2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

SOBEL FILTER: Software Implementation to RTL using High Level Synthesis

Synthesizable C design and Synthesizable SystemC IP

Bhavna Aggarwal, CircuitSutra Technologies, Noida, India (bhavna.aggarwal@circuitsutra.com)

Umesh Sisodia, CircuitSutra Technologies, Noida, India (usisodia@circuitsutra.com)

Snigdha Tyagi, CircuitSutra Technologies, Noida, India (snigdha.tyagi@circuitsutra.com)

***Abstract*— This document discusses the effort of migrating an open source image processing algorithm to a synthesizable design using the High-Level synthesizable library and HLS tool by Mentor, called Catapult. It will inspect the software simulation timings & lines of code for Synthesizable C and Synthesizable SystemC designs against the original C implementation. This paper intends to attract the attention of all System/IP designers and Verification engineers towards the benefits of High-Level Synthesis.**

Keywords—High Level Synthesis (HLS), SystemC, Electronic System-level (ESL) Design, C/C++, SoC, RTL, Image Processing, Algorithm

I. INTRODUCTION

High Level Synthesis (HLS), also referred to as C/C++ synthesis or ESL synthesis, offers an automated design platform for developing scale algorithms that can transform to a hardware solution. It provides significant benefits over traditional hardware description language flows, in terms of leveraging the software platform for implementation of digital solutions for FPGAs and ASICs. The designer typically develops the behavior and the inter-connect protocol. HLS tools transforms the untimed or partially timed functionality into fully timed RTL implementations as well as creates a cycle accurate hardware module.

This paper showcases how the design engineers can become much more productive by learning how to decouple design from behavior, reduce their design efforts by writing up to 10x less code, and benefiting from 10 to 1,000x faster simulation speeds. Many semiconductor companies are designing custom SoCs for emerging domains like Vision, Speech, Video / Image processing, 5G, Deep learning, Artificial Intelligence, Machine Learning etc. In these domains lots of algorithms are already available as software implementations, either as free open source versions, or the companies have their own software implementations. Using HLS, designers can leverage the available software implementations to develop RTL designs much quicker, instead of developing the same from scratch.

Re-usability of C/C++ testbench for RTL verification will also be covered in this paper which presents a very significant benefit of HLS to verification engineers.

Technically, it will examine the efforts in migrating an open source image processing algorithm (implemented in vanilla C) to a synthesizable C design using the HLS synthesizable library and HLS tool by Mentor, called Catapult. As the efforts bore fruit, the next idea was to develop a Synthesizable SystemC intellectual property (IP) that encapsulates the Sobel's algorithmic behavior. As this idea was successful, we gained the confidence to dive deeper in this domain, also discussed briefly at the end.

To substantiate the work, we will share results that compare the software (C & SystemC) vs hardware (Verilog) implementations of the Sobel Filter operator, in terms of Line of codes and simulation timings.

II. SOBEL FILTER

Sobel filter, popularly called as SOBEL operator, is an edge detection image processing algorithm, commonly used for image segmentation and data extraction in computer vision and machine vision, where it creates an image emphasizing edges. The operator generates a 2D map of X and Y gradients of image intensity at each pixel, finds the direction of largest increases from light to dark and rates the change in that direction. Here is an example starting image shown on the left, the gradients, and the filtered result:

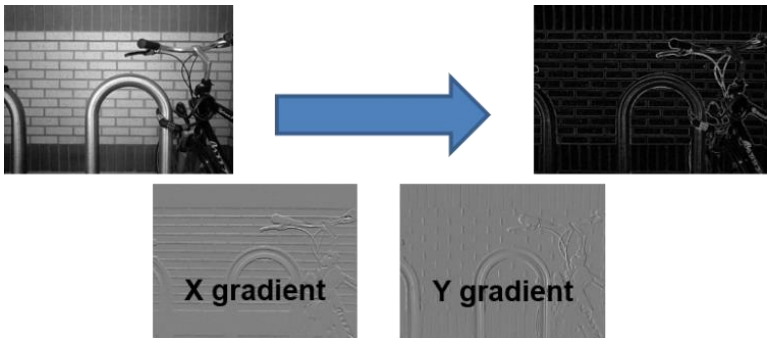

Figure 1. Sobel's Grayscale Filtered Output

An open source C implementation of the algorithm is available on GitHub. Thanks to Pedero Melgueira & Alessandro Capotondi for the contribution to open source world. This C based software design, invokes the operator using the following interface (function call)-

int sobelFilter (byte* rgb, byte* contour_img, int width, int height);

Where,

- **RGB** = RGB data of any colored PNG image
- **CONTOUR_IMG** = Convoluted Output Stream (Gray Data)
- **WIDTH** = Input for Horizontal resolution
- **HEIGHT** = Input for Vertical resolution

