Development of an FPGA Based Image Processing Intellectual Property Core

Sreejith M

Department Of Electrical Engineering

NIT ROURKELA

Rourkela, India

May 2014

Development of an FPGA Based Image Processing Intellectual Property Core

A Thesis submitted in partial fulfillment

of the requirements for the degree of

Master of Technology

in

Department Of Electrical Engineering

by

Sreejith M

ROLL NO: 212EE1201

Under Guidance of

Dr. (Prof). Supratim Gupta

Department Of Electrical Engineering

NIT ROURKELA

May 2014

Declaration of Authorship

I, Sreejith M, declare that this thesis titled, “Development of an FPGA Based Image Processing Intellectual Property Core" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________________

Date: ______________________________

NIT ROURKELA

CERTIFICATE

This is to certify that the thesis entitled, **“Development of an FPGA Based Image Processing Intellectual Property Core"** submitted by **Sreejith M** in partial fulfillment of the requirements for the award of Master of Technology Degree in **Electrical Engineering** with specialization in **Electronics System and Communication** during 2013-2014 at the National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other university / Institute for the award of any Degree or Diploma.

Date: . . . .

Dr.(Prof). Supratim Gupta
Dept. of Electrical Engineering
National Institute of Technology
Rourkela-769008
Odisha, India

Abstract

Traditional image processing algorithms are sequential in nature. When these algorithms are implemented in a real-time system, the response time will be high. In an embedded platform, such algorithms consumes more power because of more number of clock cycles required to execute the algorithm. With the advent of Field Programmable Gate Arrays (FPGA), massively parallel architectures can be developed to accelerate the execution speed of several image processing algorithms. In this work , such a parallel architecture is proposed to accelerate the SOBEL edge detection algorithm. The architecture is simulated in Modelsim 10.2C student edition platform. To simulate this architecture, a model of video acquisition system is developed. This model will convert the incoming frames to digital composite video signals which can be processed by the edge detection architecture. An external software developed in Matlab will convert the frames in to hexadecimal format, and will feed the video acquisition model. The output of the edge detection processor will be a digital composite signal. A display module will convert the digital composite video signals in to hexadecimal format. Then with the help of an external Matlab program, the original image will be reconstructed. The result shown compares the sequential and parallel environments, and shows significant improvements in FPGA based implementations. The Modelsim simulation of SOBEL based edge detection algorithm for a 256×256 frame, gave a result in 0.019 seconds for a clock speed of 10MHz, where as a Matlab based simulation took 0.22 seconds to finish this operation, which is a significant acceleration.

Moreover, a new software simulation platform was developed as a part of this project, which will let the developer to give input as image and the output will be reproduced in the same format, while all the background processing will be carried out in VHDL. The simulation results are shown in this simulation platform. This software can be used as a simulation platform for any FPGA based image processing operations.

This work has wider scope and applications. FPGA based edge detection system can serve as the basic step for implementations of complex computer vision algorithms. The implementation of fast face detection in a digital camera, fast object tracking, policing and interactive surveillance, and finally the applications like object tracking and chasing are some of the areas where this work can be made use of.

Acknowledgements

When I was about to start this work, my guide Dr.Supratim Gupta told: "There are three realms of work in this world: creation, maintenance, and destruction. There are very few people in the first category, and those belonging here can revolutionize the world". Starting with these words, he supported and motivated me through out my work. From the bottom of my heart, I would like to express my gratitude to Dr.Supratim Gupta for all the support and inspiration.

When I lost the final ray of hope, when I was frustrated seeing nothing other than some junk straight line graphs in my simulation, when I felt the horrible face of uncertainty, it was my friends and fellow lab mates, who brought me back to the world of hope. I thank them: Mathew Francis, Prathima Addanki, Ram Prabhakar, Sajith Kumar K K, Sankar Srinivasan, and Uma Goguluth: for the sleepless nights, support, motivation, and funny game sessions.

I thank Mr. Susanth Panigrahi and Jefri Lazarus for the stimulating discussions, knowledge sharing sessions, and all the support that they have given me during this work.

I thank Mr. George Tom Varghese for the help and support that he gave me to procure the Xilinx ISE software from the department of electronics, nit Rourkela.

Last but not the least, I would like to thank my family: my parents Aravindakshan.U and Sathee Devi.M, for supporting me throughout my life.

Signed: ______________________________

Date: ______________________________

"I dedicate this work to the society, to my country, and for all those who possesses a dream to revolutionize the world "

Sreejith Markkassery

Contents

List of Figures

CHAPTER 1

Introduction And Problem Statement

1.1 Introduction

Most of the image processing algorithms are sequential in nature. High-level languages like Mat-lab, C++, Open-CV etc. are the common platforms to develop and validate such algorithms. These platforms are most suited in applications, which does not have any time restrictions. In other words, where the response time is not so important. In real-time systems, the high level platforms will not be a good choice due to time and resource constraints.

With the advent of Micro-controllers, it is possible to design embedded image processing systems, which are portable, less power and time consuming. In micro-controller/dsp processors, the algorithm execution is sequential in nature. The speed of execution is greatly increased in advanced processors, which makes use of pipe-lined, and super-scalar architectures. The advanced processors incorporates parallelism at instruction level, but the over all execution of the algorithm will be sequential in nature. Thus a micro controller based system can not effectively utilize the inherent parallelism involved in most of the image processing algorithms. This imposes a limit on maximum processing rate. Thus such devices are not a suitable candidate for time critical applications.

Field Programmable Gate Arrays(FPGA) on the other hand gives a platform for parallel execution. In an FPGA based design, different hardware blocks executes the sequences of an algorithm in parallel, and thus provides quick response and high frame rate. Since the overall operations are performed in less number of clock cycles, the power consumption will be reduced considerably, compared to micro-controller/dsp-processor based designs. The following sections describes the advantage of FPGA based image processing in detail.

1.2 Embedded Image processing

An embedded system is a small computer which is embedded with in an integrated circuit. Such systems are designed to perform a specific tasks. Embedded systems which are designed to carry out image processing operations are known as Embedded Imaging Systems[1]. Most common example for such a system is a digital camera. Here the pre-processing, and compression of the image is done using micro controllers. Generally, the embedded image processing systems are time critical in nature. Such systems are known as Real-Time systems. In other words, real-time systems demands response with in a specified time, other wise the system is considered to be failed. An example for such a system is , one used to determine the area for crash landing of an unmanned areal vehicle(UAV), using passive sensors(Camera) .During crash landing, the UAV has to quickly search for a suitable premiss, and should make a decision. Such an application demands very high speed execution of image processing algorithms. In such applications, FPGA's can effectively replace embedded controllers.

A micro-controller or a dsp processor, executes algorithm sequences sequentially. The instructions will be fetched, decoded and executed. In advanced controllers, the fetching decoding and execution will be done in a pipe-line. This is shown in figure1.1. Super scalar architectures implements instruction level parallelism. The concept is similar to a pipe lined architecture. In a single clock cycle, multiple instruction will be executed using redundant functional units with in a single processor. In super scalar architecture, multiple instructions need not be in different stages, like that in a pipe-lined CPU (central processing unit). Using such advanced architectures, the overall execution speed of an algorithm can be improved by a small fraction. In-order to achieve significant acceleration, different algorithm sequences has to be executed in parallel.

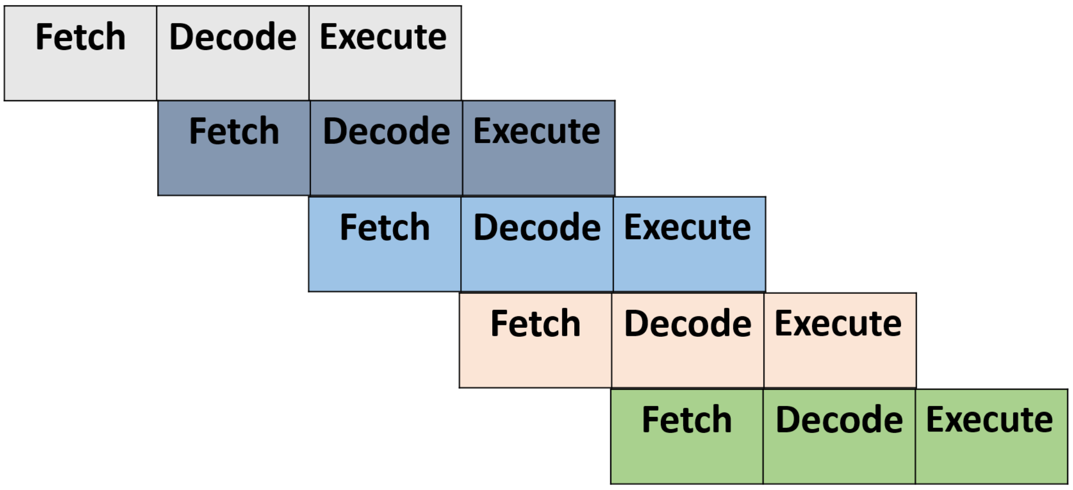


FIGURE 1.1: A 5 stage instruction level pipe-line

Such a design can not be implemented in traditional micro-controller/dsp processors.

1.3 Motivations for Hardware Image Processing

As explained in the previous section, a micro-controller/dsp processor executes algorithm sequences sequentially. If multiple hardware circuits can be designed to carry out different algorithm sequences in parallel, there will be considerable increase in overall execution speed. Suppose a system has to be designed such that the brightness of the incoming frames has to be increased. Brightness of an image can be increased by multiplying each pixel gray level with a constant α, and then adding a gain constant β to it as in equation1.1

$$\boxed{g(i,j) = f(i,j) \times \alpha + \beta} \tag{1.1}$$

In a typical micro-controller/dsp processor based design, this will involve storing the frames in a buffer, and then performing the operations mentioned in equation1.1 to each pixel gray level, in a loop. Suppose each addition instruction takes 12 clock cycle, and each multiplication instruction takes 36 clock cycle, then total number of clock cycles required to process one pixel will be 48. If the incoming frames are of size 100 $\times$ 100, then such a design will need 100 $\times$ 100 $\times$ 48 clock cycles to process the entire frame.
Now suppose, there are 10000 adder and multiplier circuits, one cosponsoring

to each pixel. In such a design, all the pixels can be processed in parallel. Thus total operation can be implemented in just two clock cycles. In principle, such a system will be 12000 times faster than that of a micro controller/dsp processor based system. In actual designs, algorithm will be divided in to parallel blocks and will be executed simultaneously. In a nutshell, the significant increase in processing speed is the major motivation behind hardware image processing. If the processing time is less, the power consumption also will be reduced. Hence it can be concluded that hardware image processing systems give better performance in time critical applications. In the current scenario, most of the image processing algorithms are running in a sequential environment. Hence a research in FPGA based image processing has grater significance and scope in time critical applications.

1.4 Parallelism

Most of the image processing algorithms have inherent parallelism in them. The processing speed can be improved by executing the sequences concurrently. In principle, all the algorithm sequences can be implemented in a separate processor. But if each step depends on previous algorithm sequences, the processors will have to wait for the results from previous stages. Thus the reduction in response time of the system will be very little. For practical implementations in a parallel architecture, algorithm should have significant number of parallel operations. This is Known as Amdahl's law [1] . Let 's' be the proportion of total number sequences in an algorithm, that has to be executed sequentially. Let 'p' be the proportion of the algorithm that can be executed in parallel, using N different processors. Then the best possible speed up that can be obtained is given by equation1.2 [1]

$$\boxed{Speedup \leq \frac{s+p}{s+\frac{p}{N}} = \frac{N}{1+(N-1)s}} \tag{1.2}$$

The equality can only be achieved if there are no additional overhead like communication, introduced as a result of conversion of sequential algorithm to a parallel one. In practical scenario, the actual speed up will be always less than the number of processors N. As N increases, the execution speed also increases.

Ideally, if N tends to infinity, the over all execution speed of the algorithm depends solely on the proportion of the algorithms, that has to be executed sequentially. This is given in equation1.3[1]

$$\boxed{\lim_{N\to\infty} Speedup = \frac{1}{s}} \tag{1.3}$$

Thus to achieve significant speed-up, the proportion of algorithms which can be executed in parallel should be more. Most of the image processing algorithms are parallel in nature.

1.5 Why FPGA

An FPGA based design is inherently parallel in nature. Different algorithm sequences will be mapped to different hardware modules in a FPGA, which operates concurrently. The main reasons for choosing FPGA as an embedded image processing platform are as given below.

- Parallel operation
- Speed of execution
- Flexibility
- Low power design

1.5.1 Parallel operation

An embedded imaging algorithm can be implemented either using a micro-controller/dsp-processor or using a FPGA. A micro controller is a mini computer embedded in a chip, which does a specific task. The hex format instructions which are burned in to the chip will be executed sequentially and will be decoded to control signals, to perform the required task.
From an outer perspective FPGA is a collection of logic elements which can be electrically re wired. FPGA implements an application by developing separate hard ware for each functionality and hence such designs are inherently

parallel[8]. Each instructions that the programmer enters will be mapped in to a separate hardware component. Thus, such a design is suitable in those image processing algorithms, which has significant amount of parallelism in them.

1.5.2 Speed of execution

Due to parallel nature of FPGA's, the execution speed[8] will be considerably increased. In practical applications, the image will be partitioned in to different sub blocks and then each block will be processed in parallel. This will speed up the over all algorithm. The total number of processor will be equal to number of parallel blocks.

1.5.3 Flexibility

An FPGA based system provides full programming flexibility[8]. Current FPGAs have sufficient logic resources to implement even complex applications in a single chip. Modern FPGA based systems will adaptively reconfigure according to the different operating environments. Hence FPGA based systems are inherently flexible.

1.5.4 Low power design

An FPGA based circuit implements several operations in one clock cycle simultaneously. This allow clock speed to be lowered significantly. In fact there will be a reduction in clock speed over a serial processor of the magnitude of 2 or more. Reduction in clock speed corresponds to reduction in dynamic power consumption[8] of the system. Thus hence FPGA based design facilitates a low power design.

1.6 Selection of an FPGA board

Many manufacturers sell several FPGA development boards and evaluation boards. In particular, most boards provide some form of external memory and some means of interfacing with a host computer. Since image processing involves large amount of parallel data manipulation, such systems should have largest possible FPGA. Basic minimum requirements of an FPGA kit which can be used for embedded image processing are[2]

- A codec to decode the composite video signal in to its color components and to digitize it (common interface includes USB camera link etc.).
- Some method to display the result of image processing.
- The system must have sufficient memory to buffer one or more frames of video data.

The above requirements are very general, and the selection of a particular board is application dependent. The basic specifications that one has to look at while going for an FPGA board are

- Number of pins for peripheral connections(Number of multi standard pins)
- Logic Density
  - This indicates the number of logic cells in side an FPGA. The capacity of an FPGA is specified in terms of number of logic cells.
- Dedicated carry logic, dedicated pipe-lined multipliers, etc to increase the execution speed
- Amount of physical memory (Distributed and Fast block ram)
- DCM (Digital Clock Manager) clock frequency range, is another parameter for applications that demands high frame rate
- Availability of an embedded controller for processing serial parts of the algorithms, and for video acquisition
- Cost and availability of support

All the above parameters are not important in all designs. But a complicated image processing system typically makes use of all the above facilities. Thus careful selection hardware is a vital step in any FPGA based real-time design.

1.7 Problem Statement

The objective of this project is to develop a hardware architecture to implement video edge detection algorithm. The work will be focussed on image processing operations over a single frame, which can be easily extended to multiple frames. The architectures should be developed with an aim to implement it in Zynq - 7000 SOC (System on Chip) FPGA board. The specifications of the targeted IP core are given below. The schematic description of the overall problem is given in figure1.2

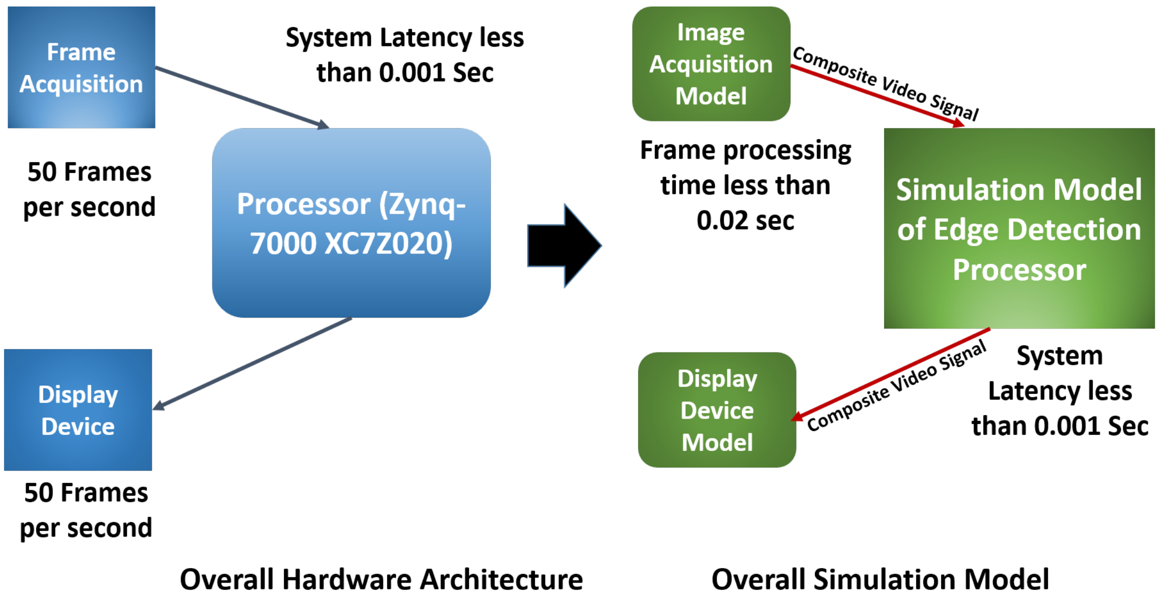

Frame Acquisition
50 Frames per second
System Latency less than 0.001 Sec
Processor (Zynq-7000 XC7Z020)
Display Device
50 Frames per second
Overall Hardware Architecture
Image Acquisition Model
Composite Video Signal
Frame processing time less than 0.02 sec
Simulation Model of Edge Detection Processor
Display Device Model
Composite Video Signal
System Latency less than 0.001 Sec
Overall Simulation Model

FIGURE 1.2: Overall system model

1.7.1 System Functionality

The developed hardware system should be able to capture continuous video stream from a digital camera and should be able to process it to detect edges in each frame. The threshold value to detect edges should be adaptively computed for each frame. The system should display the processed frames on a display device with detected edges distinguished with white color. In the simulation environment a model should be developed, to simulate the functionality of a digital camera (Acquisition System). The image data should be converted in to signals, that are identical to the signals from a digital camera. FPGA based image processing module should be simulated in software platform. This module should accept inputs from the simulation model of acquisition system, should produce the edge detected frame, and should display it in image/video format in a Graphical User Interface (GUI). The GUI should allow users to analyse the output wave forms and data flow.

1.7.2 System Performance

The performance of the system is evaluated by the number of frames the system can process per second. The targeted performance is 50 frames per second for 256×256 frames, which is far above the frame rate that can be obtained in a sequential environment like Matlab (In Matlab it is 10 to 12 frames per second). The clock speed of the system has to be designed according to the targeted system speed. Pipe-lining method will be adopted to achieve higher speeds, at comparatively lower clock speeds[11]. Another important measurement of system performance is system latency. It is the difference between the time at which pixels are fed in to the system and and time at which a completely processed pixels are obtained at the output. The system is expected to have a latency less than 0.009 seconds.

1.7.3 Operating Environment

In principle, the system should be able to detect edges in all conditions. But due to time constraints of the project the operating environment is fixed at natural day light conditions. A good constant light is assumed and the pre-processing modules to compensate for bad light will not be implemented as a part of this project.

1.7.4 Targeted Hardware Platform

The targeted hardware platform is Zynq-7000-SOC FPGA Evaluation board. This board has a in built video acquisition system (VITA 2000 Image sensor), and dedicated IP cores to acquire frames to main memory. The board has a physical memory of 1 GB, which is sufficient for most of the image processing applications. The board has an inbuilt HDMI (High Definition Media Interface) Codec(Coder Decoder) which can be utilized for HD (High Definition) display of processed outputs. In this project, the inbuilt video acquisition and arm processor will be made use of to detect the edges.

