Computational Analysis between Software and Hardware Implementation of Sobel Edge Detection Algorithm

Rajesh Kannan Megalingam, Malavika Karath, Prajitha P and Goutham Pocklassery

Abstract—The main aim of the project is to compare the variations in execution time of an Intense Computing Algorithm in Software (PL-Program Logic) and RTL (PS-Processing System). This is done by simulating and synthesizing a chosen algorithm (here we have chosen the Sobel Edge Detection Algorithm) in Xilinx Vivado HLS 2015.3 design suite. In the initial phase the C++ code for the Sobel Edge Detection is developed in Xilinx Vivado HLS. The source code is then simulated using the test bench and the execution time is evaluated. The second phase involves developing the RTL part of the algorithm. This is done by the method of High-Level Synthesis (HLS) in Xilinx Vivado HLS. The RTL counterpart thus generated is then simulated and the execution time is calculated. The execution time for both the PL and PS implementations are compared and analyzed.

Index Terms—Execution Time, High Level Synthesis, PL, PS, Sobel Edge Detection.

I. INTRODUCTION

It is observed that any application that is developed in RTL work at a faster rate as compared to its software counterpart. But the number of skilled workers in the hardware design domain is comparatively lesser than that in the software domain. The primary reason for this can be the complexity involved in the development of an application in Hardware Description Language (HDL). An application can be developed with much lesser complexity in C or C++ when compared to the hardware description languages like Verilog or VHDL. But the fact that applications developed in HDL execute at a faster rate has motivated many developers to start implementing applications in HDL. It would be an added advantage if we can incorporate the huge mass of software developers into the hardware design. This idea of clubbing hardware and software developers is fruitful if and only if the execution time is lesser in hardware implementation when compared to the software counterpart. This analysis is what is done in this research paper. Software implementation of any algorithm can be easily converted to its equivalent Register Transfer Level by using Xilinx Vivado High Level Synthesis design suite.

In our research work, we have developed the C++ code for the Sobel Edge Detection Algorithm and converted the same to RTL using Vivado HLS 2015.3 design suite. The execution time for the hardware and software implementations were estimated and compared. The comparison of the execution time for both the implementation reveals that the RTL works at a faster rate when compared to the software implementation. Thus, any application can be easily accelerated if developed in RTL. The rest of the paper is organized as follows. Section II is the motivation behind the proposed research work is mentioned. Section III discusses previous works in this field. In Section IV we describe the detailed design and implementation of the proposes RTL to Software comparator which includes the architecture, the hardware specifications, software specifications and the algorithm used. In Section V we discuss the results obtained and in Section VI concludes the paper.

II. MOTIVATION

It is very important to cope up with the speed of the ever-increasing functionalities and performance of a system. So, it is critical that the system should work at a minimum possible execution time. It is observed that some algorithms work at a faster rate when designed in RTL even though the implantation is comparatively complex. Thus, in our research work we aim to check if this is applicable to majority of the algorithms. If so, it would be advantageous if the existing software applications are migrated to RTL so that the applications can be made to run at a faster rate.

III. RELATED WORKS

In [1] a dedicated device for edge detection by implementing Sobel operation on FPGA is discussed. Sobel operator was preferred over other operators, as they ensure much lesser deterioration in high levels of noise. The Sobel Edge detection can also be implemented taking into consideration parallel processing capability as discussed in [2] that ensures high reliability, flexibility and reconfigurability. This improved design reduces the speed problem for complex image edge detection. There are different methods to implement the algorithm in FPGA chip, one such method is proposed in [3] that works on Spartan -3-XC3S200 platform. The smoothing property of Sobel operator is well used in this implementation. Sobel Edge Detection is used in many high-speed applications. One such application and its implementation is mentioned in [4]. A method to reduce the delay by 50% compared to the normal implementation is also discussed in the paper.

978-1-5386-7595-3/19/$31.00 ©2019 IEEE
In [5] Sobel Edge detection of an image is implemented in Spartan 3 FPGA. It was found that this was more efficient due to the feature of parallelism provided by large embedded multipliers in the FPGA environment. Sobel and in video and image processing application uses FPGA and Prewitt approach for edge detection are widely used in video and image processing application. These applications are developed using different approaches. The FPGA and Model Based approach is discussed in [6-7] discusses an edge detection algorithm that makes use of Sobel operator based on FPGA architecture. The architecture proposed in this paper is designed using IEEE 754-1985 floating-point standard and VHDL hardware description language. The design is synthesized for Xilinx Virtex-6 FPGA chip that operates at 160 MHz frequency. In recent past many applications make use of Sobel extract system based on High Level Synthesis methods which are implemented on ZYNQ FPGA [8]. Different programing platforms can be used to implement applications based on Sobel Edge Detection algorithm. In [9] an effective comparison of the implementation of Sobel Edge Detection in CUDA (Compute Unified Device Architecture), GPU (Graphic Processing Unit) and FPGA is done by analyzing the computational efficiencies. In [10] The Coefficient Sobel Edge Detection can be efficiently implemented on a FPGA based design by in cooperating hardware and software components [10]. The efficiency was observed to be better compared to other implementations. Real time edge detection has become an important part in many image processing applications. [11] mentions about a Sobel Compass operator that can be used in applications implemented on FPGA resource optimized hardware architecture. The architecture thus developed computes the gradient for all directions using single processing elements Xilinx Spartan-3E is used to implement many hardware applications based on image processing. An algorithm to detect edge pixels in an image using Sobel Edge Detection algorithm is discussed in [12]. It is implemented on Spartan-3E and uses RS-232 serial protocol to transfer image to the FPGA. In [13], Xilinx ISE Design Suite-14.2 software platform is used to implement the algorithm of Sobel edge detection in Verilog language. A computationally inexpensive implementation of Sobel Edge Detection on FPGA is discussed in [14]. Xilinx Spartan-6 FPGA board is used to implement the same. A Sobel operator to detect edges from a 1024x1024 grey scale image is developed in [15-17]. The implementation of edge detection algorithm is done on the Spartan-3-XC3S200.

