IMPLEMENTATION OF CANNY EDGE DETECTION USING HLS(HIGH LEVEL SYNTHESIS)

A graduate project submitted in partial fulfillment of the requirements
For the degree of Masters of Science
In Electrical Engineering

By

Darpan Daru

December 2016
The graduate project of Darpan Daru is approved:

____________________________________ _____________________
Dr. Maryam Tabibzadeh          Date

____________________________________ _____________________
Dr. Ramin Roosta          Date

____________________________________ _____________________
Dr. Shahnam Mirzaei,Chair          Date

California State University, Northridge
ACKNOWLEDGEMENTS

First, I would like to give my heartfelt appreciation to Dr. Shahnam Mirzaei for guiding me throughout the project. He inspired me to work on this complex topic. Not only he guided me, but his continuous encouragement also helped me in completing the project without any hindrance. I would also like to appreciate Dr. Ramin Roosta and Dr. Maryam Tabibzadeh for giving us invaluable inputs in the completion of the project. Also, I would like to thank George Law(Chair) for helping me to understand the basics of System of Chip.

Finally, I would take the chance to thank my family members, friends for giving me constant support during my time in school.
# TABLE OF CONTENTS

SIGNATURE PAGE ii
ACKNOWLEDGEMENTS iii
LIST OF FIGURES vi
ABSTRACT ix

## 1. INTRODUCTION 1

1.1 Overview 1

1.2 Edge Detection Image Processing Algorithm 1

1.2.1 Brief Description 1

1.2.2 How it Works 2

1.3 Detailed Description 3

1.3.1 Gaussian Filter 3

1.3.2 Find Direction and Magnitude Gradient of the Image 4

1.3.3 Non-maximum Suppression 6

1.3.4 Hysteresis 8

1.4 Design Flow 9

## 2. HIGH LEVEL SYNTHESIS TOOL 11

2.1 Overview 11

2.2 Interface(Port Level) 13

2.2.1 AXI4 Interfaces 13

2.2.2 No I/O Protocols 16

2.2.3 Wire Handshakes 16

2.2.4 Memory Interface 16

2.3 Interface(Block Level) 17

2.4 Interface(Clock and Reset) 17

2.5 Design Optimization 17

2.5.1 Throughput Optimization 17

2.5.2 Latency Optimization 20

2.5.3 Area Optimization 20

2.6 C Libraries In HLS 21

2.6.1 Line Buffer 21

2.6.2 Window Buffer 23
2.6.3 OpenCV Video Library Functions 24
2.7 RTL Verification 24

3. HLS IMPLEMENTATION OF CANNY ALGORITHM 26
   3.1 Overview 26
   3.2 Matlab Implementation to get Gradient Direction and Magnitude 26
   3.3 Non-maximum Suppression in HLS 27
      3.3.1 Non-maximum Suppression Function Implementation 30
      3.3.2 Synthesise Result of Non-maximum Suppression 32
   3.4 Hysteresis Function Implementation 34
      3.4.1 Synthesise Result of Hysteresis 36
   3.5 Testing and Verification 37
      3.5.1 C/RTL Simulation Output 37

4. SYSTEM ON CHIP RESOURCES 39
   4.1 Zynq7 Processing System 39
   4.2 Timer 41
   4.3 UART 42
   4.4 DMA controller 43
   4.5 AXI interface 44
   4.6 ZedBoard Hardware 45

5. IMPLEMENTATION OF THE DESIGN ON SoC 46
   5.1 Project Setup 46
   5.2 Block Diagram 46
   5.3 Block Diagram of Hysteresis Algorithm Implementation 50
   5.4 Resource Utilization 51
   5.5 Hardware Setup 52
   5.6 Xilinx SDK 53

6. CONCLUSION 56
REFERENCES 57
APPENDIX A 58
LIST OF FIGURES

Figure 1.1: Block Diagram of Canny Edge Detection Algorithm  2
Figure 1.2: Gaussian Mask  3
Figure 1.3: Matlab Result of Gaussian Filter  3
Figure 1.4: Edge Direction  4
Figure 1.5: Gx(left), Gy(right) Directional Gradient  5
Figure 1.6: Normalized Gradient Direction and Gradient Magnitude(right)  5
Figure 1.7: Edge Thinning  6
Figure 1.8: 3x3 Window for Non-maximum Suppression  7
Figure 1.9: Output Image from Non-maximum Suppression  7
Figure 1.10: High and Low Threshold Value  8
Figure 1.11: Matlab Output of Canny Edge Detection Algorithm  9
Figure 1.12: High Level Design  10
Figure 2.1: Vivado Design Flow  11
Figure 2.2: Vivado HLS Design Flow  12
Figure 2.3: Vivado HLS Directive Editor  13
Figure 2.4: AXI-stream Interface Implementation  14
Figure 2.5: AXI4 Lite Slave Interface  15
Figure 2.6: RAM Interface  16
Figure 2.7: FIFO Interface  16
Figure 2.8: Function Without Pipelining  17
Figure 2.9: Function With Pipelining  18
Figure 2.10: Loop Without Pipelining  18
Figure 2.11: Loop With Pipelining  18
Figure 2.12: Rolled loop, Partially unrolled loop, Unrolled loop  19
Figure 2.13: Line Buffer Initial Position of Data  22
Figure 2.14: Data after Vertical Shift up and Insert Data at the Top  22
Figure 2.15: Data after Vertical Shift down and Insert Data at the Bottom  23
Figure 2.16: Initial Memory Buffer  23
Figure 2.17: Shift up Operation  23
Figure 2.18: Shift down Operation  23
Figure 2.19: Insert new Value  
Figure 2.20: Verification Flow of RTL  
Figure 3.1: Edge Detection Algorithm Flow  
Figure 3.2: Matlab Script to Generate Gradient Magnitude and Direction  
Figure 3.3: C++ Snippet Showing Line Buffer and Window Implementation  
Figure 3.4: Gradient Magnitude and Gradient Direction Input to the Function  
Figure 3.5: Line Buffer at Time 0  
Figure 3.6: Line Buffer and Window Operation  
Figure 3.7: Flowchart of C++ Code  
Figure 3.8: C++ Snippet for Window Insert Function  
Figure 3.9: Side Channel Stream Snippet  
Figure 3.10: Flowchart for Categorized Directions  
Figure 3.11: Algorithm to Categorize Direction  
Figure 3.12: Process of Non-maximum Suppression  
Figure 3.13: Performance Estimation of Generated Hardware  
Figure 3.14: Utilization Estimation  
Figure 3.15: RTL Ports of Generated Hardware axi Lite  
Figure 3.16: RTL Ports of Generated RTL Design axi Stream  
Figure 3.17: C++ Snippet for Hysteresis Function  
Figure 3.18: Flowchart of Hysteresis Algorithm  
Figure 3.19: Timing Estimation  
Figure 3.20: Resource Utilization  
Figure 3.21: Scalar Interface Port  
Figure 3.22: Non-maximum Suppression RTL/C co Simulation Output  
Figure 3.23: Hysteresis C/RTL co-simulation Output  
Figure 4.1: Zynq Overview  
Figure 4.2: UART Function Diagram  
Figure 4.3: Block Diagram of DMA Controller  
Figure 4.4: Architecture for Write Channel  
Figure 4.5: Zedboard Block Diagram  
Figure 5.1: Project Setting
Figure 5.2: Address Editor
Figure 5.3: Block Diagram Implementation of Non-maximum Suppression
Figure 5.4: Zynq Block Design
Figure 5.5: IP core for Non-maximum Suppression
Figure 5.6: DMA Controller with Data Width of 8 bit
Figure 5.7: DMA Controller with Data Width of 16 bit
Figure 5.8: Block Design for Hysteresis Algorithm
Figure 5.9: IP for Hysteresis Algorithm
Figure 5.10: Utilization of Hysteresis Implementation
Figure 5.11: Utilization of Non-maximum Suppression
Figure 5.12: Hardware Setup
Figure 5.13: XMD Console
Figure 5.14: Output Console for Non-maximum Suppression
Figure 5.15: Output Console for Hysteresis
Figure 5.16: Non max Suppression Output
Figure 5.17: Hysteresis Output
ABSTRACT

Implementation of Canny Edge Detection using HLS (High Level Synthesis)

By

Darpan Daru

Master of Science in Electrical Engineering

Hardware implementation to process the image is useful to accomplish low power and high-speed requirement of currently embedded application. Thus, one of the excellent solutions is to combine processing systems and hardware accelerators. Edge detection is one of the primary blocks in the image processing, and Canny edge detection algorithm is most widely used edge detection algorithm. In this project hardware accelerator of different blocks of canny edge detection algorithm is proposed using High-Level Synthesis. The main approach is to target canny edge detection algorithm to Programmable Logic part of SoC and accelerate the implementation on Zynq platform. This can be further extended to perform real-time image processing.
1. INTRODUCTION

1.1 Overview
Hardware implementation of image processing algorithm is essential for embedded system to achieve low power and high performance[1]. However, the design requires high-level language to describe the algorithm and to develop optimized HDL code from that algorithm is a tough job. High-level Synthesis tool automatically converts high-level codes into hardware description language. Validation and development are much easier in high-level languages.

This project describes the development of image processing hardware accelerator and implementation of the accelerator into SoC. Here two stages of canny edge detection algorithm is developed in Vivado HLS tool and then exported to Vivado to use it as an IP. The implementation of this generated IP is synthesized and tested on Zynq7 Zedboard.

Detailed description about canny edge detection and its design flow is described in this chapter. Chapter 2 gives a brief description of Vivado HLS, different optimization technique, and libraries. Development of image processing algorithm in Vivado HLS is shown in Chapter 3 and hardware implementation in Zynq is shown in chapter 5. Brief description about Zynq is given in chapter 4.

1.2 Edge Detection Image Processing Algorithm
Digital image processing edge detection is a mathematical method to detect points in the image where the brightness of the image changes, that curved line is known as edges. Typical edge detection techniques are Sobel, Prewitt, fuzzy logic, Roberts, and Canny edge detection[2].

1.2.1 Brief Description
Canny Edge Detection Technique is very popular Edge detection technique in image processing for extraction of edges. It is used to obtain the boundaries of the object inside the image. John F. Canny developed it in 1986[3]. It is mostly used because of low error rate, proper localization, and insignificant response. This algorithm is harder to implement compared to other edge detection algorithms because of intense
Canny Edge detection gives better detection of Edge, better localization, and clear response[3].

**1.2.2 How it Works:**

Canny Edge Detection algorithm is a multi-level process.

1. **Gaussian Filter:** Noise can easily affect the edge detection result, so it is important to eliminate noise from the image to withdraw false detection of edges[1].

2. **Find the direction and magnitude gradient of the image:** Roberts, Prewitt, and Sobel operator is used to extract the parallel, perpendicular and diagonal edges from the smoothed image. From this step gradient strength and direction can be achieved.

3. **Non-maximum suppression:** Edge thinning is done in this step. Local maximum is the location where the change in intensity value is very keen. In this step, all the other value except local maximum is suppressed.

4. **Hysteresis for edge tracing[4]:** This step is to discover the true edges in particular image. Edges which lies between the minimum threshold and maximum threshold are defined as classified edges and other edges are discarded based on their connectivity.

![Figure 1.1: Block Diagram of Canny Edge Detection Algorithm](image)

It is most efficient edge detection method is acknowledged as the optimal edge detector, and it works on the grayscale image.
1.3 Detailed Description:
This part describes the four stage implementation of the Canny filter.

1.3.1 Gaussian Filter: Gaussian filter is used to advance the trade-off between edge localization and noise filtering. The 5x5 Gaussian mask is used to smoothen the image as shown here in figure 1.2.

![Gaussian Mask][4]

Here after this filter pass over the image, each and every pixel is reevaluated in the form of an aggregate of its values in the Gaussian mask multiply by corresponding Gaussian weight. Furthermore, this value is then divided by the total weight of the 5x5 mask. Implementation result of the gaussian filter is shown here.

