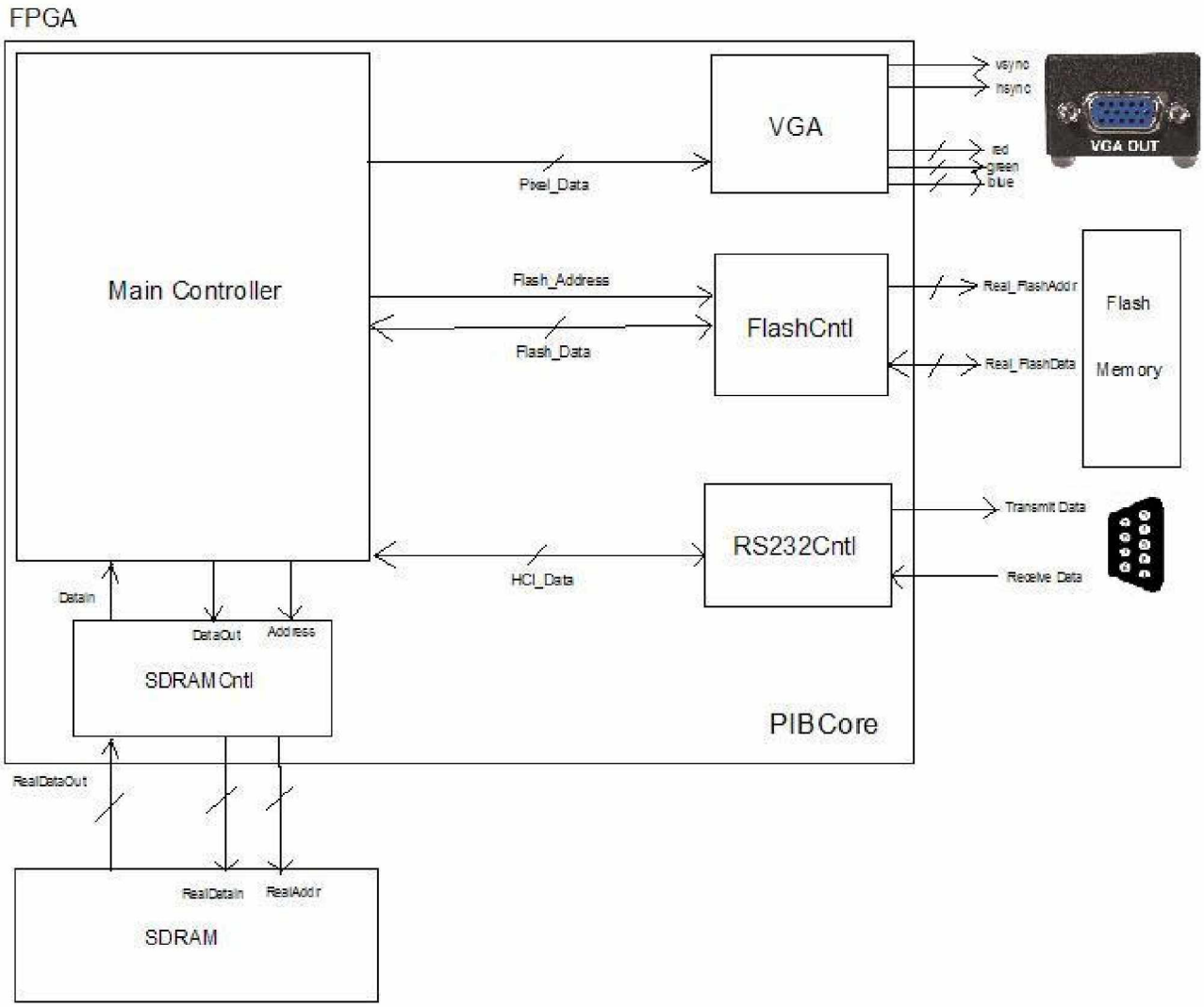


Figure 7: Block Diagram of PIBCore

4 3. Main Controller

Basic task of this controller is to generate corresponding address and data signals for the other controllers, receive data from these controllers, process and store these data. These signals are pixel data from VGACntl, Flash Address and Flash data from/to FlashCntl, SDRAM Address and Data from/to XSA SDRAM and HCI data from/to RS232Cntl.

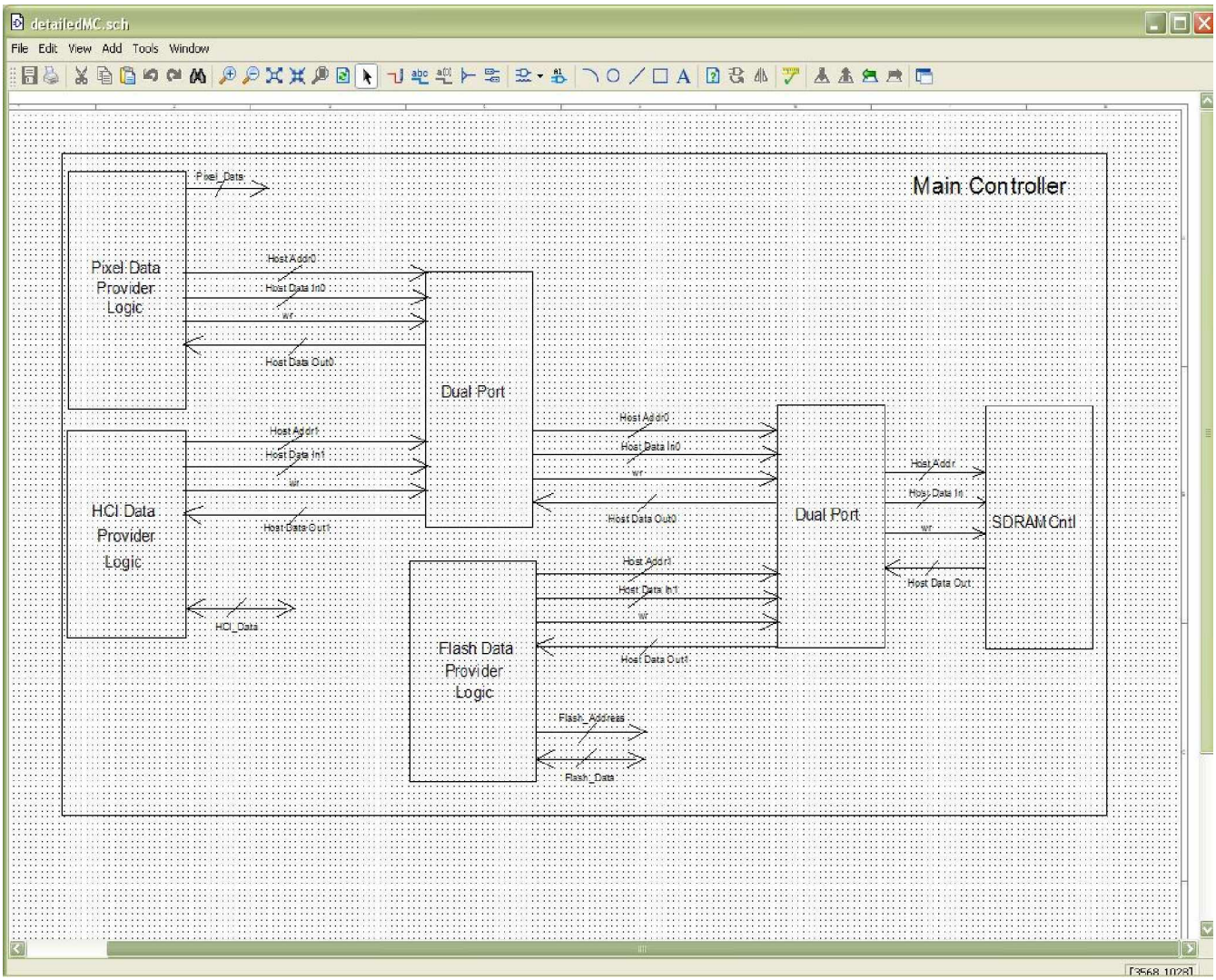


Figure 8: Block Diagram Of Main Controller

Pixel Data Provider, HCI Data Provider and Flash Data Provider are the main data and address processor and generator logic circuits of their corresponding controllers' logics. General behaviour of these logic designs is the decision of the SDRAM addresses from/to data need to pass, process the data and generate necessary controller data (this data is for example Flash Address and Flash Data to Flash Controller). In the above block diagram Dual ports provide the SDRAM controller interfaces for other logic circuits each of which can see the SDRAM controller as a dedicated controller for themselves.

5 System Design

5 1. GUI

We have a GUI design for admin user to do the necessary operations such as loading posters or configuring the poster settings. For this phase we have designed some of them.

Admin tool has an authentication system. He can authenticate the system by typing his password. After a valid authentication administrator tool searches, *see figure 9*, the near PIBs. It lists them on the “poster in blue configuration console”. The list appears on the left side of the console. The user admin can refresh, select and remove PIB or clear the PIB list by using the buttons on the left below. Also he can load new configurations or reset the configurations of the selected PIB device.

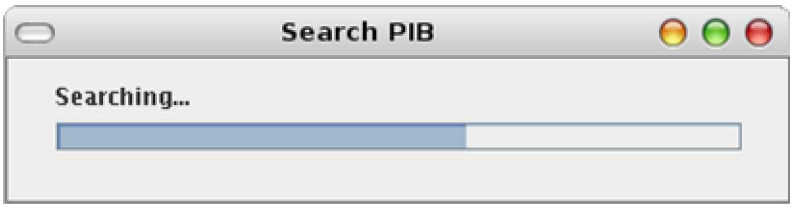


Figure 9: Searching PIBs in the range

From the GUI, *see figure10*, admin can configure the VCard and calendar properties of the supposed poster. Admin can enter the information such as name, phone number, e-mail, web address, title, company name, address, birthday and general info for VCards. All of these fields do not need to be filled and of course it is optional to prepare a VCard for a poster. Corresponding textboxes and check box on the “VCard Properties” pane implements this capability. Also to configure the calendar properties there are name, subject, start time, end time, start date, end date, location areas.

As in the VCard field, it is optional to fill the text box areas and to prepare calendar properties. After the admin enters the information he can use the Apply and Reset buttons to save them.

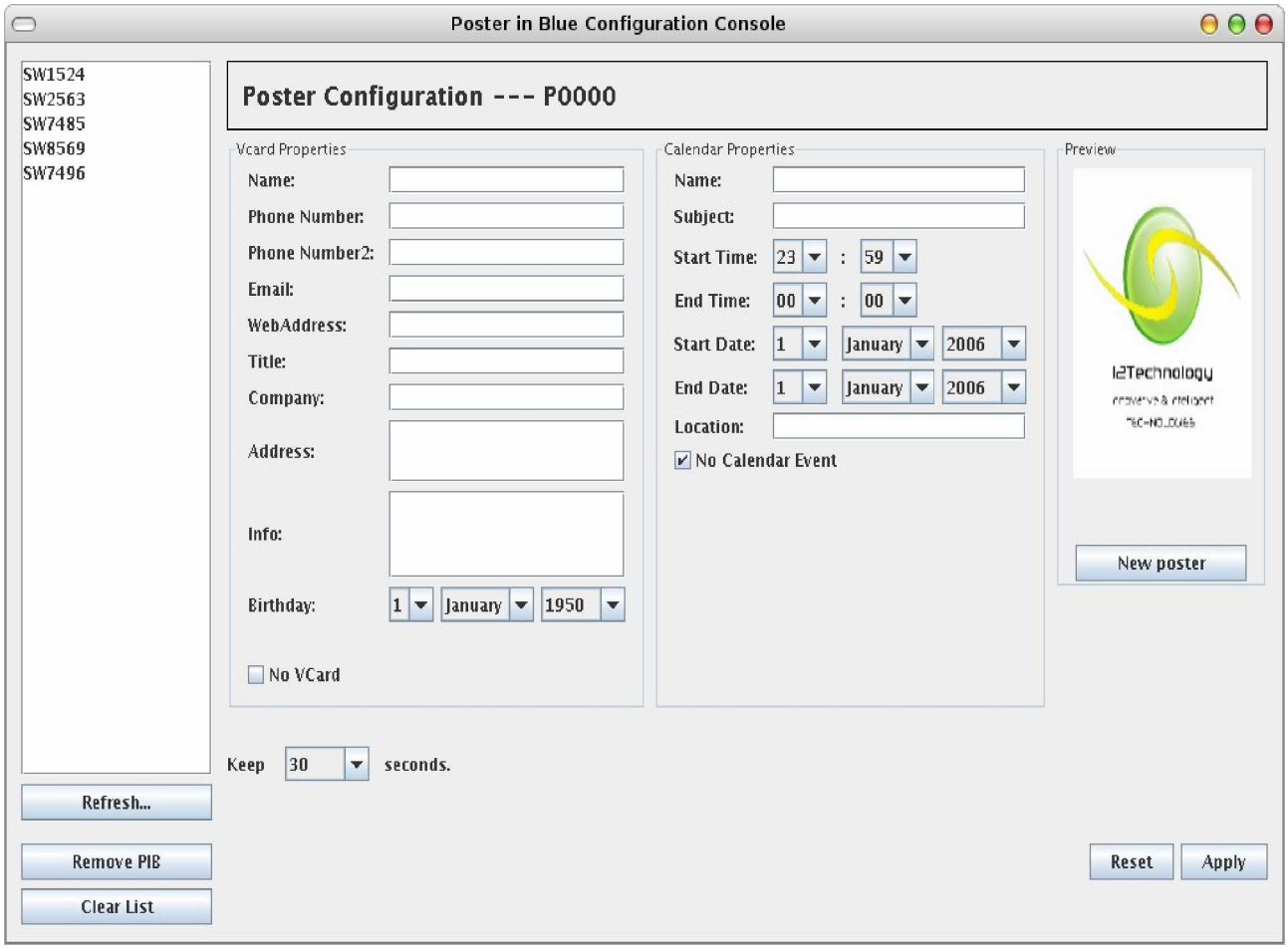


Figure 10: GUI for Poster Configuration

Also admin can preview the poster image on the right side of the GUI, *see figure 10*. He can add new poster by navigating administrator's local computer file system. From the dropdown menu on the below middle-left side on the GUI, he can decide on how many seconds the poster will stay on the VGA screen. After all the poster configurations, admin can load these configurations to the selected PIB by clicking the "Load New Configuration" button.

