2007

# System approach to embedded system design

Vikram Prabhakar Mehendale
*University of South Florida*

# System Approach to Embedded System Design

by

Vikram Prabhakar Mehendale

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Electrical Engineering
Department of Electrical Engineering
College of Engineering
University of South Florida

Date of Approval:
April 2, 2007

Keywords: FPGA, Cyclone II, CMOS sensor, VHDL, Video surveillance

## DEDICATION

I dedicate this thesis to my parents.

## ACKNOWLEDGEMENTS

I offer my deepest and most sincere “thank you” to my major professor, Dr. Wilfrido Moreno, for his encouragement and valuable guidance during this research. I also extend my gratitude to Dr. James Leffew and Dr. Paris Wiley for agreeing to serve on my supervisory committee. I am grateful to Mr. Ronnie Leighty who helped me throughout this research. I would also like to thank my friends and colleagues for their support during this research experience.

## TABLE OF CONTENTS

LIST OF FIGURES .......... iii

ABSTRACT .......... iv

CHAPTER 1: INTRODUCTION AND MOTIVATION .......... 1

1.1 Introduction .......... 1

1.2 The System of Systems Approach .......... 2

1.3 Design Methodology .......... 2

1.4 Top Down Design .......... 3

1.5 Embedded System Design .......... 6

1.6 Programmable Logic .......... 7

1.7 Embedded Processors in Programmable Logic .......... 7

1.8 Overview of Development Kits Used .......... 8

1.8.1 Altera DE2 Board .......... 8

1.8.2 Xilinx XUP V2PRO Board .......... 9

1.9 Motivation .......... 10

CHAPTER 2: BACKGROUND .......... 11

2.1 Related Work .......... 11

2.2 Analysis of Requirements .......... 13

2.3 Altera Cyclone II Device .......... 19

2.4 Kodak KAC-9630 CMOS Image Sensor........................................................20

2.5 The IrDA Interface..........................................................................20

2.6 Altera DE2 Development and Education Board.............................................21

2.7 Design Software and Support ..........................................................22

CHAPTER 3: IMPLEMENTATION ..................................................................23

3.1 Image Capture Module ....................................................................23

3.2 Image Processing Module..................................................................25

3.2.1 System Initialization ...................................................................26

3.2.2 Pixel Comparison........................................................................27

3.2.3 Error Detection...........................................................................27

3.3 IrDA Transmission..........................................................................29

CHAPTER 4: TESTING AND VERIFICATION...........................................................30

4.1 FPGA Verification..........................................................................30

4.2 VHDL Test Bench ............................................................................32

CHAPTER 5: CONCLUSION AND FUTURE WORK..................................................35

5.1 Conclusion .....................................................................................35

5.2 Future Work....................................................................................36

REFERENCES ................................................................................................37

APPENDICES ................................................................................................39

Appendix A: VHDL Source Code .........................................................40

Appendix B: VHDL Test Bench............................................................45

## LIST OF FIGURES

Figure 1: Top Down Design Flow....................4

Figure 2: Embedded System....................6

Figure 3: Altera DE2 Board....................8

Figure 4: Xilinx XUP V2PRO Board....................9

Figure 5: Typical Video Surveillance System....................15

Figure 6: Altera Cyclone II FPGA....................19

Figure 7: Proposed Interface....................24

Figure 8: Image Data Port....................24

Figure 9: Image Processing Module....................25

Figure 10: Port Map for the Image Processing Module....................26

Figure 11: FPGA Design Flow....................30

Figure 12: VHDL Test Bench....................32

## SYSTEM APPROACH TO EMBEDDED SYSTEM DESIGN

Vikram Prabhakar Mehendale

## ABSTRACT

During this research, the concepts of Systems Engineering were applied to embedded system design. The objective was to apply the Systems Engineering methodology to the design of a particular embedded system. A Video Surveillance system was chosen as the particular embedded system. Systems Engineering concepts provide the foundation for an optimized design process and for the coordination between system modules. The functionality of the Video Surveillance system was achieved through the partitioning of the overall system functionality into three separate modules. The three modules were Image Capture, Image Processing and Image Transmission. The methodology employed resulted in a system that was flexible and portable.

The three modules were designed using their own set of specifications and with completely defined linking interfaces. Following a concrete set of specifications resulted in a system, which can be modified at any later stage without the necessity of changing the whole architecture. The Video Surveillance system fulfilled the overall system requirements as well as those imposed by the subsystems. The partitioning of functionality resulted in ease of implementation and better upgradeability. Design based on Systems Engineering concepts provides for ease of integration. In addition, for

modules that follow the same protocol, the existence of well defined interfaces enables connectivity to a variety of external units.

# CHAPTER 1:

# INTRODUCTION AND MOTIVATION

## 1.1 Introduction

The motivation behind this research was the desire to facilitate the embedded system design process by applying the concepts of Systems Engineering. The embedded systems consist of many modules, which are comprised of software components, hardware components and interfaces. All these modules can be independently modeled as complex systems. In order to achieve a correct implementation of a project all of the independent designs must work in synergy. Therefore, application of system design principles to the design of embedded system can dramatically streamline the design work and avoid future problems involved with integrating the modules to constitute a larger system.

The embedded system is a system in which the processing unit is actually embedded between its peripherals and the system is designed to perform some predefined tasks. Being dedicated to certain tasks, the embedded system provides a very efficient solution compared to their general purpose counterparts. The embedded systems are currently used in many applications from automobiles to home appliances. Development of powerful microcontrollers and programmable logic has made the embedded system a very useful solution. Due to continuous development of processors and peripherals the use of embedded systems is rising exponentially.  Using embedded processors and programmable logic makes it simpler to update the system without changing much of the hardware.  The use of EDA (Electronic Design Automation) tools make it simpler to debug the system and provide patches for future problems through the use of advanced synthesis and simulation tools.

## 1.2 The System of Systems Approach

When confronted with designing complex systems it is beneficial to approach the design via an architecture, which is structured as a system of systems.  In this approach the designers identify the system requirements for subsystems, which are based on overall system requirements.  The subsystems are designed independently and then interfaced to achieve the completed system architecture.  This approach simplifies the procedures related to testing, debugging and integration of the subsystems, which are required to insure proper design of the whole system.

Approaching a complex system as a system of systems helps to divide the functionality and verify the working of the subsystems independently.  This design methodology also helps to enhance the interoperability and portability of the design.

## 1.3 Design Methodology

A good design methodology can help the system design process in many ways.  It can help to verify the system for functionality and for errors.  It can help the design team to coordinate the design effort.  The design process should provide a time line for the designers and the deliverables, which are due at different times.  The design methodology involves the global approach of the chief designers to the system design. Since the projects usually involve many teams working on the project at the same time, this methodology can help with coordination, which can be very fruitful for achieving the design goals. The Systems Engineering methodology helps realize a solution, which achieves all the goals of the system including manufacturing cost, performance and power consumption.

## 1.4 Top Down Design

The Top Down Design approach involves design procedures, which are initiated from requirements for system integration. This approach involves arriving at the right solution after considering all the possible alternatives. Every so often, the design effort may focus on trying to fit the solution within the available resources. This approach, which may provide short term gains, may lead to complications while fulfilling future needs and not necessarily arriving at an optimum implementation. The problem may have a large number of possible solutions and selections for the optimum solution based on the target application can provide a number of advantages. This can be considered as the selection of an optimum solution from a larger design space.

The idea of fitting the problem into an already available solution may sound lucrative to the designer due the ease of implementation. There is always a possibility of some other better way to implement the project. The Top Down approach looks at the requirements in an objective way. In this approach the designer will be faced with a large number of different possible ways and then the optimum way is selected.

The selection of an optimum solution is a very important decision since it involves the study of all design categories such as available resources, non-recurring engineering cost, size and power. The selection process for choosing an optimum solution provides a number of advantages in terms of lowering the required resources and better functionality of the solution. Figure 1 illustrates the investigative flow associated with the Top Down Design process. [11]

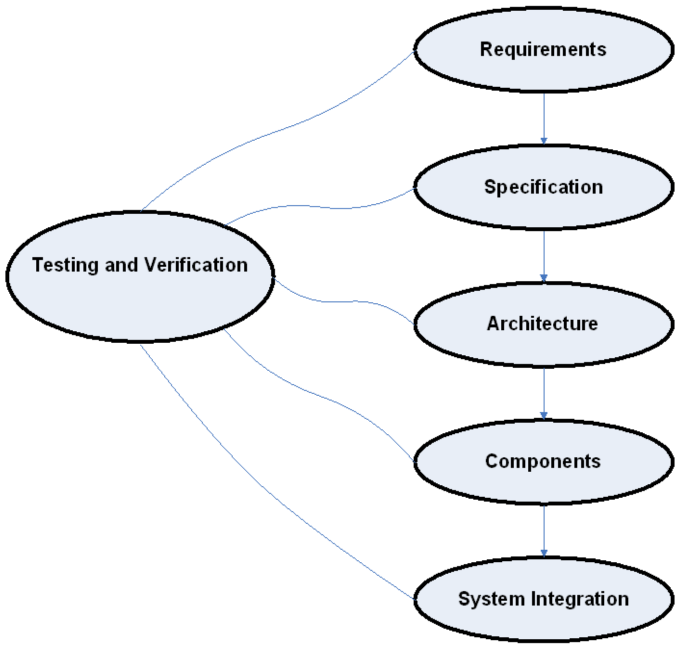

Figure 1: Top Down Design Flow

The requirements are abstract descriptions of the system, which involves functional as well as nonfunctional requirements. The requirements are the customer's expectations about what the system has to achieve. The requirements may put monetary and timing constraints on the design, which will have to be considered along with the technical specifications. The designers need to incorporate these requirements and realize a system, which can perform the expected tasks. The requirements have to be validated throughout the design process including final verification at the end product level.