![Matlab Result of Gaussian Filter][3]
1.3.2 Find Direction and Magnitude Gradient of the Image:

The next step in edge detection algorithm is to use Sobel mask to get the first derivative in vertical and horizontal direction. Here Sobel operator(Sobel operator kernel can change from 3x3 to 9x9 depends on image’s sharpness) is convolved on 3x3 neighborhood pixels matrix of current pixels described as Gx and Gy which is given as

\[
G_x = \begin{bmatrix}
-1 & 0 & +1 \\
-2 & 0 & +2 \\
-1 & 0 & +1
\end{bmatrix} \ast A \quad \text{and} \quad G_y = \begin{bmatrix}
-1 & -2 & -1 \\
0 & 0 & 0 \\
+1 & +2 & +1
\end{bmatrix} \ast A
\]

Now, edge gradient can be given as

\[
G = \sqrt{G_x^2 + G_y^2}
\]

And Edge direction is given as

\[
\Theta = \text{atan2}(G_y, G_x)
\]

Here atan2 is the arctangent function with two inputs Gy and Gx, which gives the direction of the pixel. This direction is then rounded to the four angle 0°, 45°, 90° and 135° as shown in the figure.

![Figure 1.4: Edge Direction][4]

floating point arithmetic is avoided for fast calculation[1]. Matlab implementation is given here to find magnitude and direction gradient.
Gx, Gy and gradient magnitude and gradient direction can be found as

\[
\begin{align*}
[Gx,Gy] &= \text{imgradientxy}(I); \\
[Gmag,Gdir] &= \text{imgradient}(I, 'sobel');
\end{align*}
\]

Figure 1.5: Gx(left), Gy(right) Directional Gradient

Figure 1.6: Normalized Gradient Direction(left) and Gradient Magnitude(right)
Now after getting gradient and magnitude of the image, the image is fully scanned to remove all the unwanted pixels which does not establish the edge.

1.3.3 Non-maximum Suppression:
In this action simply local maxima is marked as an edge, and others will be suppressed and considered as zero[4]. This process is also known as edge thinning technique. The algorithm can be expressed in two steps. 1. Edge strength is compared between the current pixel and the pixels in its negative and positive Gdir 2. If the current pixel’s strength is sufficient in the comparison with other pixels in Gdir, then we will keep the current pixel’s value otherwise it will be suppressed to zero.

In this implementation 3x3 window is passed to access all the pixel’s gradient magnitude and gradient direction. For every center pixel in a window is suppressed, whenever the magnitude of that pixel is not greater than the magnitude of the neighbor pixel in its gradient direction. For instance

- Whenever the direction of the current pixel is north-south, then the gradient angle will be 0°. Now that pixel will be taken on the edge only if its gradient magnitude is larger than the Gmag of the pixel in east and west.

![Figure 1.7: Edge Thinning][5]

Here as shown in the figure 1.7 if point A is upon edge then its gradient angle will be 0° and its gradient magnitude will be compared with C and D. If its value is more than C or B then only A will be defined as an edge[5].

- Now if the gradient direction is east-west, then the rounded gradient angle will be 90°. Furthermore pixel will be considered upon edge only if its magnitude is bigger than Gmag of north and south.
- Gradient angle will be $135^\circ$ and edge direction will be northeast-southwest only if current pixel’s gradient magnitude is more than $G_{mag}$ at north west and south west and that current pixel is taken as an edge.
- Gradient angle will be $45^\circ$ and edge direction will be northwest-southwest only if current pixel’s magnitude is stronger than $G_{mag}$ at pixels in southwest and northeast, and the present pixel is understood as an edge. Here as shown in 3x3 window current matrix’s gradient magnitude is 14 and edge direction in east-west. So we will compare its $G_{mag}$ with north and south pixel’s $G_{mag}$.

![3x3 Window for Non-maximum Suppression](image)

**Figure 1.8 : 3x3 Window for Non-maximum Suppression**

After non maximum suppression the output image is shown here in the figure.

![Output Image from Non-maximum Suppression](image)

**Figure 1.9: Output Image from Non-maximum Suppression**

We can notice the binary image with the thin edge. Some of the edges are unnecessary which will be removed in the next step.
1.3.4 Hysteresis:
In this stage we are going to extract the true edges from the image, as shown in the previous image there are some weak edges and noise which are going to be removed from this step.

First of all, to get rid of the inaccurate edge pixels which are present after nonmaximum suppression because of noise and color variation are removed using two threshold values. To filter out these spurious responses it is important to deal with the poor gradient value. High threshold and Low threshold values are defined and if the gradient value of the pixel is greater than low threshold and lower than the high threshold, that image pixel is marked as a weak pixel and it will be suppressed from the image.

After this only powerful edge pixel are there in the image so the last step is to identify those weak edges which are parallel to true edges and remove them.

![Figure 1.10: High and Low Threshold Value](image)

Here as shown in the figure 1.10 edge A is above the High threshold, so it will be taken as a right edge, even though edge pixel B is below the High threshold it is combined with the edge A so it is considered as a true edge. Here edge C is below the High threshold, and above Low threshold Moreover it is not connected to any of the true edges so edge C will be excluded. So Low Threshold and High threshold values play a significant part in this section of edge detection algorithm[5].
So, Here is the output of the Hysteresis step, which is the final Matlab output of the Edge Detection algorithm. We can observe from the image that this algorithm identifies edges with very low inaccuracy and most of the false edges generated from the noise are removed from the image.

![Image](image1.png)

**Figure 1.11 : Matlab Output of Canny Edge Detection Algorithm**

**1.4 Design Flow:**

To speed up canny method, there are several blocks which perform different operations. Following steps described design flow.

- Matlab implementation of initial two-step of canny edge detection algorithm(Gaussian filter, find gradient and magnitude )
- Development and validation of nonmaximum suppression and hysteresis
- Processed image on the PL side using image processing accelerator
- Save output data in the memory
- Generate image using data from memory using Matlab

Here is the high level design is shown in the figure 1.12. It includes Processing systems and programmable logic units of ZedBoard. Image processing IP and DMA
controller IP are in the PL side. DDR is available on Zedboard.

Figure 1.12: High Level Design
2. HIGH LEVEL SYNTHESIS TOOL

2.1 Overview
HLS tool is used to convert C/C++ program specification into RTL(register transfer level) implementation which can be synthesized into FPGA(field programmable gate array)[6].

HLS tool can generate intensive parallel architecture for FPGA using C, C++, System C or OpenCL(Open Computing Language) specifications with considering performance, power and cost constraints.

Benefits of High Level synthesis:
- Better productivity and easy for hardware designer. High-performance designs using the high level of abstraction can be created.
- Acceleration of computationally intensive area of the algorithm by software designers.
- Less development time for developing arithmetic algorithms as designed at a level which is ideal for the implementation information.
- We can also verify the design’s functional correctness in less time compare to traditional HDL.
- Available vast number of directives to optimize C synthesize process.
- We can generate many implementations of the same C course code by using a different number of directives.

Vivado HLS design Flow:

![Vivado Design Flow Diagram](image)

Figure 2.1 : Vivado Design Flow
Here as shown in the figure 2.1 first of all we need to compile and debug our algorithm in C, next step is to synthesize C algorithm using optimization directives. Here is the detailed design flow of Vivado HLS tool is given.

![Vivado HLS Design Flow](image)

**Figure 2.2 : Vivado HLS Design Flow[6]**

Furthermore, just one top level function can be defined and that can be synthesized, and all the other sub-functions in its hierarchy will be synthesized. After this analyze the generated report which gives the estimation about performance, utilization, and interface. So, we can change the directives for further optimization of the implementation, and we can also make different versions of the same algorithm until we get the desired performance characteristics.

Different optimization directive includes directive to instruct a task execution in a pipeline or to set the latency for the functions and loops. Other than this we can also specify the limits for the number of resources to be used. Some of the directives are to decide input and output port behavior of the algorithm.

Now, after synthesis, RTL is implemented in hardware description language. Output is generated for RTL implementation in either VHDL or Verilog. RTL implementation is verified using C/RTL simulation and further results are compared with the C simulation. This RTL implementation result is then packed into IP and different tools can utilize that in the design flow.
RTL mapping of C/C++ construct is shown below
Function - Modules
Arguments - Input/Output ports
Operators - Functional Units
Control Flows - Control Logic
Arrays - Memories

Here brief explanation about interface implementation, verification, optimization of the design, HLS library is given below.

2.2 Interface(Port Level)

2.2.1 AXI4 Interfaces:
Vivado HLS supports AXI4 interfaces which introduce AXI4 stream(axis),AXI-4 master(m_axi) and AXI4-Lite(s_axilite)[6]. Different pragmas are used to declare interface directive. Other than this Vivado HLS directive Editor can also be used to announce the directives which are shown in figure 2.3.

![Vivado HLS Directive Editor](image)

**Figure 2.3 : Vivado HLS Directive Editor**

- **AXI4-Stream Interface:** It can be defined on input argument or output argument including arrays or pointers.
  
  AXI4 stream interface always transfer the data in sequential manner and sign bit is always extended to the next byte moreover data transfer starts from first address and does not require any address management.it is useful for
burst data transfer which is good for image processing application. There are basic two types of AXI4 stream interface

- AXI-4 stream interface without side-channel
- AXI-4 stream interface with side-channels

Here C++ code struct is used to refer or control the side channel in the interface. Structs for the side channel is given in ap_axi_sdata.h header file which is shown below. To model stream interface HLS provides hls::stream class for C++ reference argument.

```cpp
#include "ap_int.h"

template<typename D, typename U, typename TI, typename TD>
struct ap_axis {
    ap_int<D> data;
    ap_uint<D> keep;
    ap_uint<D> strb;
    ap_uint<U> user;
    ap_uint<TI> last;
    ap_int<TD> dest;
};

template<typename D, typename U, typename TI, typename TD>
struct ap_axiu {
    ap_int<D> data;
    ap_uint<D> keep;
    ap_uint<D> strb;
    ap_int<U> user;
    ap_uint<TI> last;
    ap_int<TD> dest;
};
```

All the side channel port, TVALID and TREADY protocol ports are implemented with the data ports after synthesis. Tvalid and Tready must be 1 to get access from the Zynq[6]. Here example shows 32 bit signed and unsigned AXI4 stream interface declaration and implementation. Here ap_axis is for signed integer and ap_axiu is for unsigned integer. Side channel Tlast is useful in DMA operations.

```cpp
function_name(ap_axis<32,2,5,6>A[10], ap_axiu<32,2,5,6>B[10])
```

![Figure 2.4: AXI-Stream Interface Implementation[6]](image)

- **AXI4-Lite Interface**: It can be set to any argument. However, not to array argument. Moreover, we can bundle multiple arguments together.
It is useful when we need memory mapped single transfer of data. Axi lite interface does not support burst data transfer. (axi lite interface does not support array parameter in the function). In HLS all the parameter results are accessible through the axi4 lite interface and these parameters will be just some position in the memory. Next example presents Vivado HLS implementation for multiple arguments. Here return shows the function's return value. bundle will group all arguments which are specifically given with the same name.

```c
#pragma HLS INTERFACE s_axilite port=port_name bundle=multiple_argument
#pragma HLS INTERFACE s_axilite port=return bundle=multiple_argument
```

Implemented AXI4 lite port after synthesis generates the following ports[6].

- **ap_done**: set when function completes complete operation
- **ap_ready**: set when function is available to accept further data
- **ap_data**: data input or output arguments
- **ap_clk**: synchronous clock must be from same master clock
- **ap_addr**: specify the address of the interface
- **ap_rst_n**: use to reset the interface

Here in the figure example of the generated axi4 lite interface is given.