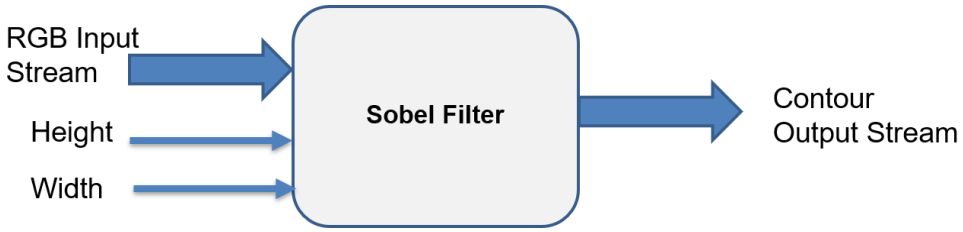

Figure 2. Sobel Filter Interface

III. TAKING THROUGH HIGH LEVEL SYTHENSIS

At first glimpse, it may appear that high level synthesis is a new language or methodology, and to make the most of it, this needs to be mastered like any other language/methodology. But that is not true. HLS is just a high-level abstraction to both software and hardware design phases, which aims to minimize the industrial gap between the two. Many companies like Mentor, Xilinx, etc. have come up with C/C++ based synthesizable libraries, that provides constructs, which can be used while developing the functional behavior of the algorithm. These constructs ensure that the software implementation can run through a tool, called as High-Level Synthesis tool, and automatically generate the RTL (Verilog/VHDL) digital solution. The Accellera community has collectively come up with a synthesizable subset of SystemC.

Now let us address the modifications that were required to open source C-implementation of Sobel Filter to derive the hardware solution using Mentor's synthesizable library and HLS tool – Catapult.

IV. SYNTHESIZABLE C IMPLEMENTATION

Firstly, the open source C implementation was modified to a synthesizable C design using the guidelines provided by HLS.

Not all C/C++ structures can be synthesized. There exists a synthesizable subset of the language that engineers utilize to write their code. Coding structures that cannot be synthesized include system calls, and input and output structures such as Printf and Scanf. There are other restrictions such as usage of pointers and pthreads are not allowed. Also, hardware is a static entity that serves to generate a well-defined behavior, hence no runtime dynamic actions are allowed.

Using Mentor's Catapult HLS tool, the modified synthesizable C design could successfully extract an RTL (Verilog/VHDL)

A. *Heirarchical Design*

Software design was converted to a hierarchical design with "sobelFilter" as the TOP module.

- Function calls inside the top operator, were marked as BLOCK, adding to the hierarchy of the design, as sub-modules.
- The hierarchical design may contain few inline functions as well, if required.
- Multiple calls to the same function need to be distinguished using a template ID, that translates to multiple instances of the same sub-modules. Refer "itConv" function interface in Figure 3.

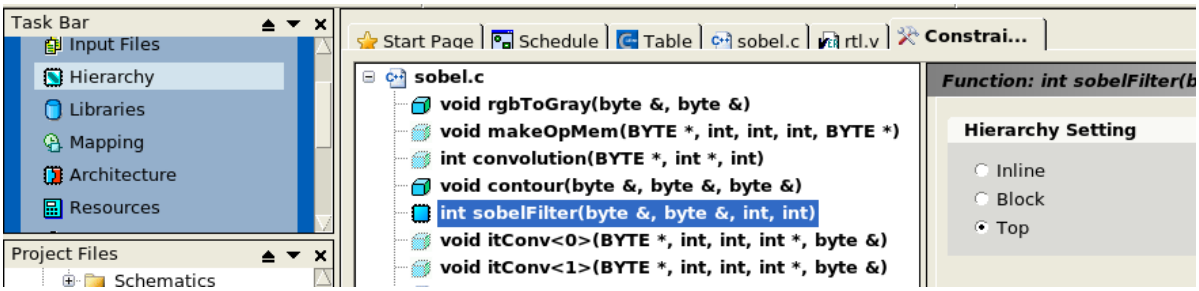

Figure 3. Sobel's Hierarchy View

Each software function call transforms to a RTL module, maintaining the design hierarchy.

B. *Pointers to Pointer, Dynamic Memory Allocation, File IOs*

Software implementation of algorithms tends to have pointers to pointers for inter-connection and performance, which are not supported for RTL extraction. Such members were either modified to static arrays or AC-channels, based on their usage.

- AC-channels are helpful in binding the function calls, as data streams from one function to other.
- Static arrays transform to registers, internal or external memory considering the MEM MAP THRESHOLD configuration.

Likewise, any dynamic memory allocations or File IOs are not supported. Hardware should be aware of memory requirements. Data to or out of the IP, should be made available in internal or external memories.