The system specifications are more focused on system implementation. They offer the designer a role map for the design of the system. The specifications have to be written carefully to ensure that they meet requirements. The specifications must be comprehensible and unambiguous so that the designer knows exactly what has to be built. Ambiguous specifications can lead to an incorrect implementation, which can defeat the whole purpose of using a system approach.

The architecture describes how the functions are implemented in the system. The architecture defines the structure of the system. The architecture is the framework on which the technical aspects of the system will be implemented. The architecture will dictate what resources are used and how the subsystems are interfaced. The architecture design also needs to consider the constraints imposed by the skills of the designers. If the designers are not skilled in some particular technology used in the architecture, the cost of providing training to the designers has to be considered.

The components are designed with the architecture in mind. The components can be hardware components as well as software components. The specifications for the component are developed with the architecture specifications in mind. The components perform specific system tasks and together they perform the architecture tasks.

The integration of the system is where the benefits of using a system approach are particularly visible. If the specifications and architecture are designed correctly and tested, then this stage can be very simple. The whole system will be put together and the working system realized. The Systems Engineering concepts help to keep track of specifications for the separate modules and allow the modules to be tested separately, which simplifies the system integration process.

## 1.5 Embedded System Design

The embedded system consists of an embedded processor, hardware peripherals such as RAM, ROM, LCD, keypad, some communication channel for the processor to communicate with other devices such as RS232, Ethernet, USB, wireless transceiver and a software code which runs on the processor to control all the peripherals.  The various elements associated with the embedded system are presented in Figure 2.

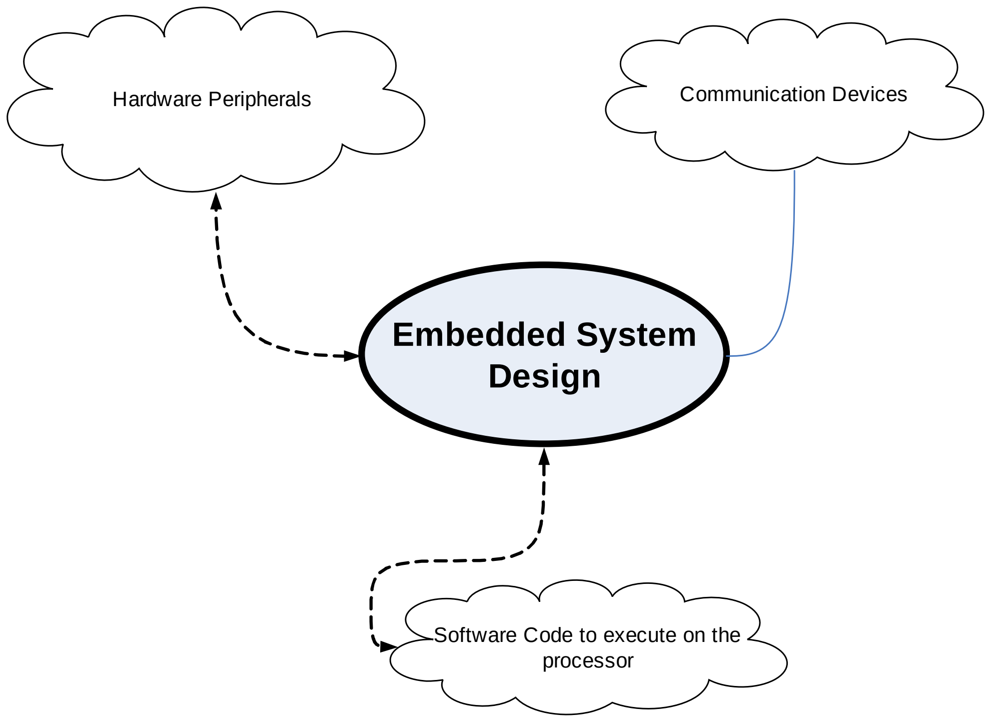

Figure 2:  Embedded System

Embedded system design involves division of system tasks between hardware and software.  Some functions are better implemented in hardware whereas some functions give better performance if implemented in software. Therefore, the designer has to carefully divide the system tasks between these two components in order to get optimum system performance and meet all system requirements. The selection of communication channels and the processor has to be accomplished in accordance with system requirements.

## 1.6 Programmable Logic

Programmable logic is hardware, which can be programmed to perform the required task using a Hardware Description Language (HDL). This programmability makes it a very good choice since it becomes easy to enhance the functionality or provide a patch for the system. The programmable logic makes it possible to accept last minute design changes, if necessary. The FPGA (Field Programmable Gate Arrays) provide an ideal platform of implementing digital logic circuits. With the advent of FPGA technology, the operating speed of these devices has increased dramatically. This makes it possible for the designer to infer high speed logic on these devices. The use of Intellectual Property (IP) cores can relieve the designer of some tasks and can significantly speed up the process. The FPGA devices from Xilinx Inc. and Altera Inc. were employed for this research.

## 1.7 Embedded Processors in Programmable Logic

The FPGA devices, which were used as the basic blocks for this research, can implement both hard core processors and the soft core processors. The Intellectual Property Core is provided by various vendors and that IP core can be used to instantiate a processor in the embedded system. The designer can also design a processor core, capable of executing the required instruction set, using the Hardware Description Languages.

The hard core processor is implemented on the FPGA chip. The advantage of using such a hard core processor is enhanced performance. Xilinx Inc. provides a hard core PowerPC core with some of its FPGA devices.

With a soft core processor, the user can modify the HDL code in order to achieve specific processor requirements. The soft core option provides the user more choices. The soft core processor offered by Altera Inc. is the NIOS II.

## 1.8 Overview of Development Kits Used

### 1.8.1 Altera DE2 Board

The DE2 (Development and Education 2) board, provided by Altera features the Cyclone II-2C35-FPGA in a 672 pins package. The DE2 board is pictured in Figure 3.

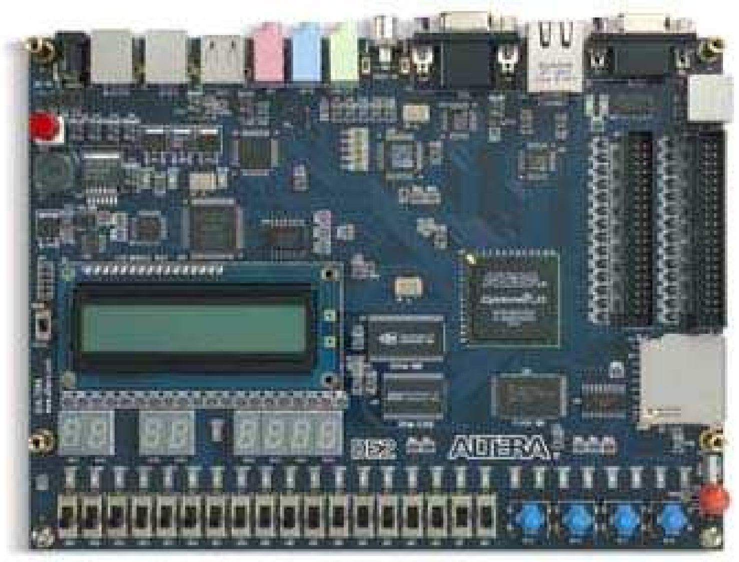

Figure 3: Altera DE2 Board
Courtesy: Altera Corporation Inc.

The DE2 board provides a ready to use development platform with many peripherals already connected to the FPGA pins. The NIOS II embedded processor on this FPGA can be used to develop embedded applications. The Quartus II web edition design software can be used to program this board. Altera also provides a NIOS II IDE to create embedded applications. This board also has SDRAM, SRAM and Flash memory, which can be used by the NIOS II core.

### 1.8.2 Xilinx XUP V2PRO Board

The Xilinx XUP V2PRO board features a Virtex-2 Pro XC2VP30 FPGA with 30,816 Logic Cells, 136 18-bit multipliers, 2,448Kb of block RAM, and two PowerPC Processors.

