MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT of COMPUTER ENGINEERING

CENG 491-COMPUTER ENGINEERING DESIGN 1

'BluePost'

DETAILED DESIGN REPORT

by

YAZILIM

Duygu CEYLAN – e1394782 **Seda ÇAKIROĞLU - e1394816**

Ertay KAYA – e1356948 **Hüseyin ÖĞÜNÇLÜ - e1395318**

Gözde ÖZBAL – e1395326

TABLE OF CONTENTS

1. INTRODUCTION

1.1 PROJECT TITLE

Our project title is “BluePost”.

1.2 PROBLEM DEFINITION

As you all know, paper posters are very common in daily life because of the fact that they are inexpensive and easy to install. However, they lead to some disadvantages as well. Most important of all, they do not provide the opportunity to make a change in the content of the poster once it is installed. In addition, it is possible for a person to forget about the details of the event unless the information is noted. Besides, it is not easy to inform other people about the event when paper posters are used. Furthermore, paper posters are easy to damage. To illustrate, when a person rips the poster, the information is lost and it is costly to bring it back. So the thought of using digital posters for everyday use arises.

1.3 PROJECT SCOPE

We defined the functionalities of our system by internet search and a questionairre about the desires and needs of the users as we have already discussed in the earlier documents.

Because user-friendliness of a system is an important issue, we will develop a computet-based user interface which will give many opportunities to the user such as configuring the images to be displayed and the time duration of each image. The user will also specify the content of the message that will be sent to the bluetooth devices.

As for the technical details about the project, we will design and implement the hardware and software required to make a monitor or television into a digital poster with bluetooth capabilities. Our system will be connected to a bluetooth converter card and a monitor via VGA. We will develop the necessary software for tasks like uploading poster images and event data. By the means of this software package each costumer will be the administrator of his/her own system. The user will be able to upload several images to display them as a slide-show. S/he will be able to specify the time each image will be displayed. When the user wants to make a change in the content, s/he will not need to re-upload all the images. Instead, s/he will upload a new image in place of the image s/he wants to change. The time duration may also be changed. The user must enter a pin number in order to complete the image uploading process, which is specific for every Bluetooth converter card. The software will do the uploading of the images via Bluetooth automatically.

Once the digital poster is ready, people will be able to see the poster on the monitor and follow a procedure on their Bluetooth devices in order to get poster event data as a calendar event (iCal VEVENT).

The project that we will develop may easily be used in places where all kinds of social and cultural activities are held including cinemas, concerts, theaters etc. Also, this project can be used for educational purposes. As an illustration, there are a lot of student clubs that give seminars and meetings. With the help of our project, it will be much easier for the sudents to become aware of these activities and share the information with each other. The project may also be used for commercial purposes. For example, when there is a campaign in some product, the comsumers will easily be informed about it. Another important area where this product can be used is charity campaigns. In that case, information like bank account numbers should be sent to the bluetooth devices. We believe that as the project evolves, it will be much more widely used in different areas.

2. ARCHITECTURAL DESIGN

2.1. SYSTEM MODULES

Our system consists of mainly two parts, getting the necessary data from the user and processing this data. Through the user interface, the user will specify the images that are to be displayed. The images selected by the user will be moved to a directory. Then each image in this directory will be processed to extract the pixel information. This pixel information will be combined by the user input specifying the time duration and the order of each image in the slide show. The combined information will be used to create a hex formatted file in the “Format Conversion Module”.

Additionally, the user will enter the message about the event details that will be sent to the bluetooth devices. This message will be written to a .txt formatted file.

The txt formatted and the hex formatted file will be sent to the board via bluetooth when the user clicks on the “Send via Bluetooth” button in the user interface. To complete the uploading process the user must enter a pin number, which is specific to the bluetooth evaluation kit attached to the board. This pin number will be stored in the Flash Memory of the XSA Board and will be determined before the product release during the programming of the FPGA. **Sending Data to the Board via Bluetooth Module** is responsible for this process.

When the data arrives to the bluetooth evaluation kit, the process of the **Retrieving Data from User via Bluetooth Evaluation Kit Module** will retrieve it. This process involves converting serial bluetooth data to parallel data. Once the data is retrieved through the parallel port of the XSA board, it is written to the SDRAM of the board via the **Register Process Module**.

The pixel information will then be used by the **VGA Process Module** to display images on a monitor in a slide show manner.

Meanwhile, the txt file stored in the SDRAM will be used to send messages about the details of the event to bluetooth devices. For this purpose, first the data stored in the SDRAM is sent to the parallel port of the XSA board via the **Register Process Module**. Then **Sending Data to User**

via Bluetooth Evaluation Kit Module will be used which in turn converts parallel data to serial data.

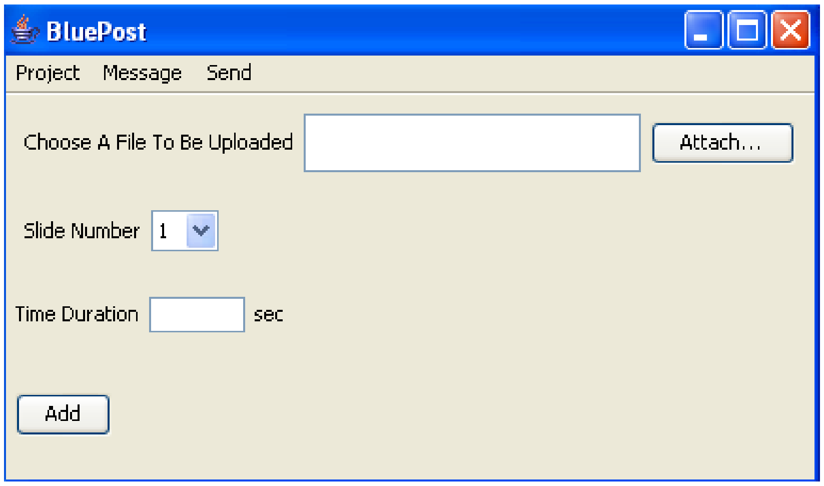
Each module will now be described in detail:

2.1.1 File Uploading Module:

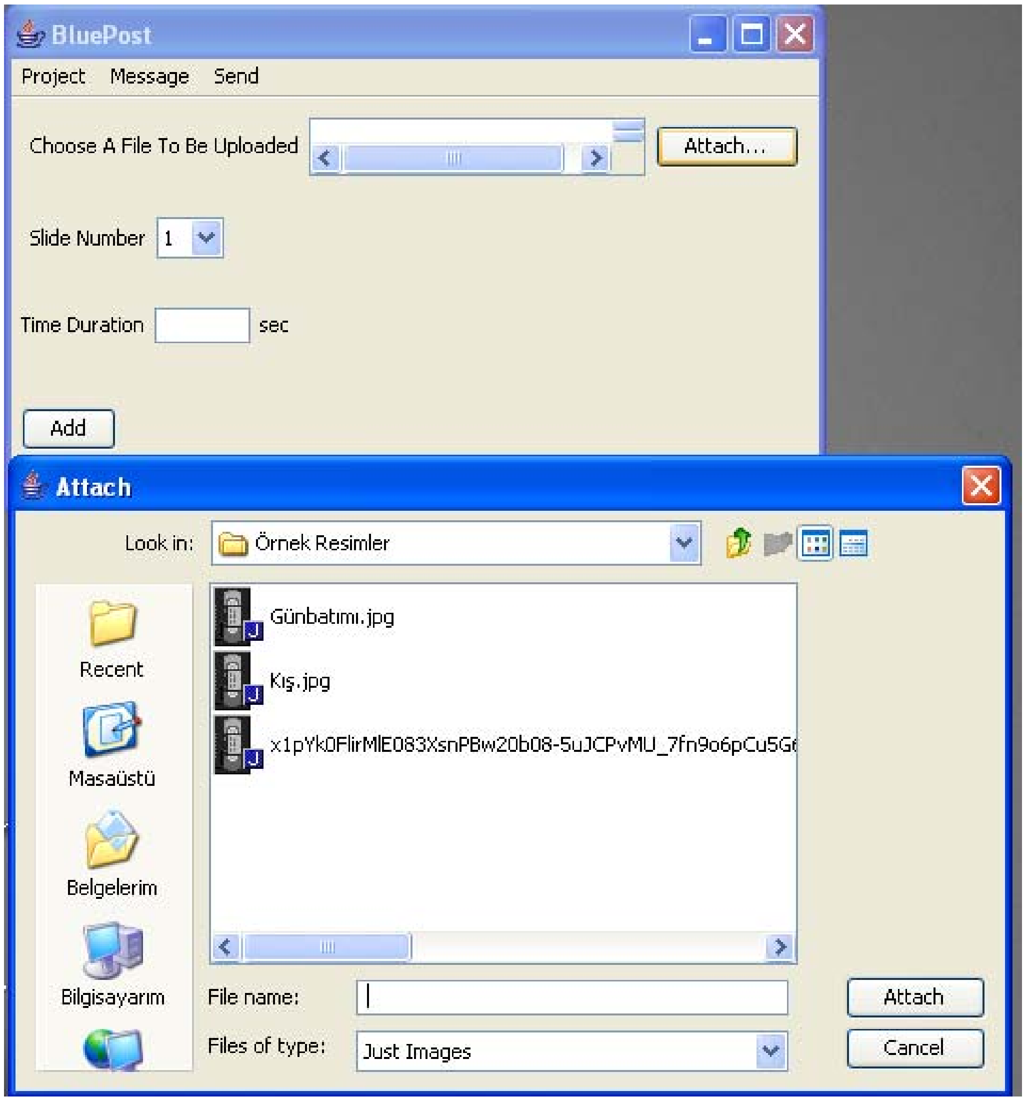
This module is responsible for creating the user interface of our system and organizes the images selected and the message entered by the user. The user interface we designed can be used by the computer users in order to upload images for a slide show that is reachable by the bluetooth device users.

Below, we will explain the user interface while uploading the images and sending the overall slide show to the bluetooth device.

When a user starts our desktop application, s/he will first see a page as below:



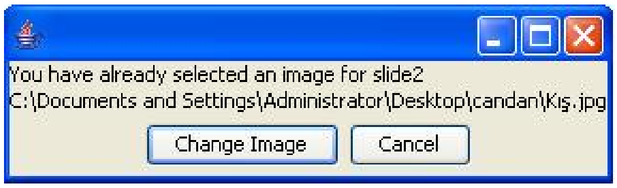
Here, the user can add the image files from his computer by pressing the “Attach” button and when
this button is pressed, the below page will be seen.



The user will be able to select only jpg and gif images.

Before the user adds an image for the slide show, he should also state the slide number of the image to be uploaded as all the images will be presented in a slide show after all the process ends. Displaying too much images on an advirtesement has the disadvantage of boring and distracting the target costumers. Taking this fact and the limitation of memory on our main board into account, we decided to let the user select 10 images at most. Thus the slide number input is an integer between 1 and 10 inclusively. The other input requested from the user is the time duration of the image during the slide show, in seconds as a unit. When all the stated inputs are ready, the user should press the ADD button.

When ADD button is pressed, some controls about the new image and the location of the image in the slide show are made. (For this purpose, the file “bilgi.txt” will be scanned whose definition is made in the following paragraphs.) For example, if the user selects a different image for a previously defined slide, (Then this slide number would exist in the file “bilgi.txt”.) s/he will be given the following warning:


You have already selected an image for slide2
C:\Documents and Settings\Administrator\Desktop\candan\Kış.jpg

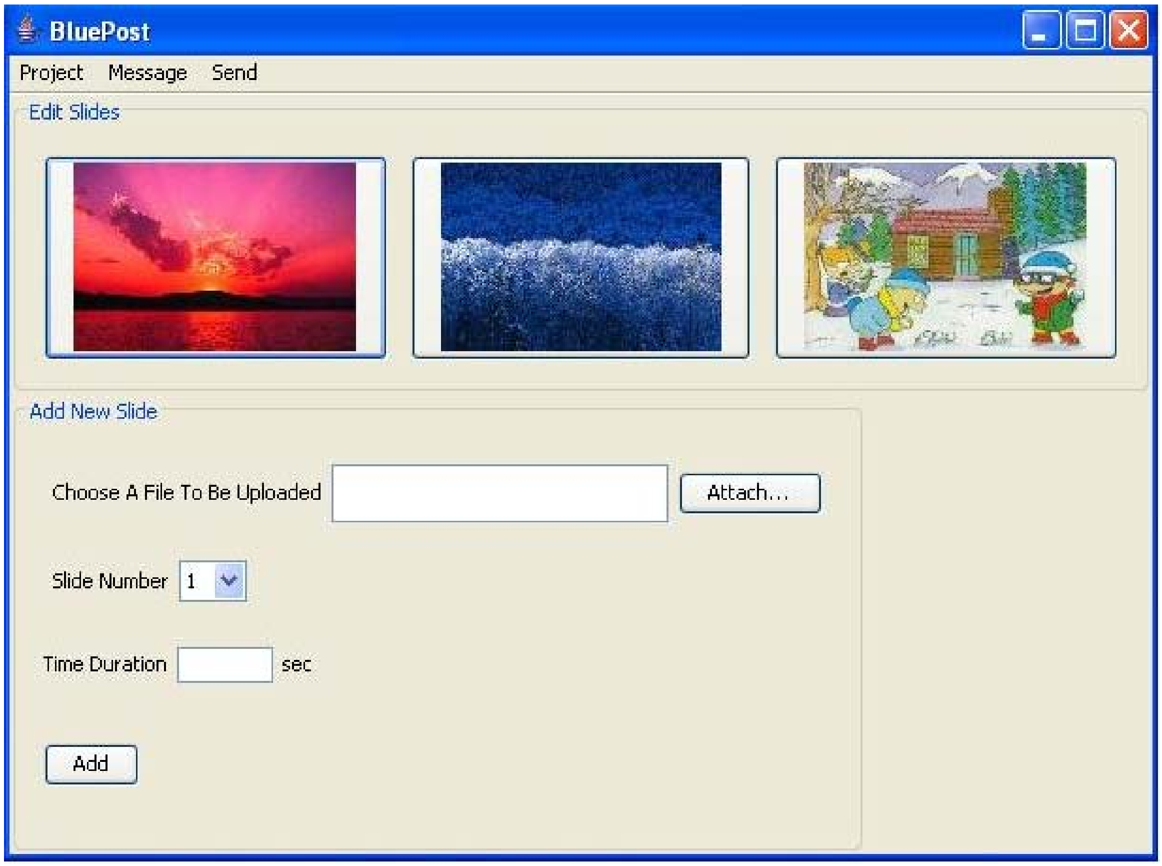
After receiving this error message, the user can change the image for the specified slide number by pressing the “Change Slide” button or cancel the operation by pressing the “Cancel button”.

The images selected by the user are stored in a directory with the initial name “new folder” and when the user saves his/her work, this directory is given the name provided by the user. This directory is created in another directory named “posterspace”. The path of the “posterspace” directory is predetermined by the user during installation of our software product. Additionally, when a new image is added, the slide number and the time duration of this image will be written to a txt file named “bilgi.txt”. This file will be kept in the same folder with the images and each line in this file has the following format:

<slide number> <image file name> <time duration>

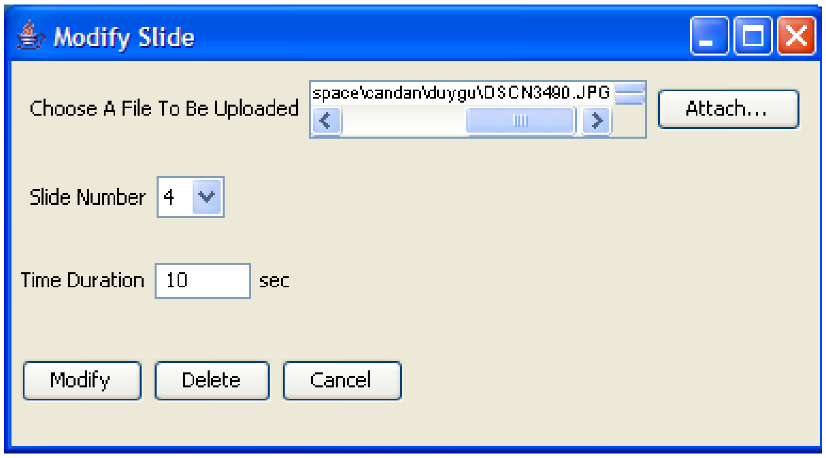
As for the menu bar in the user interface, by means of the “Project” section, the user can create a new project, open an existing project or save the project created previously.

When a project is opened, the user will see a window as below:



In this window, a new image can be added to the existing project by following the same steps stated before for adding a new image.

The user can also make the necessary modifications for an image by clicking on the image and when clicked, the user will see the below window:



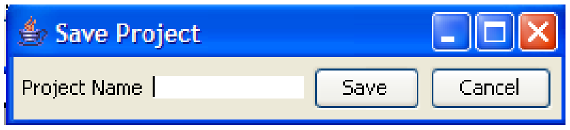
Choose A File To Be Uploaded space\candan\duygu\DSCN3490.JPG

Slide Number 4

Time Duration 10 sec

After making the changes, the user should press Modify button or he can delete the selected image by pressing the Delete button. Then the relevant information in “bilgi.txt” will also be modified or deleted.

When a project is made, and the user wants to save it, the user clicks “Save” from the Project menu bar. And when clicked, the below window will be seen:



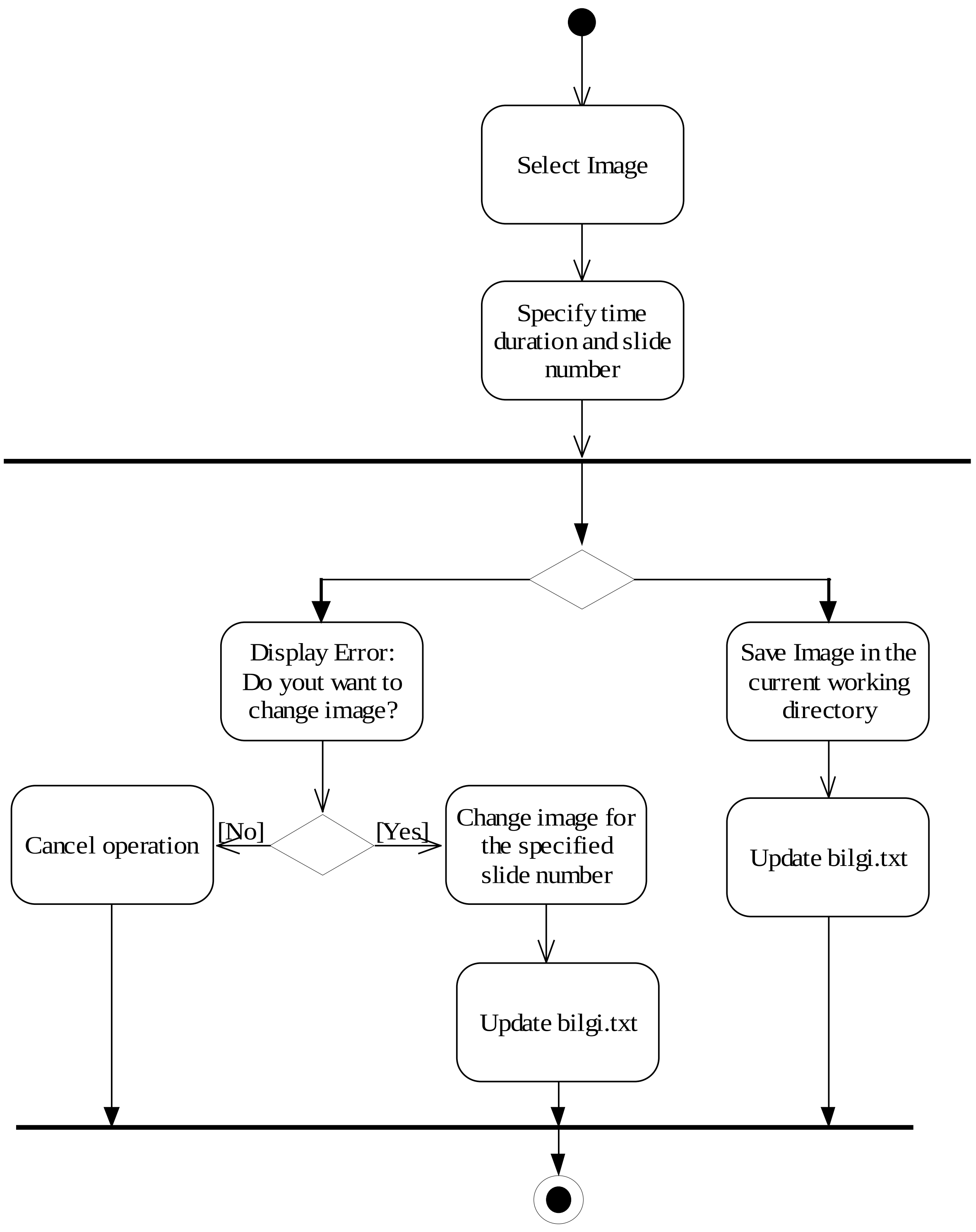
Here the user should write the Project Name he desires and should press Save button for the project he created previously to be saved. If another slide show with the specified name exists, the name entered will be modified. For example, if the user enters “Slides” as the name and there already exists a slide show with the same name, the current project will be renamed to “Slides(2)”. If the user enters the name “Slides”again for a different slide show, that show will be renamed to “Slides(3)”.

From the Message menu bar, the user can write the message to be viewed by the bluetooth device users. This message will be written to a txt file named “mesaj.txt” and this file will be kept in the same directory with the images.

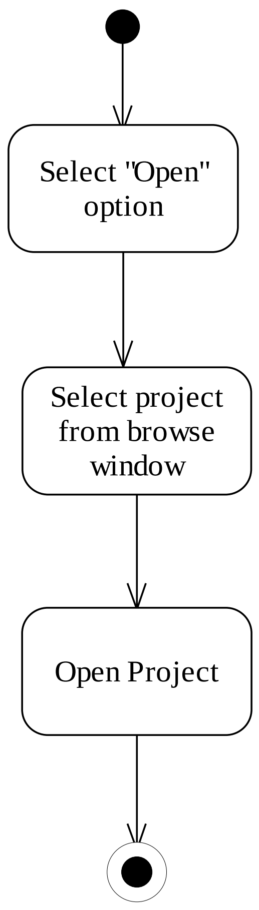
When all the previous steps are completed, the user should press “Send” in the menu bar for the bluetooth process to take place(sending the images and the text). The user will be requested to enter a pin number which is unique to the bluetooth evaluation board and when s/he enters the correct pin number, sending process begins. Here, “Format Conversion” module is initiated.

The following activity diagrams illustrate the process of this module:

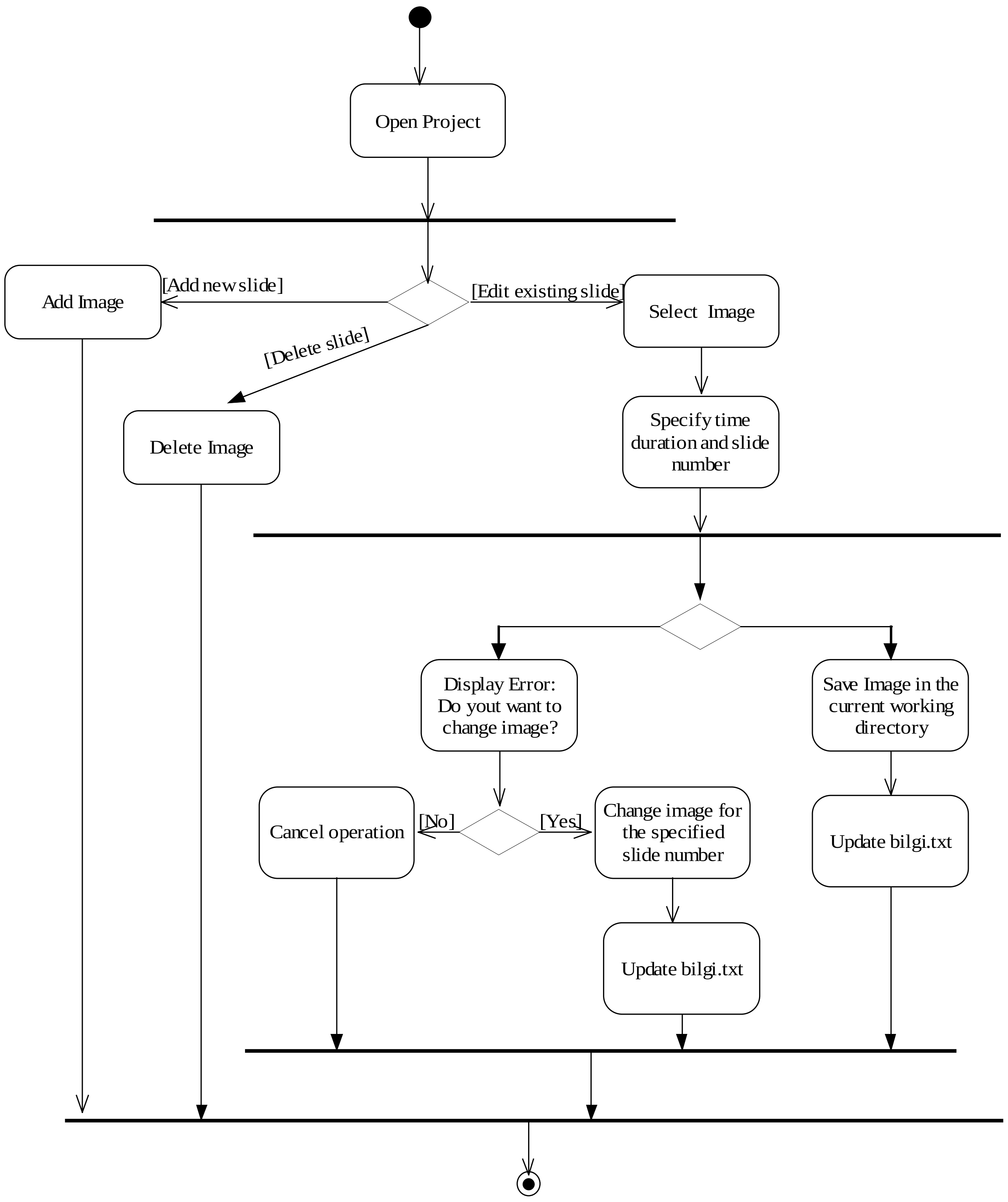
- Activity Diagram for Adding an Image:


Select Image
Specify time duration and slide number
Display Error: Do yout want to change image?
Save Image in the current working directory
Cancel operation
[No]
[Yes]
Change image for the specified slide number
Update bilgi.txt
Update bilgi.txt

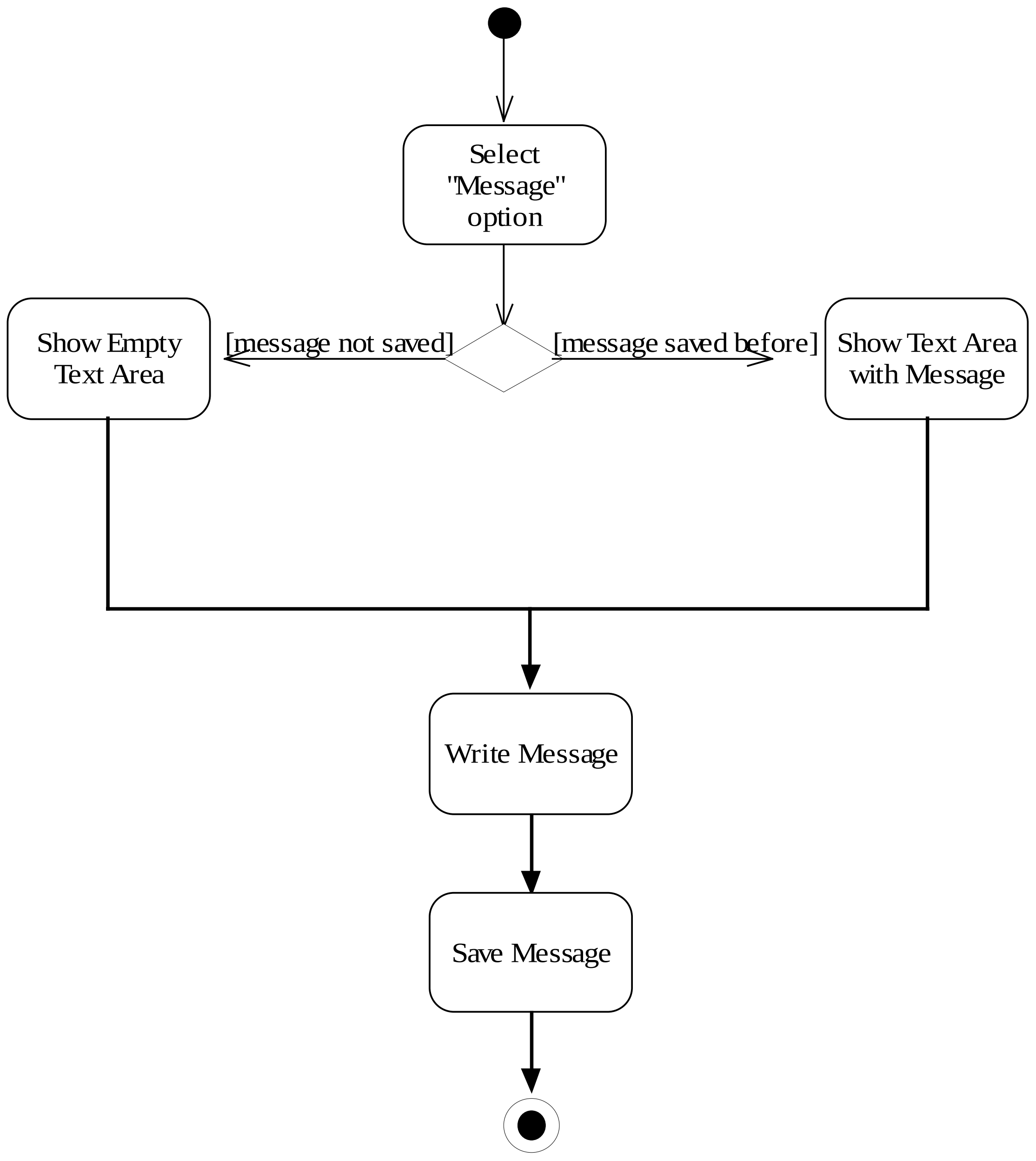
- Activity Diagram for Opening a Slide Show:



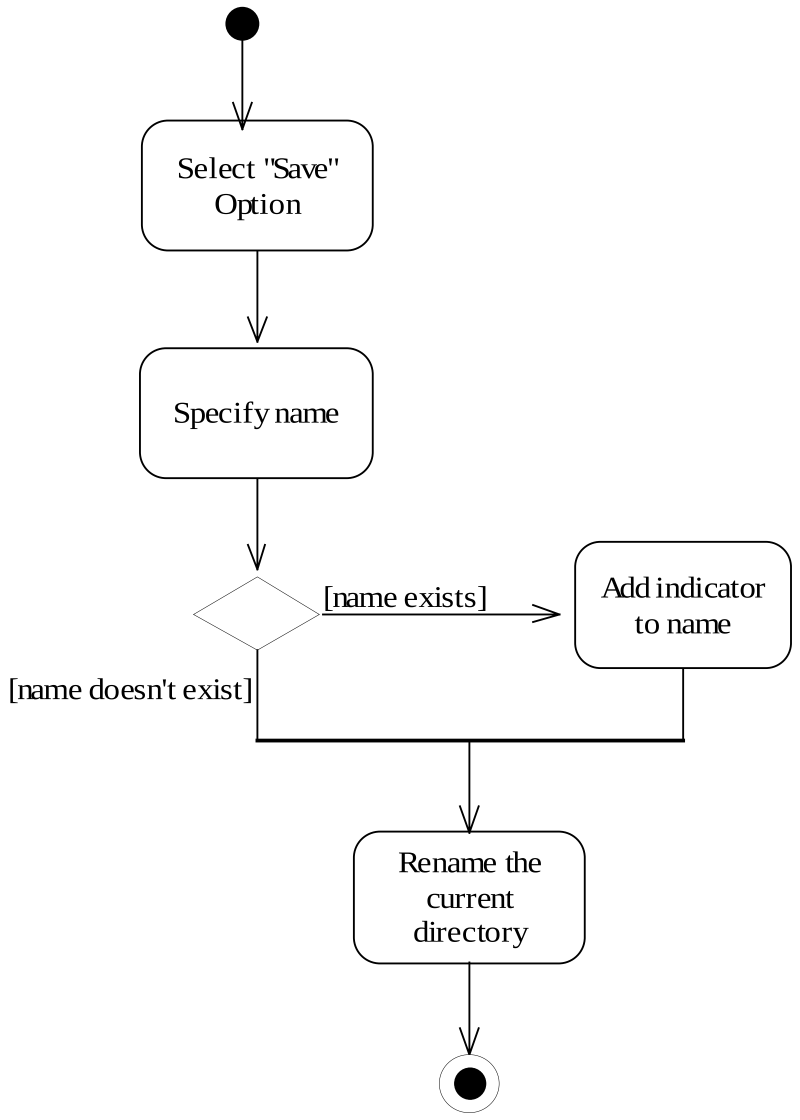
- Activity Diagram for Editing a Slide Show:


Open Project
[Add new slide]
Add Image
[Edit existing slide]
Select Image
[Delete slide]
Delete Image
Specify time duration and slide number
Display Error: Do yout want to change image?
Save Image in the current working directory
Cancel operation
[No]
[Yes]
Change image for the specified slide number
Update bilgi.txt
Update bilgi.txt

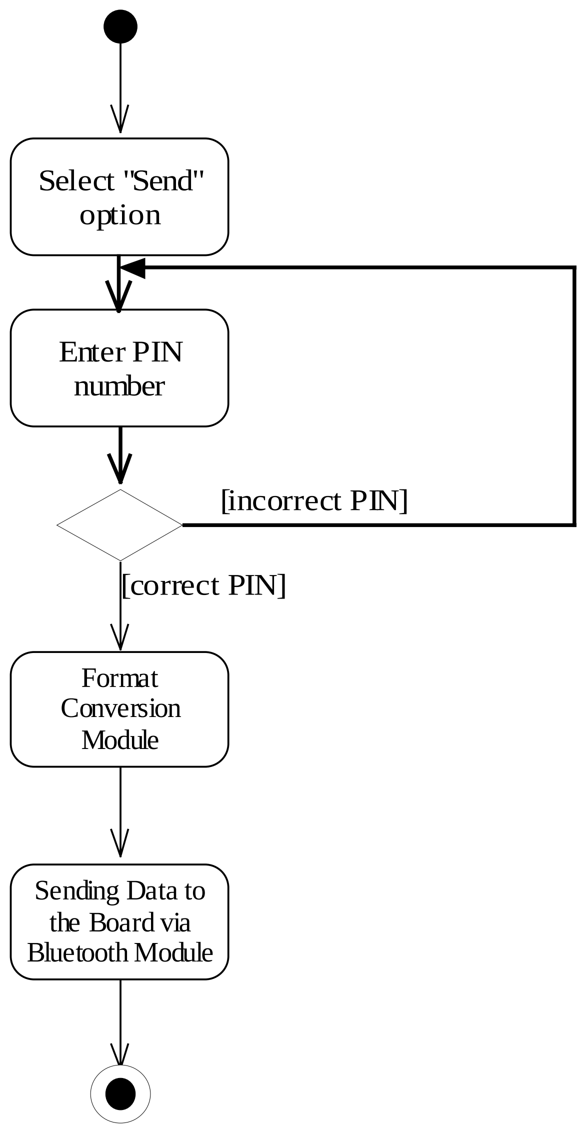
- Activity Diagram for Entering Message:



- Activity Diagram for Saving a Slide Show:



- Activity Diagram for Sending a Slide Show:



2.1.2 Format Conversion Module:

When the user selects the images to be displayed and specifies the details of the slide show, we will create a file in hex format whose contents will include the pixel data for each image and the slide show configuration details. Format conversion module is responsible for the creation of this file.

This module will create a hex file, which when sent to the board, will be capable of telling the board the user preferences via our interface. By uploading this file to the board, as described in the Bluetooth Module, data in it can be processed easily.

As we have described in the “User Interface Design”, our system allows the user to upload several images, their appearance order and time duration for each image. Moreover, system also lets the user make modifications about time duration and order of an image and new image uploading instead of an existing image in the slide show.

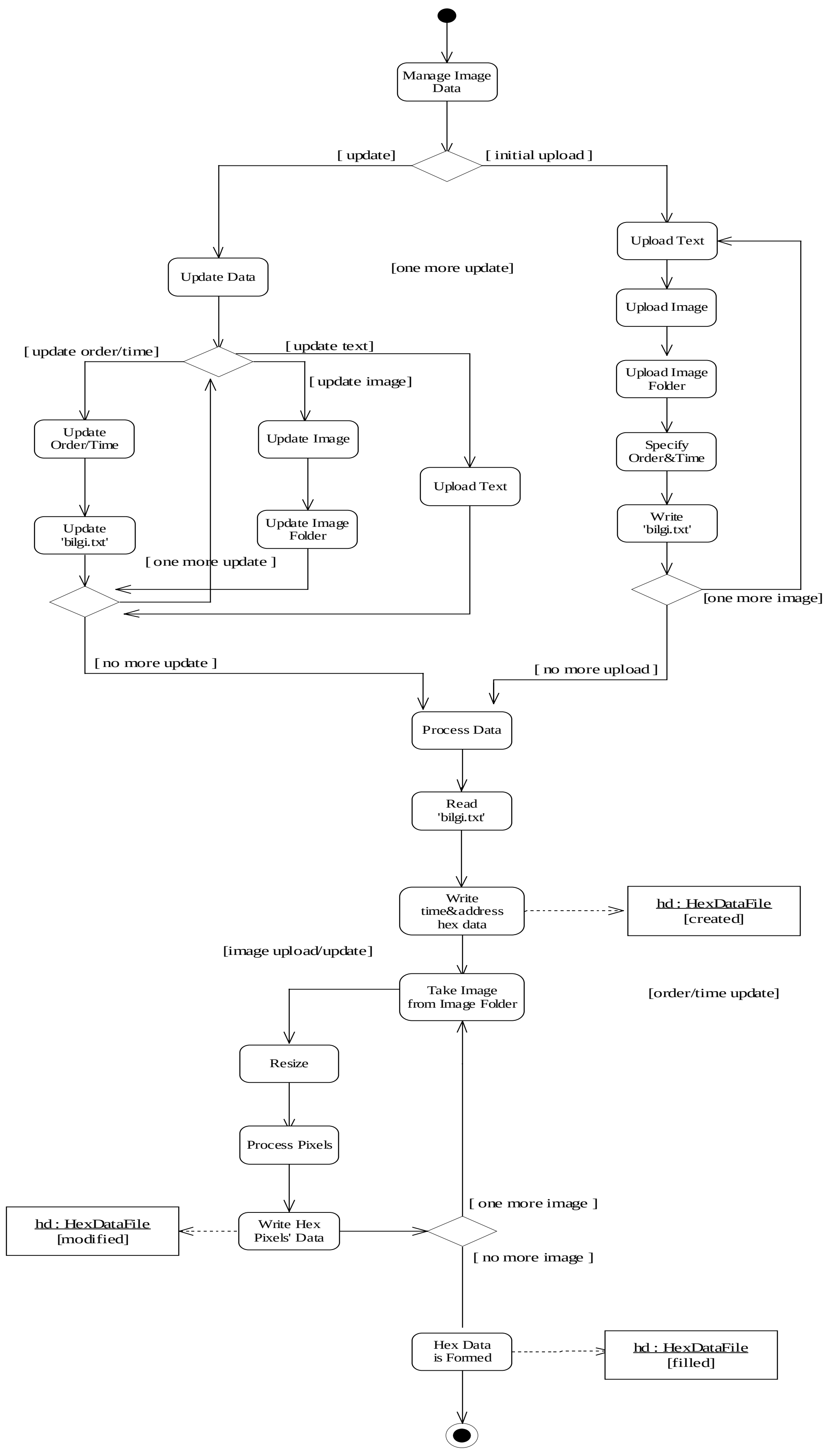
The hex format file that will be created contains structured data that specifies the process of the user. Each line in the file will contain a memory address followed by data, which is going to be written to that memory address. The basic structure of the hex file is like below;

- First line specifies the number of images that the user has uploaded,
- Next ten lines specify the time duration of the images selected by the user (As explained earlier, the user can select up to ten images. If the user has selected less images, the time duration of the remaining slides is written as “0”.)
- After that, if the number of images is different than 0, file continues with the image order and a hex stream that is identical to its canvas for every image. When the pixel data of the current image is written completely, “FF” will be written to the next line indicating the completion of the current image.

For creating this file, we decided to use object-oriented classes that are implemented by Java. At the beginning of our process, the header of the image files will be read and the basic data about them like their dimensions, resolutions and types will be obtained.
This process will support only .JPEG and .GIF formats. Therefore, we only allow the user to upload an image in one of these formats.

In the process of forming the identical hex streams of image files, the images will be resized first. The reason of resizing process is obtaining an 800*600 pixel*pixel image for our VGA port. By this way, we can centralize the scene of the image regardless of its own dimensions. One of the other advantages of this process is giving the user the chance of exchanging an image with another one even it has different dimensions. Additionally, with fixed sizes, each image will occupy same memory space and we will be able to determine the beginning address of each image easily which will help us a lot during slide show transitions in the VGA process.

The next step of format changing process is reading every pixel of the image files and assigning a value between 0-512 for its color. (Our XSA board VGA port processes 512 colors at maximum.) According to our research, we found that we should use 16-bit RGB value whose 3 bits will be used for red, 3 bits for green and 3 bits for blue components for a pixel for a considerable resolution on the monitor. Thus we will use 9 bits (2 bytes) for color identification.
According to our scaling specifications, we concluded that the hex format file that includes one image would have a size of approximately 960 KB (2*800*600 Byte).
By this module design, we aim to obtain a more flexible system for the user. The following activity diagram illustrates the process of this module:



Manage Image Data
[update]
[initial upload]
Update Data
[one more update]
Upload Text
Upload Image
Upload Image Folder
Specify Order&Time
Write 'bilgi.txt'
[one more image]
[update order/time]
[update text]
[update image]
Update Order/Time
Update Image
Upload Text
Update 'bilgi.txt'
Update Image Folder
[one more update]
[no more update]
[no more upload]
Process Data
Read 'bilgi.txt'
Write time&address hex data
hd : HexDataFile [created]
[image upload/update]
Take Image from Image Folder
[order/time update]
Resize
Process Pixels
Write Hex Pixels' Data
hd : HexDataFile [modified]
[one more image]
[no more image]
Hex Data is Formed
hd : HexDataFile [filled]

2.1.3. Sending Data to the Board via Bluetooth Module:

This module is responsible for sending the hex file created in the “Format Conversion” module and the txt file containing the message about the details of the event that will be sent to Bluetooth devices created by the “File Uploading” module to the Bluetooth Evaluation Board.

Because we use Java for the user interface design, we decided to use the Java APIs in order to be able to embed the bluetooth operations in our user interface code. Mainly we will create a “Java Client Application”. This application will search for devices and when the BlueRadios Evaluation board is discovered, it will search for the services proived by this Board. Using the “Serial Port” service of the board, the application will establish a connection and start data transfer. The details of this process is discussed below.

Bluetooth System Requirements

The underlying Bluetooth system upon which the Java APIs will be built must meet certain requirements:

- The underlying system must be "qualified," in accordance with the Bluetooth Qualification Program, for at least the Generic Access Profile, Service Discovery Application Profile, and Serial Port Profile.
- The system must support three communication layers or protocols as defined in the 1.1 Bluetooth Specifications, and the implementation of this API must have access to them: Service Discovery Protocol (SDP), Radio Frequency Communications Protocol (RFCOMM), and Logical Link Control and Adaptation Protocol (L2CAP).
- The system must provide a Bluetooth Control Center (BCC), a control panel much like the application that allows a user or OEM to define specific values for certain configuration parameters in a stack.

Packages

The Java APIs for Bluetooth define two packages that depend on the CLDC javax.microedition.io package:

- javax.bluetooth: core Bluetooth API

  This package provides classes used in device management, device and service discovery, and obtaining a connection.

- javax.obex: APIs for the Object Exchange (OBEX) protocol

Application Programming

The anatomy of a Bluetooth application has certain parts. The client application is composed of five main tasks: stack initialization, device management, device discovery, service discovery, and communication.

1. Stack Initialization

The Bluetooth stack is responsible for controlling the Bluetooth device, so we need to initialize the Bluetooth stack before we can do anything else. The initialization process comprises a number of steps whose purpose is to get the device ready for wireless communication. Unfortunately, the Bluetooth specification leaves this process to vendors, and different vendors handle stack initialization differently. On one device, it may be an application with a GUI interface, and on another it may be a series of settings that cannot be changed by the user.

There are commercial software products that provide stack initialization process. These products can be analyzed in two categories. The first type of products build everything on top of the bluetooth hardware. However these products do not support all the Java APIs packages and no other application can easily access the bluetooth communication services. The products of the second type run on top of external stacks like Microsoft Service Pack2. These products work by JNI(Java Native Interface) calls to the stack.

After analyzing certain software products, we decided to use BlueCove (http://code.google.com/p/bluecove/") in our project since it is free. It supports javax.bluetooth package and Serial Port Profile. However, BlueCove supports only Microsoft Service Pack2 (or newer) stack.

BlueCove acts as a bridge between the Java application and the bluetooth hardware. Microsoft stack API is a socket-style API that allows only C-based applications to operate. BlueCove enables an integration between the Java applications and Microsoft API via JNI.

In order to use BlueCove, bluecove.jar files are added to the classpath which include the javax.bluetooth package. During installation intelbth.dll file is generated by running the C codes that come with BlueCove. This dll file is put into the System32 directory which enables the integration between the Microsoft Stack and the Java application. Afterwards, the application is written using the classes defined in javax.bluetooth package.

2. Device Management

The Java Bluetooth APIs contain the classes LocalDevice and RemoteDevice, which provide the device-management capabilities defined in the Generic Access Profile. LocalDevice depends on the javax.bluetooth.DeviceClass class to retrieve the device's type and the kinds of services it offers. The RemoteDevice class represents a remote device (a device within a range of reach) and provides methods to retrieve information about the device, including its Bluetooth address and name. The following code snippet retrieves that information for the local device:

```
// retrieve the local Bluetooth device object
LocalDevice local = LocalDevice.getLocalDevice();
// retrieve the Bluetooth address of the local device
String address = local.getBluetoothAddress();
// retrieve the name of the local Bluetooth device
String name = local.getFriendlyName();
```

When a remote device is discovered, the same information about it can be obtained as below:

```
// retrieve the device that is at the other end of
// the Bluetooth Serial Port Profile connection,
// L2CAP connection, or RFCOMM connection
RemoteDevice remote =  RemoteDevice.getRemoteDevice(
javax.microedition.io.Connection c);
// retrieve the Bluetooth address of the remote device
String remoteAddress = remote.getBluetoothAddress();
// retrieve the name of the remote Bluetooth device
String remoteName = local.getFriendlyName(true);
```

3. Device Discovery

Because wireless devices are mobile they need a mechanism that allows them to find other devices and gain access to their capabilities. The core Bluetooth API's DiscoveryAgent class and DiscoveryListener interface provide the necessary discovery services.

A Bluetooth device can use a DiscoveryAgent object to obtain a list of accessible devices, in any of three ways:

The DiscoveryAgent.startInquiry method places the device into an inquiry mode. To take advantage of this mode, the application must specify an event listener that will respond to inquiry-related events. DiscoveryListener.deviceDiscovered is called each time an inquiry finds a device. When the inquiry is completed or canceled, DiscoveryListener.inquiryCompleted is invoked.

If the device doesn't wish to wait for devices to be discovered, it can use the DiscoveryAgent.retrieveDevices method to retrieve an existing list. Depending on the parameter passed, this method will return either a list of devices that were found in a previous inquiry, or a list of *pre-known devices* that the local device has told the Bluetooth Control Center it will contact often.

In our application, we will use the first method. When a device is discovered, we will check if it is BlueRadios Evaluation Board or not. If so, we will pass to the next stage, service discovery.

4. Service Discovery

Once the local device has discovered at least one remote device, it can begin to search for available *services* - Bluetooth applications it can use to accomplish useful tasks. Because service discovery is much like device discovery, DiscoveryAgent also provides methods to discover services on a Bluetooth server device, and to initiate service-discovery transactions. Note that the API provides mechanisms to search for services on remote devices, but not for services on the local device.

The BlueRadios Evaluation Board offers “Serial Port” service, so will search for this service. When this service is found, a *ServiceRecord (included in javax.bluetooth)* object is created and this object is used in establishing a connection.

5. Communication

For a local device to use a service on a remote device, the two devices must share a common communications protocol. So that applications can access a wide variety of Bluetooth services, the Java APIs for Bluetooth provide mechanisms that allow connections to any service that uses RFCOMM, L2CAP, or OBEX as its protocol. If a service uses another protocol (such as TCP/IP) layered above one of these protocols, the application *can* access the service, but only if it implements the additional protocol in the application, using the CLDC Generic Connection Framework.

BlueCove supports RFCOMM protocol and BlueRadios Evaluation Board offers “Serial Port” service, thus we will use this service in order to transfer data from our application program to the bluetooth board.

Serial Port Profile

The RFCOMM protocol, which is layered over the L2CAP protocol, emulates an RS-232 serial connection. The Serial Port Profile (SPP) eases communication between Bluetooth devices by providing a stream-based interface to the RFCOMM protocol. Some capabilities and limitations to note:

- Two devices can share only one RFCOMM session at a time.
- Up to 60 logical serial connections can be multiplexed over this session.
- A single Bluetooth device can have at most 30 active RFCOMM services.
- A device can support only one client connection to any given service at a time.

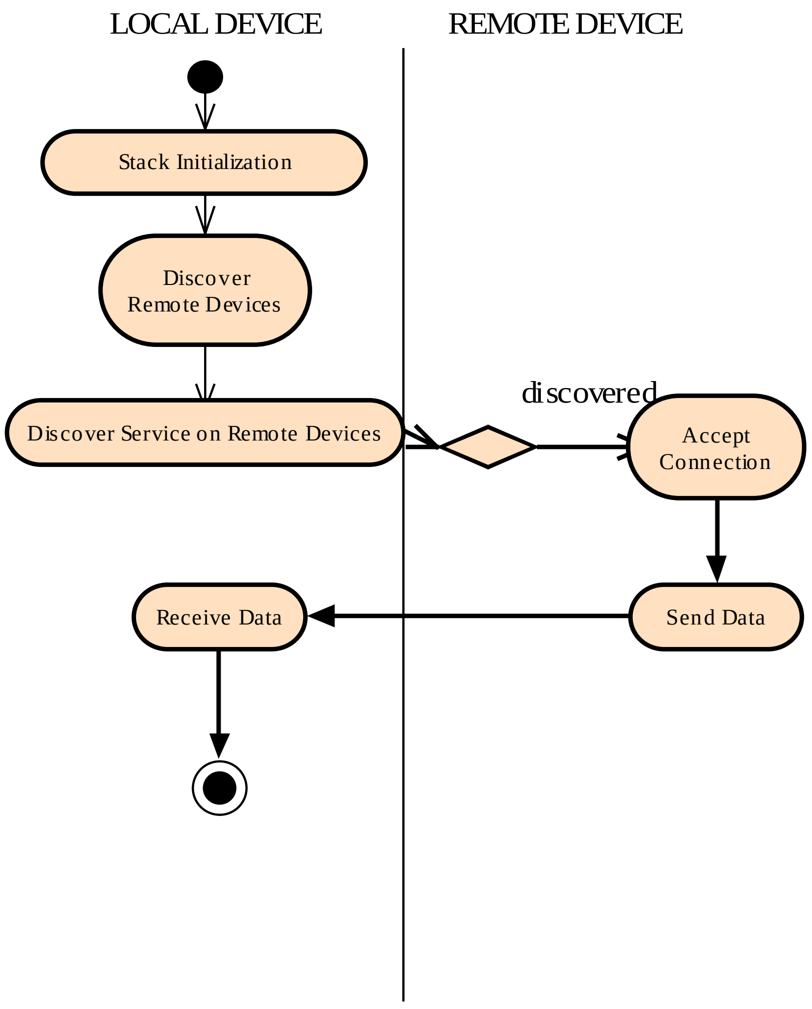
For a client to communicate using the Serial Port Profile, each must perform a few simple steps.

To set up an RFCOMM connection to a server the client must:

1. Initiate a service discovery to retrieve the service record
2. Construct a connection URL using the service record
3. Open a connection to the server
4. Send and receive data to and from the server

When service discovery step is taken in our application, the bluetooth board will appear as “BlueRadios Serial Port” and we will make use of this service in order to establish a connection.

When connection is done successfully, we will open a DataOutputStream and start processing the hex file and the txt file that need to be transferred to the bluetooth board. We will read each file line by line and write the current line to the stream until all data in the files have been transferred. The following activity diagram illustrates the process of this module:



2.1.4. Register Process Module:

The Memory Process Module will deal with sending the information that will come from the parallel port of the board to the SDRAM of the board and vice versa. In **XSA-3S1000** Board, most of the Parallel Port inputs and outputs are directly connected to the CPLD Part of the board. The CPLD will be programmed so that it acts as an interface between the Parallel Port and the FPGA so it can pass bitstream from the Parallel Port to the FPGA and vice versa.

The following steps will be held while sending the information from Parallel Port to FPGA:

1. The information from the Parallel Port will be sent to the FPGA through the CPLD.

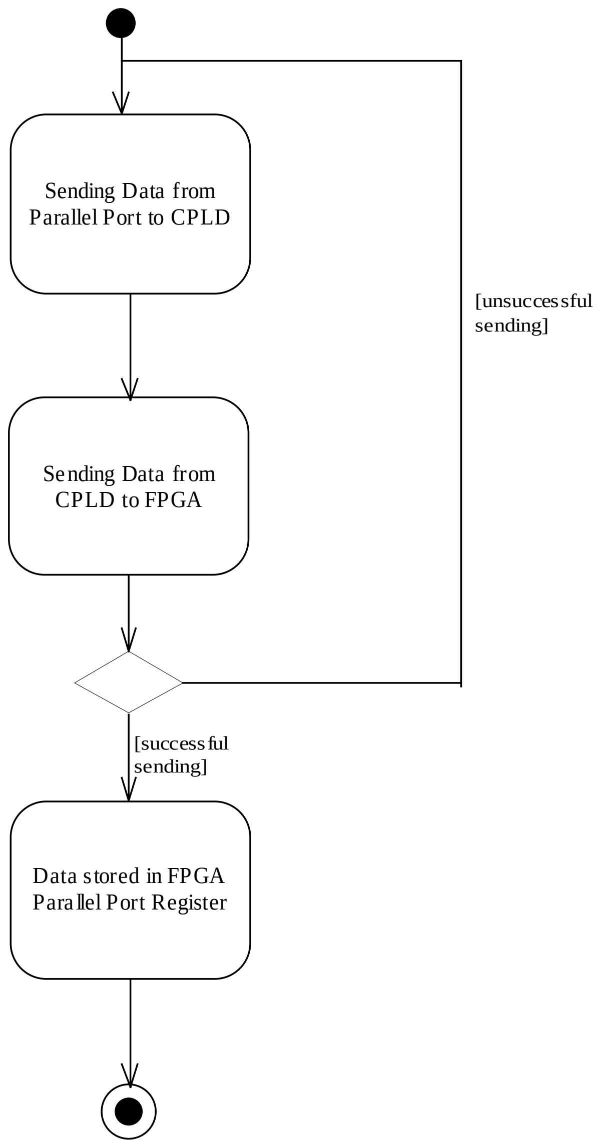
2. The FPGA will store the retrieved information from the CPLD to one of its registers that we call Parallel Port Register (i.e. the register that we will store the information gathered from Parallel Port).

Usage Note: After these steps are completed Parallel Port Register content can be sent and stored in the SDRAM.

In order to send data from the parallel port of the XSA board to the FPGA, data will be put on the D0, D1, … , D7 data pins of the parallel port. This data will pass through the CPLD and end up in A15-A8 Flash address lines. Finally the data will arrive to the FPGA. In FPGA this data will be stored in a register of length 8 bits. This data will be combined with the next coming 8 bits of data to produce a data bus of 16 bits wide since the data bus in our applications is 16 bits wide. Similarly the data received will be combined with two 8-bits of data to produce an address information of 24 bits wide. Because in our hex file, each line contains an address followed by data that is going to be written to that address, the first 24 bits of data received (We represent host addresses as 24 bits address bus.) will be the address so this data will be stored in the address bus of the Bluetooth Controller which is connected to the address bus of the SDRAM controller. Then the next 16 bits of data will be received, a byte at a time. This data bus will be connected to the data bus of the SDRAM in the Bluetooth Controller host side. This way, a write operation will be requested and the 16 bits data will be written to the SDRAM address specified by the address bus. After writing the hex file contents to the SDRAM, we will start writing the txt file contents. Since the beginning and ending addresses of the hex file content is predetermined, we will write the txt file content starting from the end of the hex file content. Since we know the number of images and the bytes contained in each image, we will understand when we have read all the image data and the txt data is in progress next. Thus when we receive

16 bits of data, a byte at a time, from the parallel port, we will increment the current address by one and put the received data to the data bus of the Bluetooth Controller. This way, SDRAM Controller will get both the address and data bus contents and the write operation will be done.

The following activity diagram illustrates this process:


Sending Data from Parallel Port to CPLD
[unsuccessful sending]
Sending Data from CPLD to FPGA
[successful sending]
Data stored in FPGA Parallel Port Register

While sending information from the FPGA to the Parallel Port, the module will follow the following steps:

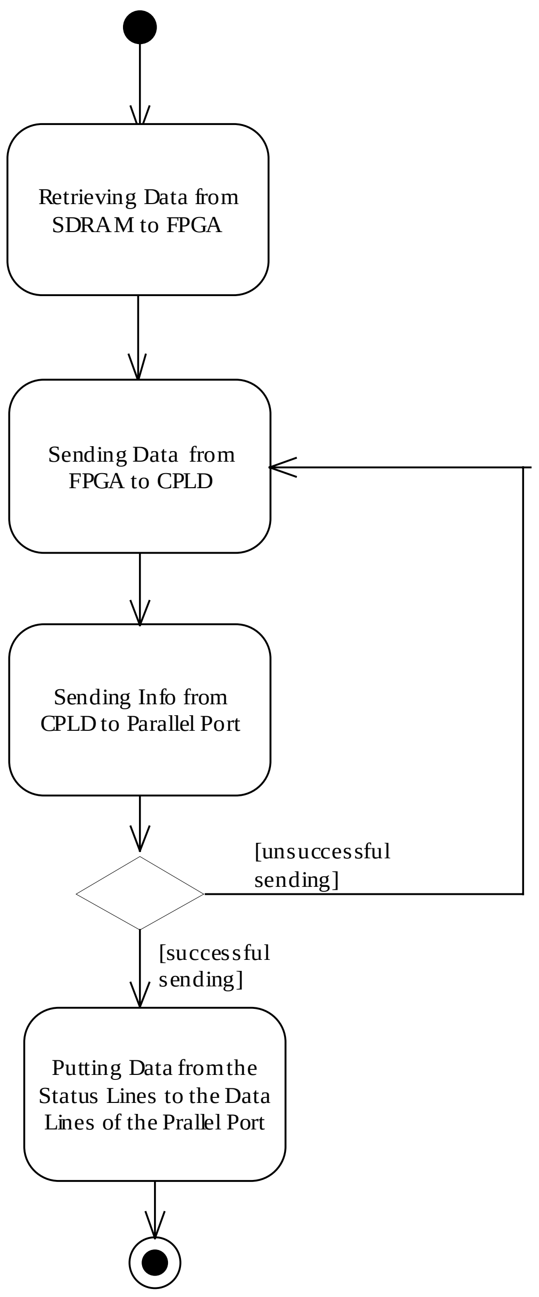
1. The information from SDRAM will be retrieved by FPGA in order to be sent to the Parallel Port.