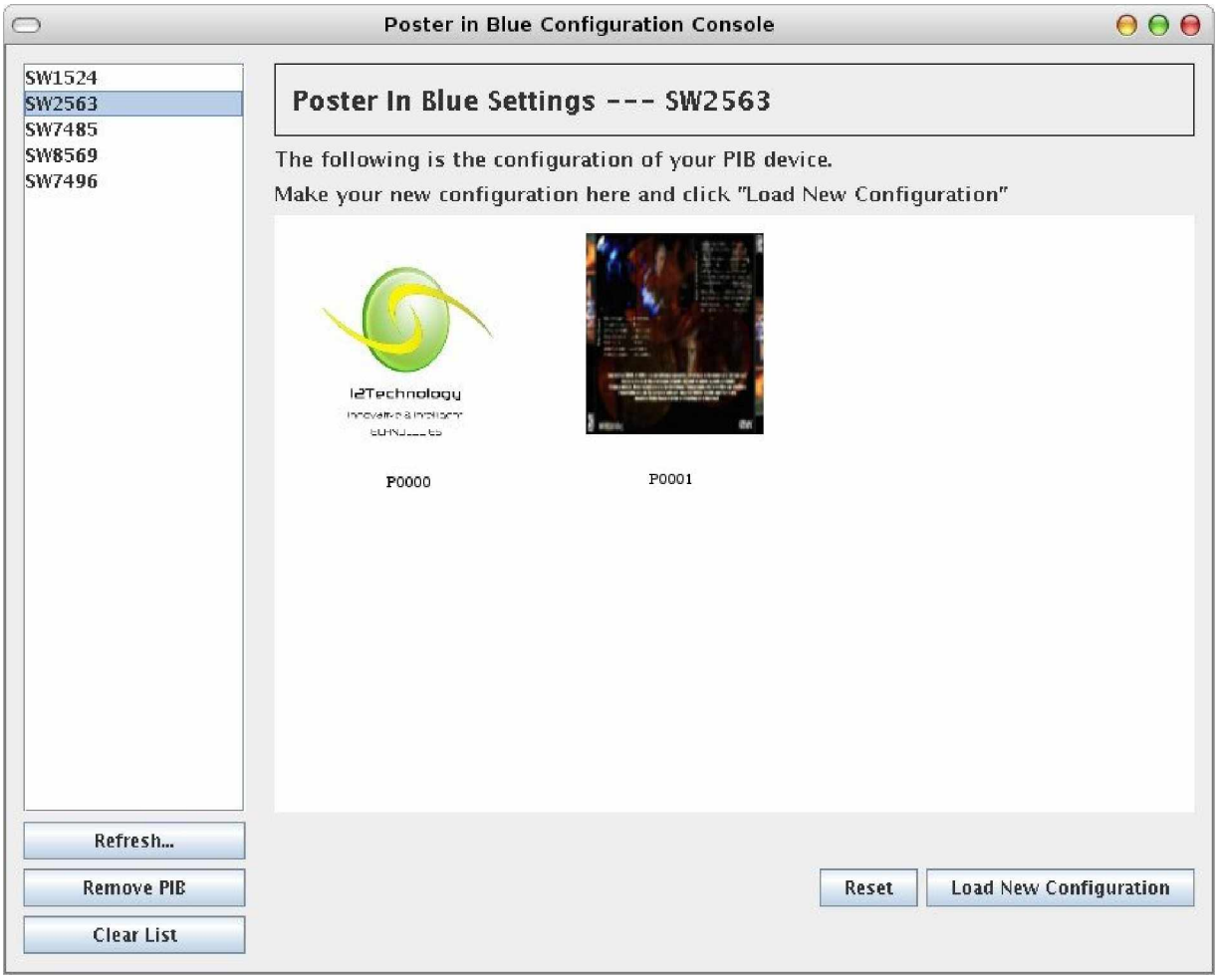


Figure11: GUI for Poster Selection

5 2. Static Modeling

Although we will not use an object oriented language and we do not have actual classes, we decided that decomposing our design into classes will be much easier to understand. Our classes are explained below.

We have revised our class diagram since we have found which classes we will use in Bluetooth part. According to this we now give explanation of the methods in Bluetooth client and server parts.

BluetoothClient

Attributes:

private javax.bluetooth.UUID DATA_SERVER_UUID

Describes the server.

private int READY = 0

Shows if the engine is ready to work.

private int DEVICE_SEARCH = 1

Shows if the engine is searching bluetooth devices.

private int SERVICE_SEARCH = 2

Shows if the engine is searching bluetooth services.

private int state = READY

Keeps the state of the engine

private Thread processorThread

Proccess the search/download requests.

private Vector devices

Collects the remote devices found during a search.

private Vector records

Collects the services found during a search.

private String dataNameToLoad

Keeps the data name to be load.

private boolean isDownloadCanceled

Informs the thread when the download is canceled

Methods:

Constructor: BluetoothClient()

Initializes processorThread, and starts the thread.

public void run() :

This method processes the search/download requests.

If bluetooth device is ready it initializes client bluetooth device's attributes and calls processDataSearchDownload function.

public void deviceDiscovered(javax.bluetooth.RemoteDevice btDevice)

Adds the discovered remote device to the device vector of that class.

public void servicesDiscovered(int ID, javax.bluetooth.ServiceRecord servRecord)

Adds the found service to the record vector of that class

public void requestSearch(int deviceOrService)

Sets the value of the state attribute of the class to DEVICE_SEARCH or to SERVICE_SEARCH

public void stopSearch()

Stops the devices/services search

public void requestLoad(String name)

Sets the value of the state attribute of the class to LOAD

public void stopLoad()

Stops loading the data and sets isDownloadCanceled to false

public void processDataSearchDownload()

If the value of the state attribute is LOAD and if there exist a service and a device discovered as server, then it shows data name to user and sets dataNameToLoad. Then it sets DATA_SERVER_UUID and calls loadData function.

public boolean searchServices()

If the value of the state attribute is SERVICE_SEARCH, then it first removes all elements of the record vector (services discovered before the search) and then adds the services the bluetooth client class finds at that time.

public byte[] loadData()

If the isDownloadCanceled value is not true, then it connects to the server through DATA_SERVER_UUID and loads the dataNameToLoad.

BluetoothServer

Attributes:

private javax.bluetooth.UUID DATA_SERVER_UUID

Describes this server.

private javax.microedition.io.StreamConnectionNotifier notifier

Notifier accepts new connections.

private javax.bluetooth.ServiceRecord record

Holds information about this server.

private Thread client

Accepts clients.

private Vector dataElements

Vector of data elements to be published.

private boolean isReady

Holds the information if the bluetooth is ready.

Methods:

Constructor: BluetoothServer()

Initializes client thread, and starts the thread.

public void run()

This method accepts new clients and sends him the requested data.

If bluetooth device is ready it initializes the notifier. And processor

public boolean changeData(String name)

Updates the data element which is in dataElements vector list. If this data can not be found in the dataElements attribute of this class the method returns false.

public void processConnection(javax.microedition.io.StreamConnection conn)

Reads the data name from the conn and sends this data by calling sendData function.

public void sendData (byte[] data)

Sends the data.

Since our project is a hardware project methods our classes about FPGA part do not have corresponding functions in software. We program our board using VHDL which is not object-oriented and we only define methods for understanding of the system.

Admin: This class holds the information about the administrator.

id:	Administrator's login id for system
password :	Administrator's login password for system
getCurrentConfiguration :	Administrator can learn the current configuration of system
sendNewConfiguration :	Administrator can change the configuration
startApplication :	Administrator can start displaying poster and sending data
stopApplication :	Administrator can stop displaying poster and sending data
login :	Administrator can login to system with his id and password

Configuration : This class holds the configuration of the displayed posters.

presentTime:	This array is used for how long each poster is displayed
presentQueue:	This array is used for when each poster is displayed
posterIds:	This array is used for which posters are being displayed

FPGABoard: This class is our XSA-3S1000 board's programmable part.

display: FPGABoard can display posters using current configuration

preparePosterDataToBeSent: When a new configuration is loaded new poster and its data is prepared

prepareConfigurationDataToBeSent: When administrator requests current configuration, FPGA calls this function

processConfigurationData: When new configuration is loaded FPGA processes this configuration to decide when to show which poster

setConfigurationData: When Admin changes the current configuration, new configuration is set

sendConfiguration: When administrator requests current configuration, FPGA sends it after preparing configuration data to be sent

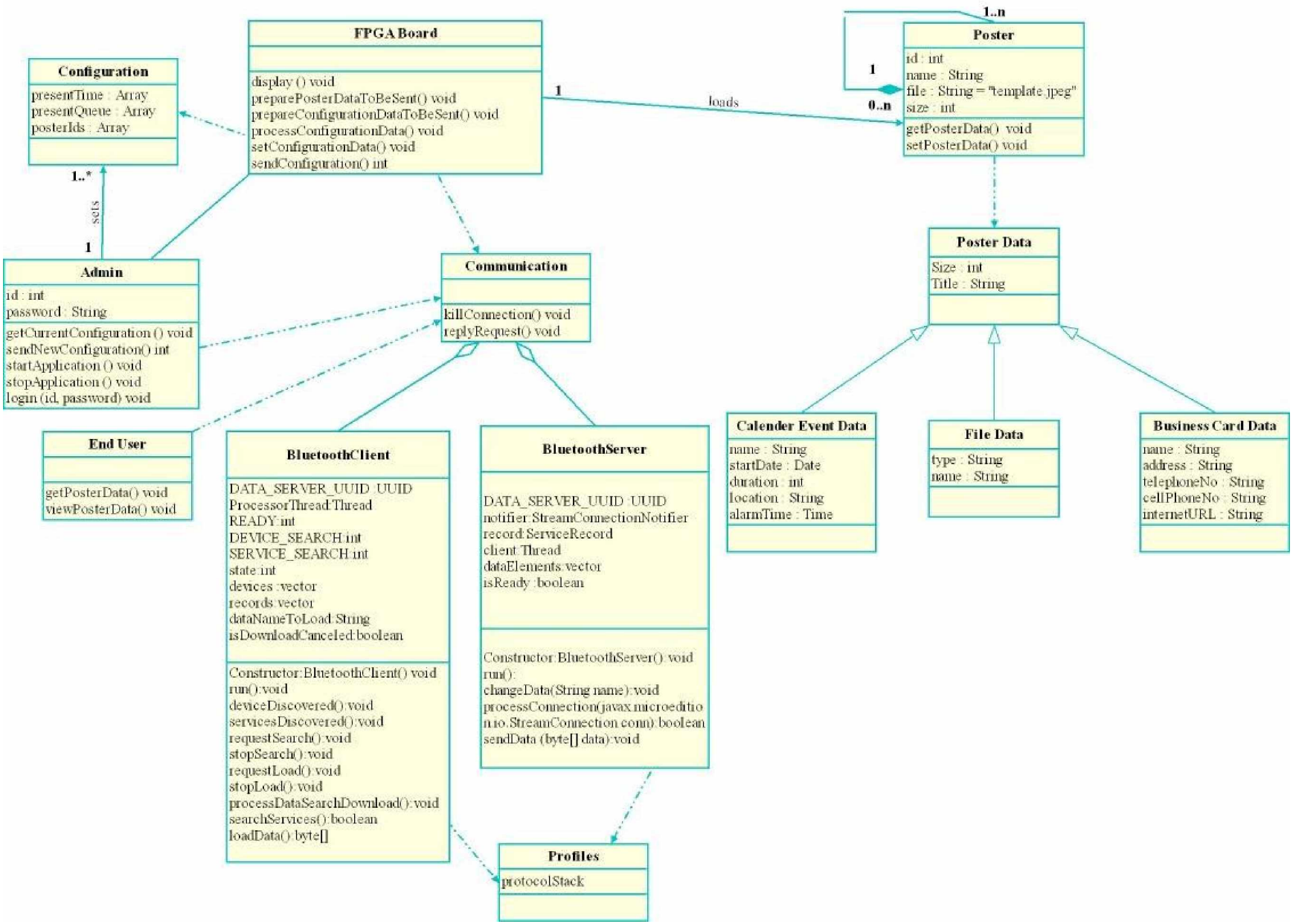
Communication: This class supplies communication between FPGA Board, Bluetooth board , Administrator and end users. Bluetooth Receiver Controller and Bluetooth Sender Controller are parts of Communication class and they use Profiles. Bluetooth Sender Controller class is responsible for sending data to End Users and Bluetooth Receiver Controller is responsible for sending data from Administrator to the board when he wants to update the configuration.

End User: End users get the data transmitted by Poster in Blue. They can getPosterData or viewPosterData by their Bluetooth devices.