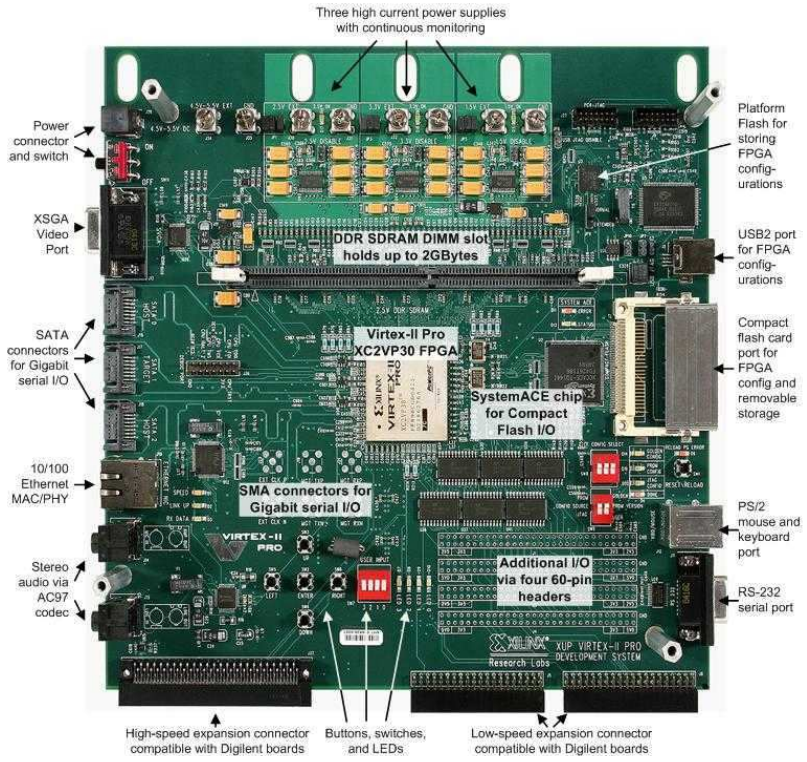
Three high current power supplies with continuous monitoring
Power connector and switch
XSGA Video Port
SATA connectors for Gigabit serial I/O
10/100 Ethernet MAC/PHY
Stereo audio via AC97 codec
DDR SDRAM DIMM slot holds up to 2GBytes
Virtex-II Pro XC2VP30 FPGA
SystemACE chip for Compact Flash I/O
SMA connectors for Gigabit serial I/O
Additional I/O via four 60-pin headers
Platform Flash for storing FPGA config-urations
USB2 port for FPGA config-urations
Compact flash card port for FPGA config and removable storage
PS/2 mouse and keyboard port
RS-232 serial port
High-speed expansion connector compatible with Digilent boards
Buttons, switches, and LEDs
Low-speed expansion connector compatible with Digilent boards

Figure 4: Xilinx XUP V2PRO Board
Courtesy: Xilinx Inc.

The Xilinx ISE foundation edition can be used to program this board. The Embedded Development Kit (EDK) must be used to develop embedded applications using the PowerPC cores. The microblaze soft core processor, provided by Xilinx, can also be used to develop embedded applications with this board.

## 1.9 Motivation

Use of embedded systems in all fields related to electrical design is rising exponentially. The design process is getting more and more complex with development of processors and peripherals. Therefore, to streamline the design process and to make it easy for keeping the designs portable and interoperable, the System Approach can prove useful. The thesis will demonstrate the application of the System Approach concepts to an example system design.

# CHAPTER 2:

# BACKGROUND

This Chapter provides a short description of previous work on Systems Engineering.  In addition, it describes how those concepts are applied to the design of embedded systems and how they were applied to the design of the example system.

## 2.1 Related Work

The design of complex systems is always a difficult task since the designers must be sure, while deciding on the specifications, that the design of the proposed system is feasible.  If it becomes impossible for some of the subsystems to follow the specifications then the specification set for the whole system of systems has to be changed.  While spelling out the specifications, it is very important for the designers to be very certain about what is expected of the subsystem designers.  As a guideline for this procedure, Dale Scot Caffall et al. proposed “a system-of-systems construct in which we consider a system-of-systems infrastructure that is composed of controlling software for the system-of systems, an information transport network, and contract interfaces” [1].  Dale Scot Caffall et al. also demonstrated application of formal methods to system of systems development for verification purpose in their publication [2].

The application of the system approach to management was studied by Group Captain W. W. Robinson in his publication ‘A system approach to management’ [3].  Dr.

John Boardman et al. presented distinguishing characteristics that can help in identification and implementation of System of Systems (SoS) [5].

The development of a Video Surveillance system using programmable logic design has been undertaken by some researchers since the programmable logic devices provide an optimum platform to develop a prototype system and implement different architectures on that platform. Suh Ho Lee et al. have implemented a motion detection system using a CMOS image sensor and ARM processor [4].

The concept of a Video Surveillance system involves more work than simple motion detection. The advanced Video Surveillance systems try to detect the object, localize the object and classify it so as to add more functionality to the Video Surveillance system. M.H. Sedky et al. proposed the classification of such smart Video Surveillance systems [6].

The use of FPGA devices in the design of surveillance systems is well documented. Fei Wang et al. implemented a FPGA based driver drowsiness detection system in order to detect if the driver has closed his eyes [7]. Patrick Dickinson et al. have designed an infant surveillance system in order to detect sleeping infants using a subtraction algorithm implemented on a Xilinx Virtex II PRO FPGA [8].

Jason Schlessman et al. implemented a tracking system based on optical flow using a Xilinx Virtex II PRO FPGA [9]. Michael McErlean proposed a hierarchical motion estimator for embedded object tracking using a FPGA [10].

Another motivation for this research work was the concepts put forward in the book “Power to the Edge”, which explains about empowering the nodes for better performance. This research work aimed to implement those concepts in the design of embedded systems. This research attempted to apply those concepts for the design of embedded system and one of the goals of this research was the development of intelligent nodes, which could work independently in case of system failure.

The book “Computers as Components” by Wayne Wolf explains the concepts related to embedded system design. The book has been very helpful in the analysis and planning of this research work [11].

## 2.2 Analysis of Requirements

During the study of any system, the requirements are the most important part in the design. The design of the system has to fulfill all the requirements. In order to streamline the development of subsystems the specifications of the subsystems have to be clearly defined. Any mistake in defining the specifications can lead to future problems.

Validation of these concepts was given the highest priority during the design of the Video Surveillance system. The system design has to fulfill the system requirements. Therefore, the planning stage is the most important stage in the design cycle since it considers all the available resources and then assigns specific tasks to subsystems by delineating, very specifically, the specifications. The technical capabilities and available resources were also considered during this stage. The specifications were formulated by considering all the possible solutions and then identifying the optimal solution for the current problem.

The architecture of the Video Surveillance system depends on the requirements. If the proposed system is to be used in some warehouse, then it needs to have some memory where the system can store some images. The cameras in shops need to have a larger memory so that they can store more data. In these types of systems it is more important to store the data rather than to process and detect some object. In some security applications it may be very important to detect the incoming object to enable quick response from the user. Thus, the architecture of the Video Surveillance system totally depends on the requirements. The requirements guide the design team in assigning priorities and level of effort to the various components and capabilities of the design such as deciding whether it is more important to spend more design effort on data compression for storage or data processing capability. Therefore, it is very important to write unambiguous specifications for the system before a team of engineers begins designing. A complete specification derived from the set of requirements is very important for the considerations involved in optimizing the design effort. In the event that specifications are not completely defined and are changed during the design cycle, the design effort can result in wasting valuable man hours and other resources.

The example system included in this thesis was designed for applications where detection and capture of motion was considered more important to the user than being able to look at the captured snapshot and take immediate necessary action. The system was designed for those applications where the user was not expecting any particular type of object so there was not any need to run an object detection algorithm. A diagram of a typical Video Surveillance system is presented in Figure 5.