2. The CPLD will be programmed so that it acts as an interface between the Parallel Port and the FPGA so it can pass bit streams from the FPGA to the Parallel Port and vice versa.

3. The information from the FPGA will be sent to the Parallel Port throurgh CPLD.

When FPGA wants to send data to the parallel port, it drives the A2, A1, and A0 address lines of the Flash. This in turn drives the S5, S4, S3 status lines of the parallel port and thus data comes to the parallel port. We will assign the values of these status lines to the D7, D6, and D5 data lines of the parallel port. We will put a 0 and 1 to the D4 data line in turns. This data will arrive to the RB7-RB4 pins of the pic. Because the RB4 pin have a different value with each coming data, the RB7-RB4 interrupt (This interrupt is raised when one of the RB7-RB4 pins chages value.) is guaranteed to be raised. Thus the pic will be notified every time we send data to the parallel port of the XSA board.

The following diagram explains this process:


Retrieving Data from SDRAM to FPGA
Sending Data from FPGA to CPLD
Sending Info from CPLD to Parallel Port
[unsuccessful sending]
[successful sending]
Putting Data from the Status Lines to the Data Lines of the Prallel Port

2.1.5. Retrieving Data from User via Bluetooth Evaluation Kit Module:

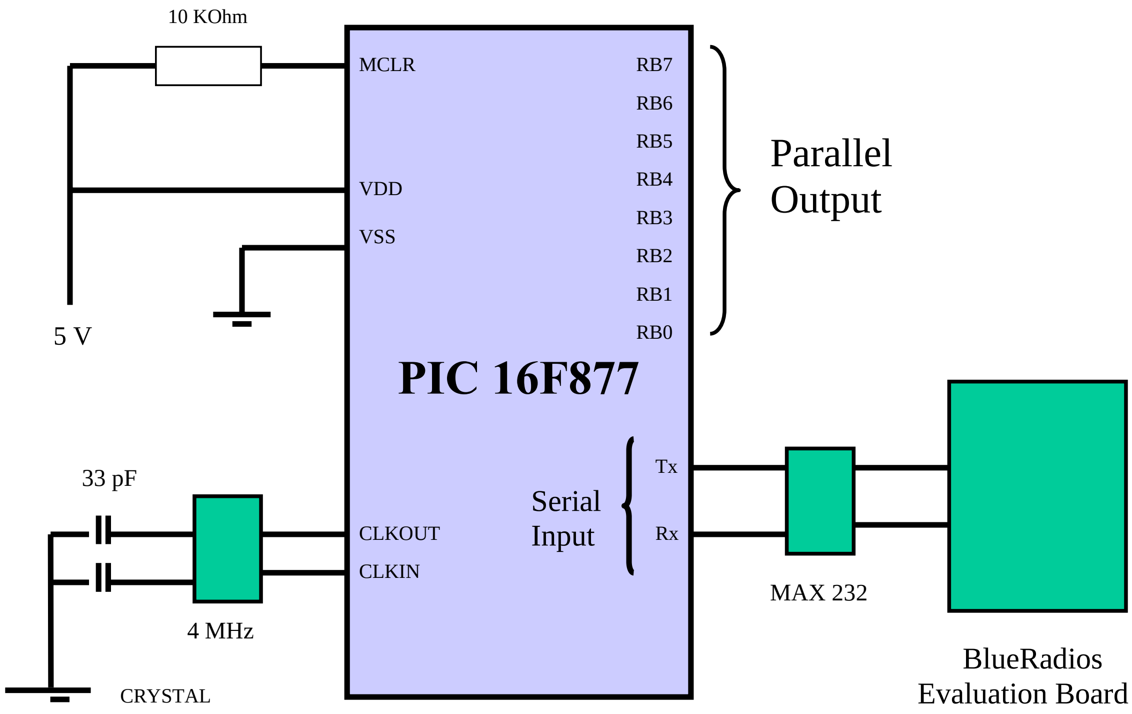
As we have described before, because our XSA board doesn't have a serial port, we need to change serial data to parallel data for sending user data gathered by the user interface to the XSA board. Similarly, we need to convert parallel data to serial data for sending messages, whose contents will be stored in the SDRAM of the XSA board, to Bluetooth devices.

According to our design principles and search on net, we decided to use an additional pic that we will design for this purpose. This pic is going to have a very simple design. It will contain a parallel port, a serial port and an energy supply.

By this basic design, after canalizing the bluetooth data to this new additional board by using its serial port, data will be sent to our main board by using the parallel port of this additional board. As you can guess, a similar process can be applied for transforming data from our main board's parallel port to our additional board's serial port.

For the new board design we will use PIC 16F877 since we are familiar to this pic from the Embedded Systems course. The design of this new additional board contains 2 main steps, hardware and software designs.

For hardware design we will first create the circuit of the board that basically contains a structure shown below:



In this diagram there are some components that need to be explained:

- Cyristal: Crystal is responsible for the generation of two clock inputs to the pic. (CLKOUT and CLKIN) These clock inputs are necessary for the proper working of the pic.
- MAX 232: We have to convert -12V and 12V RS232 logic levels to 0V and 5V in order to process RS232 signals correctly in the pic. MAX 232 simply does this conversion.
- MCLR: This is the reset to the pic. There is a 10 Kohm resistor which is connected to ground through a switch. When the switch is closed (The reset button on the pic is pushed.), 0V will be supplied to the MCLR pin and the pic will be reset.

After the hardware design step, for creating PCB(Printed Circuit Board) of our board, we are going to follow basic steps that include some chemical and structural processes. Steps are;

- Printing out to circuit on a photograph paper by a laser printer,
- Pasting the print out on a copper plate by ironing,
- Waiting it in a chemical solution for a specified time duration,
- And finally combining serial port, parallel port and power supply with our copper plate by soldering.

Once the design of the pic is finished, we will start programming the pic. For this purpose, we are going write C code in which we will take data from serial port registers and put them into parallel port registers and vice versa. For the C code we will write, we will use a compiler called Source BoostC. BoostC is a C compiler that works with PIC16. This is an ANSI-C compatible compiler that supports signed data types, structures, unions, pointers etc. When we build our C code with this compiler, we will have an asm file. At this step, we will use MPLAB to generate a hex file from the asm file and load this hex file to the pic.

By this new additional board, we will easily provide the connection between Bluetooth devices and our main board. The process of retrieving data from the bluetooth evaluation board using this new pic is described below:
For receiving data from Bluetooth device, we are going to use the serial port. When data is sent through the serial port to the pic, the 1 bit info will be put in RCREG register and the board will take an interrupt by the RCIF flag bit. At every interrupt, we will take the data from RCREG and put it in a register we choose. After taking 8 different interrupts, we will achieve collecting an 8-

bit data in the register which is convenient for sending through the parallel port to XSA. Then we will put this data to PORTB and it will be automatically sent to XSA board through parallel port.

First we should enable interrupts in our software. This is done by setting GIE and PEIE of INTCON register. Then the following steps will be taken:

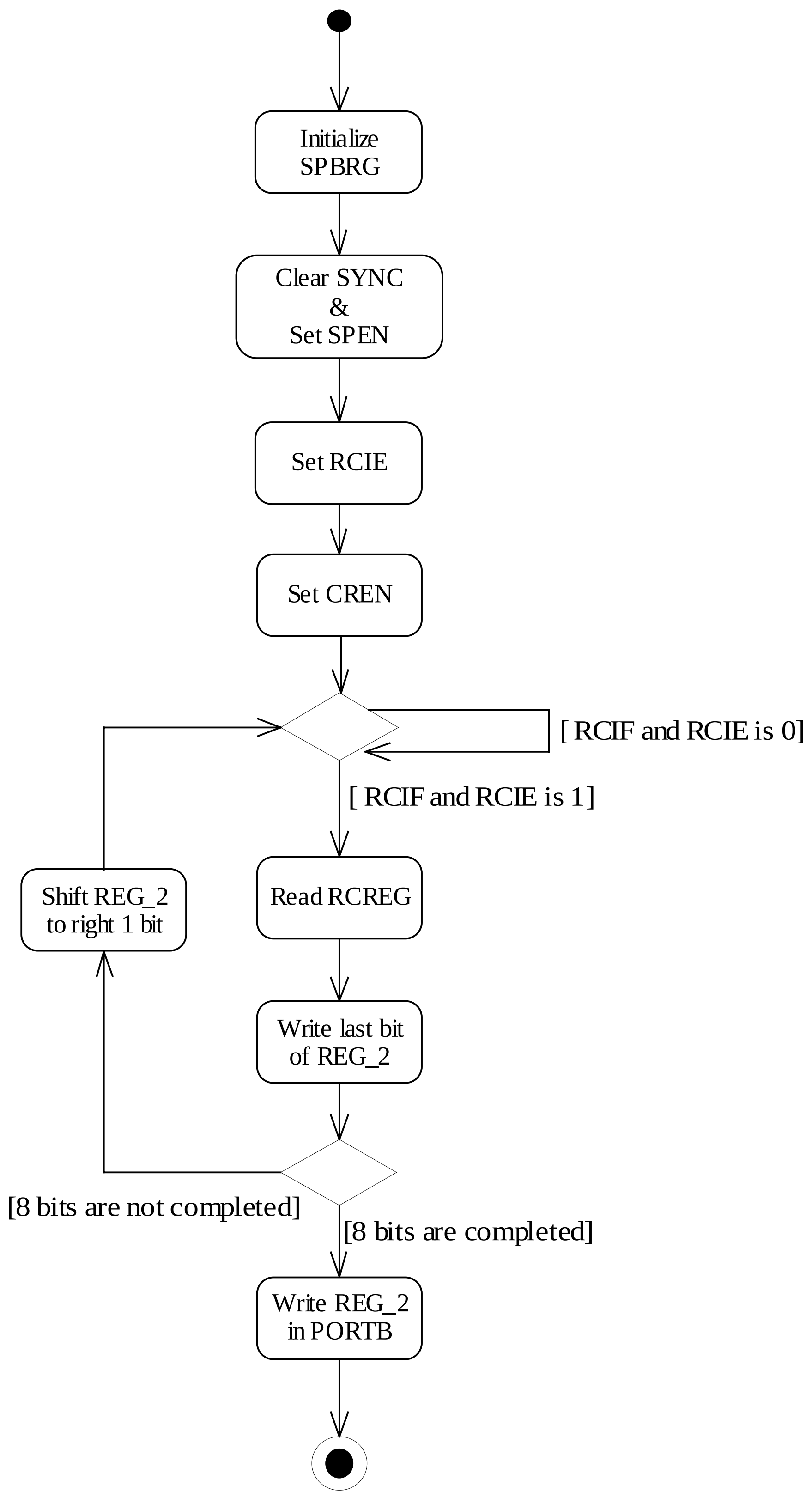
1. Initializing the SPBRG register for baud rate.
2. Enabling asynchronous serial port by clearing bit SYNC and setting SPEN.
3. Setting RCIE for interrupt enabling.
4. Enabling reception by setting CREN.
5. If RCIF and RCIE is 1, an interrupt occurs. After we should;
    a. Read 1 bit data from RCREG and write it in another register REG_2.
       (At every write process we will shift the data in register by one bit to the right.)
6. After 8 interrupts, the data which is convenient for parallel port is ready.

For transforming data to the parallel port of XSA, steps are;

1. Reading data in REG_2.
2. Writing it in PORTB.
3. Clearing REG_2 register.

By this way data in PORTB will be automatically send to XSA's parallel with converter board's hardware design.

The following activity diagram illustrates this process:


Initialize SPBRG
Clear SYNC & Set SPEN
Set RCIE
Set CREN
[RCIF and RCIE is 0]
[RCIF and RCIE is 1]
Read RCREG
Shift REG_2 to right 1 bit
Write last bit of REG_2
[8 bits are not completed]
[8 bits are completed]
Write REG_2 in PORTB

2.1.6. Sending Data to User via Bluetooth Evaluation Kit Module:

As we know, XSA board sends the information through its parallel port by 3-3-2 bits. Because it only sends three bits data at each step, we decided to use RB4-RB7 interrupt of 16F877 for detecting whether the info comes or not. We decided to add one more bit info in front of 3 bits information from XSA. This bit will have a value of 0 and 1 interchagebly so that the RB4-RB7 interrupt is generated with each coming data. (This interrupt is generated when one of the RB4-RB7 pins changes value.) Thus detection of the new info is guaranteed. At each send operation from XSA, we put 0 if the previous transform is done by 1 and we put 1 if the previous transform is done by 0. By this way, although the three bit info may be the same with the previous transform we can detect it. When data is sent through parallel port of XSA, converter board will raise an interrupt because of the new data in RB4-RB7.

After that point, we will read the new data, and collect them in a new register that we choose. After three different interrupts, we will collect the 8-bit data from the parallel port. As a final step, we are going to send the data in the register bit by bit using USART asynchronous receiver to the Bluetooth serial port.

In order to carry these operations, first we should enable interrupts in our software. This is done by setting GIE and PEIE of INTCON register.

For receiving data through the parallel port of XSA, steps are;

1. Setting RBIE of INTCON for enabling RB7-RB4 interrupt.
2. Defining RB7-RB4 pins as input.
3. Controlling RBIF flag bit of INTCON for determining if any value change occurs in RB7-RB3 bits.
4. If RBIF is 1, an interrupt occurs. After that we should;
    a) Read data from R4-R6 and write it in another register REG_1.
    b) Clear flag bit RBIF.
       (At every write process we will shift the data in register. For the first and second write we shift it 3 bits and for the third write shift it 2 bits to the right.)

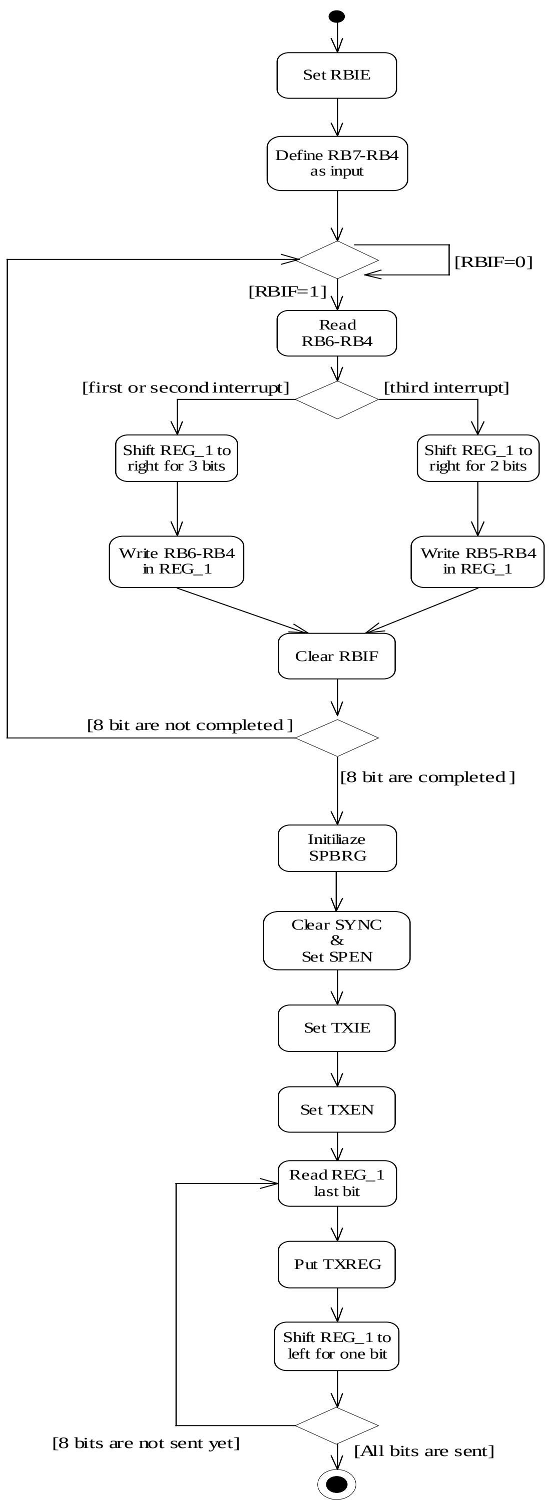
After three interrupts, we will transform it one by one through the serial port.

For transferring data through the serial port, steps are;

1. Initializing the SPBRG register for baud rate.
2. Enabling asynchronous serial port by clearing bit SYNC and setting SPEN.
3. Setting TXIE for interrupt enabling.
4. Setting TXEN for transferring.
5. Reading the last bit of REG_1, putting it in TXREG and shifting data in REG_1 one bit to the left.

After 8 transform we achieve sending received data from XSA's parallel port to Bluetooth's serial port.

The following activity digram illustrates this process:



Set RBIE
Define RB7-RB4 as input
[RBIF=0]
[RBIF=1]
Read RB6-RB4
[first or second interrupt]
[third interrupt]
Shift REG_1 to right for 3 bits
Shift REG_1 to right for 2 bits
Write RB6-RB4 in REG_1
Write RB5-RB4 in REG_1
Clear RBIF
[8 bit are not completed]
[8 bit are completed]
Initiliaze SPBRG
Clear SYNC & Set SPEN
Set TXIE
Set TXEN
Read REG_1 last bit
Put TXREG
Shift REG_1 to left for one bit
[8 bits are not sent yet]
[All bits are sent]

2.1.7. VGA Process Module:

The VGA module provides the functionality of displaying images on a monitor. This module consists of the following basic parts:

- ✓ Generating vertical and horizontal sync signals which indicate the end of a frame and line respectively.
- ✓ Reading data from the memory to the pixel buffer
- ✓ Putting data from the pixel buffer to the pixel register and shifting the pixel register content so that the current pixel is in the least significant position
- ✓ Color mapping of the current pixel

The image(s) uploaded by the user will be stored in the SDRAM of the board. The screen width (We will use w=800 pixels per line.) and the screen height (We will use h=600 lines per frame.) will be constant variables assigned by us. For images which have less pixels per line or less lines per frame, extra pixels will be blanked. Images with more pixels will be resized. Since the screen width and height will be constant, for each image we will show the same number of pixels per line and same number of frames per line, either blanked or not. In other words, for each image we will store information of equal number of pixels. The pixel width of our system will be 16 bits and we will use 3 of these bits for the red color component, 3 for the green color component , and 3 for the blue color component. Knowing the number of pixels and the width of a pixel, we will be able to determine how many memory words each image will occupy in the SDRAM. We will begin storing the images in the SDRAM, from an address again specified by us. Since we will know the starting address and the size of each image, we will be able to determine the starting and ending addresses of each image stored in the memory. We will use this information for the transitions between the images in a slide show manner.

A horizontal sync signal indicates the end of a line. The period of horizontal scan line is calculated by the formula:

*horizontal scanline period = (number of pixels per line * CLK_DIV)/frequency + 6µs*

CLK_DIV in this formula is a clock divisor used to adjust the frequency. Our board has a fixed frequency of 100 MHz and we will use a CLK_DIV of 2 to obtain a frequency of 50 MHz. Putting the values of the variable in the formula for our system, we obtain a horizontal scanline period of 22 µs. Of this time interval 16 µs is active, meaning a line of pixels is shown. The remaining 6 µs consists of the front porch (1 µs), inserted before the sync signal, back porch (1 µs) inserted after the sync signal and the sync signal (4 µs) itself. Since the screen height (h), in other words lines per frame, is known we can calculate the time period of a frame from the formula:

*frame-period = (number of lines per frame * CLK_DIV) / horizontal frequency + 1424 µs*

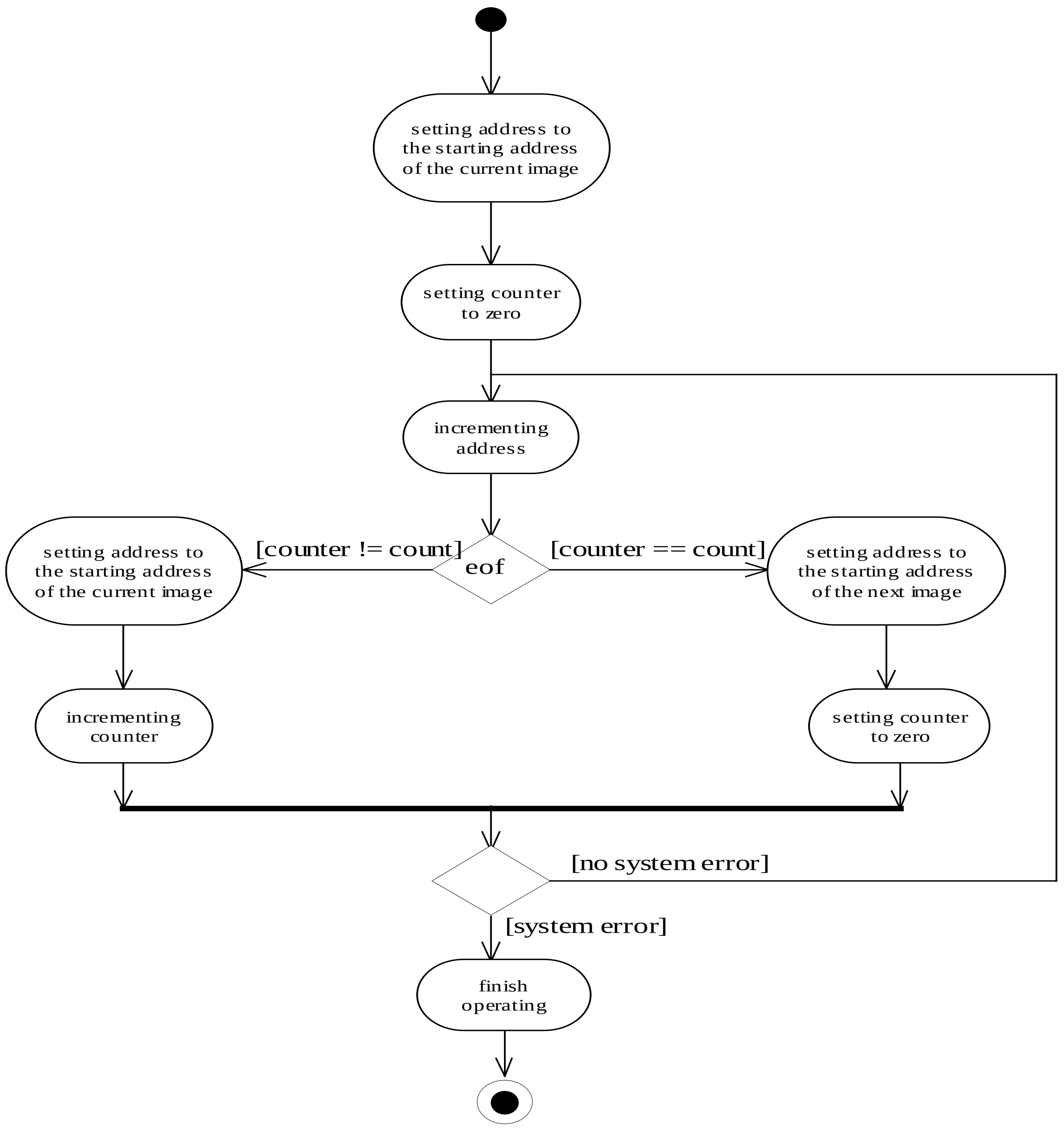
Inserting the values of the variables for our system we get a frame period of 14.624 ms. Front porch occupies 0.34 ms of this period, back porch occupies 1.02 ms, and the sync pulse occupies 0.064 ms.

For the horizontal and the vertical scanlines, the pixels should be blanked when the horizontal and vertical sync signals are generated to indicate the end of a line or a frame. For this purpose we will have a counter and increment it every clock cycle. The period of the scanlines can be calculated in terms of clock cycles by just multiplying the values found above with the system frequency, 50 MHz. For example, a horizontal scanline is active for 16 µs, which is equal to 800 clock cycles. When the counter value reaches 800, we should start generating the horizontal sync signal for 50 MHz * 4 µs = 200 clock cycles. Meanwhile we should blank the pixels for the time period when the scanline is not active, which is 22-16 = 6 µs (front porch + signal + back porch). Thus when the counter reaches 800 we should start blanking the pixels until the counter reaches 800 + 6*50 = 1100. After that the counter is reset to zero. Similarly, the vertical scanline is active for 13.2 ms, which is equal to 660000 clock cycles. Thus we should have another counter and when the value of this counter reaches 660000 we should start generating the vertical sync signal. The vertical scanline is not active for 14.624-13.2 = 1.424 ms, which is equal to 71200 clock cycles. Thus we should start blanking the pixels when the counter reaches 660000 and continue blanking until the counter becomes 731200, then the counter is reset to zero.

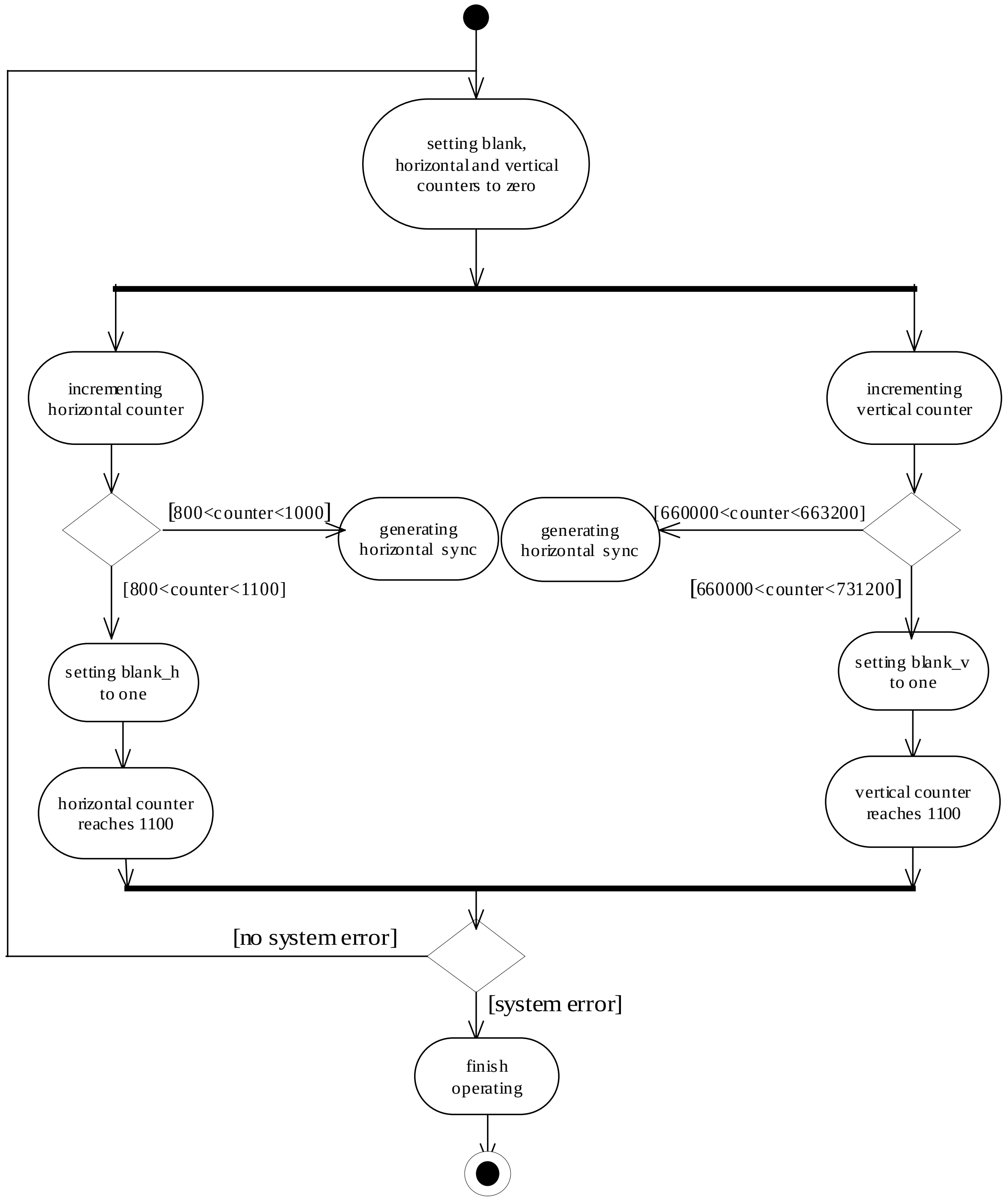
The user will specify the time interval for which each image will be displayed. This information will arrive to the board together with the images via bluetooth and we will store this information in registers. Let's assume that the first image will be shown for t seconds. This means that we will show the first frame for *count = t / frame-period* times. Thus, we will keep another counter and increment the value of the counter every time we start a new frame of the same image. While showing the same image, the value in the counter will be smaller than count. Meanwhile, at the end of a frame the memory address of the pixels will be set back to the starting address of the same image. Once the counter reaches the value *count*, we will reset the counter to zero and begin showing the next image, which also means that the memory address will now point to the starting address of this next image. We will repeat this process for all the images. When the last image in the slide show is displayed for the specified amount of time, we will begin showing the slide show again.