![Image of AXI4 Lite Slave Interface](image)

**Figure 2.5 : AXI4 Lite Slave Interface[6]**

Generated C drivers are useful to program the interface. Here ap_done and ap_ready ports indicates when the function completes its all operation and when the function is available to receive new data.

- **AXI4 Master Interface**: Specify on pointers and array. This interface can also bundled together with other arguments.
2.2.2 No I/O Protocols

Ap_none plus ap_stable are used to point that none I/O protocol be attached to the port.

2.2.3 Wire Handshakes:

Ap_hs use to generate two way handshake signal. This mode can be used with arrays to read or write sequential order.

2.2.4 Memory Interface

To implement array argument ap_memory interface is applied by default. They are used to communicate with the RAMs and ROMs when it needs to access the memory, which can be standard BRAM interface that has data, chip-enable, address, write-enable and address ports. The RESOURCE directive defines the single port or dual port. This figure shows the implementation of ap_memory interface

![Figure 2.6: RAM Interface](image)

The ap_memory and bram interface are functionally same but in ap_memory all interface presented as separate ports while in bram-interface displays single port which are grouped and ready to connect Xilinx BRAM with the single point-to-point connection.

Ap_fifo interface is used to when we need to access data from the array in a sequential manner. Ap_fifp interface enables the port to be connected with the FIFOs to have empty-full communication in both directions.

![Figure 2.7: FIFO Interface](image)
2.3 Interface (Block Level)
Ap_start, ap_done, ap_idle and ap_ready signals are added to the block. This signal controls the generated block without any influence of input-output interface constraints.

- Ap_start: start the process on the data
- Ap_ready: high if implemented block is ready to accept new data
- Ap_idle: implemented block is idle
- Ap_done: this signal indicates that the design finishes operation on data

2.4 Interface (Clock and Reset)
All the designs will have the same clock and it supports only one clock operations. There is always an uncertainty in the clock period given by the HLS tool so every time clock uncertainty is deducted from the estimated clock period. We can explicitly declare the uncertainty for the clock period but by default it takes 12.5 percent.

Ap_rst_n is a port which put FPGA registers and BRAM in an initial reset condition. There are three reset modes which we can add as an optimization directives:

- None: no reset
- Control: to reset all the control register
- State: reset all the control register, in addition to that it reset memories (all global and static variables are set back to its initial values)
- All: reset is given to all the register and all the memories.

2.5 Design Optimization
To achieve the required goals and for better performance there are optimization pragmas which can be given to the design to force Vivado HLS tool to generate design as per given specifications. Some of the optimization techniques are discussed in this section.

2.5.1 Throughput Optimization
- Pipelining: All the operations will operate concurrently that means next task can start its operation before previous task finishes its operation. Pipeline directive can be applied to loops and functions. Figure 2.8 explains how pipeline improves the throughput in the design.
The function without pipeline take 3 clock cycles till next read and takes 2 clock cycles to generate an output. Now, pipelined function can read every clock cycle with same latency and same resources.

Now, pipelining in the loop force the operations within the loop to implement in a simultaneous manner. Pipelining in the loop is described for the given ‘for’ loop.
Here as shown in Figure 2.11 pipelined loops has latency of 4 clock cycles lesser compared to the latency of the loop without pipeline. Moreover read operations on next input every clock cycle in loop with pipelining is shown in the figure. The main difference between the pipelined function and pipelined loops is that in functions pipeline never ends it runs forever while in loop it execute only for the loop iteration.

- Loop Unrolling:
  UNROLL directive[6] is used to partially or fully unroll the for loops. By default all the loops are rolled that means the same hardware resources used for the operations in the loop.
  - Rolled Loop: It takes separate clock cycles to perform each iteration in the loop which requires a multiplier and single port BRAM.
  - Partially Unrolled Loop: Here in given example as shown in Figure 2.12 partially unrolled loop by the factor of 2 requires 2 multiplier and dual port BRAM. which performs read and write in single clock cycle. Latency of rolled loop is double than the partially unrolled loop.
  - Unrolled Loop: It takes only one clock cycle to complete one operation. moreover it uses very less hardware resources.
  UNROLL directive can be given to the loops, and we can also apply this directive to the functions. In completely unrolled loop all the operation will be in parallel which depends on data dependency.

Here for-loop implementation is shown in the figure 2.12.

```plaintext
for (i=0;i<=3;i++)
    p[i]=q[i]*r[i];
```

Figure 2.12: Rolled loop, Partially unrolled loop, Unrolled loop
2.5.2 Latency Optimization

Latency pragma is used to make sure that implemented design finishes all the operations in the functions in given range of clock cycles. If the latency pragma is given inside the loop then it gives particular latency for the single repetition of the loop and if our purpose is to regulate the total latency of the loop then this pragma should be declared outside the loop.

Separate latency for all the iteration is declared as

```
Loop_A: for (i=0;i<n;i++)
  #pragma HLS latency max=10
  
  //loop body
```

Latency declaration for all the iteration is declared as

```
#pragma HLS latency max=10
Loop_A: for (i=0;i<n;i++)
  
  //loop body
```

Another option to reduce latency is to merge sequential loops for further optimization, moreover flattening of the nested loop can also improve latency. Directive for flattening loop should be given to the innermost loop of the loop body. It can be defined as

```
SetDirective_loop_flatten top/inner
```

2.5.3 Area Optimization:

For better area optimization it is necessary to use proper precision data types for the variable because use of improper bit-width can result in slower and slow hardware implementation. Which can also result in increased latency. Arrays are also implemented as RAM or registers so out of bound element may increase the hardware resources. INLINE directive is used to share components for better optimization in calling another function within the function. ARRAY_MAP pragma is used to map small arrays into one large array. This can lower down the number of block RAM requires for the hardware implementation. Vivado HLS allows us to limit the number of operators by forcing the synthesis tool to share the operators.
2.6 C Libraries In HLS

Libraries provided by HLS

- Precision Data types
- stream
- math
- video
- DSP
- linear algebra

Here in this project math, video and stream library are used. The brief description of those libraries are given below.

1. hls stream: Streaming data types have no address management moreover read and write performed in a sequential manner. hls::stream<> class is given as C++ class to design stream data structure. Without any declaration streaming structure is implemented as FIFO interface with the depth of 1 and optimization directive is applied to adjust its depth value. hls_stream.h header file is used to use this stream class. It provides blocking and nonblocking read and writes methods. Nonblocking methods allow reading FIFO even if it is empty[6].

2. hls math: This library is helpful for a synthesis of C and C++ math library functions including floating point operations. Hls_math.h is used at the time of synthesis. The only reason to use vivado hls math library instead of c math library is to get accurate C and C/RTL simulation result. Mathematical operation can be performed on the float and double data types. However, it provides less accurate output but fast hardware(RTL) implementation.

3. Hls video library: Header file is used to include all videos and image functions(hls_video.h). Memory line buffer and window buffer is used to implement our edge detection algorithm. Moreover, OpenCV functions are used to test the algorithm.

2.6.1 Line Buffer

This class is useful for the instantiation of line buffers. All the operations on the line buffer are defined as methods in this class.

- User can define total number of rows and columns in the line buffer
Debugging and implementation of line buffer is easy using methods in this class.
Data types can be parameterized.
It banks each rows into separate memory banks.

Here all the methods of line buffer class are described using example. This figure shows the line buffer with the initial data.

<table>
<thead>
<tr>
<th>Row</th>
<th>Column 0</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Row1</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Row2</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

**Figure 2.13: Line Buffer Initial Position of Data**

LineBuffer data type is used as shown in the example to instantiate the line buffer.

```cpp
hls::LineBuffer<rows,columns,type>variable;
```

```cpp
hls:: LineBuffer<3,4,char >Buffer_LINE
```

Data in the line buffer is organized in raster scan method. Different column number is used every time to add new data. To enter new data on the top or bottom of the column vertical shift shift_pixels_down[6] is useful. insert_top_row[6] is used to enter data on the top of that column. Given example is to add 20 on the top of the column 1.

```cpp
Buffer_LINE.shift_pixel_down(1)[6]; //vertical shift of column 1
Buffer_LINE.insert_top_row(20,1)[6]; //insert data into column 1 and on the top
```

<table>
<thead>
<tr>
<th>Row</th>
<th>Column 0</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row0</td>
<td>1</td>
<td>(20)</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Row1</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Row2</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

**Figure 2.14: Data after Vertical Shift up and Insert Data at the Top**

Below example shows how to enter the new value at the bottom of the line buffer at a particular column.

Here we are adding new data at the bottom of the column 0.

```cpp
Buffer_LINE.shift_pixels_up(0)[6]; // shift data up in column 0
Buffer_LINE.insert_bottom_row(20,0)[6]
```
To get value from any location of the linebuffer method getval(row,column) is used.

2.6.2 Window Buffer:

2 dimensional memory window is controlled and declared using C++ memory window class. This class has the same features as LineBuffer class. Here some of the methods from this class is explained using example.

Memory window is instantiated using $\texttt{hls::window<row,column,type>}\texttt{variable}. \texttt{Ex:}$

$hls::Window<3,3,\text{char}>\texttt{Buffer\_Window};$ //this will generate 3x3 memory window buffer

<table>
<thead>
<tr>
<th>Row</th>
<th>Column 0</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row 0</td>
<td>1</td>
<td>20</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Row 1</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Row 2</td>
<td>20</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

**Figure 2.15 : Data after Vertical Shift down and Insert Data at the Bottom**

Here to shift up and shift down the row data shift_pixels_down() and shift_pixels_up() is used. This figure shows the memory window results after the operation.

Buffer\_Window.shift_pixels_down()[6]// shift down the row and new data can be added to the first row

Buffer\_Window.shift_pixels_up()[6]// shift up the row and new data can be added to the bottom row

<table>
<thead>
<tr>
<th>Memory Window</th>
<th>C0</th>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row 0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Row 1</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Row 2</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

**Figure 2.16: Initial Memory Buffer**

<table>
<thead>
<tr>
<th>Memory Window</th>
<th>C0</th>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row 0</td>
<td>new</td>
<td>new</td>
<td>new</td>
</tr>
<tr>
<td>Row 1</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Row 2</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>
Same as above we can shift memory window in left or right using `shift_pixel_left()` and `shift_data_left()` methods. We can also insert value at any location of the window using `insert_pixel(value,row,column)` method. For example, `Buffer_Window(10,2,2)` will insert 10 at third row and third column.

<table>
<thead>
<tr>
<th>Memory Window</th>
<th>C0</th>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row 0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Row 1</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Row 2</td>
<td>7</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

![Figure 2.19: Insert new Value](image)

Insert methods are used to inset block in the window.

2.6.3 OpenCV Video Library Functions

OpenCV[8] functions can be used in vivado HLS, which can be implemented and synthesized. OpenCV interface functions are utilized to convert data to and from the OpenCV data types and AXI-4 stream data types. Video data which is declared as `hls::mat` data types can transform into AXI-4 stream data type using AXI-4 stream function, and it is implemented as the high-performance interface. Video images can be edited and processed using Video processing functions. These functions are also synthesizable.

2.7 RTL Verification

C/RTL co simulation is used to verify synthesized RTL output. Process for the verification is shown in the figure.

![Figure 2.20: Verification Flow of RTL](image)
Here as shown in the figure 2.20 output of the C simulation is applied as input vectors in RTL simulation and the after this input vectors are given to the implemented RTL module. The C test bench verifies the output of RTL simulation.

Now, next chapter shows the generation of IP of nonmaximum suppression and hysteresis using Vivado HLS tool.
3. HLS IMPLEMENTATION OF CANNY ALGORITHM

3.1 Overview:
Here is the flow chart of the implementation of canny edge detection algorithm in HLS.

![Flow Chart of HLS Implementation of Canny Algorithm](image)

Figure 3.1: Edge Detection Algorithm Flow

3.2 Matlab Implementation to get Gradient Direction and Magnitude:
Here for the implementation of the canny edge detection algorithm first RGB image is converted to the grayscale using the matlab function. It will generate 240x320 matrix in the matlab.

