TRADEOFF FPGA COST OF SOBEL IMPLEMENTATION USING APPROXIMATE DESIGNS

by

Harold Schmoyer, BS

THESIS

Presented to the Faculty of

The University of Houston-Clear Lake

In Partial Fulfillment

Of the Requirements

For the Degree

MASTER OF SCIENCE

in Computer Engineering

THE UNIVERSITY OF HOUSTON-CLEAR LAKE

DECEMBER 2020

TRADEOFF FPGA COST OF SOBEL IMPLEMENTATION USING APPROXIMATE DESIGNS

by

Harold Schmoyer, BS

APPROVED BY

_____

Xiaokun Yang, PhD, Chair

_____

Hakduran Koc, PhD, Committee Member

_____

Jiang Lu, PhD, Committee Member

RECEIVED/APPROVED BY THE COLLEGE OF SCIENCE AND ENGINEERING:

_____

David Garrison, PhD, Interim Associate Dean

_____

Miguel A. Gonzalez, PhD, Dean

Dedication

I would like to dedicate my thesis to my Mom, Dad, Kate, Svetlana, and my best friends in B*A*S*H. Thank you for your constant support, guidance, and love.

Acknowledgements

I would first like to acknowledge the faculty and staff at the University of Houston – Clear Lake. The professors from the computer engineering department have been encouraging, helpful, and kind to me. Thank you to Dr. Xiaokun Yang for sharing your knowledge and patience with me. Thank you to the member of my thesis committee. It has been a pleasure being a student at UHCL from my time I was transferring undergraduate student to now.

I would also like to acknowledge all my friends and family for their continued support and love.

ABSTRACT

TRADEOFF FPGA COST OF SOBEL IMPLEMENTATION USING APPROXIMATE DESIGNS

Harold Schmoyer
University of Houston-Clear Lake, 2020

Thesis Chair: Xiaokun Yang

This dissertation proposes a scalable algorithm for tuning the tradeoff of space, energy, and quality for the implementation on Field-Programmable Gate Array (FPGA). First, an approximate design library including exact and several imprecise designs on adders and subtractors is presented. Using the design library, then six different approximation levels of register-transfer level (RTL) designs on a Sobel edge detection algorithm is created as a case study, in order to demonstrate the advantage of approximate design and develop a specific implementation to save energy and space at the cost of accuracy. Finally, all the designs of the Sobel engine are synthesized in Xilinx Vivado where the comparison between exact design and an array of approximate designs are conducted. Experimental results show that our proposed work achieves the maximum savings of 26% energy, 24% slice count, and 21% of look-up tables (LUTs) at the cost of 1.14% accuracy when images are compared pixel by pixel. In order to further explore the difference between FPGA demonstrations of the exact design and imprecise implementations, all the Sobel

cores are integrated with an image processing platform including OV7670 camera, VGA-enabled monitor, and Xilinx Nexys 4 FPGA to show the quality of edge detection images. By integrating the designs of I2C controller, image capture interface, and the VGA master with our proposed Sobel engines, the simulation with Mentor Graphic ModelSim proves the validity of our proposed work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I:
INTRODUCTION

1.1 Background

Approximate design on integrated circuits and systems is becoming an emerging paradigm, enabling to minimize the area and energy dissipation with specifc quality constraint of the design specifications [1-3]. It leads to many implementations at different levels of accuracy, such as approximate computing to application-specific integrated circuit (ASIC) [4,5], field-programmable gate array (FPGA) [6-8], register-transfer-level (RTL) designs [9,10], and system-on-chips [11-13].

Most prior works to approximate computing focused on the individual designs of fundamental circuits such as adders and multipliers which are widely used by many applications, such as approximate adder for energy-efficient application [14], accuracy configurable adder for area and delay efficient designs [15], approximate multiplier for error-resilient applications [16], and many more [17-19]. By using the approximate circuit into various hardware applications including image and video processing [20-22], machine learning [23-26], and language/voice recognition [27-30], it enables to achieve a large amount of hardware savings in terms of area, latency, and power consumption.

The main challenge of the approximate design in system level is to find an optimal integration of approximate circuits corresponding to various error metrics. In [31] a case study of a color to grayscale converter was presented based on approximate adders and multipliers, showing different energy-quality results on FPGA implementation. Similarly, the image blending and sharpening approaches were applied using approximate adders and multipliers to analyze the image quality metric in [32]. Considering tens to thousands of approximate implementations for each arithmetic

operation, it is impossible to identify the most suitable replacement of arithmetic operations and the combination of approximate circuits to reach the best tradeoff between area-speed-power cost and the quality bound.

Therefore, prior work [33] presented a methodology for selecting and combining suitable approximate circuits from a set of available libraries to generate an approximate accelerator for a given application. Instead of performing full synthesis, the proposed methodology is able to automatically estimate quality of result based on computational models constructed using machine learning methods. Though the proposed work was shown to be quite effective under experimental scenarios, to obtain an accurate performance evaluation without hardware netlist can be a big challenge. In addition, earlier works [34] and [35] proposed a design approach being able to dynamically reconfigure the level of approximation in the hardware based on the input characteristics. As a result, it demonstrated significant energy savings while adhering to the given quality constraints.

This dissertation focuses on finding the optimal combination of the area-speed-power saving by integrating approximate circuits into system-level designs. First, a novel methodology to guide users to select the most suitable replacement of arithmetic operations with approximate circuits is proposed. As a case study, one of the most commonly used applications in image processing – sobel edge detection algorithm, is employed to test the functionality and estimate the hardware performance. Specically, the contributions are below.

- This dissertation first presents several approximate designs on adders and subtractors. In what follows, FPGA design flow including RTL design, verificaiton, synthesis, and implementation are executed to show the quality to resource tradeoff.

- Experimental results show that there can be significant resources savings for any specific applications
- Finally, a video processing platform [26] is empoyed to demonstrate the different designs with different quality bounds to demons mtrate the qualitative performance of each design using the methodologies.

1.2 Related Works

Due to the benefits of parrallel computing and reprogrammability, hardware acceleration with FPGA has been widely used in many applications such as audio processing [37,38], image and video segmentation [39,40], edge computing[41-43], etc. As a case study, in this disseration the sobel edge detection, which is one of the classical and essential algorithms in the field of image and video processing for the extraction of object edges [44], is designed and evaluated with FPGA. In prior work [45] both hardware and software implementations on the sobel engine have been evaluated and compared. By running the OpenCL with a Cortex-A9 embedded core and FPGA, it shows a significant improvement to the computation speed with hardware accelerator (2 ms) compared to the software operation (977 ms). And the resource cost to the FPGA design was very high with 5155 slice of LUTs and 269,814 BRAMs. A similar work presented by [46] achieved 95% reduction on slices of LUTs and 97% reduction on slices of FFs by performing OpenCL in register-transfer (RT) level design using Vivado.

Under this context, the software-hardware co-design platforms such as Xilinx Zynq FPGA were becoming very popular to demonstrate image processing application where the programming system (PS), or namely a CPU host, can be employed to reduce the complexity to a hardware implementation. Mean-time the programming logic (PL) part, or the FPGA, can be applied to accelerate the computation-intensive portions [46, 47]. The main concern of the combined system is that the PS itself would spend a large

amount of FPGA resources in terms of slice count and power dissipation. Additionally the PS-PL interface is based on the very complicated AMBA AXI specication which consumes numerous hardware resources as well.

Another trend to implement the image/video based processing like the edge detection application was mainly based on FPGA High-Level Synthesis (HLS), which offers a methodology for migrating algorithms from higher abstract languages like C or C++ onto the FPGA logic [46,48,49]. For example, in [48] a sobel edge detection core has been implemented by HLS as one of the three benchmarks to evaluate the security advantages and performance over the system. Synthesis results showed a hardware cost of 1,691 slice of LUTs and 3,356 slice of FFs. Likewise, in [49] the HLS based design led to a slices utilization of 1,069 slice of LUTs and 1,173 slice of FFs. For both design in [48] and [49], around ten BRAMs and DSPs were employed as well.

Compared to the HLS based implementation, the application-specific RT-level design can achieve less space utilization and computation latency. For example, in [50] an inexpensive architecture for Sobel edge detection was proposed, showing that only 114 slice of LUTs was spent and 32 us was needed to process 128×128 images at a clock frequency of 500 MHz. As the best of our knowledge, this is the optimal FPGA design on sobel engine which achieved the minimum latency and slice consumption.

The main concern of the above-mentioned works is to statically implement and evaluate the design application in sobel edge detection on FPGA. In this paper, first several imprecise designs are proposed in order to further reduce the FPGA cost in a combination of area-latency-power. Seeing the sobel core design as a case study, a scalable methodology is presented to dynamically find the optimal implementation corresponding to different quality bounds. Though being employed by sobel engine

implementation and evaluation, the proposed approach is expandable to many other designs and applications can be integrated with multiple approximate circuits.

CHAPTER II:
PROPOSED WORK

2.1 Approximate Design

Approximate designs are intended to give an answer that is close to a known exact solution. The exact implementation is a great benchmark for which to compare all designs. This is because when creating an exact solution, the top priority is that the solution is exactly correct. This can leave a lot of potential inefficiencies in the design of any project. These inefficiencies create an opportunity for approximate design.

In general, an approximate design allows engineers to trade accuracy for design cost and resources. In the realm of FPGA based designs this means energy, processing power, and space [1]. Any digital design can be large and complex, but they can all be reduced to some level. The amount of reduction depends on the requirements of the application. An image processing system built as an early detection method for finding cancer in lungs would benefit from more accuracy than processing speed. However, cameras on a toll road reading license plates in order to automate ticketing violators would benefit from increased processing speed. All tradeoff decisions are application and requirement specific.

For this work we are concerned with finding the relationship between quality and space with a case study focused on the Sobel engine. In order to do this, we will utilize Karnaugh maps (K-Maps) to reduce our design from an exact design cost to different levels of approximation. This is a continuation of the work done by [54] for approximate design method.

2.1.1 Design Algorithm

When trying to create an approximate design of any system, it's important to start at the fundamental part, the bit level.

The pseudocode for implementing a system based on hardware to quality tradeoff is outlined in Algorithm 1.

In Procedure #1 the first goal is to map the hardware cost, denoted as C in the algorithm. In line 2 the for loop is set to correspond to each approximation level k. Line 3 shows the calculation for hardware cost $C[k]$ which is the sum of 3 weighted attributes: slice count (represented as $S[k]$), latency (represented as $L[k]$), and power consumption (represented as $P[k]$). The weight of the slice count, denoted as ws, of $S[k] \times ws$ is assigned by the user as an input. This value is related to the user's specifications, requirements, or applications. The weight of the latency, denoted as wl, is also assigned by the user based on the importance of the latency of the design. Lastly, the power consumption weight shows that, if specified last, that the power consumption weight is represented as $1 - ws - wl$. This is simply to state that the combined weight of all desired attributes should sum to 1.

Procedure #2 outlines the method for creating a design that fits the quality bound Q_B. The quality boundary sets the limit to the amount of approximation that can be provided. For instance, if an application requires 75% accuracy to be considered valid, we set our Q_B to 0.75. In order to accomplish this, each component has an approximation sub design inside of it.

Algorithm 1 The Accuracy and Area-Latency-Power Tradeoff Algorithm

Input: $Bit\ Number : i;\ //from\ LSB-0\ to\ MSB-7$
Input: $Application\ quality\ bound : Q_B;\ Set\ of\ application\ quality : Q[j];$
Input: $Set\ of\ slice\ count\ and\ corresponding\ weight : S[k]-ws;$
Input: $Set\ of\ latency\ and\ corresponding\ weight : L[k]-wl;$
Input: $Set\ of\ power\ consumption\ and\ corresponding\ weight : P[k]-(1-ws-wl);$
Output: $Quality\ knob\ configurations : \vartheta_f;\ The\ least\ hardware\ cost\ (C)\ corresponding\ to\ Q.$

procedure #1 - MAPPING HARDWARE COST (C) WITH QUALITY (Q):$(C[k]-P[k])$
  for $(k=0; k<K; k=k+1)$ **do**
    $C[k]=S[k]^{ws}\times L[k]^{wl}\times P[k]^{(1-ws-wl)} <-> Q[k];$
  end for
end procedure
procedure #2 - C-Q CONFIGURATION:$(\vartheta_f,\ C-Q)$
  $Initialize : Q=1, C=0, i=0, j=0$
  while $(Q[i,0]\geq Q_B)$ **do** $//till\ Q\ violation$
    while $(Q[i,j]\geq Q_B)$ **do**
      $j=j+1;\ //increase\ approx\ degree$
    end while
    $i=i+1;\ //computation\ to\ higher\ bit$
  end while
end procedure
procedure #3 - APPROXIMATION LEVEL TO I-TH BIT:
  $\vartheta_f \leftarrow \vartheta_f[j-1];$
  $Q \leftarrow Q[j-1];$
  $C \leftarrow C[j-1];$
  return $\vartheta_f,\ C-Q;$
end procedure

Figure 2.1.1 Accuracy and Area-Latency-Power Tradeoff Algorithm

2.2 MATLAB Design

MATLAB R2020A software was used to quickly create an error detection method for the user and to help aid in the creation of the overall RTL design. MATLAB's language made it easy to create simulated digital components of variable size and test them against all possible values.