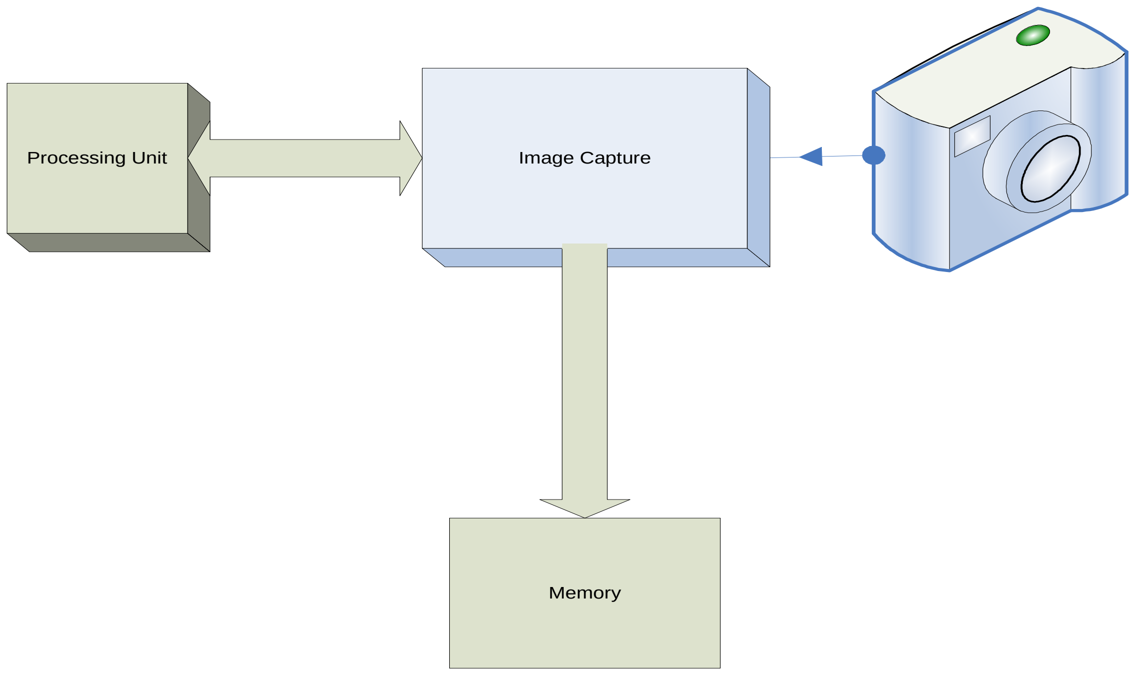

Figure 5: Typical Video Surveillance System

The criteria involved in the selection of various components of a Video Surveillance system are explained below. The working of the system will be explained later. The following paragraph explains the component selection process as a function of the requirements. The effort and resources allocated to designing the different components are decided by going over the system requirements. The digital camera represents any image sensor device. The device can be chosen to be a high speed device or a high resolution device. The primary criterion for selection of the image sensor device is the operating speed of the image sensor. The operating speed is specified in terms of frames per second. The resolution is also important and is specified by rows multiplied by columns. If the user needs to detect smaller objects, it may be necessary to select a sensor with more resolution. There is also another constraint on the image sensor selection, which derives from the operating frequency of the clock. For example, if an 8 bit parallel interface for a CMOS image sensor is selected and the operating frequency is

10 MHz, then the maximum data rate, which can be read by the system, will be 80 Mbps. A color image sensor may or may not be specified and is most often a function of the financial constraints.

The example system was designed with two sensors. A CMOS grayscale image sensor was chosen, which possessed a resolution of 126 X 98 and a speed of 580 frames per second. In addition, a CGA Red Green Blue sensor with a resolution of 640 X 480 and a speed of 60 frames per second was selected. The system was capable of employing the CMOS sensor if higher frames per second were specified or the VGA sensor if the requirements demanded better resolution.

The Image Capture block is responsible for setting up the sensor, extracting the frames and storing them in separate memories or arrays. The Image Capture block has to take input data from the sensor as per the requirements. This was specified because sometimes it may not be necessary to read the data at the maximum rate. If the system is only supposed to store the snapshots at regular intervals it can work reasonably well even with a data rate of one frame per second. These tasks are performed by the Image Capture block. The Image Capture block in the example system works at a clock rate of 50 MHz and reads data from the sensor only when required. The data read from the sensor is processed by the logic implemented on the Altera Cyclone II device. The system is not required to work as very high speed. Therefore, the Image Processing block reads a new frame only when the processing unit can accept another frame and a new frame is available from the sensor. The other frames are simply discarded.

The memory block has to be designed carefully since memories are costly in terms of operating time. Memories can slow down the whole system if not designed carefully.  The design of the memory block involved an estimate of the required memory. If the system was required to keep many past snapshots in the memory, this block would have larger capacity.  The system, which is used in some departmental stores, may need to have a lot of memory since it will need to keep a record of all the visitors for the stipulated time.

The memory block in the example system was very small since there was no need to keep track of past events.  Moreover, since the system transmits the results to another entity the memory block could also be used there, if required.  The memory used in the system was required to store only two frames.  One reference frame and an error frame. The working of the example system will be explained in Chapter 3.

The processing unit can vary in complexity from a simple comparator, in case of motion detector, to complex object detection or face recognition algorithm processors. The processing unit also decides the type of processor, which has to be used.  If the processing unit requirements are not computationally expensive, then a simple microcontroller can be used.  However, for algorithms such as face recognition, a very powerful processor could be required.  The processing unit can be designed in software as well as in the hardware.

If the system places more emphasis on record keeping, the processing unit will be the least worked on part of the system.  If the system is supposed to be used in an object detection smart surveillance system then the processing unit will be the most resource consuming part of the system.  In any circumstance the processing unit will decide the functionality of the system.

The processing unit in the example system was implemented in an Altera FPGA device. The processing unit consisted of comparators, arithmetic logic and buffer memory. The processing unit performs a pixel by pixel comparison over a period of 6 frames. Depending on the results of the comparison the processing unit detects the presence of another object. In case of detection of an unwanted object the snapshot, with the object in the frame, is transmitted over the wireless network to another entity where the frame can be further analyzed to localize, classify and detect the object.

The system thus consists of an Altera Cyclone II FPGA based development board, a CMOS image sensor, a VGA sensor and a wireless transmission channel. The system was intended to be a prototype for testing various object detection algorithms. The different modules were designed separately. Therefore, any module could be used in another system with small modification in order to meet requirements, if required.

The example system follows the philosophy of power to the edge [12]. This involves empowering the nodes of the system to attain better functionality. The idea is the algorithms can be implemented at the nodes of the system, which reduces the data that has to be transmitted over the network. This can help in reducing the network traffic and it can also be helpful in making the system more robust. Since the intelligence is distributed between various nodes, even is one node stops working the system can still function with the remaining nodes. This philosophy can greatly reduce system failures and improve performance. The only disadvantage may be that more processing units for use at nodes may be required, which will add up to the overall system cost.

## 2.3 Altera Cyclone II Device

The FPGA development board used for the design of the system was based on the Cyclone II FPGA from Altera. The FPGA device offers many features which can be used to implement advanced algorithms. The Cyclone II device was selected because it makes this system a very good platform for the implementation of prototypes. The Altera Cyclone II FPGA is pictured in Figure 6.

Figure 6: Altera Cyclone II FPGA

The Cyclone II device family provides from 4,608 to 68,416 logic elements (LE). The EP2C35F672C6 device contains 33,216 logic elements. The 105 M4K RAM blocks contain 483,840 total RAM bits, 35 embedded multipliers and 4 PLLs. This is sufficient for the current requirements since the simple algorithm employed used only 30% of these resources. This device provides a platform capable of implementing an advanced algorithm. This device provides the necessary resources to design a smart Video Surveillance system processing unit.

The Cyclone II device also comes with a soft core, NIOS II, embedded processor, which can be used for running software code. The processor provides the designer the freedom to choose whether the implementation will be hardware based or software based.

The Cyclone II device comprises the most important part of the Video Surveillance System.

## 2.4 Kodak KAC-9630 CMOS Image Sensor

The KAC-9630 provides a very high speed (580 frames per second) sensor. The disadvantage of this sensor is its low resolution. The resolution the sensor is 126 X 98 pixels, which does not allow for processing any details from the captured snapshot. The KAC-9630 sensor also possesses a parallel interface, which was easy to use for interfacing with the FPGA device. The KAC-9630 sensor can be very useful for the capture of high speed objects due to its very high frame rate.

The Video Surveillance System employs this sensor to implement simple pixel by pixel processing algorithms. The KAC-9630 sensor is present on the designed printed circuit board and will provide the designer with the option to select one of the two sensors in future research work.

## 2.5 The IrDA Interface

The Video Surveillance System used the IrDA interface to transmit the captured snapshot to another entity. The IrDA interface was implemented on the Altera DE2 board, which was connected to the Cyclone II device.

The IrDA system on the DE2 board was implemented using the HDSL 3201 transceiver. The interface is capable of data rates up to 115.2 Kbps. The HDSL 3201 was selected due to availability constraints.

The system checks the incoming frames for the presence of any unwanted objects. Since the system is not intended to look for any particular object, it will check for dissimilarities between consecutive frames.  If the system discovers that any frame differs significantly from the previous frame, the frame with dissimilarities will be transmitted over the IrDA to another entity where object detection algorithms are implemented.

## 2.6 Altera DE2 Development and Education Board

The Altera DE2 board, which is pictured in Figure 3 of Chapter 1, provides an excellent platform for a broad range of academic research, industrial research and prototyping applications.  The board is based on the Cyclone II device and comes with a USB blaster interface for download of a program into an FPGA.  The DE2 board also provides many other features such as an Ethernet 100/10 Mbps, RS232 and PS/2 interfaces.  In addition, the DE2 board contains an IrDA, which was used during this research.  The DE2 board also provides LED and DIP switches, which can be very helpful for testing small subset modules of the system.  The NIOS II processor provides an excellent soft core processor, which can be used to interface different peripherals and implement different algorithms.

The DE2 board also comes with on board SDRAM, SRAM, Flash and a SD card connector.  These features can be utilized if the designer runs out of the on chip memory resources on the Cyclone II device.  The DE2 board provides a very good platform for both the design and for the testing stage of the prototype.

## 2.7 Design Software and Support

The support material provided by Altera Inc. is very helpful in any project. The University program with Altera Inc. provides Intellectual Property (IP) cores for use with the DE2 board. Altera Inc. also provided, for download from their website, the Quartus II synthesis tool, required to program the FPGA, and the Modelsim software, required for simulation of the design. These resources along with the hardware make this platform suitable for use in research work.

# CHAPTER 3:

# IMPLEMENTATION

The Video Surveillance system consists of three different modules from the implementation point of view. The modules are named Image Capture, Image Processing and Image Transmission. The source code was written in VHDL and the Modelsim simulator was used for functional and timing simulation. The Quartus II web edition software was used as the synthesis tool. The VHDL code was tested with a VHDL test bench for functional verification.