Poster: One or more posters forms a poster (since more than one poster can be shown). Poster uses poster data.
id: Size of the poster
name: Name of the poster
file: Name of the file the poster is using for display
size: Size of the poster
getPosterData : Gets the data that will be sent while this poster is being displayed
setPosterData: Sets the data that will be sent while this poster is being displayed

Poster Data: Poster data can be in 3 forms. It can be a file, a calendar event or a business card.

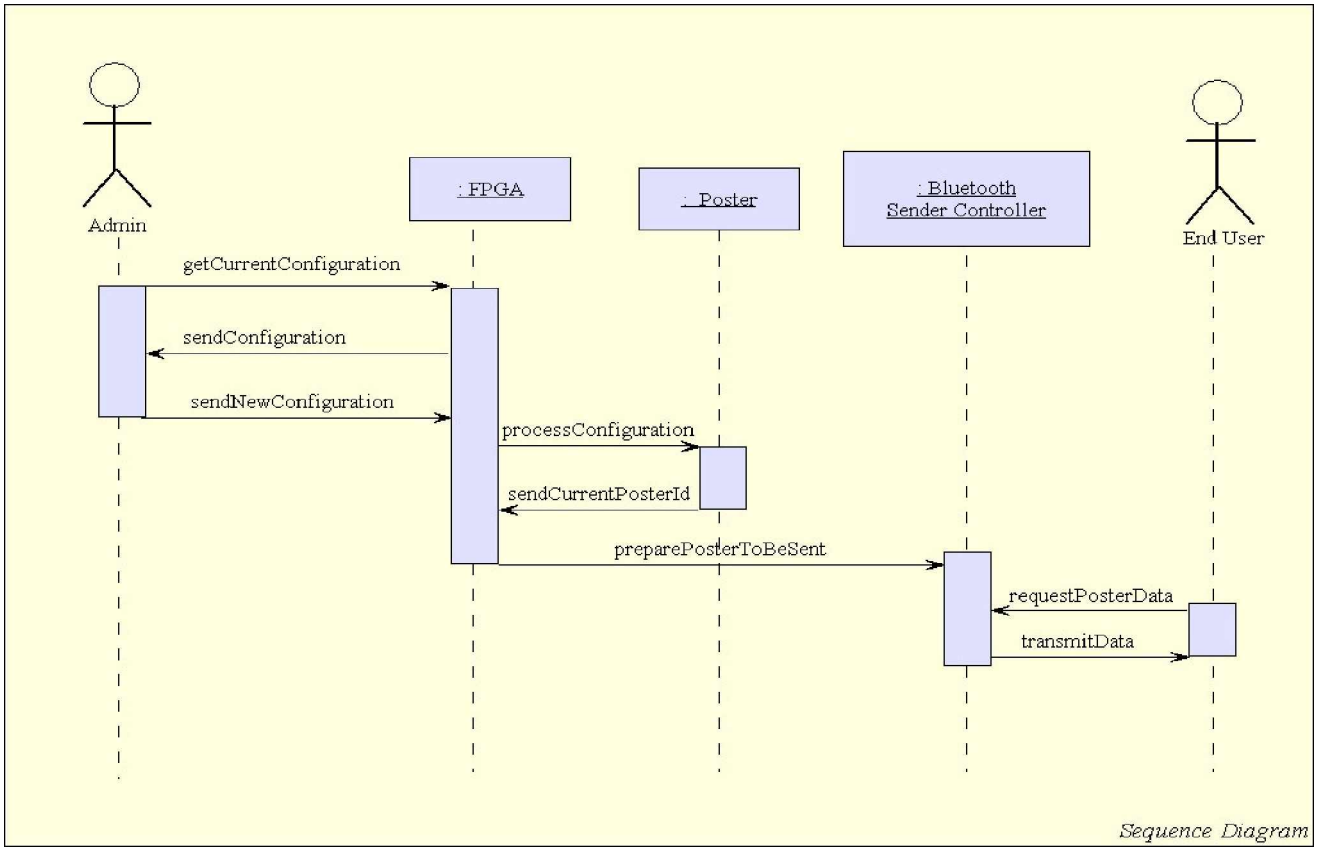
Below is our Revised Class Diagram:


FPGA Board
display () void
preparePosterDataToBeSent() void
prepareConfigurationDataToBeSent() void
processConfigurationData() void
setConfigurationData() void
sendConfiguration() int
1
loads
1..n
1
0..n
Poster
id : int
name : String
file : String = "template.jpeg"
size : int
getPosterData() void
setPosterData() void
Configuration
presentTime : Array
presentQueue : Array
posterIds : Array
1..*
sets
1
Admin
id : int
password : String
getCurrentConfiguration () void
sendNewConfiguration() int
startApplication () void
stopApplication () void
login (id, password) void
Communication
killConnection() void
replyRequest() void
Poster Data
Size : int
Title : String
End User
getPosterData() void
viewPosterData() void
BluetoothClient
DATA_SERVER_UUID :UUID
ProcessorThread:Thread
READY:int
DEVICE_SEARCH:int
SERVICE_SEARCH:int
state:int
devices :vector
records:vector
dataNameToLoad:String
isDownloadCanceled:boolean
Constructor:BluetoothClient() void
run():void
deviceDiscovered():void
servicesDiscovered():void
requestSearch():void
stopSearch():void
requestLoad():void
stopLoad():void
processDataSearchDownload():void
searchServices():boolean
loadData():byte[]
BluetoothServer
DATA_SERVER_UUID :UUID
notifier:StreamConnectionNotifier
record:ServiceRecord
client:Thread
dataElements:vector
isReady :boolean
Constructor:BluetoothServer() void
run():
changeData(String name):void
processConnection(javax.microedition.io.StreamConnection conn):boolean
sendData (byte[] data):void
Profiles
protocolStack
Calender Event Data
name : String
startDate : Date
duration : int
location : String
alarmTime : Time
File Data
type : String
name : String
Business Card Data
name : String
address : String
telephoneNo : String
cellPhoneNo : String
internetURL : String

5 3. Dynamic Modeling

To describe the dynamic model of our system we use sequence diagram.

First of all, admin requests to see the current configuration of the FPGA board and FPGA board sends the current configuration info. Having checked the current configuration, admin may like to set the new configuration and send it. After receiving the new configuration info, FPGA processes it and gets the current poster id. According to the new configuration, it prepares the new poster to be sent to bluetooth sender controller. After end user requests the poster data Bluetooth sender controller transmit the poster data to our end users Bluetooth device.



5 4. Behavioral Modeling

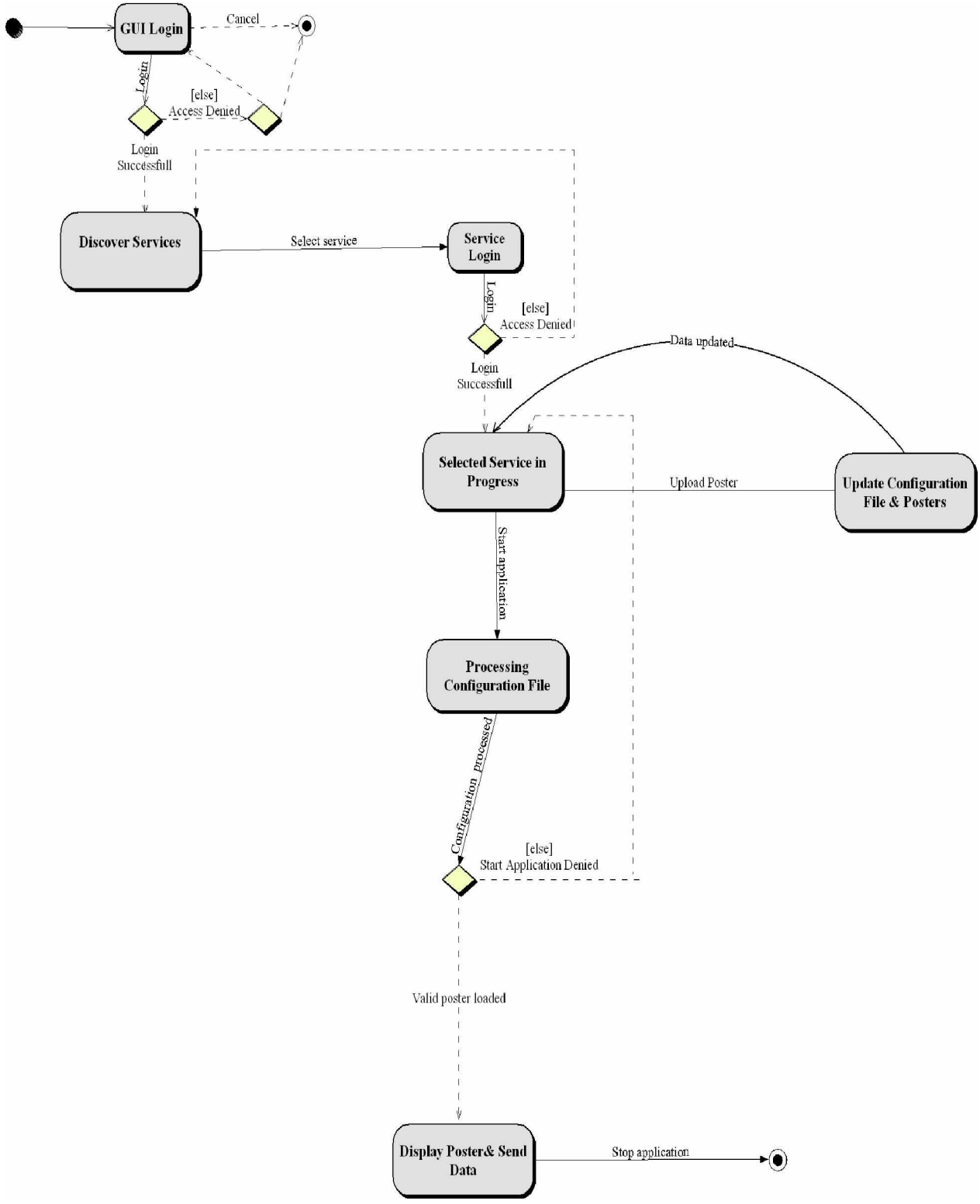
To describe the behavioral model of our system we use state transition diagram.

Since we have realized we should consider security again and provide a two level security system for handling multiple posters we revised our state diagram and reflected these changes to our state transition diagram accordingly.

In our revised state transition diagram first state is *GUI Login* for administrator. If not authenticated system goes back to login state. If password is verified and login is successful we go to *Discover Services* state in which our system searches and finds the available 'PosterInBlues' in the range. From these available PIBs our admin selects one of them which is owned by him. Since he owns this PIB he knows its password and he will be asked to enter this.

After getting authentication from this PIB he can fetch the configuration data, update it and upload to board. This new configuration is processed when the user starts the application, in other words when he clicks the 'Load new Configuration' button. According to result of the process user either gets an error message indicating the possible error of configuration information or poster data is successfully sent and displayed on LCD.

Below is our revised state transition diagram:


GUI Login
Cancel
Login
[else]
Access Denied
Login
Successfull
Discover Services
Select service
Service
Login
Login
[else]
Access Denied
Login
Successfull
Data updated
Selected Service in
Progress
Upload Poster
Update Configuration
File & Posters
Start application
Processing
Configuration File
Configuration processed
[else]
Start Application Denied
Valid poster loaded
Display Poster& Send
Data
Stop application

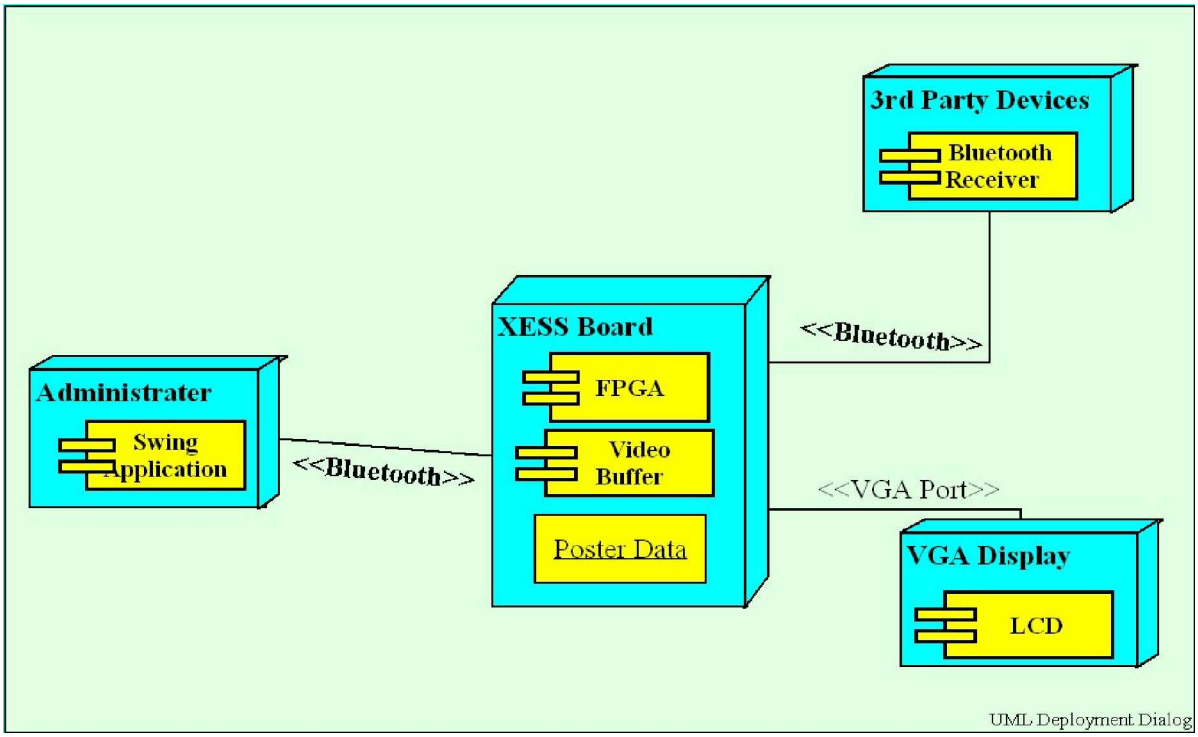
5 5. Deployment View

To illustrate the static deployment view of our run-time system we draw a deployment diagram. We have mainly 4 nodes in our diagram which are connected by communication associations.
The nodes are:
Administrator: It is computer of admin which configures our PIB through the graphical user interface. It has swing application as a component in it. This computer should have a JRE to run this swing application to be able to use GUI. It should also have a Bluetooth connection to be able to load configuration XESS board.