MATLAB's base language supports the digital design operations for AND gates (&), OR (|), and NOT (~). These bitwise operations Verilog HDL design more intuitive and transferable.

The guided algorithm aids the user in creating approximate designs that are appropriate for the application. As a case study, the work of Dr. Xiaokun Yang on the Sobel engine was translated into MATLAB.

2.3 Component Designs

The following subsections outline the proposed work of implementing approximate designs of the adder, subtractor, and Sobel engine. The adder and subtractors designs are a continuation of the work set out by Yunxiang Zhang.

Both the adder and subtractor are single bit components. Adders can be cascaded together to form a ripple adder with a carry-in bit. The subtractors can be cascaded to do the same but with a borrow-in bit.

2.3.1 Subtractor

Subtraction inside a digital system can be accomplished with a two's-compliment addition operation. Converting a number into its two's compliment form transforms it into a negative number. This requires inverting all the bits and adding "1" to create the negative number. The two numbers then need to be added together. This would require an n-number of inverters for each bit, an n-number of single-bit adders to add the one, and an additional n-number of single-bit adders to sum the two's complimented number and the original addend.

The Sobel algorithm design presents a unique opportunity for the implementation of a signed subtractor. The Sobel engine is designed to handle grayscale pixel values as inputs. Since the subtractors are only needed at the first stage of the Sobel algorithm we can create our design to reflect the fact that its inputs are always positive. This will save us resources as opposed to using this potentially costly two's compliment method.

Approximations begin at the bit level. Four approximate designs of single bit subtractors was created. The following is the exact design of a single bit subtractor. Difference and borrow-out bits are denoted as Diff and B_{OUT}. X and Y are two single bit inputs being subtracted and B_{IN} denotes the borrow-in bit.

The following is the K-map representation of the exact design of a single-bit signed subtractor.

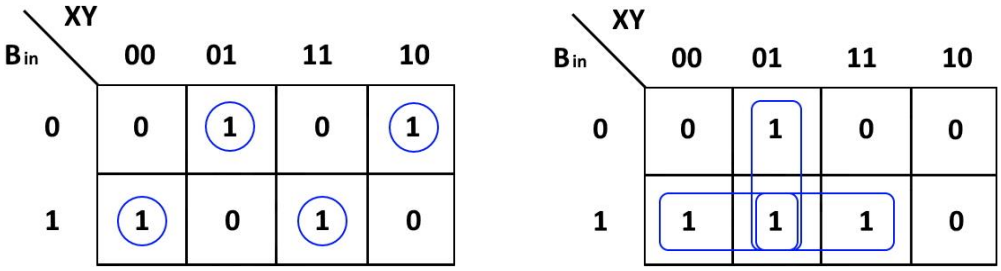


Figure 2.2.1
Subtractor K-Map: Exact Design

$$Diff\ (Exact) = X'YB_{IN}' + XY'B_{IN}' + XYB_{IN} + X'Y'B_{IN}$$

$$B_{OUT}\ (Exact) = X'B_{IN} + X'Y + YB_{IN}$$

K-map representation of these designs aids us in approximation. Changing the result of the K-map allows us to design a system that will give us an output that is close to the result of the exact design. In the following diagram shows the K-map representation of the approximate subtractor *subApprox1*. The bits shown in red represent the bits that were changed from the exact design. In this case the gate $X'Y'B_{IN}$ is eliminated.

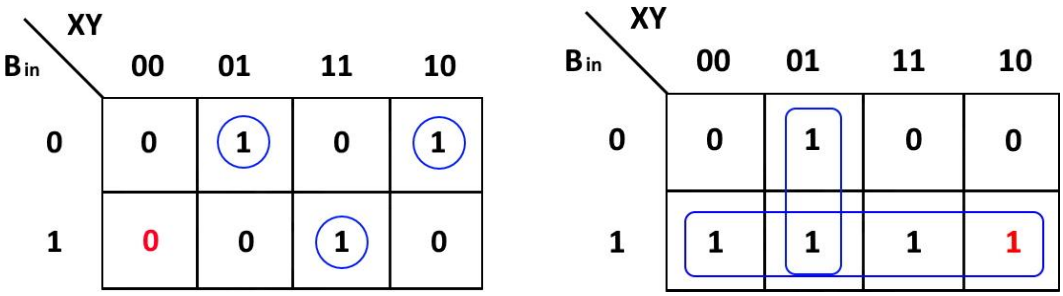


Figure 2.2.2
Subtractor K-Map: subApprox1

$$Diff\ (subApprox1) = X'YB_{IN}' + XYB_{IN} + XY'B_{IN}'$$

$$B_{OUT}\ (subApprox1) = X'B_{IN} + YB_{IN}$$

This is the first subtractor approximation. For the difference the 1 at X'Y' B_{IN} was changed from 0 to 1. The B_{OUT} changed the result where X = 1, Y = 0, and B_{IN} = 1, from 0 to 1. This equation is the result after all the 1s are grouped and reduced.

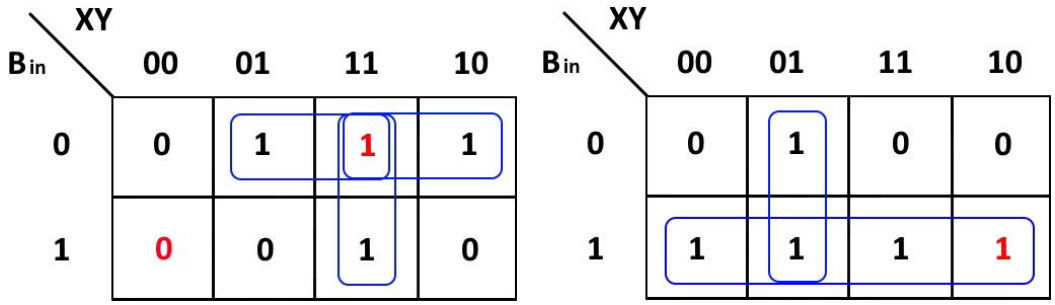


Figure 2.2.3
Subtractor K-Map: subApprox2

$$Diff\ (subApprox2) = YB_{IN}' + XB_{IN}' + XY\,B_{IN} + XY'\,B_{IN}'$$

$$B_{OUT}\ (subApprox2) = X'B_{IN} + YB_{IN}$$

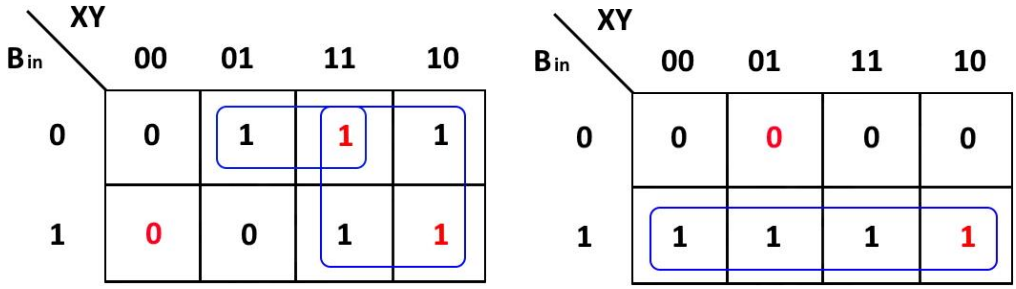


Figure 2.2.4
Subtractor K-Map: subApprox3

$$Diff\ (subApprox3) = X + YB_{IN}'$$

$$B_{OUT}\ (subApprox3) = B_{IN}$$

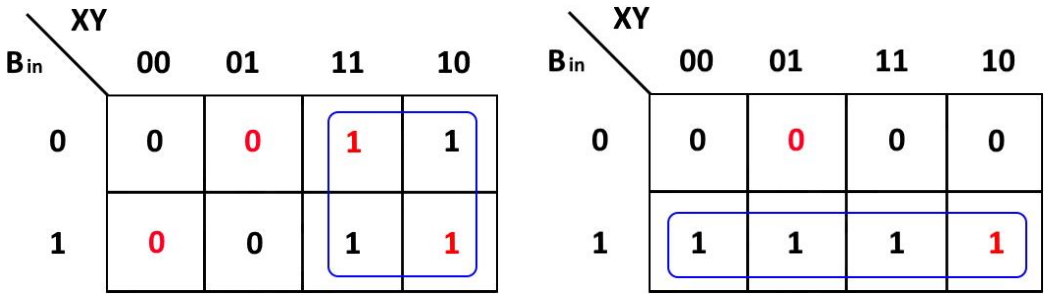


Figure 2.2.5
Subtractor K-Map: subApprox4

$$Diff\ (subApprox4) = X$$

$$B_{OUT}\ (subApprox4) = B_{IN}$$

For each approximate design we are increasing the level of approximation for the output of that particular component. Each reduction in our K-map gives us extra space on the FPGA as well as a reduction in energy usage. In the exact design we are using a total of 16 logic gates. This is the exact design, so it will give us the most accurate result. However, this accuracy comes at a cost of space, latency, and power due to the longer critical path and number of gates. We attempt to strike a tradeoff between accuracy and power by reducing the gate count with the aid of our K-Maps.

2.3.2 Adder

The signed full adder is built by combining an n-number of single bit adders. The sum and carry-out bits are represented by Sum and C_{OUT} respectively. Each bit being summed is represented by X and Y. Lastly, C_{IN} represents the carry-in bit which comes from the carry out.