When displaying an image, we will read the pixel data from the memory to a pixel buffer. This buffer will generate two signals, full and empty, indicating whether the buffer is full or empty. When the buffer becomes empty, new data is read from the memory. The pixel data in the buffer is put into a pixel register. A memory word is 16 bits and we will store pixels as 8 bits. This means that the pixel register will contain two registers at a time. The content of the pixel register is shifted so that the current pixel is in the least significant position. Once the current pixel is at the correct position, the r , g, and b components are read and sent to the digital-to-analog-converter of the vga port where the color information is extracted and the pixel is shown.
The following activity diagrams illustrate the process of this module:

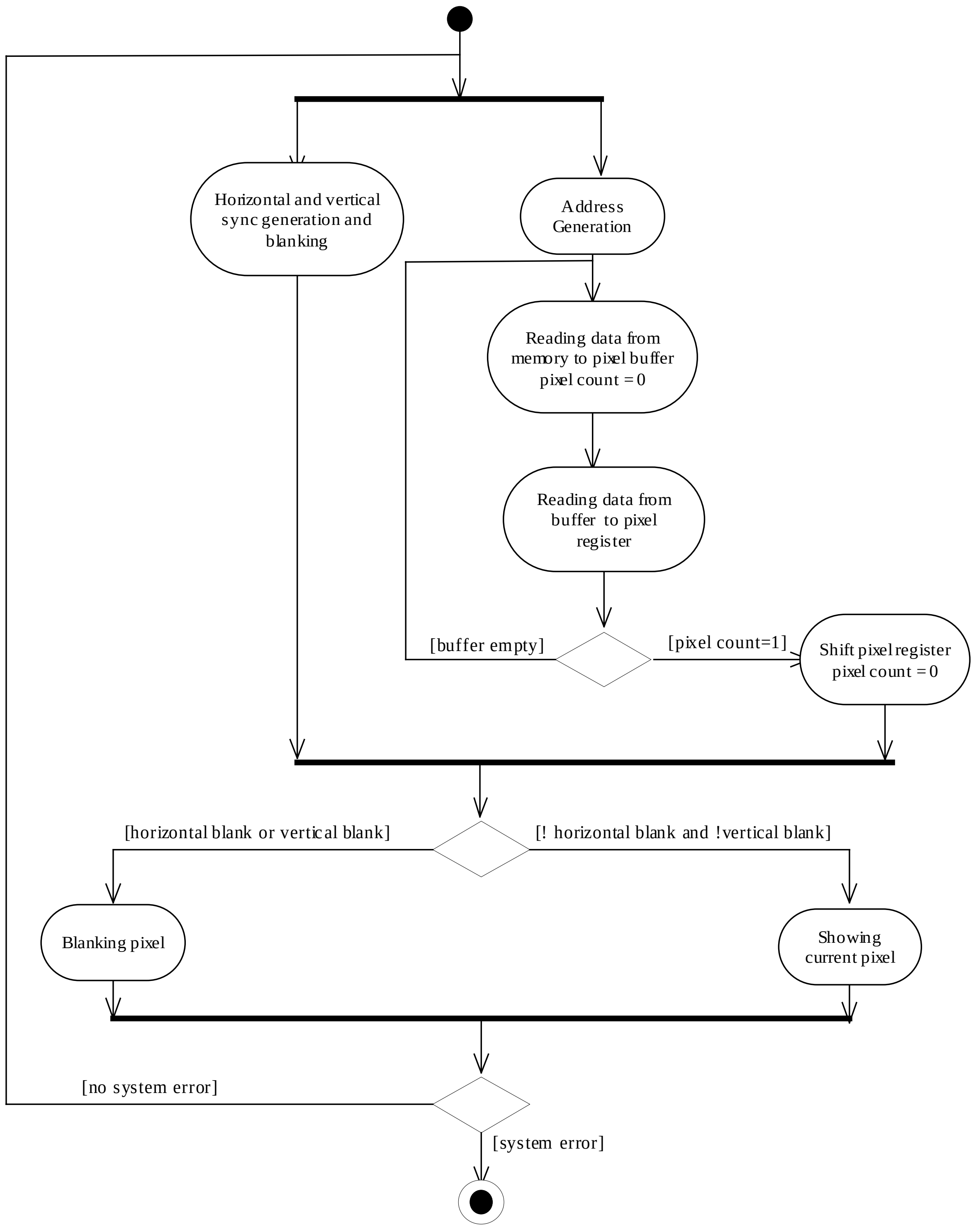
- Activity Diagram for Address Generation:


setting address to the starting address of the current image
setting counter to zero
incrementing address
eof
[counter != count]
[counter == count]
setting address to the starting address of the current image
setting address to the starting address of the next image
incrementing counter
setting counter to zero
[no system error]
[system error]
finish operating

- Activity Diagram for Vertical and Horizontal Blanking Signals:


setting blank, horizontal and vertical counters to zero
incrementing horizontal counter
incrementing vertical counter
[800<counter<1000]
generating horizontal sync
generating horizontal sync
[660000<counter<663200]
[800<counter<1100]
[660000<counter<731200]
setting blank_h to one
setting blank_v to one
horizontal counter reaches 1100
vertical counter reaches 1100
[no system error]
[system error]
finish operating

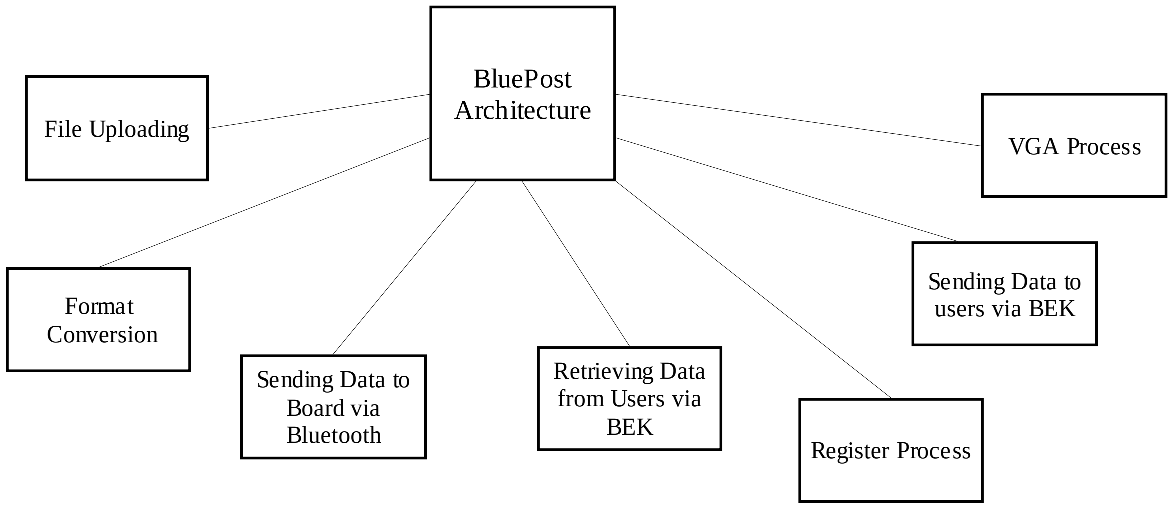
- Activity Diagram for the Complete VGA Process:


Horizontal and vertical sync generation and blanking
Address Generation
Reading data from memory to pixel buffer pixel count = 0
Reading data from buffer to pixel register
[buffer empty]
[pixel count=1]
Shift pixel register pixel count = 0
[horizontal blank or vertical blank]
[! horizontal blank and !vertical blank]
Blanking pixel
Showing current pixel
[no system error]
[system error]

2.2. STRUCTURE CHART AND MODULAR DEPENDENCIES

2.2.1. Structure Chart:

The following chart is the structure chart of our architecture and illustrates the modules and their relations with each other.


BluePost Architecture
File Uploading
VGA Process
Sending Data to users via BEK
Format Conversion
Sending Data to Board via Bluetooth
Retrieving Data from Users via BEK
Register Process

2.2.2. Modular Dependencies and Flow:

In the 'BluePost Architecture' most of the modules work in a sequential order (one after another) except for the "Sending Data to Users via BEK Module", beginning with "File Uploading Module" till "VGA Process Module".

The workflow starts with "File Uploading Module". By using this module, the user uploads the image files and enters the event information. Afterwards, the module forms a directory that includes the uploaded image files, a file named "mesaj.txt" that contains the event information and a file named "bilgi.txt" that contains the information about order and shows duration of each uploaded image and completes its process.

After "File Uploading Module", "Format Conversion Module" takes the role and reaches the directory formed by the previous module. In this module a .hex file ("slideshow.hex") is formed according to the image files and "bilgi.txt" file and saved in the same directory.

The .hex file formed by Format Conversion Module and the "mesaj.txt" file formed by File Uploading Module are sent to Bluetooth Evaluation Kit with the process of "Sending Data to Board via Bluetooth" module. This module also reaches the directory formed by "File Uploading Module" and sends the "slideshow.hex" and "mesaj.txt" files to the kit by doing the necessary bluetooth operations like device discovery,service discovery, and establishing connection.

Once the bluetooth data arrives to the bluetooth evaluation board, this data is sent to the 16F877 pic through serial port. Here, “Sending Data from User via Bluetooth Evaluation Kit Module” starts working. Serial data is received by the use of USART interrupt and the data received is collected as 8 bits and sent to the parallel port of the pic. From here, data arrives to the parallel port of the XSA board. “Register Process Module” sends data from the parallel port of the XSA board to FPGA so that it can be stored in the SDRAM by the “Memory Operations” module.

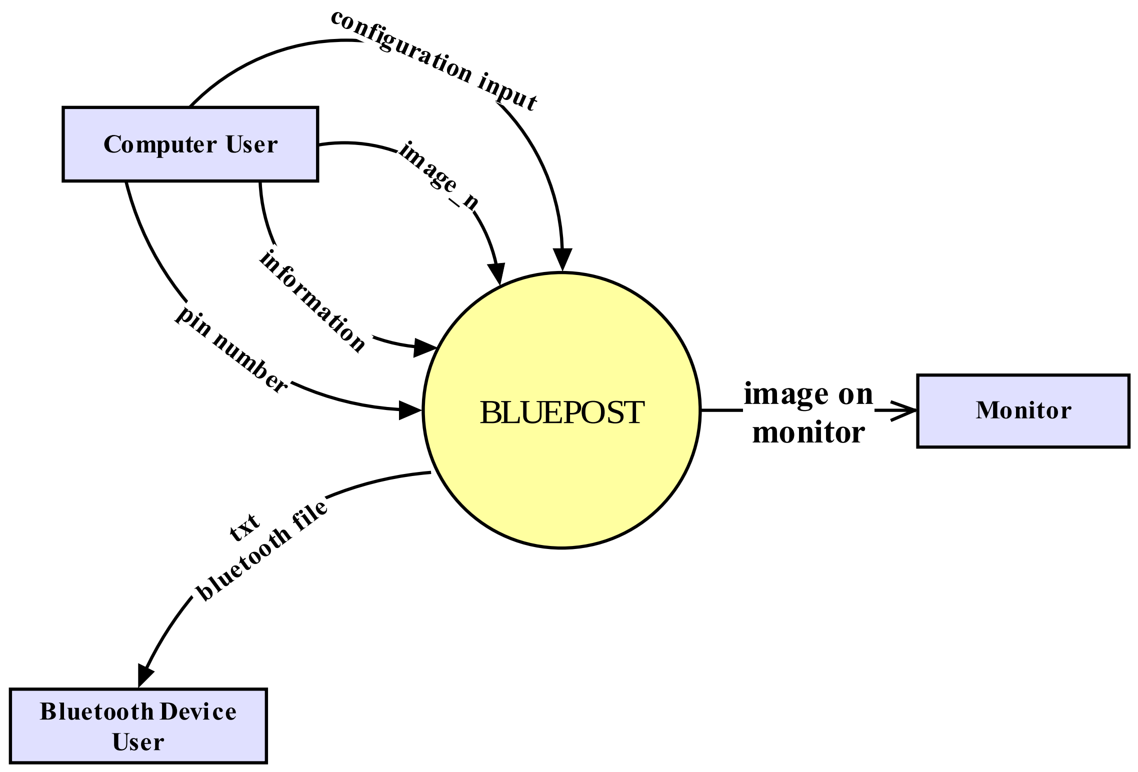
The "VGA Process Module" reads the contents of the hex file from the SDRAM of XSA3S1000 board and according to r, g, b values of each pixel, it displays the images in the .hex data on the VGA monitor according to the show duration data of each image in the .hex file.

Meanwhile, “Register Process Module” reads the contents of the txt file from the SDRAM, and sends this data on three status lines to the parallel port. Here this data is combined with one more bit (A bit whose value is toggled so that RB7-RB4 interrupt will be generated in the 16F877 pic.) and assigned to the D7-D4 data lines of the parallel port. From here, data comes to the parallel port of the 16F877. In this pic, data is received by the use of RB7-RB4 interrupt and sent to the serial port of the pic bit by bit. Then this data is sent to the serial port of the Bluetooth Evaluation Borard. At this board, this data will be sent to bluetooth devices as bluetooth messages.

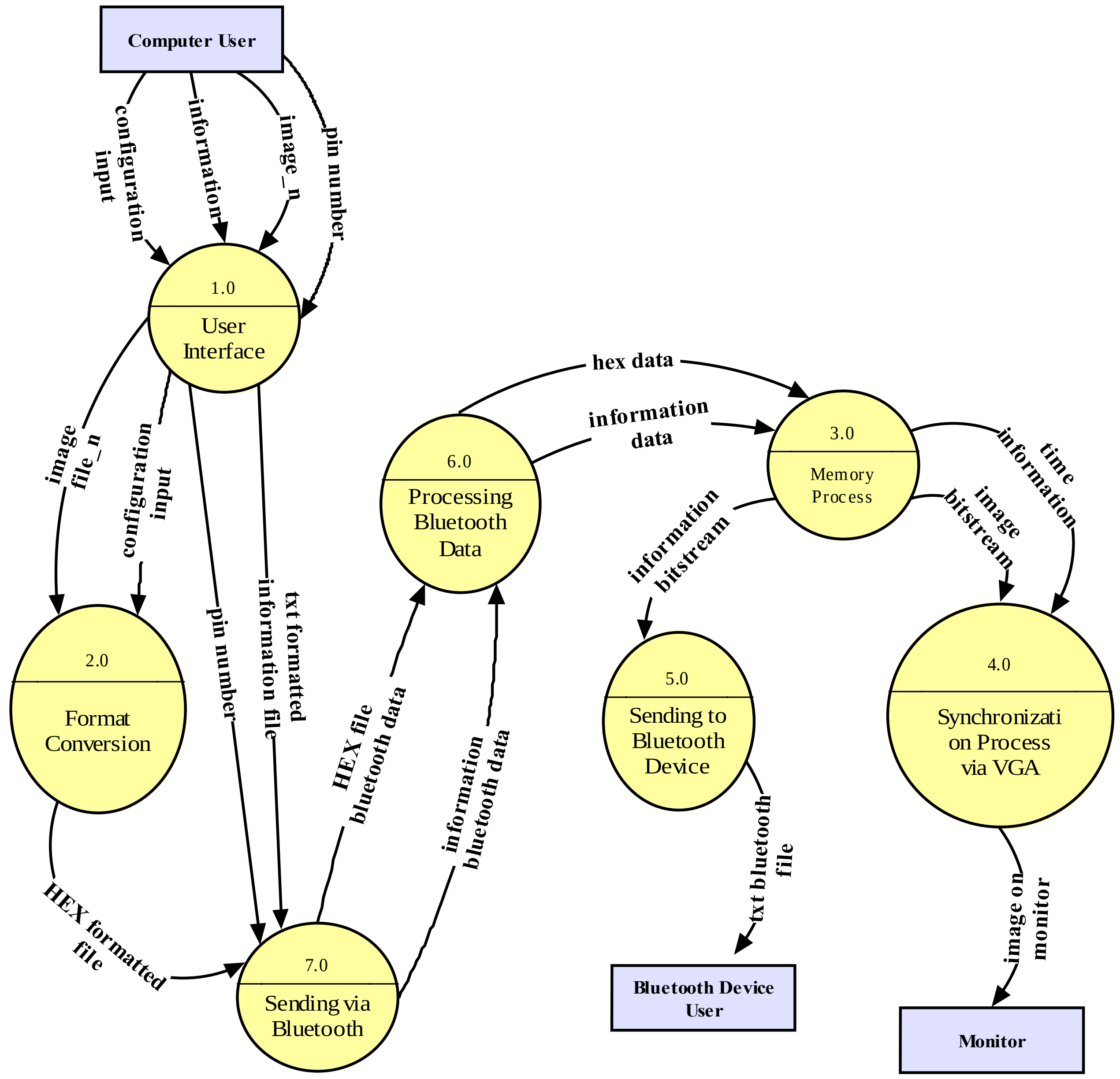
2.3. FUNCTIONAL DESIGN

2.3.1. Data Flow Diagrams (DFDs):

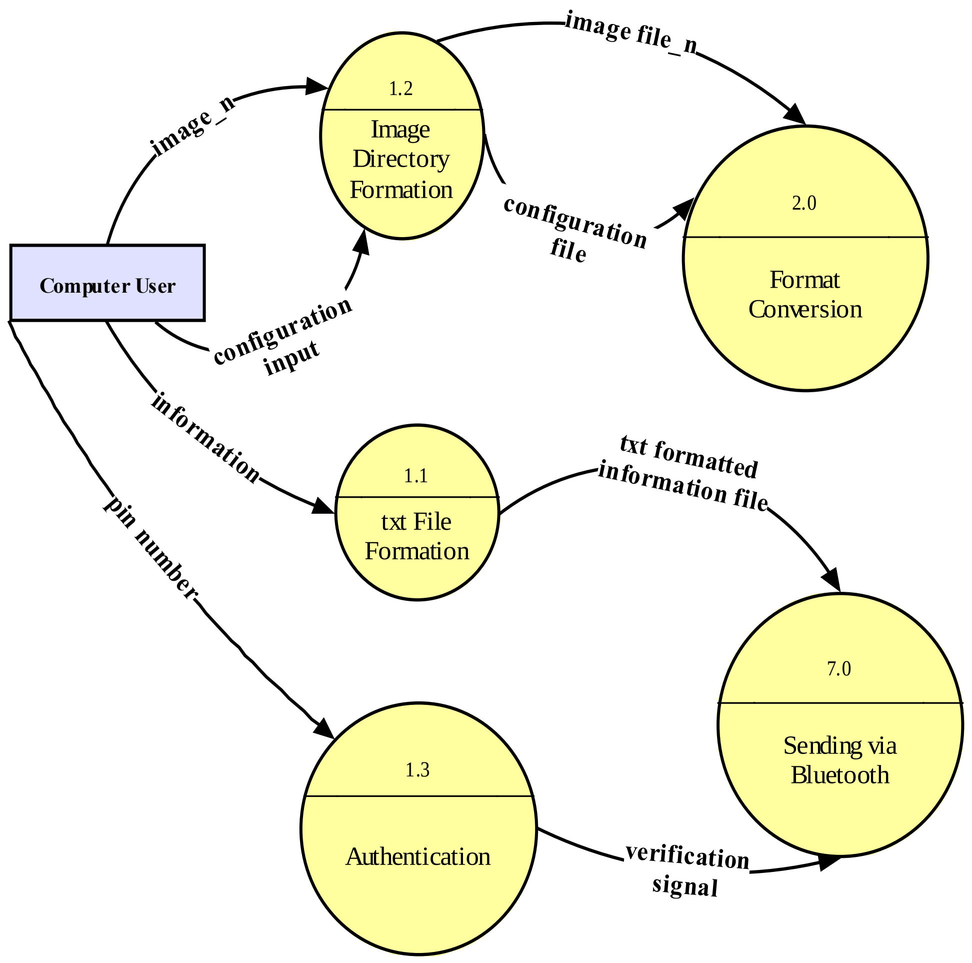
LEVEL 0:



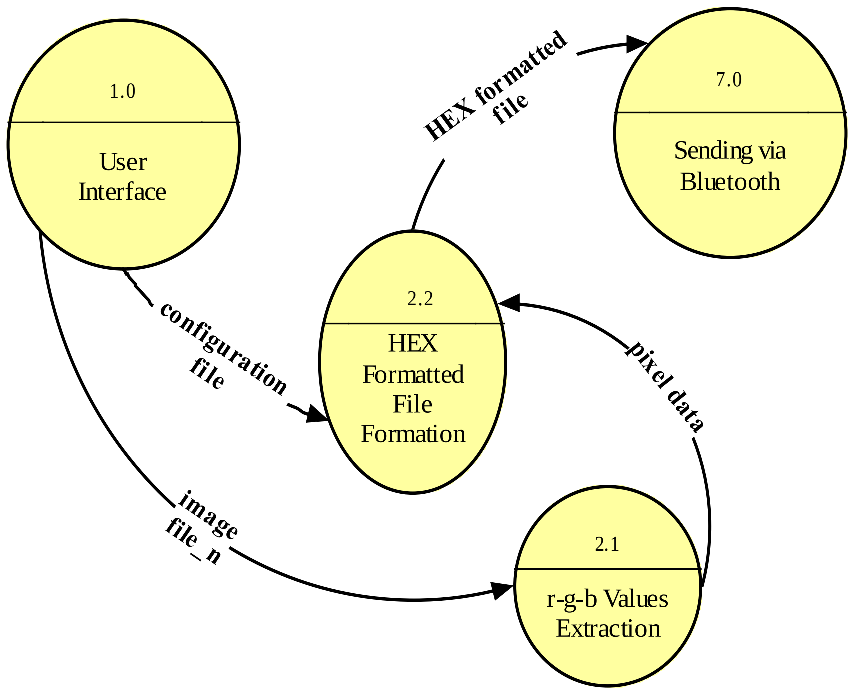
LEVEL 1:


Computer User
configuration input
information
image_n
pin number
1.0
User Interface
image file_n
configuration input
pin number
txt formatted information file
2.0
Format Conversion
HEX formatted file
7.0
Sending via Bluetooth
HEX file bluetooth data
information bluetooth data
6.0
Processing Bluetooth Data
hex data
information data
3.0
Memory Process
information bitstream
image bitstream
time information
5.0
Sending to Bluetooth Device
txt bluetooth file
Bluetooth Device User
4.0
Synchronizati on Process via VGA
image on monitor
Monitor

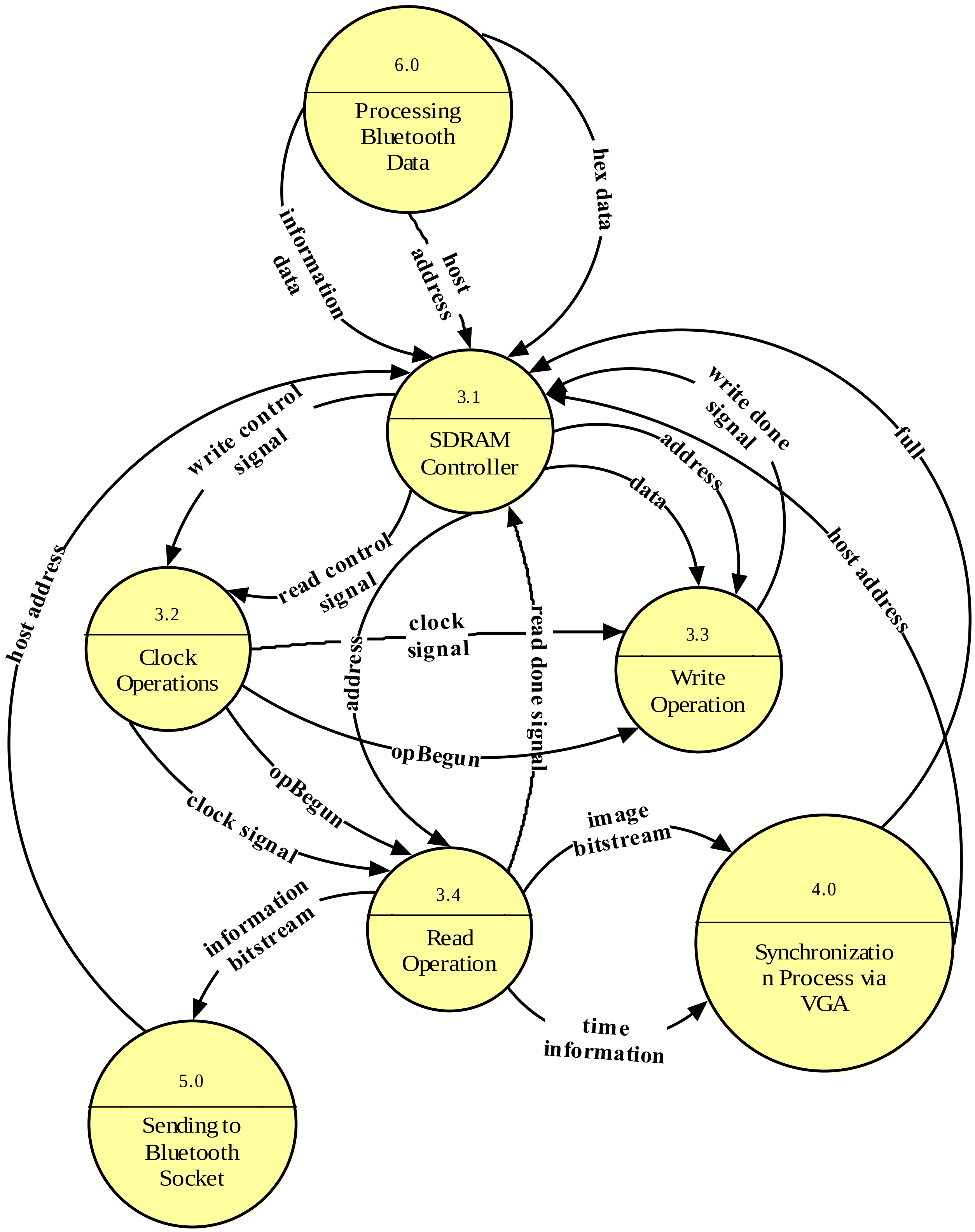
LEVEL 2 FOR USER INTERFACE:



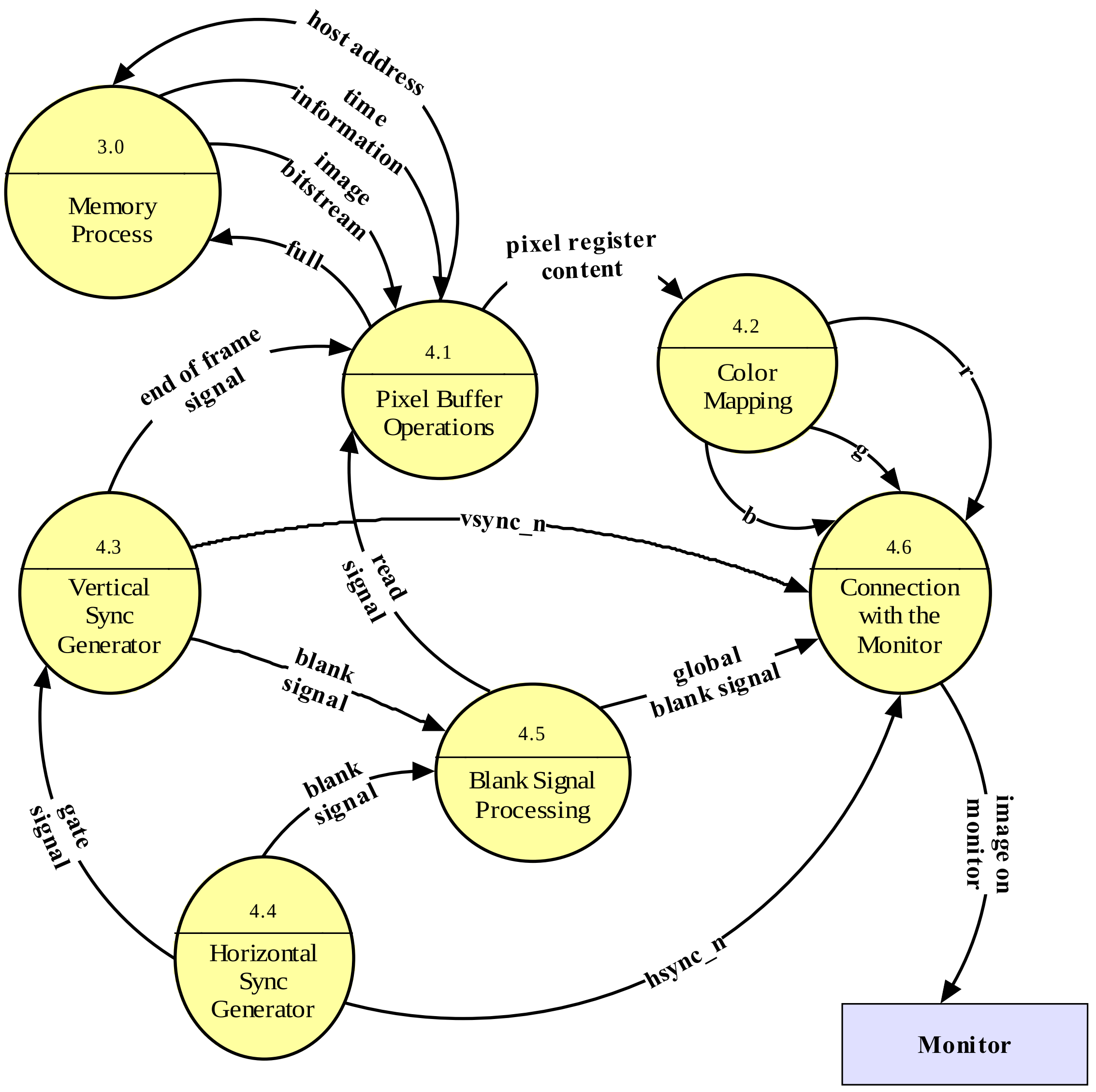
LEVEL 2 FOR FORMAT CONVERSION:



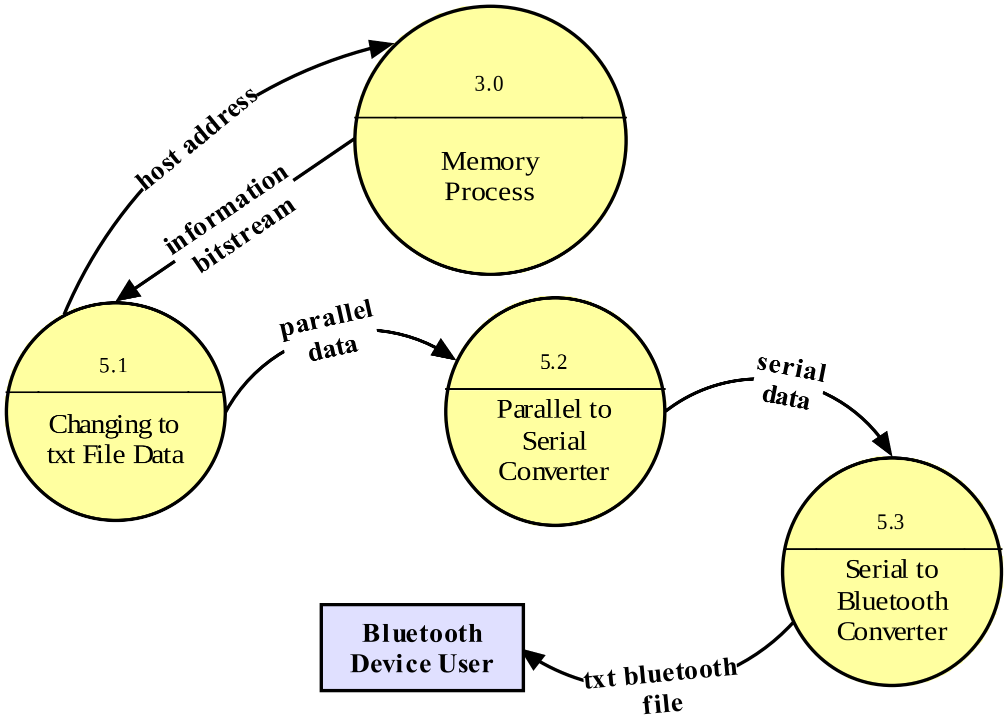
LEVEL 2 FOR MEMORY PROCESS:



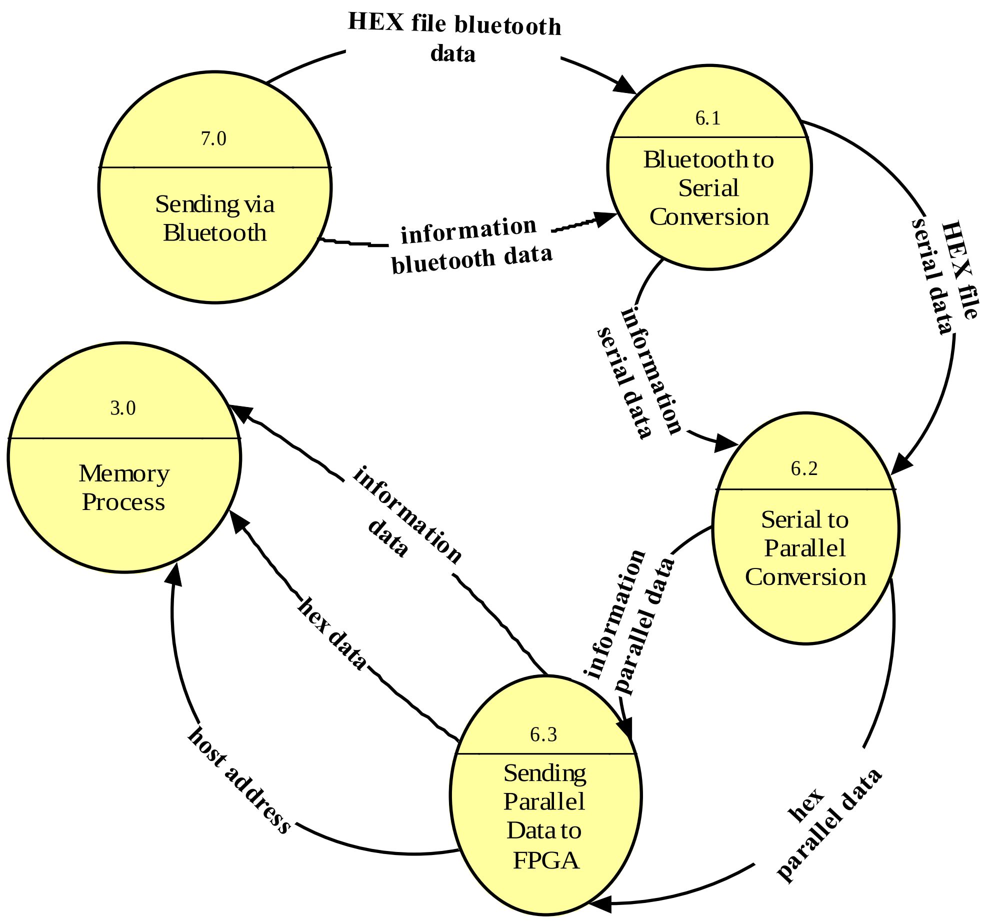
LEVEL 2 FOR SYNCHRONIZATION PROCESS VIA VGA:



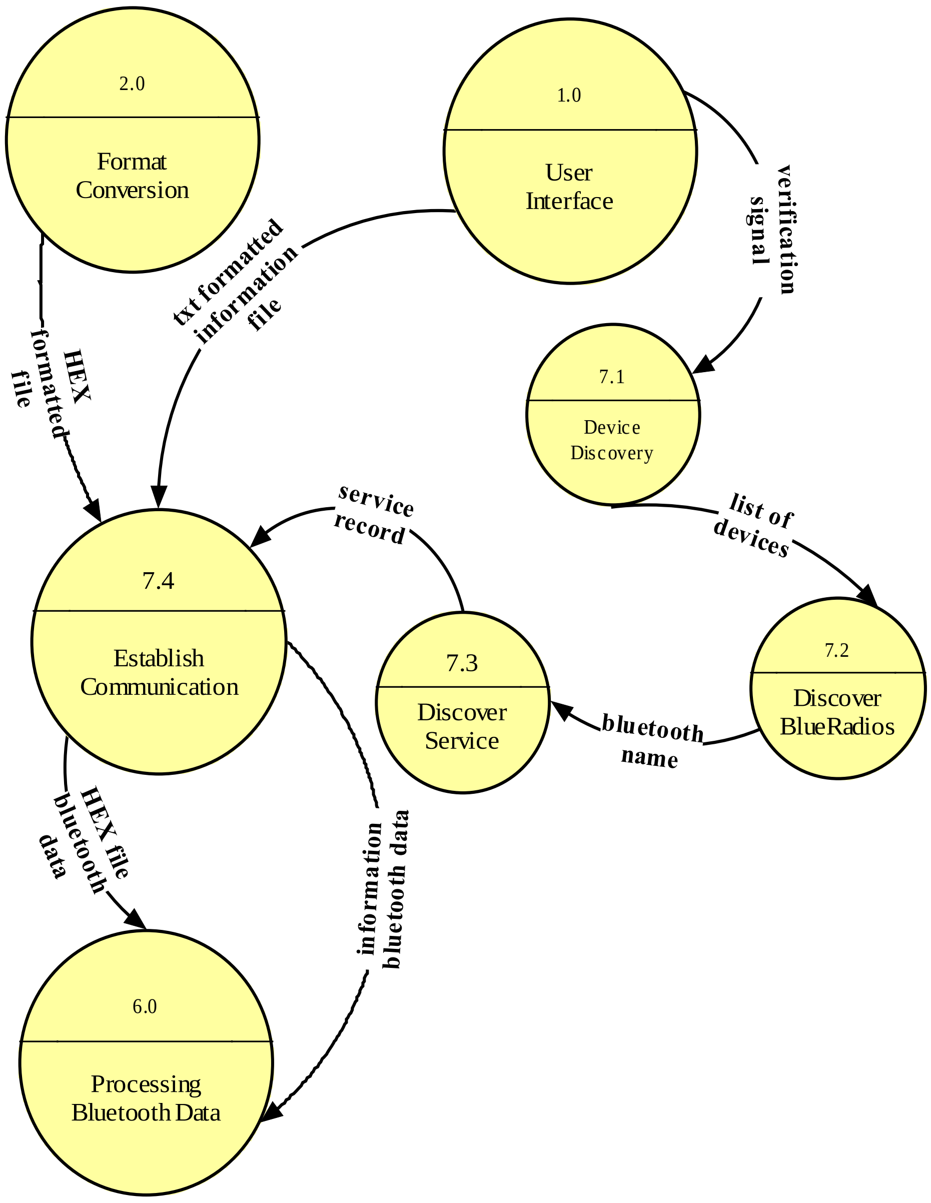
LEVEL 2 FOR SENDING TO BLUETOOTH DEVICE:



LEVEL 2 FOR PROCESSING BLUETOOTH DATA:



LEVEL 2 FOR SENDING VIA BLUETOOTH:



2.3.2. Data Dictionary:

Name	r
Input to	4.6 Connection with the Monitor
Output from	4.2 Color Mapping
Description	The red color signal.
Format	std_logic_vector composed of 3 bits*

Name	g
Input to	4.6 Connection with the Monitor
Output from	4.2 Color Mapping
Description	The green color signal.
Format	std_logic_vector composed of 3 bits

Name	b
Input to	4.6 Connection with the Monitor
Output from	4.2 Color Mapping
Description	The blue color signal.
Format	std_logic_vector composed of 3 bits

Name	address
Input to	3.3 Write Operation 3.4 Read Operation
Output from	3.1 SDRAM Controller
Description	The row and column address of the data to be read or the address to which data will be written given to the SDRAM. This data is obtained from host address and is provided to the SDRAM.
Format	VHDL unsigned type composed of 12 bits

Name	blank signal
Input to	4.5 Blank Signal Processing
Output from	4.3 Vertical Sync Generator 4.4 Horizontal Sync Generator
Description	The blanking signals produced by the vertical and the horizontal sync generators are combined to produce a *global blank signal* and the *read signal*. When the blank signal is high, the pixel should not be displayed.
Format	std_logic : YES when blanking is necessary within a scanline or within a frame, NO otherwise

Name	bluetooth name
Input to	7.3 Discover Service
Output from	7.2 Discover BlueRadios
Description	This is the bluetooth name of the BlueRadios Evaluation board used to discover the services provided by the evaluation board.

Format	This is a string and our device has the name “BlueRadios”.

Name	clock signal
Input to	3.3 Write Operation 3.4 Read Operation
Output from	3.2 Clock operations
Description	The clock signal obtained from the oscillator that is used to clock the SDRAM operations.
Format	std_logic : Main clock input.

Name	configuration input
Input to	1.0 User Interface 1.2 Image Directory Formation
Output from	This is an input to the system.
Description	This is the input entered by the user about the time intervals for which each image will be displayed in a slide show.
Format	The user enters an integer in the range 1-10 for slide number and an integer, specifying seconds, greater than 5 for time duration.

Name	configuration file
Input to	2.0 Format Conversion 2.2 HEX Formatted File Formation
Output from	1.0 User Interface 1.2 Image Directory Formation
Description	This is a file containing the slide number, time duration and the image file name for each image selected by the user.
Format	This file is txt file. Each line in this file looks like: <slide number> <image file name> <time duration>

Name	data
Input to	3.3 Write Operation
Output from	3.1 SDRAM Controller
Description	The data to be stored in the SDRAM. The bluetooth data coming to our main board (hex data and information data) will arrive to the SDRAM Controller and the controller will pass it to the SDRAM through this bus.
Format	This is the data bus composed of 16 bits, that contains data that’s going to be written to a memory word. This data is represented as “unsigned” in VHDL code.

Name	end of frame signal
Input to	4.1 Pixel Buffer Operations
Output from	4.3 Vertical Sync Generator

Description	The signal indicating the end of a frame.
Fomat	std_logic: YES when the end of a frame is reached, NO otherwise.

Name	full
Input to	3.0 Memory Process 3.1 SDRAM Controller
Output from	4.0 Synchronization Process via VGA 4.1 Pixel Buffer Operations
Description	The signal indicating whether the pixel buffer is full or not.
Fomat	std_logic: YES when the pixel buffer is full, NO otherwise.

Name	gate signal
Input to	4.3 Vertical Sync Generator
Output from	4.4 Horizontal Sync Generator
Description	The signal which is used to update the counter of the vertical sync generator correctly.
Format	std_logic : YES at the end of each scanline, NO otherwise.

Name	global blank signal
Input to	4.6 Connection with the Monitor
Output from	4.5 Blank Signal Processing
Description	The signal indicates when the red, green, or blue video signals are blanked. This signal is produced by the *blank signal*s coming from the vertical and horizontal sync generators.
Format	std_logic : YES when a pixel should be blanked,NO otherwise.

Name	hex data
Input to	6.2 Memory Process 3.1 SDRAM Controller
Output from	6.0 Processing Bluetooth Data 6.2 Serial to Parallel Conversion
Description	The data contained in the hex file which will be sent via bluetooth and processed by “serial to parallel conversion” operations will arrive to the parallel port of the board and will be sent to the FPGA so that it can be stored in the SDRAM. This signal represents the data bus that contains data that will be sent to the FPGA from the parallel port of the XSA board.
Format	This parallel data is composed of 8 bits and it is declared as “unsigned” in VHDL code. When the 8 bit data arrives to the FPGA, it will be combined with the next coming 8-bit data and will be passed to the 16-bit data bus of the SDRAM.

Name	HEX file bluetooth data
Input to	6.0 Processing Bluetooth Data 6.1 Bluetooth to Serial Converter
Output from	7.0 Sending via Bluetooth 7.4 Establish Communication
Description	The HEX file formed in “Format Conversion” will be send to our board via bluetooth. “HEX file bluetooth data” represents this incoming data. This data contains both the configuration input and image pixel data.
Format	This data is arrives in Hex format.

Name	HEX file serial data
Input to	6.2 Serial to Parallel Conversion
Output from	6.1 Bluetooth to Serial Conversion
Description	When “HEX file bluetooth data” comes to the bluetooth evaluation board, it is sent to “Serial to Parallel Converter” as serial data.
Format	This data arrives bit by bit and bit and when combined, these bits represent the HEX formatted data.

Name	HEX formatted file
Input to	7.0 Sending via Bluetooth 7.4 Establish Communication
Output from	3.0 Format Conversion 2.2 HEX formatted file formation
Description	This file contains information about the configuration input entered by the user and the pixel data for the images uploaded. The format of this file is described in detail in “Process Specifications.”
Format	This file is written in Hex format.

Name	hex parallel data
Input to	6.3 Sending Parallel Data to FPGA
Output from	6.2 Serial to Bluetooth Conversio
Description	When “hex bluetooth data” comes to the bluetooth evaluation board, it is sent to “Serial to Parallel Conversion” operations and then arrives to the XSA parallel port. This signal represents the data bus that carries data to the parallel port of the XSA board from the pic (designed by us) that converts the bluetooth serial data to parallel data.
Format	This data arrives to the parallel port as a data bus of 8 bits and is represented as “unsigned” in VHDL.

Name	host address
Input to	3.1 SDRAM Controller
Output from	4.0 Synchronization Process via VGA 4.1 Pixel Buffer Operations 5.0 Sending to Bluetooth Device 5.1 Sending to Parallel Port 6.0 Processing Bluetooth Data 6.2 Serial to Parallel Conversion
Description	The address of the data to be read from or to be written to produced by the FPGA applications.
Format	This is a bus defined of type unsigned composed of 24 bits.

Name	hsync_n
Input to	4.6 Connection with the Monitor
Output from	4.4 Horizontal Sync Generator
Description	This signal derives the horizontal sync input of the monitor.
Format	std_logic : This signal becomes positive when the visible area is in progress within a scanline and negative to indicate the start and end of a scanline.

Name	image_n
Input to	1.2 User Interface 1.2 Image Directory Formation
Output from	It is an input to the system.
Description	The images stored in the PC, available for selection to be displayed on the monitor.
Format	The images can be in jpg or gif formats.

Name	image bitstream
Input to	4.0 Synchronization Process via VGA 4.1 Pixel Buffer Operations
Output from	3.0 Memory Process 3.4 Read Operation
Description	The image bitstream read from the SDRAM to the VGA port.
Format	VHDL unsigned type. The bitstream is read as words of 16 bits. Since we will have 16-bit pixels, the bitstream will be read as pixels.

Name	image file_n
Input to	7.0 Format Conversion 2.2 r-g-b values extraction
Output from	1.0 User Interface 1.2 Image Directory Formation
Description	The image files uploaded by the user which will go through the format conversion process.
Format	The format of the image can be jpg or gif.

Name	image on monitor
Input to	It is an output of the system.
Output from	8.0 Synchronization Process via VGA 4.6 Connection with the monitor
Description	The image displayed on the monitor

Name	information
Input to	1.1 User Interface 1.1 Txt file formation
Output from	It is an input to the system.
Description	The information entered by the user related to the event date and time.
Format	User enters a message and this message will be stored in a txt formatted file named “mesaj.txt”.

Name	information bitstream
Input to	5.0 Sending to Bluetooth Device 5.1 Sending to Parallel Port
Output from	3.0 Memory Process 3.4 Read Operation
Description	The bitstream read from the SDRAM, which contains the message about the event data, ready to be processed by the bluetooth functionalities of the system so that a bluetooth message can be sent to devices.
Format	This data comes on a data bus composed of 16 bits and contains the message file which was written in txt format.

Name	information bluetooth data
Input to	7.3 Processing Bluetooth Data 6.1 Bluetooth to Serial Converter
Output from	7.0 Sending via Bluetooth 7.4 Establish Communication

Description	The bluetooth data recieved from the bluetooth device containing the information about the event, namely the data in the “txt formatted information file”.
Format	This data arrives as txt file.

Name	information data
Input to	3.0 Memory Process 3.1 SDRAM Controller
Output from	6.0 Processing Bluetooth Data 6.3 Sending Parallel Data to FPGA
Description	The time and place information of the event sent from the parallel port of the board to the FPGA so that it can be stored in the SDRAM.
Format	This parallel data is composed of 8 bits and it is declared as “unsigned” in VHDL code. When the 8 bit data arrives to the FPGA, it will be combined with the next coming 8-bit data and will be passed to the 16-bit data bus of the SDRAM.

Name	information parallel data
Input to	6.3 Sending Parallel Data to FPGA
Output from	6.2 Serial to Bluetooth Conversion
Description	When “information bluetooth data” comes to the bluetooth evaluation board, it is sent to “Serial to Parallel Conversion” operations and then arrives to the XSA parallel port. This signal represents the data bus that carries data to the parallel port of the XSA board from the pic (designed by us) that converts the bluetooth serial data to parallel data.
Format	This data arrives to the parallel port as a data bus of 8 bits and is represented as “unsigned” in VHDL.

Name	information serial data
Input to	6.2 Serial to Parallel Conversion
Output from	6.1 Bluetooth to Serial Conversion
Description	When “information bluetooth data” comes to the bluetooth evaluation board, it is sent to “Serial to Parallel Converter” as serial data.
Format	This data arrives as a txt formatted file.

Name	list of devices
Input to	7.2 Discover BlueRadios
Output from	7.1 Device Discovery
Description	In order to send bluetooth data from the user pc, our code will request for a list of available bluetooth devices, and this list contains the available devices to connect.
Format	The availabe devices will be represented by the Java RemoteDevice class and this list is an array of RemoteDevice objects.

Name	opBegun
Input to	3.3 Write Operation 3.4 Read Operation
Output from	3.2 Clock Operations
Description	This signal becomes high with the rising edge of the SDRAM clock input when a read or a write is requested. It initiates the requested operation.
Format	std_logic: YES when a read or a write operation is requested and clock input is high, NO otherwise

Name	parallel data
Input to	5.2 Parallel to Serial Conversion
Output from	5.1 Sending to Parallel Port
Description	While sending bluetooth messages, the message content which is stored in the SDRAM, needs to first come to the parallel port of the XSA board. From the parallel port, this data will be sent to the parallel port of the pic(designed by us). This data bus represents the data that is sent from the parallel port of the XSA board to the parallel port of the other pic.
Format	The parallel port of the XSA board, keeps data in parallel port registers as a byte. So this data is represented as a data bus composed of 8 bits. It is represented by unsigned type in VHDL.

Name	pin number
Input to	1.0 User Interface 1.3 Authentication
Output from	This is an input to the system.
Description	The pin number, which is specific to the bluetooth converter card, the user must enter in order to send images to the board.
Format	The user enters an integer.

Name	pixel register content
Input to	4.2 Color Mapping
Output from	4.1 Pixel Buffer Operations
Description	The data in the pixel buffer is shifted to the pixel register and the contents of this register are processed to produce color signals.
Format	std_logic_vector composed of 16 bits

Name	read control signal
Input to	3.2 Clock Operations
Output from	3.1 SDRAM Controller
Description	The signal indicating a read request from the memory.
Format	std_logic : YES when a read operation is pending, NO otherwise.

Name	read done signal
Input to	3.1 SDRAM controller
Output from	3.4 Read Operation
Description	The signal shows that the current read operation is completed.
Format	std_logic: YES when the current read operation is finished.

Name	read signal
Input to	4.1 Pixel Buffer Operations
Output from	4.5 Blank Signal Processing
Description	The signal which indicates when to read more data from the pixel buffer.
Format	std_logic: YES when the buffer is empty, NO otherwise.

Name	serial data
Input to	5.3 Serial to Bluetooth Conversion
Output from	5.2 Parallel to Serial Conversion
Description	The parallel data obtained from the parallel port of the XSA board, will be sent to the pic (designed by us) responsible for sending this data to the serial port on it. The pic will get the parallel data and send to it to its own serial data. This signal represents the data that arrives to the serial port of this pic.
Format	Since this signal represents the serial data, it will be processed by bit by.

Name	service record
Input to	7.4 Establish Communication
Output from	7.3 Discover Service
Description	We will discover the services offered by the bluetooth evaluation board get a record for the service, namely “Serial Port” service. This record will be used in connection.
Format	This record is represented by the Java ServiceRecord class.

Name	time information
Input to	4.0 Synchronization Process via VGA 4.1 Pixel Buffer Operations
Output from	3.0Memory Process 3.4 Read Operation
Description	The time information stored in the SDRAM which is obtained from the configuration file and which is used to determine which image should be displayed at a specific moment.
Format	VHDL unsigned type. The information is read as data words of 16 bits.

Name	txt bluetooth file
Input to	This is an output of the system.
Output from	5.0 Sending to Bluetooth Device 5.3 Serial to Bluetooth Conversion
Description	From the serial port of the pic(designed by us) data is sent to the bluetooth evaluation board. The evaluation board receives this data and sends it as a txt message to bluetooth devices. This signal represents the txt file ready to be sent to the bluetooth devices containing information about the event.
Format	The file will be in txt format.

Name	txt formatted information file
Input to	7.0 Sending via Bluetooth 7.4 Establish Communication
Output from	1.0 User Interface 1.1 Txt File Formation
Description	The file created by the information entered by the user about the details of the event.
Format	txt formatted file

Name	verification signal
Input to	7.0 Sending via Bluetooh 7.1 Device Discovery
Output from	1.0 User Interface 1.3 Authentication
Description	A verification signal indicating that the user has entered the correct pin number.
Format	Java Boolean type

Name	vsync_c
Input to	4.6 Connection with the Monitor
Output from	4.3 Vertical Sync Generator
Description	This signal derives the vertical sync input of the monitor.
Format	std_logic: This signal becomes positive when the visible area within a frame is in progress and negative when the end of a frame is reached.

Name	write control signal
Input to	3.2 Clock Operations
Output from	3.1 SDRAM Controller
Description	The signal indicating the write request to the memory.
Format	std_logic: YES when a write operation is pending, NO otherwise.

Name	write done signal
Input to	3.1 SDRAM Controller
Output from	3.3 Write Operation
Description	The signal indicating that the current write operation is completed.
Format	std_logic: YES when the current write operation is completed.

* : When wires are described as signals in VHDL, they are defined with the type *std_logic*. With this type the signal can be set to low (0), high (1), or high impedence. *std_logic_vector* is an array of the *std_logic* type. It represents a bus which has a dimension associated with it. Type *unsigned* is again an array of *std_logic* used to declare variables. This type also has a dimension.

2.3.3. Process Specifications:

1.1 txt File Formation:

This process gets the message related to the poster event entered by the user via the user interface. Afterwards, this message is written in a txt file named "mesaj.txt". This file is kept in the directory formed by "1.2 Image Directory Formation".

1.2 Image Directory Formation:

When the user selects jpg or gif images for creating a slide show, this process creates a directory and the selected images are kept in this directory. The directory is created with the name "new folder" and when the user saves the slide show the directory is renamed to the name specified by the user. As the user selects an image and specifies a slide number and time duration, this process writes this data (format: slide number – image file name – time duration) in a configuration file named "bilgi.txt". This file is in txt format and it is also kept in the directory that is used to store the images.

1.3 Authentication:

The bluetooth evaluation kit attached to the main XSA board will have a predetermined pin number and in order to send data to this kit one has to enter this pin number. Thus when the user decides to send the contents of the slide show, s/he is requested to enter a pin number by this process. If the pin number matches with the pin number of the bluetooth evaluation kit, a verification signal is raised so that bluetooth operations can start.

2.1 r-g-b Values Extraction:

When the user decides the send the slide show contents to the board, this process is initiated first. The selected images that are stored in this directory formed by “1.2 Image Directory Formation” are first resized to 800 * 600 dimensions first so that each image occupies same space in the SDRAM and there’s a consistency in the display process. Then each resized image is processed and the r, g, and b color information of each pixel in an image is obtained. These values are send to “2.2 HEX File Formation”.

2.2 HEX File Formation:

This process starts by reading the configuration file produced by “1.2 Image Directory Formation”. The data in the configuration file is written in a new file named “slideshow.hex” in hex format. The data in the configuration file will be stored in the first 16 memory words of the SDRAM. This process will write the hex data in the following format:

address – configuration data that is going to be stored in this SDRAM address

To be more specific, the format will be like:

address – number of images

address – time duration of the 1st slide

address – time duration of the 2nd slide

…

address – time duration of the 10th slide

The user may have selected less than 10 images, in that case the time duration of the slides not specified will be given the value zero. Here the address specifies one of the first 16 memory words.

After the first phase is complete, the second phase in this process starts. In our design, each image is stored in the predetermined address ranges of the SDRAM. Thus this process writes the obtained pixel color information from “2.1 r-g-b Values Extraction” again in “slideshow.hex” with appending the SDRAM address to which this data will be written. In other words the format of the file is :

address - pixel data that is going to be stored in this SDRAM address

After the pixel data of an image is completely written, we will write “FF” to the next memory location which will indicate the end of the current image. Thus the hex filw will contain the following line, after the pixel data of each image is written:

address – FF

We will divide the SDRAM into two parts and the first part will contain the data related to the VGA operations. Thus, here the address specifies an address location from the first part of the SDRAM, excluding the first 16 memory words.

3.1. SDRAM Controller:

This process communicates with our applications running on the FPGA. The SDRAM operation type (read from or write to the memory) is determined by the signals coming from these applications. According to the type of the operation, this process raises either a write control or a read control signal. The process is used for gathering data (if the operation is a read operation) and address information from “4.0 Synchronization Process with VGA”, “5.0 Sending to Bluetooth Device”, and “6.0 Processing Bluetooth Data” and for dividing this host address information into memory address rows and columns. If the operation is a write operation, the data and address information is sent to "3.3 Write Operation Process" to be written in the memory according to the address row and column information. If the operation is a read operation the address row and column information is sent to "3.4 Read Operation" to read the related data from memory.

3.2. Clock Operations:

This process uses the oscillator signal and creates a new signal for SDRAM by concerning the delays between the SDRAM and FPGA. This created clock signals are used by SDRAM for read and write operations. When the process retrieves the write control signal or read control signal from "3.1 SDRAM Controller", it creates the new clock signal and opBegun signal and sends this signal to either "3.3 Write Operation" or "3.4. Read Operation" according to write/read control signals . The SDRAM write and read operations start immediately after the first rising edge of this created clock.

3.3. Write Operation:

This process is used for writing the data information gathered from "3.1 SDRAM Controller" using the address (row and column) information gathered from the same process. Before this process starts to operate the address and data information gets ready in the address and data buses of the SDRAM. After the clock signal and opBegun signal is gathered from "3.2 Clock Operations" this process starts to operate with the rising edge of the clock signal and writes the data in the data bus to the address specified by the row and column information in the address bus.

3.4. Read Operation:

This process is used for reading the data information from the SDRAM using the address (row and column) information gathered from "3.1 SDRAM Controller". Before this process starts to operate the address information gets ready in the address buses of the SDRAM. After the clock signal and opBegun signal is gathered from "3.2 Clock Operations" this process starts to operate with the rising edge of the clock signal and reads the data from the SDRAM according to the address specified by the row and column information in the address bus. We will use the "dual-port" feature of the SDRAM Controller. The read data from the first part of the SDRAM will always contain the following information: The first 16 memory words of the SDRAM will be read first. These words will contain the time duration information of each frame uploaded by the user. After these 16 words, the frame data will be read for each frame according to the address information. All this read data will be sent to "4.0 Synchronization Process via VGA" as image bitstream and time information. While VGA Controller gets time and image information from the first part of the SDRAM, this process will read and send the event information to "5.0. Sending to Bluetooth Socket" to be sent to bluetooth device users.

4.1 Pixel Buffer Operations:

This process reads data from the SDRAM and processes this data so that images are displayed on a monitor. The process first reads time information kept in the first 16 memory words and stores this data in variables later to use. Then this process starts reading pixel data from the first portion of the SDRAM. The read data is stored in a pixel buffer. As long as the buffer is not full, this process updates the address and initiates a read operation. When the buffer becomes full, the full signal becomes high and no read operations occur until the full signal becomes low again. The address is incremented in each clock cycle unless end of a frame is reached. If eof signal becomes high, it is checked whether the time duration of the current image has passed. If so the address is updated to the address of the next image and incrementing process starts again. If not, the address is updated to the beginning address of the current image. Meanwhile, a pixel data is put into the pixel register so that the pixel can be displayed. Because the pixel register is 16 bits wide and we store each pixel as 2 bytes, the register can hold one pixel at a time. Thus a pixel data is put into the register from the buffer and when that pixel is displayed another pixel is put and so on. In this process the read signal coming from the “4.5 Blank Signal Processing” is taken into account. This read signal makes sure that a pixel is read from the buffer when the global blanking signal is low. (The pixel will not be blanked.)

4.2 Color Mapping:

This process gets the pixel in the pixel register and extracts the r, g, b values of the pixel and sends these values to “4.6 Connection with the Monitor”. Because each pixel is 16 bits wide, bits 0-2 contain blue color information, bits 3-5 contain green color information, and bits 6-8 contain red color information. And since these bits are packed in the lower nine bits, they directly map to the RGB values.