XESS Board: This is the main board of our system which has FPGA, Video Buffer as components and Poster Data as an object which is an entity at a particular point in time with a specific value and has different values over time. XESS Board communicates with all other nodes through Bluetooth or serial communication.

VGA Display: VGA display node communicates with our board to display images on a screen. The only component of VGA display is LCD (may call it a screen) in which the Poster Data is displayed.

3rd Party Devices: These are the devices such as cell phones or PDAs of the end users to which we will send Poster Data. They may have many components but the only component related to our application is Bluetooth Receiver which enables them to communicate with XESS Board through Bluetooth.

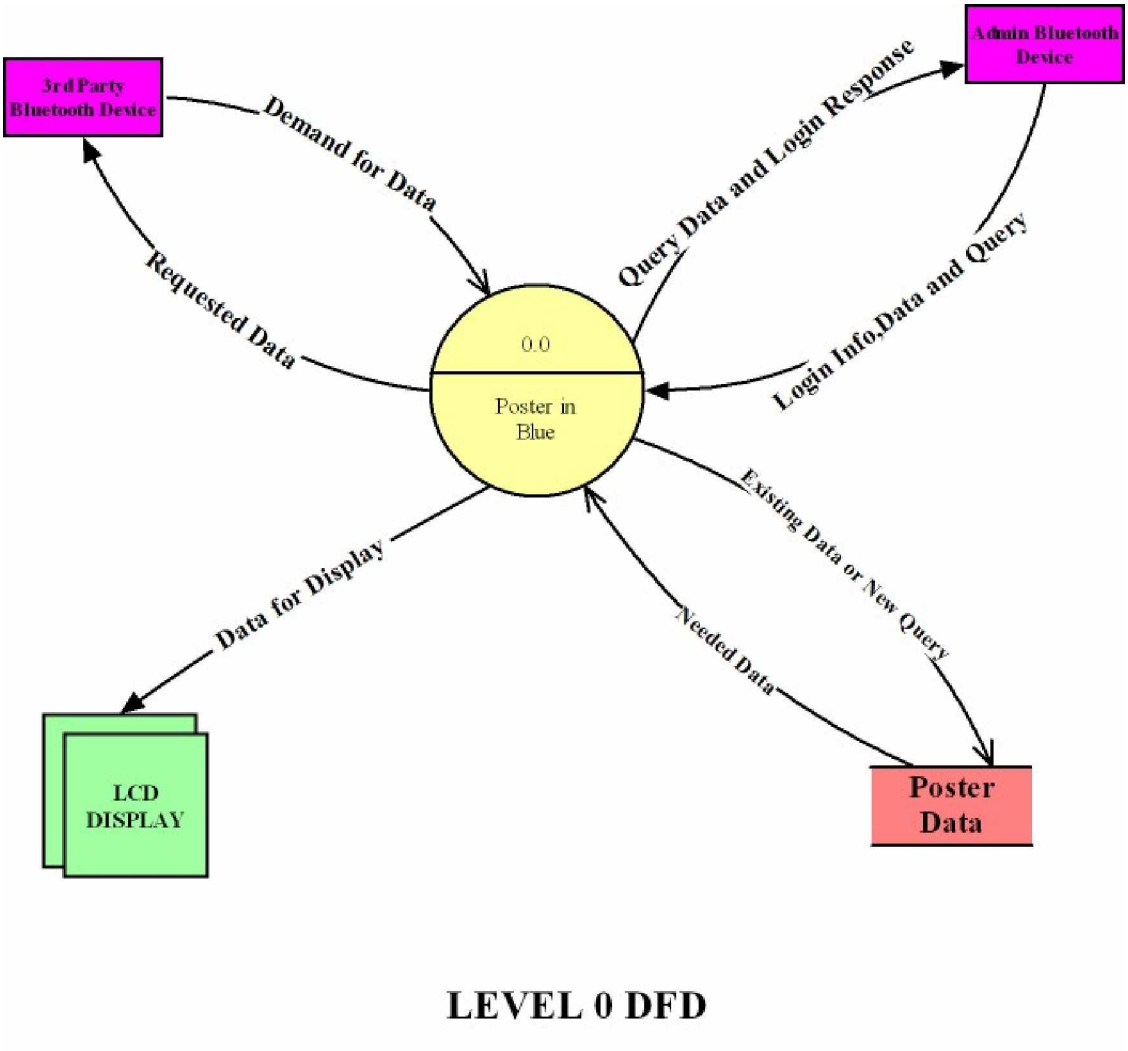

3rd Party Devices
Bluetooth Receiver
XESS Board
FPGA
Video Buffer
Poster Data
<<Bluetooth>>
Administrater
Swing Application
<<Bluetooth>>
<<VGA Port>>
VGA Display
LCD
UML Deployment Dialog