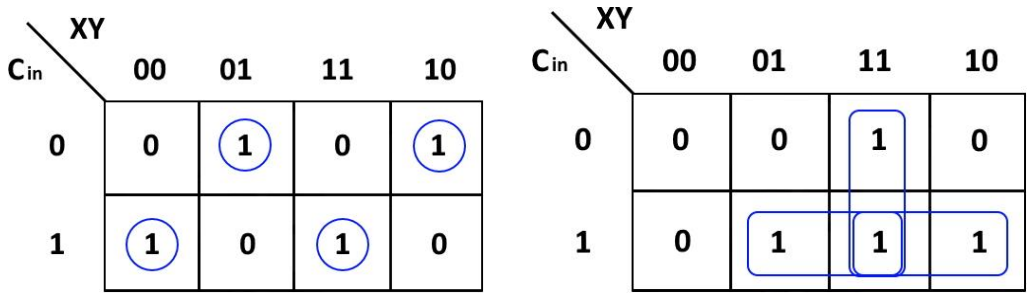


Figure 2.3.2.1
Adder K-Map: Exact Design

$$Sum\ (Exact) = X'Y'C_{IN} + X'YC_{IN}' + XY'C_{IN}' + XYC_{IN}$$

$$C_{OUT} = X'C_{IN} + X'Y + YC_{IN}$$

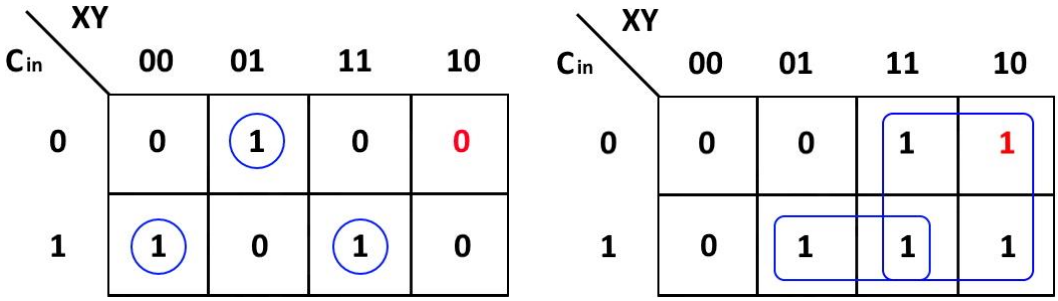


Figure 2.3.2.2
Adder K-Map: AP1

$$Sum\ (AP1) = X'YC_{IN}' + XYC_{IN} + X'Y'C_{IN}$$

$$C_{OUT}\ (AP1) = X + YC_{IN}$$

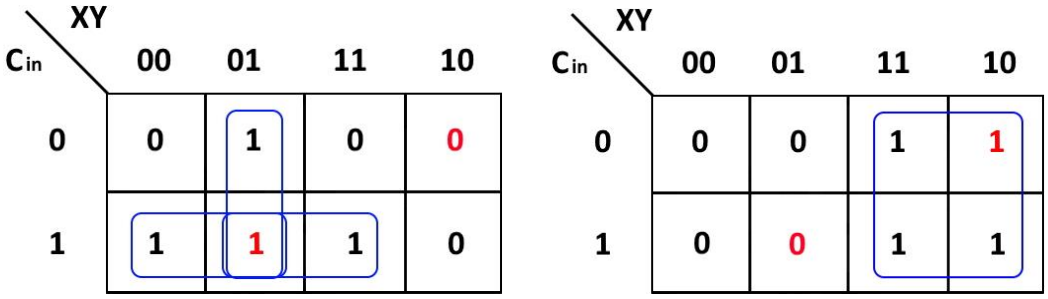


Figure 2.3.2.3
Adder K-Map: AP2

$$Sum\ (AP2) = X$$

$$C_{OUT}\ (AP2) = X'C_{IN} + X'Y + YC_{IN}$$

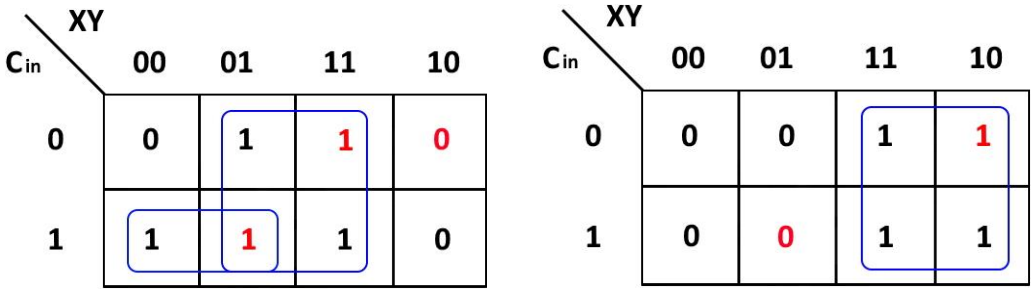


Figure 2.3.2.2
Adder K-Map: AP3

$$Sum\ (AP3) = X$$

$$C_{OUT}\ (AP3) = X'C_{IN} + Y$$

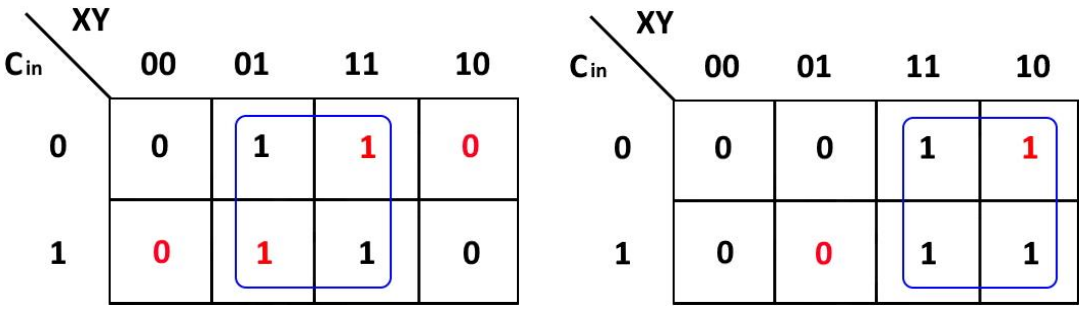


Figure 2.3.2.2
Adder K-Map: AP4

$$Sum\ (AP4) = X$$

$$C_{OUT}\ (AP4) = Y$$

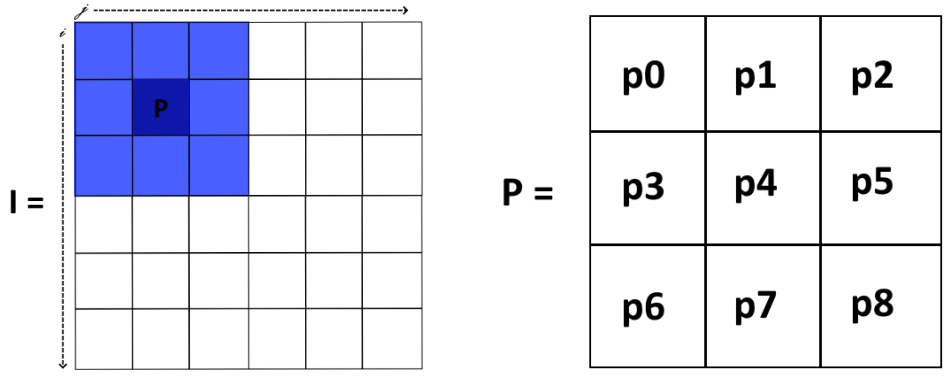
2.4 Sobel Core

The Sobel algorithm is a well-established method of edge detection in the image processing field. Its task is to apply a filter, pixel-by-pixel, that emphasizes perceived edges in an image. In this paper we use it as a case study for demonstrating the accuracy and power tradeoff algorithm discussed in this paper. The algorithm involves the combination of adders, subtractors, and shifters, making it an especially suitable candidate as a case study for the demonstration of the benefits of approximate design.

Figure 2.4.1 (a) shows the full image capture of an input image I represented in matrix form, *i* for rows and *j* for columns. The input image is the resolution of the still image or camera capturing device.

In order to apply the Sobel filter to the entire image, we must first extract a 3×3 kernel to be processed. The kernel to be extracted first is shown in Figure 2.4.1 (a) in blue. This extraction happens pixel-by-pixel within I from P, at the center of the blue kernel.

The extracted pixel kernel, P, as shown in Figure 2.4.1 (b), is separated into pixel 0 through pixel 8 (p0 to p8). This is the coordinate system within the kernel P. The pixels values within P will be the inputs to the Sobel algorithm for calculation.



(a) Input Image (b) Kernel Extraction

Figure 2.4.1
Pixel Extraction

Once the kernel is extracted from the input image into the pixel kernel P it is multiplied it is ready to be processed into its individual components. The two components of the Sobel algorithm are the x-gradient (G_x) and y-gradient (G_y) shown in Figure 2.4.2 (a) and (b). These two 3×3 matrixes are used to calculate the relative difference in pixel intensity with the x and y direction respectively. This is done by multiplying P by G_x and G_y separately. Looking at Figure 2.4.2 (a) we can see that there is no calculation happening in the middle vertical segment of the matrix. G_x is calculating for the relative difference between the left side and the right side of P. A large non-zero value as the outcome of P * G_x indicates the presence of pixel value difference in the x-direction. The same is true for G_y in the y-direction. These two masks will you give you the two individual components of detecting edges when scanning in the x-direction and y-direction.

$$G_X = (p2 - p0) + 2 * (p5 - p3) + (p8 - p6)$$

$$G_Y = (p6 - p0) + 2 * (p7 - p1) + (p8 - p2)$$

G_x =

1	0	-1
2	0	-2
1	0	-1

G_Y =

1	2	1
0	0	0
-1	-2	-1

(a) X-Gradient G_x (b) Y-Gradient G_y

Figure 2.4.2
Sobel Gradient Masks

Lastly, we want to combine the two results as a sum. Taking the sum of the two masks will more definitively define real edges. This is because the resultant sum is higher when an edge is detected in both directions. Once the sum is taken, we can display the pixel value in S as shown in Figure 2.4.3. This process is repeated pixel-by-pixel for the entirety of the input image *I*.

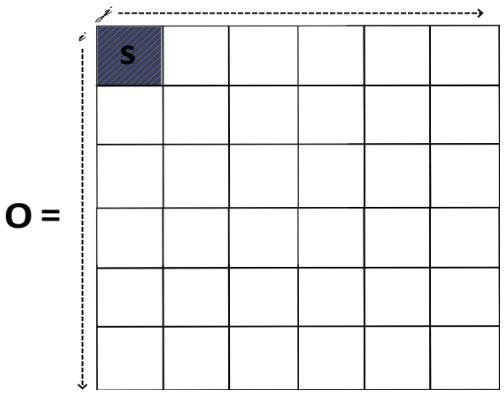


Figure 2.4.3
Sobel Pixel Output

Figure 2.4.3 shows the first processed pixel being input to the resultant output image *O*. This process occurs pixel by pixel until the entirety of *I* has been processed.

2.5 Image Processing Platform

An open-source image/video processing platform [25] was implemented for the simulation of the Sobel engine on FPGA. The image processing platform can capture an image, write to the buffer, allow for any type of image processing to be implemented, and the result to be output onto a VGA monitor. The VGA monitor output is restricted to 640×480-pixel resolution. The monitor output is split equally into 4 Regions.

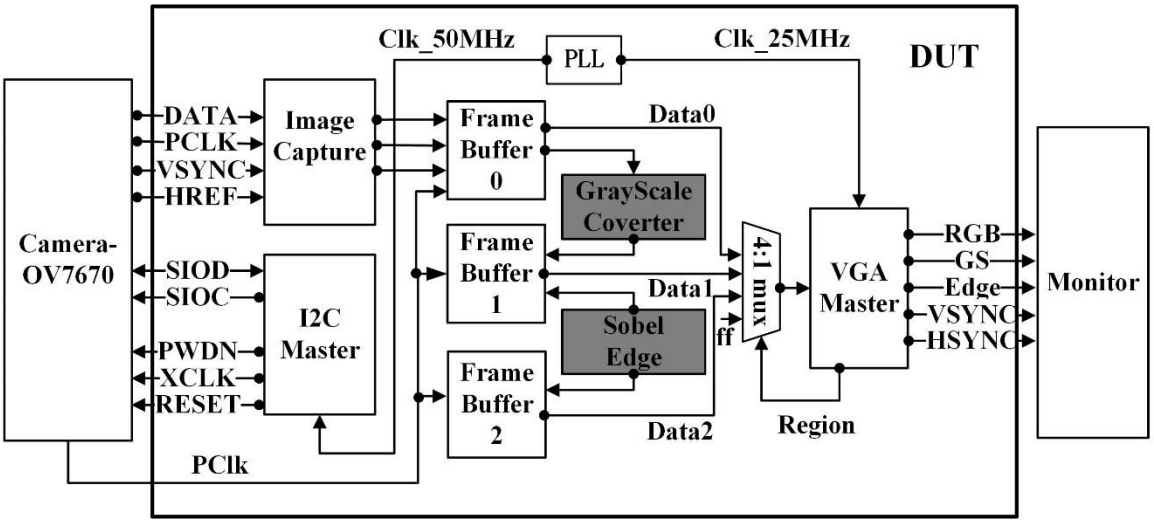


Figure 2.5.1
Sobel Edge Detection with Image Processing Platform on FPGA

Figure 2.5.1 is graphical representation of the image processing platform design under test (DUT). Firstly, the OV7670 camera is integrated as the capture device for the platform and is responsible for all initial input values in red, green, blue (RGB) color.

Data from the camera is sent to the appropriate buffers. *Buffer 0* is responsible for the original RGB input values. From there the data is sent to the *Grayscale Converter* for image processing and read into *Frame Buffer 1*. *Frame Buffer 1* supplies the grayscale values to be read out to the monitor, and to be used by the *Sobel*

Edge. The *VGA Master* controls which section of the 640 × 480 VGA display to read out to.