IV. DESIGN AND IMPLEMENTATION

The outline of the proposed design is shown in Fig. 1. The input is the algorithm chosen for the experiment. In our research work, we have chosen the Sobel Edge detection Algorithm. The RTL and the software implementations of the Sobel Edge Detection Algorithm are separately given as input to the Xilinx Vivado design suite. Once the implementations are simulated the Execution time analyzer analysis the execution time of both the PS and PL. The analyzed execution time for both the implementations are then given as output. By comparing the output values, we can determine which implementations works at a faster rate.

The detailed implementation is discussed in the Fig. 2. The Sobel Edge Detection algorithm is developed in C level and RTL. It is then given as input to both the Program Logic (PL) and Processing System (PS) of the Xilinx Zynq FPGA respectively. The ARM processor acts as the PL and the Xilinx Vivado HLS design suite acts as the PS. The Sobel Edge Detection algorithm developed in C++ is given to the ARM processor and the same developed in RTL is simulated in Xilinx Vivado HLS. The execution time for both RTL and C is then evaluated. The execution times are estimated from the simulation reports.

A. Architecture

The block level description of the detailed architecture is given below.

1) Input-Algorithm: The input here refers to the intense computing algorithm. Here we have chosen the Sobel Edge detection algorithm, that detects the edges of an input image stored in a predefined memory location or from a real time video input by creating image frames of the input video. The RTL and C++ implementations of the Sobel Edge Detection is developed and given as input to the Program Logic and Processing System of the FPGA model respectively.

2) Program Logic (PL): The Program Logic refers to the ARM processor of any FPGA board. The software implementation of the algorithm developed in C++ or SystemC is made to run on the PL. In our research work we have developed the C++ code for the Sobel Edge Detection in Xilinx Vivado HLS by invoking libraries and header files for image processing. The code developed is then verified using a testbench.

3) Processing System (PS): The Processing System refers to the hardware part of the FPGA. The RTL that has to be loaded onto the hardware is developed by the method of High Level Synthesis (HLS) using Xilinx Vivado HLS Design Suite. The C++ code is converted to RTL by synthesizing the C++ files in Vivado HLS. The Verilog and VHDL files thus generated are validated using the same testbench by co-simulating RTL/C.
4) Time Analyzer: The execution time for both the PL and PS are calculated in this phase. The execution time for the software implementation can be calculated by adding the code snippet for execution time display or by using Xilinx SDK. The RTL execution time is calculated from the synthesis report generated after synthesizing the C++ code in Xilinx Vivado HLS. It is calculated by multiplying the latency and time period.

5) Comparator: The comparator compares the execution time for both the implementation.

B. Sobel Edge Detection Algorithm

Edge Detection algorithms are critical in the field of Image Processing. Most of the critical information are enclosed within the edges of an image, thus raising the necessity of an efficient edge detection algorithm for image processing. There are many algorithms currently available for edge detection. The most commonly used algorithms are:

- Sobel Edge Detection
- Prewitt Operator
- Laplacian Operator
- Canny Edge Detection

The general criteria for edge detection include:

- Detection of edges with low error rate. This means the method should detect as many edges as possible.
- The edge point detected by the operator must accurately localise at the centre of the image as possible.
- The edges in the image should only be marked once, where ever possible the image noise should not create false edges. It can detect the edges of an image stored in a memory location or the edges of the frames obtained from a real time video input efficiently.

In our research work we have chosen the Sobel Edge Detection Algorithm for the detection of the edges of an image given as input. The Sobel Edge Detection algorithm is the classical algorithm used in most of the image processing applications. Initially the algorithm first converts the image frames to grey scale. The edges are then identified as shades of grey or white. It basically detects the change in gradient along the horizontal and vertical direction. This is done with the help of two convolution filters and they are combined to get the final gradient magnitude $G$

$$G = \sqrt{G_x^2 + G_y^2}$$

In Fig. 3, The original image is convolved with two 3x3 kernels to calculate the approximation of derivatives – one for the vertical changes and the other one for the horizontal. Here $G_x$ and $G_y$ are the two images obtained after the original image is convolved with the kernel. Each point of $G_x$ and $G_y$ contains the vertical and horizontal derivative approximations respectively. The two gradient approximation are then combined to get the final gradient magnitude $G$.

