Technical Report

Prabir Kumar Choudhury
School of Electrical Engineering and Computer Science
Kungliga Tekniska Högskolan
Stockholm, Sweden
pkch@kth.se

Nakita Sunil Oza
School of Electrical Engineering and Computer Science
Kungliga Tekniska Högskolan
Stockholm, Sweden
oza@kth.se

I. INTRODUCTION

The basic functionality of this application is a Sobel filter for edge detection in images [9]. They are on three implementations: single processor hardware design that is running a RTOS, single processor hardware design bare metal, and, five core bare metal hardware design. The design constraints considered here are as below:

For a 32x32 pixel image

- Throughput > 320 images/sec
- Throughput > 200 images/sec with total code size <= 45kB

The application is implemented on Altera Nios II FPGA hardware.

II. SPECIFICATION

A. *Sobel Filter Application*

Sobel filter algorithm is used in image edge detection by emphasizing it [9]. This algorithm [10] is a mathematical and the below steps are followed:

- GrayScale – The 32x32 pixel image is first converted into a grayscale image. Each RGB value of a pixel is converted into a single gray value. This uses Map/farm pattern. In Map pattern, a function is applied to all elements of a collection, usually producing a new collection with the same shape as the input[11].

$$grayValue = R * 0.3125 + G * 0.5625 + B * 0.125$$

- Resize Image – The grayed image is then resized by to half 16x16 pixel image. Each pixel value is calculated by averaging the 4 pixels into one value. The pattern used her is Map Reduce. In this pattern, every element in a collection into a single element using an associative combiner function (addition and in our case). Map is used on a set of elements and individual set is reduced[11].

$$newPixelValue = \frac{4\ old\ pixel\ values}{4}$$

In this process, the maximum and minimum value of a pixel in also identified as

$$brightnessMax = \max(allPixelValues)$$

$$brightnessMin = \min(allPixelValue)$$

- Correct Image – The values of maximum and minimum brightness show that pixel values in the image have a very small range. Correctness is applied to expand this range into [0,255].

$$diff(brightnessMax, brightnessMin)$$

$$1/\propto\ stretching\ the\ image\ range$$

- Sobel Filter – After correcting the image, Sobel filter is applied to the image. Convolution of the image with G_x and G_y is done. The modulo of the values obtained are added to get the final pixel value [7]. Each pixel values obtained here are divided by 128 to get a value from {[0,15]} to fit in the ASCII value range. Sobel uses Stencil pattern. In Stencil pattern, an elemental function accesses not only a single element in an input collection but also a set of "neighbors"[11].

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

Gx

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

Gy

$$|G| = |G_x| + |G_y|$$

$$G_f = \frac{G}{128}$$

- ASCII Values – The output of sobel is an image's pixel value with values ranging from {[0,15]}. This is mapped onto an ASCII level array and the printed on the console. This is where the edge detected is highlighted.

The complete application in single core bare metal is implemented using the above equations. Application flow is formally represented using Synchronous Data Flow (SDF) [10] as shown in Fig 1.

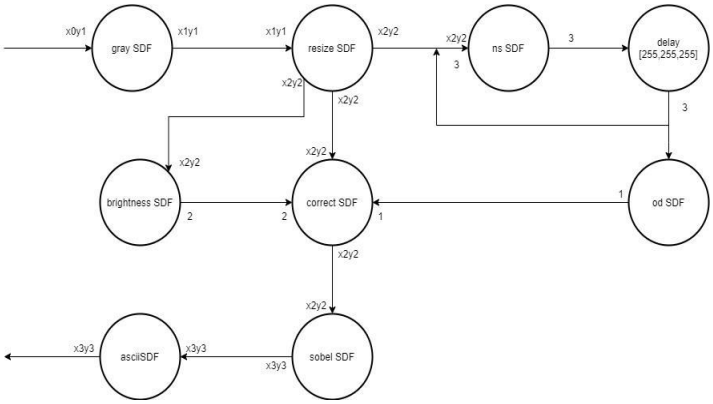

Fig. 1. Application SDF

SDF shows the data dependencies between actors, the expected input data and output data. The Actors also show which processes/functions can be implemented in parallel, as