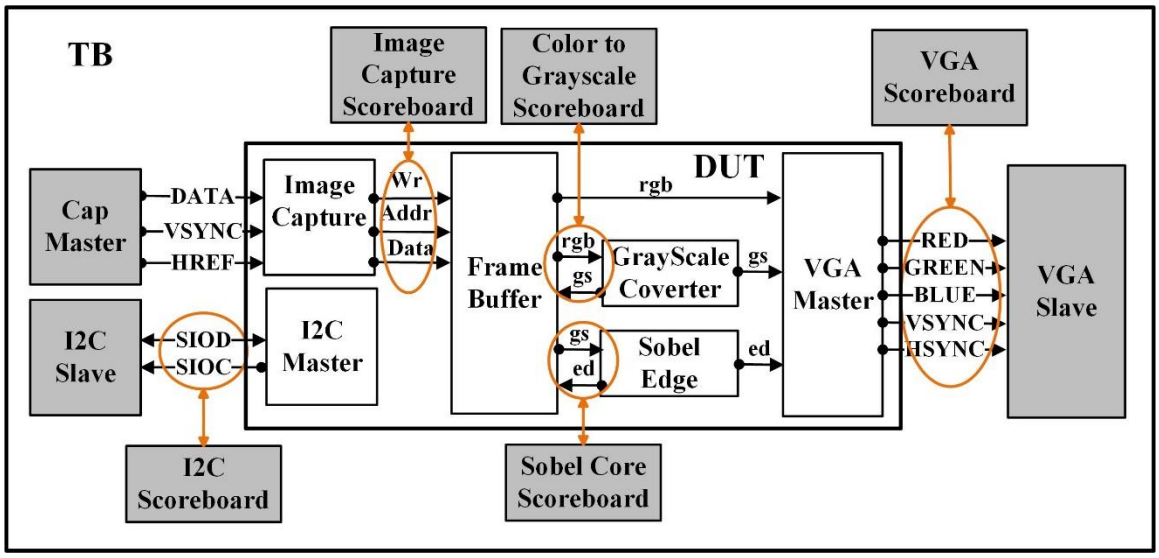


Figure 2.5.2 Image Processing Platform Testbench

Figure 2.5.2 is the testbench (TB) of the DUT. The functionality of each component of the DUT needs to be tested. Each scoreboard shows the inputs, expected output, and the actual output. These scoreboards allow for the isolation of components for easier troubleshooting.

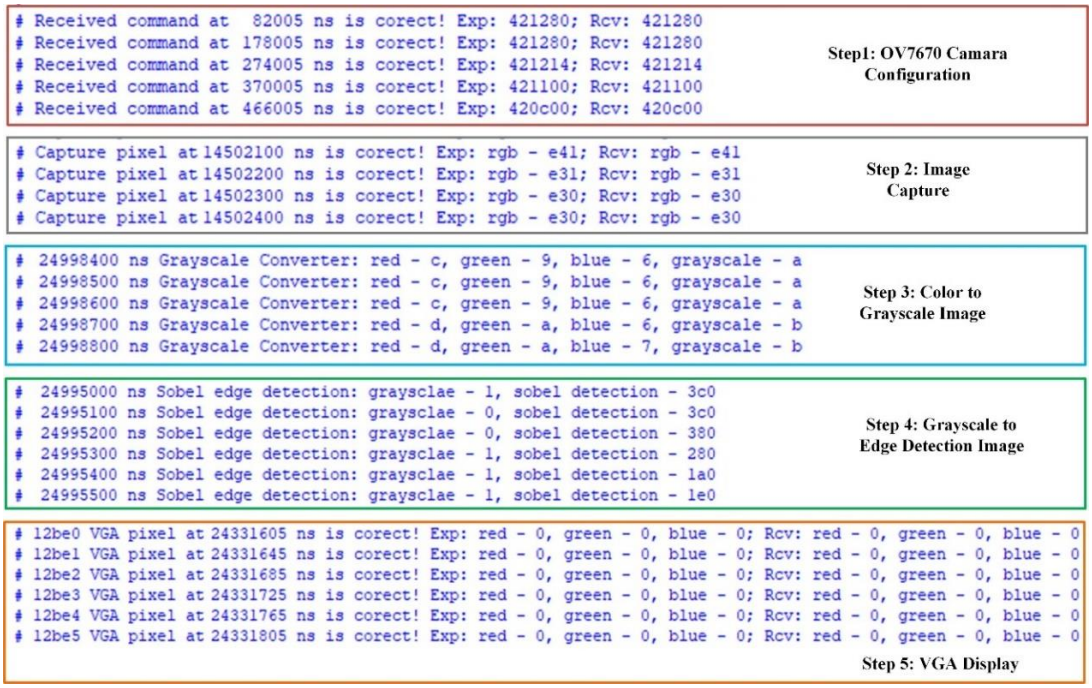


Figure 2.5.3 Simulation Report

Figure 2.5.3 displays the results of the simulation of the design.

Step 1 is the OV7670 configuration which simulates receiving data to the camera. 'Exp' is the expected data value for the camera to receive, and 'Rcv' is the actual value received by the simulated camera.

Step 2 is a simulation of the data being sent to the image capture from the OV7670 camera. The same convention between 'Exp' and 'Rcv' is used here. The image is captured as a RBG image.

Step 3 shows the first image processor. Data is sent to the *Frame Buffer 0,* the buffer reads the data to the *Grayscale Converter* where the data is converted into grayscale (8-bit) format. The simulation results show the components of the single pixel which are red, green, and blue, and the resultant grayscale pixel value after conversion. As you can see from Figure 2.5.1, the grayscale converter passes it's result to the VGA master as well as the frame buffer. The resulting grayscale pixel can be used again by

anything that the frame buffer can pass information to. This is an important step for the Sobel integration because it uses the grayscale formatted pixel to run its algorithm.

In Step 4 the grayscale pixel that was passed back to the frame buffer is sent to the *Sobel Edge* in Figure 2.5.1. The ‘sobel detection’ result is calculated from these grayscale converted values, its result read into a MUX, and the VGA master sends the result to its region of the VGA monitor.

Step 5 is the VGA scoreboard from 2.5.2. It confirms each output pixel by pixel in each region.

CHAPTER III:

IMPLEMENTATION

3.1 Hardware Implementation

The design of the adder, subtractor, and Sobel engine were presented in the proposed design section. In the following section, the functionality of the register-transfer (RT) level design is tested. This test is done by comparing the results produced by MATLAB to the results of the RTL design.

3.1.1 Adder

The implementation of the approximate adders is a continuation of previous work done in [21]. The adders shown below are the RTL configurations based on the K-Maps in Figure 2.2.1. Each figure, 3.1.1.1 – 3.1.1.5, is the result of the elaborated design supplied by Vivado for the exact design, AP1, AP2, AP3, and AP4 respectively.

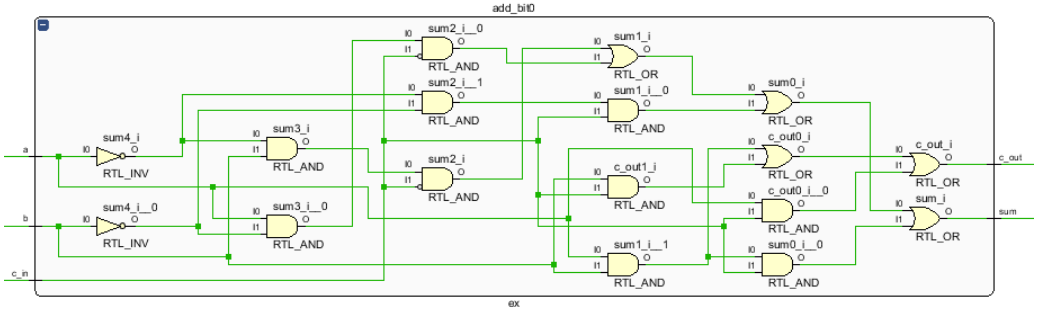


Figure 3.1.1.1
Exact Design: Adder

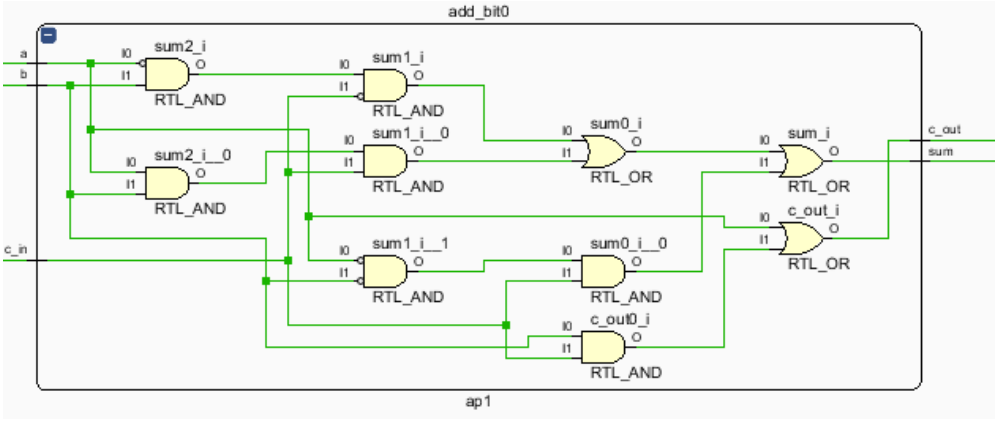


Figure 3.1.1.2
Approximate Design: AP1

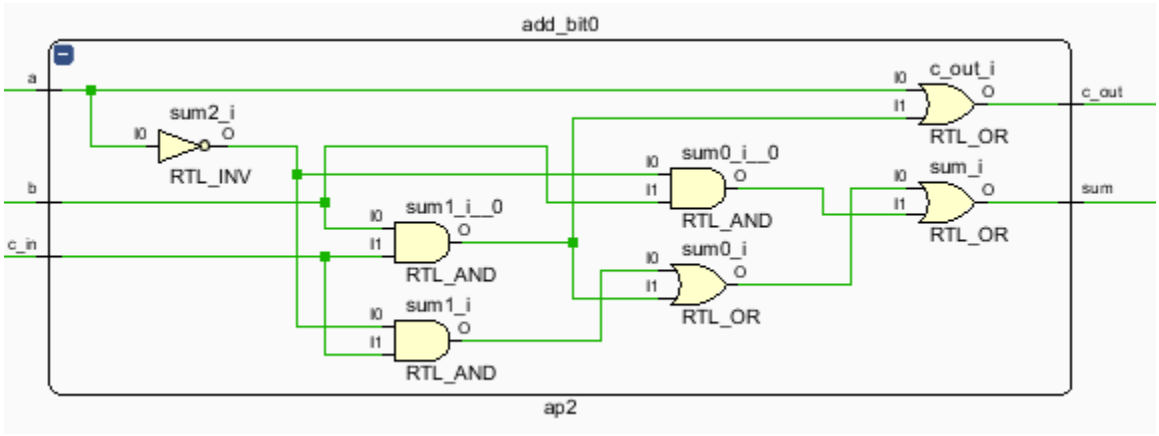


Figure 3.1.1.3
Approximate Design: AP2

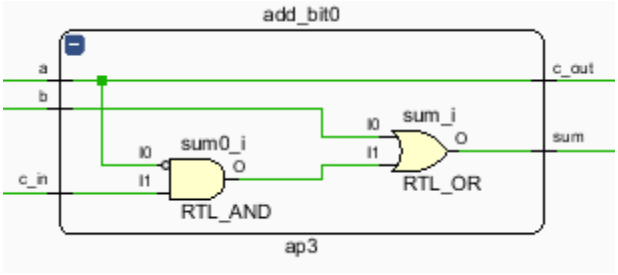


Figure 3.1.1.4
Approximate Design: AP3

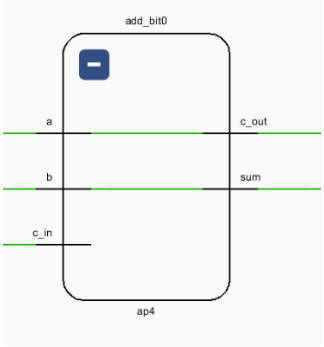


Figure 3.1.1.5
Approximate Design: AP4

Table 3.1.1 shows the number of gates within each design. The number of OR gates, denoted “RTL_OR”, and AND gates, denoted “RTL_AND”.

This table shows the level of approximation relative to the exact design. The output will become less accurate as the approximation level goes up. However, the reduced number of gates leads to less power and space consumption.

Table 3.1.1

Adder Gate Count

Design Name	AND Gate	OR Gate
Exact	10	5
AP1	7	3
AP2	3	3
AP3	1	1
AP4	0	0

The gate count saving between AP1 and exact design is 29%. The gate count savings between AP2 and exact is 57%. AP3 has a saving of 86%. AP4 is a special case where no logic is being done, we are simply connecting the inputs to outputs with a wire. This should yield the greatest amount of space and power saving but at the cost of the greatest amount of accuracy.