C. *Float/Double Data*

Floating point numbers in software are used to represent higher range of values for accuracy and precision. Considering the hardware overhead for saturation logic of floating-point numbers, C based float/double data members need to be modified to AC_FIXED or AC_FLOAT synthesizable constructs.

D. C/C++ Math Library

For commonly used math operations like square root, power, absolute etc. Mentor's HLS library provides equivalent math functions that can successfully transform to RTL logics. Hence, any C/C++ math functions used in algorithm, was modified to synthesizable math function from the "ac_math" library.

Table I. Some examples of "ac_math" library functions

C/C++ Math Functions	AC_MATH library synthesizable functions
pow	ac_pow_pwl
sqrt	ac_sqrt_pwl
abs	ac_abs

V. SYNTHESIZABLE SYSTEMC IP

Successful migration of open source software algorithm to hardware solution, led to the idea of having a SystemC IP that would encapsulate the functionality.

Growing popularity of SystemC for systems and hardware designs, motivated Accellera community to release a Synthesizable SystemC subset. Following the guidelines in the subset and using Mentor's Catapult HLS tool, we developed a subset compliant SOBEL SystemC IP that synthesizes to RTL (Verilog/VHDL)

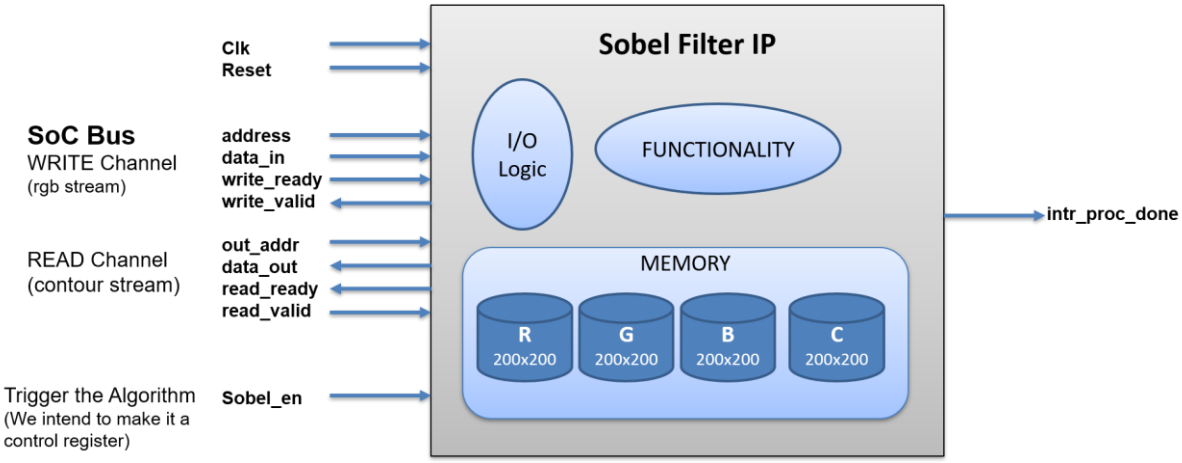

Figure 4. Sobel Filter SystemC IP block diagram

- Each THREAD/METHOD of the SystemC IP synthesizes to RTL module.
- Every thread and method should be sensitive at least one clock and one reset signal.
- For sharing data across threads, IP defines shared memory using Mentor's CCS Sample memory library.
- For bit-accuracy, it is advisable to use SystemC data types or AC algorithmic data types.

VI. VERIFICATION USING CATAPULT HLS TOOL

Verification holds utmost importance both in the hardware and software world. To validate intended design behavior, functional verification is the first and the foremost step. Considering the same, we ensured the following-

- Synthesizable C design and the RTL generated is functionally correct
- Synthesizable SystemC IP and the RTL generated is functionally correct

How did we do the verification? What was the effort invested? Did we write separate test for each design? Did we write a UVM or System Verilog stimulus to verify the RTL generated? Here are the answers:

High level synthesis tool offers a co-simulation verification environment "SCVerify" that uses same C/C++ testbench to verify both the software and hardware behavior. Thus, one C testbench and one SystemC Testbench Module is required to functionally validate the C design and SystemC IP respectively. And the same testbench is used to functionally validate the generated RTL. **Thus, no new UVM or System Verilog stimulus was required.**