1.8 Implementation Strategy

The overall project is split in to four phases as in figure1.3. In the first phase, the system functionality and performance parameters were

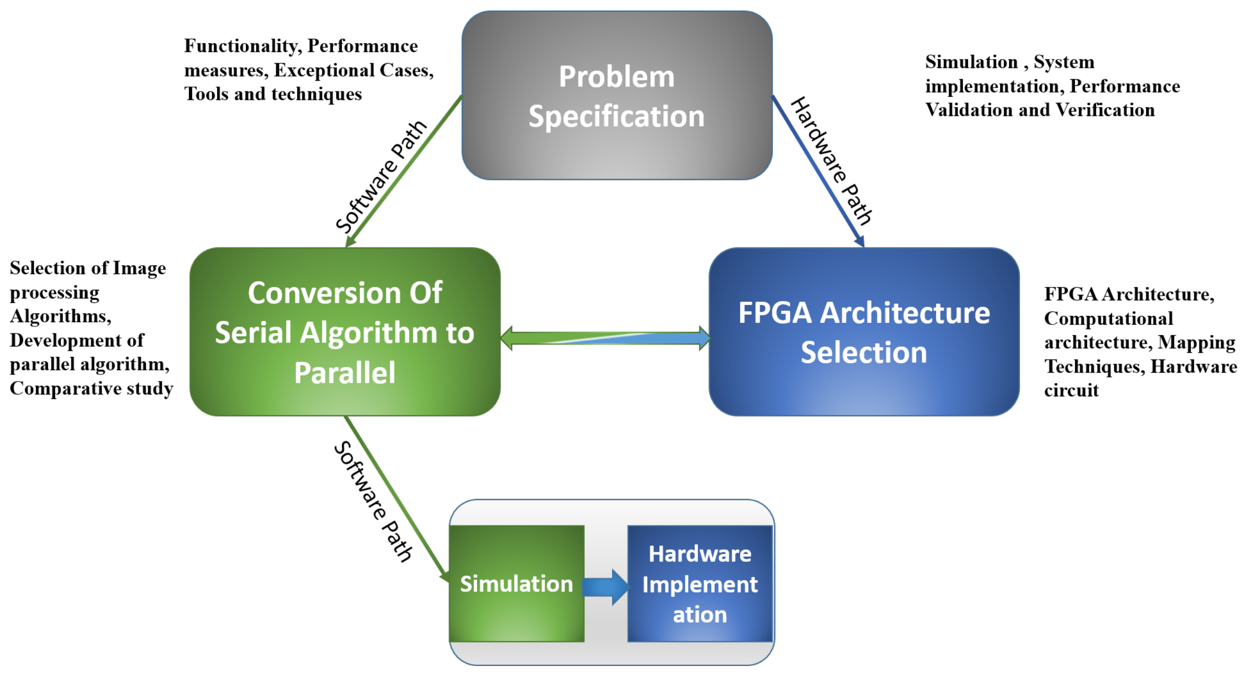

Functionality, Performance measures, Exceptional Cases, Tools and techniques
Problem Specification
Simulation , System implementation, Performance Validation and Verification
Software Path
Hardware Path
Selection of Image processing Algorithms, Development of parallel algorithm, Comparative study
Conversion Of Serial Algorithm to Parallel
FPGA Architecture Selection
FPGA Architecture, Computational architecture, Mapping Techniques, Hardware circuit
Software Path
Simulation
Hardware Implement ation

FIGURE 1.3: Project Implementation Strategy

established. Literature on standard embedded imaging techniques(Chapter2) was conducted. Furthermore, a study on FPGA and its application in embedded image processing was carried out. The embedded image processing techniques(Chapter2), to realize the performance parameters were identified and short listed in this phase. Exceptional cases in which the system fails, were identified and documented.

The second and third phase will be carried out in parallel. In the second phase, the image processing algorithm was selected. The sequences of algorithm that can be converted in to parallel were estimated(Chapter4). Based on this estimation, FPGA system architecture was selected(Chapter2). At this stage a comparative study was carried out to estimate the speed up achieved by parallel processing.

The next phase is FPGA architecture selection. This was done in parallel with the phase two. Based on the parallel algorithm developed, suitable FPGA system architecture(Chapter2) was selected. Thereafter, the FPGA computational architecture(Chapter2) was selected based on the established system performance measurements .Next step in this phase was to select the FPGA mapping techniques(Chapter2) so as to meet the timing and band width constraints.

The last phase of the project is system implementation and simulation. The overall architecture was be implemented in VHDL, and was simulated using ModelSim and Xilinx ISE(Chapter4). A test bench was developed, to model the image acquisition system and this model sourced the actual edge detection architecture(Chapter3). A detailed system testing was carried out in this phase and the variations from the expected results were mitigated in each iteration. Moreover, a new software platform was developed to effectively simulate the overall FPGA architecture(Chapter5). The new simulator will show the result in image format, unlike the traditional VHDL simulators like modelsim and Xilinx ISE.

CHAPTER 2

Background on Embedded Image Processing

2.1 Introduction

An image processing algorithm implemented in an embedded platform is known as embedded image processing. There are two types of embedded image processing systems. Hardware and software based. Hard ware embedded systems are relatively faster and mostly designed using an FPGA . Since FPGA's are reconfigurable, the same flexibility to that of a software based embedded system with an improved speed can be achieved, but at the expense of increased cost and difficulty level in system design.This chapter briefly describes the general properties of an embedded imaging system.

2.2 Embedded Imaging Techniques

2.2.1 Real-time System

A real-time system[1] is one in which the response to an event must occur within a time limit, otherwise the system is considered to have failed. From an Image processing perspective, a real-time imaging system is one that acquires images, processes those image to produce some results, and then utilize this results for further processing. The response to the event should occur with in the specified time. The examples are Robot vision system, in which captured images will be analyzed to find out obstacles in its path.

Real-time systems are categorized into two types: hard and soft real time.In a hard real-time system, the system is considered to be failed if the response doesn't happen with in the specified time. The crash landing of an unnamed areal vehicle is an example . If the landing site is not determined with in a specified time, the system is considered to be failed. On the other hand, a soft real-time system is one in which system will not be completely failed , even if the responses are late. An example is video transmission via the internet. If a frame is delayed or not decoded properly then it will exacerbate the quality of the video.

2.2.2 Performance Measurements of a Real-Time System

2.2.2.1 System Latency

In simple words latency[1] of an embedded imaging system is the difference between the the time at which a pixel is read and the time at which it is displayed after internal processing. Lesser the latency the better the system is. One method to improve the latency is to increase the system clock speed. But this would result in larger power consumption. Another and most efficient method is to implement a multi-stage pipeline. A pipeline is collection of parallel hardware units, which simultaneously processes different pixels at the same time.

2.2.2.2 System Throughput

The system throughput is synonymous to system band width. From the perspective of an embedded imaging system, system throughput is the number of pixels processed in a single clock pulse. More than one pixel operation will demand more time, and lesser clock speed. This will reduce overall system speed, but the throughput will be increased. Increased system throughput can be achieved through multi-stage Pipe-lined designs(Chapter5). In such designs, while the current pixels is in it's last processing stage, the previous one might be in the penultimate stage. Thus the number of pixels processed in a single clock cycle will depend on the number of pipe-line stages. The individual pipe-line stages should posses equal timings, to achieve maximum throughput.

2.2.2.3 System Bandwidth

System band width is defined as the total memory usage of the system in one clock cycle. In embedded imaging Systems, frames will be stored in a buffer. The pixels will be accessed from this main memory, and will be processed and written back in to the memory. Such a system will have frequent memory accesses to fetch a single pixel, and to write back the processed pixel. Frequent memory access will create pixel bottlenecks, and will increase the system latency. High system band width will adversely affect the system efficiency. The system band width can be optimized using cache memories. The pixels can be fetched as packets of four or eight, and can be stored in cache memories. Access of cache memories are faster than that of main memories like block ram.

2.2.3 Serial Vs Parallel Processing

Sequential image processing platforms are based on serial computer architecture. Such a serial processor operates by fetching the instructions sequentially, and by decoding it in to arithmetic and logic operations. This task will be performed by the ALU (Arithmetic Logic Unit). The rest of the CPU (central processing unit)feed ALU with necessary data. A compiler will compile the algorithm in to sequence of instructions, and these instructions will be decoded by the CPU(Central Processing Unit) and ALU during each clock cycle. The basic operation of the CPU is therefore to fetch an instruction from memory, decode the instruction to determine the operation to perform, and execute the instruction.

An image processing algorithm consists of a sequence of image processing operations. This is a form of temporal parallelism. This parallel nature can be utilized with a pipe-lined multiple processor architecture as shown below in the figure 2.1 . The data passes through each processor while it proceeds. In other words each processor applies its operations on the data and passes it to next stage. Considerable amount of acceleration can be achieved if processors don't have to wait for the input from any other stages.If the algorithm has significant amount of sequential operations, one processor will have to wait for the result of other. This will incur additional communication overhead. However, the system throughput can improve since the first processor is processing data while a part of the data is being processed in the second processor. Data will be sent to the output device, before the completion of total operations, to reduce the system latency .

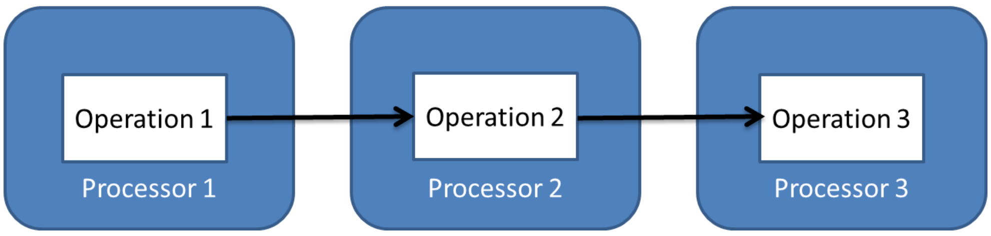


FIGURE 2.1: A processor array used to execute operations in parallel

2.2.4 Conversion of serial Algorithms to Parallel

An algorithm can be implemented in a massively parallel architecture, only if it has significant amount of sequences that can be executed in parallel. Algorithm sequences which has inter dependencies , will create bottle necks and additional communication overhead. Thus it is important to convert all possible sequential algorithm sequences in to parallel ones. Once the parallel sequences are identified, each sequence will be mapped to a hardware architecture/circuit. This process will be repeated for all the identified parallel processes. In the next step, these mapped sub-circuits will be integrated, and the miscellaneous communication overhead between each sub-circuit will be investigated. Aggregation of all individual circuit will collectively produce the desired result.

After establishing the overall architecture for parallel sequences of the algorithm, next stage will be the design of sequential part. Here the communication over head between the parallel blocks will be estimated. The communication channel and protocol between parallel and sequential processors will be designed. A detailed description such techniques are presented in chapter5. The integration between the parallel and sequential processors is critical. There will be stealth interactions between two modules, which were not identified during the initial stages. These faults should be mitigated iteratively. Rigorous integration testing is essential in each iteration.

2.2.5 Resource Vs Speed

In an FPGA based design, the availability of resource is a key factor. FPGA resources are specified in terms of logic density. It indicates the number of logic cells inside an FPGA. The more the resources, higher will be the speed. Moreover, if the selected board has dedicated carry logic, pipe-lined multipliers and ALUs, the execution speed will be further improved. The improved speed comes with high cost. The high-end FPGAs with advanced resources, are highly expensive. Hence a trade-off between the cost and speed will have to be maintained.

2.2.6 System Band-width Vs Resources

In an embedded image processing design, high system band width will lead to increased system latency, and hence will create pixel bottle necks. The system band width can be minimized, if the the high speed memory resources in an FPGA are sufficiently high. A large cache memory will considerably reduce the number of frequent access to main memories, and will save the system band width. A design with cache memories to hold both data and results is highly efficient.

2.2.7 Band width Vs Speed

The system band width and speed of operation are inversely proportional to each other. Higher bandwidth will result in higher system latency and the system will be sluggish. System speed can be improved by increasing the clock speed. But the clock speed should be synchronized with the incoming data stream, other wise data will be lost. Furthermore, the clock speed should be sufficient to complete a pixel operation in a single stage of a multi stage pipelined design.

2.3 FPGA System Architectures

An FPGA system architecture defines the overall structure of the system, that is going to be implemented. The selection of this architecture depends on the proportion of sequences in an algorithm that can be executed in parallel. At the architecture selection stage of a project, available FPGA system architectures should be short listed.

2.3.1 Standalone architecture

In this architecture the entire application is implemented using an FPGA based parallel processor[1]. Several hardware blocks will execute different sequences of the algorithm concurrently . The maximum benefit can be extracted if the significant portion of the algorithm can be executed in parallel. Such architectures provides maximum speed of operation and higher efficiency. In general such architectures are known as massively parallel architecture. This project is implemented in a stand alone massively parallel architecture.

2.3.2 Co-processor architecture

In complicated image processing algorithms, there will be sequences, which will have to wait for the execution of previous or future sequences. If such sequential operations are implemented using a parallel processor, it will exacerbate situation because of the communication overhead. In these scenarios a co-processor architecture[1] can be utilized. Major chunk of the algorithm will be executed by parallel processor and the sequential part will be implemented in a serial processor. Proper communication protocol has to be defined between two processors. Parallel processor will be an FPGA based processor and generally the sequential processor will be micro-controller based.

2.3.3 Hardware Accelerator

This architecture can be utilized when most of the algorithm sequences has inter-dependencies[1]. As a result, most of the sequences will have to be executed sequentially. In this kind of designs, a sequential processor will do the major chunk of algorithm and some operations which can be executed in parallel will be fed to parallel processor. The communication overhead between parallel and serial processors should be properly addressed. Best example of such an architecture is floating point processor associated with high end processors. The complex floating point arithmetic operations will be executed by a dedicated processor, and the result will be passed on to a cache memory, which will be shared between both the processors (serial and parallel).

2.3.4 Hybrid processing

This architecture is similar to a co-processor architecture except the fact that the algorithm load will be equally shared between the serial and parallel processors. This architecture can be utilized in situations where only inner most loops of a sequential algorithm can be executed in parallel.[1].

2.4 FPGA Computational Architectures

The computational architecture defines how the computational aspects of the algorithm are implemented. In other words it describes how each image processing operations are performed inside an FPGA. The selection of computational architectures are application dependent. The incoming data rate, inter dependencies in algorithm sequences, system architecture of an FPGA, available resources etc. are some key parameters that has to be considered while selecting computational architectures.

2.4.1 Stream Processing

In any embedded vision application, the data is captured using a digital camera. One method to process the incoming frames are to store them in a frame buffer and then process each frame in parallel. But in this case the number of memory access will be large and consequently the response time will increase. If the pixels do not have dependency with previous one, they can be processed on the fly. In other words the pixels can be processed while they are being read from the camera. This processing is known as stream processing[1]. To reduce the response time, maximum processing has to be done while streaming the image. A pictorial representation of stream processing is given figure2.2

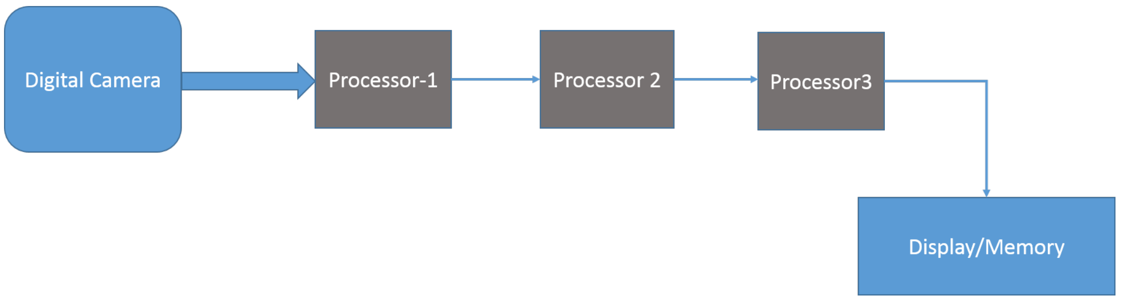


FIGURE 2.2: Stream line processing

One of the main disadvantages of stream processing is fixed clock rate. The clock rate is constrained by the input frame rate. If the clock rate is slower than the input frame speed, several frames will be lost. System latency will be minimum in stream processing.

2.4.2 Systolic arrays

A systolic array is a one or 2 dimensional array of processors, in which data will be processed at the time of streaming, and will be passed between adjacent processors. A systolic array[7] differs from stream processing in the direction of data flow. In the later it is unidirectional, whereas in the former data can be moved in both the directions. Thus if a preceding pixel has any dependency on the current pixel, it will be fed back to the previous processor stage. Compared to the stream processing, the communication overhead is large because of the feedback involved. The operations will be performed in each clock cycle. This kind of architecture is mostly used in object detection and tracking applications. The figure2.3 shows a pictorial description of systolic arrays[11].

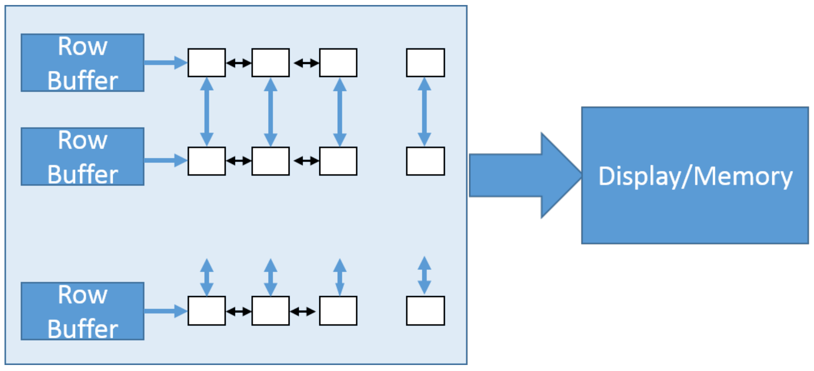


FIGURE 2.3: Computational architecture using systolic arrays

2.4.3 Random Access Processing

In a random access processing[2] method, the pixels can be accessed anywhere from the frame. To achieve this, a frame buffer should be implemented. The incoming frames will be accumulated in the buffer and any pixels form this buffer may be processed randomly. This kind of architecture is useful when algorithm has significant portion of sequences, which cannot be executed in parallel. This kind of processing is equivalent to a sequential data processing. One of the added advantages is that there will not be any hard constraint on the system clock as in the case of stream processing. But system latency will be more, due to frequent and large number of memory access.

Data parallelism can be achieved by partitioning different parts of the image to separate processors. This needs multiple copies of the hardware block, corresponding to each image part. Each part of the image will be in the local buffer of its corresponding hardware block. Implementation of parallelism in random access processing is shown in figure2.4. In this example the frame is divided in to four parts.

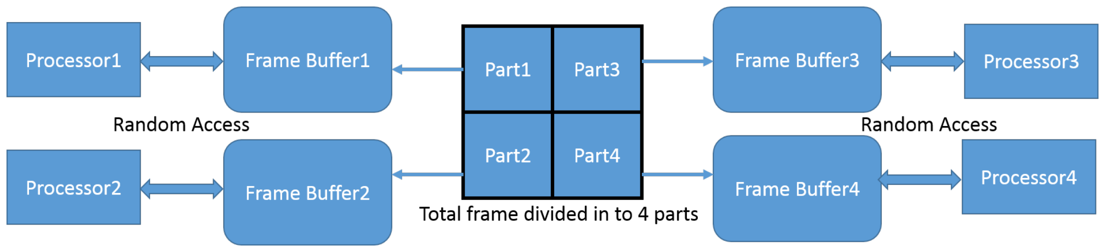


FIGURE 2.4: Example of parallel processing using random access architecture

2.5 FPGA Mapping Techniques

FPGA mapping techniques describes how efficiently each image processing operations are mapped on to FPGA resources. In other words, these are the standard techniques adopted to achieve desired system performance in terms of speed, memory and resources. During the architecture selection phase of the project, a detailed list of constraints and method to improve the system efficiency will be short-listed. Three major constraints has to be addressed, while mapping sequential algorithm sequences in to an FPGA. These are:

- Time constraints
- Bandwidth Constraints
- Resource constraints

The resource utilization depends on the efficiency of the design. It has to be taken care at the logic design level. Following sections describes standard techniques used to address first two constraints.

2.5.1 Time constraints

In real-time applications incoming data rate is one of the major constraint. The data processing has to be done at pixel rate or faster to prevent the data lose. Low level pipe-lining[7] is one of the technique used to overcome this difficulty.