3.1.2 Subtractor

The subtractor designs shown below are the RTL configurations based on the K-Maps in Figure 2.3.1. Each figure, 3.1.2.1 – 3.1.2.5, is the result of the elaborated design supplied by Vivado for the exact design, subApprox1, subApprox2, subApprox3, and subApprox4 respectively. Each of the design's gate counts are provided in Table 3.1.2 at the end of this section.

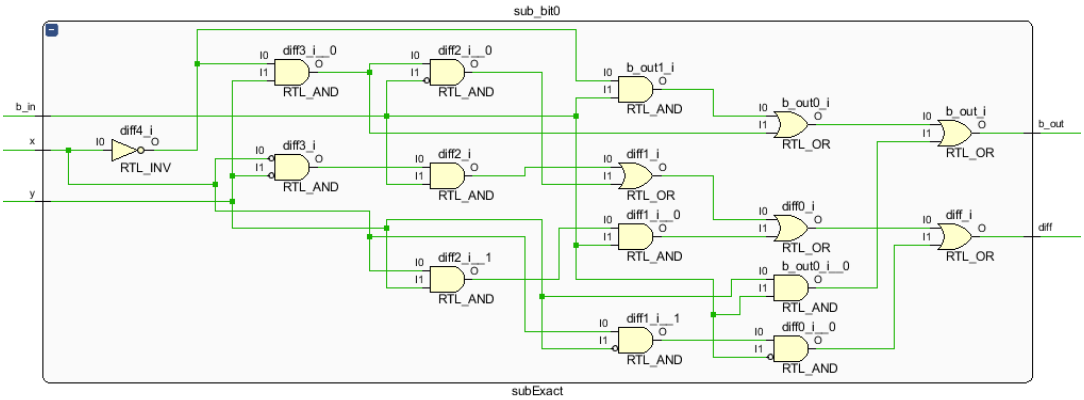


Figure 3.1.2.1
Exact Design: Subtractor

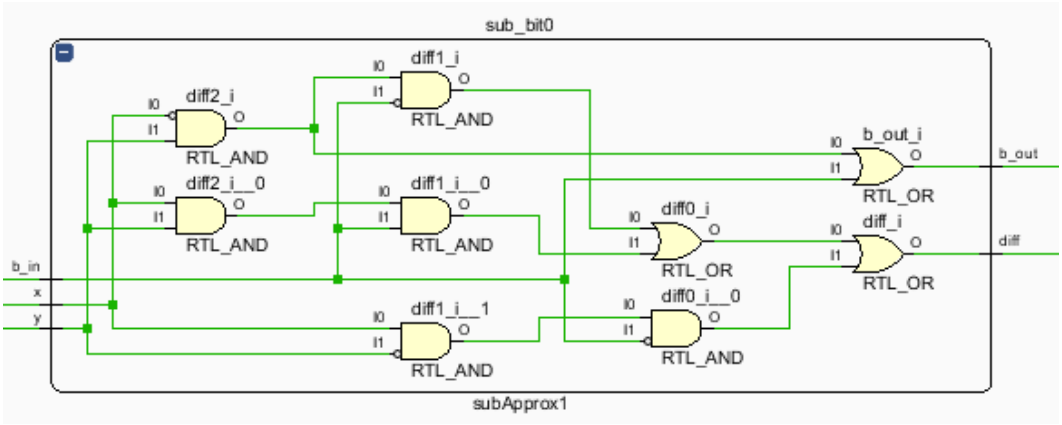


Figure 3.1.2.2
Approximate Design: SubApprox1

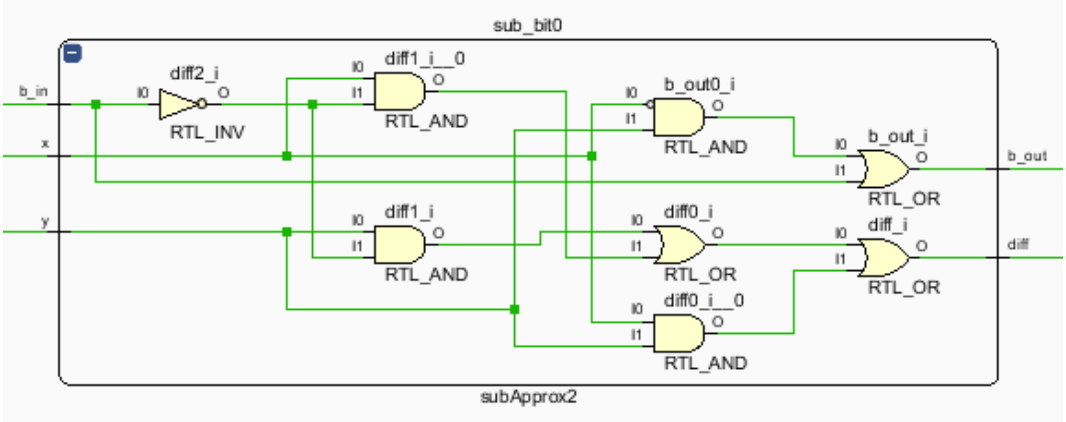


Figure 3.1.2.3
Approximate Design: SubApprox2

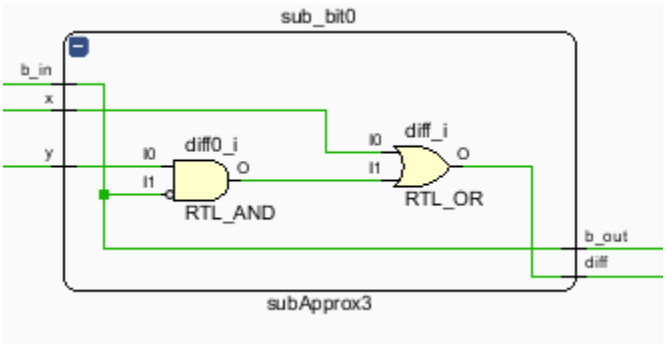


Figure 3.1.2.4
Approximate Design: SubApprox3

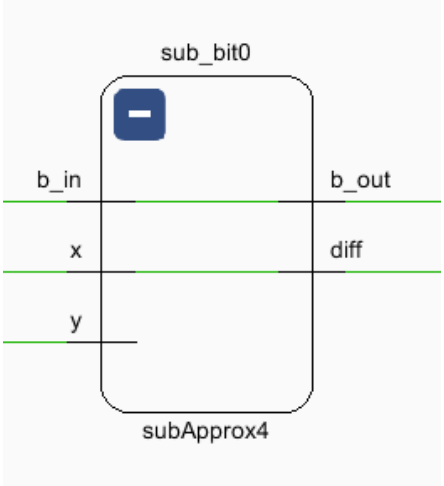


Figure 3.1.2.5
Approximate Design: SubApprox4

Table 3.1.2 shows the number of gates within each design. The number of OR gates, denoted “RTL_OR”, and AND gates, denoted “RTL_AND” for each subtractor design.

This table shows the level of approximation relative to the exact design. Like the adder, the output will become less accurate as the approximation level goes up. However, the reduced number of gates leads to less power and space consumption.

Table 3.1.2

Subtractor Gate Count

Design Name	AND Gate	OR Gate
Exact	10	5
subApprox1	6	3
subApprox2	4	3
subApprox3	1	1
subApprox4	0	0

We can see from the table that compared to the exact design, the subApprox1 is a gate count savings of 40%. The design subApprox2 saves 53%, subapprox3 saves 86%. Like AP4, out subApprox4 is a straight connection between input and output so therefore there is no gate logic. This will give us the greatest savings in energy and space at the the greatest cost of accuracy.

3.1.3 Sobel Algorithm

Sobel design was used as a case study of approximate design method. The Sobel design was implemented structurally. Each module represents a step in the process to the edge detected result.

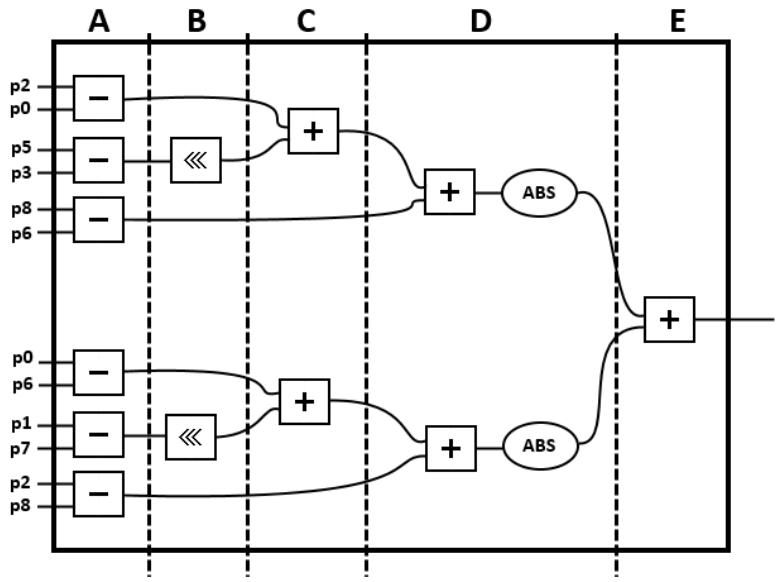


Figure 3.4.1
Sobel Implementation

The Sobel algorithm is broken down into two components. These components scan an image for edges in the x-direction, labeled G_x, and the y-direction, G_y. They calculate the gradient level of each direction. Below are the equations representing these two masks.

$$G_X = (p2 - p0) + 2 * (p5 - p3) + (p8 - p6)$$

$$G_Y = (p6 - p0) + 2 * (p7 - p1) + (p8 - p2)$$

Once the gradients are calculated they are finally summed together with their respective absolute values.

$$S = |G_X| + |G_Y|$$

As discussed earlier, the Sobel engine was implemented structurally. This allowed for quick access to submodules to test accuracy in MATLAB and energy usage in

Vivado. Each stage within Sobel algorithm is broken into what are called modules as shown by Figure 3.4.1.

Module A is a signed 8-bit subtractor. It takes two unsigned 8-bit pixel values and outputs a 9-bit signed value. Each pixel will first be subtracted, and their sign maintained. The maximum possible values of Module A are $(2^8 – 1)$ to $(– (2^8) +1)$ or 255 to -255 so we need 8 bits to hold the value and an additional leading most significant bit (MSB) to retain our sign.

Module A is responsible for taking the difference between p2 and p0 (p2 – p0) for the x-gradient and the difference between p0 and p6 for the y-gradient.

$$X – Gradient: \ (p2 – p0)$$
$$Y\text{ -}Gradient: \ (p5 – p3)$$

Module B is responsible for taking the difference between two pixels and left shifting multiplying it by two or left shifting it by one in order to multiply the result by 2. The important distinction in this step is that the sign bit is preserved by adding an addition bit to the bus size to represent the sign. Typically, only 9 bits are required to represent the signed difference of two 8-bit inputs, but an additional 10th bit is created to left shift the difference by 1. This multiplies the result by two without having to create a multiplier.

$$X – Gradient: \ 2 * (p5 – p3)$$
$$Y\text{ -}Gradient: \ 2 * (p1 – p7)$$

Module C adds the two differences in module A and B together and retains the sign.

*X – Gradient: (p2 – p0) + 2 * (p5 – p3)*

*Y -Gradient: (p5 – p3) + 2 * (p1 – p7)*

Module D adds the sum of module C to the difference of module A, p8 – p6, for the x-direction mask. Module D then outputs the absolute value of that sum.

X – Gradient: Absolute Value [(p8 – p6) + Module C_x]

Y -Gradient: Absolute Value [(p2 – p8) + Module C_y]

Module E adds both Module D outputs together. The sum will show the areas in which there is overlap between the x and y gradient values. This will make edges within an image more conspicuous.

$$S = |G_X| + |G_Y|$$

A multiplexer (MUX) that limits the output of the Sobel engine so that more definitive edges are shown. As discussed at the beginning of this section, the Sobel algorithm is intended to calculate the difference between sections of a pixel kernel in the x and y direction. It does this calculation for all pixels within a picture or video. The intent of this case study is to show all relevant edges in a picture or video. The final portion of Module E has a MUX that sets the out to "1" for any final sum that exceeds 512. This is particularly low cost to implement thanks to the 9th bit begins 512. The final sum is read, the top n-to-9 bits are OR-ed together to be used as a select signal. Anything bit above 8 that contains a "1" will be used to set the entire sum to 8 bits all reading "1".

If the number is below 512 the sum reads “0”. Since our image is being processed in the grayscale color scheme.