## 3.1 Image Capture Module

The Image Capture module was implemented partly on the dedicated Printed Circuit Board and partly on the Cyclone II FPGA device. The function of this module is to set up the sensor and to provide the processing module with separate frames of data. This module also provides the sensor with the appropriate clock signal. The data is sent to the processing module with the necessary handshaking signals. Figure 7 presents a block diagram of the interface between the Image Capture module and the Image Processing module.

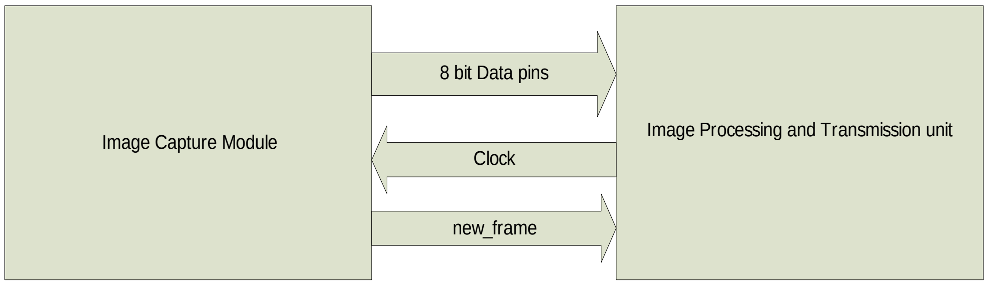

Figure 7: Proposed Interface

The Image Capture module provides the specified interface to the Image Processing block. The KAC-9630 is equipped with a parallel digital image data port. The details of the data port are presented in Figure 8.

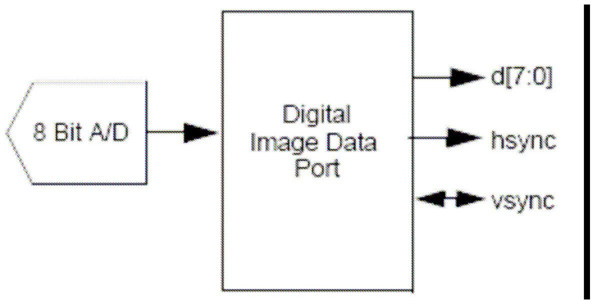

Figure 8: Image Data Port

The sensor output is presented at the input of an 8 bit A/D converter and the output of the A/D converter, along with the synchronization signals, is placed on the image data bus. This data is synchronized to the positive edge of the clock. The hsync ( horizontal synchronization) signal is used for synchronization of ROW data. The vsync (vertical synchronization) signal is used for synchronization of frame data in video mode and is also used as input in the snapshot mode. The hsync and vsync signals are used by all the other modules.

The function of the Image Capture block is to read the data from the sensor and to transmit it in required format to the Image Processing block. The clock, which is used by this block, is provided by the Image Processing block in order to maintain synchronization. This arrangement presents no problems as long as both blocks are implemented on the same device. If required, the Image Capture block can be made to work with different clocks.

The vsync signal is provided by the module as the new frame signal. The same clock was used for both Image Capture and Image Processing modules. The data bus at the output of the digital image bus is the output data port.

## 3.2 Image Processing Module

A block diagram of the Image Processing module is presented in Figure 9.

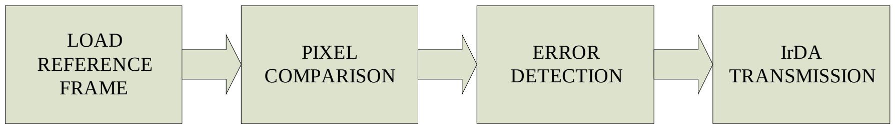

Figure 9: Image Processing Module

The Image Processing module only accepts data from the Image Capture module when it is ready for processing of the next frame. The function of the Image Processing module is to compare the incoming frame with the existing reference and check for errors. Whenever a determination of error occurs, the Image Processing module transmits the suspected frame over the wireless network. This block performs the processing on a pixel by pixel basis. A port map, of the input and output signals for the Image Processing module, is presented in Figure 10.

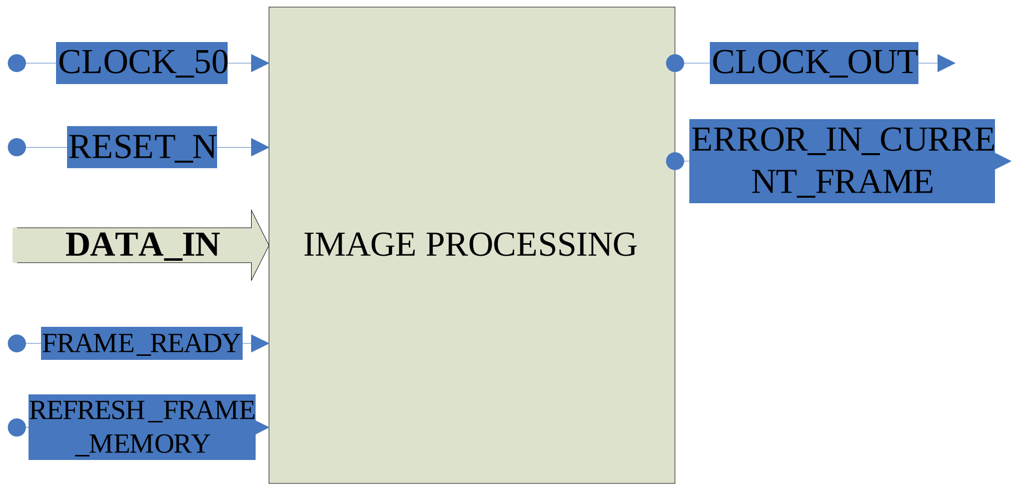

Figure 10: Port Map for the Image Processing Module

### 3.2.1 System Initialization

The Image Processing module has two memories, one for storing the frame under analysis and the other for store of the reference frame. Initially, the system loads the first incoming frame in the reference frame memory so that further analysis can be performed. The reference memory can be refreshed anytime by asserting a load reference memory signal.

The system uses two counters for keeping track of rows and columns. The data coming in from the Image Capture module is stored in the reference frame memory. The reference frame memory was instantiated using the memory bits available in the Cyclone II device. Once the reference frame is loaded an internal signal is asserted, which paves the way for further processing of the incoming frames.

### 3.2.2 Pixel Comparison

Once the reference frame is loaded, in the reference memory, future data can be compared to the data in the reference frame. The row counters and column counters were used to keep the track of current frame. Once the system receives a byte of data on its input lines, the corresponding pixel value in the reference frame memory is fetched by providing a correct address to the reference frame memory address bus and the two pixels are compared. The comparison is not bit by bit. The required sensitivity can be defined by the designer. If the incoming pixel value is found to be out of the tolerance range of the system, an error counter is incremented. The error counter keeps track of the number of pixels in every frame, which are out of the tolerance zone.

### 3.2.3 Error Detection

The error detection counter was used to decide the threshold value. If the number of pixels, which are different than the previous frame, is more than the threshold value the system flags the particular frame as a suspected frame and transmits it for more processing. If the threshold value is set to a very small value the system may flag some frames as suspected due to noise and changes in illumination. Therefore, deciding the threshold is very important for correct operation. The threshold value for error detection decides the minimum object size that can be detected.

If a frame is marked as suspected, that frame is taken out of frame memory and sent over the IrDA channel. If a frame is not suspected, then the frame memory is overwritten. Since the frame memory has to be accessed frequently it was implemented using the on chip resources. The frame memory is written to during almost every clock cycle and the reference frame memory is read during every clock cycle. Therefore, the memory access times play a major role in deciding the combinational delay of the module.

It has to be noted that a smaller object may appear bigger in frame if it is close to the camera. Therefore, the minimum size is the size which is detected anywhere in the range of the Video Surveillance system. The smaller objects may be detected if they are too close to the camera and the larger objects may go unnoticed if they are far away from the camera. Thus, while providing the specifications for the Video Surveillance system it is very important to know the altitude of the camera and the area which is required to be covered. Furthermore, an object may go unnoticed if it is not captured in any frame. Therefore, the frame rate places an upper limit on the speed of motion of the object being detected. This speed can not be specified without considering the operating condition. An object moving near the camera will appear to be moving at a higher speed than an object moving at the same speed but further away from the camera.

The threshold value was kept programmable so that any changes which may be required during operation may be performed. The system will perform better if it is fine tuned by considering the operating conditions.

The sensitivity can be increased by changing the comparison operation in the processing unit. If a bit by bit comparison is performed, the system will be more sensitive. However, bit by bit comparison may generate many more false alarms. Therefore, the comparison operation and the error detection threshold have to be decided by careful review of the requirements. In addition, the nature of the objects to be detected and the environment where the system is going to be used must be considered carefully and their impact on the requirements taken into account. If the system is supposed to be used inside a warehouse then the changes due to the daylight will be negligible since the system will always work under artificial light. However, a system which is to be deployed in an outdoor environment will have, as a primary consideration, factors associated with the changes in illumination.