2.5.1.1 Pipe-lining

Pipe-lining splits an operation in to smaller stages, and completes the operation in several stages[3] [1]. As a result the propagation delay in a single stage or the delay in a single clock cycle will be reduced. Since only a portion of the operation is implemented in a single clock cycle, the over all clock speed can be increased. Thus to complete one operation, more than one cycle will be needed. If this hardware is duplicated, multiple pixels can be processed at a time. The number of pixels that can be processed simultaneously depends on the total

stages in a pipe-line.
As an example consider the below equation.

$$\begin{aligned} y &= ax^2 + bx + c \\ &= (a \times x + b) \times x + c \end{aligned} \tag{2.1}$$

The above equations can be implemented using one,two and four stage pipes as shown in 2.5. But the optimum latency can be achieved by a 2 stage pipe as it improves the system throughput. In figure2.5, the two stage pipeline is most efficient. This is because the addition and multiplication operations are evenly distributed between two stages. In the case of four stage pipeline, stages one and three has multiplication operations, where as stages two and four has addition operations. A multiplication operation need more time than an addition operation. In other words, distribution of stage timing is not even. This will generate imbalance, and will increase the system latency. This effect can be mitigated by re-timing the sequences. In this technique, a part of multiplication operation from stage one will be completed in stage two. So proper selection of number of pipe-line stages is most important in FPGA designs.

2.5.1.2 Process synchronization

If all of the external events driving the system follows a specific pattern and if the processing time or latency is evenly distributed among the operations, then global scheduling[1] can be used. This can be accomplished by using a global event counter[1] which synchronizes and schedules the events, as shown in Figure 2.6. The various operations are then scheduled by determining when each operation requires its data and matching the count.

2.5.2 Band width constraints

The incoming frames should be partially or completely stored in most of the image processing operations. Current FPGA systems have large off chip memory resources. But the inefficient use of these resources will lead to a poor system

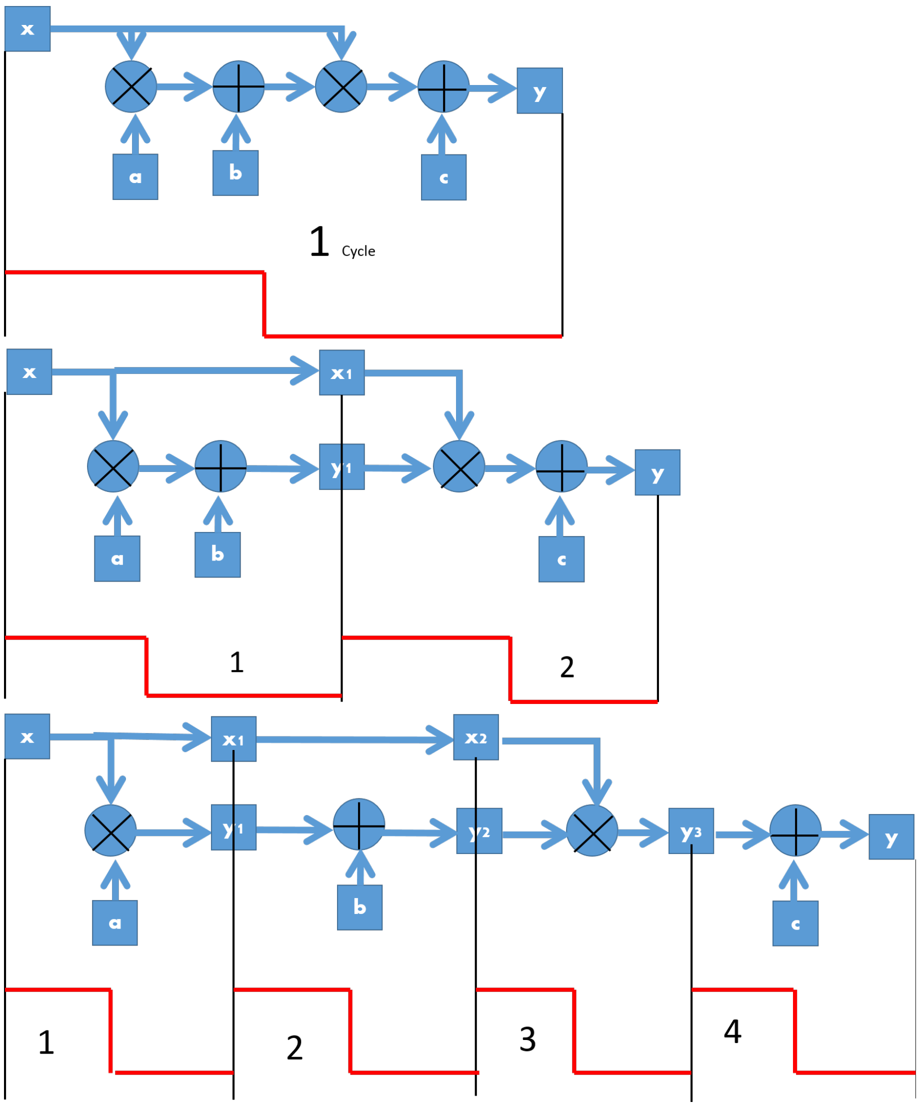


FIGURE 2.5: Pipe-lining example. Top: Performing the clock cycle in one stage; middle: implementation using a two-stage pipeline; bottom: Spreading the calculation over four stages.[1]

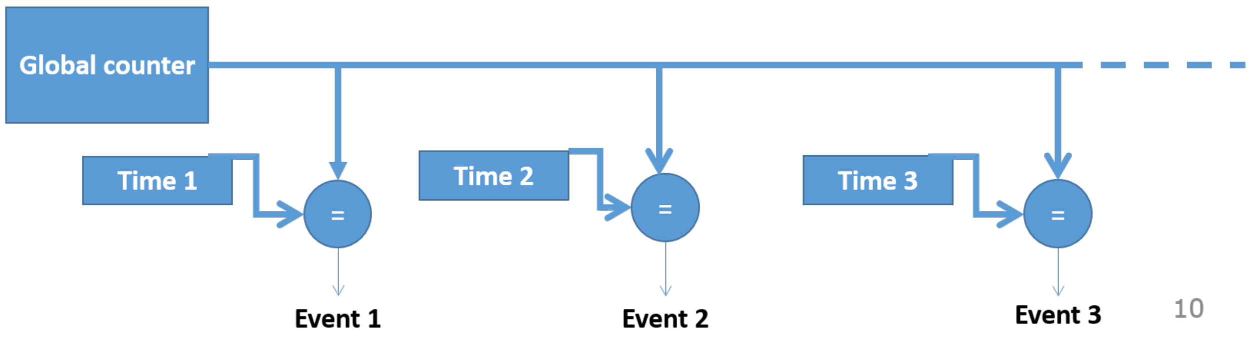


FIGURE 2.6: Synchronization of events using global scheduling

design. If the system off-chip memory is frequently used , it will affect the response time of the system as well as system latency. Techniques used to mitigate memory constraints are summarized in below sections.

2.5.2.1 Double buffering

Double buffering[1] is used in FPGA mapping to reduce number of memory operations. It is used between two successive image processing operations to avoid the bottle neck, when using a shared memory. This technique is mostly used with pipe lining or with random access processing. Data will be processed on the fly, and will be loaded to buffer for random access processing. This technique fits exactly in between the steam and random access processing path. It uses two connected memory banks, as shown in Figure 2.7. The upstream process captures data from input and writes one of the memory banks. Downstream process reads the data in parallel from the other bank and displays it. When the frame is complete, the role of the two banks will be reversed, so that the data just uploaded by the upstream will be now available for the downstream process. Consequently one frame period will be added to system latency.

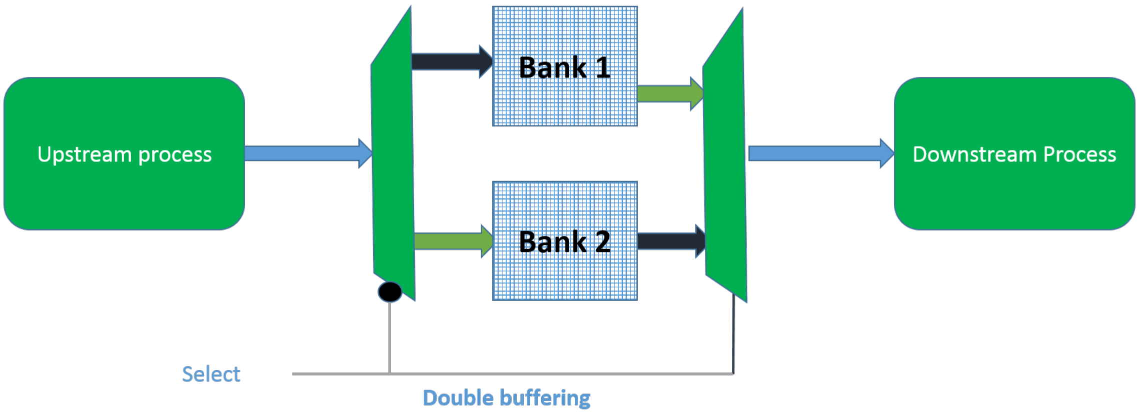


FIGURE 2.7: Double Buffering

2.5.2.2 Cache Memory

A cache memory[7] is a high speed memory located close to the processor, which holds most frequently used data or results of the previous operations.

It acts as a high speed buffer. The advantage of such memory is that the data access will be considerably faster compared to off chip memories, and hence the response time will be considerably improved.

On an FPGA, a cache can be used to improve the system latency. In most of the designs each processing element will be accompanied with a cache memory. Hence the system band width will be considerably reduced. In some configurations there will be a data cache and result cache to store normal data and the computed results. In some applications, the cache is placed beside the processor and the main memory, as shown in 2.8, with accesses made to the cache rather than the memory.

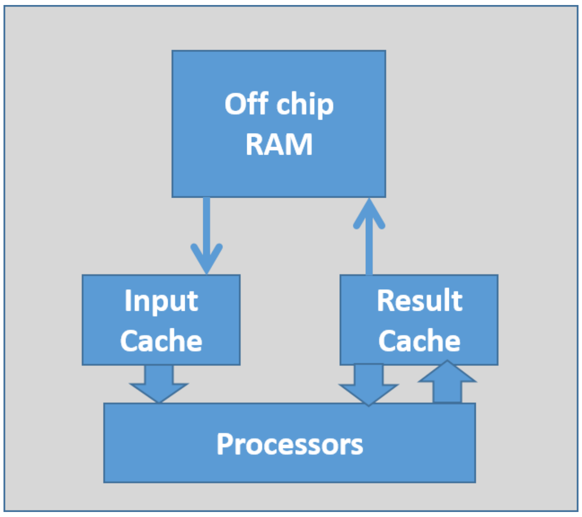


FIGURE 2.8: Caching as an interface to memory

2.5.2.3 Row buffering

One of the method in which caching is implemented in image processing operations is row buffering[1]. Consider 3×3 window filter as in figure2.9– each output pixel is a function of 8 neighbouring pixel in the window. In normal implementation each pixel should be sequentially read from off chip memory to calculate the current pixel value (each clock cycle for stream processing) and each pixel must be read nine times as the window slides through the image. To reduce the number of memory accesses, each pixel which are required in successive clock cycles can be buffered in a register and those can be accessed from the buffer. In row buffering the pixels that has to be accessed from the previous 2 rows will be buffered. This is explained in the figure 2.10. A 3×3 filter

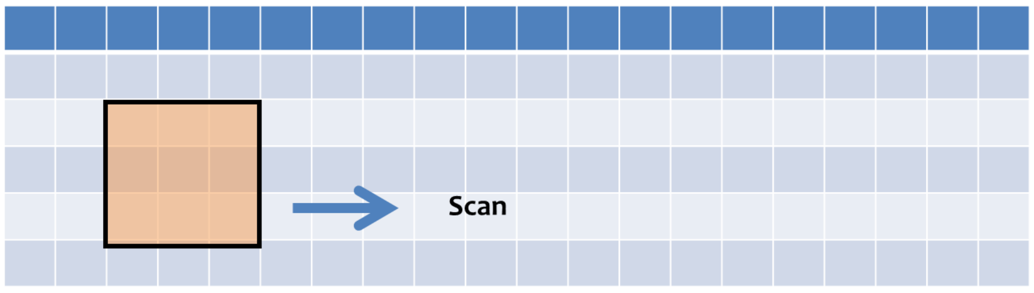


FIGURE 2.9: General row scanning in an image processing design

spans three rows, the current row and two previous rows; so two row buffers are necessary to cache the pixels of previous rows. This is pictorially explained in figure2.10

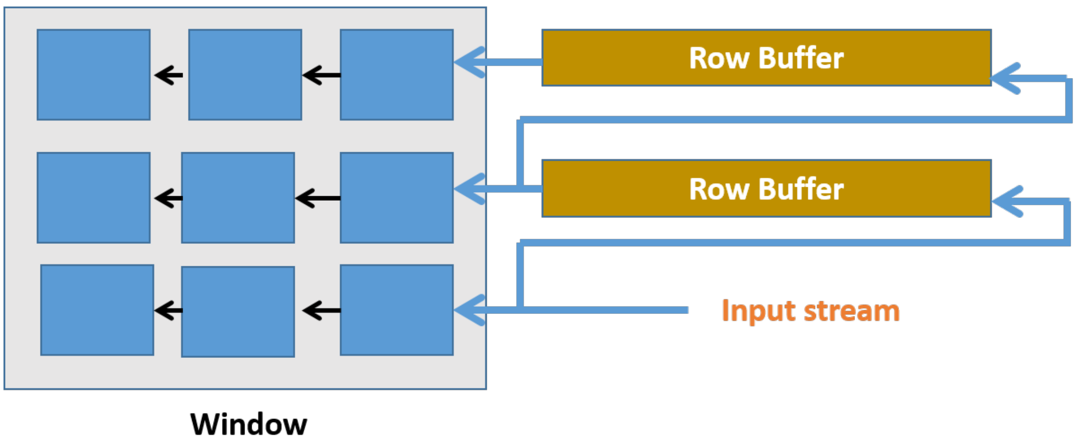


FIGURE 2.10: Row buffering using cache memory

2.5.3 Summary

Selection of the FPGA mapping technique is application dependent. a subset of the available techniques should be selected and used appropriately in all the designs. The selection of mapping technique is done at the design stage of the process. Some times to achieve, higher efficiency a combination of all these techniques will be used (Hybrid techniques). Complex image processing operations make use of hybrid techniques.

CHAPTER 3

Design Of A VHDL Test-bench For Real-time Image Processing

3.1 Introduction

A real-time [2] system is one in which the response for an input or an event should occur with in a predefined time limit, else the system is considered to be filed. From an image processing perspective the real-time system should process the incoming frames and produce or extract the desired features with in the specified time limit. To achieve maximum performance, the real-time image processing systems are designed as hard ware systems, in which the parallelism is well exploited.

3.2 Motivation for the Test-Bench

Before implementing a real-time system, its validation in a software platform is a necessity. This validations can serve as a proof of concept, in earlier stages of the design. Moreover, if the validation is done after the hard ware implementation, it will be very difficult to trace back the design and identify the exact problems if there are any. Another problem with direct hardware implementation is the time, effort and cost incurred while tracing back through design.

Hence in a nutshell a hardware implementation of a real-time system without proper validation in a simulation environment will make the design inefficient and cumbersome.

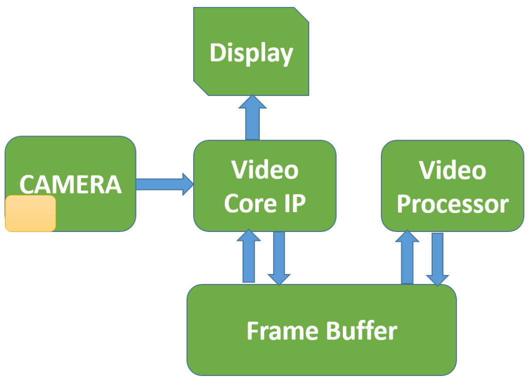
Having developed the motivation for validation of a real-time design before hardware implementation, the next objective is to develop a generic software model for the simulation. Hence a software model to simulate the FPGA based image processing designs was developed. In particular, a software model was designed to simulate a video acquisition system. This model sources the actual edge detection processor.

3.3 A Real-time Video Processing System

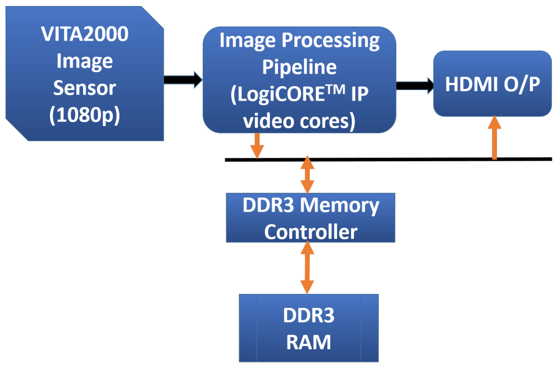
The block diagram of a real-time video processing system is shown in figure 3.1a. The video will be captured by the image sensor inside the camera. A video intellectual property (IP) core, which may be a soft-core sequential processor or a micro-controller, will pre-process this video and will store the frames in a buffer. The frame buffer can be a DDR3 random access memory (RAM). The video pre-processing may include color image filtering, conversion of color frames in to gray level, smoothing of frames to reduce noise etc. The stored frames will be accessed by a main video processor which may be an FPGA based parallel processor, for further processing.

The purpose of the video processor can be any image processing operation like edge detection, face detection etc. The output of the processor will be again written back to the frame buffer. The same video core IP will act as a driver between external display and the frame buffer. The buffered frames will be then sent to the external device. The figure 3.1b shows a real-time video acquisition system [10] implemented in Zynq -7000 SOC (System on Chip) board. Here the frame buffer is a DDR3 RAM (Double Data rate type 3 random access memory), and is written through a DDR3 memory controller. The external device will be connected to HDMI port of the kit. The acquisition unit includes, a VITA -2000 image sensor and a LogiCORE^{TM} IP video cores(A soft core processor) [10].

The objective of the design is to model the video acquisition system shown in figure3.1a. Here the video acquisition system should include the camera and



(A) Generic


VITA2000 Image Sensor (1080p)
Image Processing Pipeline (LogiCORE™ IP video cores)
HDMI O/P
DDR3 Memory Controller
DDR3 RAM

(B) Specific to Zynq-7000

FIGURE 3.1: Real time video processing system

the video core IP. The design developed [13] will generate signals similar to that of a video acquisition system.

3.4 Composite Video Signal

In order to simulate a camera module the characteristics of camera output signal has to be studied. A composite video signal[14] representing a single row of a frame from a camera is shown in figure 3.2. As the figure indicates the composite signal contains both the video as well as synchronization signals. At the end of each row a horizontal synchronous pulse will be generated, which will inform the following processor about an end of row event. At the end of each frame a vertical synchronous pulse will be generated. Putting in another words, a horizontal synchronous pulse will differentiate between different rows in a frame, where as a vertical synchronous pulse will differentiate between different frames. There are another class of synchronization pulses which are used to retrace from one end of the row to the beginning of the next row. In normal FPGA based image processing systems, this event can be ignored.

The simulation model of the video acquisition system should be able to generate a composite video signal at its output. The generated signal will be fed to the input of FPGA based parallel processor. In digital perspective, the signals will not be superimposed. The horizontal, vertical synchronous pulses and the data will be given as separate inputs to the processor as in figure3.3.