Gaussian filter is used to remove the noise from the image and smoothed image is used to find the gradient magnitude and direction of the image using sobel filter. Matlab script to generate Gradient direction and Gradient magnitude is given below.
Figure 3.2: Matlab Script to Generate Gradient magnitude and Direction

3.3 Non-maximum Suppression in HLS:

Now, Non maximum suppression is implemented and verified in Vivado HLS then the generated IP is used in next stage to implement it on SoC.

Non maximum suppression function has three streaming input.

1. Input gradient magnitude
2. Input gradient direction
3. Output image stream

```matlab
%input: RGB image file
RGB=imread('apple.jpg');
%Generate grayscale image
I_Gray=rgb2gray(RGB);
%gaussian filter to smoothen the image
Gaussian=imgaussfilt(I_Gray);
%find gradient direction and gradient magnitude
%using sobel filter
[GMag,GDir] = imgradient(Gaussian,'sobel');
%generate the text file
%Array of 76800(240x320) for gradient magnitude and gradient direction
Gmagcast=cast(GMag, 'int8');
Gdircast=cast(GDir, 'int16');
for i=1:320
    for j=1:240
        image_array(k)=I_Gray(i,j);
        Gmagnitude(k)=Gmagcast(i,j);
        Gdirection(k)=Gdircast(i,j);
        k=k+1;
    end
end
cswrite('Gmagnitude_apple.txt',Gmagnitude);
cswrite('Gdirection_apple.txt',Gdirection);
cswrite('nonmax_apple.txt',image_array);
```

Figure 3.3: C++ Snippet Showing Line Buffer and Window Implementation

Here gradient magnitude is taken as an unsigned and direction is taken as signed hls stream variable. Moreover as shown in the figure 3x240 line buffer is generated for both gradient and direction. Window memory buffer is implemented with the size of 3X3 for both the streaming inputs.
Here HLS INTERFACE directive is used to declare all the input and output arguments which are defined as axi4 stream interface moreover function will have all axi4 lite return ports to start and stop the function execution. Line buffer and window operation is described in the below figure.

<table>
<thead>
<tr>
<th>R1</th>
<th>R1.1</th>
<th>R1.2</th>
<th>R1.3</th>
<th>R1.4</th>
<th>R1.5</th>
<th>R1.6</th>
<th>R1.7</th>
<th>...</th>
<th>R1.236</th>
<th>R1.237</th>
<th>R1.238</th>
<th>R1.239</th>
<th>R1.240</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2</td>
<td>R2.1</td>
<td>R2.2</td>
<td>R2.3</td>
<td>R2.4</td>
<td>R2.5</td>
<td>R2.6</td>
<td>R2.7</td>
<td>...</td>
<td>R2.236</td>
<td>R2.237</td>
<td>R2.238</td>
<td>R2.239</td>
<td>R2.240</td>
</tr>
<tr>
<td>R3</td>
<td>R3.1</td>
<td>R3.2</td>
<td>R3.3</td>
<td>R3.4</td>
<td>R3.5</td>
<td>R3.6</td>
<td>R3.7</td>
<td>...</td>
<td>R3.236</td>
<td>R3.237</td>
<td>R3.238</td>
<td>R3.239</td>
<td>R3.240</td>
</tr>
<tr>
<td>R4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.4 : Gradient Magnitude and Gradient Direction Input to the Function

Figure 3.5 : Line Buffer at Time 0

Figure 3.6: Line Buffer and Window Operation

In line buffer, data can be accessed concurrently from three rows per clock cycle. Before starting calculating algorithm, there should be enough time, so line buffer is filled with enough data for the computation[9][10]. In this case, the window is 3x3, so there should be data in two row and first three column of the third row before we start the calculation for image processing algorithm. First of all first row of input data is given to the bottom of the line buffer and then line buffer is shifted up and new data is added to the lower part of the line buffer.
This line buffer will be implemented as dual port block RAM in the FPGA so there will be 3 pixels in one clock cycle, however it is not sufficient because algorithm needs to work on 9 pixels per clock cycle. Memory Window fulfills this requirement as we can access the data at the same time. It is implemented as flip flops in the FPGA. The central pixel computation will be done at every single clock cycle and data of the memory window will shift left and new data will be added to the right most column. So data in line buffer and window moves concurrently. When line buffer gets 240*2+3 pixels from the input then computation can be started. Program flow of the C++ code is given below.

Figure 3.7: Flowchart of c++ Code
First of all local memory buffer for magnitude and for gradient is defined. Its size will be 240X3 in this case as the size of the image is 240x320 and size of the window is 3x3. Magnitude buffer data will be 8 bit unsigned, and direction data types will be signed 16 bit. Now, for this algorithm 3x3 window memory buffer is defined. After this step for loop will iterate on all the pixels of the image.

Data from the axi4 stream is read using port.read() method as shown in the figure. This data is given to the line buffer using method described above.insert method is used to put the data from the line buffer to the 3x3 window.

```c++
signed short Dirval = edgex_srttuffer.getval(window_row,w1
// populate window with the data from the line buffer
Gradwindow.insert(Gradval,window_row,window_col);
Edgwindow.insert(Dirval,window_row,window_col);
```

**Figure 3.8: C++ Snippet for Window Insert Function**

Hereafter putting the first data in the line buffer, there is some wait time before start computation. For the time when line buffer is not filled till two rows and first three column, the last signal of the side channel is set to 0 which indicate the output is not valid for that time. After wait time the last signal of the side channel is configured to 1. After this, the output is valid. Next step is to write this side channel data on the output stream. The snippet of the code is displayed in the figure 3.9.

```c++
dataOutSideChannel.data = final_pixel;
dataOutSideChannel.keep = GradientSideChannel.keep;
dataOutSideChannel.strb = GradientSideChannel.strb;
dataOutSideChannel.user = GradientSideChannel.user;
dataOutSideChannel.last = 0;
dataOutSideChannel.id = GradientSideChannel.id;
dataOutSideChannel.dest = GradientSideChannel.dest;
// output data on the output stream
outStream.write(dataOutSideChannel);
```

**Figure 3.9 : Side Channel Stream Snippet**

**3.3.1 Non-maximum Suppression Function Implementation:**

Function for non maximum suppression has two arguments, first is magnitude window and gradient window from the main function. Here direction and magnitude at the position (1,1) in the window is first stored in a variable, which is used for further algorithm computation.
Figure 3.10: Flowchart for Categorized Directions

Here as shown in the Figure 3.10 using getval(row,col), the value for the center pixel is stored in a variable. As shown in the figure gradient direction from the previous step is then grouped into 0 degree, 90 degree, 45 degree, and 135 degree.

```c
// categorize direction
if ((tan < 22.5) && (tan > -22.5)) || (tan > 157.5) || (tan < -157.5)
    tan_direction = 0;
if ((tan > 22.5) && (tan < 67.5)) || ((tan < -112.5) && (tan > -187.5))
    tan_direction = 45;
if ((tan > 67.5) && (tan < 112.5)) || ((tan < -67.5) && (tan > -112.5))
    tan_direction = 90;
if ((tan > 112.5) && (tan < 187.5)) || ((tan < -112.5) && (tan > -157.5))
    tan_direction = 135;
```

Figure 3.11: Algorithm to Categorize Direction
After this center pixel is compared with other two pixels as per direction of the current pixel. After this process, the pixel which has the maximum value is left, and other pixels are suppressed from the image. Flowchart of this process is shown in this figure 3.12. For example here magnitude at the pixel(1,1,) is 5 and its value is examined with other two-pixel value depends on the value of direction of that pixel. If its direction is 0, then it is compared with pixel(0,1) and pixel(2,1) here in flowchart it is given as p(4) and p(6).

**Figure 3.12 : Process of Non-maximum Suppression**

3.3.2 Synthesise Result of Non-maximum Suppression:

Now, next step is to synthesize C++ algorithm using hls synthesizer and then export it to vivado. Generated synthesis report is displayed here. Which show our design’s latency and timing estimations. As shown in the figure design takes an 8.42ns time to finishes its operation.
Figure 3.13: Performance Estimation of Generated Hardware

Here as shown in the figure 3.13 implemented design uses very less resources of the FPGA. BRAM_18K is used for the implementation of the line buffer and flip flops are used to implement the memory window buffer.

<table>
<thead>
<tr>
<th>Name</th>
<th>BRAM_18K</th>
<th>DSP48E</th>
<th>FF</th>
<th>LUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>509</td>
</tr>
<tr>
<td>FIFO</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Instance</td>
<td>0</td>
<td>-</td>
<td>36</td>
<td>40</td>
</tr>
<tr>
<td>Memory</td>
<td>5</td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Multiplexer</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>412</td>
</tr>
<tr>
<td>Register</td>
<td>-</td>
<td>-</td>
<td>419</td>
<td>-</td>
</tr>
<tr>
<td>Total</td>
<td>5</td>
<td>0</td>
<td>455</td>
<td>961</td>
</tr>
<tr>
<td>Available</td>
<td>280</td>
<td>220</td>
<td>106400</td>
<td>53200</td>
</tr>
<tr>
<td>Utilization (%)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 3.14: Utilization Estimation

<table>
<thead>
<tr>
<th>RTL Ports</th>
<th>Dir</th>
<th>Bits</th>
<th>Protocol</th>
<th>Source Object</th>
<th>C Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_axi_CRTL_BUS_AVALID</td>
<td>in</td>
<td>1</td>
<td>s_axi</td>
<td>CRTL_BUS</td>
<td>return void</td>
</tr>
<tr>
<td>s_axi_CRTL_BUS_AREADY</td>
<td>out</td>
<td>1</td>
<td>s_axi</td>
<td>CRTL_BUS</td>
<td>return void</td>
</tr>
<tr>
<td>s_axi_CRTL_BUS_AWVALID</td>
<td>in</td>
<td>5</td>
<td>s_axi</td>
<td>CRTL_BUS</td>
<td>return void</td>
</tr>
<tr>
<td>s_axi_CRTL_BUS_AWREADY</td>
<td>out</td>
<td>1</td>
<td>s_axi</td>
<td>CRTL_BUS</td>
<td>return void</td>
</tr>
<tr>
<td>s_axi_CRTL_BUS_WDATA</td>
<td>in</td>
<td>32</td>
<td>s_axi</td>
<td>CRTL_BUS</td>
<td>return void</td>
</tr>
<tr>
<td>s_axi_CRTL_BUS_WSTRB</td>
<td>in</td>
<td>4</td>
<td>s_axi</td>
<td>CRTL_BUS</td>
<td>return void</td>
</tr>
<tr>
<td>s_axi_CRTL_BUS_ARVALID</td>
<td>in</td>
<td>1</td>
<td>s_axi</td>
<td>CRTL_BUS</td>
<td>return void</td>
</tr>
<tr>
<td>s_axi_CRTL_BUS_AREADY</td>
<td>out</td>
<td>1</td>
<td>s_axi</td>
<td>CRTL_BUS</td>
<td>return void</td>
</tr>
<tr>
<td>s_axi_CRTL_BUS_VVALID</td>
<td>in</td>
<td>5</td>
<td>s_axi</td>
<td>CRTL_BUS</td>
<td>return void</td>
</tr>
<tr>
<td>s_axi_CRTL_BUS_VREADY</td>
<td>in</td>
<td>1</td>
<td>s_axi</td>
<td>CRTL_BUS</td>
<td>return void</td>
</tr>
<tr>
<td>s_axi_CRTL_BUS_RDATA</td>
<td>out</td>
<td>32</td>
<td>s_axi</td>
<td>CRTL_BUS</td>
<td>return void</td>
</tr>
<tr>
<td>s_axi_CRTL_BUS_RRESP</td>
<td>out</td>
<td>2</td>
<td>s_axi</td>
<td>CRTL_BUS</td>
<td>return void</td>
</tr>
<tr>
<td>s_axi_CRTL_BUS_BVALID</td>
<td>out</td>
<td>1</td>
<td>s_axi</td>
<td>CRTL_BUS</td>
<td>return void</td>
</tr>
<tr>
<td>s_axi_CRTL_BUS_BREADY</td>
<td>in</td>
<td>1</td>
<td>s_axi</td>
<td>CRTL_BUS</td>
<td>return void</td>
</tr>
<tr>
<td>s_axi_CRTL_BUS_BRESP</td>
<td>out</td>
<td>2</td>
<td>s_axi</td>
<td>CRTL_BUS</td>
<td>return void</td>
</tr>
<tr>
<td>ap_clk</td>
<td>in</td>
<td>1</td>
<td>ap_clk</td>
<td>s_axi</td>
<td>return value</td>
</tr>
<tr>
<td>ap_n</td>
<td>in</td>
<td>1</td>
<td>ap_n</td>
<td>s_axi</td>
<td>return value</td>
</tr>
<tr>
<td>interrupt</td>
<td>out</td>
<td>1</td>
<td>interrupt</td>
<td>s_axi</td>
<td>return value</td>
</tr>
</tbody>
</table>
Here as shown in the Figure 3.15, function return was declared as a axi-4 Lite interface. These ports are necessary to get access to the IP core from the Zynq. This interface includes ports to start and stop the execution of the IP core, moreover ready, valid are used to know the status of the IP core.