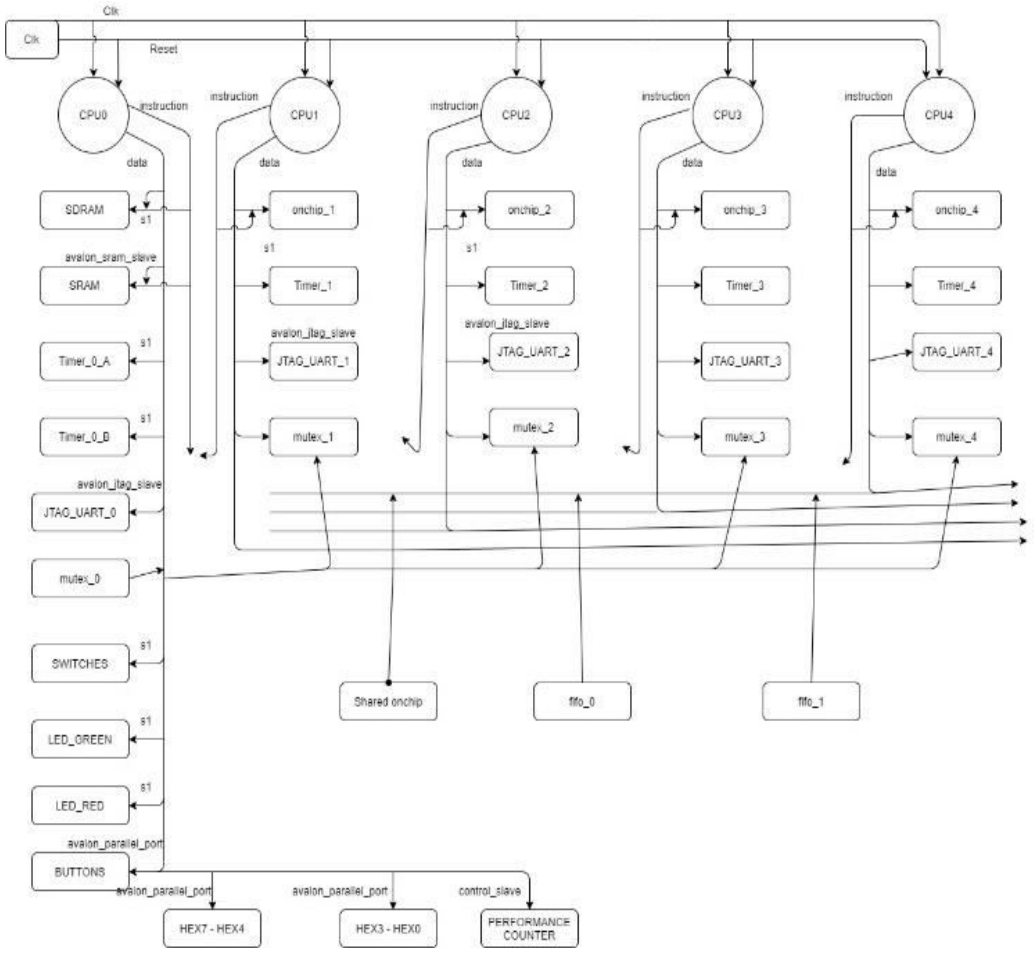

Fig. 2. Hardware design of Altera Nios II

well as deadlocks can be identified. Hence, identifying concurrency is easy, which is needed in multiprocessor architecture [3]. In the case of our application, example of concurrency is graySDF in which all the multiple rows can be processed independently, and hence in parallel.

For single processor RTOS, each actor is represented in an individual single task. MicroC/OS II is the RTOS for Altera Nios II. Semaphores are used to synchronize the task execution and flow. Highest priority is assigned to the grayscale task [6].

For multiprocessor processor bare metal design, we have divided the work between 5 workers. CPU_0 is the main CPU delegating tasks. Mutex is used to synchronize between processors [6]. CPU_0 monitors the status of all the processors via shared memory. For each processor, the input image and other variables needed for processing are stored in the shared memory by CPU_0.