3.1.3.1 Sobel RTL

Sobel algorithm was then transferred to RTL level design. A structural approach was implemented. This allowed for the designs to be interchanged quickly and without excessive troubleshooting. Figure 3.1.3.1 is the exact design of the Sobel algorithm with each module being represented by a component box with the module’s name underneath. Each adder and subtractor within the exact Sobel design uses the exact models of the adders and subtractors.

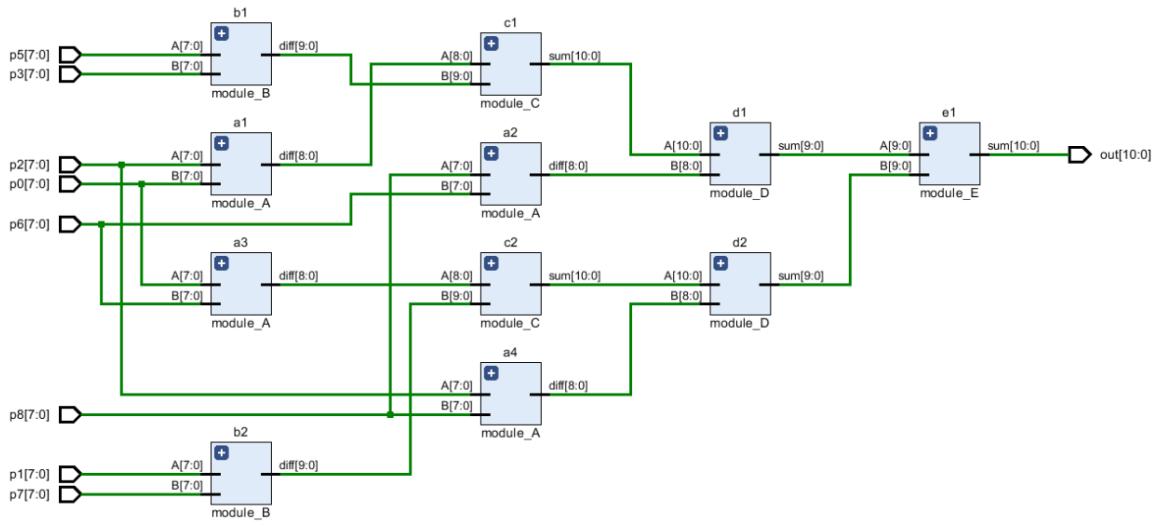


Figure 3.1.3.1
Exact Sobel Design RTL

Figure 3.1.3.2 shows the expanded view of module_A which is responsible for the difference between p2 and p0. It is implemented by instantiating 8 exact subtractors.

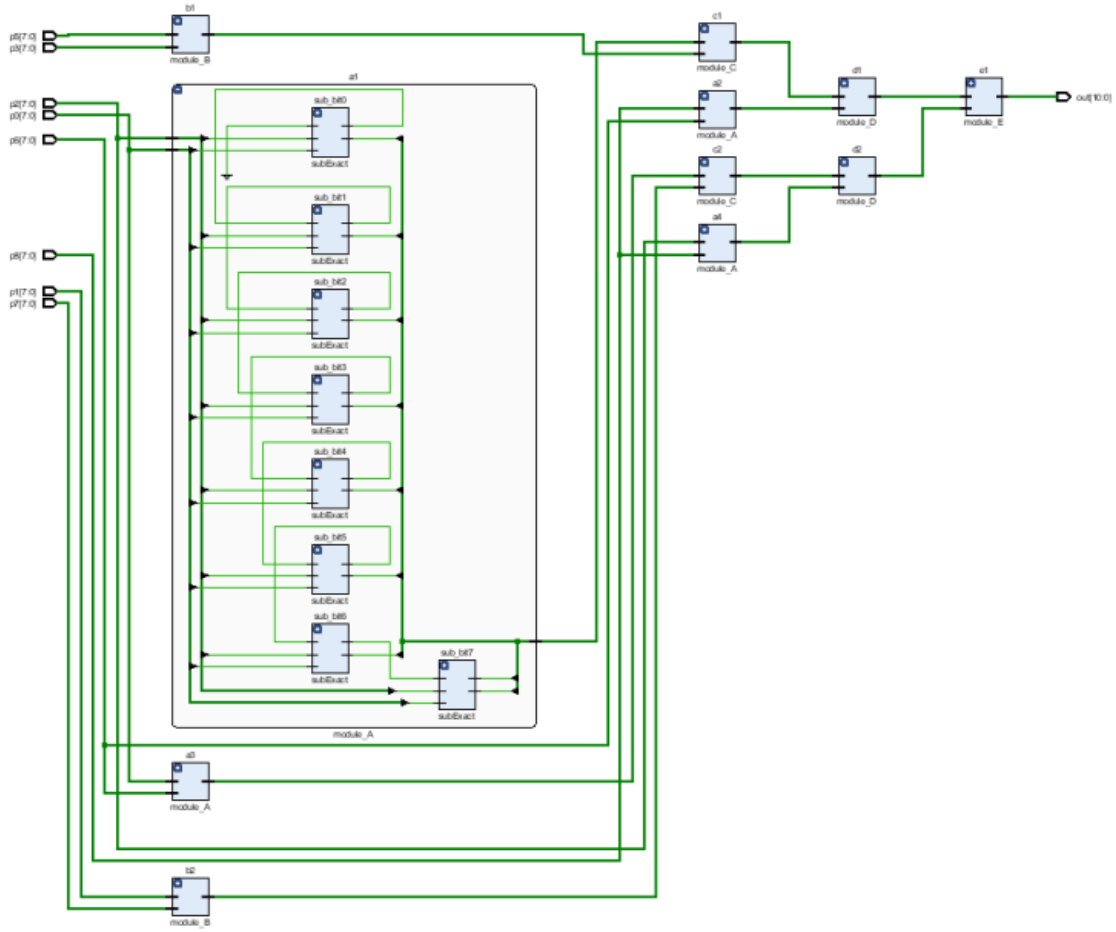


Figure 3.1.3.2
Exact Sobel: Structural Expansion

3.1.3.2 Sobel Approximate Designs

From the list of approximate adder and subtractor designs discussed in Section 2.3, 6 custom Sobel filters were created. Each module uses a subtractor or adders, so each module was given an approximation design and an approximation level. The approximation designs are outlined in the following graphs. The approximation level is a value that dictates how many bits, from least significant bit to most significant bit, using a specific design.

Custom 1	Function	Approximation
x_diff1	sub1	5
x_diff2	sub1	5
x_diff3	sub1	5
x_sum1	ap1	5
x_sum2	ap1	5
y_diff1	sub1	5
y_diff2	sub1	5
y_diff3	sub1	5
y_sum1	ap1	5
y_sum2	ap1	5
final_sum	ap1	5

Figure 3.1.3.2.1
Sobel Design: Custom 1

Custom model 1 from Figure 3.1 uses approximate subtractor 1 (subApprox1) for the lowest 5 bits of the 8-bit subtractor for horizontal gradient (x_diff1, x_diff2, x_diff3) and the vertical gradient (y_diff1, y_diff2, y_diff3). The approximate adder 1 (addApprox1) was used for the lowest 5 bits in the adders and the remaining higher bits used the exact design.

Custom 2	Function	Approximation
x_diff1	sub1	6
x_diff2	sub1	6
x_diff3	sub1	6
x_sum1	ap1	5
x_sum2	ap1	5
y_diff1	sub1	6
y_diff2	sub1	6
y_diff3	sub1	6
y_sum1	ap1	5
y_sum2	ap1	5
final_sum	ap1	5

Figure 3.1.3.2.2
Sobel Design: Custom 2

Custom model 2 from the figure above uses approximate subtractor 1 (subApprox1) for the lowest 5 bits of the 8-bit subtractor for horizontal gradient (x_diff1, x_diff2, x_diff3) and the vertical gradient (y_diff1, y_diff2, y_diff3). The approximate adder 1 (addApprox1) was used for the lowest 6 bits in the adders and the remaining 2 higher bits used the exact design. Custom models 1 and 2 share the same arithmetic approximate designs sub1 and ap1. The number of bits approximated by sub1 was increased from 5 to 6 in order to test the amount of error introduced to the entire system by each bit approximated.

Custom 3	Function	Approximation
x_diff1	sub2	4
x_diff2	sub2	4
x_diff3	sub2	4
x_sum1	ap2	4
x_sum2	ap2	4
y_diff1	sub2	4
y_diff2	sub2	4
y_diff3	sub2	4
y_sum1	ap2	4
y_sum2	ap2	4
final_sum	ap2	4

Figure 3.1.3.2.3
Sobel Design: Custom 3

Custom model 3 from the figure above uses approximate subtractor 2 (subApprox2) for the lowest 4 bits. The approximate adder 2 (ap2) was used for the lowest 6 bits.

Custom 4	Function	Approximation
x_diff1	sub2	5
x_diff2	sub2	5
x_diff3	sub2	5
x_sum1	ap2	5
x_sum2	ap2	5
y_diff1	sub2	5
y_diff2	sub2	5
y_diff3	sub2	5
y_sum1	ap2	5
y_sum2	ap2	5
final_sum	ap2	5

Figure 3.1.3.2.4
Sobel Design: Custom 4

Custom model 4 integrates the same approximate designs as custom 3. The level of approximation was increased by 1 for both the subtractors and the adders.

Custom 5	Function	Approximation
x_diff1	sub3	4
x_diff2	sub3	4
x_diff3	sub3	4
x_sum1	ap3	4
x_sum2	ap3	4
y_diff1	sub3	4
y_diff2	sub3	4
y_diff3	sub3	4
y_sum1	ap3	4
y_sum2	ap3	4
final_sum	ap3	4

Figure 3.1.3.2.5
Sobel Design: Custom 5

Custom model 5 utilizes subApprox3 for the lowest 4 bits of all of its subtractors. All adders in this custom design use AP3 for the lowest 4 bits. The rest of the higher bits use the exact design. This design should give a significant power and space savings.

Custom 6	Function	Approximation
x_diff1	sub3	5
x_diff2	sub3	5
x_diff3	sub3	5
x_sum1	ap3	5
x_sum2	ap3	5
y_diff1	sub3	5
y_diff2	sub3	5
y_diff3	sub3	5
y_sum1	ap3	5
y_sum2	ap3	5
final_sum	ap3	5

Figure 3.1.3.2.6
Sobel Design: Custom 6

Custom model 46integrates the same approximate designs as custom 5. The level of approximation was increased by 1 for both the subtractors and the adders. This is to demonstrate the difference in accuracy while decreasing the power and space consumption.

CHAPTER IV:

EXPERIMENTAL RESULTS

4.1 Image Output

The following images were generated using results created by MATLAB. Each custom Sobel design was implemented in both RTL and MATLAB.

Figure 4.1.1 is the image used as a static input to the system. 4.1.2 is the grayscale conversion. The Sobel algorithm is designed to take an 8-bit grayscale data input to perform calculations.