In order to be able to adapt to changing conditions, the refresh frame memory signal was provided as an input port. The reference frame memory can be refreshed every few minutes when the system is in an outdoor environment in order to take care of the illumination issues. The refresh frame memory signal has to be efficiently controlled to obtain optimum performance of the system under different kinds of requirements.

## 3.3 IrDA Transmission

If any frame is found to be suspect it is sent over the IrDA channel. The IrDA channel has adata rate of 115 Kbps. The IrDA transmission module is activated by the error signal from the Image Processing module. The IrDA reads the frame memory and transmits the data in the event of an error.

# CHAPTER 4:

# TESTING AND VERIFICATION

The functional verification was the most important stage in the design flow since this stage is where logical errors can be identified and rectified. The functional verification involved testing the design to see if it operated the way it was supposed to operate. The verification of the VHDL code was performed by a VHDL test bench.

## 4.1 FPGA Verification

A block diagram of the design flow for testing during FPGA verification is presented in Figure 11.

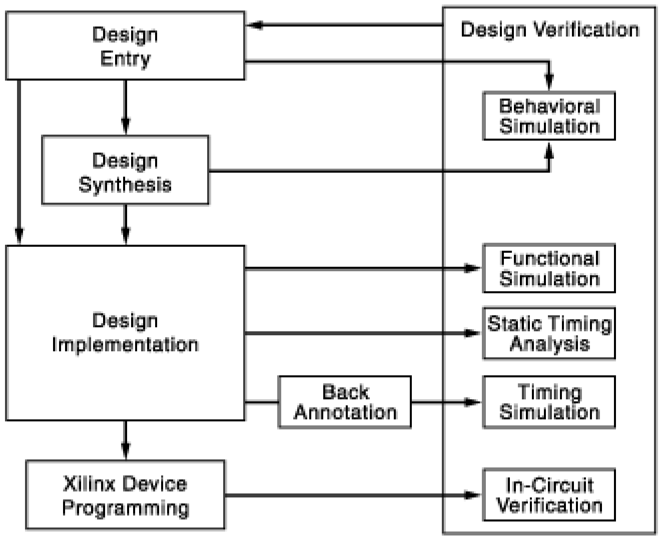

Figure 11: FPGA Design Flow
Courtesy: Xilinx Inc.

The design entry stage included generating the VHDL code for the design. The behavioral simulation did not consider the combinational delays and the routing delays.

The behavioral simulation was intended only for detecting logical inaccuracies. Once the design was cleared by the behavioral simulation it was synthesized by the synthesis tool. The synthesis tool translated the HDL (Hardware Description Language), description to a logic circuit. The synthesis tools used the logic components from its component library. Therefore, once the synthesis process was complete the design had to be simulated again to check whether it worked at the desired frequency.

After successful synthesis and behavioral simulation the design moved to the implementation stage. During implementation the combinational delays, associated with the logic components, were considered and the design was, once again, simulated to check for any timing violations. If the design failed to follow the timing constraints it could be fixed, if required, by editing the VHDL code. During this stage the synthesis tool can also assume the routing delay in order to provide a designer a rough estimate of the delay, which will be present when the device is actually downloaded to the FPGA board.

Next, the EDA tool routes the design on the target FPGA device and provides the designer with fairly exact numbers with respect to routing delays and combinational delays. At this stage the timing simulation is performed and if the design is cleared by this simulation it can be downloaded to the FPGA.

The EDA tools for synthesis and simulation provide the user with very accurate delay estimates and functionality checks, which avoids future problems related to timing violations and design failures. The design was tested once again, with the aid of a Logic Analyzer, after it was downloaded to the FPGA board.

In the design flow, verification is the most important block and at times it takes more resources than the design team. The verification is extremely important for large scale orders since it may prove to be very costly to fix any bugs in the design after the solutions are shipped.

During this research, all verification was performed by the Modelsim tool and logic analyzer. The VHDL test bench was used to perform behavioral simulation, functional simulation and timing simulation. Once the design passed these tests it was ready to be downloaded to the FPGA board.

## 4.2 VHDL Test Bench

The VHDL test bench concept is presented figuratively in Figure 12.

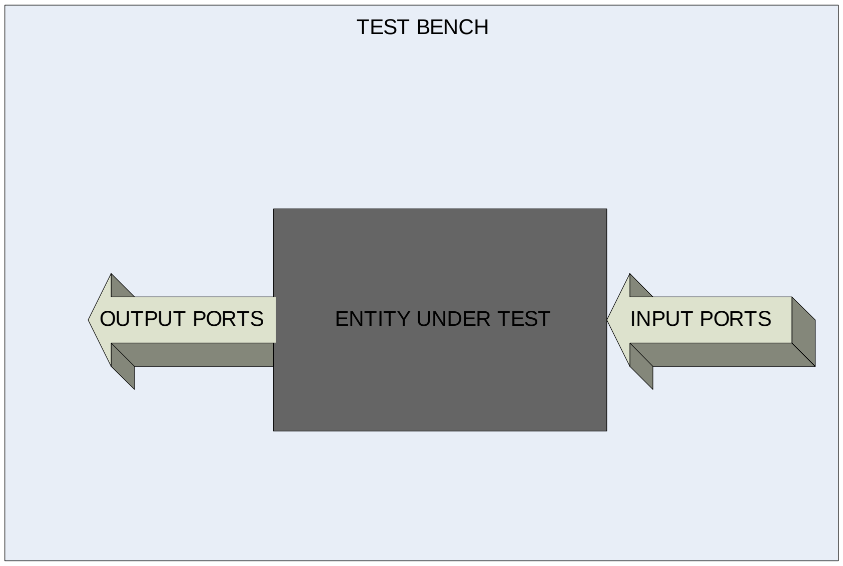

Figure 12: VHDL Test Bench

A VHDL test bench is VHDL code, which drives the input ports with the simulated input signals and checks the output ports for expected data. This procedure can assure the designer of correct functionality so the design can be taken to next stage.

Simulator software can be used for functional verification. For this research, the Modelsim simulator from Mentor Graphics Inc. was employed.

The VHDL test bench simulated the input signals and checked for the output signals to determine functionality of the system. During this research, separate test benches were developed for different modules. The intention was to ensure correct functionality of every module before assembly of the whole system.

To verify the Image Capture block, the test bench simulated the Image Processing block. The VHDL test bench code was used to drive the input signals such as clock and check for the new frame and data signals. The frames received from the Image Capture block were saved in raw format and checked. Since the generation of the image depended on the sensor, this module had be tested in hardware. It was not possible to complete the functional verification of this module without the actual hardware board.

The test bench for the Image Processing block simulated the signals from the Image Capture module and Image Transmission module. Two files on the computer hard drive were read by the test bench as input data and the output observed. The test bench, for the Image Processing block, involved the use of the VHDL package, TextIO. Different image files on the computer hard drive were read and provided as inputs to the image processing block. The objective was to check whether the module could identify the differences in two image frames.

The advantage of writing a test bench is that the same test bench can be used for all simulations. The test benches developed during this research were all used from the behavioral simulation stage to the timing simulation stage.

The final system could not be completely tested by a VHDL test bench.

Therefore, for testing of the final system, a hardware testing platform was employed.

# CHAPTER 5:

# CONCLUSION AND FUTURE WORK

## 5.1 Conclusion

The concepts of the system approach were applied to the design of a Video Surveillance system and it yielded results as per the system requirements. Separate modules were designed to implement the functionality of the system. As a result of the application of Systems Engineering concepts, the system is easy to modify and debug. It is very easy to use some module of this system as a subsystem in another design.

The overall system requirements were studied and separate module system specifications were written. The various requirements were divided into separate modules. Due to the modular approach a designer can target specific characteristics of the system for improvement without modifying the complete system. For example, if it is desired and/or required that the resolution of the system be improved, only the Image Capture block would need to be modified. The changes which would be required in the Image Processing block are simple, since generics were used in the VHDL code. Hardware changes would not occur since the same FPGA device can be used to implement any new functionality.

Any effort to increase the intelligence of this system will be directed at the Image Processing block. Since programmable logic was used, further revisions simply involve reprogramming.

The system can be interfaced to any wireless transmission medium by simply changing the transmission module.

## 5.2 Future Work

Some improvements could be performed in the future to enhance the functionality of the system:

- The Image Capture module could be implemented completely on one PCB board. The Printed Circuit Board has a MSP240 processor, which could be used for setting up the sensor and providing the output rather than using the Cyclone II device.
- The Image Processing block could be modified to reduce the number of false alarms. The system could be made immune to changes in illumination. Rather than the current scheme of pixel by pixel processing, block processing could be implemented, which would help to localize the object and could reduce the amount of data sent over the network.
- An Ethernet interface could be provided for transmission.