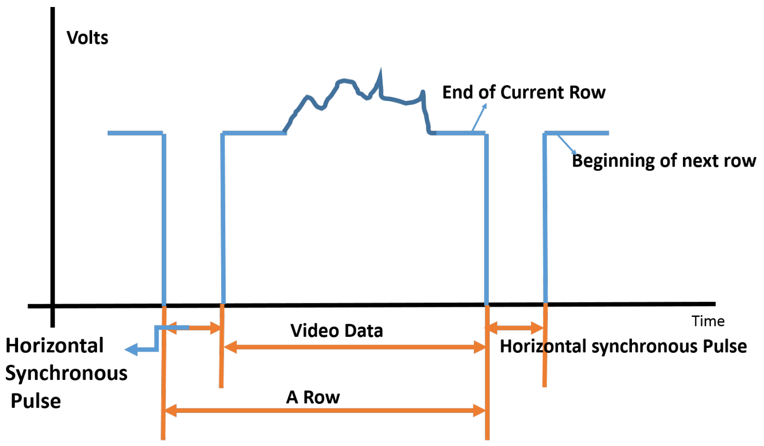


FIGURE 3.2: Composite video signal for a single row in analog form

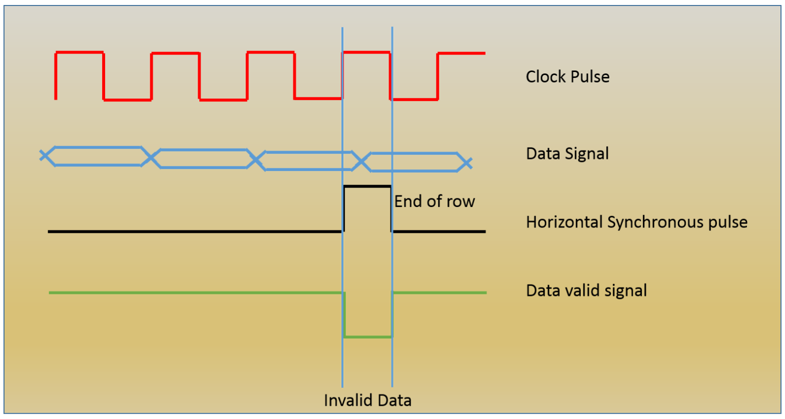


FIGURE 3.3: Composite video signal for a single row in digital form

3.5 VHDL as an Image Processing Platform

In an FPGA based real-time designs, the image processing algorithm should be coded in VHDL or any other hardware description language (HDL). Thus it is necessary to investigate about the image processing capabilities of HDL, in the early stages of the project. The hard ware description language used for this project will be very high speed IC hardware description language (VHDL)[8].

VHDL is a not a high level language and each instruction will map to a corresponding hardware component in an FPGA. So most of the image processing capabilities of VHDL can be extracted, if it is used as a platform in FPGA based design. An efficient VHDL design can boost up several sequential image processing algorithms in FPGA based systems.

On the other hand, the image processing capabilities of VHDL are not so great if it is used as a modelling platform in a sequential environment. This is because unlike the high-level languages, VHDL lacks an in-built codec which can decompress the image and read it. More over as an HDL, it has limited addressing techniques like pointers.

These difficulties can be overcome through the clever usage of TEXT IO package, and 2D x 2D arrays. If the image can be provided as a text file to the input of a VHDL system, then using the above tools the image processing operations can be implemented and simulated.

3.6 Image Processing Test Bench

In a nutshell, a video processing test bench should be able to capture frames, should be able to generate a composite video signal and the clock pulse, and should be able to source a video processor. To implement such a test bench in VHDL, the video frames should be available as an input text file. Due to the inherent inability of VHDL to read an image, an external program which converts video frames to text format was developed. The same external program will reconstruct the image from output text file, generated by the VHDL module. The overall block diagram of the test bench [13] is as shown in figure 3.4.

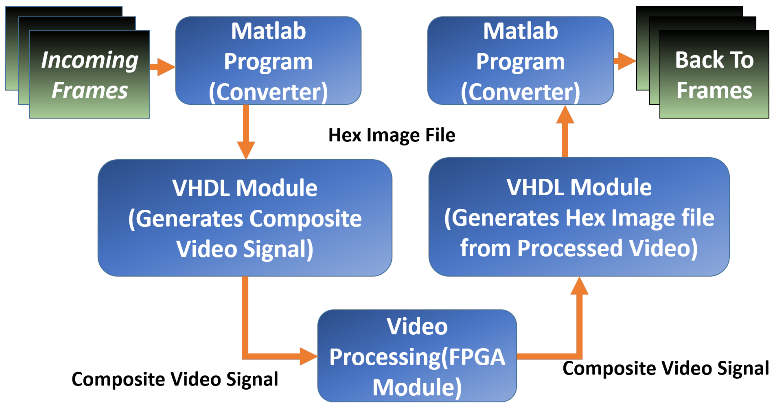

Incoming Frames
Matlab Program (Converter)
Matlab Program (Converter)
Back To Frames
Hex Image File
VHDL Module (Generates Composite Video Signal)
VHDL Module (Generates Hex Image file from Processed Video)
Video Processing(FPGA Module)
Composite Video Signal
Composite Video Signal

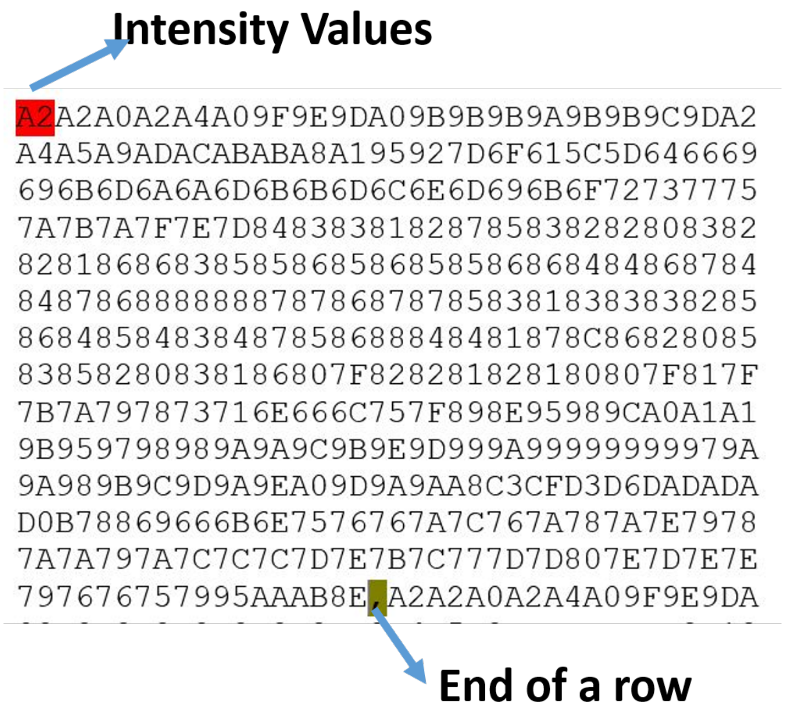
FIGURE 3.4: Block diagram of a video processing test bench

3.6.1 Image to Hex-Image Converter

This is a Matlab program which converts the incoming video frames in to hex image file. The video will be read in to the Matlab environment and each row will be converted in to hex format.

3.6.1.1 Hex-Image File

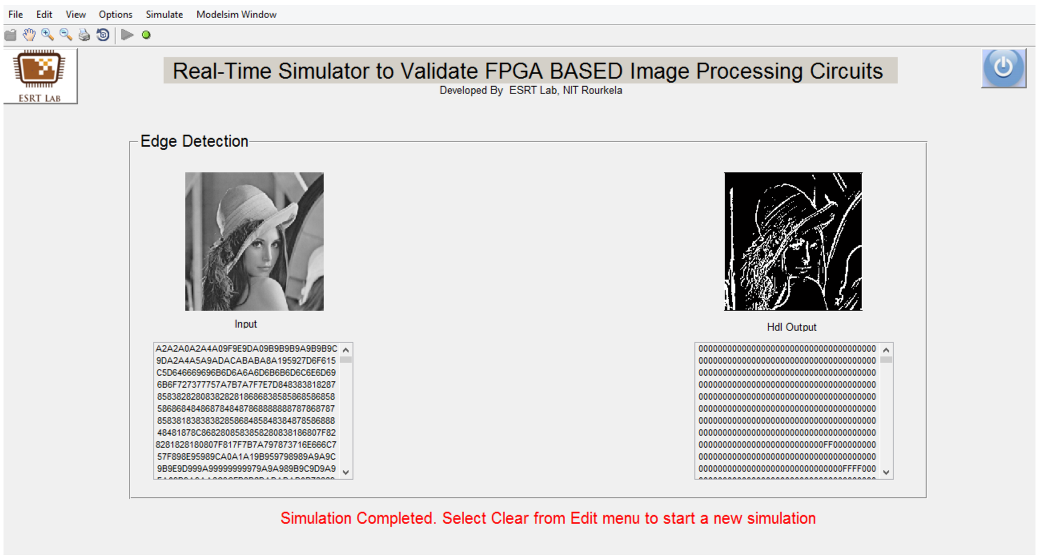
This image file[13] will be generated by an external Matlab program. Each intensity values in the frames will be converted in to hexadecimal values and stored as a continuous sequence of 2 characters. For example the intensity value 8 will be stored as '08' and the intensity value 255 will be stored as 'FF'. In other words, the intensity values will be coded as '00' to 'FF'. At the end of each row a ',' character will be inserted. A '*' character will be inserted at the end of each frame. This coding scheme is shown in figure 3.5. The hex image file generated



```
A2A2A0A2A4A09F9E9DA09B9B9B9A9B9B9C9DA2
A4A5A9ADACABABA8A195927D6F615C5D646669
696B6D6A6A6D6B6B6D6C6E6D696B6F72737775
7A7B7A7F7E7D84838381828785838282808382
82818686838585868586858586868484868784
84878688888887878687878583818383838285
86848584838487858688848481878C86828085
8385828083818680 7F828281828180807F817F
7B7A797873716E666C757F898E95989CA0A1A1
9B959798989A9A9C9B9E9D999A99999999979A
9A989B9C9D9A9EA09D9A9AA8C3CFD3D6DADADA
D0B78869666B6E7576767A7C767A787A7E7978
7A7A797A7C7C7C7D7E7B7C777D7D807E7D7E7E
797676757995AAAB8E,A2A2A0A2A4A09F9E9DA
```

FIGURE 3.5: Coding scheme of a hex image file

for a 256 × 256 Lena image is shown in figure3.6. The hex-image file is shown in a graphical user interface (GUI) that was developed as a part of the project.


File Edit View Options Simulate Modelsim Window
ESRT LAB
Real-Time Simulator to Validate FPGA BASED Image Processing Circuits
Developed By ESRT Lab, NIT Rourkela
Edge Detection
Input
Hdl Output

```
A2A2A0A2A4A09F9E9DA09B9B9B9A9B9B9C
9DA2A4A5A9ADACABABA8A195927D6F615
C5D646669696B6D6A6A6D6B6B6D6C6E6D69
6B6F727377757A7B7A7F7E7D848383818287
858382828083828281868683858586858 6858
586868484868784848786888888787868787
858381838383828586848584838487858 6888
48481878C868280858385828083818680 7F82
8281828180807F817F7B7A797873716E666C7
57F898E95989CA0A1A19B959798989A9A9C
9B9E9D999A9999999979A9A989B9C9D9A9
```

```
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
00000000000000000000000FF000000000
0000000000000000000000000000000000
000000000000000000000000000FFFF000
```

Simulation Completed. Select Clear from Edit menu to start a new simulation

FIGURE 3.6: Hex image file generated for a 256 × 256 Lena image

3.6.2 VHDL Camera Module

This is a program written in VHDL. The programs were developed in XILINX ISE 14.2 and ModelSim 10.2c platforms. This program reads the input hex image file sequentially and then converts the hexadecimal values in to a standard logic vector data type of 8 bit length. Moreover, the program generates the clock pulse required for the operation of following FPGA processor and display driver.

Every 100ns the clock pulse will toggle its state. On the rising edge of the clock pulse the program will read a character from the input file. Program then checks for ',' and if encountered, it generates a horizontal synchronous pulse which is of 100ns in duration. If a '*' character is encountered, program will generate a vertical synchronous pulse signal which will be of 1 clock pulse duration. If the program encounters a valid hexadecimal character, it will be converted in to a standard logic signal of 4 bit length. The next character will be read from the file to produce lower 4 bits of the intensity value under process. A combination of two characters represents a pixel intensity level in the image. The detailed flow chart of the VHDL camera module is shown in figure 3.7.

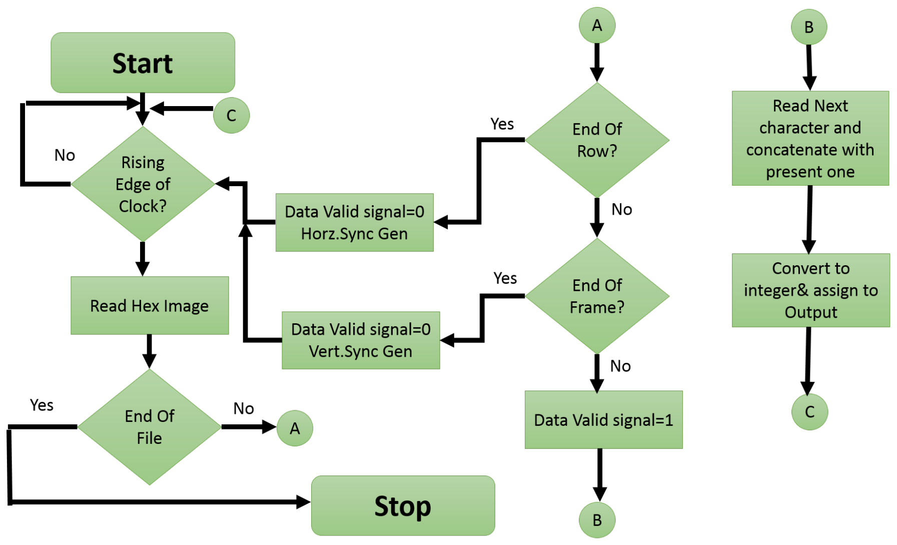

Start
C
No
Rising Edge of Clock?
Read Hex Image
Yes
End Of File
No
A
Stop
A
Yes
End Of Row?
Data Valid signal=0 Horz.Sync Gen
No
Yes
End Of Frame?
Data Valid signal=0 Vert.Sync Gen
No
Data Valid signal=1
B
B
Read Next character and concatenate with present one
Convert to integer& assign to Output
C

FIGURE 3.7: Algorithm to generate composite video signal from hex-image file

3.6.3 VHDL Display Module

This module does the reverse process of camera module. It accepts clock, data and synchronization signals as its input and generate hex mage file from them. On the rising edge of the clock pulse the program will check for synchronization pulses. Upon detection of them, they will be converted to either a ',' or a '*' symbol, depending on whether the vertical or horizontal synchronous pulse is high. The data signal which is in standard logic vector will be converted in to hexadecimal values and will be stored in hex image file. The algorithm for display module is a shown in figure 3.8

3.6.4 Test Bench Output Signals

The main output signals of VHDL camera module are:

- A clock pulse
- Horizontal synchronous pulse
- Vertical synchronous pulse
- Data valid signal

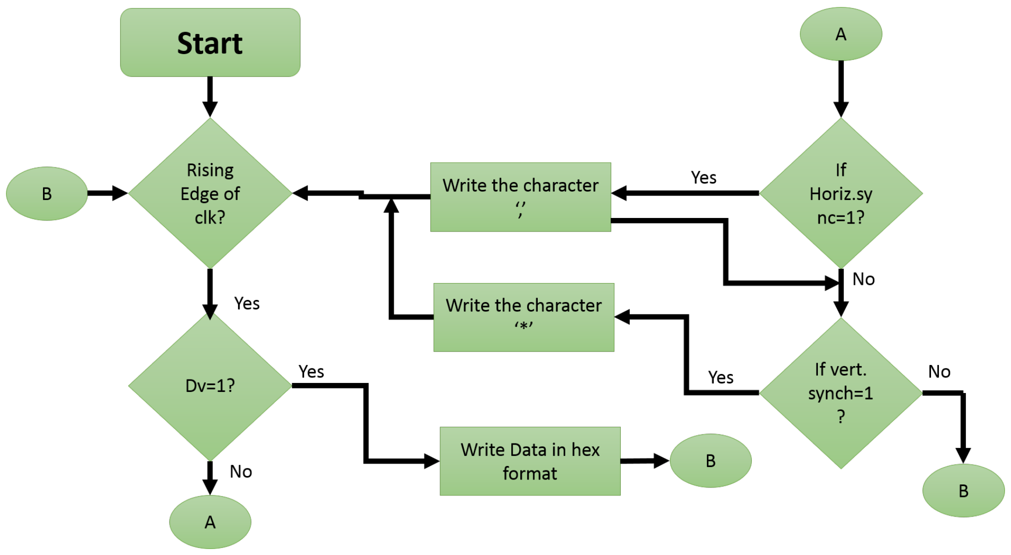
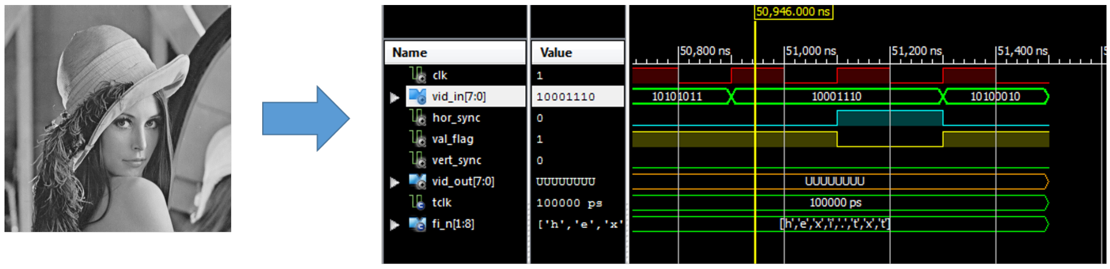


FIGURE 3.8: Algorithm to generate hex image file from composite video signal

The simulation results are shown in figure 3.9 . The simulation was carried out in XILINX ISE 14.2 platform.



Signal Variable	Signal Name
clk	Clock Pulse
vid_in	Data Signal
hor_sync	Horizontal Synchronous Pulse
val_flag	Data Valid Signal
vert_sync	Vertical Synchronous Pulse

FIGURE 3.9: Simulation of composite video signal using VHDL camera module

3.6.4.1 Clock Pulse

By default clock pulse is designed to toggle its state in every 100ns. As the application demands, this value can be modified to meet the specifications. All other events in camera, display and video processor module will be synchronized with this clock pulse. The signal shown in red color in the simulation results (Figure3.9) is the generated clock pulse.

3.6.4.2 Horizontal Synchronous Pulse

When the VHDL module detects an end of row event it will generate a horizontal synchronous pulse. This signal is shown is in blue color in figure 3.9. The total number of horizontal synchronous pulse will be equal to total number of rows in the frame.

3.6.4.3 Vertical Synchronous Pulse

This signal is shown in green color in figure 3.9. The signal will go high at the end of each frame. Total number of vertical synchronous pulses will be equal to total number of frames in one second. The system will be reset after the completion of each frame.

3.6.4.4 Data Signal

This is the video data signal. The data in standard logic vector format, will be sent to the video processor. The data is of 8 bit length 3.9. The display module will receive similar kind of data signal and will be converted in to hex format.

3.6.4.5 Data valid signal

This signal informs the following processor about the status of data signal. If this signal is low the processor will treat the incoming data signal as invalid. This signal will go low, if either of the horizontal or vertical synchronous pulses are high. Signal is shown in yellow color in the figure 3.9. As long as horizontal/vertical synchronous pulses are high, data valid signal will be low. Furthermore, this signal is used to indicate junk data from main processor.

3.7 Summary

In an FPGA based real-time design, a simulation platform is necessary for validation. Such a system provides an easy method to detect and debug the errors and to optimize the existing design. Hence a test-bench which simulates a real-time video acquisition system was developed and simulated. This test bench can be used for any general purpose video processing designs. The system parameters like clock speed should be modified for specific applications.

CHAPTER 4

Real-time Video Edge Detection System Design

4.1 Introduction

The necessity of a video processing test bench, its design and simulation were the topics covered in chapter 3. This chapter describes the design of a real time video processing edge detection system which make use of the test bench developed in chapter 3. The design and simulation was done for a single frame. This design can be extended to any number of frames.