Figure 4.1.1
Input Image

Figure 4.1.2
Grayscale Converted Image

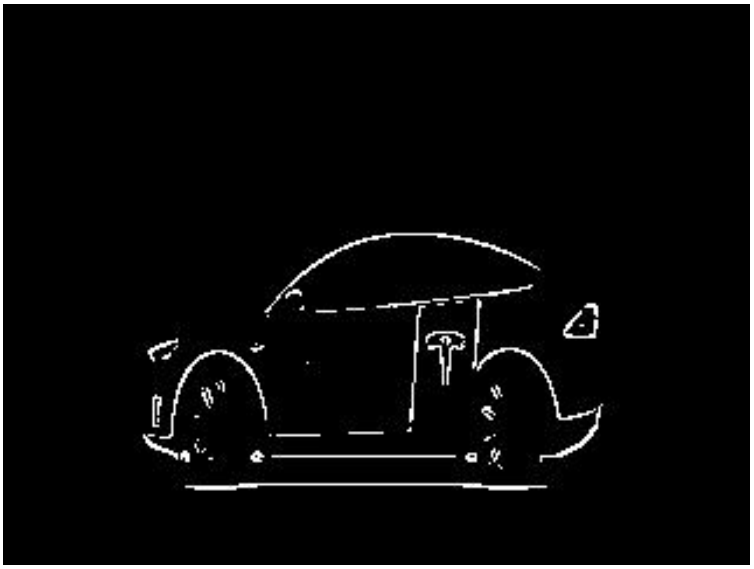


Figure 4.1.3
Sobel Exact Design

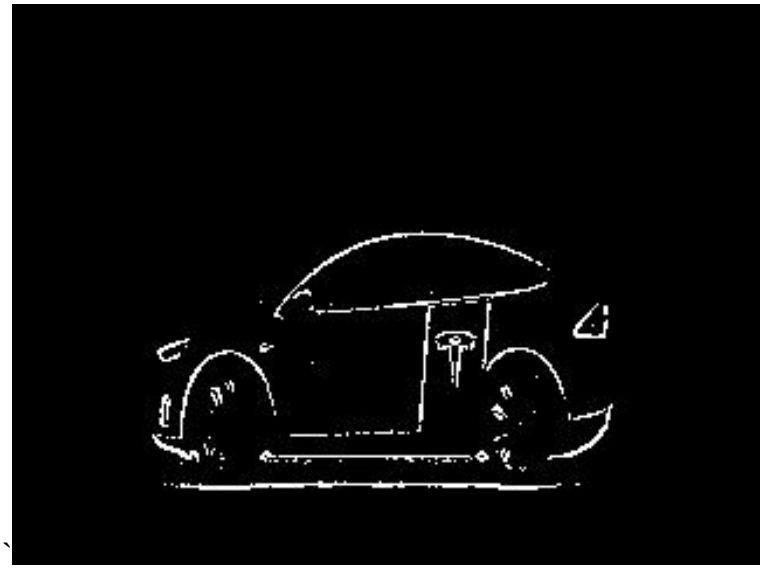


Figure 4.1.4
Sobel Custom Image 1

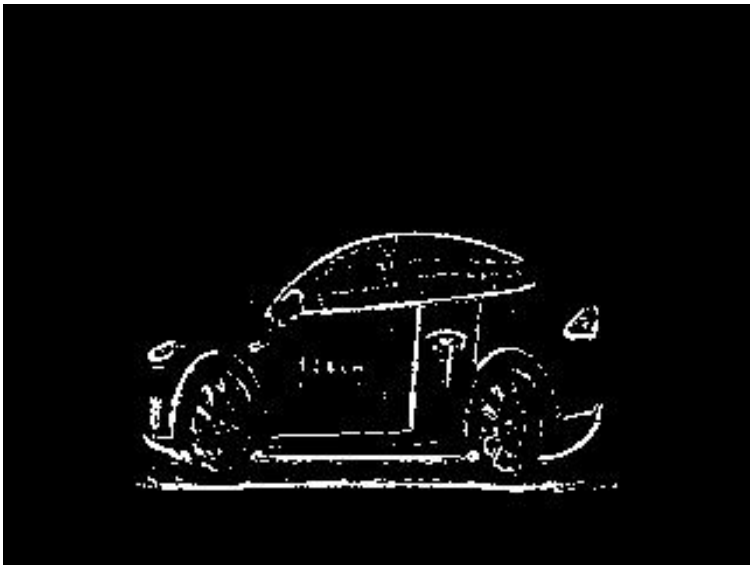


Figure 4.1.5
Sobel Custom Image 2

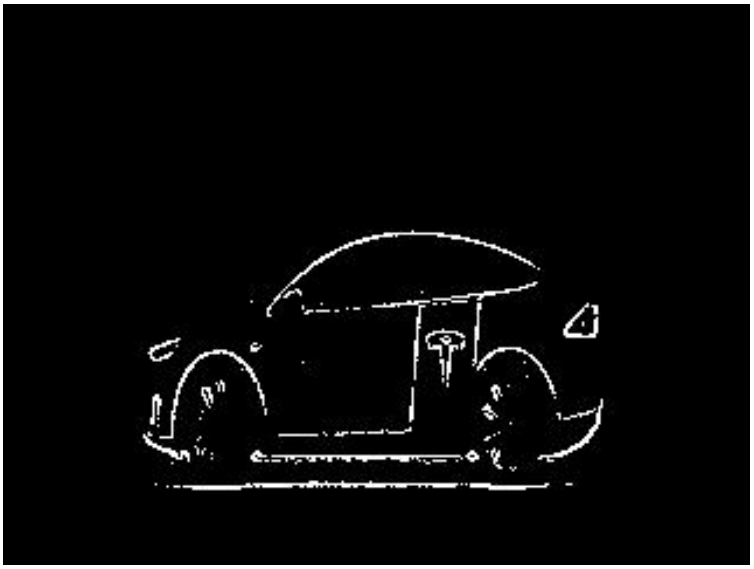


Figure 4.1.6
Sobel Custom Image 3

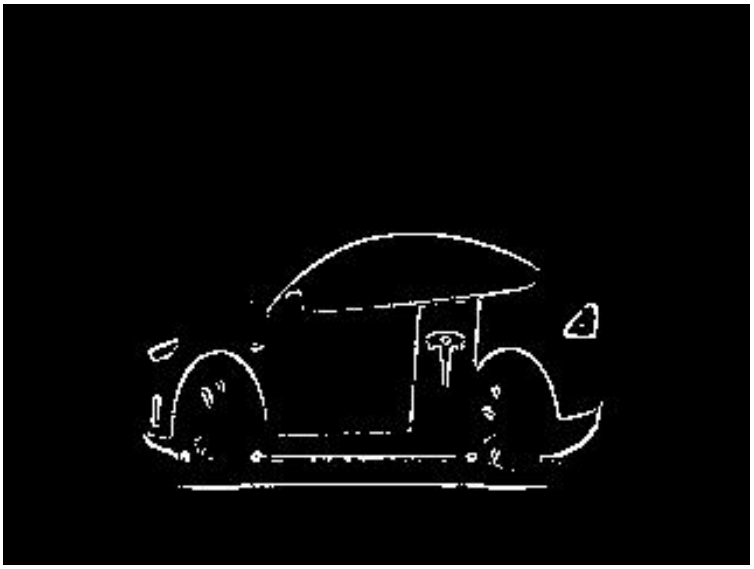


Figure 4.1.7
Sobel Custom Image 4

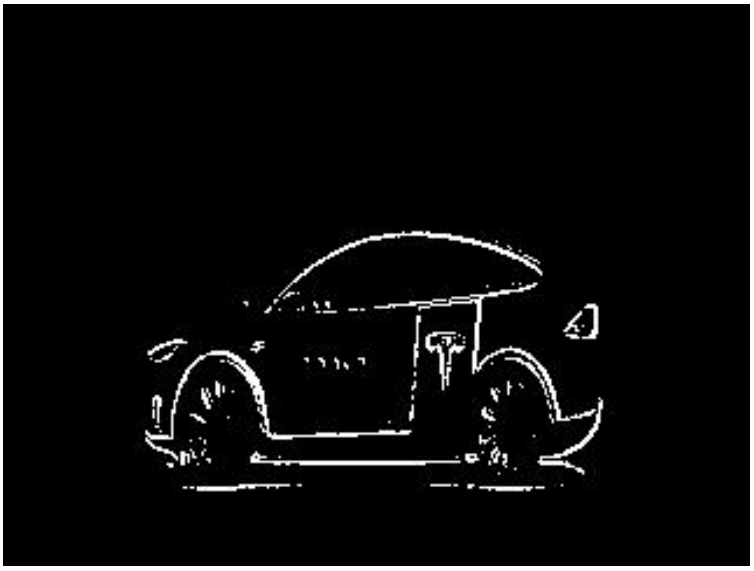


Figure 4.1.8
Sobel Custom Image 5

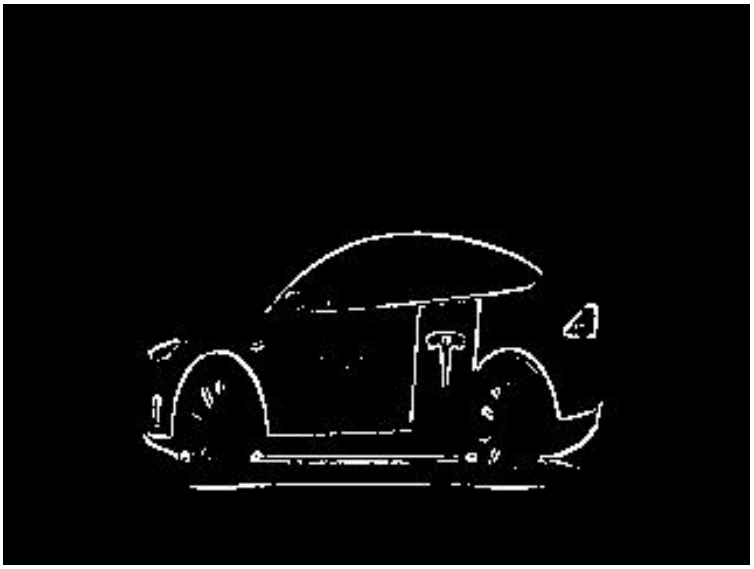


Figure 4.1.9
Sobel Custom Image 6

Each figure from Figure 4.1.3 to 4.1.9 is the result of the Sobel algorithm finding edges from the image in Figure 4.1.2. The graysale image was transferred in a 3×3 kernel, pixel by pixel, to the Sobel algorithm.

Each design was implemented to study the affect, if any, that increasing approximation has on the accuracy of the edge detection. The designs were created by adjusting the knob of accuracy to decrease the resource usage at the cost of accuracy.

Looking back at the designs outlined in Section 3.1.3.2, we know that the image outputs are paired by approximate design. The intent was to show a qualitative loss of the image for the reader to see. Table 4.1.1 shows the quantitative difference between the exact and approximated design.

4.2 FPGA Resources and Error Rate

Xilinx Vivado was used to implement the designs, and power and ultilization reports were used to get power and look-up table (LUT) usage.

Table 4.2.1

Power and Slice Count

Design Name	Power (W)	LUTs	Slice as LUTs
Exact Design	18.813	118	38
Custom 1	16.877	125	38
Custom 2	16.705	125	40
Custom 3	15.953	124	40
Custom 4	15.446	127	40
Custom 5	16.076	101	32
Custom 6	13.937	93	29

The results show that the highest level of power usage came from the exact design, as we expected. This design has the longest critical path and the most components used. The amount of LUTs increased by a max of 7.1%, however that same comparison resulted in a decrease of power usage from 18.813 W to 15.446 W.

Looking at Section 3.1.3.2 we see that Sobel approximate designs are paired together by approximate adder and subtractor design. Custom designs 1 and 2 use the same approximators while increasing the amount approximated. Continuing this trend, we link 3 and 4 by approximator 2 in both the adder and subtractor. Custom design 5 and 6 are linked by approximation 3's design of the adder and subtractor.

This fact helps explain the image error rate between each custom model which is gathered in Table 4.1.1.

Table 4.1.1

Custom Sobel Error Rate

Custom Model	Image Error
1	0.75%
2	1.80%
3	0.60%
4	1.39%
5	1.14%
6	2.68%

$$Image\ Error = \left[\frac{Sobel_{Exact}[i,j] - Sobel_{Custom}[i,j]}{Sobel_{Exact}[i,j]} \times 100\%\right] \times \frac{1}{Image\ Resolution}$$

The equation above was used to calculate the image error amount. The resultant image is scanned by its row, denoted *i*, and columns, denoted *j*, and compared to the Sobel exact design. The first portion of the equation in brackets calculates the error within a single pixel. The error rate is then divided by the image resolution in order to determine the percentage difference between the two images' pixel values. The difference between the $Sobel_{Exact}$ and the $Sobel_{Custom}$ The same picture was used to test the accuracy of each Sobel custom model. The $Sobel_{Exact}$ image was recorded and used as a rule against each custom design. Comparing the two images pixel by pixel and taking the average against the size of the image. This way we can show the accuracy of each binary image output.

CHAPTER V:

CONCLUSION

5.1 Conclusion

This chapter concludes the contributions presented in this dissertation are summarized. First the alogrithm of quality and space tradeoff is presented and explained. This is followed by the concept of approximate design methodology, approximate design library, Sobel core design, proposed designs, and testing the validity of the energy to quality tradeoff.

The implementation of 6 approximate Sobel designs and 4 subtractor approximate designs. Each increase in approximation reduced the amount of energy required at the cost of accuracy. This was a demonstration of the original quality knob algorithm suggested at the beginning of this disertation. It was expressed by incrementally increasing the approximation level of the Sobel design to showcase the tradeoff cost of accuracy with energy.