4.3 Vertical Sync Generator:

This process produces the vsync_n signal that derives the vertical sync input of the monitor. This process gets “gate signal” from “4.4 Horizontal Sync Generator” and updates its counter because this gate signal shows that a scanline has finished. Because we resize the images to constant dimensions (800 * 600), when the counter becomes 800, in other words all scanlines in a frame have been displayed, this process raises the end of frame signal (eof) end sends it to “4.2 Color Mapping.” Additionally, this process raises the vsync_n signal sends it to “4.6 Connection with the Monitor”. The negative pulses on this signal indicate the start and end of frame so that the monitor displays the scanlines between the top and bottom visible area. During this period, a blanking signal is sent to “4.5 Blank Signal Processing”. In our design, vertical sync generator period is 14.624 ms. Of this period, 13.2 ms is the visible area and the rest 1.424 ms (front porch + back porch + sync pulse) indicates the start and end of a frame. (Refer to the VGA process module for details.)

4.4 Horizontal Sync Generator:

This process produces the hsync_n signal that derives the horizontal sync input of the monitor. This process produces a gate signal at the end of a scanline and sends it to “4.3 Vertical Sync Generator”. This process also raises the hsync_n signal and sends it to “4.6 Connection with the Monitor”. The negative pulses on this signal indicate the start and end of a scanline so that the pixels between the left and right edges of the visible screen area are displayed. During this period a blanking signal is activated and sent to “4.5 Blank Signal Processing”. In our design, horizontal sync signal period is 22 µs. 16 µs of this period is visible and the rest 6 µs (front porch + back porch + sync pulse) indicates the start and end of a frame. (Refer to the VGA process module for details.)

4.5 Blank Signal Processing:

This process gets blanking signals from "4.3 Vertical Sync Generator" and "4.4 Horizontal Sync Generator" and these signals are logically or ed to produce a global blanking signal, which is sent to "4.6 Connection with the Monitor". These blanking signals are also used to determine when to read more pixels from the buffer. A read signal is raised when the global blanking signal is low and the current pixel in the pixel register has been processed and this signal is sent to "4.1 Pixel Buffer Operations".

4.6 Connection with the Monitor:

This process gets the RGB components from the "4.2 Color Mapping" and global blanking signal from "4.5 Blank Signal Processing". The RGB components are displayed when the global blanking signal is not high. This process also gets vsycn_n from "4.3 Vertical Sync Generator" and hsync_n from "4.4 Horizontal Sync Generator". These signals derive the vertical and horizontal sync inputs of the monitor respectively.

5.1 Sending to Parallel Port:

For sending messages to the bluetooth devices, data about the message content should be first read from the SDRAM and sent to the parallel port of the XSA board. This process generates host address from which data will be read from the SDRAM. Because we know the starting address of the message data in the SDRAM, we will start generating addresses beginning with this address. Then the address will be incremented with each read operation. The read data from the SDRAM will arrive to the FPGA. FPGA will send this data as 3 bits by deriving the A2, A1, and A0 address lines of the Flash. Then the S3, S2, and S1 status lines of the parallel port will be derived and data will come to the parallel port. Finally this process will assign these lines to the D7, D6, and D5 data lines of the parallel port. A value of 0 and 1 will be put on the D4 data line.

5.2 Parallel to Serial Conversion:

This process retrieves the parallel data from "5.1. Sending to Parallel Port" and send this data to Serial-Parallel Converter Board so that the parallel data is converted to serial data. Details of the conversion process are explained in "Sending Data to User via Bluetooth Evaluation Kit

Module” specifications. This converted serial data then sent to “5.3. Serial to Bluetooth Conversion” process.

5.3 Serial to Bluetooth Conversion:

Once serial data arrives to the Bluetooth Evaluation board, this process sends the coming data to the nearby devices as bluetooth messages (as a .txt file).

6.1 Bluetooth to Serial Conversion:

When the user wants to send slide show contents to the XSA board, the hex formatted file and the txt file are sent via bluetooth operations. This data arrives to the bluetooth board and sent to the 16F877 pic through serial port by this process.

6.2 Serial to Bluetooth Conversion:

When data arrives to the serial port of the 16F877 pic, this data is received by the use of USART interrupt and when 8 bits of data is collected bit by bit, this 8-bit data is sent to the parallel port of the pic. From here, this data arrives to the parallel port of the XSA board.

6.3 Sending Parallel Data to FPGA:

When the data sent by the user via bluetooth arrives to the parallel port of the XSA board, after the serial to conversion operations, this process sends the coming data to the FPGA so that it can be stored in the SDRAM. For this purpose, the data will be put on the D0-D7 pins of the parallel port. Then this data will pass through the CPLD and arrive to the FPGA. While sending the hex file format, first the address and then data that needs to be written to that address will arrive. When an 8-bit data arrives to the FPGA, this data will be combined with the next two coming 8-bit data to produce the address data composed of 24 bits. This address will be put to the host address data bus of the SDRAM controller. Then, two coming 8-bit data will be combined to produce data composed of 16 bits (hex data). Then this data bus will be transferred to the data bus of the SDRAM controller. Then the write operation can start in the Memory Process module. After all the hex data is transferred, the transfer of the txt data will start. (Because we know the number of images and the number of bytes in an image, we will be able to recognize when all hex data has been received.) This data will be stored starting with the end address of the hex data in the SDRAM. Thus, with every two 8-bit data coming, the current address will be incremented and the combined 16-bit data will be transferred to the data bus of the SDRAM so that a write operation can start.

7.1 Device Discovery:

When the user decides to send a slide show content to the XSA board and enters the correct pin number for the bluetooth evaluation board, a verification signal arrives and this process starts. In order to establish a connection with the bluetooth evaluation board, this process searches for the available bluetooth devices returns a list of the available devices.

7.2 Discover BlueRadios:

This process gets the list of available bluetooth devices from “7.1 Device Discovery” and finds the bluetooth evaluation board attached to our main board in this list. It returns the name of the bluetooth evaluation board, which will be “BlueRadios”.

7.3 Discover Service:

Once the bluetooth evaluation board is discovered and its name is retrieved, this process searches for the available services provided by this evaluation board. We will use the “Serial Port” service so when this process discovers this service, it returns a service record, which is passed to “7.4 Establish Communication”.

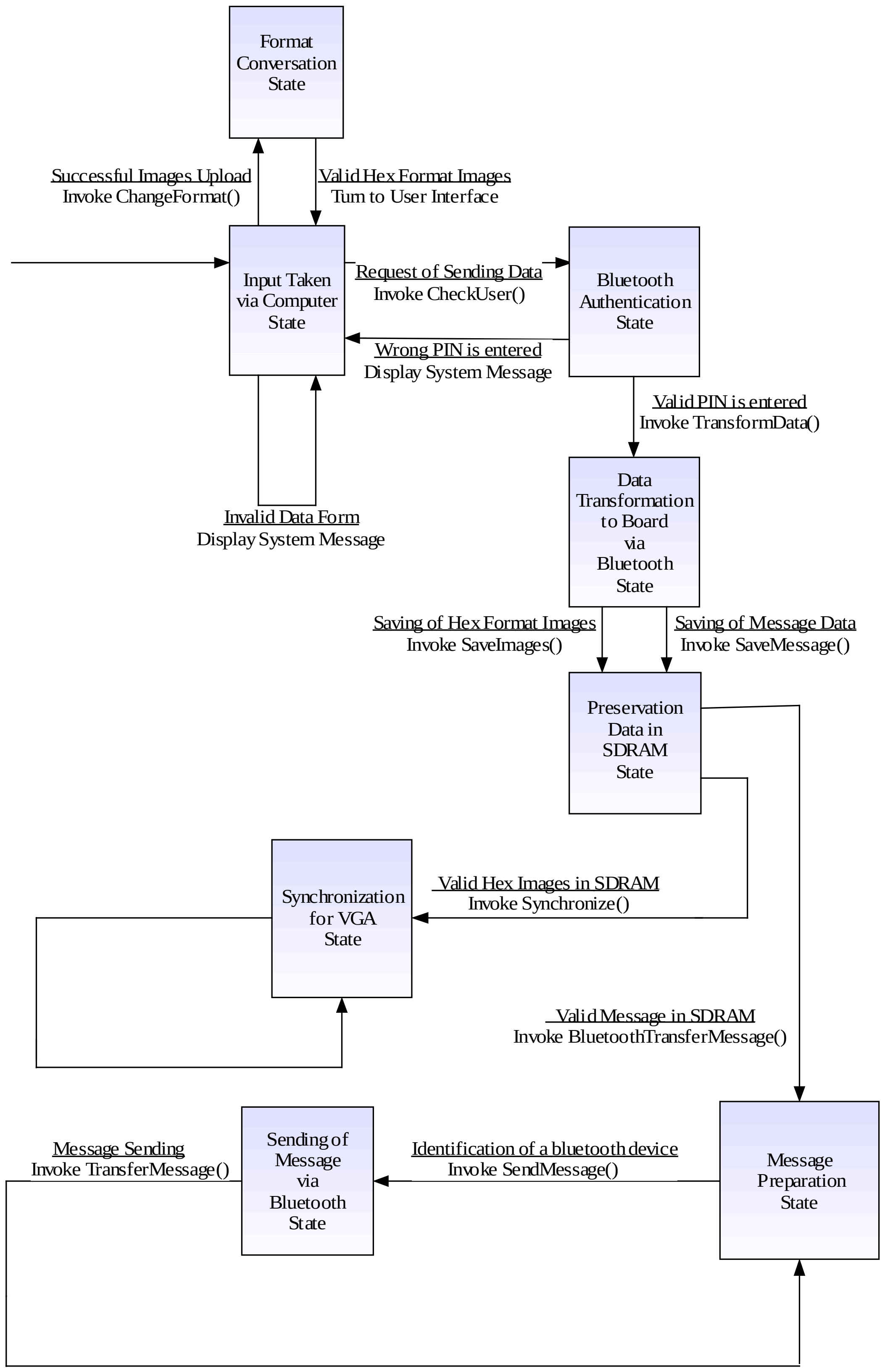
7.4 Establish Communication:

When the “Serial Port” service is found and the service record is obtained, this process first establishes a connection with the bluetooth evaluation board. After the communication is successfully established, this process starts sending data to the evaluation board. The data sent consists of the HEX formatted file produced by “2.2 HEX Formatted File Formation”, and the txt formatted information file produced by “1.1 txt File Formation.” The hex file contains configuration and image data so the data sent by this process arrives to “6.1 Bluetooth Serial Data” as image bluetooth data, configuration bluetooth data, and information bluetooth data.

2.4. BEHAVIORAL DESIGN

3.3.3. State Transition Diagram:

The following diagram illustrates the possible states and their transitions to other states.



Format Conversation State
Successful Images Upload
Invoke ChangeFormat()
Valid Hex Format Images
Turn to User Interface
Input Taken via Computer State
Request of Sending Data
Invoke CheckUser()
Bluetooth Authentication State
Wrong PIN is entered
Display System Message
Valid PIN is entered
Invoke TransformData()
Invalid Data Form
Display System Message
Data Transformation to Board via Bluetooth State
Saving of Hex Format Images
Invoke SaveImages()
Saving of Message Data
Invoke SaveMessage()
Preservation Data in SDRAM State
Synchronization for VGA State
Valid Hex Images in SDRAM
Invoke Synchronize()
Valid Message in SDRAM
Invoke BluetoothTransferMessage()
Sending of Message via Bluetooth State
Message Sending
Invoke TransferMessage()
Identification of a bluetooth device
Invoke SendMessage()
Message Preparation State

3. SYSTEM DESIGN

3.1. USE CASES & USE CASE DIAGRAM

3.1.1. Use Cases

Use Case 1: **Uploading Poster and Information**

This use case is for uploading the poster and event information via a computer or a bluetooth device.

Actors: File Uploader

Pre-Condition: The user should have the right to upload files about the event.

Post Conditions: The poster and information is uploaded successfully to BLUEPOST SYSTEM.

Basic Flow:

1. File Uploader runs the file uploading software of BLUEPOST SYSTEM on his/her computer in order to browse and upload the event image and information files.

2. After browsing the files people uploads the file to be stored in BLUEPOST SYSTEM via interactive bluetooth.

Alternative Flow:

If the file formats that the File Uploader intends to send are not compatible or the files do not contain any information or the File Uploader do not have the right to upload a file (pin error) uploading is simply rejected.

Use Case 2: Store Poster and Information into the System

This use case is for storing poster images and information into the system.

Actors: BLUEPOST SYSTEM

Pre Condition: Poster images and event information have to be already uploaded by the File Uploader correctly.

Post Conditions: The images and information is stored into the system and the images are ready to
be displayed and the information is ready to be sent to Information Recievers.

Basic Flow:
1. The images and information that File Uploader wants to upload come to the system to be stored.
2. The system stores the images and the information.
3. The poster and information is ready to be displayed and sent to the Information Recievers.

Use Case 3: Observing the Digital Poster from the Monitor

Actors: Information Reciever
Pre Condition: The user wonders about the event.
Post Condition: The user gets the information about the event and decides to participate in the event.

Basic Flow:
The user observes the poster and information on the poster.

Use Case 4: Broadcasting Event Information from BLUEPOST SYSTEM to Information Recievers

Actors: BLUEPOST SYSTEM and Information Reciever
Pre Conditions: An event information should already be stored in the BLUEPOST SYSTEM, and a bluetooth connection should already be established between BLUEPOST SYSTEM and Information Reciever.
Post Conditions: The event information has successfully transferred to Information Recievers.

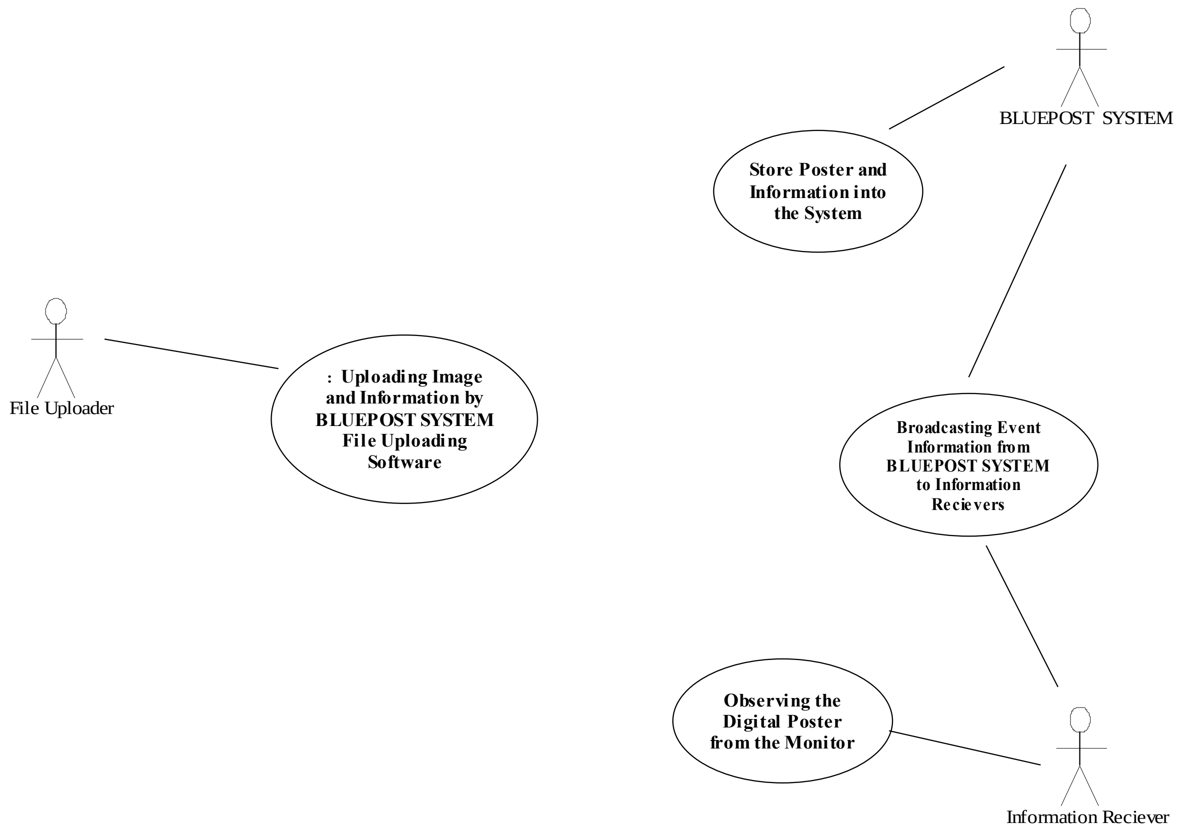
Basic Flow:
1. Information Reciever establish a connection with BLUEPOST SYSTEM that is already ready to establish a connection.
2. File transfer operation occurs.
3. Connection closes after successful completion of File Transfer.

Alternative Flows:

1. If connection is not established, file transfer request is simply rejected.

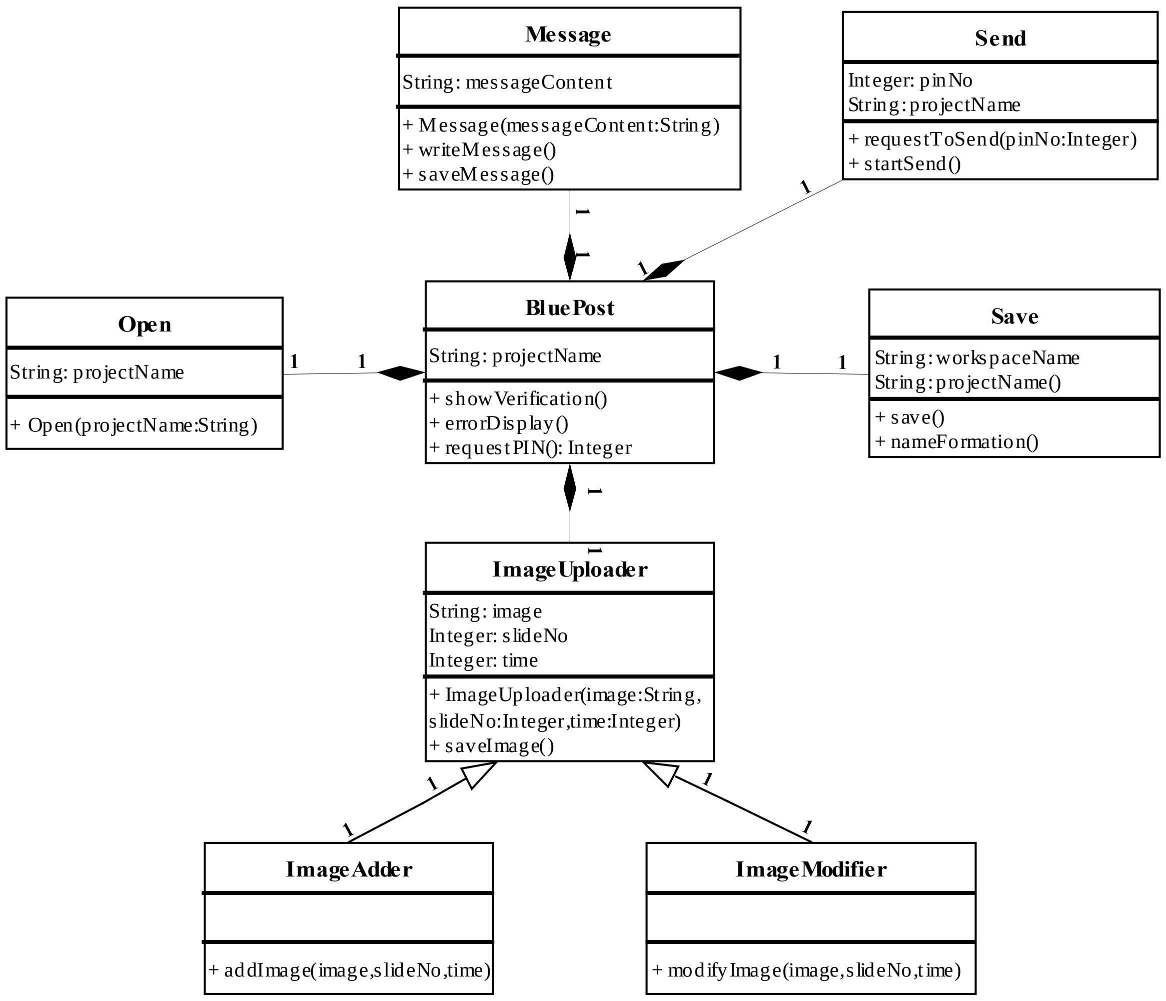
2. If connection is lost during file transfer operation, file transfer request is not completed successfully.

3.1.2. Use Case Diagram


BLUEPOST SYSTEM
Store Poster and Information into the System
File Uploader
: Uploading Image and Information by BLUEPOST SYSTEM File Uploading Software
Broadcasting Event Information from BLUEPOST SYSTEM to Information Recievers
Observing the Digital Poster from the Monitor
Information Reciever

3.2. CLASS AND SEQUNCE DIAGRAMS

3.2.1. File Uploading Module Class Diagram



BluePost Class:

Attributes of the Class:

Attribute Name	Attribute Type	Description
projectName	String	This is the current slide show the user is working on. If the slide show has not been saved yet, it has the name “new folder”.

Methods of the Class:

Method Name	Parameter Types	Return Type	Description
showVerification	void	void	A verification message is displayed when the user adds or modifies an image, or saves the slide show.
errorDisplay	void	void	When the user specifies an already given slide number, an error message is displayed.
requestPIN	void	void	When the user enters a wrong pin number to complete the send process, user is forced to enter another pin.

Message Class:

Attributes of the Class:

Attribute Name	Attribute Type	Description
messageContent	String	This is the message entered by the user that is going to be sent to the bluetooth devices. If the user has not entered a message yet it has a null value.

Methods of the Class:

Method Name	Parameter Types	Return Type	Description
Message	String	void	This is a constructor that creates a message object with the specified message content.
writeMessage	void	void	This function displays a text area with the current message content so that the user can edit the current message. If the message content is null, an empty text area will be displayed.
saveMessage	void	void	When the user enters a message to the

			text area, this function updates the message content with the text in the text area.

Send Class:

Attributes of the Class:

Attribute Name	Attribute Type	Description
pinNo	Integer	This is the pin number entered by the user in order to complete the send process. It should be equal to the pin number of the bluetooth evaluation board.
projectName	String	This is the name of the slide show that the user wants to send.

Methods of the Class:

Method Name	Parameter Types	Return Type	Description
requestToSend	Integer	Boolean	This function returns true if the pin number entered by the user matches the pin number of the bluetooth board.
startSend	void	void	This function starts the sending process of the current slide show. Here the Format Conversion module starts working.

Open Class:

Attributes of the Class:

Attribute Name	Attribute Type	Description
projectName	String	This is the current slide show the user is working on. If the slide show has not been saved yet, it has the name “new folder”.

Methods of the Class:

Method Name	Parameter Types	Return Type	Description
Open	String	void	This constructs an Open object with the given slide show name. If the slide show name is not null, the current images in the show are displayed. Otherwise, the user can start adding new images to a new slide show.

Save Class:

Attribute of the Class:

Attribute Name	Attribute Type	Description
workspaceName	String	This is the name of the workspace our software is running on. The slide shows created by the user will be saved under this directory.
projectName	String	This is name the user has entered to save the current slide show.

Methods of the Class:

Method Name	Parameter Types	Return Type	Description
save	void	void	This function renames the current slide show to the projectName attribute.
nameFormation	void	void	If a slide show that has the name of projectName attribute, this function changes the projectName as explained in module description.

ImageUploader Class:

Attributes of the Class:

Attribute Name	Attribute Type	Description
image	String	Name of the image file the user as selected.
slideNo	Integer	The slide number specfied for the selected image.

Time	Integer	Time duration specified for the selected image.

Methods of the Class:

Method Name	Parameter Types	Return Type	Description
ImageUploader	String,Integer,Integer	void	This is a constructor that creates a ImageUploader object with the specified parameters.
saveImage	void	void	This function saves the image that comes from ImageAdder or ImageUploader class to the current slide show and updates “bilgi.txt” as explained in module description.

ImageAdder Class:

Methods of the Class:

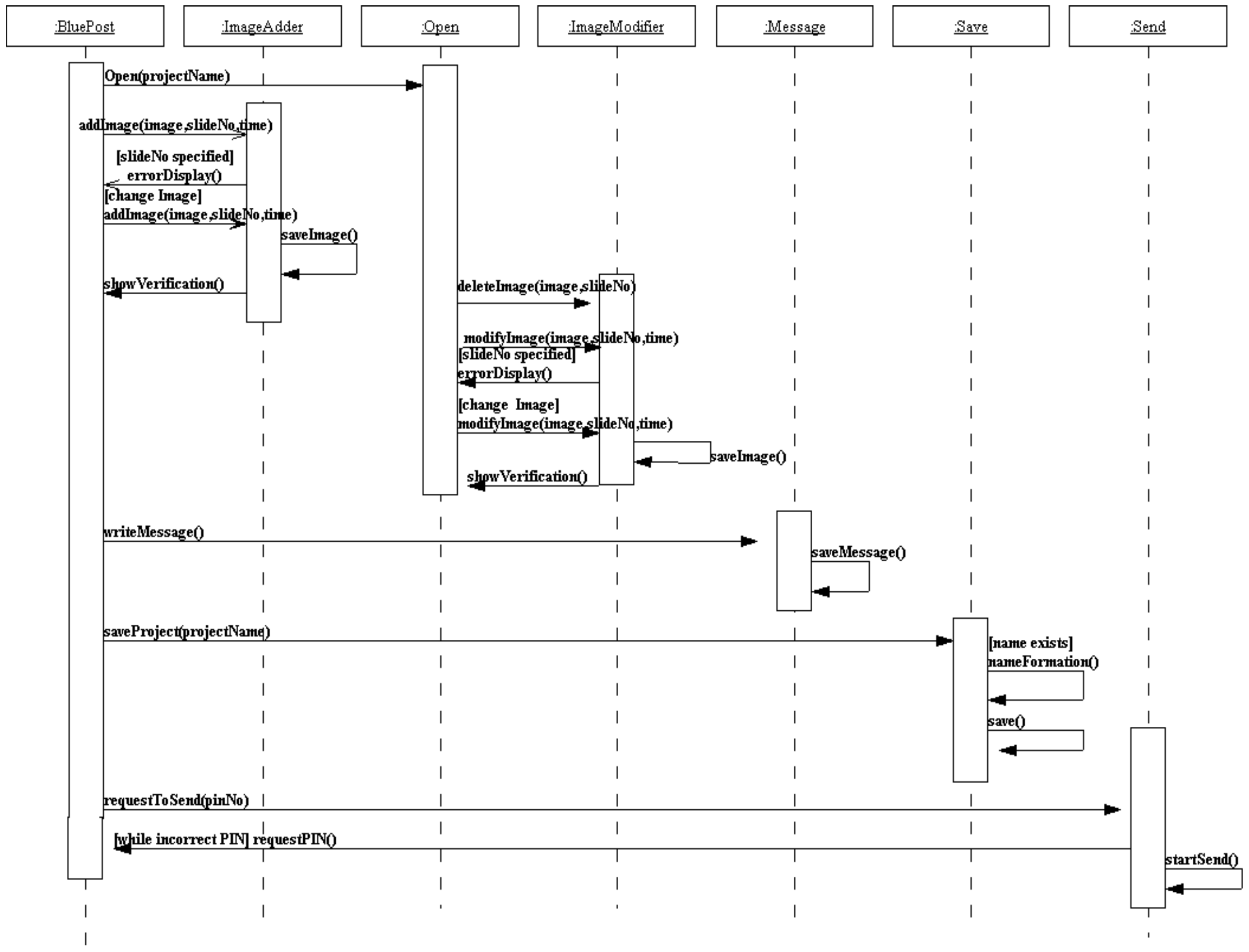
Method Name	Parameter Types	Return Type	Description
addImage	String,Integer,Integer	void	This function adds a new image to the current slide show, with the specified parameters. If the slide number has been specified before, an error message is dispayed. If the user selects to change the image, adding continues.

ImageModifier Class:

Methods of the Class:

Method Name	Parameter Types	Return Type	Description
modifyImage	String,Integer,Integer	void	This function modifies an image in the current slide show, with the specified parameters. If the slide number has been specified before, an error message is dispayed. If the user selects to change the image, modifying continues.
DeleteImage	String,Integer	void	The specified image is deleted from the slide show and the slide number will be freed so that the user can select an image for that slide number.