C. Software Specification

The C++ code for the Sobel Edge detection to be loaded on to the PL of the FPGA was developed. It basically consists of two C++ files namely core.cpp and testbench.cpp. The core.cpp file defines the actual functionality of the algorithm and the testbench.cpp basically tests the functionality of the algorithm implemented by comparing the edge detected output image obtained after running the core.cpp with the expected output image. Both the files are developed in Xilinx Vivado HLS 2015.3 by invoking necessary header files and libraries for image processing.

1) Core.cpp: The core.cpp takes as input set of images stored in predefined memory locations. The path to the input images are mentioned in the beginning of the code. Real time video input can also be given as input. In that case, frames are generated out of each video input and the edge detection will be performed on each frame. In our project we have only taken into consideration images that are stored in predefined memory locations. The input RGB image is converted to grey scale using the RGB to grey built function. The Sobel edge detection algorithm is then performed on grey scale image. The output edge detected grey scale image is then displayed. The code snippet to print the execution time using gettimeofday() function is also included in the core.cpp file.

2) Testbench.cpp: The core.cpp file that is developed can be validated for its correctness and functionality using the testbench.cpp program. Thus the simulation of the core.cpp is done by invoking the testbench. The testbench has the expected output image which it compared with the final output image of the core.cpp execution. If the actual output matches the test image the test is passed successfully, and it is concluded that the C specification works fine. This testbench can also be used to validate the RTL equivalent of the core.cpp that is developed.

D. Hardware Specification

The hardware specification or the RTL is developed by synthesizing the C specification i.e. the core.cpp file developed. This is done in Xilinx Vivado HLS 2015.3. The design suite can synthesize any C++ code to generate its equivalent RTL. The synthesis of the C Specification gives a set of Verilog and VHDL files with comparatively more lines of code. This is basically because all the inbuilt libraries and the functions are converted to its corresponding RTL. After synthesizing a synthesis report that contains the latency, the clock period, the memory usage etc. is generated.
By analyzing the report, the essential features of the hardware implementation can be easily understood. The RTL synthesis report also tells us whether we must opt for a better solution. A better solution can be obtained by creating a new solution and comparing the new and the old one using the compare solution tool of the Xilinx Vivado HLS. The validation of the RTL developed is carried out using the powerful co-simulation feature available in Xilinx Vivado HLS. This feature uses the test bench for the C simulation to validate the RTL files generated. The result of RTL co-simulation gives the detailed description of its functionality. It contains reports of timing analysis and performance estimation. The execution time of the RTL specification is found from the synthesis report as shown in Table I. The execution time is calculated as:

\[
\text{Execution time} = \text{Latency} \times \text{Time period of Clock Cycle} \quad (1)
\]

<table>
<thead>
<tr>
<th>CLOCK</th>
<th>TARGET</th>
<th>ESTIMATED</th>
<th>UNCERTAINTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP_CLK</td>
<td>10.00</td>
<td>10.05</td>
<td>1.25</td>
</tr>
</tbody>
</table>

V. EXPERIMENTAL RESULTS

The Sobel edge detection algorithm developed in both C++ and RTL were executed in Xilinx Vivado HLS design suite and the following results were obtained.

- Edge Detected image
- The execution time of C simulation
- The synthesis report of RTL
- The execution time of RTL simulation

The Sobel Edge detection algorithm developed in C++ is made to run on the Xilinx Vivado Design suite. The input to the algorithm is the image of an FPGA board as shown in Fig. 4. The image is accessed from the destination folder, whose path is mentioned in the code. The output of the experiment is the edge detected image shown in Fig. 5.

![Fig. 4. Input image for Sobel Edge Detection](image)

![Fig. 5. Edge Detected image](image)

The execution time of the C simulation is obtained using the gettimeofday() function is 40ms. The C implementation is then synthesised to obtain the corresponding RTL counterpart. The synthesis report of RTL is shown in Fig. 6. The synthesis report mainly contains three parts – Timing, Latency and Memory utilization. The timing summary gives the target time, estimated time and the uncertainty. The latency summary gives the maximum and minimum latency, maximum and minimum interval and the type. The utilization estimates give the number of Look Up Tables (LUTs) used, number of flip flops used etc.

The execution time for RTL can be calculated from the Timing summary as given below:

\[
\text{time for execution} = \text{latency x time period} = 942498*10.05\text{(ns)} = 9.47\text{ms.} \quad (2)
\]

![Fig. 6. Synthesis Report](image)

VI. CONCLUSION

By comparing the analysis reports, it is very clear that the execution time for RTL is much less when compared to the C specification. Hence the assumption we have made has been proved to be true. Therefore, it is advantageous to convert any C based application to RTL so that the execution time gets reduced and we get better results. The RTL thus generated can be further accelerated to improve the performance by finding the best solution. In this scenario if we can convert all the applications to RTL, we can incorporate the hardware and software developers for the further improvement of the technology.
REFERENCES