## REFERENCES

1. Caffall, D.S. and Michael, J.B., “Formal Methods in a System-of-Systems Development”, Systems: Man and Cybernetics, 2005 IEEE International Conference, Volume 2, Page(s): 1856-1863, 10-12 Oct. 2005

2. Caffall, D.S. and Michael, J.B., “Architectural Framework for a System-of-Systems”, Systems: Man and Cybernetics, 2005 IEEE International Conference, Volume 2, Page(s): 1876-1881, 10-12 Oct. 2005

3. Robinson, W.W., “A Systems Approach to Management”, Engineering Management Journal, Volume 6, Issue 4, Page(s):172-176, Aug. 1996

4. Eisner, H., “A Systems Engineering Approach to Architecting a Unified System of Systems”, Systems: Man and Cybernetics, 1994, “Humans, Information and Technology”, 1994 IEEE International Conference, Volume 1, Page(s): 204-208, Oct. 1994

5. Boardman, J. and Sauser, B., “System of Systems - the Meaning of System of Systems Engineering”, 2006 IEEE/SMC International Conference, Page(s): 24-26, 6 April 2006

6. Sedky, M.H., Moniri, M. and Chibelushi, C.C., “Classification of Smart Video Surveillance Systems for Commercial Applications Advanced Video and Signal Based Surveillance”, 2005 IEEE Conference on AVSS, Page(s): 638-643, 15-16 Sept. 2005

7. Fei Wang and Huabiao Qin, “A FPGA Based Driver Drowsiness Detecting System”, Vehicular Electronics and Safety, 2005. IEEE International Conference, Page(s): 358-363, 14-16 Oct. 2005

8. Dickinson, P., Appiah, K., Hunter, A. and Ormston, S., “An FPGA-Based Infant Monitoring System”, 2005 IEEE International Conference on Field-Programmable Technology, Page(s): 315-316, 11-14 Dec. 2005

9. Schlessman, J., Cheng-Yao Chen, Wolf, W., Ozer, B., Fujino, K. and Itoh, K., “Hardware/Software Co-Design of an FPGA-Based Embedded Tracking System” 2006 Conference on Computer Vision and Pattern Recognition Workshop, Page(s):123-123, 17-22 June 2006

10. McErlean, M., “Hierarchical Motion Estimation for Embedded Object Tracking”, 2006 IEEE International Symposium on Signal Processing and Information Technology, Page(s): 797-802, Aug. 2006

11. Wolf, W.H., “Computers as Components”, Principles of Embedded Computing System Design, Morgan Kaufmann Publishers, San Francisco, CA, 2001

12. David S. Alberts and Richard E. Hayes, “Power to the Edge: Command and Control in the Information Age”, CCRP Publications, 2003

## APPENDICES

## Appendix A: VHDL Source Code

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY object_detect IS
  GENERIC (
    ROWS: INTEGER:=98;
    COLUMNS: INTEGER:=126;
    VECTOR_LENGTH: INTEGER := 13);
                        -- the generic to pass the number of pixels to be read
  PORT (
    clock_50                : IN  std_logic;  -- system clock from on board source
    reset_n                 : IN  std_logic;  -- system reset signal
    clock_out               : OUT std_logic;
                        -- output clock to read data from the frame capture block
    data_in                 : IN  std_logic_vector(7 DOWNTO 0);
                              -- data from the frame read
    frame_ready             : IN  std_logic;
                        -- the frame ready signal from the frame capture block
    refresh_frame_mem       : IN std_logic;
    error_in_current_frame  : OUT std_logic;
 DUMMY_ref_frame_loaded : OUT std_logic;  -- DUMMY signal for testing
    data_read_out1          : OUT std_logic_vector(7 DOWNTO 0);
    new_frame_analysis1     : OUT std_logic;  -- DUMMY signal for testing
    ------------------------------------------------------------------------
    address_frame_mem       : IN std_logic_vector(VECTOR_LENGTH DOWNTO 0);
    data_frame_mem          : OUT std_logic_vector(7 DOWNTO 0));

                      -- output signal showing presence of some unwanted objects
                      -- in the image

END ENTITY object_detect;
```

```
ARCHITECTURE object_detect_1 OF object_detect IS
  SIGNAL clock_enable, start_clock_enable, clock_out1, sig1 : std_logic;
                                  -- the clock enable signal for the output clock going to the frame
capture block
  SIGNAL clock_enable_counter : std_logic_vector(VECTOR_LENGTH DOWNTO 0);
                                  -- counter to ensure complete frame is read in
  SIGNAL rdaddress,wraddress : std_logic_vector(VECTOR_LENGTH DOWNTO 0);
                                  -- the address to drive address bus of referance frame
  SIGNAL ref_frame_loaded, load_new_frame, new_frame_analysis : std_logic;
                                  -- signals for loading a new frame and to specify that new frame
is loaded
  SIGNAL data_read_out,write_databus_refmem : std_logic_vector(7 DOWNTO 0);
                                  -- the read and write databusses for the DPRAM
  SIGNAL data_in1, data_in2, data_in3 : std_logic_vector(7 DOWNTO 0);
                                  -- the registered data_in to compensate for the registered output
from the DPMEM
  SIGNAL error_count : std_logic_vector(VECTOR_LENGTH DOWNTO 0);
                                  -- the counter to count how many pixels are in error
BEGIN  -- object_detect_1

  -----------------------------------------------------------------------------
  -- DUMMY signals for testing
  -----------------------------------------------------------------------------
-----------------------------------------------------------------------------
DUMMY_ref_frame_loaded <= ref_frame_loaded;
data_read_out1 <= data_read_out;
new_frame_analysis1 <= new_frame_analysis;
-- -----------------------------------------------------------------------------
-- The clock enable control and reference frame loading processes
-----------------------------------------------------------------------------
            -- purpose: this process controls the clock, which is going as output to the frame
            --capture block

  clock_enable_process: PROCESS (clock_50, reset_n) IS
  BEGIN  -- process clock_enable_process
   IF reset_n = '0' THEN                -- asynchronous reset (active low)
      clock_enable <= '0';
      clock_enable_counter <= (OTHERS => '0');
   ELSIF clock_50'event and clock_50 = '1' THEN  -- rising clock edge
    IF frame_ready = '1' THEN
    clock_enable_counter <=
            conv_std_logic_vector((ROWS*COLUMNS),VECTOR_LENGTH+1);
        clock_enable <= '1';
      END IF;
      IF clock_enable_counter/=conv_std_logic_vector(0, VECTOR_LENGTH+1) THEN
        clock_enable_counter <= clock_enable_counter - '1';
      ELSE
        clock_enable <= '0';
      END IF;
    END IF;
  END PROCESS clock_enable_process;

  clock_out1 <= clock_50 WHEN clock_enable = '1' else
                        '0';
  clock_out <= clock_out1;

-- purpose: this process accepts data coming in from the frame capture block

  accept_frame: PROCESS (clock_50, reset_n) IS
  BEGIN  -- process accept_frame
    IF reset_n = '0' THEN               -- asynchronous reset (active low)
      load_new_frame <= '0';
      wraddress <= (OTHERS => '0');
      ref_frame_loaded <= '0';
    ELSIF clock_50'event and clock_50 = '1' THEN  -- rising clock edge
      IF (refresh_frame_mem AND frame_ready) = '1'THEN
      load_new_frame <= '1';
      wraddress <= conv_std_logic_vector((ROWS*COLUMNS),VECTOR_LENGTH+1);
      ref_frame_loaded <= '0';
      END IF;
      IF load_new_frame = '1' THEN
        IF wraddress /= "0000000000000" THEN
        wraddress <= wraddress - '1';
        write_databus_refmem <= data_in;
        ELSE
         ref_frame_loaded <= '1';
         load_new_frame <= '0';
        END IF;
      END IF;
    END IF;
  END PROCESS accept_frame;
  ---------------------------------------------------------------------------
  -- -------------------------------------------------------------------------
  -- Component instantiation for the DP ref memory
  ---------------------------------------------------------------------------
  ---------------------------------------------------------------------------

  Reference_memory_DPRAM : ENTITY work.ref_frame PORT MAP (
    clock     => clock_50,
    data      => write_databus_refmem,
    rdaddress => rdaddress,
    wraddress => wraddress,
    wren      => load_new_frame,
    q         => data_read_out);
  sig1 <= new_frame_analysis OR load_new_frame;
  current_frame_mem_DPRAM : ENTITY work.ref_frame PORT MAP (
    clock     => clock_50,
    data      => data_in,
    rdaddress => address_frame_mem,
    wraddress => rdaddress,
    wren      => sig1,
    q         => data_frame_mem);

  -----------------------------------------------------------------------
  -- ---------------------------------------------------------------------
  -- The processing after loading the ref frame
  -----------------------------------------------------------------------
  -----------------------------------------------------------------------

-- purpose: this process takes the input bit stream and compares it with data inside the
reference memory block

  processing_data: PROCESS (clock_50, reset_n) IS
  BEGIN  -- process processing_data
    IF reset_n = '0' THEN               -- asynchronous reset (active low)
      rdaddress <= (OTHERS => '0');
      error_count <= (OTHERS =>'0');
    ELSIF clock_50'event AND clock_50 = '1' THEN  -- rising clock edge
      IF ref_frame_loaded = '1' THEN
        IF frame_ready = '1' THEN
          rdaddress <=  conv_std_logic_vector((ROWS*COLUMNS), VECTOR_LENGTH
+ 1);
          new_frame_analysis <= '1';
        END IF;
        IF rdaddress /= conv_std_logic_vector(0, VECTOR_LENGTH + 1) THEN
```