| gradient_TDATA  | in  | 8 axis | gradient_V_data_V pointer |
| gradientVALID  | in  | 1 axis | gradient_V_data_V pointer |
| gradient_TREADY | out | 1 axis | gradient_V_dest_V pointer |
| gradient_TDEST  | in  | 6 axis | gradient_V_dest_V pointer |
| gradient_TKEEP  | in  | 1 axis | gradient_V_keep_V pointer |
| gradient_TSTRB  | in  | 1 axis | gradient_V_strb_V pointer |
| gradient_TUSER  | in  | 2 axis | gradient_V_user_V pointer |
| gradient_TLAST  | in  | 1 axis | gradient_V_last_V pointer |
| edgedir_TDATA   | in  | 16 axis | edgedir_V_data_V pointer |
| edgedir_TVALID  | in  | 1 axis | edgedir_V_data_V pointer |
| edgedir_TREADY  | out | 1 axis | edgedir_V_dest_V pointer |
| edgedir_TDEST   | in  | 6 axis | edgedir_V_dest_V pointer |
| edgedir_TKEEP   | in  | 2 axis | edgedir_V_keep_V pointer |
| edgedir_TSTRB   | in  | 2 axis | edgedir_V_strb_V pointer |
| edgedir_TUSER   | in  | 2 axis | edgedir_V_user_V pointer |
| edgedir_TLAST   | in  | 1 axis | edgedir_V_last_V pointer |
| outStream_TDATA | out | 8 axis | outStream_V_data_V pointer |
| outStream_TVALID | out | 1 axis | outStream_V_dest_V pointer |
| outStream_TREADY | in  | 1 axis | outStream_V_dest_V pointer |
| outStream_TDEST | out | 6 axis | outStream_V_dest_V pointer |
| outStream_TKEEP | out | 1 axis | outStream_V_keep_V pointer |
| outStream_TSTRB | out | 1 axis | outStream_V_strb_V pointer |
| outStream_TUSER | out | 2 axis | outStream_V_user_V pointer |
| outStream_TLAST | out | 1 axis | outStream_V_last_V pointer |
| outStream_TID   | out | 5 axis | outStream_V_id_V pointer |

Figure 3.16: RTL Ports of Generated RTL Design axi stream

Gradient, magnitude, and outstream are declared as axi4 streaming data types. Here as shown in the figure 8 bit axis gradient_TDATA and 16 bit edgedir_TDATA are two input data stream. outstream_TDATA is 8 bit output data stream moreover TLAST signal port is useful for the DMA operations to notify DMA that stream data transfer is done. In addition to this Tvalid and Tready has to be high for the data transfer.

3.4 Hysteresis Function Implementation:

Hysteresis function has three arguments two as an input and one output argument.

1. Output of the non maximum suppression function is of axi4 stream data type
2. 8 bit unsigned for lower threshold
3. 8 bit unsigned for upper threshold
4. Output of the function which has axi4 stream data type
Figure 3.17: C++ Snippet for Hysteresis Function

As shown in the Figure 3.17 input stream is taken as an axi4 stream with unsigned 8-bit data width and same data type is used for the output stream of the function. Instantiation of line buffer of the size 240x3 and memory window of size 3x3 is shown. Line buffer and window operation is same as the previous function. Below hysteresis function is described on the memory window.

```c
void doHyst(lvl::streamIn_t<32, 64> input, lvl::streamOut_t<32, 64> output, unsigned char lowerThreshold, unsigned char upperThreshold)
{
    // line buffer and memory window
    lvl::memBuffer<128, 32, int, unsigned> lineBuffer;
    lvl::memWindow<9, 9, unsigned> memoryWindow;

    int pixel(0);
    pixel(0) = input(0);
    pixel(1) = input(1);
    pixel(2) = input(2);
    pixel(3) = input(3);
    pixel(4) = input(4);
    pixel(5) = input(5);
    pixel(6) = input(6);
    pixel(7) = input(7);
    pixel(8) = input(8);
    pixel(9) = input(9);

    if (pixel(0) > upperThreshold)
    {
        pixel(0) = 255;
    }
    else if (pixel(0) < lowerThreshold)
    {
        pixel(0) = 0;
    }
    else
    {
        int high = 0;
        int low = 0;

        for (int i = 0; i < 9; i++)
        {
            if (pixel(i) > upperThreshold)
            {
                high = 1;
            }
            else if (pixel(i) < lowerThreshold)
            {
                low = 1;
            }
        }

        if (high == 1)
        {
            pixel(0) = 255;
        }
        else
        {
            pixel(0) = 0;
        }
    }

    output(0) = pixel(0);
    output(1) = pixel(1);
    output(2) = pixel(2);
    output(3) = pixel(3);
    output(4) = pixel(4);
    output(5) = pixel(5);
    output(6) = pixel(6);
    output(7) = pixel(7);
    output(8) = pixel(8);
}
```

Figure 3.18: Flowchart of Hysteresis Algorithm

Figure 3.18 shows if the value of the pixel at location(1,1) in the window is higher than the higher threshold, then it is set to 255. If its value is less than lower threshold, then that value is set to 0.
Now, if that value is between higher and lower threshold then all the other pixels in the window is compared with the higher threshold, and if any of them is greater than the higher threshold, then the value of pixel(1,1) is set to 255.

3.4.1 Synthesise Result of Hysteresis:

Synthesis result of this function is described below.

![Figure 3.19: Timing Estimation](image)

<table>
<thead>
<tr>
<th>Name</th>
<th>BRAM_18K</th>
<th>DSP48E</th>
<th>FF</th>
<th>LUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>499</td>
</tr>
<tr>
<td>FIFO</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Instance</td>
<td>0</td>
<td>-</td>
<td>94</td>
<td>112</td>
</tr>
<tr>
<td>Memory</td>
<td>3</td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Multiplexer</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>300</td>
</tr>
<tr>
<td>Register</td>
<td>-</td>
<td>-</td>
<td>450</td>
<td>-</td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
<td>0</td>
<td>544</td>
<td>911</td>
</tr>
<tr>
<td>Available</td>
<td>280</td>
<td>220</td>
<td>106400</td>
<td>53200</td>
</tr>
<tr>
<td>Utilization (%)</td>
<td>1</td>
<td>0</td>
<td>&lt;.0</td>
<td>1</td>
</tr>
</tbody>
</table>

![Figure 3.20: Resource Utilization](image)

Memory utilization shows that for every row of line buffer individual block RAM with 240 words is implemented.
Here all axi4 stream and axi4 lite return interface will be synthesized same as non maximum suppression. Here scalar 8 bit unsigned input is defined as an axi lite interface.

3.5 Testing and Verification:
OpenCV functions are used to test non maximum suppression and hysteresis algorithm. First of all axi stream variable is populated with image input. That variable is then passed to the image processing function to test that function. Then using OpenCV functions output stream data are converted into the image. Here minMaxTdx function is used to find minimum and maximum element value in the array. Maximum value from this function is used to convert an array into the matrix using the convertScaleAbs function. The imwrite function is used at last to convert the matrix into the image and store it at desired location.

3.5.1 C/RTL Simulation Output:
Now, RTL/C co simulation is used to generate and compare the output of the implemented hardware with the C simulation. RTL/C co simulation uses the same input stimulus used by C simulation. Here is the output image from the RTL/C co simulation.
Here as shown in the Figure 3.22 there are some weak edges in the picture so next stage is used to extract the true edges from this image. This image is given as an input to the hysteresis function.

Now, simulation output of hysteresis is shown below. Which is the final output of the canny edge detection algorithm.

Now, this synthesized output is exported as an IP to the Vivado to use it with the Zynq to implement this accelerator on the Zedboard.
4. SYSTEM ON CHIP RESOURCES

4.1 Zynq7 Processing System

Zynq7 processing system consists dual-core ARM Cortex A9[11] processor and Xilinx programmable logic. It consumes very low power with the benefit of high performance. At the time of booting, the processor system is booted and then the programmable logic unit is booted. It can be configured totally, partially or dynamically. Different power domain is given for the Zynq7 Soc unit. Block diagram illustrates all the functional units of the Zynq processing system.

As shown in the block diagram both PL and Ps has different power management unit so use of either one is possible for the power management.

- **PS(Processing System) blocks**
  - APU(Application Processor Unit)
  - Memory interface
  - Interconnect
  - IOP(Input/Output peripherals)

- **PL(The Programmable Logic)**

Description of processing system is given below.

1. Application Processor unit:
   - Two ARM cortex A9 processor
● NEON 128b co processor
● Level 2 cache with parity -512 kb
● Timers and watchdogs

SCU(snoop control unit) is provided for maintaining level 1 and level 2 coherency. In addition to this ACP(accelerator coherency port) is given to slave PS to master programmable Logic unit. ACP port can also access L2 cache, On-Chip Memory, and 64b AXI slave[11].

256kb of dual port On-Chip Memory is provided with parity support. For data transfer between any memory in the system, four channel DMA controller is provided in the PS, and four-channel DMA controller is provided for data transfer to and from memory and PL[11].

2. Memory interface:
- DDR controller
- SMC(static memory controller)
- SPI-Quad Controller
- Transaction Scheduler and DDR controller

3. Input output peripherals
- GPIO
- Two gigabit ethernet controller
- USB 2.0
- Two SDIO/SD controller
- Master and Slave SPI controllers
- Two I2C controller

Description for Programmable Logic is described below.
- CLB-configurable logic blocks: look up table with 6 input, adders,
- Dual port block RAM:36kb up to 72 bit wide
- DSP48E1 Digital signal Processing: high resolution 48 bit signal processor
- Clock management unit: buffers for high speed and low skew clock
- Configurable input outputs: lowest power and high speed input output
- High performance low power gigabit transceivers: transceivers with the speed up to 6.25 Gb/s
- XADC- analog to digital converter: up to 17 analog input with on chip temperature and power supply sensor
- PCI express

Clock and Reset system:
Dedicated 33.3333Mhz clock is provided for the processor subsystem moreover 100 Mhz clock is provided for the PL part. Physically spread out frequency programmable clocks are provided for the PL part.
Reset for the PS resets all the debugging sessions and configuration moreover system reset wipes out all the memory contents. System reset is completely independent of the PL portion.