6 Data Modeling

For Data Dictionary of our Data Flow Diagrams see **APPENDIX C**

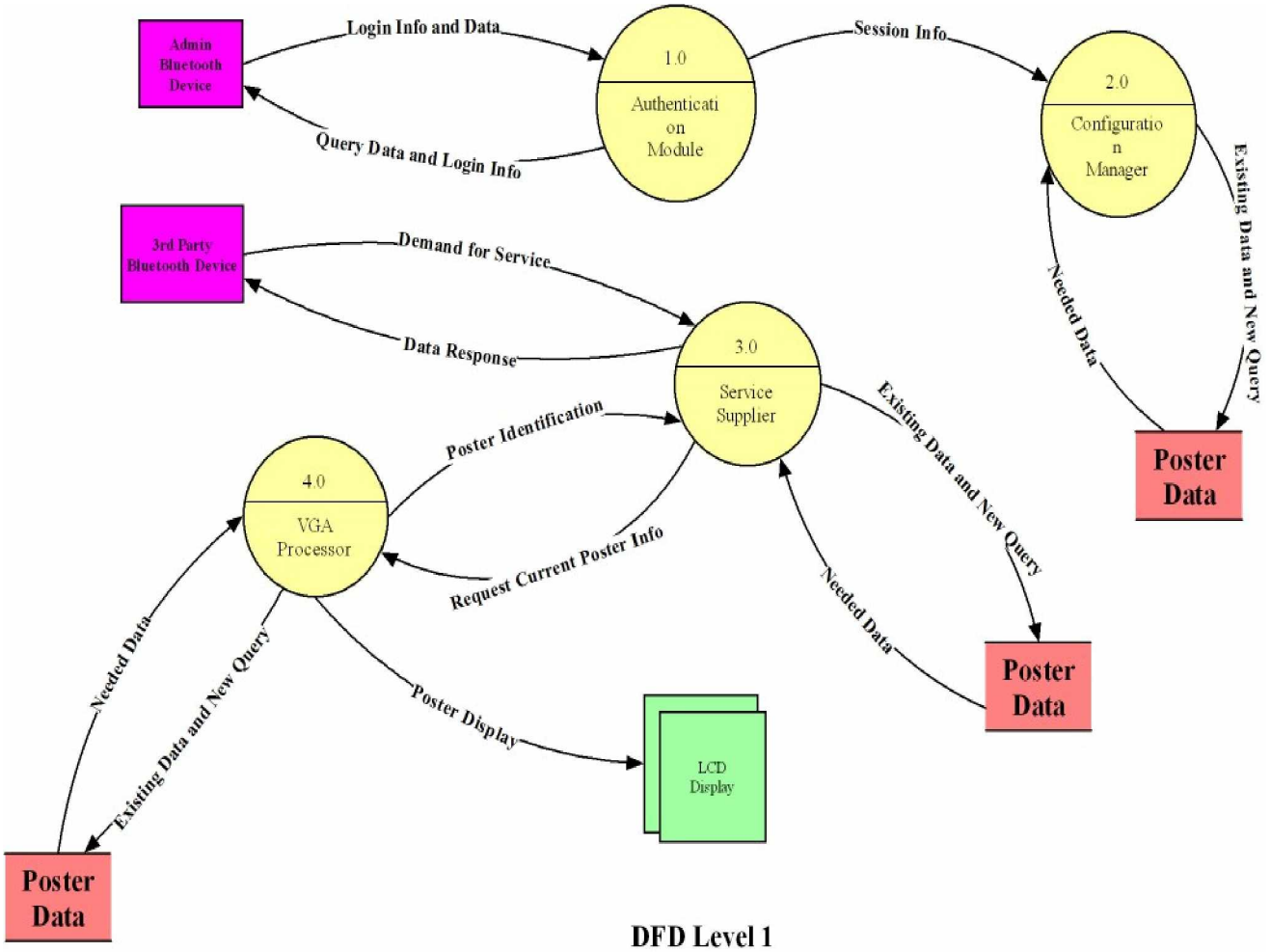
6 1. Data Flow Revised

6.5.1 DFD Level 0



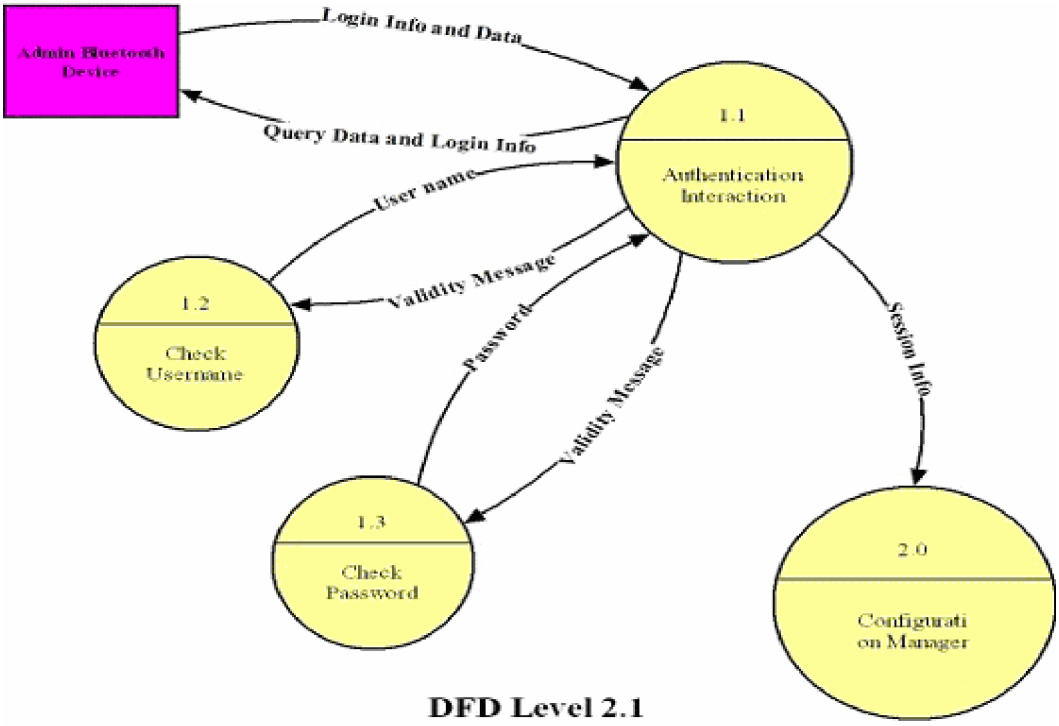
LEVEL 0 DFD

6.5.2 DFD Level 1

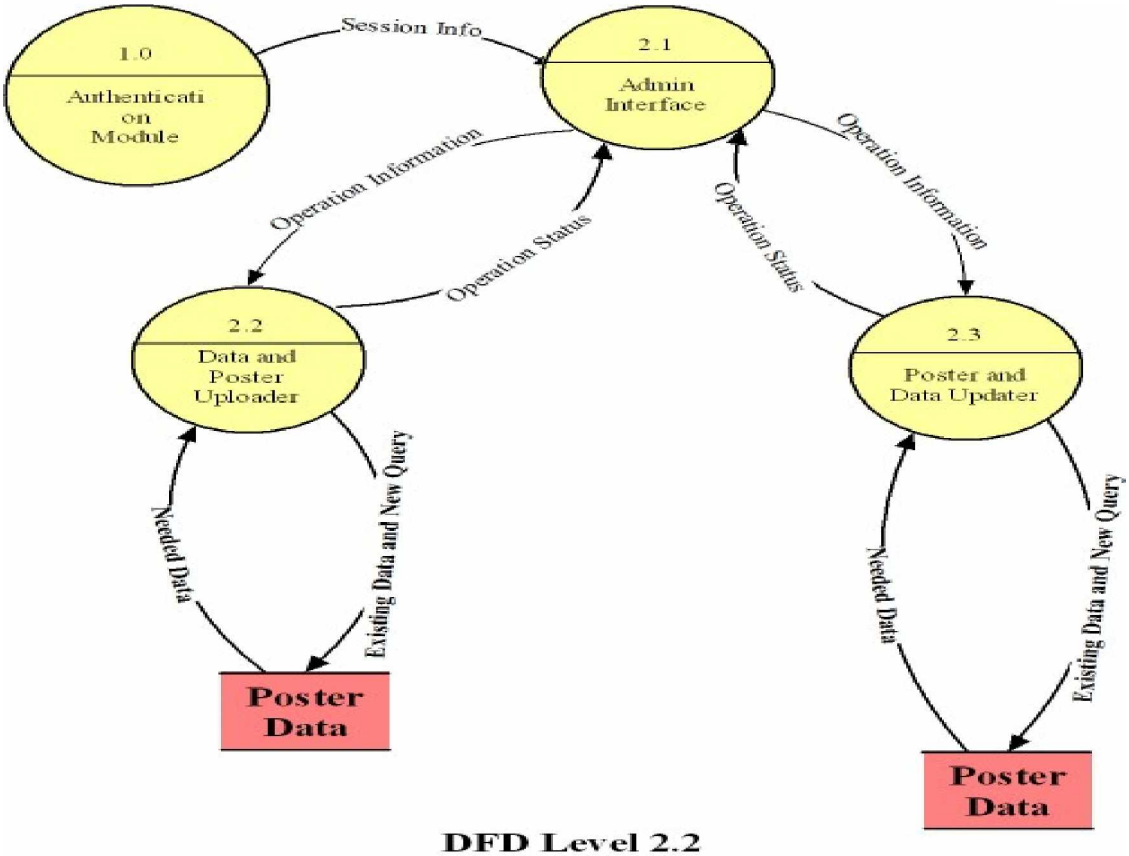


DFD Level 1

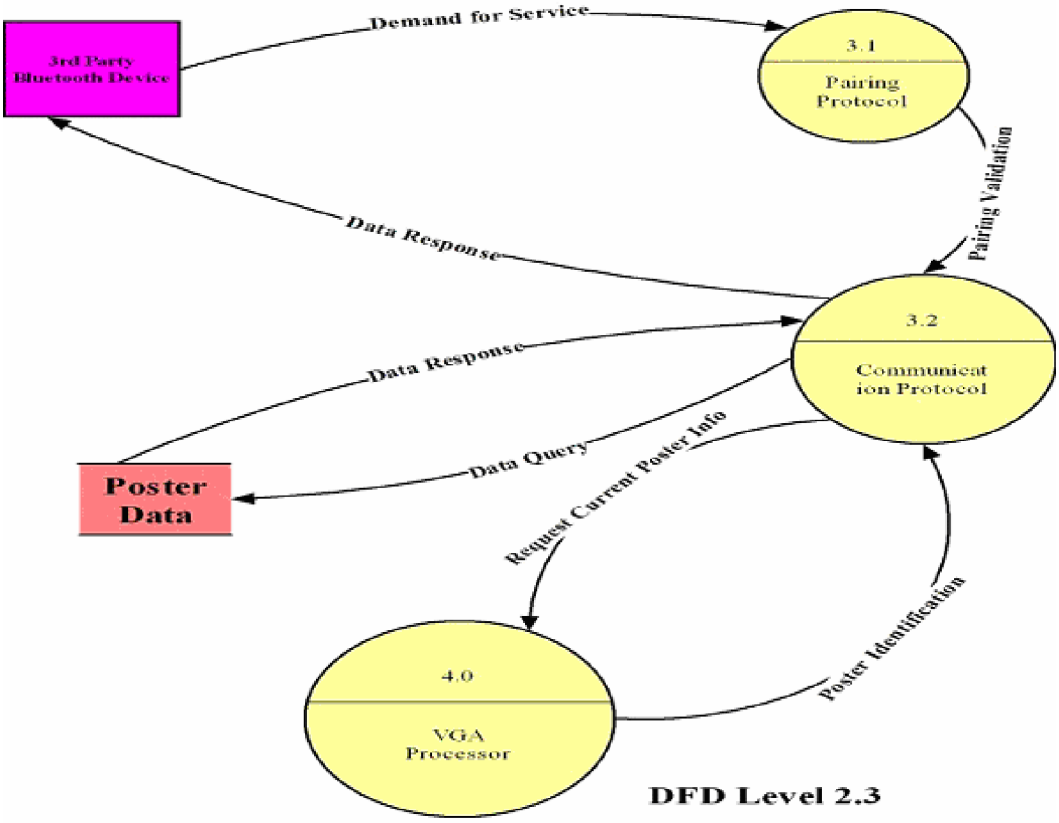
6.5.3 Level 2 DFDs



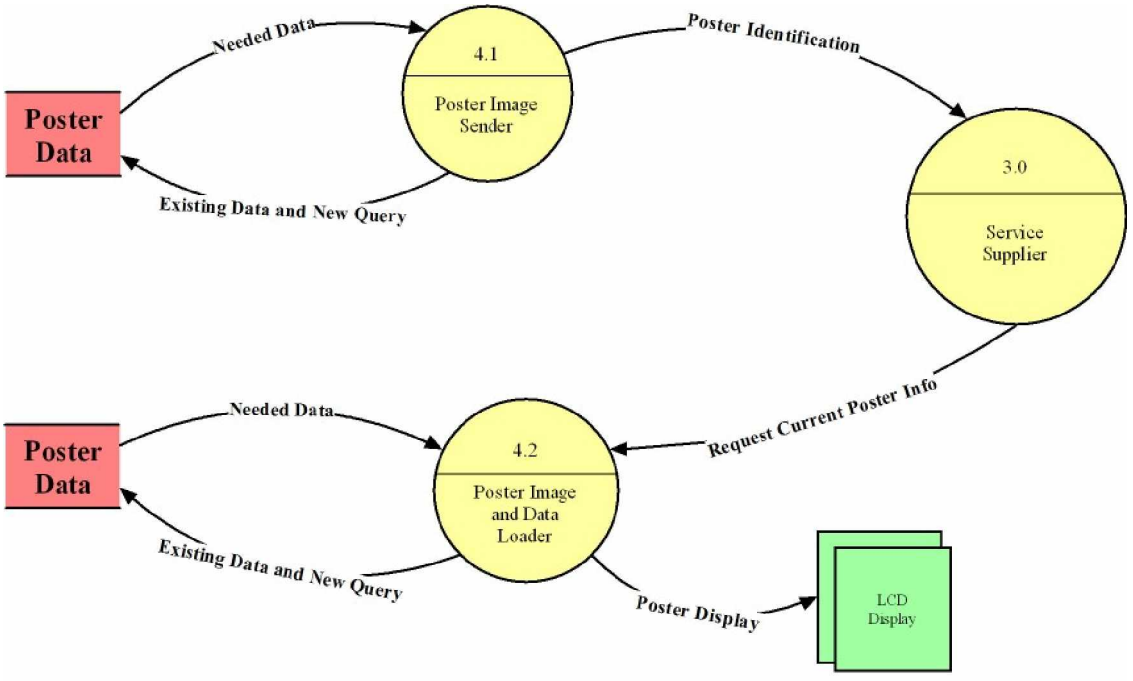
DFD Level 2.1



DFD Level 2.2



DFD Level 2.3



DFD Level 2.4

6 2. Usage Scenarios

Basically we have three major users: admins, server and clients.

Admins are the users who buy our product and display their posters and send their information to the clients via Poster in Blue. Clients are the end users; having a Bluetooth device such as cell phones, PDAs, Palms or laptops and taking the advantage of data sent via Poster in Blue by accepting the request. Finally server is the system that supplies LCD display and data transfer.

Admin Use Cases:

1- Firstly Admins must log in and get authentication to be able to do all Admin activities, *see figure 12*.
2- Admins can start Poster in Blue Application to enable data transmission and LCD Display or they can start Poster in Blue Application and only show the poster in LCD Display, *see figure 13.*
3- Admins can upload posters and upload the data they are sending via Bluetooth. For this purpose they must update the configuration file first. In the configuration file, information about all posters that will be displayed and the data that will be sent via Bluetooth and their display/send durations will be held (Since more than 1 poster can be shown in the LCD in turn, system must know when to show which poster and when to send which data).
   While uploading posters Admins can use their own posters or template posters. While uploading information Admins can only change the information that is being sent or may choose to send other kind of data such as Calendar Events, Files or Business Cards, *see figure 14*.
4- Admins can close Poster in Blue Application in two ways. They can either close the whole application or they can only stop the Bluetooth transmission. Because they may sometimes want not to send the information that is being displayed on LCD, *see figure 15*.
5- Admins can get the current configuration from the server, *see figure 16.*