B. Altera Nios II FPGA Hardware

The basic hardware architecture of Nios II with 5 CPUs is shown in Fig 2. Each CPU_0 to CPU_4 have access to Shared Memory of 8KB. CPU_1 to CPU_4 have an individual on-chip memory of 8KB.

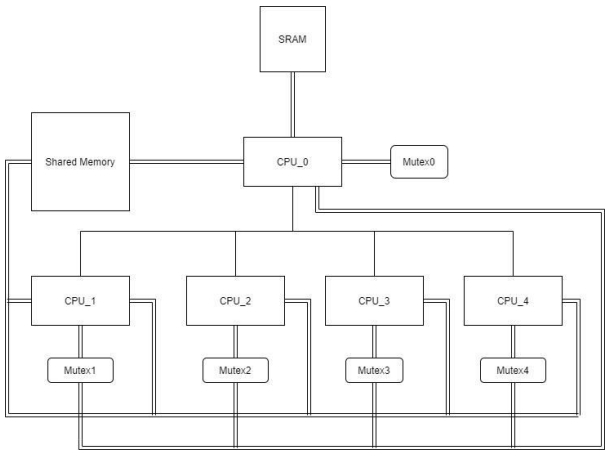

Fig. 3. Multiprocessor Architecture for Sobel Application

A Total of five processors are present in the multiprocessor implementation. CPU_0 being connected to the SDRAM and SRAM acts as the main worker. When the software is downloaded onto the FPGA, 5 JTAG connections download code into 5 processors: CPU_0,

CPU_1, CPU_2, CPU_3, CPU_4[4]. Image file is downloaded in SRAM. Since CPU_1 to CPU_4 do not have access to SRAM, this image is copied from SRAM (512KB) to Shared Memory which is of 8192 bytes.

The interconnection network here is Altera Avalon Switch Fabric. It is a network on chip connection with separate command and response network. Parallel communication is possible but not in case of same slave. In cases where the same slave is used, arbitration is used to decide which master get the access to slave first. The width of this connection is 32 bit, which allows integer copying instead of character copying, improving the speed of copying by more than two times.

On code download, CPU_0 starts and waits on CPU_1 to CPU_4 to start up. Their status is monitored on memory location 7200 in the shared memory. Once all the workers are up, the status flags are set to 1. This triggers the CPU_0 to start delegation of work to the workers.

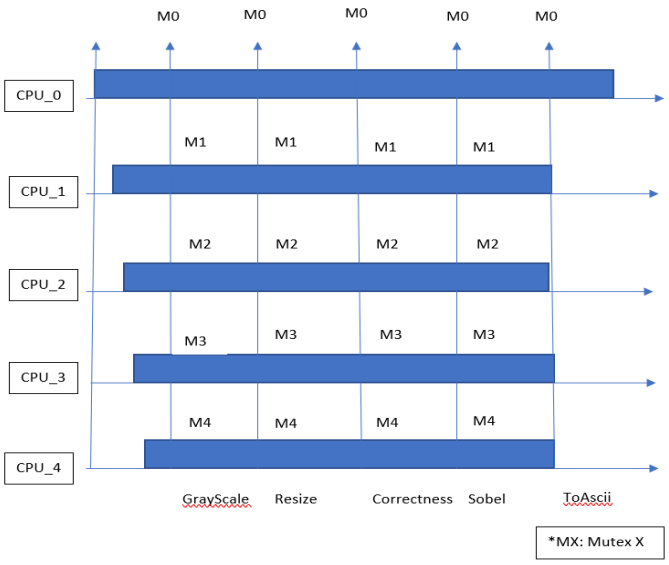

Fig. 4. Multiprocessor Scheduling for one image

All the intermediate image files generated while processing is stored in the shared memory. Shared memory also contained address information needed by each worker for their respective image section to be processed. Mutex was used to lock/unlock workers based on the needed synchronization [4].

Scheduling diagram for five processor bare metal design is shown in Fig 3. It shows that each CPU1 to CPU4 wait for Mutex 0 to be released. Once Mutex 0 is released, they lock it, acquire their Mutex (1-4) respectively and release Mutex 0 back. Once Mutex0 is released back, CPU_0 acquires it and starts with the further processing. CPU_0 is the only CPU to work on the last step of Ascii conversion.