The targeted system should acquire frames from an acquisition system and should detect edges in them. The system should be able to process 50 frames per second, which maps to a processing time of 0.025 seconds per frame. Moreover, the system will be designed for 480×640 resolution frames, but the experiments and result will be shown on a 256×256 frame. By the selection of appropriate clock speed, the above constraints can be met. As discussed in chapter3, the test bench will convert the incoming video in to a virtual composite video signals, and will feed the edge detection system. A general block diagram of the system is as shown in figure 4.1

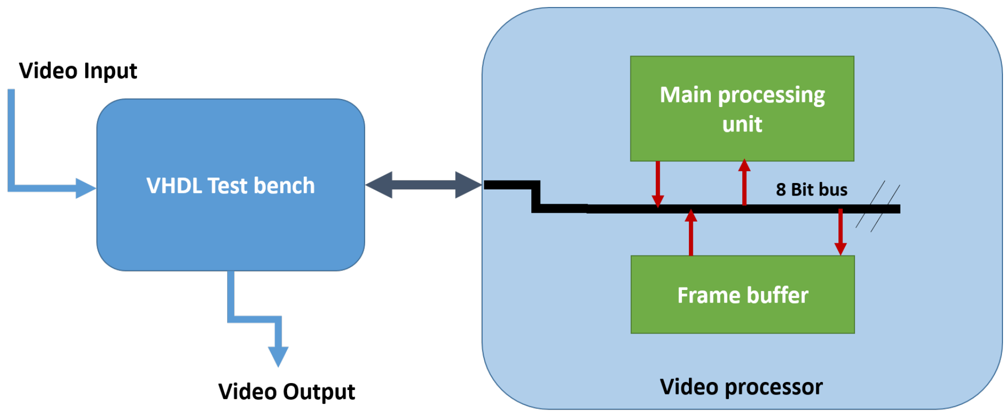


FIGURE 4.1: Block diagram of a video edge detection system

4.2 Edge Detection

From an image processing point of view an edge is a sudden change in intensity. As in the case of a one dimensional signal, a sudden change is detected as either a maxima or a minima. Hence this sudden change can be detected by computing a derivative image and equating each intensity level to a predefined threshold. As an image is a two dimensional signal derivative has to be calculated along x and y direction. The resulting derivative image is a vector, in the sense it has both magnitude and direction[6]. In order to get an approximation of the edge just magnitude will be sufficient.

$$\boxed{D = \sqrt{D_x^2 + D_y^2}} \tag{4.1}$$

$$\boxed{\alpha(x, y) = \arctan(\frac{D_x}{D_y})} \tag{4.2}$$

where
D= The derivative image
D_x = Derivative along X direction
D_y = Derivative along Y direction
$\alpha(x, y)$ = Phase image

The square root and squares are difficult to compute in an FPGA platform. Hence a better approximation of the equation 4.1 can be used as below [6]

$$\boxed{|D| = |D_x| + |D_y|} \tag{4.3}$$

Both equation 4.1 and equation 4.3 are monotonous, which means as the left part of the equations increases, the right part also increases monotonically. Hence the selection of equation 4.3 as an approximation to 4.1 can be justified.

Another edge detection approach is double derivative or lapalcian[6] method. Compared to derivative method, a double derivative is more sensitive to an edge. But this approach is highly porn to external noise and hence the accuracy of the output will be less. A lapacian method may produce a double edge. Thus this method is just used as a localization technique to identify the location of the edge.

More advanced and complex algorithms like canny edge detectors gives better edge approximations. Canny edge detectors has considerable amount of sequential operations. Thus it will utilize much more resources, when implemented in hardware platform. Due to complexity of the algorithm, the frame rate and system latency will be affected, but the accuracy will be improved.

4.2.1 Digital Approximations of Gradient

As an image is a two dimensional digital signal, a digital approximation for derivatives has to be used. There are several available digital approximations to derivative operation. Some are listed below

- Sobel mask [6]
- Roberts mask[6]
- Prewit mask[6]

Any of the above mask will be convoluted with original image to produce the derivative pixel. In this design a SOBEL mask will be considered.

4.2.2 Sobel Operator

A SOBEL mask [6] is given in figure 4.2. The mask is designed by taking the difference between adjacent pixels. The central pixel is given a boost by 2 to get smoothing effect which will reduce the noise. To implement SOBEL mask

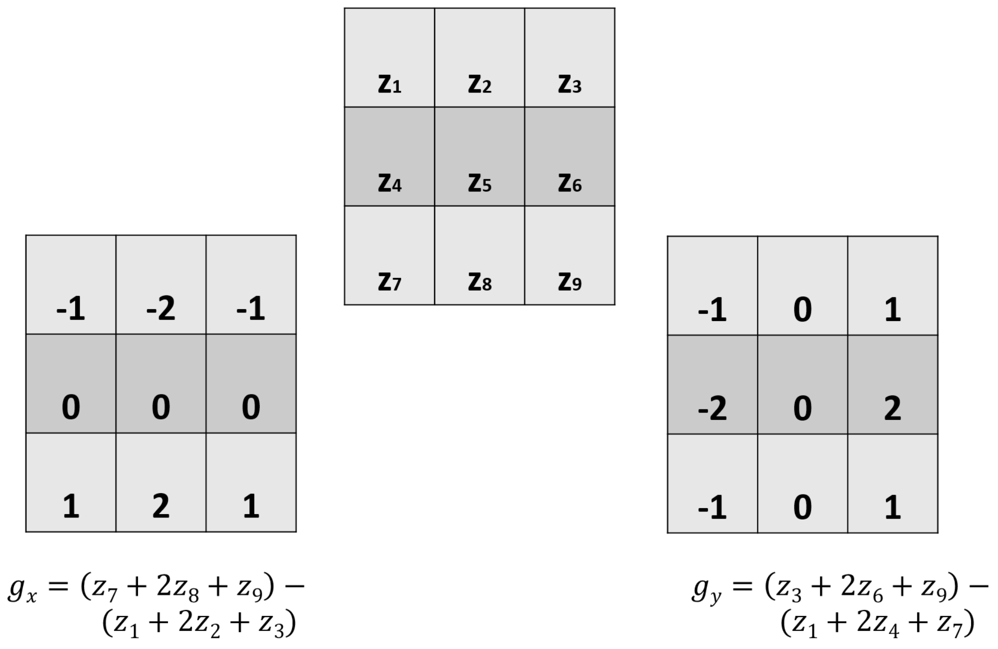

z1
z2
z3
z4
z5
z6
z7
z8
z9
-1 -2 -1
0 0 0
1 2 1
-1 0 1
-2 0 2
-1 0 1
$g_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$
$g_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$

FIGURE 4.2: A sobel mask [6]

in FPGA, the amount of parallelism in this algorithm has to be identified. From the diagram it is clear that the partial products are independent of each other and can be computed in parallel. Each partial product will be computed by a dedicated hardware and will be added in parallel. Both the D$_x$ and D$_y$ can be computed in parallel.

A careful observation of the mask reveals that there is no need of a dedicated multiplier to implement the mask. Instead a multiplication by two can be carried out by a simple left shift[**?**] and a multiplication by -1 can be carried out by a negation operation. Multiplication by a -2 shall be implemented by first negating the data and then left shifting it. Hence the mask can be implemented efficiently in a VHDL platform.

4.3 Architecture Selection/Parallel Algorithm Development

4.3.1 FPGA System Architecture Selection

Before going in to the design of an FPGA based circuit, different system performance parameters has to be estimated. The first and foremost step is to select the FPGA system architecture. A SOBEL based edge detection algorithm has considerable amount of parallelism in it. The entire masking operations along 'x' and 'y' directions can be implemented in parallel. Thus to effectively utilize this parallelism, an architecture which is completely parallel in nature will be selected. A standalone FPGA architecture is the best choice for this design, as it is completely parallel. An in-depth description of FPGA system architectures can be obtained from chapter2.

4.3.2 FPGA Computational Architecture Selection

In the next step FPGA memory (computational) architectures has to be selected. This design involves a buffers at both input and output, to store the incoming frames and derivative frames respectively. Hence the incoming pixels can not be processed on the fly. The best method to access data from a buffer is random access processing. At the same time, the path between input and out buffers are continuous. There is no need to intermittently store the data. Hence, stream processing can be utilized in this section. Thus in the whole design, a combination of stream and random access processing were used. Detailed descriptions about FPGA computational architectures can be obtained in chapter2.

4.3.3 Selection of FPGA Mapping Techniques

Next stage is to identify the FPGA mapping techniques that can be used to achieve desired system performance . In this design, pipe-lining and caching has been used. A pip-line is implemented to accelerate the processing of the pixels. The SOBEL mask is implemented in a pipe-line. The pipe-line has three stages. The derivative pixel will be stored to the output cache memory in the third stage. While the current pixel is being stored to output cache, the derivative of the next pixel will be calculated simultaneously in the second stage of the pipe. At the same time, a third pixel will be fetched in first stage of the pipe. This helps the system to process to more than one pixel in one clock cycle.

4.3.3.1 Selection of Cache Memory

Caching is implemented to improve memory band width and system latency. Here a four pixel wide cache is set-up to avoid frequent reading of data from main memory. Basically to find out the derivative of a single pixel, nine neighbouring pixels are needed (3 rows of 3 pixels). But normally the size of cache memory is defined in powers of two. Hence a 4 pixel wide cache memory is setup to store one row. Three such memories are designed to store three neighbouring rows. In particular caching is implemented by row buffering technique. Detailed descriptions about FPGA mapping techniques are given in chapter 2

4.4 Clock Speed and Data Bus Width

4.4.1 Selection of Clock Speed

The next step in the design is to identify the clock speed [3] and data bus width necessary for optimum performance. Clock speed defines the total system performance in terms of speed and the data bus width defines the total system band width.
Suppose the incoming frames are 256 $\times$ 256 resolution . If the system has to process 50 frames per second then the clock speed will be calculated as

$$\boxed{ClockSpeed = 256 \times 256 \times 50Hz \approx 3MHz} \tag{4.4}$$

The clock speed selected for this design is 10MHz.
The above calculation is done with an assumption that only one pixel will be processed in a single clock cycle. But a pipe-lined architecture process more than one pixel per clock cycle. Thus the clock speed can be reduced by a factor, which is proportional to the number of pixels proceed in one clock cycle. This

information is not directly accessible. Because in one clock cycle, none of the pixels are completely processed. A part of the the processing will be done in a clock cycle, but multiple pixels will be processed. Moreover, additional clock cycles, which will be required to flush the pipe-line at the end of each row, to begin the processing of new row will have to be considered. Another factor which influence the selection of clock speed is the operating speed of output device. In the simulation environment, the output is written in to a file, and if the file writing operation is slow, the data will have to wait in a buffer, which will increase the system latency in simulation environment.

4.4.2 Selection of Data Bus Width

To calculate the minimum data bus width we have to estimate the maximum size of the partial products. Normally image intensity values will be stored as eight bit. The partial product operation include a multiplication by 2 and then an addition operation. Hence maximum integer value that can be produced in partial product is $(255 \times 2 + 255 + 255 = 1020)$ which need 10 bits to be represented [3]. If we include another bit for sign, the intermittent data bus width should be of atleast 11 bit.

4.5 Over all architecture

The over all architecture of the edge detection system is as shown in figure 4.3. The entire circuit can be divided in to four parts. The buffering part, indexing part, caching part and pipeline. The data from the test bench will be buffered in the internal frame buffer, with one pixel in every clock pulse. This data will be then routed to cache memory through appropriate control signals.
To calculate a derivative pixel, eight neighbouring pixels has to be fetched from buffer. If eight read operation is performed for each result pixel, total system bandwidth utilization will be inefficient. Hence to compensate that, four pixels will be fetched at a time from the buffer and will be stored in a 4 byte cache. Four pixels of previous row, present row and next row will be buffered in to 3 different cache memories, with each memory has a size of 4 bytes.
When the first row of buffer is full the test bench will generate a horizontal synchronous pulse and this will enable the data path between buffer and the cache

memory corresponding to present row (present row cache). In a similar way data will flow from buffer to the cache memories corresponding to current and next rows. The path from pipeline to the cache memory to hold the result pixels will not be enabled unless the input row cache memories are full, to avoid junk data at the result cache. When the result cache is full the 4 bytes will be written to result frame buffer and from there data will be sent to VHDL display module of the test bench. The sequence of events involved in data flow is shown in figure 4.4. A red line indicates enabled path and a green line indicates disabled path.

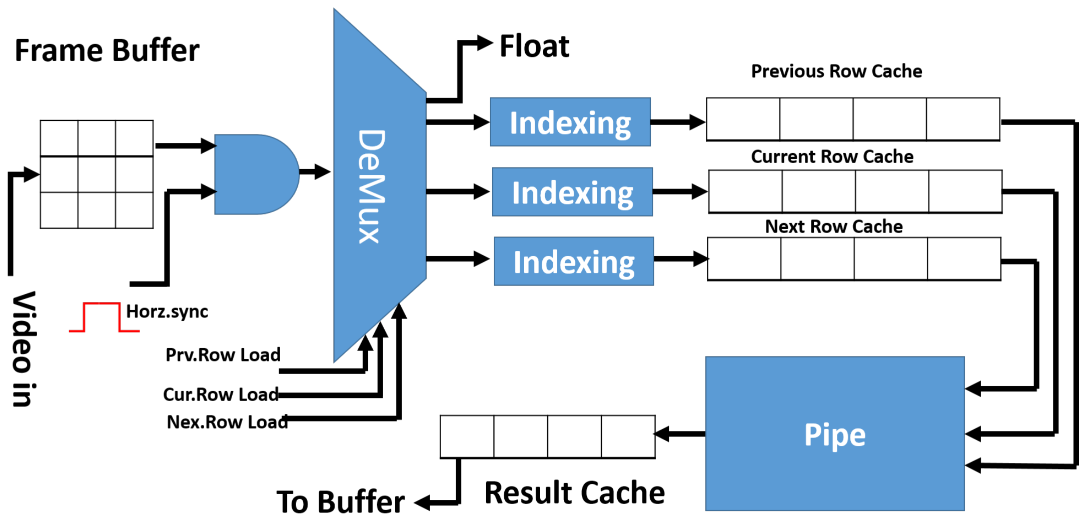


FIGURE 4.3: Over all edge detection system architecture

4.5.1 Buffering Section

The data flow in to the system begins from this block. The system need to access eight neighbouring rows to compute the derivative of a single pixel. Thus pixels can not be processed on the fly as the system has to wait for dependent pixels. Since the buffer is being filled sequentially, the minimum wait time for system will be the time till first three rows were fetched from test-bench. once the third horizontal synchronous pulse is received from the test bench,the data flow to the system will be begun .The fetching of other rows from the test bench will be continued, while the the first three rows were being processed by the SOBEL processor. The the buffered data will be in standard logic format. For further processing, this will be converted to unsigned format, because of the arithmetic

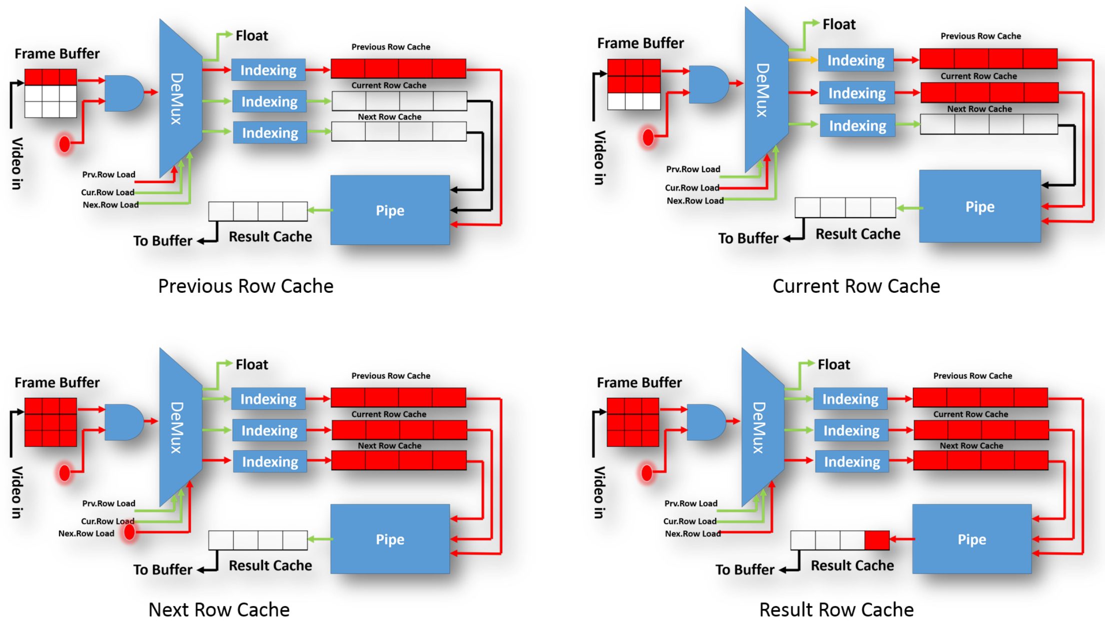


FIGURE 4.4: Over all edge detection system data flow sequences

operations inside the SOBEL processor. The fetching of data will be continued till a vertical synchronous pulse is received from the test-bench.

4.5.2 Caching Section

The data fetched from the frame buffer will be stored in a group of three row cache [3]. Each cache memory can hold four pixels of data. The pixels from three consecutive rows will be fetched in to this cache. Cache memory accelerates the process by reducing the number of access to main memory. The data from cache will be then shifted serially in the pipe line for further processing. Each pixel in a frame is eight bit wide, providing an overall size of 32 bytes for each cache memory. A detailed descriptions of row buffering and caching are given in chapter2

4.5.3 Indexing Section

This section routes the data from frame buffer to respective cache memory. In an FPGA board, the acquired frames will be stored in the main memory. The embedded processor will give the starting address of the memory locations at which the frames are stored. In order to avoid frequent access to main memory, four pixels will be fetched at a time. The four pixels corresponding to previous, present and next rows will be fetched in parallel. A counter will be incremented by four, up on fetching of every four pixels from the frame buffer. Hence to obtain next four pixels from previous row, the counter value will be added to the base address. The same theory is applicable for current and next row pixels. To obtain the next four pixels corresponding to the current row, the counter value, base address and total number of columns will be added together. Similarly for the next row pixels, the base address, counter value and the two times the number of columns will be added together to get the starting index of next four pixels. The operations are summarized below.

$$\begin{aligned} PreviousRowAddress &= BaseIndex + CounterValue \\ PreviousRowAddress &= BaseIndex + CounterValue \\ NextRowIndex &= BaseIndex + CounterValue + 2 \times Numberofcolumns \\ Counter &= Counter + 1 \end{aligned} \tag{4.5}$$

This way the indexing circuit maintains the continuity between the pixels being processed by the SOBEL processor and the pixels in the frame buffer. The conceptual diagram of the indexing circuit is shown in figure 4.5. The sequences of