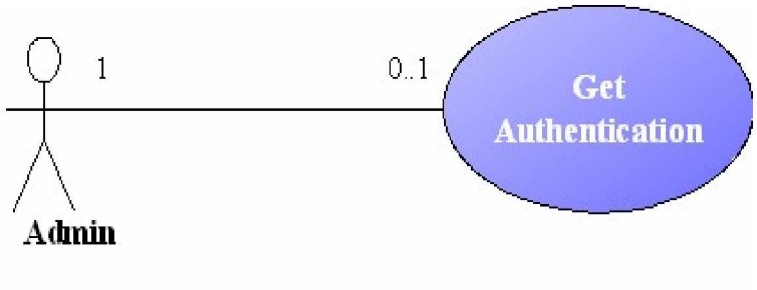


Figure 12: Admin Use Case1

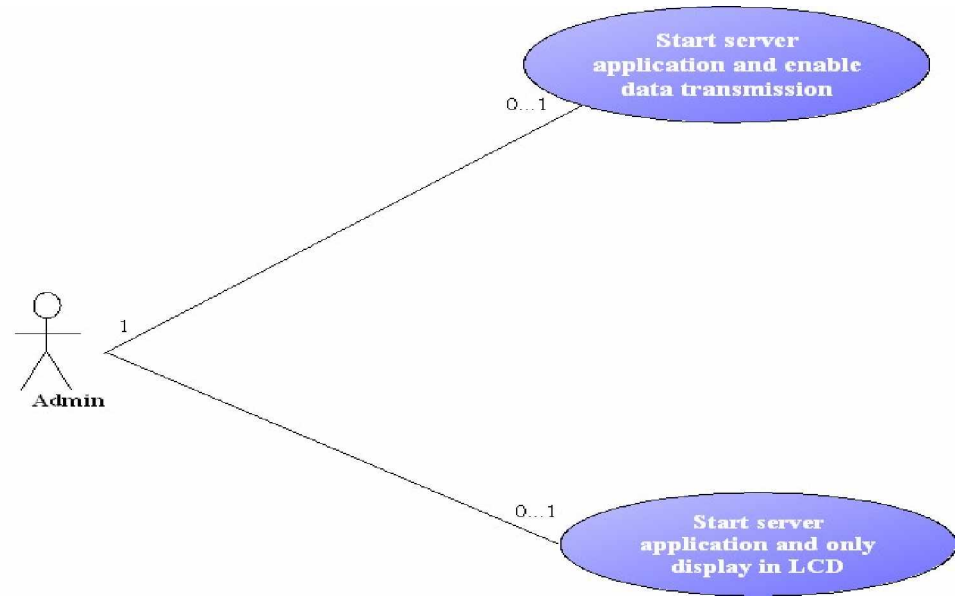


Figure 13 : Admin Use Case 2

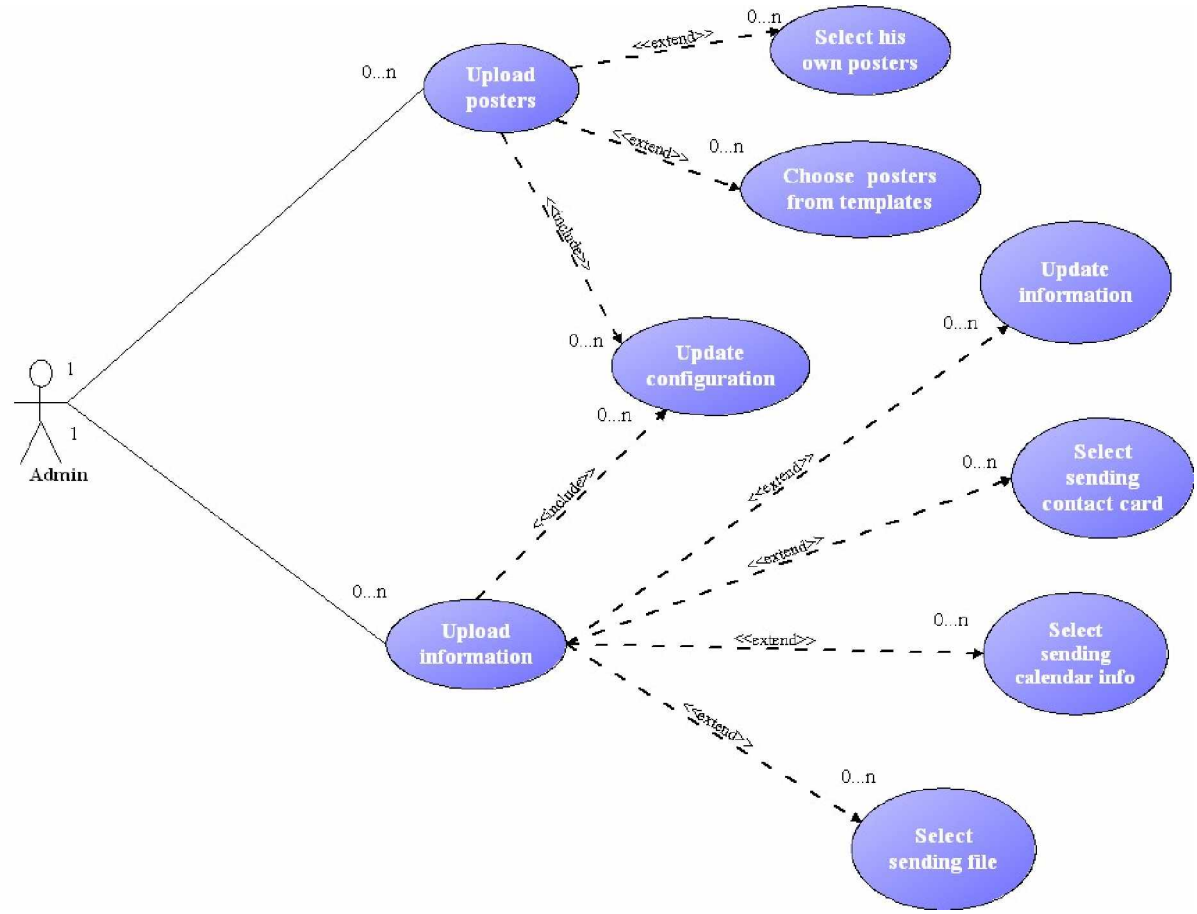


Figure14: Admin Use Case 3

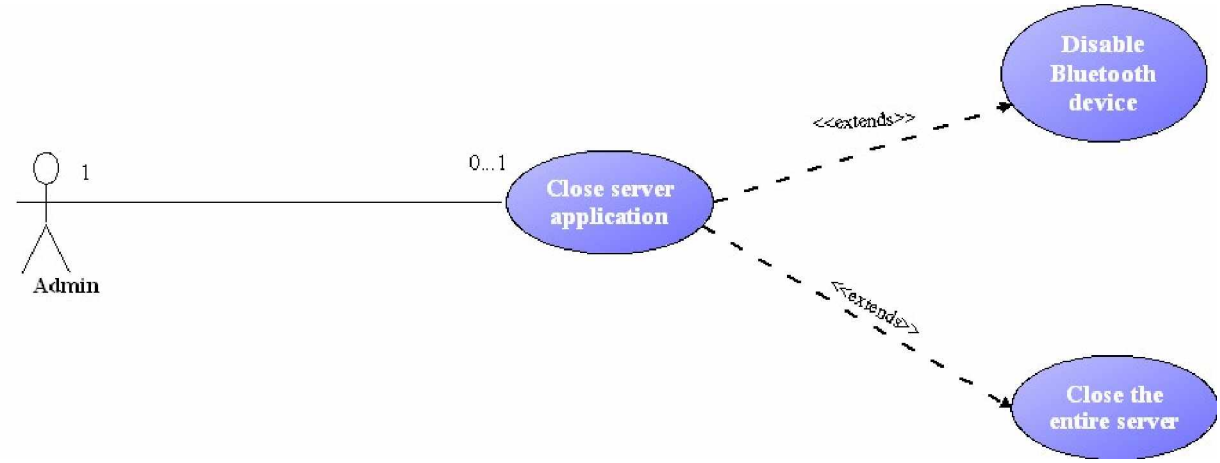


Figure15: Admin Use Case 4

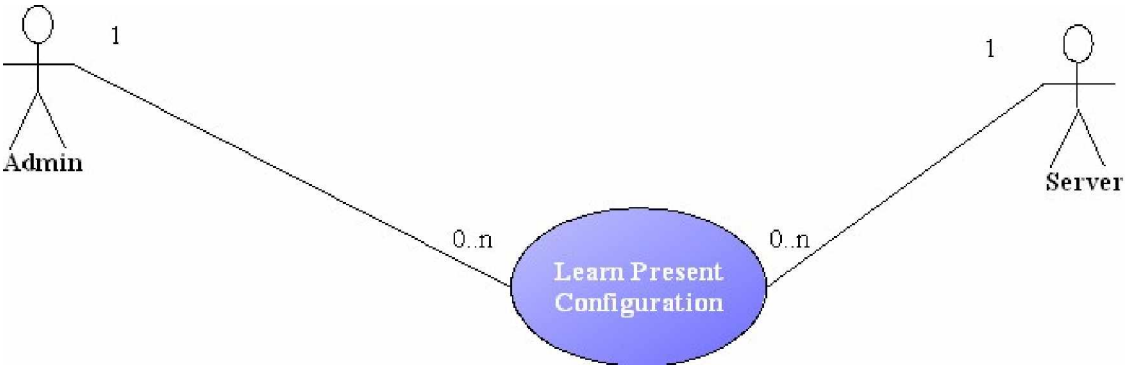


Figure 16: Admin Use Case 5

Client Use Case:

They accept or reject the data sent by Poster in Blue Application after discovering the servers in their range, *see figure 17.*

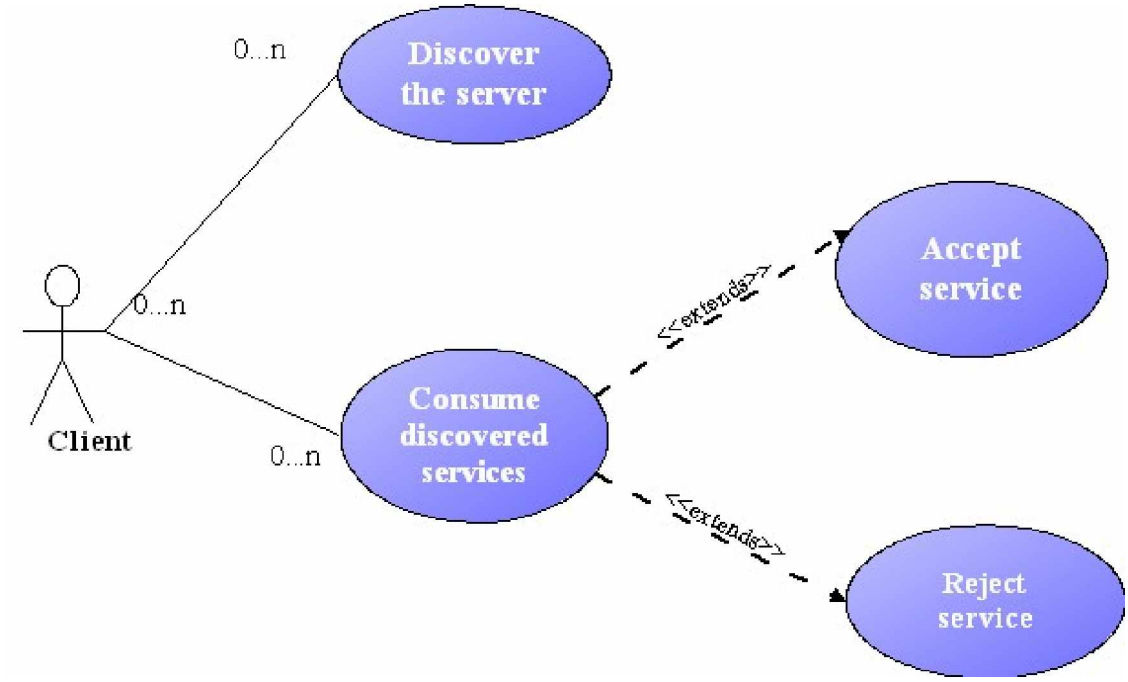


Figure17: Client Use Case

Server Use Case:

After server is started, it is going to advertise the services that it supplies. After processing the configuration files, it decides on which data to send and which poster to display. It waits for the clients and handles their requests by sending calendar event, business card or a file to the clients. It also displays the poster in the LCD. And finally it stops the advertising service. It can also send the current configuration of the application to the Admin, *see figure 18.*

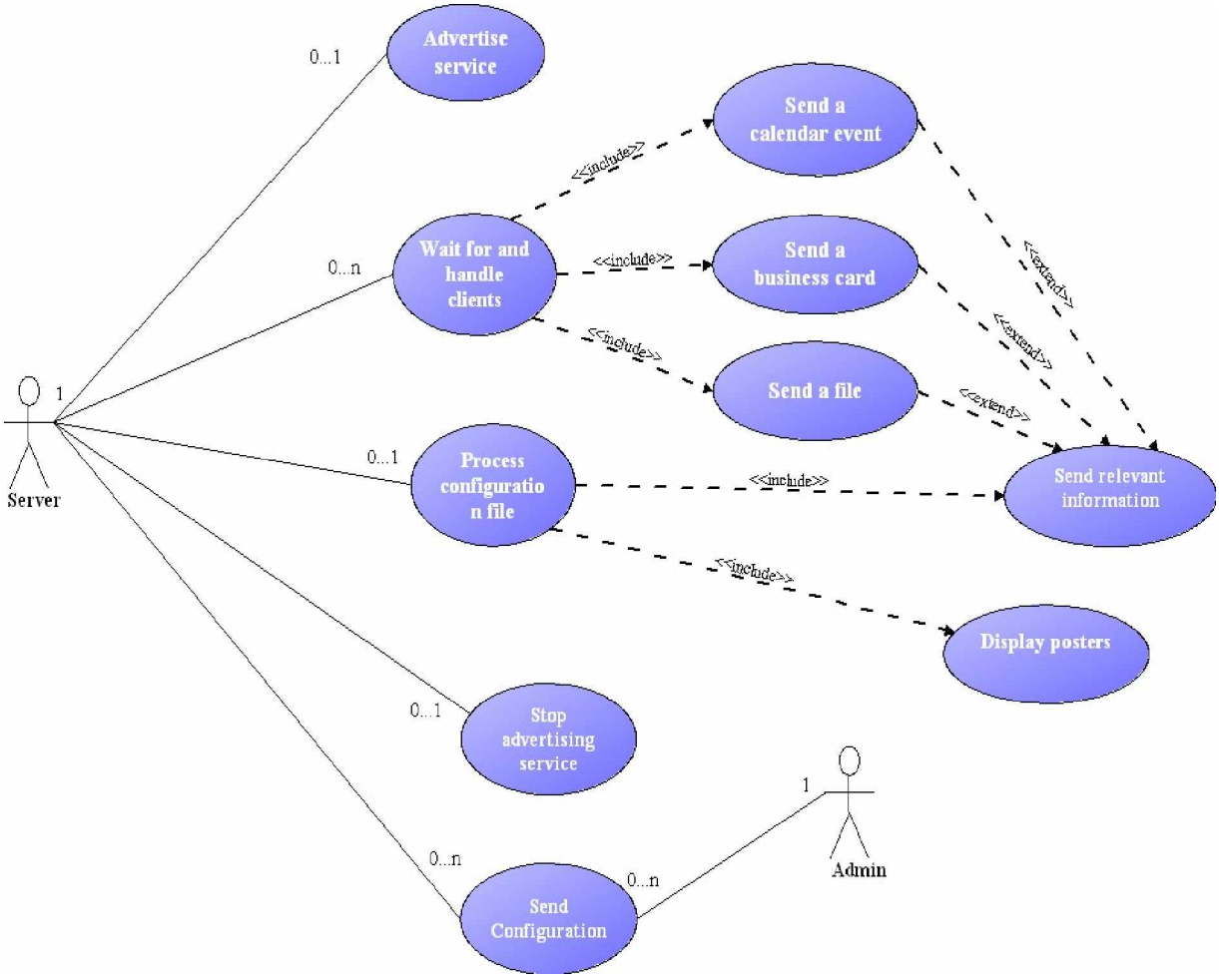


Figure18: Server Use Case

7 Bibliography

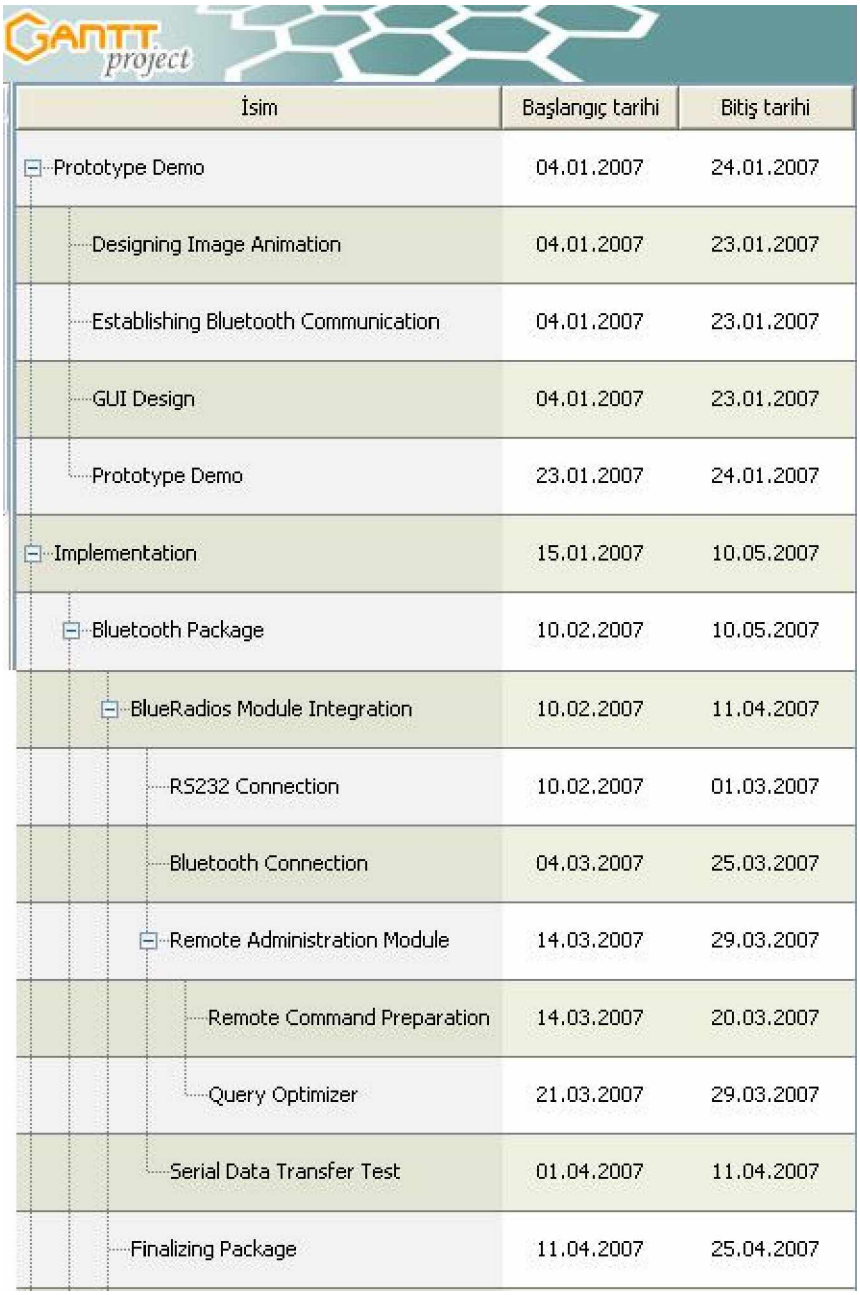
[1] Source, http://www.cs.utk.edu/~dasgupta/bluetooth