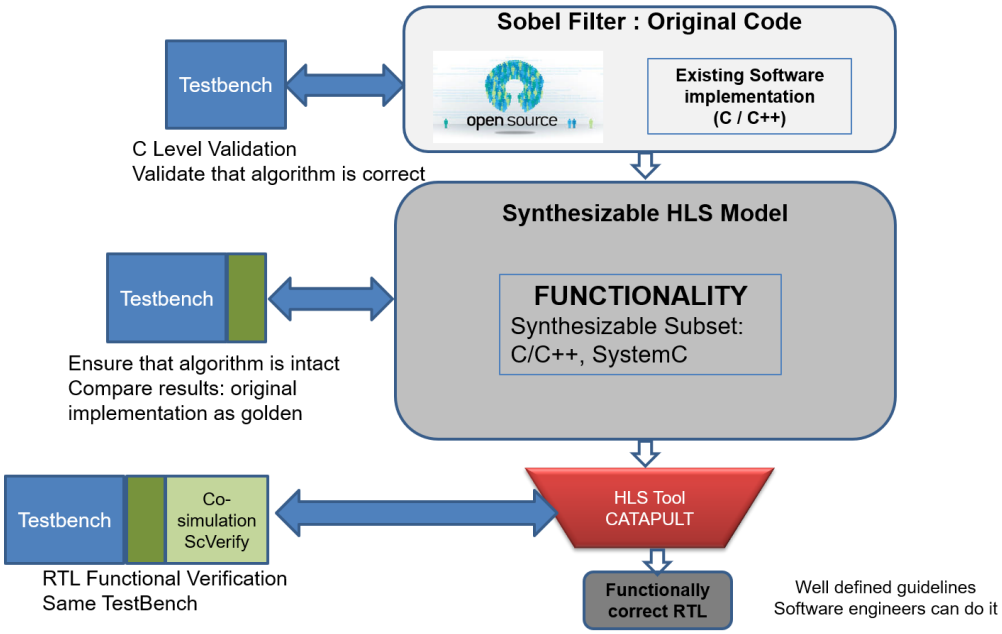

Figure 5. HLS Verification Flow

VII. SOFTWARE SIMULATION

Table II. depicts the software simulation timings for all the software versions of the SOBEL operator, executed using GCC. Undoubtedly C being one of the fastest programming languages, shall simulate the design in least time, clearly visible from the table. Synthesizable C/C++ and SystemC libraries add features to pure C/C++ language, thus adding to simulation timings.

Table III. SW simulation real time

SW version	REAL Time
Open Source C design	0m0.011s
Synthesizable C design	0m0.351s
Synthesizable SystemC design	0m0.970s

VIII. RTL (VERILOG) SIMULATION

Table III. depicts the RTL simulation timings for two variants of RTL generated from High level design of SOBEL operator, using QuestaSim. RTL simulation results also demonstrates that synthesizable constructs in SystemC offers better performance than ones available in synthesizable C/C++ library.

Table IIIII. HW simulation time

HW version	Simulation Time
RTL from Synthesizable C design	12760123 NS
RTL from Synthesizable SystemC design	04439999 NS

IX. HW vs SW Simulation of SystemC Design

Table IV. compares the simulation timings for hardware and software simulation of the SystemC IP that encapsulated the Sobel algorithm. Memory read and write latencies adds up to total clock cycles, during RTL simulation.

Table IV. SW/HW simulation real time

Simulation	Time
Software (using GCC)	1560006 NS
Generated RTL (using QuestaSim)	4439999 NS

X. Lines of Code (LoC)

Table V. outlines the number of lines of code, a developer may write to develop various versions of the same algorithm. It is distinctly visible that the RTL code would be 10-100x times more than the software version. However, modification of pure C/C++ implementation to synthesizable implementation is relatively simple, only 1.5-2x times effort required.

Table V. SW vs HW lines of code

Lines of Code	
Open Source C Implementation	***RTL (Verilog) Generated***
~150	-
Synthesizable C Design	***RTL (Verilog) Generated***
~210	~10k
Synthesizable SystemC Design	***RTL (Verilog) Generated***
~300	~5k

XI. Future Scope to the Study

- Optimize the RTL for Power Performance Analysis, by using HLS Tool directives, constraints, and code re-structuring for macro architecture.
- Expand the Sobel's SystemC design to use synthesizable hardware functions or components in Nvidia's open source HLS library, viz. MatchLib.

XII. Conclusion

This paper illuminates the benefits of High-Level synthesis in terms of automated RTL generation from an open source Sobel Filter C/C++ design and providing a co-verification platform for functional equivalence of software vs hardware. A software practitioner can easily take the original software implementation and generate a functionally correct RTL, without requiring in-depth knowledge of RTL. They just need to understand the synthesizable subset guidelines.

To maximize the HLS capabilities, both software and RTL design engineers can collaborate to develop the ASIC/FPGA solutions at the earliest. Also software test engineers and RTL verifications engineers can collaborate to validate and verify the solution in parallel, thus minimizing time to market by apportioning both RTL design and verification responsibilities.