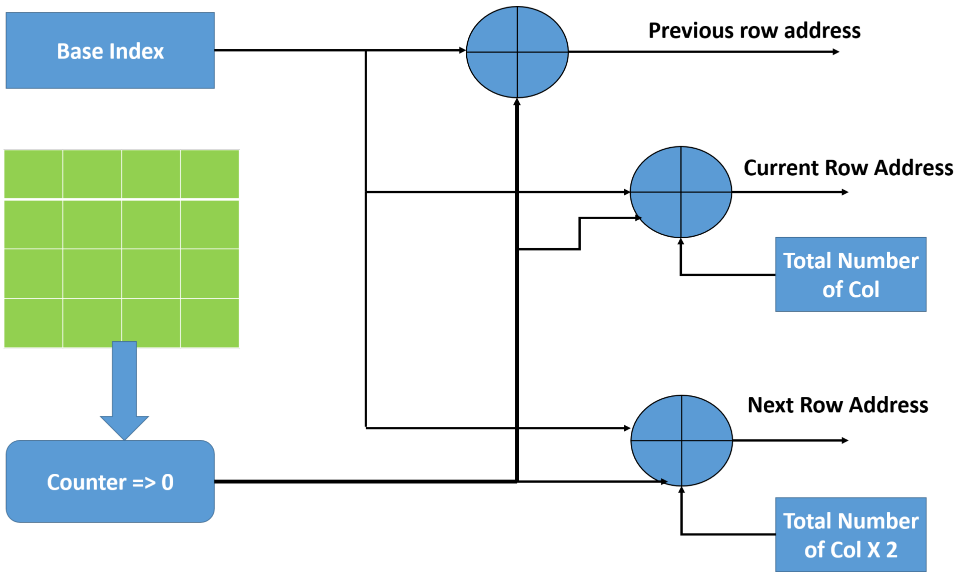


FIGURE 4.5: Indexing Circuit

operations in indexing blocks are given in figure4.6

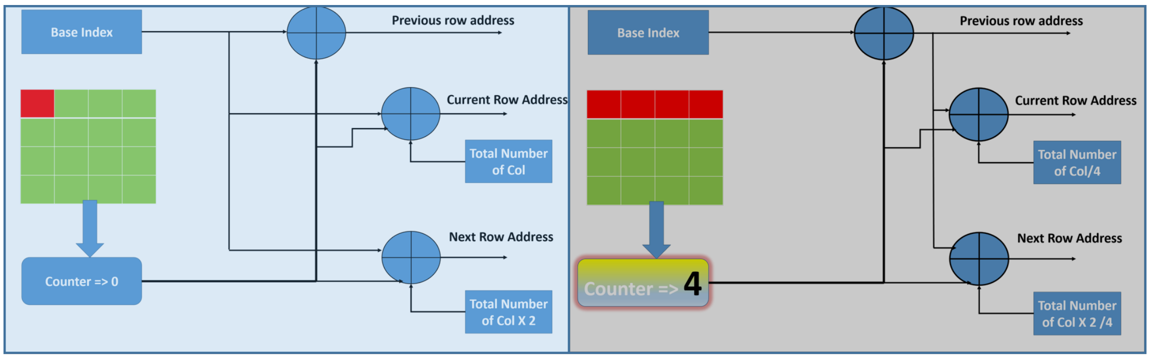


FIGURE 4.6: Sequence of operations happening at indexing module

4.5.4 The Pip-line

The SOBEL mask is implemented in the pipe-line[1] section. Pipe-line, generally called as pipe has 3 $\times$ 3 registers, with each register storing three pixels of previous, current and next row. Total size of internal registers in the pipeline is 12 bytes. The SOBEL masks along 'x' and 'x' directions are implemented as shown in figure 4.7 [3]. The derivatives along 'x' and 'y' directions will be computed in parallel with in the pipe. The pixels from corresponding row cache will be serially shifted in to the pipe's internal registers. Here the coefficients of SOBEL masks will be stored in another register banks. The sums obtained by convolving along 'x' and 'y' directions will be computed in parallel and will be stored in D$_x$ and D$_y$ registers. In the next clock cycle the modulus value of this pixels will be calculated. This values will be stored in the $|D|$ register. This value will be written to result cache in the next clock cycle. While the modulus of current pixel is being calculated, the D$_X$ and D$_Y$ sum of the next pixel will be processed. Also while the current pixel is being stored to the result cache, the modulus value of the next pixel will be computed. Thus the pipe has 3 stages. In first stage the D$_X$ and D$_Y$ will be calculated, modulus will be calculated in the second stage and pixels will be stored in to result cache in the third stage. The overall pipeline architecture is shown in figure 4.7. The sequence of data flow in the pipe-line is shown in the below figure 4.8. The red square blocks indicate the pixel, and the red arrow indicate an enabled path. At the end of each row of input pixels, the pipeline will be flushed. This is being done to avoid junk data. After the flushing operation, the data path to the output cache

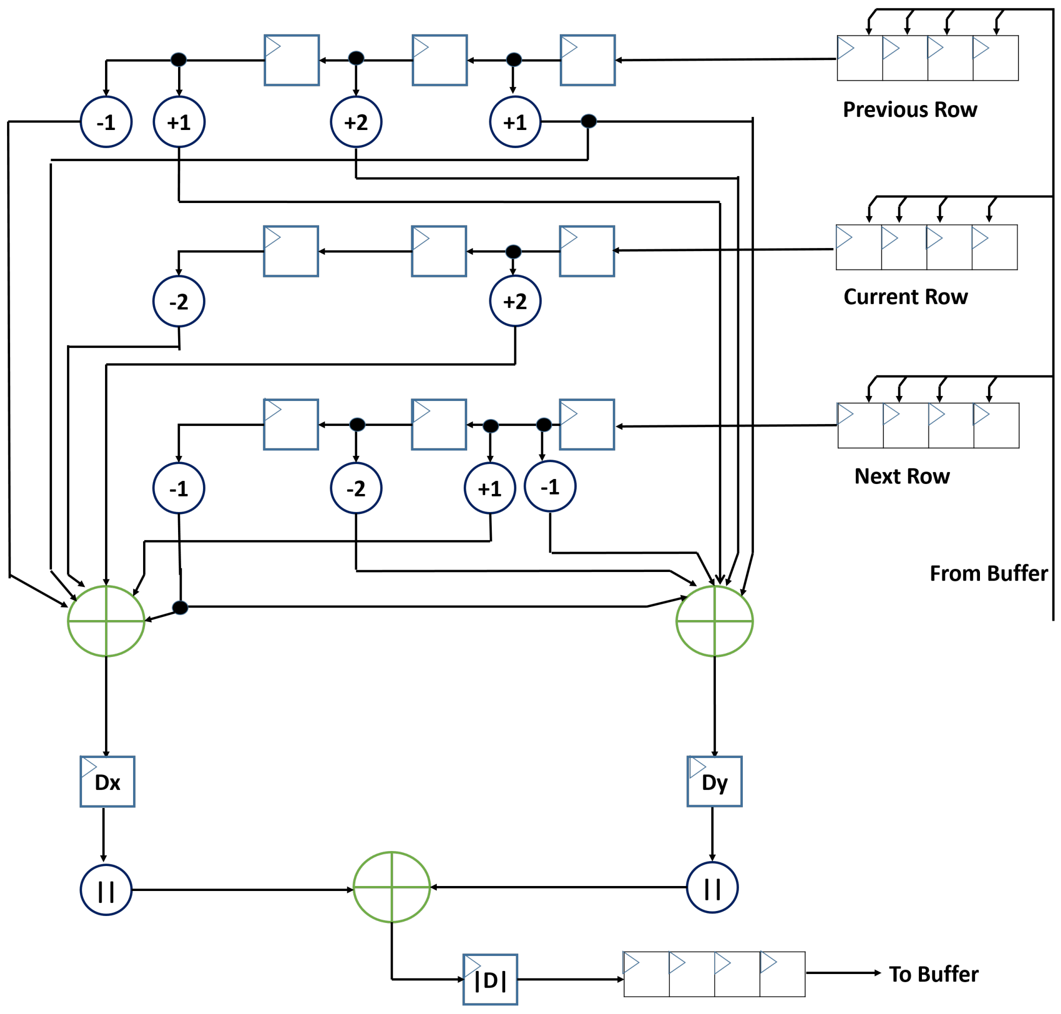


FIGURE 4.7: Pipe-line Circuit

memory will be deactivated for 3 clock pulses. During these 3 clock pulses, the data from new row will be shifted in to the pipeline. After 3 clock pulses the pipe line will be full with new pixel data, and the path to output cache memory will be enabled.

The pipeline will be flushed at the end of frame as well. The result row will be written in to an output frame buffer. Four pixels in the result cache will be directed in to the output frame buffer using indexing circuits. The operation of indexing circuit at the output stage is similar to that of the input stage.

The derivative values from the output buffer will be fed to display module of the test-bench. In this module, the output composite signal will be converted in to hex-image file.

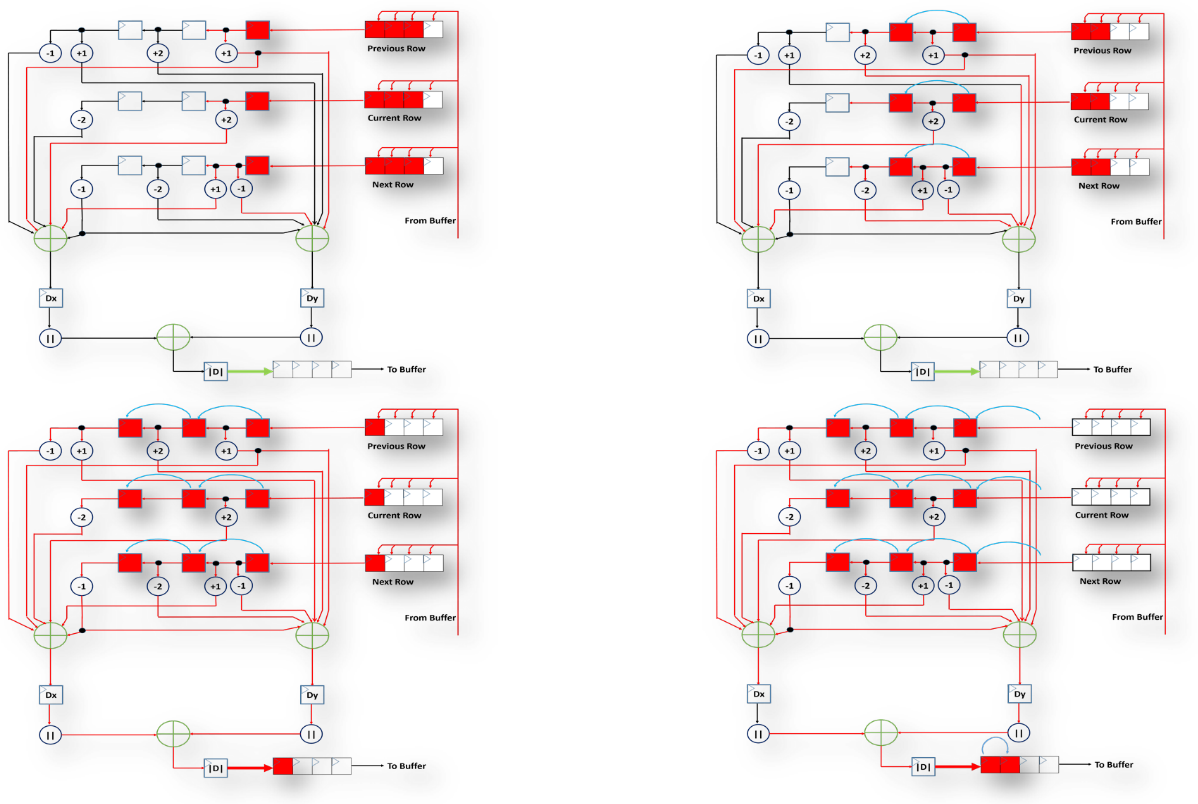
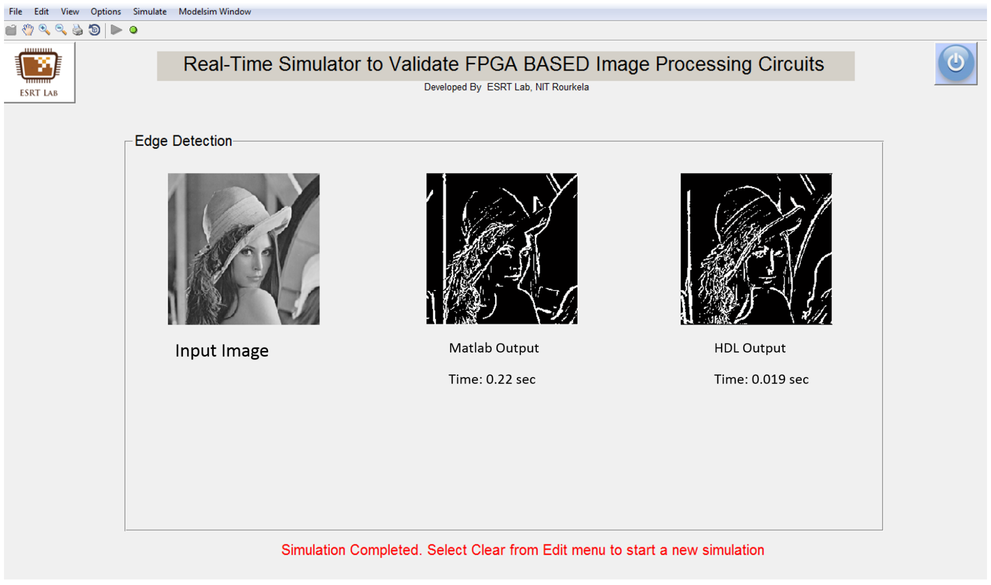


FIGURE 4.8: Pipe-line Data flow

4.6 Simulation Results

The overall architecture explained above is simulated in Modelsim 10.2c student edition. A new software patform was developed to integrate test-bench and VHDL simulation under a common platform. The new simulator developed is named as realsim. The front end of the simulation was done in this software platform. A detailed description of the realsim software is given in chapter5 and appendix i. The figure4.9 shows the key simulation results in realsim. The figure4.10 shows the key simulation results in Modelsim. The figure shows the binary wave forms of input and output images.
The description of key results are given below.

The developed hardware circuit took 0.019 seconds to detect the edges of a 256 × 256 frame at the clock speed of 10 MHz. The same simulation over a Matlab environment took 0.25 seconds. Thus the hardware design is 13 times faster than the traditional sequential environment. Considering the fact that, the hardware simulations were carried out in ModelSim student edition, which is 100 times slower than the business version[15], the actual hardware implementation will be several 10 times faster than the currently obtained result. The system latency achieved is 0.0065 seconds.

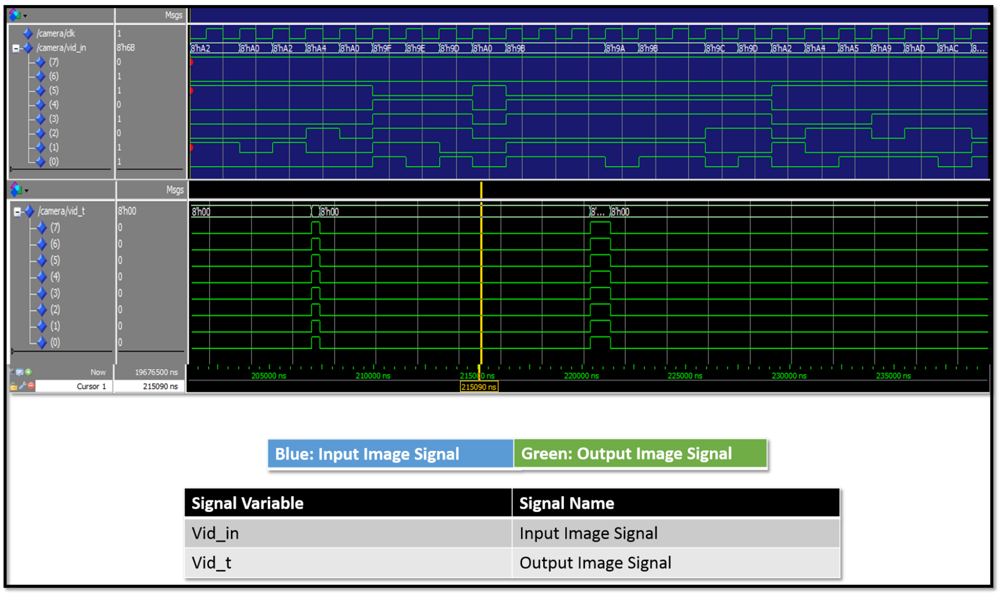

File Edit View Options Simulate Modelsim Window
ESRT LAB
Real-Time Simulator to Validate FPGA BASED Image Processing Circuits
Developed By ESRT Lab, NIT Rourkela
Edge Detection
Input Image
Matlab Output
Time: 0.22 sec
HDL Output
Time: 0.019 sec

Simulation Completed. Select Clear from Edit menu to start a new simulation

FIGURE 4.9: Simulation of FPGA based SOBEL edge detection architecture in realsim

4.7 Summary

In this chapter a real-time image edge detection architecture has been developed. The required system parameters has been estimated and the method of implementation was discussed in preceding sections. From the result obtained, it can be inferred that the parallel implementation of the image processing algorithms will be several ten times faster than the traditional sequential designs.



Signal Variable	Signal Name
Vid_in	Input Image Signal
Vid_t	Output Image Signal

FIGURE 4.10: Simulation of FPGA based SOBEL edge detection architecture

The improvement in speed comes with additional effort, cost and complexity associated with hardware designs. In real-time applications, FPGA based image processing designs has wider scope and significance.

CHAPTER 5

Software Platform to Validate FPGA Based Real-Time Applications

5.1 Introduction

Testing is an inevitable part of any project. In a hardware design project, the simulation should be carried out in software platform, and its functionality should be tested against the objectives. It is a thump rule that if a design does not work in simulation environment, it will not work in hardware platforms as well. Thus each of the hardware module should be separately tested against its functionality and performance measurements. Apart from unit testing, integration testing of the hardware modules are also important as the stealth errors due to interaction between different modules will be manifested in this phase. From an image processing point of view, to carry out unit and integration testing, input and output should be visualized as an image, rather than binary waves. This idea led to the development of anew generic software platform for FPGA based image processing applications.

5.2 Motivation for New Simulation Environment

Traditional HDL simulators like Modelsim, Xilinx ISE etc. shows the simulation results in the form of binary waves. If the circuit complexity is less, it will not be much difficult to decipher the binary waves. Most of the image processing operations are complex in nature and includes a large number of signals. Hence it will be extremely difficult to interpret the outcomes of the image processing operations from binary wave forms. This idea was the key motivation behind the development of new simulation platform.

The simulation of any FPGA based image processing algorithm will have three basic entities.

- A test bench to simulate an image acquisition system
- An image processor
- Test-bench to simulate display device

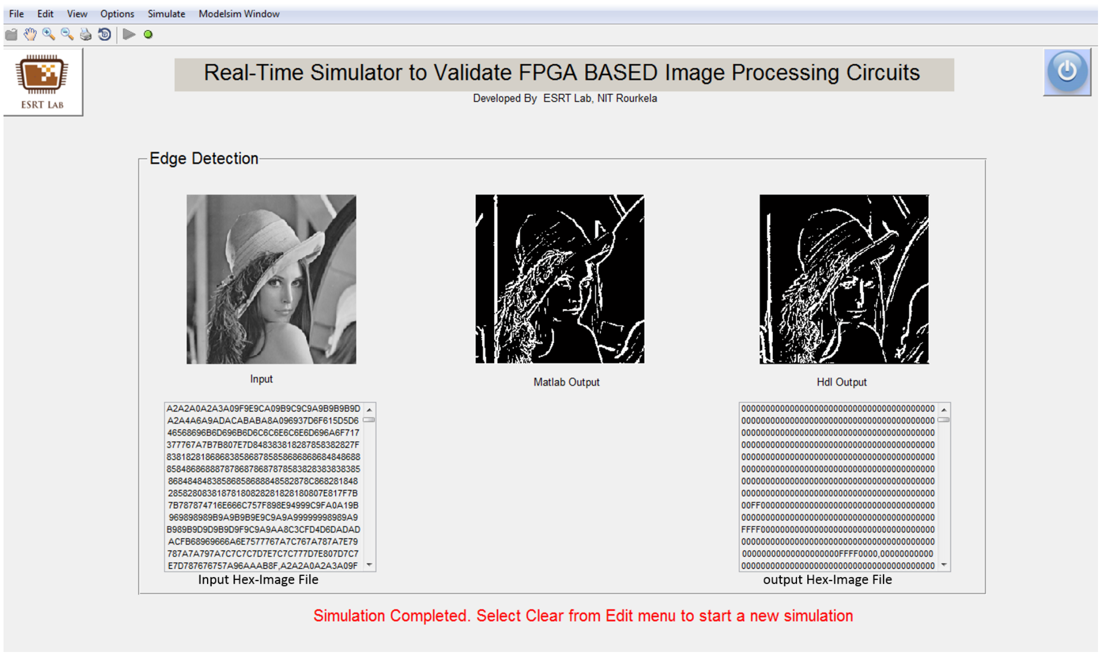
Effective integration of all the three entities is essential to make a design neat and reproducible. The new simulator named as realsim, integrates all the above entities, and provides advanced visual perception compared to traditional HDL simulators.

5.3 Features of Realsim

Realsim let the developer to provide the input to the system under test, in the form of an image. Furthermore, the output of the system will be reconstructed from binary waves in to image format. The front end of the software is developed in Matlab-2012b, and the back end is designed in Modelsim-10.2c student edition. The interface between Matlab and Modelsim is done using microsoft dos shell and tcl scripts.
The main features of realsim are given below. A detailed tutorial of realsim is given in AppendixA.

5.3.1 Test Bench

The test-bench of any FPGA based image processing operation consists of an acquisition and display modules, as explained in chapter3. Realsim will integrate both the display and acquisition modules under a common platform. The simulator allows users to visualize the input and output images along with the corresponding hex-image files (Chapter3). The figure5.1 shows the input, simulation result and the test bench in a single screen.



```
A2A2A0A2A3A09F9E9CA09B9C9C9A9B9B9B9D
A2A4A6A9ADACABABA8A096937D6F615D5D6
46568696B6D696B6D6C6C6E6C6E6D696A6F717
377767A7B7B807E7D848383818287858382827F
83818281868683858687858586868686848486688
858486868887878687868787858382838383838385
868484848385868586888485828 78C868281848
2858280838187818082828182818080 7E817F7B
7B787874716E666C757F898E94999C9FA0A19B
969898989B9A9B9B9E9C9A9A99999998989A9
B989B9D9D9B9D9F9C9A9AA8C3CFD4D6DADAD
ACFB68969666A6E7577767A7C767A787A7E79
787A7A797A7C7C7C7D7E7C7C777D7E807D7C7
E7D787676757A96AAAB8F,A2A2A0A2A3A09F
```

```
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
00FF000000000000000000000000000000000000
0000000000000000000000000000000000000000
FFFF000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000FFFF0000,00000000000
0000000000000000000000000000000000000000
```

Simulation Completed. Select Clear from Edit menu to start a new simulation

FIGURE 5.1: View of input, output and test bench in a single screen

5.3.2 Improved Visual Perception

The simulator allow users to give input in the form of image and the output will be generated in the same format. All the background processing will be carried out in HDL. The simulator displays the background processing in a status bar. Detailed tutorial of the simulator is available in AppendixA. Realsim provides three different perceptions of input and system output.