4.2 Timer
Cortex A9 Processor timer: it has the private 32-bit timer and 32-bit watchdog timer, and both processors use the simple 64-bit global timer. All these timer works on half of the CPU frequency.
- Global Timer:
  Global counter has the frequency which is half of the clock frequency and has auto increment of 64 bit.
- Private Timer:
  Two modes are given for private mode. One is single shot and the second is auto reload mode. Interrupt is generated when 32-bit timer value reaches zero. The private timer can be configured for starting value.
- System watchdog timer:
  To handle signal catastrophic system failure watchdog timer is used. Watchdog timer can take input from PL bus clock or external or internal clock moreover it has 24 bit internal counter. At a reset it can generate the interrupt or reset the system.
- Triple Timer Counters (TTC):

Independent timer counter has 16 bit up-down counter and 16 bit prescaler. Internal or external clock can be the input for this timers and all of them have individual interrupts.

4.3 UART

UART is a full duplex and UART controller supports large-scale baud rate and full duplex communication. Two 64 byte FIFO are used for transmission and reception of the data. UART controller can control serialization and deserialization of the transmitter FIFO and receiver FIFO. Status reg, interrupt status reg, and modem status register are applied to read states of FIFOs, modem signal and other controller function. The UART functions are controlled by mode register and configuration register.

As shown in the figure UART controller and APU communicate using APB bus. Data arrived from memory is stored in TxFIFO and received data is stored in RxFIFO. It operates on 600,9600,28800,115200,460800,921600 baud rates and also generates this baud rates using UART reference clock.

![UART Function Diagram](image)

**Figure 4.2: UART Function Diagram[11]**

Data width for transmitter and receiver FIFO is 8 bit. It can operate on one of the four mode, Normal mode, local loopback mode, automatic mode and remote loopback mode.
4.4 DMA Controller

To transfer a large amount of data without any interference of the processor DMA controller is used. Data transfer can be between anywhere system memories and PL peripherals[12]. It uses 64 bit AXI master interface with clock_2x frequency. DMA controller contains eight channels, and all of them are configurable. DMA engine is used to push memory request for read or write. All the status and control register are accessible through software. Below figure 4.3 shows the block design of the DMA controller.

![Figure 4.3 : Block Diagram of DMA Controller](image)

DMA transmission is controlled along with processing the program code by DMA transmission execution engine. Instruction cache stores instruction temporarily. Read and write instruction is used as a storage buffer for instruction before start any transmission on AXI and multi-channel data FIFO is used as the storage buffer for read and write during DMA transmission. DMA to PL peripheral interface supports asynchronous request from PL peripherals.
4.5 AXI Interface

Advanced Extensible Interface protocol is mainly used in Xilinx IPs. This protocol is a part of ARM advance microcontroller bus architecture(AMBA)[13]. AXI4[13] is advance version of AMBA.

Overview of AXI:

AXI interface exchange information between AXI slave and AXI master peripherals. Memory mapped AXI slave and AXI master blocks are connected through structure called AXI interconnect. Xilinx AXI interconnect consist of five channels.

- Read address channel
- Read data channel
- Write data channel
- Write address channel
- Write response channel

![Figure 4.4: Architecture for Write Channel][13]

This interface works bidirectionally and can have different data width. However it supports only 256 data transfer in a burst mode. only AXI4 stream supports burst data
transfer while AXI4 lite can have only one data per transaction. Bidirectional data transfer is achievable because of separate data and address connection. To maintain timing closure it supports different pipeline stages moreover both AXI master and AXI slave has different clock. There are three kind of AXI4 interface. AXI4, AXI4-lite and AXI4 stream.

4.6 ZedBoard Hardware

There are so many board peripherals available on the Zedboard. Some of the peripherals are accessible by processing system while some of them are available only for programmable logic. Oscillators to generate clocks, UART, DDR peripherals are used in this project. Generated bit stream is dumb into FPGA using micro USB cable via JTAG.

![Figure 4.5: Zedboard Block Diagram](image)

Now, next chapter describe the implementation of canny edge detection algorithm on Zedboard.
5. IMPLEMENTATION OF THE DESIGN ON SoC

5.1 Project Setup:
Vivado Design Suite[14] 2015.2 tool is adopted to implement non maximum suppression algorithm and hysteresis algorithm. Zynq 7020(xc7z020clg484-1) provided on Zedboard[14] is used for target project. Here in the figure 5.1 project setting is shown for the project.

![Figure 5.1: Project Setting](image)

5.2 Block Diagram:
Figure 5.3 presents the block diagram of the project which consists of AXI interconnect, Zynq & processing system, Axi DMA controller, Axi memory interconnects, AXI timer, Processor system reset and IP generated from the vivado HLS is instantiated.

![Figure 5.2: Address Editor](image)

All the IPs connected to the Zynq are memory mapped, and their address is shown in the address editor which is shown in figure 5.2. Address for the generated IP nonmaximum suppression is 0x43c1_0000 so this going to be offset to operate on this IP from the Xilinx SDK. Programmable System side works at 33.33 Mhz clock and programmable logic side works at 100Mhz.

Here Processor System Block Ip is generated by automation. Its goal is to reset the peripheral and interconnects. Peripheral_aresetn port is connected to all the
peripherals for reset while interconnet_aresetn is used to reset axi interconnects for memory and processing system.

Figure 5.3: Block Diagram Implementation of Non-maximum Suppression

Zynq processing system configuration is shown in the figure. The parts which is used in the system are marked as a green.
UART 0 is useful for the communication between Zynq processing system and the Host computer. AXI high-performance slave interface is enabled to communicate with the DMA controller moreover AXI general purpose master interface is connected using the AXI interface.
Here generated IP from HLS tool is imported from the IP repository manager. IP to perform nonmaximum suppression is named as doMaxSup. Detailed IP block is presented in this figure 5.4. Data Width of edgedir is 16 bit and 8 bit for gradient and outsStream. All the input signal will be connected to master memory mapped stream port of the DMA controller, and output port is connected to slave stream to memory mapped port of the DMA controller to write back data to the DDR. Here DMA controller act as a slave.

Figure 5.4: Zynq Block Design

Figure 5.5: IP Core for Non-maximum Suppression
DMA controller is added to transfer the data between PL to PS without any interruption from the processor. Here two DMA controller is instantiated for the data transfer between AXI-4 streaming IP core and AXI-4 memory mapped. Configuration for the DMA controller is shown in figure 5.6 and figure 5.7.

Axi_dma_0 is set to stream data width of 16 bit with read and enabled write channel to write back to the DDR. Axi_dma_1 is configured to stream data width of 8 bit and write channel is disabled because doMaxSup IP has only one output port and which is
connected to memory mapped stream port of the dma_1. Both DMA controllers are set to allow the unaligned transfer.

Axi_timer Ip block configuration is set to 64-bit mode for higher accuracy. This Ip block is used to calculate total processing time.

5.3 Block Diagram of Hysteresis Algorithm Implementation

Implementation of hysteresis image processing algorithm's block diagram is almost same as presented in Figure 5.8. Exported IP from vivado HLS is instantiated using vivado IP repository manager. Moreover, this design will have only one DMA controller as doHyst Ip has only one input and one output stream, data width for read and write set to 8 bit.axi timer ip is connected to the Zynq via axi interconnect. Two axi interconnect is used, one is for axi peripherals, and one is for memory interconnect. Axi memory interconnect used for connecting DMA to zynq high-performance bus for stream data transfer.

Figure 5.8 : Block Design for Hysteresis Algorithm
Scaler input higher threshold and lower threshold are implemented as axi lite interface. It is connected with the master interface of AXI interconnect port and outstream has 8-bit stream data width with all the axi stream side channels. Input stream port nonMax also has 8-bit data width. Data flows from master memory mapped slave to IP for the nonMax input port and from outputstream slave stream to memory mapped as shown in the figure 5.8.

5.4 Resource Utilization

Resource utilization of both design is given below.
5.5 Hardware Setup

Micro USB cables is used to interact with Zedboard and host PC. Hardware setup is shown in the Figure 5.11. 12V power cable is connected to power up the board. To communicate between the device and the PC micro USB port is connected to the UART of the Zedboard.

Figure 5.11: Utilization of Non-maximum Suppression

Figure 5.12: Hardware Setup
5.6 Xilinx SDK

Next step is to generate bit stream from the implemented design and create hdl wrapper. This design is then exported to the Xilinx SDK with the bitstream.

The program which is going to be executed by the Zynq is stored in DDR. Here image file is taken as an array and will be transferred to the Ip core using DMA transmission. The output from the image processing IP core is stored to at the particular DDR memory location. Drivers to use generated IP are already produced by the Vivado HLS which can be used to start, stop or to get the status of the IP.

Here are steps to use generated Ip with the Zynq are provided.

- First of all Ip for the nonmaximum suppression or hysteresis are initialized
- After that IP core for DMA controller is initialized and timer is also initialized to get the total elapsed time
- Now, IP for hysteresis and nonmaximum suppression is started.
- Start the timer
- Flush the cache to erase all unwanted garbage data at particular locations by giving address and length of the memory
- Simple transfer from and to device and DDR memory and start transmission
- Check the DMA status to know the transmission is done or not
- Invalidate the cache for the received buffer
- Stop the timer
- Output image is saved in the DDR

Now, XMD console is used to fetch the data from the DDR memory and store it into a .txt file. After this using Matlab we can generate an image from the array of data received from the DDR memory.

Here code to extract data from the memory is shown.

```
Accepted a new GDB connection from 127.0.0.1 on port 56441
Software Breakpoint 0 Hit, Processor Stopped at 0x00010000
Software Breakpoint 5 Hit, Processor Stopped at 0x000100cb
set logfile [open "C:\\temp\\VivadoHLS\\CennyHardware\\CennyLog.txt" "w"]
file4dfad00
XMD% puts $logfile [mrd 0x14000000 76800 b]
XMD% close $logfile
XMD% 
```

Figure: 5.13 XMD Console

Here 0x1400000 shows the location from where the data need to be extracted and 76800 show the number of memory location need to be read from the memory.
Console shows the time taken for the whole operation. Output of both hysteresis and nonmaximum suppression is given here.

Figure 5.14: Output Console for Non-maximum Suppression
Here as shown in the figure total execution time taken by the whole process is 0.01 seconds.

Figure 5.15: Output Console for Hysteresis
Total execution time taken is only 5ms and output image is stored in DDR location start at 0x01300000.

Output Image:

Figure 5.16 Non-max suppression Output  Figure 5.17 Hysteresis Output
This images are generated using received data by the DDR from output stream of IP core of non maximum suppression and hysteresis. Output from the hysteresis is the final output from the canny edge detection algorithm.
6. CONCLUSION

6.1 Conclusion and Future Work
As per the analysis of chapter 3 and chapter 5, nonmaximum suppression and hysteresis block of the canny edge detection algorithm is successfully implemented in HLS and designed stream accelerator generated from Vivado HLS is implemented on Zynq SoC and appropriate output is achieved. Apart from this, different skills and the knowledge of different tools is obtained, which can be used to extend this project for real-time image processing. Moreover, during the development and verification of hardware accelerators and designing complex designs on SoC different challenges have been seen and solved. Skills and knowledge obtained in this field can be applied in future projects.