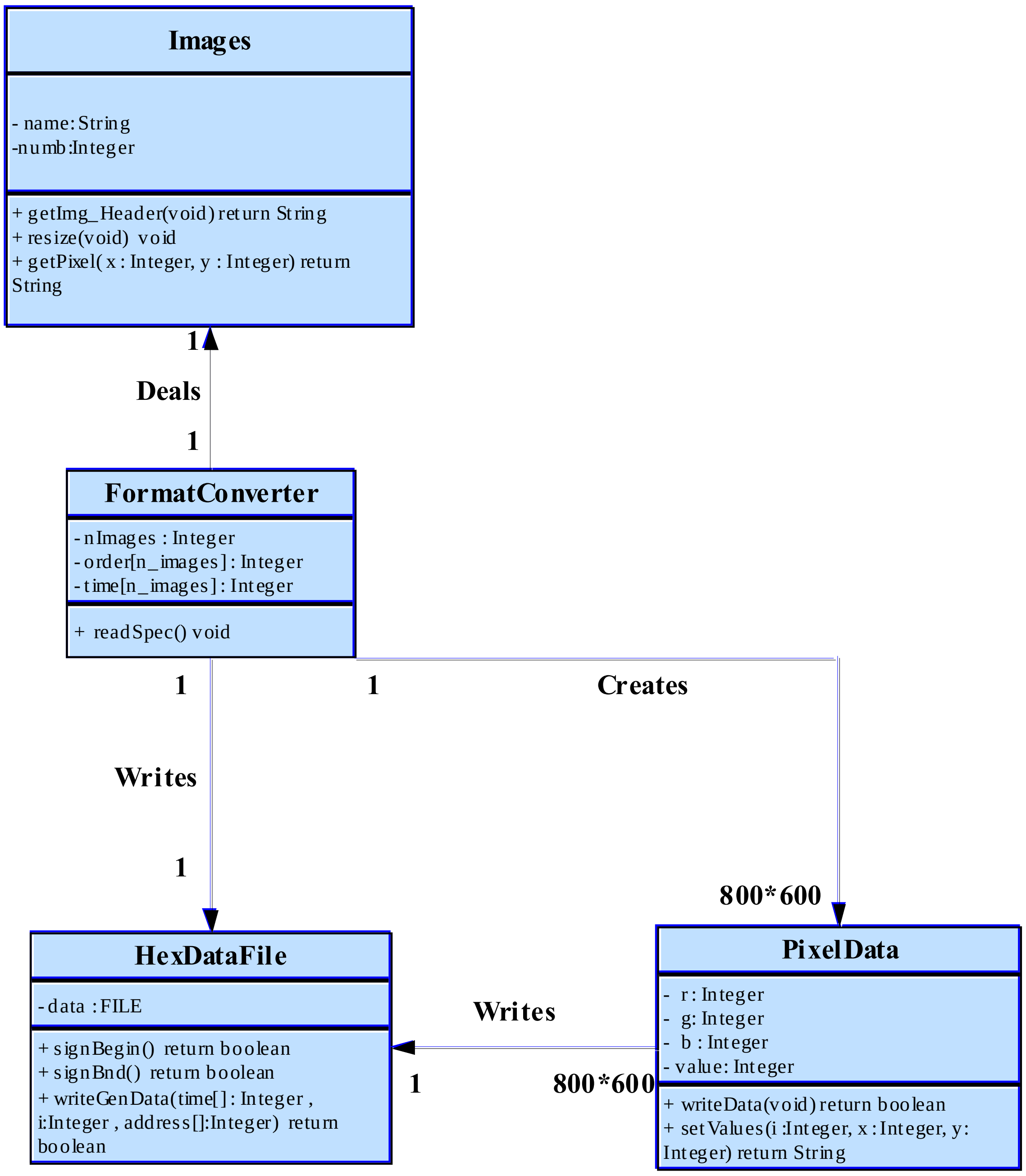
3.2.2. File Uploading Module Sequence Diagram


:BluePost
:ImageAdder
:Open
:ImageModifier
:Message
:Save
:Send
Open(projectName)
addImage(image,slideNo,time)
[slideNo specified] errorDisplay()
[change Image] addImage(image,slideNo,time)
saveImage()
showVerification()
deleteImage(image,slideNo)
modifyImage(image,slideNo,time)
[slideNo specified] errorDisplay()
[change Image] modifyImage(image,slideNo,time)
saveImage()
showVerification()
writeMessage()
saveMessage()
saveProject(projectName)
[name exists] nameFormation()
save()
requestToSend(pinNo)
[while incorrect PIN] requestPIN()
startSend()

Sequence of Events for File Uploading Module	
Main Sequence	1. The user either starts a new slide show or opens a previously saved slide show. 2. The user can add a new image to the slide show. If the slide number is specified, a warning message is displayed. If the user wants to change the image for the specific slide, adding operation continues accordingly. (The for the current slide is changed.) Otherwise the operation is canceled. 3. The user can modify an image in the slide show. If the new slide number entered for the image has been already specified, a warning message is displayed. If the user chooses to modify the image anyway, the operation continues accordingly. (The for the current slide is changed.) Otherwise the operation is canceled.

	4. The user can delete an image from the slide show. 5. The user can enter the message that is going to be sent to the bluetooth devices. If no message has been saved, an empty text area will be displayed. Otherwise, the current message will be displayed. The user will make any changes and save the message. 6. The user can save the current slide show. If the name specified already exists, the name will be changed as described in the module description and the current slide show directory will be renamed. 7. The user can send the slide show contents to our main board. S/he will be asked a pin number, if the pin number matches with the pin number of the bluetooth board, operation continues. Otherwise, user is asked to enter the pin number again.

3.2.3. Format Conversion Module Class Diagram


Images
- name: String
-numb:Integer
+ getImg_Header(void) return String
+ resize(void) void
+ getPixel(x : Integer, y : Integer) return String
1
Deals
1
FormatConverter
- nImages : Integer
- order[n_images] : Integer
- time[n_images] : Integer
+ readSpec() void
1
1
Creates
Writes
1
800*600
HexDataFile
- data : FILE
+ signBegin() return boolean
+ signBnd() return boolean
+ writeGenData(time[] : Integer , i:Integer , address[]:Integer) return boolean
Writes
1
800*600
PixelData
- r : Integer
- g: Integer
- b : Integer
- value: Integer
+ writeData(void) return boolean
+ setValues(i :Integer, x : Integer, y: Integer) return String

Images Class:

Attributes of The Class:

Attribute Name	*Attribute Type*	*Description*
numb	Integer	The identification number of the image.
name	String	Name of the image.

Methods of the Class:

Method Name	*Parameters Types*	*Return Type*	*Description*
getImgHeader	void	String	Returns the header file information of the image file.
resize	void	void	Resize the image file.
getPixel	Integer,Integer	String	Return RGB string of the specified pixel.

FormatConverter Class:

Attributes of The Class:

Attribute Name	*Attribute Type*	*Description*
nImages	Integer	Number of images in the slide show.
order	Integer[]	The specified order of images in the slide show.
time	Integer[]	The specified time durations of images in the slide show.

Methods of the Class:

Method Name	*Parameters Types*	*Return Type*	*Description*
readSpec	Void	Void	Reads 'bilgi.txt' file and initializes class attributes.

HexDataFile Class:

Attributes of The Class:

Attribute Name	*Attribute Type*	*Description*
data	FILE	The hex format file.

Methods of the Class:

Method Name	*Parameters Types*	*Return Type*	*Description*
signBegin	Void	Boolean	Puts a flag to the hex format file to sign the beginning of a new RGB stream.

signEnd	Void	Boolean	Puts a flag to the hex format file to sign the end of a RGB stream.
writeGenData	Integer[], Integer[]	Boolean	Writes the general specifications about slide show.

PixelData Class:

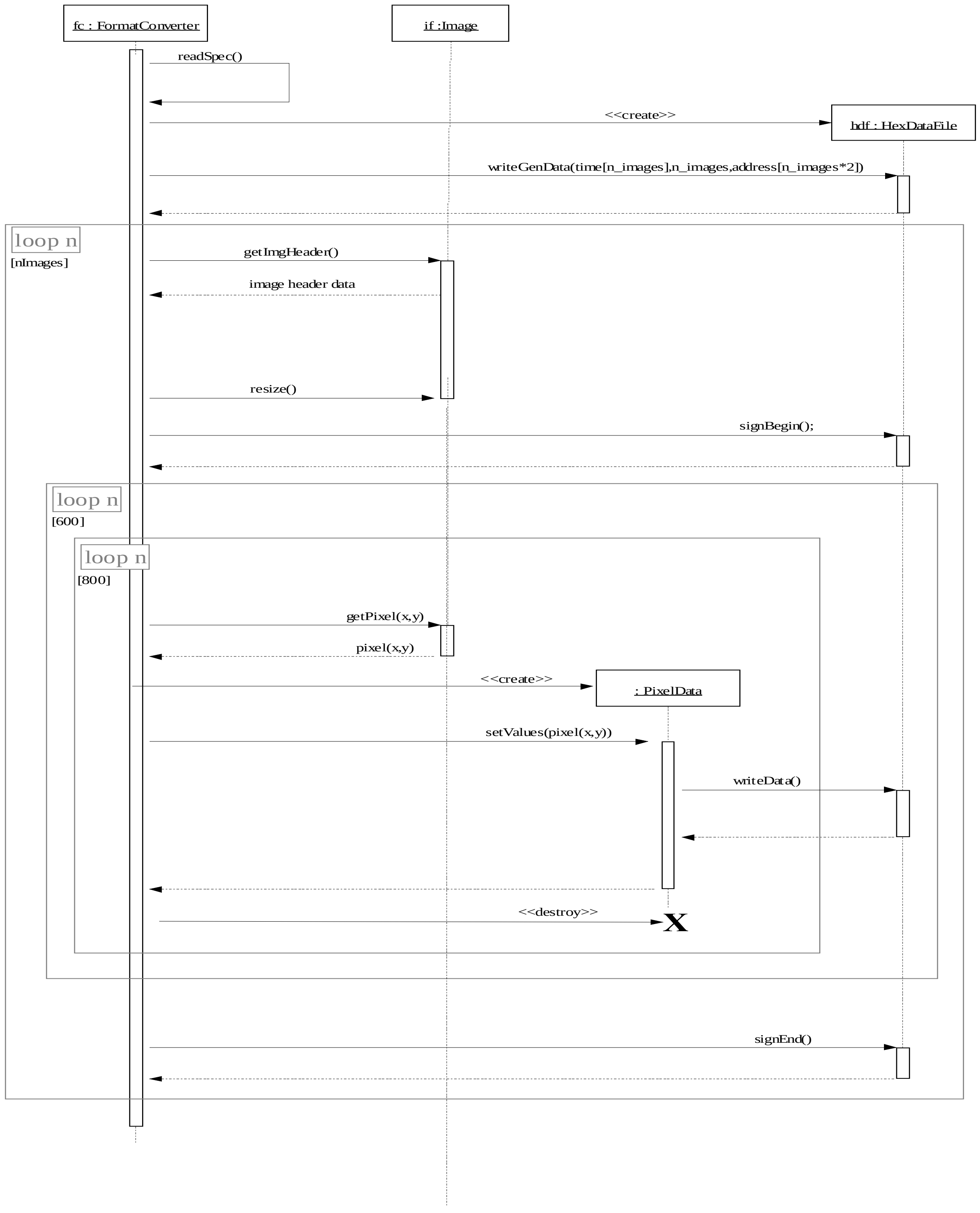
Attributes of the Class:

Attribute Name	*Attribute Type*	*Description*
r	Integer	Red value of a pixel.
g	Integer	Green value of a pixel.
b	Integer	Blue value of a pixel.
value	String	Evaluated RGB value of a pixel.

Methods of the Class:

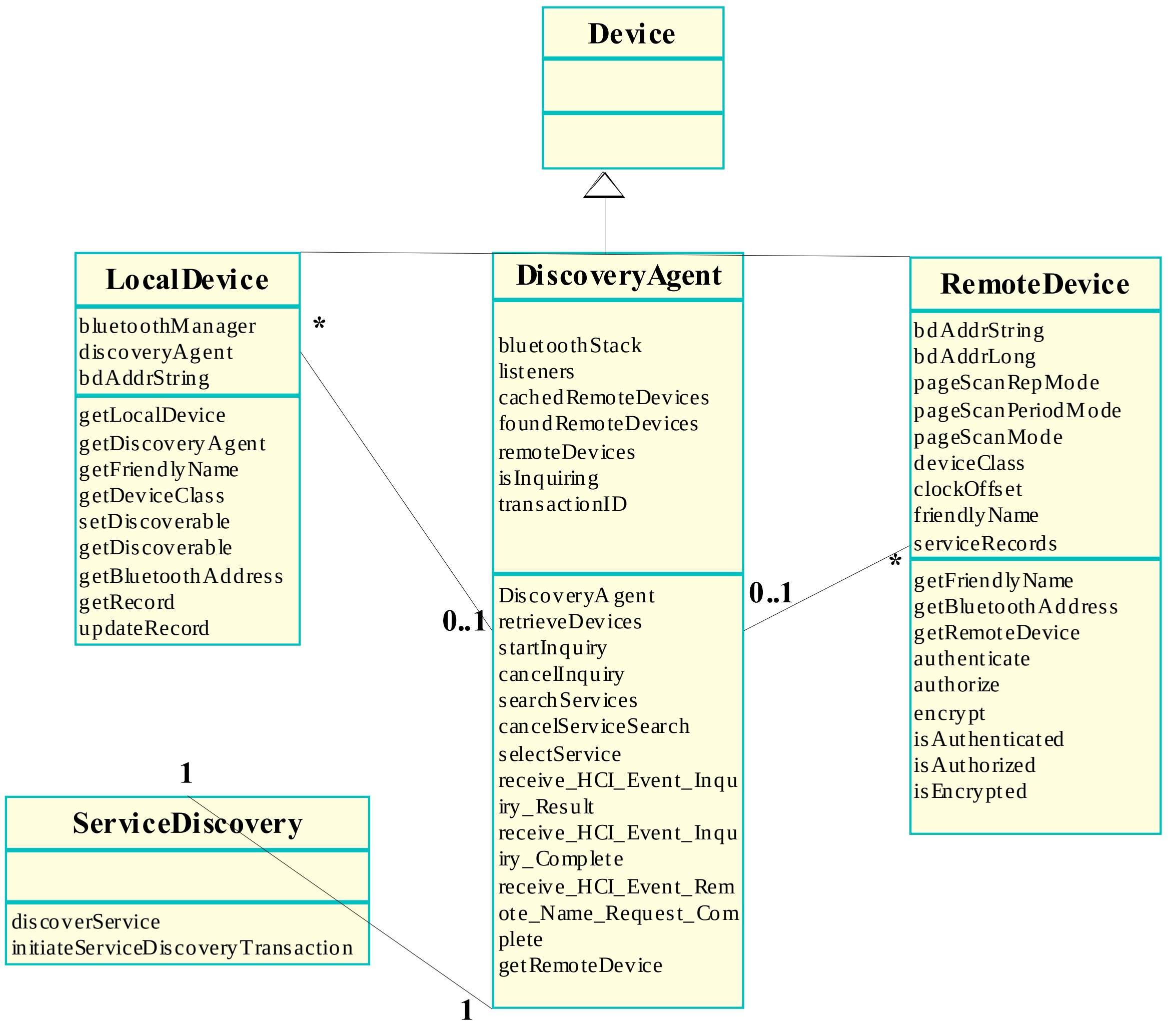
Method Name	*Parameters Type*	*Return Type*	*Description*
writeData	String	Boolean	Writes value attribute in hex format file.
setValues	Integer,Integer	Integer	Initialize the attributes of the class.

3.2.4. Format Conversion Module Sequence Diagram



Sequence of Events for Format Conversion Module:	
Main Sequence	1. The general specifications about slide show is read and written to the hex format file. 2. For every image file in the slide show; a. New image beginning sign is written to the hex format file. b. Image header is read. b. Image is resized. c. Every pixel of the image is written to hex format file after its examination. d. Image ending sign is written to the hex format file.

3.2.5. Sending Data to Board via Bluetooth Class Diagram



LocalDevice Class:

Attributes of The Class:

Attribute Name	*Attribute Type*
bluetoothManager	BluetoothStack
DiscoveryAgent	DiscoveryAgent
bdAddrString	String

Methods of the Class:

Method Name	*Parameters Types*	*Return Type*	*Description*
getLocalDevice	void	LocalDevice	return an object that represents the local Bluetooth device
getDiscoveryAgent	void	DiscoveryAgent	Returns the discovery agent for this device
getFriendlyName	void	String	Retrieves the name of the local device
getDeviceClass	void	DeviceClass	Retrieves the DeviceClass object that represents the service classes, major device class, and minor device class of the local device.
setDiscoverable	mode	Boolean	Sets the discoverable mode of the device
getProperty	Property	String	Retrieves Bluetooth system properties.
getDiscoverable	void	int	Retrieves the local device's discoverable mode.
getBluetoothAddress	void	String	Retrieves the Bluetooth address of the local device.
getRecord	Connection	ServiceRecord	Gets the service record corresponding to a btspp btl2cap or btgoep notifier.
updateRecord	ServiceRecord	void	Updates the service record in the local SDDB that corresponds to the ServiceRecord parameter.

DiscoveryAgent Class:

Attributes of The Class:

Attribute Name	*Attribute Type*
BluetoothStack	BluetoothStack
listeners	Vector
cachedRemoteDevices	Vector
foundRemoteDevices	Vector

Methods of the Class:

Method Name	*Parameters Types*	*Return Type*	*Description*
retrieveDevices	int	RemoteDevice []	Returns an array of Bluetooth devices that have either been found by the local device during previous inquiry requests or been specified as a pre-known device depending on the argument.
startInquiry	int,DiscoveryListener	boolean	Places the device into inquiry mode.
cancelInquiry	DiscoveryListener int	boolean	Removes the device from inquiry mode.
searchServices	int, UUID RemoteDevice ,DiscoveryListener	int	Searches for services on a remote Bluetooth device that have all the UUIDs specified in uuidSet
cancelServiceSearch	int	Boolean	Cancels the service search transaction that has the specified transaction ID.
selectService	UUID, int, boolean	String	Attempts to locate a service that contains uuid in the ServiceClassIDList of its service record.
getDiscoverablereceive_HCI_Event_Inquiry_Result	byte[]	void	Retrieves the local device's discoverable mode. This method is called from BluetoothStack.receive_HCI_Event_Inquiry_Result
receive_HCI_Event_Inquiry_Complete	byte[]	void	Retrieves the Bluetooth address of the local device.This method is called from <code>BluetoothStack.receiv

			e_HCI_Event_Inquiry_Complete
getRemoteDevice	long	RemoteDevice	Resolves a Bluetooth Address to a RemoteDevice.

RemoteDevice Class:

Attributes of The Class:

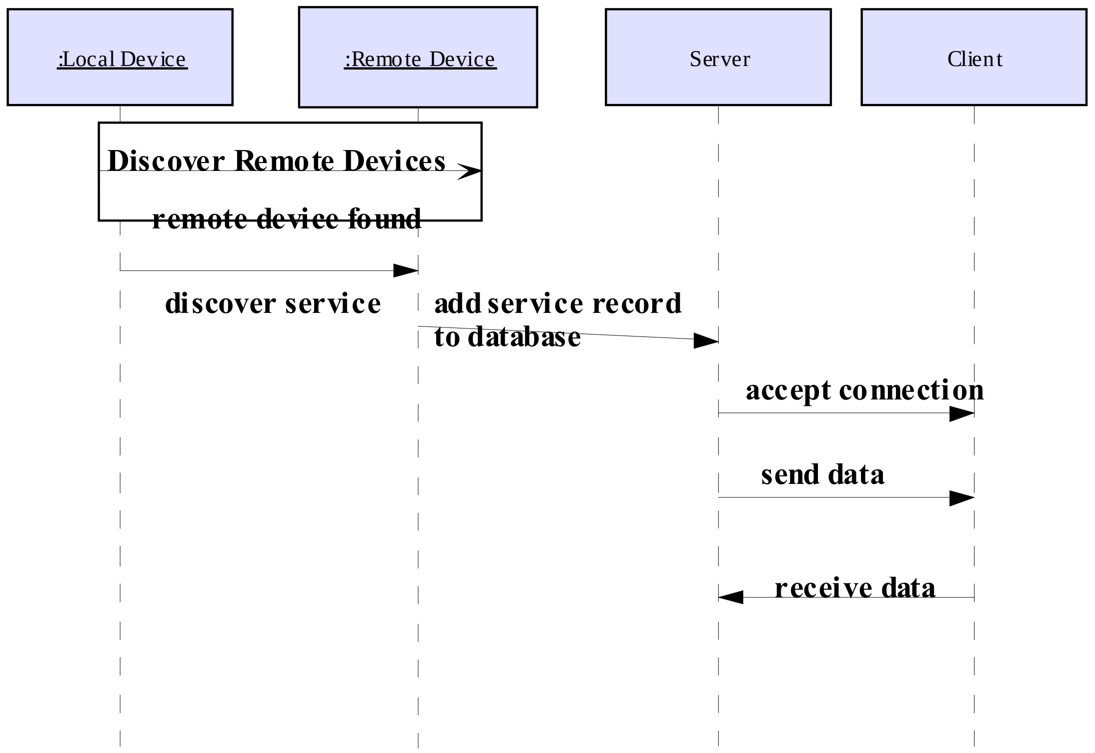
Attribute Name	*Attribute Type*
bdAddrString	String
bdAddrLong	long
pageScanRepMode	byte
pageScanPeriodMode	byte
pageScanMode	byte
deviceClass	DeviceClass
clockOffset	short
friendlyName	String
serviceRecords	HashTable

Methods of the Class:

Method Name	*Parameters Types*	*Return Type*	*Description*
getFriendlyName	boolean	String	Returns the name of this device.
getBluetoothAddress	void	String	Retrieves the Bluetooth address of this device.
getRemoteDevice	Connection	RemoteDevice	Retrieves the Bluetooth device that is at the other end of the Bluetooth Serial Port Profile connection, L2CAP connection, or OBEX over RFCOMM connection provided
authenticate	void	Boolean	Determines if this RemoteDevice should be allowed to continue to access the local service provided by the Connections.
authorize	Connection	Boolean	Sets the discoverable mode of the device
encrypt	Connection,boolean	Boolean	Attempts to turn encryption on or off for an existing

			connection.
isAuthenticated	void	Boolean	Determines if this RemoteDevice has been authenticated.
isAuthorized	Connection	Boolean	Determines if this RemoteDevice has been authorized previously
isEncrypted	void	Boolean	Determines if data exchanges with this RemoteDevice are currently being encrypted.

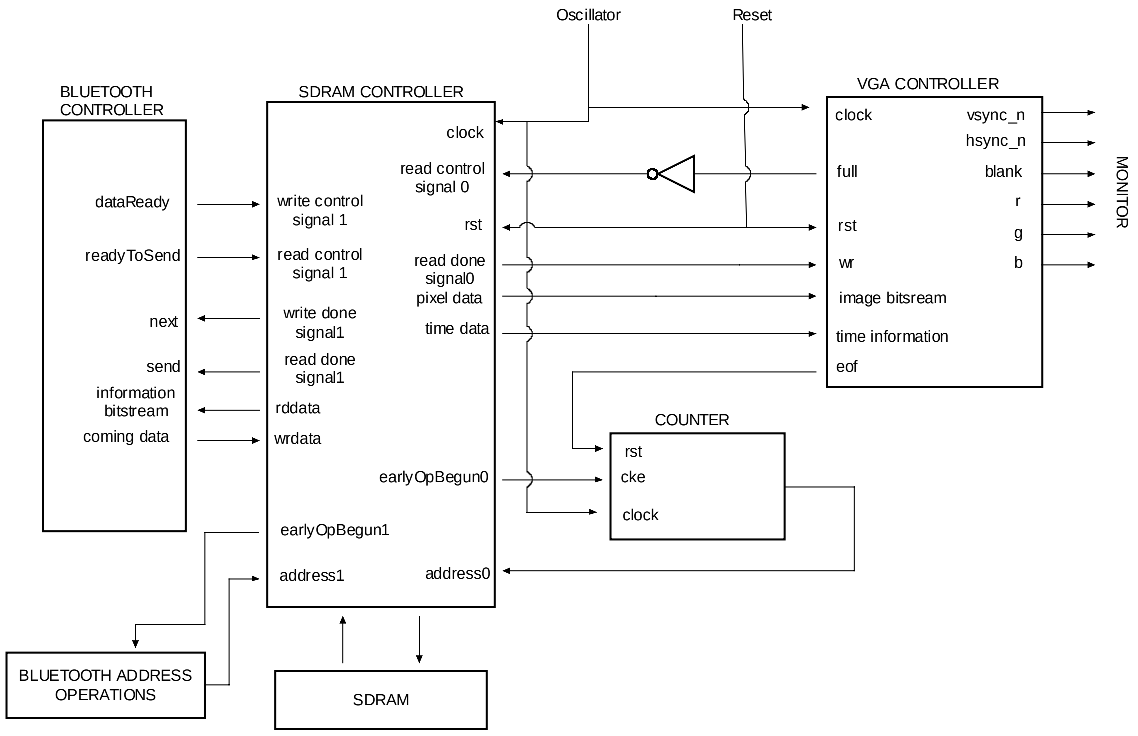
3.2.6. Sending Data to Board via Bluetooth Sequence Diagram



Sequence of Events for BluetoothModule:	
Main Sequence	1. Local Device tries to discover the remote devices around. 2. The service is discovered. • Remote Device accepts connection. • The information transfer between the local and the remote device occurs

4. HARDWARE DESIGN

The following block diagram is a brief overview of the hardware of our system. We have an SDRAM Controller, which communicates with the VGA Controller, which is responsible for the image display, and the Bluetooh Controller, which is responsible for the operations involving bluetooth. SDRAM Controller is like a bridge between the SDRAM and other applications. The Counter calculates the address from which the next pixel information is going to be read for the VGA Controller. Similarly, Bluetooth Address Operations part calculates the address from which next data is going to be read and the address to which next data is going to be written in the Bluetooth Controller. The SDRAM Controller is composed of a “host side” which is connected to our VGA and Bluetooth applications and an “SDRAM side” which is connected to the SDRAM. Since the SDRAM should be in communication with two applications, we will build a dualport module in the SDRAM Controller host side. With this module, the host side will be divided into two and each of the smaller parts will act just like the original host side port. For this reason, some signals related to the SDRAM are duplicated (ex. read control signal, write control signal, earlyOpBegun etc.) and each copy is attached to the one of the two applications. The duplicated signals ending with a “0” are attached to the VGA Controller and those ending with a “1” are attached to the Bluetooth Controller. The host side of the SDRAM, gets the address information from

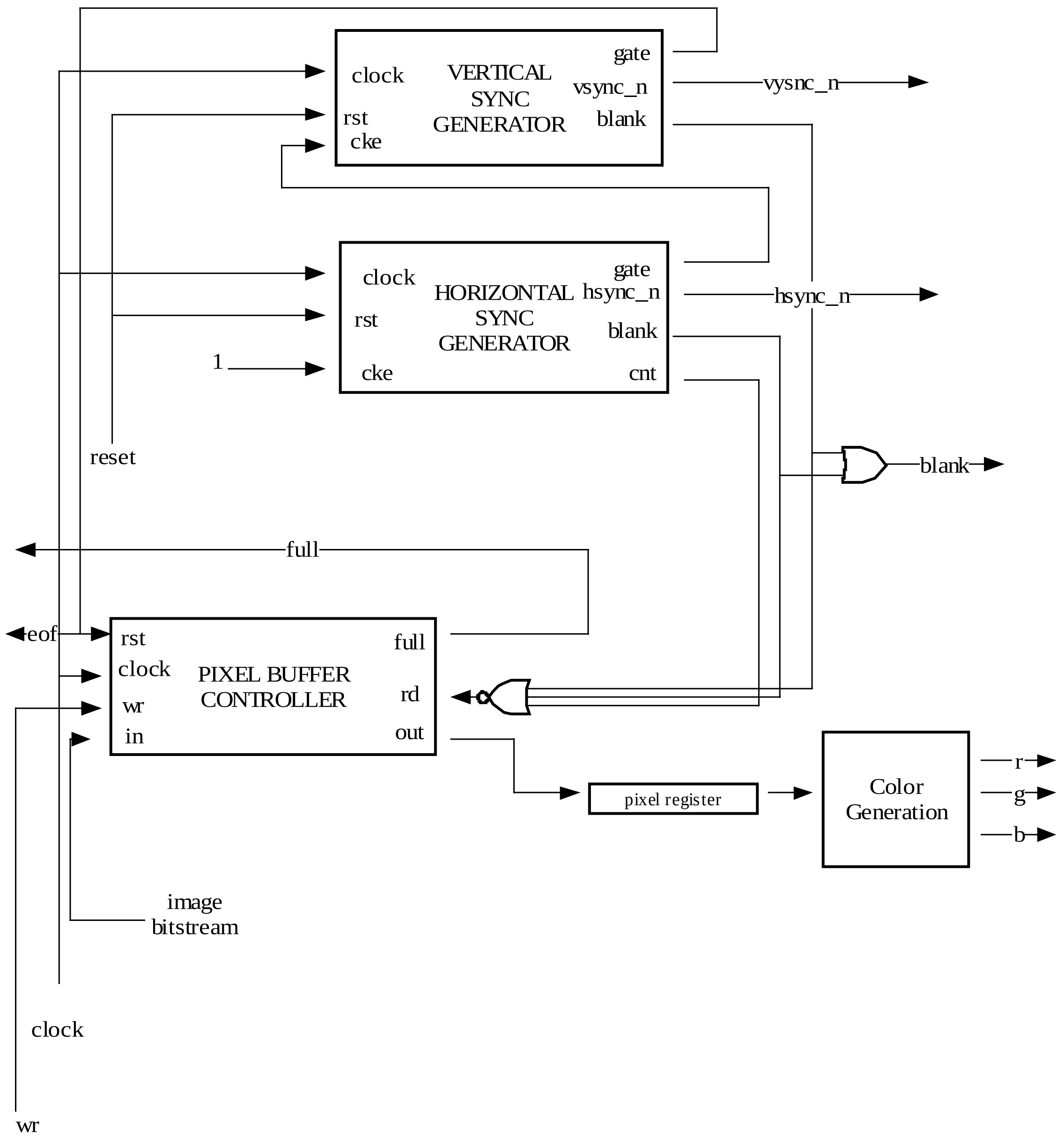


In this block diagram, there are some signals, which are not specified in the data dictionary since these signals take place in the level–3 data flow diagrams. Here is the explanation of these signals:

- cke: This is a clock-enable signal of type std_logic. This signal arrives to the Counter from the SDRAM Controller. When the next current read operation has started, a signal is raised in the SDRAM Controller (earlyOpBegun), and this signal arrives to the Counter so that the counter updates the address to the address of the next pixel, which is going to be read.
- coming data: This is the data bus for the Bluetooth Controller that holds the data to be written to the SDRAM. It has a width of 16 bits and it is of type unsigned. This bus will carry the data coming through our board via bluetooth, namely "hex data" and "information data" as specified in the level-2 data flow diagrams and the data dictionary. This data will be passed to the SDRAM Controller so that it is written to the SDRAM.
- dataReady: This signal is of type std_logic and becomes true (YES) when a byte of data becomes ready to be written to the SDRAM after the serial to parallel conversion operations. This initiates a write request in the SDRAM controller for the bluetooth port.