- As an image in the image window(figure5.1)
- As a hex-image file (figure5.1)
- As binary waves

The three views helps to understand the design and underlying concepts. Moreover, it makes the back tracing and debugging of the design easier.

5.3.3 Integrated Waveform Analysis

Realsim shows the simulation results in the form of images. In order interpret the output in detail, ie. in terms of simulation events and deltas, the waveform view can be selected. In this window, the output will be shown in binary wave form.

5.3.4 Hardware Simulation Reports

Realsim provides detailed report of hardware simulation. The report will include simulation events in each simulation deltas. The reports can be made use to debug the design in case of any errors. Moreover, realsim provides a detailed report on input and output hex image-files.

5.4 Conclusion and Future Scope

In a nutshell, realsim provides improved visual perception and added features compared to traditional HDL simulators. Realsim can be extended to any FPGA based real-time image processing applications, like face detection, object tracking etc. Furthermore, the simulator can be enhanced by adding facilities to incorporate real-time video inputs. In future this simulator may serve as a standard platform to validate any FPGA based real-time designs.

CHAPTER 6

Conclusion and Future Scope

6.1 Conclusion

The results obtained in this work emphasize the significance FPGA based designs in real-time image processing applications. In the current scenario, most of the real-time applications are implemented either in micro-controllers or dsp processors. For example, most of the digital cameras in the market has a micro-controller based pre-processing circuits. Further more, in the world of portable devices and gadgets, it is extremely important to have less power consumption for prolonged battery life. From an image processing perspective, speed of execution is a direct measurement of power consumption. To achieve minimum power consumption, the sequences should be executed in minimum number of clock cycles. This is precisely what an FPGA based design achieves. Hence it is the need of the era, to focus more on FPGA based image processing designs.

The improvements in speed and performance comes with additional cost and effort in hardware designs. The hardware design need much more expertise and resources compared to a traditional sequential design. The final target of any FPGA based design is an ASIC (Application specific integrated circuit). The process of converting an FPGA based design to an ASIC is time consuming and costly. This additional effort and costs are the factors which slows down the growth of FPGA based design in current industry (Especially Industries in

India). This work could serve as a motivation and a gate way to the enticing world of research in FPGA based computer vision.

The central highlight of the work is the design of a simulator to validate any FPGA based real-time designs. Traditional VHDL simulators like Modelsim, Xilinx ISE etc do not provide a platform in which the output of image processing operations can be visualized. The realsim converts the output of the main processing module from binary wave forms to image format for improved visual perception. The simulator also provides advanced features like integration of test-benches, simulation reports etc. Concisely, this simulator is a general purpose platform for the validation and simulation of any FPGA based real-time image processing designs. The immediate enhancement to the simulator will be the incorporation of the video processing facilities. This will let the developer to give input in the form of real-time video, and see the output. Addition of this facility will be a revolutionary change in the field of FPGA based image processing. This simulator will then serve as the basic tool for FPGA implementation of any computer vision applications.

6.2 Applications and Future Scope

Any FPGA based real-time designs, that will be developed in future can be integrated in to realsim. An immediate enhancement to this simulator will be the addition of video processing features. Furthermore, other edge detection algorithms like lapacian of gaussian, canny edge detectors etc. will be added to the edge detection feature of the simulator. An option to compare the performance of similar image processing techniques will help the designer to choose the best implementation strategy and will make the simulator more professional. In-addition, the current design can be extended to implement more complicated algorithms like viola-jones face detection algorithm, optical flow etc. These implementations has significant amout of sequential operations in the algorithms. Special strategy should be employed to mitigate the effects of communication overhead.

The work presented here has wider scope and applications. The edge detection techniques are basic steps of several complex computer vision applications.

The implementation of fast face detection in a digital camera, fast object tracking, policing and interactive surveillance, and finally the applications like object tracking and chasing are some of the areas where this work can be made use of. More importantly, this work will serve as a motivation to explore the endless scopes of FPGA based computer vision designs.

APPENDIX A

Realsim 1.0 Tutorial

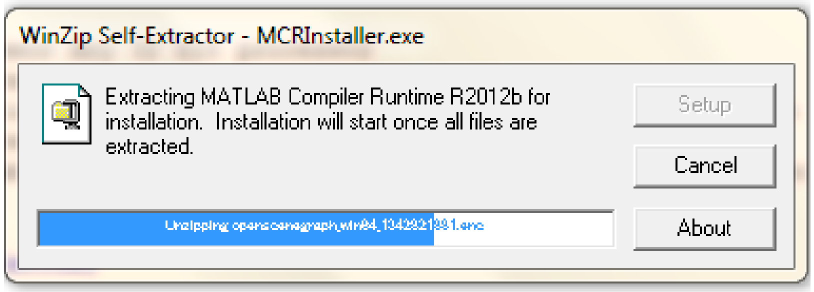
The procedure to carry out simulation in realsim 1.0 is presented in this appendix. Realsim 1.0 is a comprehensive simulation and validation environment for FPGA based real-time image processing designs. This software platform was developed as a part of the project "Development of FPGA based Image processing IP core ", at embedded system and real-time laboratory, National Institute of Technology Rourkela . The objective was to integrate the different modules of the project and to simulate and validate the FPGA based designs. The software was packed in to a single executable file to run in windows environments. The tutorial on how to use reasim is explained in the following sections, starting from the installation.

A.1 System Requirements

- Processor : Any 32 bit processor
- Physical Memory : Minimum 512 MB. 2 GB recommended
- Operating system : Windows XP, Windows 7, Windows 8 32 bit
- Required soft-wares : Modelsim 10.2c or higher
- Matlab run-time environment (Comes along with the package)

A.2 Installation

- Run install_pkg.exe
- This will extract matlab-run time environment, and will install it. It will generate a file named realsim.exe. theis is shown in figureA.1


WinZip Self-Extractor - MCRInstaller.exe

Extracting MATLAB Compiler Runtime R2012b for installation. Installation will start once all files are extracted.

Unzipping openscenegraph_win64_1342921281.exe

FIGURE A.1: Installation of realsim

- realsim.exe and associated files will be extracted in to the current directory, which can be copied to any other directories.

A.3 Over View of Graphical User Interface

- Run realsim.exe. The opening screen is shown in figureA.2.
- The top right corner has the shut down button, which will be used to exit the program. The different tools of the software are labeled in the figureA.3
- The input and outputs will be displayed in the process window.
- The status bar will update the status of background processes.

A.4 Simulation

- Select the test image using either the menu or the open icon as shown in figureA.4. Direct the path to the test image. The image will be opened in the process window as shown in figureA.5.

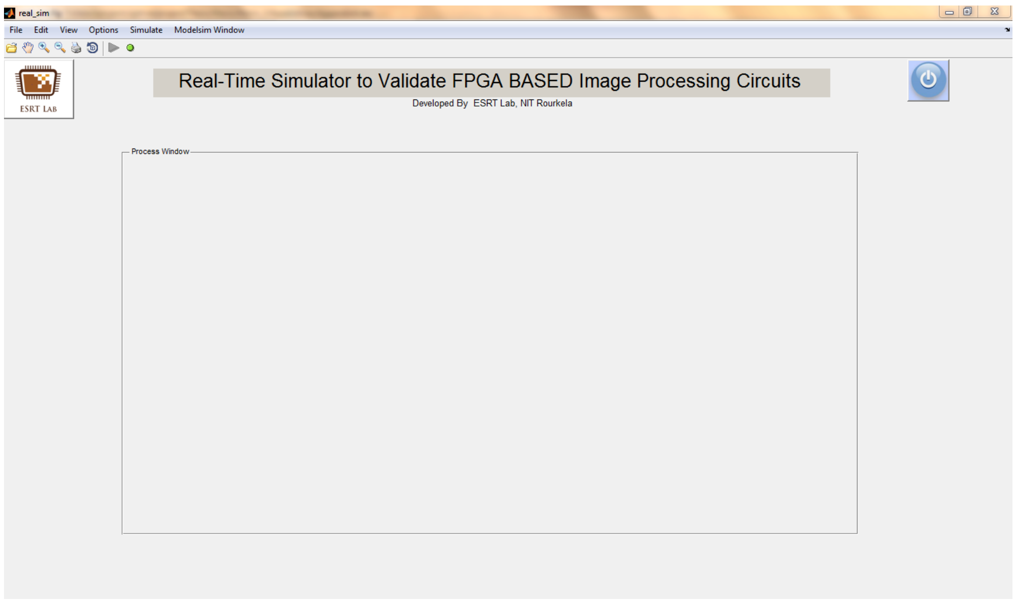

real_sim
File Edit View Options Simulate Modelsim Window
ESRT LAB
Real-Time Simulator to Validate FPGA BASED Image Processing Circuits
Developed By ESRT Lab, NIT Rourkela
Process Window

FIGURE A.2: starting screen of realsim

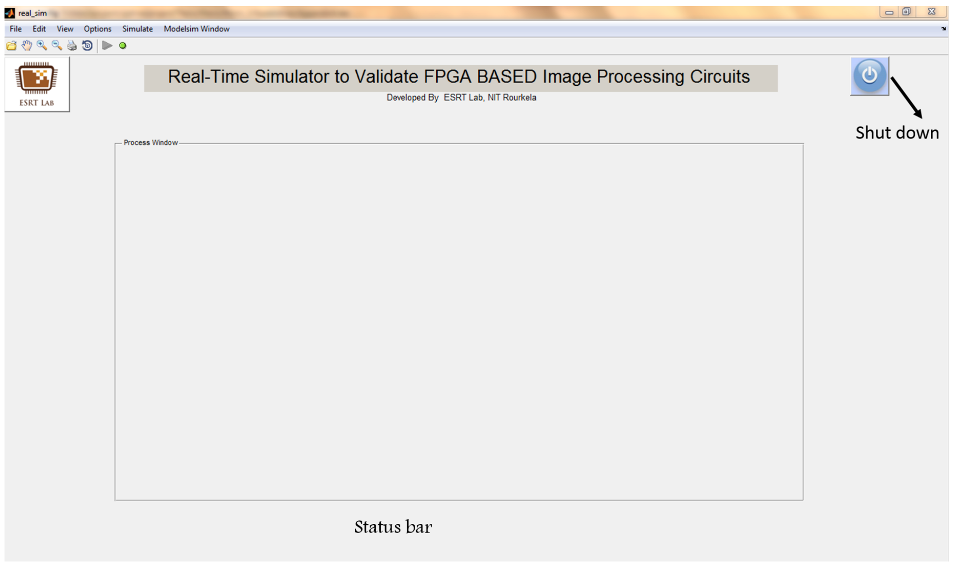

real_sim
File Edit View Options Simulate Modelsim Window
ESRT LAB
Real-Time Simulator to Validate FPGA BASED Image Processing Circuits
Developed By ESRT Lab, NIT Rourkela
Shut down
Process Window
Status bar

FIGURE A.3: The tools and windows are labeled

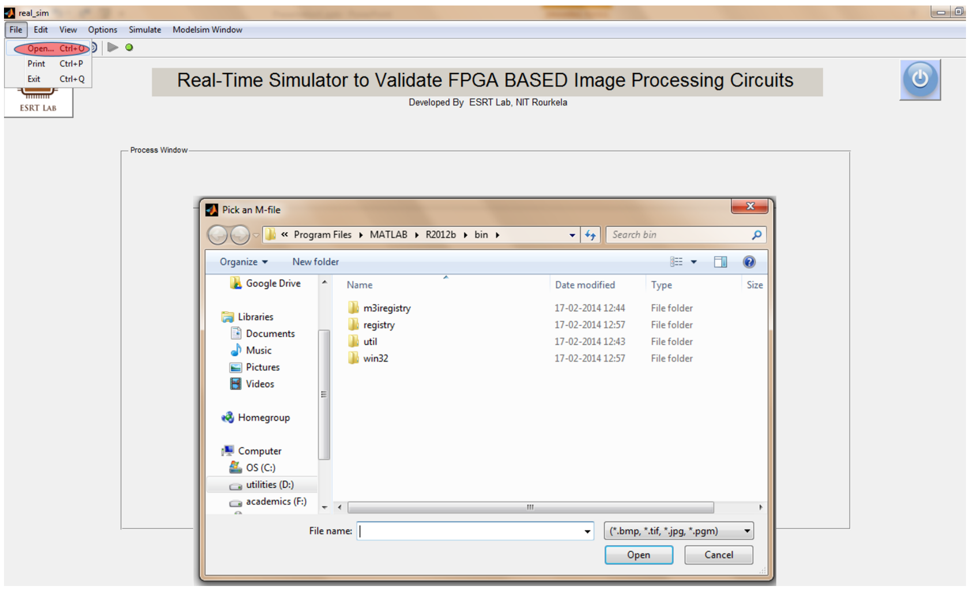

real_sim
File Edit View Options Simulate Modelsim Window
Open... Ctrl+O
Print Ctrl+P
Exit Ctrl+Q
ESRT LAB
Real-Time Simulator to Validate FPGA BASED Image Processing Circuits
Developed By ESRT Lab, NIT Rourkela
Process Window
Pick an M-file
« Program Files ▸ MATLAB ▸ R2012b ▸ bin ▸
Search bin
Organize New folder
Google Drive
Libraries
Documents
Music
Pictures
Videos
Homegroup
Computer
OS (C:)
utilities (D:)
academics (F:)
Name Date modified Type Size
m3iregistry 17-02-2014 12:44 File folder
registry 17-02-2014 12:57 File folder
util 17-02-2014 12:43 File folder
win32 17-02-2014 12:57 File folder
File name:
(*.bmp, *.tif, *.jpg, *.pgm)
Open
Cancel

FIGURE A.4: Select the test image

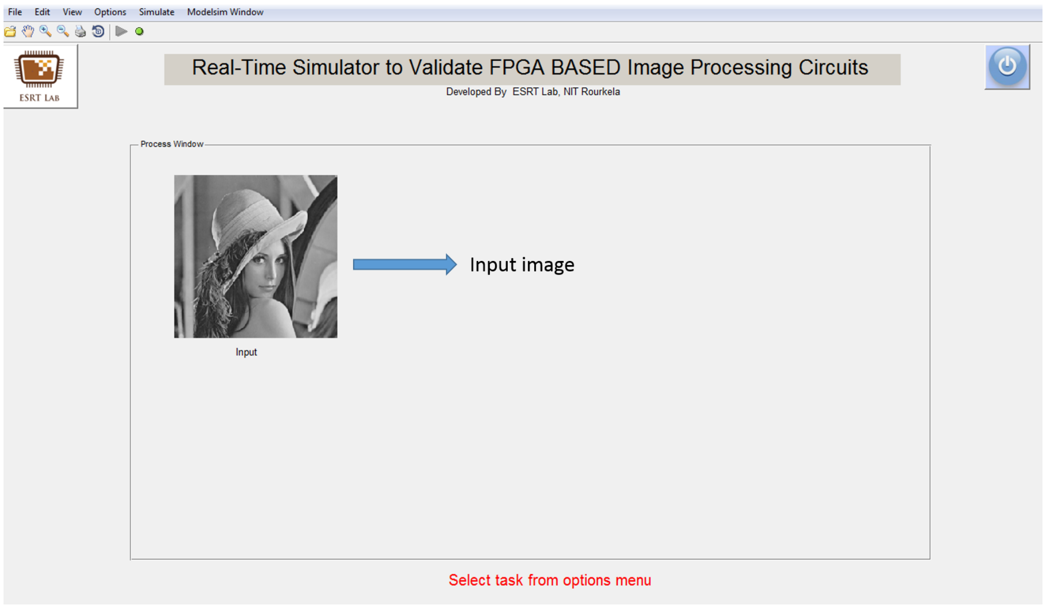

File Edit View Options Simulate Modelsim Window
ESRT LAB
Real-Time Simulator to Validate FPGA BASED Image Processing Circuits
Developed By ESRT Lab, NIT Rourkela
Process Window
Input image
Input
Select task from options menu

FIGURE A.5: Input Image

- Select Edge detection task from options menu as shown in figureA.6. After that the process window title will be changed to edge detection.

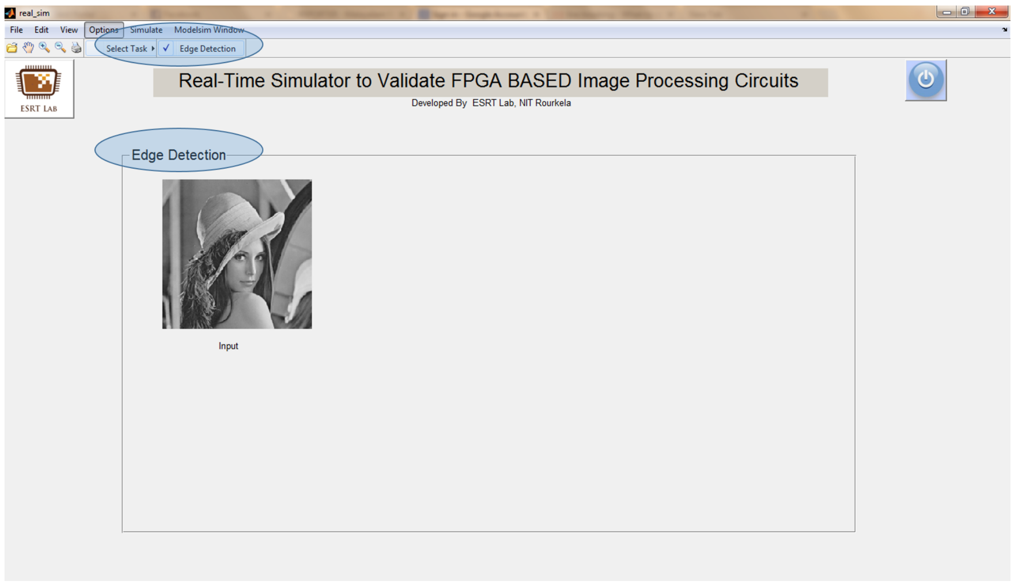

real_sim
File Edit View Options Simulate Modelsim Window
Select Task ▸ ✓ Edge Detection
ESRT LAB
Real-Time Simulator to Validate FPGA BASED Image Processing Circuits
Developed By ESRT Lab, NIT Rourkela
Edge Detection
Input

FIGURE A.6: Edge detection selection

- Begin the simulation by pressing the simulate button from the tool bar or by selecting simulate option from the simulate menu. This will open up a progress bar, indicating the progress of the simulation. The internal events will be displayed in the status bar, at the bottom of the screen. This is shown in A.7.

A.5 Viewing Results

- The edge detected output will be shown in the process window as in figureA.8.

- To view the test bench, select appropriate heximage panel from the view menu. Also this menu provides an option to see test bench as a report. To do this select view, and then the heximage file option. This operations are shown in figureA.9.

- To view the detailed simulation report, select HDL simulation report option from view menu. This will generate the report in a text file. This is shown in figureA.10.