```
          rdaddress <= rdaddress - '1';
          data_in1 <= data_in;
          data_in2 <= data_in1;
          data_in3 <= data_in2;
          IF data_in3(7 DOWNTO 5) /= data_read_out(7 DOWNTO 5) THEN
            error_count <= error_count + '1';
          END IF;
        ELSE
          new_frame_analysis <= '0';
          IF error_count > conv_std_logic_vector(200, VECTOR_LENGTH + 1) THEN
            error_in_current_frame <= '1';
          ELSE
            error_in_current_frame <= '0';
          END IF;
          error_count <= (OTHERS => '0');
        END IF;
      END IF;
    END IF;
  END PROCESS processing_data;
END ARCHITECTURE object_detect_1;
```

## Appendix B: VHDL Test Bench

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE STD.TEXTIO.ALL;
USE IEEE.ALL;
ENTITY object_detect_tst_img IS
END ENTITY object_detect_tst_img;

ARCHITECTURE object_detect_tst_a OF object_detect_tst_img IS
  SIGNAL clock_50, reset_n, clock_out, frame_ready, refresh_frame_mem,
  SIGNAL error_in_current_frame : STD_LOGIC :='0';
                         -- signals for the ports
  SIGNAL data_in : STD_LOGIC_VECTOR(7 DOWNTO 0);
  SIGNAL DUMMY_ref_frame_loaded : STD_LOGIC;
                         -- Signal taken out of entity for testing
  SIGNAL done : STD_LOGIC := '0';  -- flag set when simulation finished
  CONSTANT number_of_pixel_values : STD_LOGIC_VECTOR(13 DOWNTO 0) :=
                                             "00000000001111";
                         -- generic value
  SIGNAL address_frame_mem : STD_LOGIC_VECTOR(13 DOWNTO 0) :=
                                             "00000000000000";
  SIGNAL data_frame_mem : STD_LOGIC_VECTOR(7 DOWNTO 0);
  CONSTANT N : INTEGER := 12347;  --Number of bytes in file minus one
  SUBTYPE file_element IS STD_LOGIC_VECTOR(7 DOWNTO 0);
  TYPE mem_array IS ARRAY(N DOWNTO 0) OF file_element;
```

```
 COMPONENT object_detect IS
   GENERIC (
     ROWS : INTEGER := 98;
     COLUMNS : INTEGER := 126;
     VECTOR_LENGTH : INTEGER := 13);
   PORT (
   clock_50 : IN  STD_LOGIC; -- system clock from on board source
   reset_ : IN  STD_LOGIC;  -- system reset signal
   clock_out : OUT STD_LOGIC;
                              -- output clock to read data from the frame capture block
   data_in : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);  -- data from the frame read
   frame_ready           : IN STD_LOGIC;
                              -- the frame ready signal from the frame capture block
   refresh_frame_mem : IN STD_LOGIC;
   error_in_current_frame : OUT STD_LOGIC;
   DUMMY_ref_frame_loaded : OUT STD_LOGIC;
   data_read_out1        : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
   ------------------------------------------------------------------------
   address_frame_mem : IN STD_LOGIC_VECTOR(VECTOR_LENGTH DOWNTO
0);
   data_frame_mem : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
   new_frame_analysis1 : OUT STD_LOGIC);

                           --output signal showing presence of some unwanted
                           --object in the image

 END COMPONENT;
 SIGNAL tmp_data : STD_LOGIC_VECTOR(7 DOWNTO 0); -- temporary data
 SIGNAL new_frame_analysis1 : STD_LOGIC;
BEGIN  -- object_detect_tst_a

 U1 : COMPONENT object_detect PORT MAP (
             clock_50 => clock_50, reset_n => reset_n,
             clock_out => clock_out, frame_ready  => frame_ready,
             refresh_frame_mem => refresh_frame_mem,
             error_in_current_frame => error_in_current_frame,
             data_in => data_in,
             DUMMY_ref_frame_loaded => DUMMY_ref_frame_loaded,
             address_frame_mem      => address_frame_mem,
             data_frame_mem         => data_frame_mem,
             new_frame_analysis1    => new_frame_analysis1);
   clock_50 <= NOT clock_50 AFTER 10 ns;
 Reset_control: PROCESS IS
 BEGIN  -- process Test
   reset_n <= '0';
   WAIT FOR 150 ns;
   reset_n <= '1';
   WAIT;
 END PROCESS Reset_control;

 load_new_frame : PROCESS IS
 BEGIN  -- process load_new_frame
   WAIT UNTIL reset_n = '1';
   frame_ready <= '1';
   refresh_frame_mem <= '1';
   WAIT UNTIL (clock_50'event AND clock_50 = '1');
   frame_ready <= '0';
   refresh_frame_mem <= '0';
   data_in <= "10101010";
   WAIT UNTIL DUMMY_ref_frame_loaded = '1';
   WAIT FOR 300 ns;
   frame_ready <= '1';
   WAIT UNTIL (clock_50'event AND clock_50 = '1');
   frame_ready <= '0';
    data_in <= "01010101";
   WAIT;
 END PROCESS load_new_frame;

 read_file : PROCESS IS  -- read file_io.in (one time at start of simulation)
     SUBTYPE INTEGER_8bit IS INTEGER range 0 TO 255;
     TYPE char_type IS FILE of INTEGER;  -- file type for declaring the file
     VARIABLE ch : INTEGER;
     VARIABLE tmp1, tmp2, tmp3, tmp4 : INTEGER;  -- temporary to extract byets
                                  -- the character being read from the file
     FILE my_input : char_type OPEN READ_MODE IS "pirates.raw";
     FILE my_check : char_type OPEN READ_MODE IS "pirates.raw";
     FILE my_check2 : char_type OPEN WRITE_MODE IS "output.raw";
 BEGIN
     WAIT UNTIL reset_n = '1';
     WAIT UNTIL refresh_frame_mem = '1';
     LOOP
       EXIT WHEN (endfile(my_input) OR DUMMY_ref_frame_loaded = '1');
       read(my_input, ch);
       tmp4:= ch MOD 256;
       WAIT UNTIL clock_50 = '1' AND clock_50'event;
       WAIT FOR 5 ns;
```

```
        data_in <= CONV_STD_LOGIC_VECTOR (tmp4, 8);
        ch := ch / 256;
        tmp3:= ch MOD 256;
        WAIT UNTIL clock_50 = '1' AND clock_50'event;
        WAIT FOR 5 ns;
        data_in <= CONV_STD_LOGIC_VECTOR (tmp3, 8);
        ch := ch/256;
        tmp2:= ch MOD 256;
        WAIT UNTIL clock_50 = '1' AND clock_50'event;
        WAIT FOR 5 ns;
        data_in <= CONV_STD_LOGIC_VECTOR (tmp2, 8);
        ch := ch/256;
        tmp1:= ch;
        WAIT UNTIL (clock_out'event AND clock_out = '1');
        WAIT FOR 5 ns;
        data_in <= CONV_STD_LOGIC_VECTOR (ch, 8); --to_stdlogicvector(ch_vector);
    END LOOP;
      REPORT "out of the ref frame loading loop" SEVERITY note;
      WAIT FOR 250 ns;
      WAIT UNTIL frame_ready = '1';
      LOOP
        EXIT WHEN endfile (my_check);
        Read (my_check, ch);
        tmp4:= ch MOD 256;
        WAIT UNTIL clock_50 = '1' AND clock_50'event;
        WAIT FOR 5 ns;
        data_in <= CONV_STD_LOGIC_VECTOR (tmp4, 8);
        ch := ch / 256;
        tmp3:= ch MOD 256;
        WAIT UNTIL clock_50 = '1' AND clock_50'event;
        WAIT FOR 5 ns;
        data_in <= CONV_STD_LOGIC_VECTOR (tmp3, 8);
        ch := ch/256;
        tmp2:= ch MOD 256;
        WAIT UNTIL clock_50 = '1' AND clock_50'event;
        WAIT FOR 5 ns;
        data_in <= CONV_STD_LOGIC_VECTOR (tmp2, 8);
        ch := ch/256;
        tmp1:= ch;
        WAIT UNTIL (clock_out'event AND clock_out = '1');
        WAIT FOR 5 ns;
        data_in <= CONV_STD_LOGIC_VECTOR (ch, 8); --to_stdlogicvector (ch_vector);
      END LOOP;
      REPORT "Simulation complete!!!!!!!" SEVERITY note;
      ----------------------------------------------------------------------
      -- reading from the cuddern frame memory
      ----------------------------------------------------------------------
      WAIT UNTIL new_frame_analysis1 = '0';
      address_frame_mem <= CONV_STD_LOGIC_VECTOR (12347,14);
      WAIT FOR 2 ns;
      WHILE address_frame_mem > "00000000000000" LOOP
        address_frame_mem <= address_frame_mem - '1';
        WAIT UNTIL clock_50'event AND clock_50 = '1';
        ch := CONV_INTEGER (data_frame_mem);
        write (my_check2, ch);
      END LOOP;
      REPORT "reading the current memory complete" SEVERITY note;
      WAIT;
    END PROCESS read_file;
  END ARCHITECTURE object_detect_tst_a;
```