- earlyOpBegun: This signal is of type std_logic and becomes high just after the indication of the start of a read or a write operation so that the application can update the address value for the next read or write operation.
- next: This signal is of type std_logic and becomes YES when the current write operation in the SDRAM Controller for the bluetooth port has been completed so that the Bluetooth Controller can start processing the next byte of data to be written to the SDRAM.
- pixel Data: This is the data bus for the SDRAM Controller for the VGA port that holds the data read from the SDRAM. It has a width of 16 bits and it is of type unsigned. It holds data related to pixel information.
- rdData: This is the data bus for the SDRAM Controller for the bluetooth port that holds the data read from the SDRAM. It has a width of 16 bits and it is of type unsigned.
- readyToSend: This signal is of type std_logic and becomes true (YES) when the Bluetooth Controller is ready to read data from the SDRAM in order to send bluetooh messages. This initiates a read request in the SDRAM controller for the bluetooth port.
- rst: This is a std_logic signal that resets the SDRAM Controller and causes the initialization of the SDRAM when it becomes high. It also resets the internal circuitry for generation the vertical and horizontal sync signals and the counter in the VGA Controller.
- send: This signal is of type std_logic and becomes true when the current read operation in the SDRAM Controller for the bluetooth port has been completed. Thus the Bluetooth Controller can process this read data.
- time data: This is the data bus for the SDRAM Controller for the VGA port that holds the data read from the SDRAM. It has a width of 16 bits and it is of type unsigned. It holds data related to timing operations such as the number of slides and the time duration of each slide.
- wr: This signal is of type std_logic and becomes true when the read operation of the pixel data is complete so that the read data can be written to pixel buffer.
- wrdata: This is the data bus for the SDRAM Controller for the bluetooth port that holds the data to be written to the SDRAM. It has a width of 16 bits and it is of type unsigned.

The VGA Controller is responsible for first reading the information related to the order of the slide show. This information is written in the first 16 memory words of the SDRAM. When a read operation is in progress if the address corresponds to this first portion of the SDRAM, the data is stored in “time information”. (Contents of the bus “time data” are passed to the bus “time information”.) While this data is being read, the Counter increases the address field by 1 after an “earlyOpBegun” signal is raised. This signal indicates that a read operation has begun and the Counter can update the address for the next read operation. After this information is read, the controller starts reading the pixel information from the rest of the SDRAM and this data is stored in the “image bitsream” and put into the pixel buffer as long as the buffer is not full. (Contents of the bus “pixel data” are passed to the bus “image bitstream”.) When the pixel buffer if full, “full” signal becomes high, and the next read operation is requested when this signal becomes low again. (The pixel buffer is 256*16 which means that it can hold 256 pixels. The data arrives to the buffer with “in” data bus and when a pixel should be passed to pixel register, it is passed with the “out” data bus. Data is passed to the pixel register, as long as no blanking is necessary. In this case “rd” signal becomes high. When the “eof” signal becomes high, the buffer contents are refreshed.) During this operation, the counter increments the address field by 1 unless end of a frame is reached. When the “eof” signal becomes high, indicating that the end of a frame is reached, the Counter checks whether the time duration for the current image has passed or not. (This comparison is done easily, since the time duration for each slide has already been read, and the period of a frame is shown.) If the time duration has passed, the address is updated with the beginning address of the next image. If not, the address is updated with the beginning address of the current image. When a pixel data is extracted from the buffer to the pixel register, the color components are extracted in the Color Generation part of the controller and the pixel is displayed. Meanwhile, Vertical and Horizontal Sync Generators generate appropriate signals for indicating the end of a scan line and a frame. The current pixel is displayed as long as the pixel should not be blanked. The following block diagram is a detailed overview of the VGA Controller:

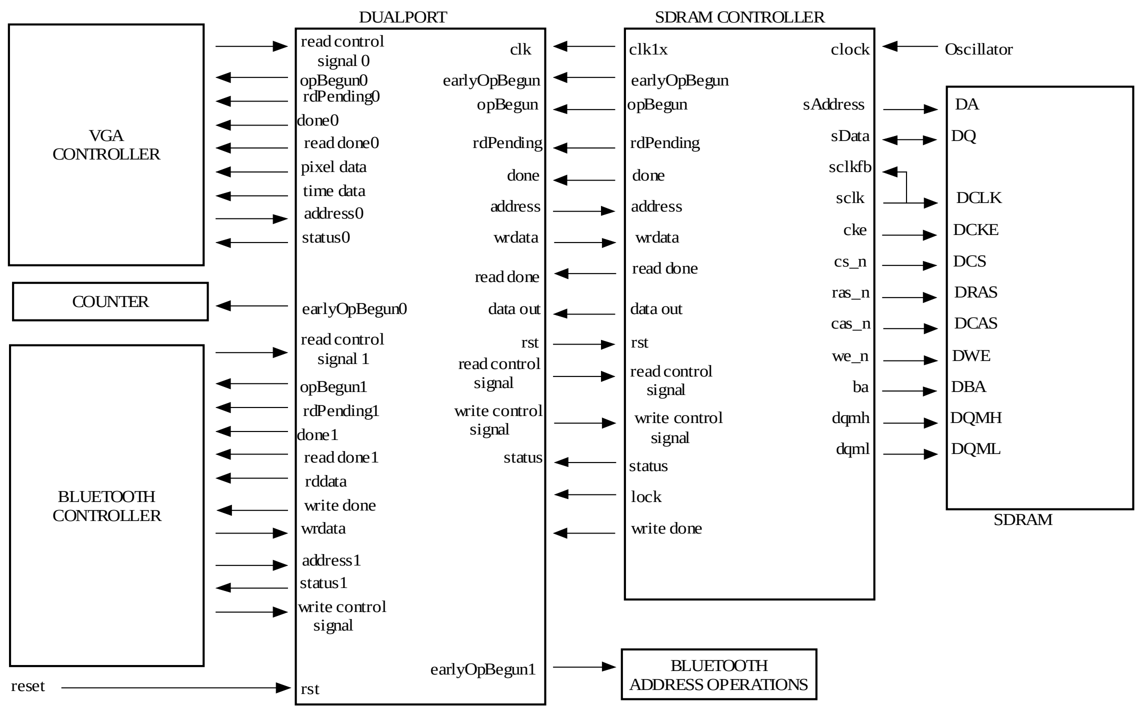


Here is the explanation of the signals that are not covered in the data dictionary:

- in: This is the data bus connected to the pixel buffer that holds the image bitsream arriving to the VGA Controller. It is of type unsigned and has a width of 16 bits.
- out: This is the data bus connected to the pixel register that holds the data extracted from the pixel buffer. It is of type unsigned and has a width of 16 bits.

As already explained, SDRAM Controller implements the dualport feature so that both the VGA Controller and the Bluetooth Controller can access the SDRAM simultaneously. The SDRAM

Controller has two ports for the host side, one for each application. Thus, again some signals are duplicated. There's also the SDRAM side that is attached to the SDRAM itself. SDRAM controller enables the communication between the host applications and the SDRAM itself, by passing information from the host side to the SDRAM side. The data provided by the host applications are passed to the data bus of the SDRAM side. The address provided by the host applications is used to determine the row and column address of the memory word specified and this information is passed to the SDRAM side by "sAddr" bus. When a read or write operation begins, the controller raises the "opBegun" signal. This signal causes the "earlyOpBegun" signal to become high with the following rising edge of the clock. The "earlyOpBegun" signal arrives to both Counter and Bluetooth Address Operations, and the address is updated for the next memory operation. When the current read operation is completed, the "read done" signal becomes high. Likewise, when the current write operation is completed, "write done" signal is raised. When the current operation is a read operation, the data read is put on the "sData" bus by the SDRAM and the controller passes this data to the "data out" bus. Then for the VGA Controller, the "data out" bus is copied either to "pixel data" or "time data" bus according to the address. (Explained above.) For the Bluetooth Controller, "data out" bus is copied to the "rddata" bus. For a write operation, Bluetooth Controller puts the data to be written to the "wrdata" bus. This bus is passed to the SDRAM side of the SDRAM controller and then copied to "sData" bus of the SDRAM. Here's the more detailed block diagram for the SDRAM Controller:



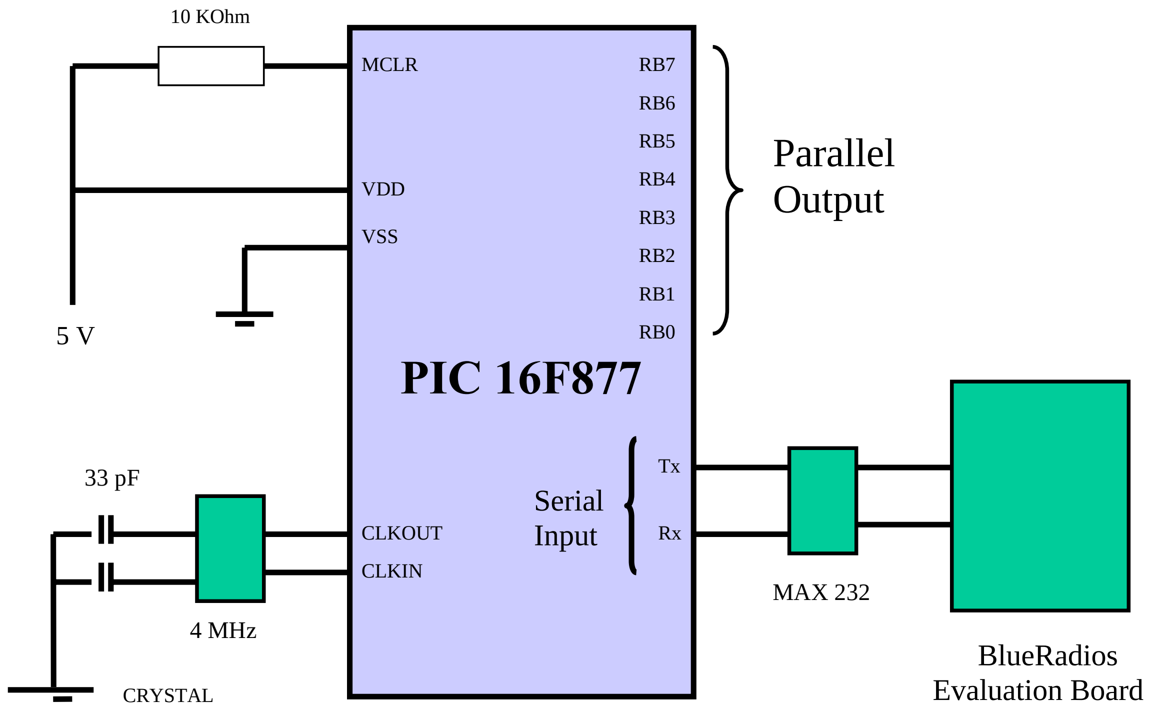
The definitions for the new signals in this block diagram are as follows:

- ba: This two-bits wide unsigned signal chooses one of the four memory banks in the SDRAM.
- cas_n: This std_logic signal is the column address strobe and when a data is needed from the SDRAM, this signal is activated to specify the column where the data is.
- clk: This signal, which if of type std_logic, is the master clock input coming from the oscillator.
- clk1x: This std_logic signal is derived from the master clock input and it clocks the host side of the circuit.
- cs_n: This std_logic signal derives the chip select input of the SDRAM.
- data out: This data bus holds the data read from the SDRAM. This data must be processed by the host side logic on the rising clock edge after the done (or read done) signal becomes high.
- done: This std_logic signal becomes true when the current read or write operation is completed. This signal remains high for a single clock cycle.
- dqmh: This std_logic signal becomes high when the upper byte of the SDRAM data bus is enabled.
- dqml: This std_logic signal becomes high when the lower byte of the SDRAM data bus is enabled.
- lock: This std_logic signal becomes YES when the clk1x signal is synchronized with the master clock so that the host side circuit operations can be clocked.
- opBegun: (revisited) This std_logic signal becomes high to indicate the initiation of a read or write operation. In fact, the previously defined "earlyOpBegun" signal becomes high immediately after this signal is activated.
- ras_n: This std_logic signal is the row address strobe and when a data is needed from the SDRAM, this signal is activated to specify the row where the data is.
- rdPending: This is a std_logic signal and becomes true if there are any read operations in the pipeline that have not delivered their data yet.
- sAddress: This 12-bit wide bus contains the row and column address fields of the SDRAM memory location. This information is extracted from the address coming from the host side.
- sclk: This std_logic signal is obtained from the master clock and derives the clock input for the external SDRAM.

- sclkfb: This std_logic signal is a copy of the sclk signal with delays concerning the passage of this data from the FPGA to the SDRAM and back to the FPGA. This allows the synchronization between the FPGA and the SDRAM operations.
- sData: This is a 16 bit wide data bus. The data to be written exits from the FPGA and the data read exits from the SDRAM through this bus.
- status: This std_logic_vector bus, which is four bits wide, holds the current status of the SDRAM.
- we_n: This std_logic signal derives the write-enable input of the SDRAM and becomes high when write operations will occur.

The Bluetooth Controller is responsible for the communication between the XSA board and the BlueRadios Evaluation board. When a user wants to send the slide show contents, the hex file and the txt file are sent to the BlueRadios Evaluation Board as already explained in the description of the modules. Once this data arrives to bluetooth board, it needs to be transferred to the XSA board. Because, data arrives to the bluetooth board serially and there's no serial port on the XSA board, we will make a serial to parallel conversion. For this purpose we design a 16F877 PIC, as described earlier. The bluetooth board will be connected to the serial port of the pic and as data arrives to the pic, USART interrupt will be raised. When the interrupt is raised, we will read the coming bit and store in a register. We will append the next coming seven bits to the same register and obtain a data of 8 bits. Then we will put this data to the parallel port of the pic, from where it will come to the parallel port of the XSA board.

The following block diagram shows the connection between the pic and the bluetooth board:

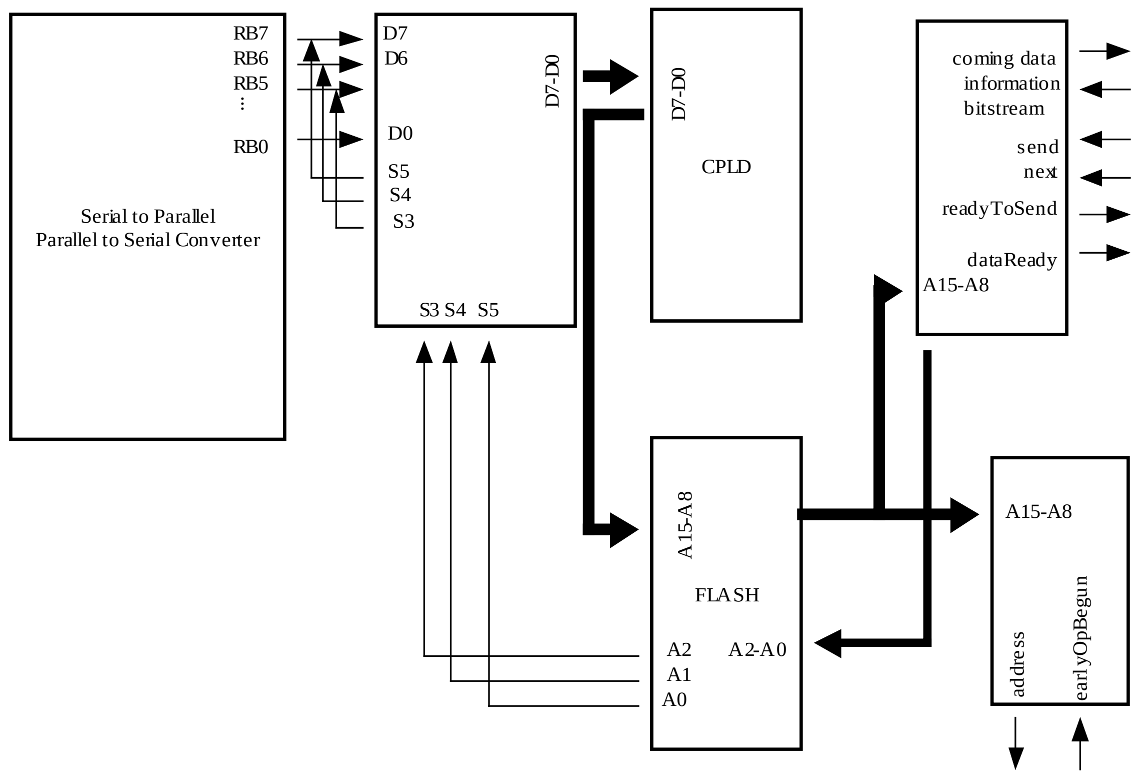


In this diagram there are some signals that need to be explained:

- RB7…RB0: These are the parallel port pins of the pic which will be connected to the parallel port of the XSA board. When 8 bit serial data is received, this data will be put to these pins so that they are transferred to the XSA board. Similarly, when data arrive from the XSA board to pic, it will arrive on these pins.
- Rx: This is the serial data receiver pin. Data coming from the bluetooth board will be received through this pin.
- Tx: This is serial data transmitter pin. While sending data from the pic to the bluetooth board, data will be put to this pin.

While sending bluetooth data to the XSA board, we will receive bluetooth data bit by bit through the serial port of the pic. When we have received 8 bits of data, we will put this data to the parallel port of the pic. Then this data will arrive to the parallel port of the XSA board. Once data comes to the parallel port of the XSA board, it will pass through the CPLD and arrive to the FPGA as already described in the description of the modules. While retrieving the hex data, first address data will arrive, then the data that is going to be written to that address. (The reason is that, the hex file we create in the format conversion module has an address followed by data that needs to be written to that address.) In our VHDL code, the address data bus is composed of 24 bits. Thus, after receiving 8 bits of data from the parallel port, we will receive the next two 8-bit data and combine them to make a 24-bit data. This data will be put to the “address” bus by the Bluetooth Address Operations. Then, the next two coming 8-bit of data will be combined to obtain a 16-bit data. This data will be presented to the “coming data” bus. With the given address and the data, the write operation can be carried out in the SDRAM Controller. This process will be repeated until all the hex file is retrieved. (Since we know the number of images sent and the number of bytes of each image, we will be able to recognize when all the hex file has been retrieved.) After that we will start retrieving the contents of the txt file. The address range for the hex data is predetermined by us and we will start writing the txt file contents at the end of this address range. We will again combine the coming two 8-bit data to produce a 16-bit data and put this into the “coming data” data bus. When the “earlyOpBegun” signal comes to the Bluetooth Address Operations, the current address will be incremented and the address for the next write operation will be defined. This way we will know to which address we will write the coming data. Thus, at the end we will be able to store the data sent by the user to the SDRAM.

While VGA operations are in progress, we will send bluetooth messages to the nearby devices. For this process, we will need to send the txt file content stored in the SDRAM to the bluetooth board. For this purpose, we will need to first send data from the FPGA to the parallel port of the SDRAM. The address from which data is read is obtained from the Bluetooth Address Operations. Since we know the beginning address of the txt data, we will incerement the address when the earlyOpBegun signal goes high. FPGA uses the S5, S4, S3 status lines of the parallel port to send data to the parallel port. The FGPA, drives the A2, A1, A0 address lines of the Flash and this in turn drives the S5, S4, S3 address lines of the parallel port. Then the data on these lines will be put to the D7, D6, and D5 data lines of the parallel port. Then this data will arrive to the pins RB7, RB6, RB5 of the pic. We will understand that data has come by the port change interrupt. This interrupt is generated when a change occurs in one of the RB7-RB4 pins. In order to guarantee that every data we send causes an interrupt, while mapping the S5-S3 pins to the D7-D5 pins, we will put a 0 and 1 to the D4 pin interchangeably. This way, the value of the RB4 pin will change with each coming data and an interrupt will be raised. We will send 3 bits followed by 3 bits followed by 2 bits of data from the FPGA. Similary, we will receive 3 bits followed by 3 bits followed by 2 bits of data in the pic. This data will be combined to produce 8-bit data. Then, the pic will send this data to bluetooth board, through the serial port bit by bit. Thus we will be able to send the bluetooth message data to the bluetooth board. The following block diagram describes the hardware used in this step:



Here, “Serial to Parallel, Parallel to Serial Converter” refers to the pic whose block diagram was included above.

There are some signals that need to be explained in this diagram:

- D7-D0: These are the data pins of the parallel port of the XSA port. They are used to send and receive data.
- S5,S4,S3: These are the status lines that carry the data sent from the FPGA to the parallel port in the XSA board.
- A15-A8: When the parallel port sends data to the FPGA, these address lines are derived with the data on D7-D0 and carried to the FPGA.
- A2,A1,A0: When FPGA want to send data to the parallel port, these address lines are derived which drive the S5,S4,S3 lines of the parallel port and carry information.

5. SYNTAX SPECIFICATIONS

According to conversations between our group members, we decided not to limit our members with strict syntax specifications. We only declared some basic rules for the understandability of our code and easy analyzing.

The most important point of our specifications is using comments efficiently. For every new item in the code (except local variables), we decided to force members to include comments about the process in a detailed way. Moreover we expect the members to include a text file that tells the capabilities and constraints of their code, for every new created package.

Near this, because we will use only two programming languages in our project, which are Java and VHDL, we decided to use the syntax conventions of these languages for a more considerable code-design.

For Java, the names of the classes will be mixed-case, starting with a capital letter. If the name is composed of a phrase, each word in the phrase will start with a capital letter. *(ex. ClassName)* The constant names will be all upper case. Words in phrases will be separated by underscores. *(ex: CONSTANT_NAME)* Finally function and variable names will start with a lower-case word and if the name contains other names, then those words will start with a capital letter. *(ex: functionName)*

For VHDL, the names of the generic variables will be all in upper case, words in a phrase being separated by underscores. *(ex: LINES_PER_FRAME)* The variable names assigned to the ports will be lower case and words will be again separated by underscores. (*ex: pixel_data_in*) Constant names will be similar to generic variable names. They will be upper case and the words will be separated by underscores. (*ex: HSYNC_START*) Component, architecture, and entity names will be lower case where words are distinguished again by underscores. (*ex: component, component_arc, sync*) Procedure names will be defined similarly. (ex: map_pixel)

6. DEVELOPMENT SCHEDULE AND GANNT CHART

Besides completing the required reports before the deadlines, we also planned our other work. We have completed the design part of our project with this Final Design Report.

Besides the design process, we have started working on the implementation part of our project. We started working on the prototype development by the end of November. First we analyzed the design examples about the VGA port and started working on our own design. We successfully made a transition between different images in a slide show manner. Then we worked on reading data from the SDRAM according to the address ranges. We assigned the data read from the first 16 memory words to one data bus and the rest of data to another data bus. This is what we should do in order to distinguish the configuration data about the slide show from the image data itself. We have little work to do about the usage of the VGA port and we plan to finish this part by the end of February as indicated in the Gantt chart.

In addition, we started implementing the user interface. We have finished the parts we have explained in the “File Uploader Module”. We are planning to complete the remaining parts by the end of January. Meanwhile, we started working on the “Format Conversion” module. Now, we are able to extract pixel information from image files. What is left is writing this information in a .hex file and embedding this code in the user interface design. We will accomplish this task before the prototype demonstrations.

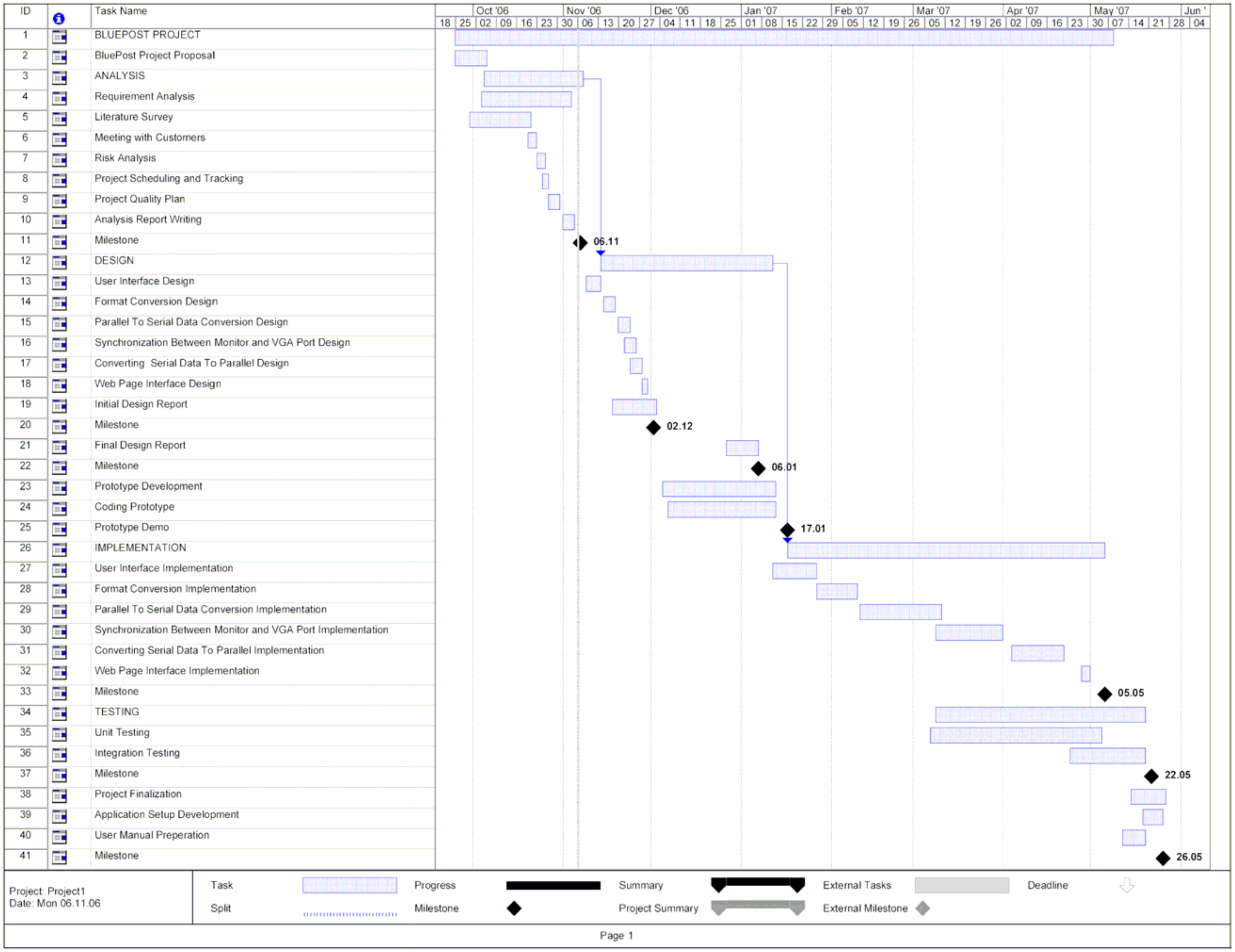
For the prototype demonstrations, we are responsible for sending bluetooth messages when the BlueRadios Evaluation Board is attached to our computer. We have to accomplish this task by AT Commands. We are currently working on this part of our project.

We are planning to work on the design of the 16F877 pic during the vacation and we hope to finish this task by the end of March as shown in the Gantt chart. Meanwhile, some of the group members will work on programming the CPLD so that it acts as a bridge between the FPGA and the parallel port. Once we design this pic, we will start working on programming this pic and establishing a connection with this pic and the XSA board.

Besides these tasks, we have to send slide show contents to our BlueRadios Evaluation board. We have started working on this part of our project. We found the Java Bluetooth Stack (BlueCove) we will make use of. So far, we were able to discover the nearby bluetooth devices and search for their services. What is left is establishing a connection with the BlueRadios Evaluation Board and sending data by the serial port service. We are planning to focus on this part after we progress in our pic design.

According to our process so far, we believe that sending data to our XSA board from the bluetooth evaluation board and vice versa is one of the most challenging parts of our project. In the beginning of the second term, we will focus on this part. Meanwhile, we will try to make progress in the other parts.

In the second term, one other task we will work on is the web page design. We are planning to work on this part at the end of March as indicated in the Gantt chart.
We are planning to finish the implementation of our project by the beginning of April. From then on we will focus on testing.



ID	Task Name
1	BLUEPOST PROJECT
2	BluePost Project Proposal
3	ANALYSIS
4	Requirement Analysis
5	Literature Survey
6	Meeting with Customers
7	Risk Analysis
8	Project Scheduling and Tracking
9	Project Quality Plan
10	Analysis Report Writing
11	Milestone
12	DESIGN
13	User Interface Design
14	Format Conversion Design
15	Parallel To Serial Data Conversion Design
16	Synchronization Between Monitor and VGA Port Design
17	Converting Serial Data To Parallel Design
18	Web Page Interface Design
19	Initial Design Report
20	Milestone
21	Final Design Report
22	Milestone
23	Prototype Development
24	Coding Prototype
25	Prototype Demo
26	IMPLEMENTATION
27	User Interface Implementation
28	Format Conversion Implementation
29	Parallel To Serial Data Conversion Implementation
30	Synchronization Between Monitor and VGA Port Implementation
31	Converting Serial Data To Parallel Implementation
32	Web Page Interface Implementation
33	Milestone
34	TESTING
35	Unit Testing
36	Integration Testing
37	Milestone
38	Project Finalization
39	Application Setup Development
40	User Manual Preperation
41	Milestone

Project: Project1
Date: Mon 06.11.06

Page 1

REFERENCES

[1] XSA-3S1000 Board User Manual
http://www.xess.com/manuals/xsa-3S-manual-v1_0.pdf

[2] VGA Generator for the XSA Boards
http://www.xess.com/appnotes/an-101204-vgagen.pdf

[3] Spartan-3 Capabilities
http://www.xilinx.com/products/silicon_solutions/fpgas/spartan_series/spartan3_fpgas

[4] Xilinx : Logic Design
http://www.xilinx.com/ise/logic_design_prod/index.htm

[5] XSA Board SDRAM Controller

http://www.xess.com/appnotes/an-071205-xsasdramcntl.html

[6] VGA Generator Test Application with an Embedded Parallel Port Interface
http://www.xess.com/appnotes/an-103005-vgagen.html

[7] Bluetooth Radios, A Wireless World
http://www.blueradios.com/evaluationkit.htm

[8] Getting Started with Java and Bluetooth

http://today.java.net/pub/a/today/2004/07/27/bluetooth.html

[9] The Java APIs for Bluetooth Wireless Technology

http://developers.sun.com/techtopics/mobility/midp/articles/bluetooth2/

[10] Sundar Rajan, “Essential VHDL : RTL Synthesis Done Right”, USA:Sundar Rajan, 1998.

[11] Downloading XESS FPGA and CPLD Software Tools : img2xes.zip
http://www.xess.com/ho07000.html