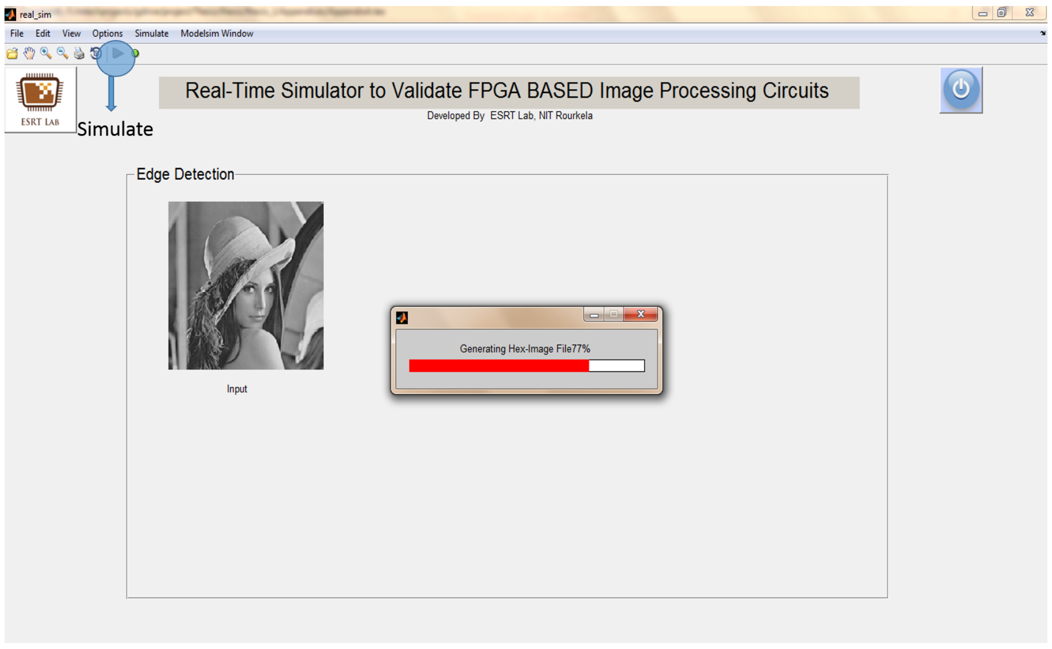
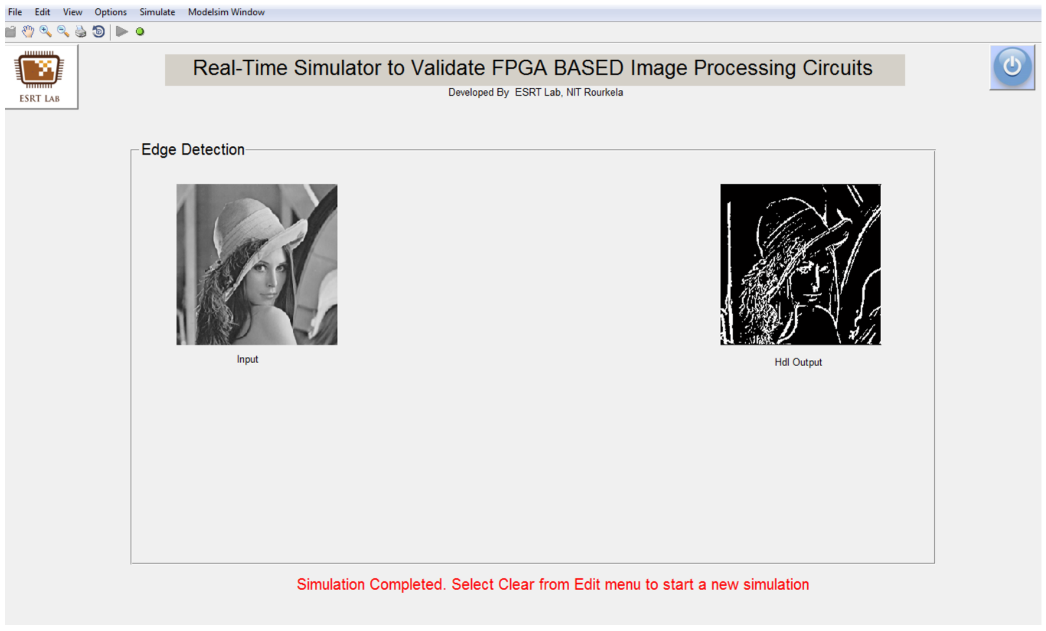
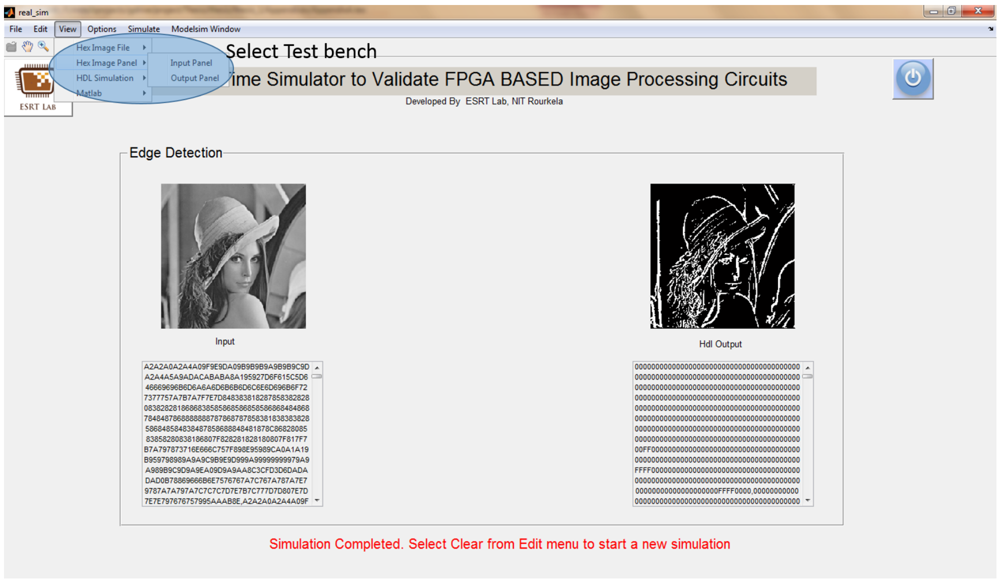

real_sim
File Edit View Options Simulate Modelsim Window
ESRT LAB
Simulate
Real-Time Simulator to Validate FPGA BASED Image Processing Circuits
Developed By ESRT Lab, NIT Rourkela
Edge Detection
Input
Generating Hex-Image File77%

FIGURE A.7: Simulation in realsim


File Edit View Options Simulate Modelsim Window
ESRT LAB
Real-Time Simulator to Validate FPGA BASED Image Processing Circuits
Developed By ESRT Lab, NIT Rourkela
Edge Detection
Input
Hdl Output

Simulation Completed. Select Clear from Edit menu to start a new simulation

FIGURE A.8: Viewing simulation result

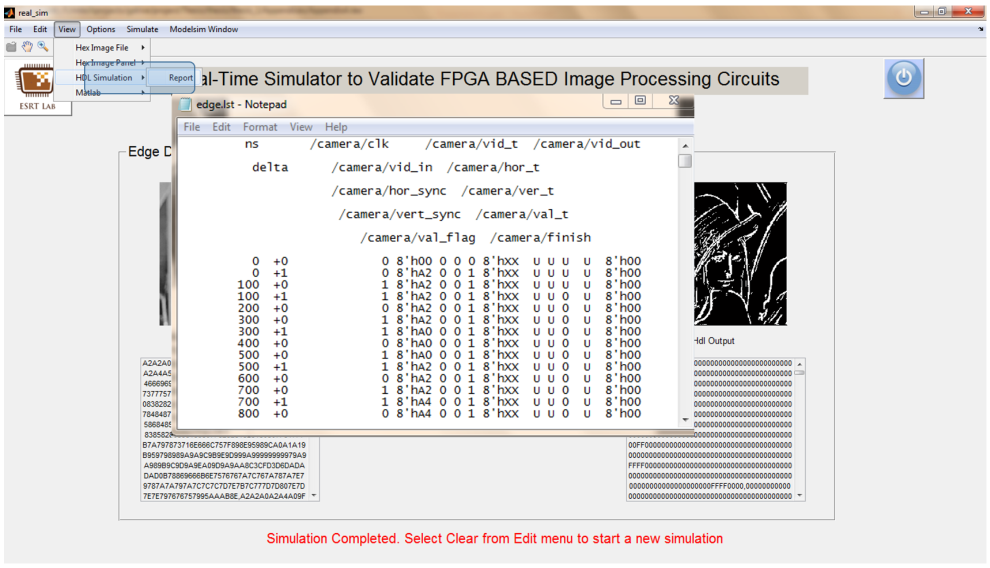

real_sim
File Edit View Options Simulate Modelsim Window
Hex Image File
Hex Image Panel
HDL Simulation
Matlab
Input Panel
Output Panel
Select Test bench
ime Simulator to Validate FPGA BASED Image Processing Circuits
Developed By ESRT Lab, NIT Rourkela
ESRT LAB
Edge Detection
Input
Hdl Output

```
A2A2A0A2A4A09F9E9DA09B9B9B9A9B9B9C9D
A2A4A5A9ADACABABA8A195927D6F615C5D6
46669696B6D6A6A6D6B6B6D6C6E6D696B6F72
7377757A7B7A7F7E7D848383818287858382828
083828281868683858586858685858686848486
8784848786888888878786878785838183838382
58684858483848785868884848187 8C86828085
8385828083818680 7F828281828180807F817F7
B7A797873716E666C757F898E95989CA0A1A19
B959798989A9A9C9B9E9D999A99999999979A9
A989B9C9D9A9EA09D9A9AA8C3CFD3D6DADA
DAD0B78869666B6E7576767A7C767A787A7E7
9787A7A797A7C7C7C7D7E7B7C777D7D807E7D
7E7E797676757995AAAB8E,A2A2A0A2A4A09F
```

```
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
00FF000000000000000000000000000000000000
0000000000000000000000000000000000000000
FFFF000000000000000000000000000000000000
0000000000000000000000000000000000000000
00000000000000000000FFFF0000,0000000000
0000000000000000000000000000000000000000
```

Simulation Completed. Select Clear from Edit menu to start a new simulation

FIGURE A.9: Viewing the test-bench in realsim


real_sim
File Edit View Options Simulate Modelsim Window
Hex Image File
Hex Image Panel
HDL Simulation
Matlab
Report
al-Time Simulator to Validate FPGA BASED Image Processing Circuits
ESRT LAB
Edge D
Hdl Output
edge.lst - Notepad
File Edit Format View Help

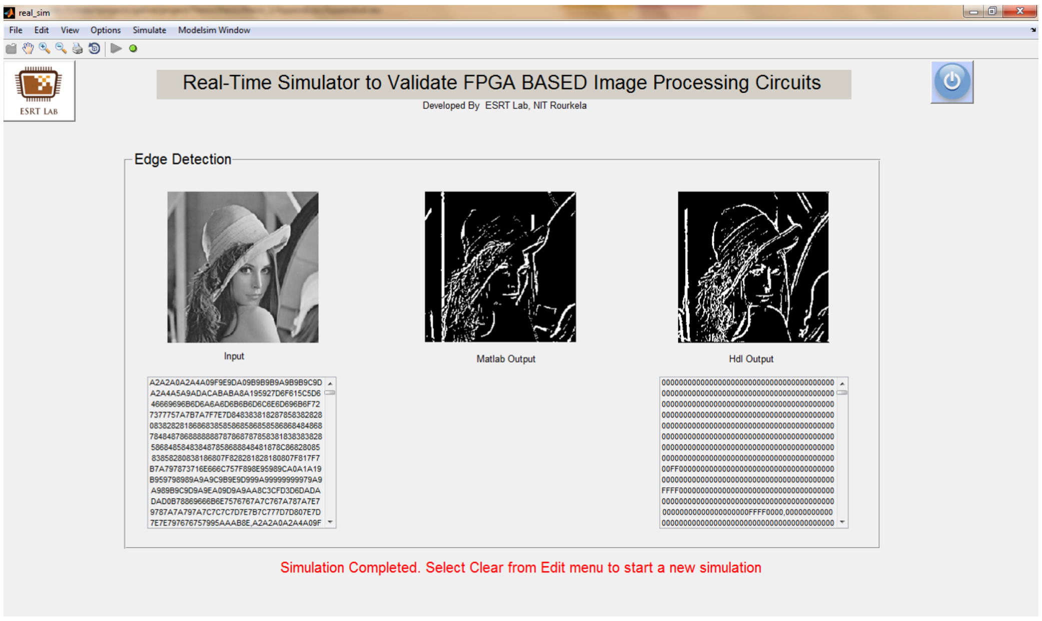
```
      ns      /camera/clk     /camera/vid_t  /camera/vid_out
       delta      /camera/vid_in  /camera/hor_t
                  /camera/hor_sync  /camera/ver_t
                   /camera/vert_sync  /camera/val_t
                      /camera/val_flag  /camera/finish

       0  +0            0 8'h00 0 0 0 8'hXX  U U U  U  8'h00
       0  +1            0 8'hA2 0 0 1 8'hXX  U U U  U  8'h00
     100  +0            1 8'hA2 0 0 1 8'hXX  U U U  U  8'h00
     100  +1            1 8'hA2 0 0 1 8'hXX  U U 0  U  8'h00
     200  +0            0 8'hA2 0 0 1 8'hXX  U U 0  U  8'h00
     300  +0            1 8'hA2 0 0 1 8'hXX  U U 0  U  8'h00
     300  +1            1 8'hA0 0 0 1 8'hXX  U U 0  U  8'h00
     400  +0            0 8'hA0 0 0 1 8'hXX  U U 0  U  8'h00
     500  +0            1 8'hA0 0 0 1 8'hXX  U U 0  U  8'h00
     500  +1            1 8'hA2 0 0 1 8'hXX  U U 0  U  8'h00
     600  +0            0 8'hA2 0 0 1 8'hXX  U U 0  U  8'h00
     700  +0            1 8'hA2 0 0 1 8'hXX  U U 0  U  8'h00
     700  +1            1 8'hA4 0 0 1 8'hXX  U U 0  U  8'h00
     800  +0            0 8'hA4 0 0 1 8'hXX  U U 0  U  8'h00
```

Simulation Completed. Select Clear from Edit menu to start a new simulation

FIGURE A.10: Viewing HDL simulation report

- There is an option in view window to see the output of the same image processing operation in sequential environment. The comparison between two results can be obtained by clicking the difference option from view menu. This is described in figureA.11.
- To see the waveform analysis of the output, and input images, select HDL window from Modelsim Window menu. It will open up the Modelsim


real_sim
File Edit View Options Simulate Modelsim Window
ESRT LAB
Real-Time Simulator to Validate FPGA BASED Image Processing Circuits
Developed By ESRT Lab, NIT Rourkela
Edge Detection
Input
Matlab Output
Hdl Output

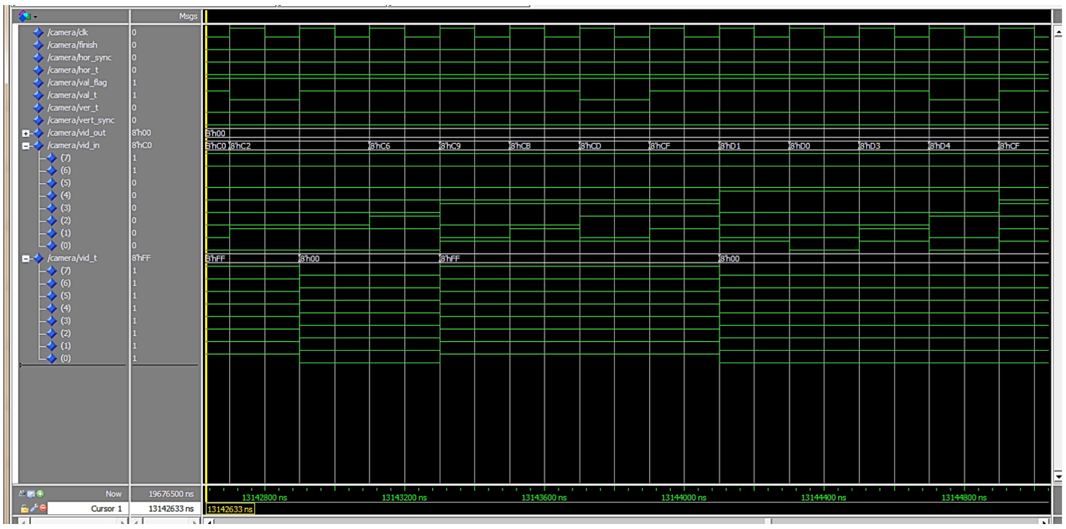
```
A2A2A0A2A4A09F9E9DA09B9B9B9A9B9B9C9D
A2A4A5A9ADACABABA8A195927D6F615C5D6
46669696B6D6A6A6D6B6B6D6C6E6D696B6F72
7377757A7B7A7F7E7D84838381828785838282 8
083828281868683858586858685858686848486 8
78484878688888887878687878583818383838 28
5868485848384878586888484818 78C86828085
838582808381868 07F82828182818080 7F817F7
B7A797873716E666C757F898E95989CA0A1A19
B959798989A9A9C9B9E9D999A99999999979A9
A989B9C9D9A9EA09D9A9AA8C3CFD3D6DADA
DAD0B78869666B6E7576767A7C767A787A7E7
9787A7A797A7C7C7C7D7E7B7C777D7D807E7D
7E7E79767675799 5AAAB8E,A2A2A0A2A4A09F
```

```
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
00FF000000000000000000000000000000000000
0000000000000000000000000000000000000000
FFFF000000000000000000000000000000000000
0000000000000000000000000000000000000000
000000000000000000000FFFF0000,0000000000
0000000000000000000000000000000000000000
```

Simulation Completed. Select Clear from Edit menu to start a new simulation

FIGURE A.11: Comparing HDL and Matlab output in realsim

wave editor window, in which the waveform analysis can be done. This is shown in figureA.12.



```
                      Msgs
/camera/clk           0
/camera/finish        0
/camera/hor_sync      0
/camera/hor_t         0
/camera/val_flag      1
/camera/val_t         1
/camera/ver_t         0
/camera/vert_sync     0
/camera/vid_out       8'h00
/camera/vid_in        8'hC0
  (7)                 1
  (6)                 1
  (5)                 0
  (4)                 0
  (3)                 0
  (2)                 0
  (1)                 0
  (0)                 0
/camera/vid_t         8'hFF
  (7)                 1
  (6)                 1
  (5)                 1
  (4)                 1
  (3)                 1
  (2)                 1
  (1)                 1
  (0)                 1
Now                   19676500 ns
Cursor 1              13142633 ns
```

FIGURE A.12: Wave form analysis from realsim interface

Bibliography

[1] Donald G Baily, *Design for embedded image processing on FPGAS*, :John Wiley And Sons 2011

[2] Nasser Kehtarnavaz and Mark Gamadia, *Real-Time Image and Video Processing* :From Research to Reality: Morgan And Clay pool 2006

[3] Peter J. Ashenden, *Digital Design, An Embedded Systems Approach Using VHDL*:MorganKaufmann Publishers 2008

[4] Charles H. Routh, Jr *Digital Systems Design Using VHDL*:PWS Publishers 1998

[5] James R.Armstrong & ,F.Gail Gray *VHDL Design Representation and Synthesis 2nd ed* :MorganKufmann Publishers 2008

[6] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins, *Digital Image Processing Using MATLAB 2nd ed* :Tata McGraw Hill Education Private Limited 2010

[7] Michel J. flinn, *Computer Architecture: Pipelined and Parallel Processor Design 1st ed* :Johns and Barlet publishers 1995

[8] Volnei.A.Pedroni, *Digital Electronics and design with VHDL 1st ed* :MorganKufmann 2008

[9] *XtremeDSP^{TM} DevelopmenPlatform:SPARTAN-3A DSP 3400A Edition User Guid,Xilinx* ,UG498 (v2.2) November 17, 2008

[10] *Camera Image Processing Reference Design:Zynq-7000 All Programmable SoC Video and Imaging Kit,Xilinx Application notes* ,XAPP794 (v1.2) January 2, 2013

[11] Christos Kyrkou, And Theocharis Theocharides , "A Flexible Parallel Hardware Architecture for AdaBoost-Based Real-Time Object Detection" *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS*,VOL. 19, NO. 6, JUNE 2011

[12] J. Matai, A. Irturk And R.Kastner , "Design and Implementation of an FPGA-based Real-Time Face Recognition System" *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*,On page(s): 141 – 148, 2011

[13] Aitzol Zuloaga, Unai Bidarte, Jose L. Martín and Joseba Ezquerra, "Optical Flow Estimator Using VHDL for Implementation in FPGA" *Proceedings XIII design of circuits and systems conference*, on pages. 36-41, November 1998.

[14] "Composite Video Signals", http://www-inst.eecs.berkeley.edu/ cs150/sp99/sp99/project/compvideo.htm,June 7, 2004.

[15] "Composite Video Signals", http://www.mentor.com/company/higher_ed/modelsim-student-edition, May 12, 2014.