III. Results and Discussion

The implementations work in two modes: Debug and Performance mode. Debug mode processes the images once and shows their corresponding edge output in ASCII. Performance mode processes the images 32 times and shows the performance metrics such as number of images executed, total execution time, execution time per image, throughput in images per sec.

The multiprocessor implementation resulted in 100 images per second through put. To reduce the time and code size for computation, while maintain correctness the further mentioned measure where taken. Multiplication was replaced by left and right shift operators. Tertiary operators where used instead of long if statements. Image was copied in block of 4 bytes rather than 1 byte by using integer instead of character. Memory reuse and achieved by overwriting memory locations' whose use is completed. Square and square root used in sobel filter was replaced by an approximated equation which used 2's complement only.

The table 1 shows the throughput in images/sec and code size for each implementation: single core bare metal, single core RTOS and five core bare metal design. The image is written back to SRAM for easy access to external modules

TABLE I.

Result of the three different Implementations

Implementations	**Measures**	
	Throughput	***Code Size***
Single processor Bare Metal	100	24KB
Single processor RTOS	83	91KB
Multiprocessor Bare Metal	250	23 + (3+3+3+3) = 35KB
Multiprocessor Bare Metal with Write Back	250	23 + (3+3+3+3) = 35KB

The design constraint of throughput > 200 images/sec with total code size <= 45KB is achieved by the Multiprocessor implementation. The advantage of using multiprocessor implementation is parallel processing is possible. Although, scheduling and copying the image is an overhead. With single processor RTOS implementation, the overhead is on context switching but the scheduling is less tedious.

Possible methods to increase the throughput and reduce the code size are as below:

- Inline functions – saves number of jumps, but it reduces the readability
- Perform grayscale on CPU_0 – On performing gray scale, the size of the image would be 32x32 instead of 32x32x3 bytes. This one-third reduction will save time in copying this image to the shared memory.
- On Chip Memory instead of Shared Memory - Move working image to on-chip memory instead of shared memory. The time taken on access shared memory is larger than access to on-chip memory
- Load balancing – Computation and copying of images can be balanced between the processors to obtain maximum throughput.

Appendix

For the initial project work, both Prabir and Nakita worked together on understanding the provided Sobel filter specification written in Haskell-ForSyDe. Further, they

worked on developing a basic flow chart on how the Sobel filter is to be implemented. Then, an SDF graph for the algorithm was developed.

Development and implementation of the software for single core and multicore was done together by Prabir and Nakita. Prabir primarily worked on developing the hardware design for the implementations with Nakita as a secondary on it, while Nakita developed a tool to convert images into arrays directly for ease of testing. Troubleshooting and result analysis was done by both Prabir and Nakita together.

Nakita worked on creating the project report with incorporating the developments in the project on weekly basis. Prabir reviewed the report and, any modifications needed were discussed and updated.

REFERENCES

[1] José Henrique de Magalhães Simões Calado, "Synchronization of Tasks in Multiprocessor Systems-on-Chip," Kungliga Tekniska Högskolan (KTH), 2010.

[2] Jean J. Labrosse. *MicroC/OS-II - The Real-Time Kernel*. CMP Books, Second edition, 2002.

[3] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*, Second Edition, MIT Press, 2017.

[4] Altera Corporation. Embedded Peripherals IP User Guide, 2011.

[5] Altera Corporation. Nios II Processor Reference Handbook, 2014.

[6] Altera Corporation. Nios II Software Developer's Handbook, 2014.

[7] HIPR2, Sobel Edge Detector , 2020 [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm, [Accessed: 15 Feb 2020]

[8] Wikipedia.org, Netpbm, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Netpbm#PPM_example. [Accessed: 7-Feb- 2020]

[9] Wikipedia.org, Sobel operator, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Sobel_operator. [Accessed: 25-Feb-2020]

[10] Hackage.haskell.org, 2020. [Online]. Available: https://hackage.haskell.org/package/forsyde-shallow. [Accessed: 7 Feb 2020]

[11] Michael McCool, Arch D. Robison, James Reinders. *Structured Parallel Programming Patterns for Efficient Computation,* Elsevier, 2012.