[2] Bluetooth passkey (PIN) is the key used to authenticate two Bluetooth devices (that have not previously exchanged link keys). The Bluetooth PIN has different representations at different levels. PINBB is a 128 bit (16 bytes) key used in base band level during pairing procedure while PINUI is the character representation of PINBB (coded using Unicode UTF-8) used at User Interface level.
Source, http://www.securityfocus.com

[3] XILINX ISE 8.2i, www.**xilinx**.com/support/sw_manuals/**xilinx**82/index.htm

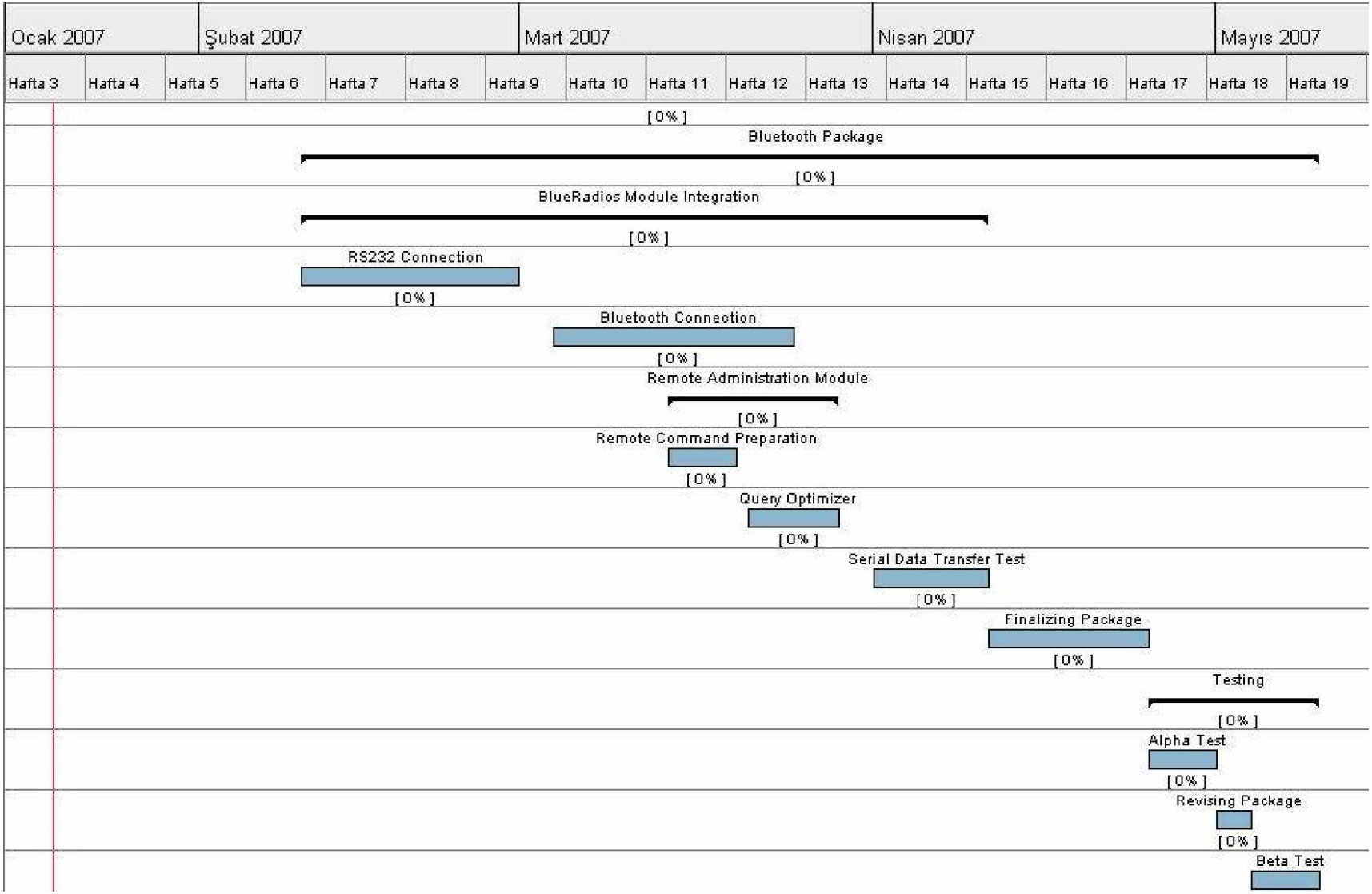
8 Appendix

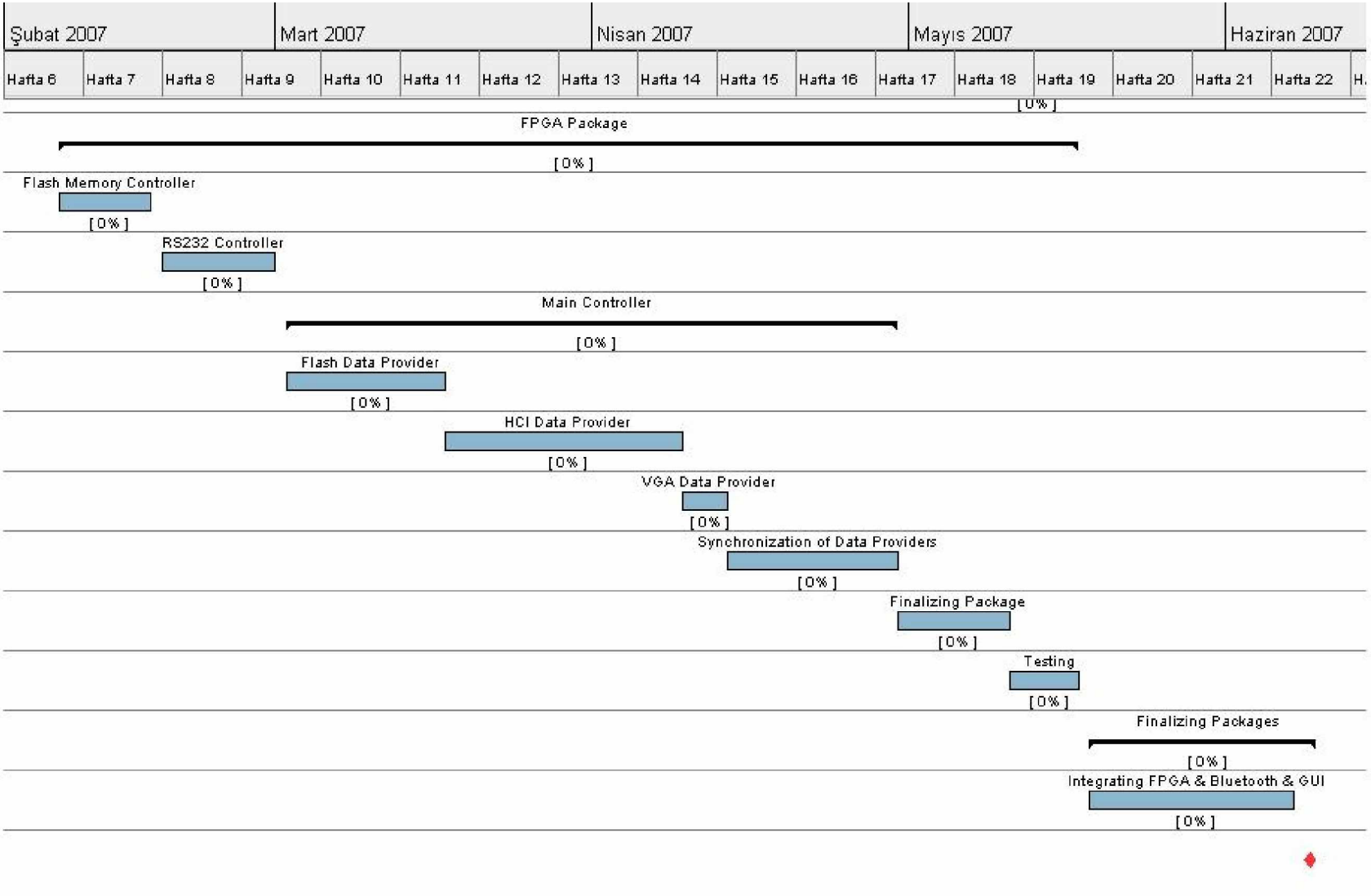
Appendix A: Ganntchart



İsim	Başlangıç tarihi	Bitiş tarihi
Prototype Demo	04.01.2007	24.01.2007
Designing Image Animation	04.01.2007	23.01.2007
Establishing Bluetooth Communication	04.01.2007	23.01.2007
GUI Design	04.01.2007	23.01.2007
Prototype Demo	23.01.2007	24.01.2007
Implementation	15.01.2007	10.05.2007
Bluetooth Package	10.02.2007	10.05.2007
BlueRadios Module Integration	10.02.2007	11.04.2007
RS232 Connection	10.02.2007	01.03.2007
Bluetooth Connection	04.03.2007	25.03.2007
Remote Administration Module	14.03.2007	29.03.2007
Remote Command Preparation	14.03.2007	20.03.2007
Query Optimizer	21.03.2007	29.03.2007
Serial Data Transfer Test	01.04.2007	11.04.2007
Finalizing Package	11.04.2007	25.04.2007

Testing	25.04.2007	10.05.2007
Alpha Test	25.04.2007	01.05.2007
Revising Package	01.05.2007	04.05.2007
Beta Test	04.05.2007	10.05.2007
FPGA Package	10.02.2007	11.05.2007
Flash Memory Controller	10.02.2007	18.02.2007
RS232 Controller	19.02.2007	01.03.2007
Main Controller	02.03.2007	25.04.2007
Flash Data Provider	02.03.2007	16.03.2007
HCI Data Provider	16.03.2007	06.04.2007
VGA Data Provider	06.04.2007	10.04.2007
Synchronization of Data Providers	10.04.2007	25.04.2007
Finalizing Package	25.04.2007	05.05.2007
Testing	05.05.2007	11.05.2007
Finalizing Packages	12.05.2007	01.06.2007
Integrating FPGA & Bluetooth & GUI	12.05.2007	30.05.2007
Final Release Demo	31.05.2007	01.06.2007


Ocak 2007
Şubat 2007
Mart 2007
Nisan 2007
Mayıs 2007
Hafta 3
Hafta 4
Hafta 5
Hafta 6
Hafta 7
Hafta 8
Hafta 9
Hafta 10
Hafta 11
Hafta 12
Hafta 13
Hafta 14
Hafta 15
Hafta 16
Hafta 17
Hafta 18
Hafta 19
[0%]
Bluetooth Package
[0%]
BlueRadios Module Integration
[0%]
RS232 Connection
[0%]
Bluetooth Connection
[0%]
Remote Administration Module
[0%]
Remote Command Preparation
[0%]
Query Optimizer
[0%]
Serial Data Transfer Test
[0%]
Finalizing Package
[0%]
Testing
[0%]
Alpha Test
[0%]
Revising Package
[0%]
Beta Test


Şubat 2007
Mart 2007
Nisan 2007
Mayıs 2007
Haziran 2007
Hafta 6
Hafta 7
Hafta 8
Hafta 9
Hafta 10
Hafta 11
Hafta 12
Hafta 13
Hafta 14
Hafta 15
Hafta 16
Hafta 17
Hafta 18
Hafta 19
Hafta 20
Hafta 21
Hafta 22
[0%]
FPGA Package
[0%]
Flash Memory Controller
[0%]
RS232 Controller
[0%]
Main Controller
[0%]
Flash Data Provider
[0%]
HCI Data Provider
[0%]
VGA Data Provider
[0%]
Synchronization of Data Providers
[0%]
Finalizing Package
[0%]
Testing
[0%]
Finalizing Packages
[0%]
Integrating FPGA & Bluetooth & GUI
[0%]