5.2 Future Work

Future wor could include the considering the implementation of the approximate design methodology into areas that require high volume processing with with low accuracy. For instance, an image processing system that can detect fruits or vegetables that are not ripe or overipened. This would be a high volume application, where the stakes are low when mistakes are made.

REFERENCES

1. S. Amanollahi, M. Kamal, A. Afzali-Kusha, and M. Pedram, “Circuit-Level Techniques for Logic and Memory Blocks in Approximate Computing Systems," Proceedings of the IEEE, PP. 1-28, 2020.
2. H. Jiang, F. J. H. Santiago, H. Mo, et al., “Approximate Arithmetic Circuits: A Survey, Characterization, and Recent Applications," Proceedings of the IEEE, PP. 1-28, 2020.
3. S. Reda and M. Shaque, “Approximate Circuits," PP. 249-370, ISBN 978-3-319-99321-8, 2019.
4. F. E. Azandaryani, O. Akbari, M. Kamal, et. al, Block-Based Carry Speculative Approximate Adder for Energy-Efficient Applications," IEEE Transactions on Circuits and Systems II: Express Briefs, Vo. 67, No. 1 , PP. 137-141, Jan. 2020.
5. B. Sakthivel and A. Padma,” Area and delay efficient GDI based accuracy configurable adder design," Microprocessors and Microsystems, Vol. 73, March 2020.
6. J. Hu, Z. Li, M. Yang, Z. Huang, and W. Qian, “A high-accuracy approximate adder with correct sign calculation, " Integration, the VLSI, Vol. 65, PP. 370-388, March 2019.

7. Y. Guo, H. Sun, P. Lei, S. Kimura, “Approximate FPGA-Based Multipliers Using Carry Inexact Elementary Modules," IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, Vol.E103-A, No.9, PP.1054-1062, 2020.
8. S. Vahdat, M. Kamal, A. A. Kusha, and M. Pedram, “TOSAM: An Energy-Efficient Truncation- and Rounding-Based Scalable Approximate Multiplier," IEEE Trans. on Very Large Scale Integration (VLSI) Systems Vol. 27, No. 5, May 2019.
9. N. Van Toan and J. Lee, “FPGA-Based Multi-Level Approximate Multipliers for High-Performance Error-Resilient Applications," IEEE Access, Vol. 8, PP. 25481-5497, 2020.
10. Sakthivel M. Imani, R. Garcia, A. Huang, and T. Rosing, “CADE: Configurable Approximate Divider for Energy Efficiency," 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), PP. 1-4, May 2019.
11. Y. Zhang, X. Yang , L.Wu, and J. Andrian, “A Case Study On Approximate FPGA Design With an Open-Source Image Processing Platform," IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Student Forum, PP.372-377, Miami, FL, US, 2019.
12. B. K. Mohanty, “Parallel VLSI Architecture for Approximate Computation of Discrete Hadamard Transform," IEEE Transactions on Circuits and Systems for Video Technology,PP. 1-9, 2020.
13. M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek and J. Han, “Improving the Accuracy and Hardware Efficiency of Neural Networks Using Approximate Multipliers," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 28, No. 2, PP. 317-328, Feb. 2020.

14. B. Adarsha, U. Salim, D. Anup, and K. Akash “Design Methodology for Embedded Approximate Artificial Neural Networks," Proceedings of the 2019 on Great Lakes Symposium on VLSI (GLSVLSI'19), PP. 489-494, 2019.
15. B. Liu, Z. Wang, S. Guo, et al., “An Energy-Efficient Voice Activity Detector Using Deep Neural Networks and Approximate Computing," Microelectronics Journal, Vol. 87, PP. 12-21, 2019.
16. G. Anusha and P. Deepa, “Design of approximate adders and multipliers for error tolerant image processing," Microprocessors and Microsystems, Vol. 72, Feb. 2020.
17. A. Raha, H. Jayakumar, and V. Raghunathan, “Input-based Dynamic Reconfiguration of Approximate Arithmetic Units for Video Encoding", IEEE Trans. on VLSI Syst., Vol. 24, No. 3, PP. 846-857, 2016.
18. A. Raha, S. Venkataramani, V. Raghunathan, et al, “Quality Configurable Reduce-and-rank for Energy Efficient Approximate Computing", Design, Automation and Test in Europe Conference and Exhibition (DATE), PP. 89-98, 2015.
19. S. Bose, B. Kar, M. Roy, et. al, “ADEPOS: A Novel Approximate Computing Framework for Anomaly Detection Systems and Its Implementation in 65-nm CMOS", IEEE Transactions on Circuits and Systems I: Regular Papers, PP. 1-14, Dec. 2019.
20. S. Liu, F. Lau, and B. Schafer, “Accelerating FPGA Prototyping through Predictive Model-Based HLS Design Space Exploration", 2019 56th ACM/IEEE Design Automation Conference (DAC), PP. 1-6, Aug. 2019.
21. Y. Zhang, X. Yang, L. Wu, and J. Andrian, “A Case Study On Approximate FPGA Design With an Open-Source Image Processing Platform," IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Jan. 13, 2020.

22. S. Sinha and W. Zhang, “Low-Power FPGA Design Using Memoization-Based Approximate Computing", IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 24, No. 8, PP. 2665-2678, Aug. 2016.

23. V. Mrazek, M. Hanif, Z. Vasicek, et al., “autoAx: An Automatic Design Space Exploration and Circuit Building Methodology utilizing Libraries of Approximate Components," Proceedings of the 56th Annual Design Automation Conference (DAC), No.: 123, PP. 1-6, June 2019.

24. X. Yang,& S. Sha, “Exploiting Energy-Quality (E-Q) Tradeoffs: A Case Study on Color to Grayscale Converters with Approximate Design on FPGA," Journal of Circuits, Systems and Computers (JCSC), 2020.

25. X. Yang, Y. Zhang, and L. Wu, “A Scalable Image/Video Processing Platform with Open Source Design and Verification Environment," 20th Intl. Symposium on Quality Electronic Design (ISQED 2019), PP. 110-116, Santa Clara, CA, USA, 2019.

26. D. Sujeet, “Comparison of Various Edge Detection Technique," International Journal of Signal Processing, Image Processing and Pattern Recognition, Vol. 9, No. 2, PP. 143-158, 2016.

27. B. You, W. Sheng, H. Ma, Y. Gu, and Y. Qin, “Implementation of Sobel Edge Detection on FPGA based on OpenCL," 2017 IEEE 7th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems, 2017.

28. S. Eetha, S. Agarwal, and S. Neelam. Zynq FPGA Based System Design for Video Surveillance with Sobel Edge Detection," 2018 IEEE International Symposium on Smart Electronic Systems (iSES), 2018.

29. M. V. Fernando, K. Christian, and J. Pallav, “Zynq All Programmable SoC Sobel Filter Implementation Using the Vivado HLS Tool," Xilinx Application Note: Vivado HLS Tool, XAPP890 (v1.0), Sept. 2012.
30. B. Hong, H.-Y. Kim, M. Kim, T. Suh, et al., “FASTEN: An FPGA-Based Secure System for Big Data Processing," IEEE Design & Test, Vol. 35, No. 1, PP. 30{38, 2018.
31. M. T. Obaid, “Efficient Implementation of Sobel Edge Detection with Zynq-7000," Purdue University Graduate School - Thesis, 2020.
32. A. Cortes, I. Velez, and A. Irizar, “High level synthesis using Vivado HLS for Zynq SoC:Image processing case studies," 2016 Conference on Design of Circuits and Integrated Systems (DCIS), 2016.
33. N. Nausheen, A. Seal, P. Khanna, and S. Halder, “A FPGA based implementation of Sobel edge detection," Microprocessors and Microsystems, Vol. 56, PP. 84-91, 18.
34. A. Raha, H. Jayakumar, and V. Raghunathan, “Input-based Dynamic Reconfiguration of Approximate Arithmetic Units for Video Encoding", IEEE Trans. on VLSI Syst., Vol. 24, No. 3, PP. 846-857, 2016.
35. A. Raha, S. Venkataramani, V. Raghunathan, et al, “Quality Configurable Reduce-and-rank for Energy Efficient Approximate Computing", Design, Automation and Test in Europe Conference and Exhibition (DATE), PP. 89-98, 2015.
36. K. Vaca, M. Jefferies, and X. Yang, “An Open Real-Time Audio Processing Platform on Zync FPGA,” International Symposium on Measurement and Control in Robotics (ISMCR), PP. D1-2-1-D1-2-6, Houston, TX, USA, 2019.

37. K. Vaca, A. Gajjar, and X. Yang , "Real-Time Automatic Music Transcription (AMT) with Zync FPGA," IEEE Computer Society Annual Symposium on VLSI (ISVLSI, Acceptance Rate: 17%) , PP. 378-384, Miami, FL, USA, 2019.

38. A. Gajjar, et. al., “An FPGA Synthesis of Face Detection Algorithm using HAAR Classifiers,” Intl. Conference on Algorithms, Computing and Systems (ICACS 2018), PP.133-137, July 27-29, Beijing China, 2018.

39. A. Gajjar, et. al., "An IoT-Edge-Server System with BLE Mesh Network, LBPH, and Deep Metric Learning," he 22nd Int'l Conf on Artificial Intelligence (ICAI 2020), IN Press, March 2020.

40. X. Yang, et. al., “A Vision of Fog Systems with Integrating FPGAs and BLE Mesh Network,” Journal of Communications (JoC), Vol. 14, No. 3, PP. 210-215, March 2019.

41. X. Yang and X. He, “Establishing a BLE Mesh Network using Fabricated CSRmesh Devices,” The 2nd ACM/IEEE Symposium on Edge Computing (SEC 2017), No. 34, San Jose/Fremont, CA, US, 2017.

42. A. Gajjar, Y. Zhang, and X. Yang, “A Smart Building System Integrated with An Edge Computing Algorithm and IoT Mesh Networks,” The Second ACM/IEEE Symposium on Edge Computing (SEC 2017), Article No. 35, San Jose/Fremont, CA, US, 2017.

43. D. Sujeet, “Comparison of Various Edge Detection Technique," International Journal of Signal Processing, Image Processing and Pattern Recognition, Vol. 9, No. 2, PP. 143-158, 2016.

44. B. You, W. Sheng, H. Ma, Y. Gu, and Y. Qin, “Implementation of Sobel Edge Detection on FPGA based on OpenCL," 2017 IEEE 7th Annual International

Conference on CYBER Technology in Automation, Control, and Intelligent Systems, 2017.

45. A. Cortes, I. Velez, and A. Irizar, “High level synthesis using Vivado HLS for Zynq SoC:Image processing case studies," 2016 Conference on Design of Circuits and Integrated Systems (DCIS), 2016.
46. S. Eetha, S. Agarwal, and S. Neelam. Zynq FPGA Based System Design for Video Surveillance with Sobel Edge Detection," 2018 IEEE International Symposium on Smart Electronic Systems (iSES), 2018.
47. M. V. Fernando, K. Christian, and J. Pallav, “Zynq All Programmable SoC Sobel Filter Implementation Using the Vivado HLS Tool," Xilinx Application Note: Vivado HLS Tool, XAPP890 (v1.0), Sept. 2012.
48. B. Hong, H.-Y. Kim, M. Kim, T. Suh, et al., “FASTEN: An FPGA-Based Secure System for Big Data Processing," IEEE Design & Test, Vol. 35, No. 1, PP. 30{38, 2018.
49. M. T. Obaid, “Efficient Implementation of Sobel Edge Detection with Zynq-7000," Purdue University Graduate School - Thesis, 2020.
50. N. Nausheen, A. Seal, P. Khanna, and S. Halder, “A FPGA based implementation of Sobel edge detection," Microprocessors and Microsystems, Vol. 56, PP. 84-91, 18.48.
51. OV7670 Datasheet, Version 1.01, OmmiVision Technologies, Sunnyvale, CA, USA, 2005.
52. Nexys 4 FPGA Board Reference Manual, Rev. B, Digilent, Sunnyvale, CA, USA, April 2016.

53. X. Yang, N. Wu, and J. Andrian, “A Novel Bus Transfer Mode: Block Transfer and A Performance Evaluation Methodology,” Elsevier, Integration, the VLSI Journal, Vol. 52, Issue: C, PP. 23-33, Jan. 2016.

54. X. Yang, et.al., "An Edge Detection IP of Low-cost System-on-Chip for Autonomous Vehicles," The 22nd Int'l Conf on Artificial Intelligence (ICAI 2020), In Press, March 2020.