Enhancement of current output can be done using floating point data types. The different optimization techniques can be implemented for more optimum results. Moreover, this can be applied to pre-processing block in real-time image processing algorithm.
REFERENCES


7. Introduction to FPGA design with vivado HLS UG998 (v1.0) July 2, 2013

8. OpenCV applications using Zynq7000


12. AXI DMA PG021 October 5, 2016

13. How to Use the Three AXI configurations By Xilinx,March 7, 2011

14. The Zynq Book
APPENDIX A

Source Codes:

```cpp
/#*.
-- Engineer: Darpan Daru
-- Create Date: 26/07/2016
-- File Name:nonmaximumsupp.cpp
-- Description: this file performs non maximum suppression
-- gets the input as gradient magnitude and gradient direction
*/
#include "core.h"
void 
doMaxSupp(hls::stream<unsigned int>&gradient,hls::stream<int>&edgedir,hls::stream<uint>&outStream)
{
    #pragma HLS INTERFACE axis port= gradient
    #pragma HLS INTERFACE axis port=edgedir
    #pragma HLS INTERFACE axis port=outStream
    #pragma HLS INTERFACE s_axilite port=return bundle=CXL BU

    // Line buffer for gradient and direction of each pixel
    // line buffer size 3x240
    hls::LineBuffer<BUFFER_SIZE,IMAGE_WIDTH,unsigned char> gradient_buffer;
    hls::LineBuffer<BUFFER_SIZE,IMAGE_WIDTH,signed short> edgedir_buffer;

    // window size 3x3
    hls::Window<BUFFER_SIZE,BUFFER_SIZE,unsigned char> Gradwindow;
    hls::Window<BUFFER_SIZE,BUFFER_SIZE,signed short> Edgedirwindow;

    // variable for row and col
    int total_col = 0;
    int total_row = 0;
    int window_shift = 0;
    // variable for wait time
    int wait_time = 0;
    int wait_ticks = 41;
    int pixel_sent=0;
    // axi stream side channel data
    stream 8 uint GradientSideChannel;
    stream 16 int EdgedirSideChannel;
    stream 8 uint dataOutSideChannel;

    // Iterate on all pixel
    for (int pixel_in = 0; pixel_in < (IMAGE_WIDTH*IMAGE_HEIGHT); pixel_in++)
    {
        #pragma HLS PIPELINE
        // read data from the input stream
        GradientSideChannel = gradient.read();
        EdgedirSideChannel = edgedir.read();

        // Get the pixel data
        unsigned char gradientIn = GradientSideChannel.data;
        signed short directionIN = EdgedirSideChannel.data;

        // Put data on the LineBuffer by shifting it up
        gradient_buffer.shift_up(total_col);
        // put value on bottom row
        gradient_buffer.insert_top(gradientIn,total_col);
        edgedir_buffer.shift_up(total_col);
        // will put the value on bottom row
        edgedir_buffer.insert_top(directionIN,total_col);

        // Put data on the window and multiply with the kernel
        for (int window_row = 0; window_row < BUFFER_SIZE; window_row++)
        {
```
for (int window_col = 0; window_col < BIFFER_SIZE; window_col++)
{
    // window_col + pixConvolved, will slide the window ...
    unsigned char Gradval =
        gradient_buffer.getval(window_row+,window_col+window_shift);
    signed short Dirval = edgedir_buffer.getval(window_row,window_col+window_shift);
    // populate window with the data from the line buffer
    Gradwindow.insert(Gradval,window_row,window_col);
    Edgwindow.insert(Dirval,window_row,window_col);
}
@end of for window_col
@end of for window_row

// no calculation for image boundaries
unsigned char final_pixel = 0;
if (!total_row || total_col >= BIFFER_SIZE-1)
{
    final_pixel = nonMaxSup(&Gradwindow,&Edgwindow);
    window_shift++;
}

// total row and total columna increment
if (total_col < IMAGE_WIDTH-1)
{
    total_col++;
}
else
{
    // New line
    total_col = 0;
    total_row++;  
    window_shift = 0;
}

// delay for line buffer
// (240+2) + 3)/2 = 241
// put 241 zeros for wait time
// put the data on the side channel
// list for dma operation
wait_time++;  
if (wait_time > wait_ticks)
{
    dataOutSideChannel.data = final_pixel;
    dataOutSideChannel.keep = GradientSideChannel.keep;
    dataOutSideChannel.strb = GradientSideChannel.strb;
    dataOutSideChannel.user = GradientSideChannel.user;
    dataOutSideChannel.last = 0;
    dataOutSideChannel.id = GradientSideChannel.id;
    dataOutSideChannel.dest = GradientSideChannel.dest;
    // output data on the output stream
    outStream.write(dataOutSideChannel);
    pixel_sent++;
}
@end of for pixel in

for (wait_time = 0; wait_time < wait_ticks; wait_time++)
{
    dataOutSideChannel.data = 0;
    dataOutSideChannel.keep = GradientSideChannel.keep;
    dataOutSideChannel.strb = GradientSideChannel.strb;
}
dataOutsideChannel.user = GradientSideChannel.user;
if (wait_time < wait_ticks - 1)
dataOutsideChannel.last = 0;
else
dataOutsideChannel.last = 1;
dataOutsideChannel.id = GradientSideChannel.id;
dataOutsideChannel.dest = GradientSideChannel.dest;
// Send to the output stream
outStream.write(dataOutsideChannel);
}

// doMaxSupression

// function for calculating non maximum suppression
unsigned char nonMaxSup
(unsigned char *Gradwindow,
unsigned char *Edgwindow)
{
    // gradient magnitude at pixel(1,1)
    unsigned char pixel = Gradwindow->getval(1,1);
    // direction at pix(1,1)
    signed short tan = Edgwindow->getval(1,1);
    short tan_direction = 0;
    // categorize direction
    if ( ( (tan < 22.5) || (tan > -22.5) ) || (tan > 157.5) || (tan < -157.5) )
tan_direction = 0;
    if ( ( (tan > 22.5) || (tan < -22.5) ) || (tan < -112.5) || (tan > 112.5) )
tan_direction = 45;
    if ( ( (tan > 67.5) || (tan < -67.5) ) || (tan < -112.5) || (tan > 112.5) )
tan_direction = 90;
    if ( ( (tan > 112.5) || (tan < 157.5) ) || (tan < -22.5) || (tan > -67.5) )
tan_direction = 135;

    // edge thinning process
    switch(tan_direction)
    {
    case 0:
        if (pixel<Gradwindow->getval(1,1) || pixel<Gradwindow->getval(1,2))
        pixel=0;
        break;
    case 45:
        if (pixel<Gradwindow->getval(0,0) || pixel<Gradwindow->getval(2,2))
        pixel=0;
        break;
    case 90:
        if (pixel<Gradwindow->getval(0,1) || pixel<Gradwindow->getval(2,1))
        pixel=0;
        break;
    case 135:
        if (pixel<Gradwindow->getval(0,2) || pixel<Gradwindow->getval(2,0))
        pixel=0;
        break;
    }
    // end of switch

    return pixel;
}

// end of nonMaxSup
// declaration of stream data
typedef
// declaration of functions, output path
#include "hls_video.h"
#include <ap_axi_data.h>
#define IMAGE_WIDTH 240
#define IMAGE_HEIGHT 320
// 3x3 kernel
#define BUFFER_SIZE 3
// Image file path
#define IMAGE_LOCATION_OUT "C:\\temp\\VivadoHls\\NonMaxWithMatlab\\NonMaxMat\\final_lena.bmp"
// axi stream side-channel (TLAST,TKEEP,TUSR,TID)
typedef ap_axiu<8,2,5,6> stream_8_uint;
typedef ap_axi<8,2,5,6> stream_16_int;
// Our IP core
void
dcMaxSupp(hls::stream<stream_8_uint>&gradient,hls::stream<stream_16_int>&edgedir,hls::stream<stream_8_uint>&outStream);
unsigned char nonMaxSupp(hls::Window<BUFFER_SIZE,BUFFER_SIZE,unsigned char> *Gradwindow,
hls::Window<BUFFER_SIZE,BUFFER_SIZE,short> *Edgewidth);
#include "core.h"

void doDetection(hls::stream<stream 8 uint>& nonMax, hls::stream<stream 8 uint>& outStream, unsigned char LowerThreshold, unsigned char UpperThreshold)
{
    #pragma HLS INTERFACE s_axilite port=UpperThreshold bundle=THRESHOLD_BUS
    #pragma HLS INTERFACE s_axilite port=LowerThreshold bundle=THRESHOLD_BUS
    #pragma HLS INTERFACE axis port= nonMax
    #pragma HLS INTERFACE axis port= outStream
    #pragma HLS INTERFACE s_axilite port=return bundle=RTL BUS
    // line buffer for gradient and direction of each pixel
    // linebuffer size 3x240
    hls::.lineBuffer<BUFFER_SIZE, IMG_WIDTH, OR_COLS, unsigned char> nonMax_buffer;
    //window size 3x3
    hls::Window<BUFFER_SIZE, BUFFER_SIZE, unsigned char> nonMaxWindow;
    // variable for row and col
    int total_row = 0;
    int total_col = 0;
    int window_shift = 0;
    // variable for wait time
    int wait_time = 0;
    int wait_ticks = 41;
    int pixel_cnt = 0;
    // axi stream side channel data
    stream 8 uint nonMaxSideChannel;
    stream 8 uint dataOutSideChannel;
    // Iterate on all pixel
    for (int pixel in = 0; pixel_in < (IMAGE_WIDTH*IMAGE_HEIGHT); pixel_in++)
    {
        #pragma HLS PIPELINE
        // Read and cache (Block here if FIFO sender is empty)
        nonMaxSideChannel = nonMax.read();
        // Get the pixel data
        unsigned char nonMaxIn = nonMaxSideChannel.data;
        // Put data on the LineBuffer
        nonMax_buffer.shift_up(total_col);
        nonMax_buffer.insert_top(nonMaxIn, total_col); // Will put in val[2] of line buffer (Check Debug)
        // Put data on the window and multiply with the kernel
        for (int window_row = 0; window_row < BUFFER_SIZE; window_row++)
        {
            for (int window_col = 0; window_col < BUFFER_SIZE; window_col++)
            {
                // window col + pixConvolved, will slide the window ...
                unsigned char nonMaxval =
                    nonMax_buffer.getval(window_row, window_col + window_shift);
                nonMaxWindow.insert(nonMaxval, window_row, window_col);
                //end of for window col
            }
        }
        // Avoid calculate out of the image boundaries and if we can convolve
        unsigned char final_pixel = 0;
        if ((total_row > BUFFER_SIZE-1) & (total_col > BUFFER_SIZE-1))
        {
            
        }
    }
}
final_pixel = Hysteresis(\&nonMaxWindow, LowerThreshold, UpperThreshold);

    window_shift++;
}

// Calculate row and col index
if (total_col < IMAGE_WIDTH-1)
{
    total_col++;
}
else
{
    // New line
    total_col = 0;
    total_row++;
    window_shift = 0;
}

// Put data on output stream (side-channel\{last\} way...)
wait_time++;
if (wait_time > wait_ticks)
{
    dataOutsideChannel.data = final_pixel;
    dataOutsideChannel.keep = nonMaxSideChannel.keep;
    dataOutsideChannel.strb = nonMaxSideChannel.strb;
    dataOutsideChannel.user = nonMaxSideChannel.user;
    dataOutsideChannel.last = 0;
    dataOutsideChannel.id = nonMaxSideChannel.id;
    dataOutsideChannel.dest = nonMaxSideChannel.dest;
    outStream.write(dataOutsideChannel);
    sentPixels++;
}