Appendix B: VGA Design Sample

```
1  ----------------------------------------------------------------------------
2  -- This design reads an image from SDRAM and displays it on a VGA monitor
3  ----------------------------------------------------------------------------
4  library IEEE, unisim, work;
5  use IEEE.std_logic_1164.all;
6  use IEEE.numeric_std.all;
7  use unisim.vcomponents.all;
8  use work.vga_pckg.all;
9  use work.xsasdram.all;
10 use work.common.all;
11
12 entity test_vga is
13   generic(
14     SDRAM_NROWS     :       natural := 8192;  -- 4096 for XSA-50, XSA-100; 8192 for XSA-200, XSA-3S1000
15     SDRAM_NCOLS     :       natural := 512;  -- 256 for XSA-50; 512 for XSA-100, XSA-200, XSA-3S1000
16     DATA_WIDTH      :       natural := 16;  -- SDRAM databus width
17     SADDR_WIDTH     :       natural := 13;  -- # of SDRAM address bits
18     HADDR_WIDTH     :       natural := 24;  -- host-side address width
19     FREQ            :       natural := 100_000;  -- 50 MHz for XSA-50, XSA-100; 100 MHz for XSA-200, XSA-3S1000
20     CLK_DIV         :       natural := 2;  -- pixel clock = FREQ / CLK_DIV
21     PIXEL_WIDTH     :       natural := 8;  -- width of a pixel in memory
22     NUM_RGB_BITS    :       natural := 3;  -- #bits in each R,G,B component of a pixel
23     PIXELS_PER_LINE :       natural := 800;  -- width of image in pixels
24     LINES_PER_FRAME :       natural := 600;  -- height of image in scanlines
25     FIT_TO_SCREEN   :       boolean := true  -- adapt video timing to fit image width x height
26     );
27   port(
28     rst_n           : in    std_logic;  -- reset
29     clk             : in    std_logic;  -- master clock (frequency set by FREQ)
30     vsync_n         : out   std_logic;  -- VGA vertical sync
31     hsync_n         : out   std_logic;  -- VGA horizontal sync
32     red             : out   std_logic_vector(NUM_RGB_BITS-1 downto 0);  -- VGA red signals
33     green           : out   std_logic_vector(NUM_RGB_BITS-1 downto 0);  -- VGA green signals
34     blue            : out   std_logic_vector(NUM_RGB_BITS-1 downto 0);  -- VGA blue signals
35     -- SDRAM I/O
36     sclkfb          : in    std_logic;  -- clock from SDRAM after PCB delays
37     sclk            : out   std_logic;  -- SDRAM clock sync'ed to master clock
38     cke             : out   std_logic;  -- clock-enable to SDRAM
39     cs_n            : out   std_logic;  -- chip-select to SDRAM
40     ras_n           : out   std_logic;  -- SDRAM row address strobe
41     cas_n           : out   std_logic;  -- SDRAM column address strobe
42     we_n            : out   std_logic;  -- SDRAM write enable
43     ba              : out   unsigned(1 downto 0);  -- SDRAM bank address bits
44     sAddr           : out   unsigned(SADDR_WIDTH-1 downto 0);  -- SDRAM row/column address
45     sData           : inout unsigned(DATA_WIDTH-1 downto 0);  -- SDRAM in/out databus
46     dqmh            : out   std_logic;  -- high databits I/O mask
47     dqml            : out   std_logic   -- low databits I/O mask
48     );
49 end entity;
50
51 architecture arch of test_vga is
52
53   signal clklx        : std_logic;      -- sync'ed clock from SDRAM controller
54   signal rst          : std_logic;      -- reset signal
55   signal eof          : std_logic;      -- end-of-frame signal from VGA controller
56   signal earlyOpBegun : std_logic;      -- indicates when an SDRAM read operation has begun
57   signal rdDone       : std_logic;      -- indicates when data read from the SDRAM is available
58   signal full, full_n : std_logic;      -- indicates when the VGA pixel buffer is full
59   signal address      : unsigned(HADDR_WIDTH-1 downto 0);  -- SDRAM address counter
60   signal pixel        : unsigned(DATA_WIDTH-1 downto 0);  -- pixel values from SDRAM
61
62 begin
63
64   rst <= not rst_n;
65
66   -- update the SDRAM address counter
67   process(clklx)
68   begin
69     if rising_edge(clklx) then
70       if eof = YES then
71         address <= TO_UNSIGNED(0, address'length);  -- reset the address at the end of a video frame
72       elsif earlyOpBegun = YES then
73         address <= address + 1;         -- go to the next address once the read of the current address has begun
74       end if;
```

```
75      end if;
76    end process;
77
78    -- XSA SDRAM controller used to get pixel data from the external SDRAM
79    u0 : XSASDRAMCntl
80      generic map(
81        FREQ         => FREQ,
82        PIPE_EN      => true,             -- use pipelining for maximum speed
83        MAX_NOP      => 1000000,          -- disable self-refresh since it takes too long to re-awaken the SDRAM with video timing
84        DATA_WIDTH   => DATA_WIDTH,
85        NROWS        => SDRAM_NROWS,
86        NCOLS        => SDRAM_NCOLS,
87        HADDR_WIDTH  => HADDR_WIDTH,
88        SADDR_WIDTH  => SADDR_WIDTH
89        )
90      port map(
91        -- host side
92        clk          => clk,              -- master clock
93        clklx        => clklx,            -- master clock resync'ed to account for delays to external SDRAM
94        rst          => rst,
95        rd           => full_n,           -- initiate a read when the VGA pixel buffer is not full
96        hAddr        => address,          -- the address to read from is stored in the address counter
97        earlyOpBegun => earlyOpBegun,     -- indicate when the read operation has actually begun
98        rdDone       => rdDone,           -- indicate when the data from the read operation is available
99        hDOut        => pixel,            -- this is the pixel data that was read from the SDRAM
100       wr           => '0',              -- no SDRAM writing is needed in this application
101       hDIn         => TO_UNSIGNED(0, DATA_WIDTH),  -- set the SDRAM write-data bus to zeroes
102       -- SDRAM side
103       sclkfb       => sclkfb,
104       sclk         => sclk,
105       cke          => cke,
106       cs_n         => cs_n,
107       ras_n        => ras_n,
108       cas_n        => cas_n,
109       we_n         => we_n,
110       ba           => ba,
111       sAddr        => sAddr,
112       sData        => sData,
113       dqmh         => dqmh,
114       dqml         => dqml
115       );
116
117   -- VGA generator
118   ul : vga
119     generic map (
120       FREQ            => FREQ,
121       CLK_DIV         => CLK_DIV,
122       PIXEL_WIDTH     => PIXEL_WIDTH,
123       PIXELS_PER_LINE => PIXELS_PER_LINE,
124       LINES_PER_FRAME => LINES_PER_FRAME,
125       NUM_RGB_BITS    => NUM_RGB_BITS,
126       FIT_TO_SCREEN   => FIT_TO_SCREEN
127       )
128     port map (
129       rst             => rst,
130       clk             => clklx,         -- use the resync'ed master clock so VGA generator is in sync with SDRAM
131       wr              => rdDone,        -- write to pixel buffer when the data read from SDRAM is available
132       pixel_data_in   => std_logic_vector(pixel),  -- pixel data from SDRAM
133       full            => full,          -- indicates when the pixel buffer is full
134       eof             => eof,           -- indicates when the VGA generator has finished a video frame
135       r               => red,           -- RGB components
136       g               => green,
137       b               => blue,
138       hsync_n         => hsync_n,       -- horizontal sync
139       vsync_n         => vsync_n,       -- vertical sync
140       blank           => open
141       );
142   full_n <= not full;                   -- negate the full signal for use in controlling the SDRAM read operation
143
144 end arch;
145
```

Appendix C: Data Dictionary

Name:	Login Info and Data
Aliases:	none
Where used/how used:	Admin Bluetooth Device (output) Authentication Interaction (input)
Description: Login Info and Data = Admin Login Info + Poster Data Admin Login Info = *a string of 15 characters* Poster Data = * file of 2.5 MB *	

Name:	Query Data and Login Info
Aliases:	none
Where used/how used:	Authentication Interaction (output) Admin Bluetooth Device (input)
Description: Query Data and Login Info = Poster Query Data + Admin Login Response Poster Query Data = * a string of 200 characters * Admin Login Response = * a string of 15 characters *	

Name:	User name
Aliases:	none
Where used/how used:	Check Username (output) Authentication Interaction (input)
Description: User name = * a string of 15 characters *	

Name:	Validity Message
Aliases:	none
Where used/how used:	Authentication Interaction (output) Check Username (input)
Description:	Validity Message = *a string of 25

Name:	Password
Aliases:	None
Where used/how used:	Check Password (output) Authentication Interaction (input)
Description: Password = * a string of 15 characters *	

Name:	Validity Message
Aliases:	none
Where used/how used:	Authentication Interaction (output) Check Password (input)
Description: Validity Message = *a string of 25 characters*	

Name:	Session Info
Aliases:	none
Where used/how used:	Authentication Interaction (output) Admin Interface (input)
Description: Session Info = * a string of 50 characters*	

Name:	Operation Information
Aliases:	none
Where used/how used:	Operation Status (output) Data and Poster Uploader (input)
Description: Operation Information = *string of 25 characters*	

Name:	Operation Status
Aliases:	none
Where used/how used:	Data and Poster Uploader (output) Operation Status (input)
Description: Operation Status = * string of 25 characters *	

Name:	Needed Data
Aliases:	none
Where used/how used:	Poster Data (output) Data and Poster Uploader (input)
Description: Needed Data = Image + Date + General Info Image = *image file of 2 MB* Date = *date* General Info = * string of 250 characters*	

Name:	Existing Data and New Query
Aliases:	none
Where used/how used:	Data and Poster Uploader (output) Poster Data (input)
Description: Existing Data and New Query = Poster Data + Data Query Poster Data = Image + Date + General Info Image = *image file of 2 MB* Date = *date* General Info = * string of 250 characters	

Name:	Operation Information
Aliases:	none
Where used/how used:	Operation Status (output) Poster and Data Updater (input)
Description:	Operation Information = *string of 25 characters*

Name:	Operation Status
Aliases:	none
Where used/how used:	Data and Poster Updater (output) Operation Status (input)
Description: Operation Status = * string of 25 characters *	

Name:	Needed Data
Aliases:	none
Where used/how used:	Poster Data (output) Data and Poster Updater (input)
Description: Needed Data = Image + Date + General Info Image = *image file of 2 MB* Date = *date* General Info = * string of 250 characters*	

Name:	Existing Data and New Query
Aliases:	none
Where used/how used:	Data and Poster Updater (output) Poster Data (input)
Description: Existing Data and New Query = Poster Data + Data Query Poster Data = Image + Date + General Info Image = *image file of 2 MB* Date = *date* General Info = * string of 250 characters	

Name:	Demand for Service
Aliases:	none
Where used/how used:	3rd Party Bluetooth Device (output) Pairing Protocol (input)
Description: Demand for Servis =Is_Service_Available Is_Service_Available = * boolean *	

Name:	Data Response
Aliases:	none
Where used/how used:	Poster Data (output) Communication Protocol (input)
Description: Data Response = Poster Data Poster Data = Image + Date + General Info Image = *image file of 2 MB* Date = *date* General Info = * string of 250 characters	

Name:	Pairing Validation
Aliases:	none
Where used/how used:	Pairing Protocol (output) Communication Protocol (input)
Description: Pairing Validation = Is_Pairing_Valid Is_Pairing_Valid = *bolean*	

Name:	Data Response
Aliases:	none
Where used/how used:	Poster Data (output) Communication Protocol (input)
Description: Data Response = Poster Data Poster Data = Image + Date + General Info Image = *image file of 2 MB* Date = *date* General Info = * string of 250 characters	

Name:	Data Query
Aliases:	none
Where used/how used:	Communication Protocol (output) Poster Data (input)
Description: Data Query = *string of 100 characters*	

Name:	Request Current Poster Info
Aliases:	none
Where used/how used:	Communication Protocol (output) Poster Image and Data Loader (input)
Description: Request Current Poster Info = Is_Poster_Info_Valid Is_Poster_Info_Valid = * boolean *	

Name:	Poster Identification
Aliases:	none
Where used/how used:	Poster Image Sender (output) Communication Protocol (input)
Description: Poster Identification = Identification code Identification code = * a string 0f 20 characters*	

Name:	Poster Display
Aliases:	none
Where used/how used:	Poster Image and Data Loader (output) LCD Display (input)
Description: Poster Display = * responding screen*	

Name:	Needed Data
Aliases:	none
Where used/how used:	Poster Data (output) Poster Image Sender (input)
Description: Needed Data = Image + Date + General Info Image = *image file of 2 MB* Date = *date* General Info = * string of 250 characters*	

Name:	Existing Data and New Query
Aliases:	none
Where used/how used:	Poster Image Sender (output) Poster Data (input)
Description: Existing Data and New Query = Poster Data + Data Query Poster Data = Image + Date + General Info Image = *image file of 2 MB* Date = *date* General Info = * string of 250 characters	

Name:	Needed Data
Aliases:	none
Where used/how used:	Poster Data (output) Poster Image and Data Loader (input)

Description:

Needed Data = Image + Date + General Info
Image = *image file of 2 MB*
Date = *date*
General Info = * string of 250 characters*

Name:	Existing Data and New Query
Aliases:	none
Where used/how used:	Poster Image and Data Loader (output) Poster Data (input)

Description:

Existing Data and New Query = Poster Data + Data Query
Poster Data = Image + Date + General Info
Image = *image file of 2 MB*
Date = *date*
General Info = * string of 250 characters