SOBEL FILTER: Software Implementation to RTL using High Level Synthesis

Bhavna Aggarwal, CircuitSutra Technologies, Noida, India
Umesh Sisodia, CircuitSutra Technologies, Noida, India
Snigdha Tyagi, CircuitSutra Technologies, Noida, India
High-Level Synthesis
Chip designing at Higher Level of Abstraction

- HLS Tool: RTL Implementation from abstract description
  - 5 – 10x Less Code: Reduce design efforts
  - 10 – 1000x Faster Simulation: Increased productivity
- Bridges Hardware & Software Domain
- Existing C / C++ implementations available to start with
- C/C++ based synthesizable libraries
- SystemC Synthesizable Subset by Accellera
- Re-usability of C/C++ testbench for RTL verification
Sobel Filter – Taking through HLS

• Used in image processing and computer vision
• Used for edge detection
• 2D filtering operation
• Generates 2D Map of the gradient, X & Y Gradient of image intensity at each pixel
• Open Source Implementation
  • Contributors: Pedro Melgueira, Alessandro Capotondi
Sobel Filter
Architectural Interface : Open Source C

- `int sobelFilter (byte* rgb, byte* contour_img, int width, int height);`
HLS Implementation 1: Synthesizable C

We modified the C code to make it compliant with **Synthesizable Subset**. Used **Mentor Catapult** to synthesize C code and generate Verilog

- Created Hierarchical Design with SobelFilter as “Top”
- Sub-function calls marked as BLOCK
- Contains few inline functions
- Used AC_Channels as inter-connect between two blocks
HLS Implementation 1: Synthesizable C

NOT SUPPORTED

• Pointers-to-pointer
• Dynamic Memory allocations
• Float/Double
• C/C++ Math library functions
  – abs
  – pow
  – sqrt
• Unbounded Loops (i.e. loops with termination condition dependent on some variable) are expected to give bad results.
• Multiple same function calls

CHANGED TO

• STATIC ARRAY or AC_CHANNEL
• Bounded Arrays
• AC_FIXED
• Include <ac_math.h> library
  – ac_abs
  – ac_pow_pwl
  – ac_sqrt_pwl
• Bounded Loops using constant integers
• Templated for ID to differentiate the call instance
HLS Implementation 2: Synthesizable SystemC

We developed a SystemC IP that would encapsulate the functionality. Used Mentor Catapult to synthesize SystemC code and generate Verilog.

• **Top Level Design:**
  – SystemC Module (DUT)
  – C++ class should be instantiated in top level C function.
  – Identified using CCS_DESIGN construct

• **Threads/Methods:**
  – One clock and reset
  – Output ports must be part of RESET block in the THREAD
  – Bounded Loops
HLS Implementation 2: Synthesizable SystemC (Cont.)

• **Arrays:**
  – Static Array
  – Arrays synthesize to Memory: based on MEM_MAP_THRESHOLD configured

• **SC Module Data Members:**
  – Cannot be shared across Threads/Methods
  – Use shared memories, for sharing data.
  – Used CCS sample memory: mem_1r1w

• **Data Types**
  – ac_fixed
  – sc_uint

• **Math Operations**
  – Include <ac_math.h> library
    • ac_abs
    • ac_pow_pwl
    • ac_sqrt_pwl
Encapsulated the Algorithm in a Semiconductor IP

SoC Bus
WRITE Channel (rgb stream)

READ Channel (contour stream)

Trigger the Algorithm

Sobel Filter IP

I/O Logic

FUNCTIONALITY

MEMORY

Clk
Reset

address
data_in
write_ready
write_valid

out_addr
data_out
read_ready
read_valid

intr_proc_done

Sobel_en

R 200x200
G 200x200
B 200x200
C 200x200
Synthesis & Verification

Sobel Filter: Original Code

Existing Software implementation (C / C++)

FUNCTIONALITY
Synthesizable Subset: C/C++, SystemC

Testbench
C Level Validation
Validate that algorithm is correct

Synthesizable HLS Model

Testbench
Ensure that algorithm is intact
Compare results: original implementation as golden

HLS Tool CATAPULT

RTL Functional Verification
Same TestBench

Co-simulation ScVerify

Functionally correct RTL

Well defined guidelines
Software engineers can do it
Software Simulation REAL Time

<table>
<thead>
<tr>
<th>SW version</th>
<th>REAL Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Source C design</td>
<td>0m0.011s</td>
</tr>
<tr>
<td>Synthesizable C design</td>
<td>0m0.351s</td>
</tr>
<tr>
<td>Synthesizable SystemC design</td>
<td>0m0.970s</td>
</tr>
</tbody>
</table>

© Accellera Systems Initiative
## HW vs SW Simulation of SystemC Design

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software (using GCC)</td>
<td>1560006 NS</td>
</tr>
<tr>
<td>Generated RTL (using QuestaSim)</td>
<td>4439999 NS</td>
</tr>
</tbody>
</table>
## RTL (Verilog) Simulation

<table>
<thead>
<tr>
<th>HW version</th>
<th>Simulation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTL from Synthesizable C design</td>
<td>12760123 NS</td>
</tr>
<tr>
<td>RTL from Synthesizable SystemC design</td>
<td>04439999 NS</td>
</tr>
</tbody>
</table>
## Lines of Code (LoC)

<table>
<thead>
<tr>
<th>Lines of Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Open Source C Implementation</strong></td>
<td>RTL (Verilog) Generated</td>
</tr>
<tr>
<td>~150</td>
<td>~10k</td>
</tr>
<tr>
<td><strong>Synthesizable C Design</strong></td>
<td>RTL (Verilog) Generated</td>
</tr>
<tr>
<td>~210</td>
<td>~10k</td>
</tr>
<tr>
<td><strong>Synthesizable SystemC Design</strong></td>
<td>RTL (Verilog) Generated</td>
</tr>
<tr>
<td>~300</td>
<td>~5k</td>
</tr>
</tbody>
</table>
Refining the code for HLS...

**Original Code**
- Existing Software implementation (C / C++)

**Synthesizable HLS Model**
- **FUNCTIONALITY**
  - Synthesizable Subset: C, C++, SystemC
- **OPTIMIZATION**
  - Further Code Restructuring
    - Optimization directives in the code: Loop unrolling, Loop pipelining etc..
    - Capture Macro architecture: Registers, Memory, Interfaces etc..)
  - HLS Tool: Directives
  - Constraints

**Test Suite**
- C Level Validation
  - Validate that algorithm is correct
  - Ensure that algorithm is intact
  - Use same Test Suite
  - Compare results: original implementation as golden
- RTL Functional Verification
  - Same Test Suite

**Co-simulation**
- HLS Tool
  - Functionally correct RTL
- HLS Tool
  - Optimized RTL

**C Level validation**
- Faster Simulation
- Less code to verify
- Catch bugs early
- Reduce efforts for RTL verification

**Test Suite**
- Develop Comprehensive Test Suite for high functional coverage
- Reuse:
  - Test suite of original software
  - Compliance test suite of protocols

© Accellera Systems Initiative
Future Scope to Study

- Optimize & Refine 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.
Thank you for your time.