} // end of for idxPixel

for (wait_time = 0; wait_time < wait_ticks; wait_time++)
{
    dataOutsideChannel.data = 0;
    dataOutsideChannel.keep = nonMaxSideChannel.keep;
    dataOutsideChannel.strb = nonMaxSideChannel.strb;
    dataOutsideChannel.user = nonMaxSideChannel.user;
    // Send last on the last item
    if (wait_time < wait_ticks - 1)
        dataOutsideChannel.last = 0;
    else
        dataOutsideChannel.last = 1;
    dataOutsideChannel.id = nonMaxSideChannel.id;
    dataOutsideChannel.dest = nonMaxSideChannel.dest;
    // Send to the stream (Block if the FIFO receiver is full)
    outStream.write(dataOutsideChannel);
} // end of final outstream

unsigned char Hysteresis(unsigned char* data, WINDOW BUFFER SIZE, BUFFER SIZE, unsigned char* nonMaxWindow, unsigned char LowerThreshold, unsigned char UpperThreshold)
{
    unsigned char pixel = nonMaxWindow->getval(1,\');

    if(pixel >= UpperThreshold)
        pixel = 255;
    else if (pixel <= LowerThreshold)
        pixel = 0;
    else if (LowerThreshold < pixel < UpperThreshold)
{ if (nonMaxWindow->getval(0,0)>UpperThreshold ||
   nonMaxWindow->getval(0,1)>UpperThreshold ||
   nonMaxWindow->getval(0,2)>UpperThreshold ||
   nonMaxWindow->getval(1,0)>UpperThreshold ||
   nonMaxWindow->getval(1,1)>UpperThreshold ||
   nonMaxWindow->getval(1,2)>UpperThreshold ||
   nonMaxWindow->getval(2,0)>UpperThreshold ||
   nonMaxWindow->getval(2,1)>UpperThreshold ||
   nonMaxWindow->getval(2,2)>UpperThreshold)
   pixel=255;
   else
   pixel=0;
} // END OF ELSE IF (LowerThreshold<pixel<UpperThreshold)
return pixel;
} // end of Hysteresis
// declaration of stream data typedef 
// declaration of functions, output path
#include "his_video.h"
#include <ap_axi_data.h>
define IMAGE_WIDTH 240 
define IMAGE_HEIGHT 320 
// 3x3 kernel 
define BUFFER_SIZE 3 
// Image file path
#define OUTPUT_IMAGE_Q0RGB "D:\\temp\\VivadoHls\\Hysteresis\\LowAndHigh\\Canny_final_45_60.bmp"

// Use the axi stream side-channel (TLAST, TKEEP, TUSR, TID) 
define ap_axiu<8,2,6,8> stream 8 uint; 
// Our IP core
void doHystre(his::stream<stream 8 uint>&nonMax,his::stream<stream 8 uint>&outStream, unsigned char LowerThreshold, unsigned char UpperThreshold); 
unsigned char Hysteresis(his::Window<BUFFER_SIZE, BUFFER_SIZE, unsigned char> *nonMaxwindow, unsigned char LowerThreshold, unsigned char UpperThreshold); 


/ * Empty C++ Application for Zynq
 */

/*
 * nonMaxSupp.cpp on Zynq
 *
 * Created on: Oct 26, 2016
 * Author: dsd41069
 * This program gets gradient and direction from an array stored in DDR, perform nonmaximum suppression and store image in DDR
 */
#include <stdio.h>
#include "XAxiDma.h"
#include "direction.h"
#include "gradient.h"
#include "AxlTimer.h"
#include "xdomaxsupp.h"
#include "xparameters.h"

#define BASE_ADDR 0x001000000
#define TX0 (BASE_ADDR + 0x00100000)
#define RX0 (BASE_ADDR + 0x00400000)
#define TX1 (BASE_ADDR + 0x00300000)

// image size array
#define ARRAY (320*240)
// Memory used by DMA from the DDR
// 32 bit address for 8 bit memory
// pointers to configure DMA
// 32 bit pointers to point 8 bit memory location
unsigned char *TX buffer1 = (unsigned char*) TX1;
// 32 bit pointers to point 8 bit memory location
short *TX buffer0 = (short*) TX0;

unsigned char *RX_BUFFER0 = (unsigned char*) RX0;

// array to store direction and gradient magnitude
short direction_in[ARRAY];
unsigned char gradient In[ARRAY];

XAxiDma dma0;
XAxiDma dma1;
XDomaxsupp nonmaxsupp;

int initialization nonmaxsupp()
{
    int current state;
    XDomaxsupp_Config *nonmaxsupp_conf;
    nonmaxsupp_conf = XDomaxsupp_ConfigReadConfig(XPAR_DOMAXSUPP_0_DEVICE_ID);
    if (!nonmaxsupp_conf)
        printf("configuration error\n");
    current_state = XDomaxsupp_ConfigInitialize(&nonmaxsupp, nonmaxsupp_conf);
    if (current_state != XST_SUCCESS)
        printf("initialization error\n");
    return current state;
}

//end of initialization sobel
// initialization of instance dma0
int initialization dma 0()
{
    XAxiDma_Config *config dma0;
    config dma0 = XAxiDma_ConfigReadConfig(XPAR_AXI_DMA_0_DEVICE_ID);
    XAxiDma_ConfigInitialize(&dma0, config dma0);
    //Disabling the Dma Interrupt
    XAxiDma_DmaDisable(dma0, XPAR_DMA_IRQ_ALL_MASK, XPAR_DMA_DEVICE_TO_DMA);
    XAxiDma_DmaDisable(dma0, XPAR_DMA_IRQ_ALL_MASK, XPAR_DMA_DMA_TO_DEVICE);

    return current state;
}
return XST_SUCCESS;

} //end of int initialization_dma

// initialization of instance dma
int initialization_dma_1() { 
XAXilDma_Config *config_dma1;
config_dma1 = XAXilDma_LookupConfig(XPAR AXI_DMA 1 DEVICE ID);
XAXilDma_ConfigInitialize(config_dma1, config dma1);
//Disabling the DMA Interrupt
XAXilDma锦标Enable(config_dma1, XAXIDMA_IRQ_ALL_MASK, XAXIDMA_DEVICE_TO_DMA);
XAXilDma_interruptEnable(config_dma1, XAXIDMA_IRQ_ALL_MASK, XAXIDMA_DMA_TO_DEVICE);
return XST_SUCCESS;
}

} //end of int initialization_dma

int main() {
// initialize Ip instances for DMA and Non maximum suppression
initialization_dma_0();
initialization_dma_1();
initialization_nonmaxsupp();

// Get the Data from header file
for (int pixel = 0; pixel < ARRAY; pixel++) {
    direction_in[pixel] = Boring[pixel];// direction of each pixel(240x320)
    gradient_in[pixel] = Grading[pixel];// direction of each pixel(240x320)
} //end of for int pixel

AXI_Timer timer; //initialize the Timer

printf("Start Non Maximum Suppression Starts on Hardware"); //Starts the Hardware
XDomaxsupp Start(Nonmaxsupp); timer.startTimer();

// flush the cache for given address
XIL_DCacheFlushRange((u32)direction_in,ARRAY*sizeof(short));
XIL_DCacheFlushRange((u32)RX_BUFFER,ARRAY*sizeof(short));
XIL_DCacheFlushRange((u32)gradient_in,ARRAY*sizeof(unsigned char));
double current_time1 = timer.current_time();
printf("In DDR to DMA transmission starts at : %f sec\n", current_time1);
// simple transfer of image from DDR to IP core
XAXilDma_SimpleTransfer(gdma0,(u32)direction_in,ARRAY*sizeof(short),XAXIDMA_DMA_TO_DEVICE);
XAXilDma_SimpleTransfer(gdma1,(u32)gradient_in,ARRAY*sizeof(unsigned char),XAXIDMA_DMA_TO_DEVICE);
double current_time2 = timer.current_time();
printf("In DMA to DDR transmission starts at : %f sec\n", current_time2);
// receiving data from hardware IP
XAXilDma_SimpleTransfer(gdma0,(u32)RX_BUFFER,ARRAY*sizeof(unsigned char),XAXIDMA_DEVICE_TO_DMA);

//check if the dma0 is busy or not
while(XAXilDma_Busy(gdma0,XAXIDMA_DMA_TO_DEVICE));
while(XAXilDma_Busy(gdma0,XAXIDMA_DEVICE_TO_DMA));
//check if dma1 is busy or not
while(XAXilDma_Busy(gdma1,XAXIDMA_DMA_TO_DEVICE));

//invalidate the cache in receiver buffer of the dma

67
X11 DCacheInvalidateRange((u32)RX BUFFER0,ARRAY*sizeof(short));

timer.stopTimer();
printf("Hardware for Non maximum suppression is stopped"); // stopping the hardware

double total_time = timer.total_time_second();
printf("\nTotal Execution Time : %f Sec\n", total_time);

return 0;

}// end of main file
/ * Empty C++ Application for Zynq *
*/

/*
 * Hysteresis.cpp on Zynq
 * Created on: Oct 26, 2016
 * Author: ds41069
 * This program gets input from nonmaximum suppression stored in DDR
 * perform hysteresis and store image in DDR
 */
#include <stdio.h>
#include "XAxiDma.h"
#include "edgeonCODE.h"
#include "AxiTimer.h"
#include "xdohyst.h"

// base address for the ddr
// Memory used by DMA from the DDR
#define BASE_ADDR 0x01000000
#define TX (BASE_ADDR + 0x00100000)
#define RX (BASE_ADDR + 0x00300000)
#define ARRAY (320*240)

// 32 bit address for 8 bit memory

unsigned char *TX_BUFFER = (unsigned char*) TX;
unsigned char *RX_BUFFER = (unsigned char*) RX;
unsigned char frame[ARRAY];

XAxiDma dma;
int initialization_dma() {
    XAxiDma_Config *CfgPtr;
    CfgPtr = XAxiDma_LookupConfig(XPAR_AXI_DMA_0_DEVICE_ID);
    XAxiDma_CfgInitialize(&dma, CfgPtr);
    //Disabling the DMA interrupt
    XAxiDma_IntrDisable(&dma, XPAR_DMA_IRQ_ALL_MASK, XPAR_DMA_DEVICE_TO_DMA);
    return XST_SUCCESS;
}

} //end of int initialization_dma

XDoxyst hyst;
int initialization_hysteresis()
{
    int status;

    XDoxyst_Config *Hyst_cfg;
    Hyst_cfg = XDoxyst_LookupConfig(XPAR DOXYST 0 DEVICE ID);
    if (!Hyst_cfg) {
        printf("configuration error\n");
    }
    status = XDoxyst_CfgInitialize(&hyst, Hyst_cfg);
    if (status != XST_SUCCESS) {
        printf("initialization error\n");
    }
    return status;
}

} //end of initialization sobel

int main()
{
    initialization_dma(); //Initialize the DMA
initialization hystersys(); //Initialize the Sobel Filter

// Get the data from the header file LenaOnCode.h
for (int pixel = 0; pixel < ARRAY; pixel++) {
    image_in[pixel] = NonMaxImage[pixel];
} //end of for int pixel

AxlTimer timer; //initialize the Timer
printf("Starting Hardware......\n"); //Starts the Hardware

XDocharted Set LowerThreshold(Gayst, 45);
XDocharted_Set UpperThreshold(Chyst, 80);
XDocharted Start(Gyyst);

timer.startTimer();
double current_time1 = timer.current_time();
printf("\n DDR to DMA transmission starts at : %f sec\n", current_time1);

XII_DCacheFlushRange((u32)image_in, ARRAY*sizeof(unsigned char));
XII_DCacheFlushRange((u32)k BUFFER, ARRAY*sizeof(unsigned char));

XAxIDma SimpleTransfer(&dma, (u32)image_in, ARRAY*sizeof(unsigned char), XAXIDMA DMA TO DEVICE);
XAxIDma SimpleTransfer(&dma, (u32)k BUFFER, ARRAY*sizeof(unsigned char), XAXIDMA DEVICE TO DMA);

double current_time2 = timer.current_time();
printf("\n DDR to DMA transmission ends at : %f sec\n", current_time2);

//Check whether the dma is busy or not
while(XAxIDma Busy(&dma, XAXIDMA_DMA TO DEVICE));
while(XAxIDma Busy(&dma, XAXIDMA DEVICE TO DMA));

//Invalidate the cache
XII_DCacheInvalidateRange((u32)RX_BUFFER, ARRAY*sizeof(unsigned char));

timer.stopTimer(); //Stop the timer

printf("Stopping the Hardware......\n"); //Stopping the hardware

double total_time = timer.total time second();
printf("\nTotal Execution Time : %f sec\n", total_time);

return 0;

} //end of main