S2CBench : Synthesizable SystemC Benchmark Suite for High-Level Synthesis

Benjamin Carrion Schafer\textsuperscript{1}, Anshueree Mahapatra\textsuperscript{2}
The Hong Kong Polytechnic University
Department of Electronic and Information Engineering
b.carrionschafer@polyu.edu.hk\textsuperscript{1}, anushree.mahapatra@connect.polyu.hk\textsuperscript{2}
Outline

• Motivation for a Synthesizable SystemC Benchmark Suite
• S2C Bench overview
  – 12 synthesizable design
  – 1 non synthesizable (tests floating point and trigonometric functions)
• Benchmark composition overview
• Detail benchmark characteristics
  – Size
  – Complexity
  – Arithmetic operations
• How to download
• Conclusions
Motivation for S2C Bench (I)

- HLS tools evaluation cycles is typically very long (multiple tools are evaluated using multiple designs)
  - Companies don’t have the expertise in HLS
  - Companies don’t have ANSI-C, C++ or SystemC models for their RTL designs in order to compare the QoR of the HLS tools
- C/C++ supported by most vendors include vendor specific constructs. E.g. data types, port declarations
- SystemC only true language supported by all major HLS vendors

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Tool Name</th>
<th>Supported Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cadence (Forte)</td>
<td>Csynthesizer</td>
<td>SystemC</td>
</tr>
<tr>
<td>Cadence</td>
<td>C-to-Silicon</td>
<td>C, C++, SystemC</td>
</tr>
<tr>
<td>Calypto</td>
<td>CatapultC</td>
<td>C++, SystemC</td>
</tr>
<tr>
<td>NEC</td>
<td>CyberWorkBench</td>
<td>C, SystemC</td>
</tr>
<tr>
<td>Xilinx</td>
<td>Vivado HLS</td>
<td>C, C++, SystemC</td>
</tr>
</tbody>
</table>
Motivation for S2CBench (II)

• Dedicated HLS benchmarks available are based on ANSI-C, e.g. CHStone
• Typically Multimedia applications written in ANSI-C used, e.g. MiBench or MediaBench or need to create their own ones:
  – Need to be edited to be made synthesizable for state of the art commercial HLS tools
  – Do not support fixed point data types
  – Do not test specific HLS features (are just a collection of C programs)

⇒ SystemC Benchmark suite will enable the direct comparison of commercial HLS tools
S2CBench Overview

• 12+1 SystemC Benchmarks which comply with latest SystemC synthesizable subset draft (12 synthesizable+1 non synthesizable)
• From different domains
  – Automotive
  – Security
  – Telecommunication
  – Consumer
• Control dominated and Data dominant designs
• Each test unique HLS features
  1. Tool language support (e.g. templates, structures, fixed point data types)
  2. Synthesis optimizations (e.g. loop unrolling, pipelining, function inlining, array synthesis)
  3. Tool performance (e.g. running time, accuracy of synthesis report)
## S2CBench: 12+1 designs

<table>
<thead>
<tr>
<th>Design</th>
<th>Type</th>
<th>Domain</th>
<th>Optimizations Tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>qsort</td>
<td>dd</td>
<td>Auto/In</td>
<td>Loops, arrays, functions pointers</td>
</tr>
<tr>
<td>sobel</td>
<td>dd</td>
<td>Auto/In</td>
<td>Loops, functions, IO array expansion, multi-dimensional arrays expansion, fixed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>arrays (ROM, logic)</td>
</tr>
<tr>
<td>aes cipher</td>
<td>dd</td>
<td>Security</td>
<td>IO array expansion, multi-dimensional arrays expansion, large fixed arrays</td>
</tr>
<tr>
<td>kasumi</td>
<td>dd</td>
<td>Security</td>
<td>Multi-processes, delay report accuracy</td>
</tr>
<tr>
<td>md5c</td>
<td>dd</td>
<td>Security</td>
<td>#define macros, delay report accuracy</td>
</tr>
<tr>
<td>snow3G</td>
<td>dd</td>
<td>Security</td>
<td>Templates, delay report accuracy, function synthesis</td>
</tr>
<tr>
<td>adpcm</td>
<td>Cd</td>
<td>Telecom</td>
<td>Structure synthesis</td>
</tr>
<tr>
<td>FFT</td>
<td>dd</td>
<td>Telecom</td>
<td>Floating point, trigonometric functions</td>
</tr>
<tr>
<td>FIR</td>
<td>dd</td>
<td>Consu</td>
<td>IO array expansion, arrays, loops, functions, sum of products</td>
</tr>
<tr>
<td>Decimation</td>
<td>dd</td>
<td>Consum</td>
<td>Resource sharing across loops, fixed point data types</td>
</tr>
<tr>
<td>Interp</td>
<td>dd</td>
<td>Consum</td>
<td>Polynomial decomposition, fixed point data types, sum or products</td>
</tr>
<tr>
<td>IDCT</td>
<td>dd</td>
<td>Consum</td>
<td>#include statement to initialize arrays, loops, functions,</td>
</tr>
<tr>
<td>Disparity</td>
<td>cd/dd</td>
<td>Consum</td>
<td>Hierarchical design, multi-dimensional array expansion, synthesis running time</td>
</tr>
</tbody>
</table>
Benchmark Block Diagram

- TB sends stimuli data stored in files (editable) to UUT
- TB receives the data and compares it against golden output (stored in file)
- TB reports if results match or not
- Option to dump VCD file
Detail Benchmark Contents for each Design

• Makefile
  – Make : generates executable binary (default option)
  – Make wave : Generates binary which dumps a VCD file
  – Make debug: generates debug version (e.g. with gdb)
  – Make clean: cleans object file

• SystemC files
  – Main.cpp : top module includes UUT and TB
  – <benchmark>.cpp/.h : Main design description
  – tb_<benchmark>.cpp/.h : Testbench, sends receive and compares results against golden output

• Stimuli :
  – Inputs.txt : test vectors
  – Outputs_golden.txt : golden outputs
  – BMP : inputs for Sobel and disparity estimator
Quick – Quick Sort

• Description
  – sort design sorts data in ascending order using the well-known quick sort algorithm

• Main options to be tested
  – loop unrolling
  – array synthesis (register or memory)
  – function synthesis with pointer argument support
Sobel

• Description
  – edge-detection algorithm that takes a bitmap image directly as the input and returns a new bitmap image solely consisting of the edges of the original image.

• Main options to be tested
  – nested loop unrolling and pipelining optimizations
  – I/O ports expansion (expand inputs specified as arrays to individual ports)
  – multi-dimensional arrays expansion
  – fixed arrays synthesized as logic or ROMs
  – pointer arguments to functions
AES - Advanced Encryption Standard Cipher

• Description
  – Advanced Encryption Standard Cipher encryption algorithm performs AES encryption

• Main options to be tested
  – contains a large number of small for loops having inter-loop data dependencies.
  – input port expansion
  – array synthesis (memory or registers)
  – function synthesis (inline, goto)
  – large fixed arrays synthesized as logic or ROMs.
Kasumi

• Description
  – block cipher algorithm used in mobile communication systems
  – Composed of two sc_threads and multiple functions

• Main options to be tested
  – Contains large amount of logic operations (e.g. and, or, xor). HLS tools are notably not efficient, for accurately estimating the critical path of these applications, because the discrete delay of all the operations are simply added, thus overestimating the critical path
  – Multi-process systems verification
MD5C - Message Digest Algorithm

• Description
  – generates hash functions and check data integrity.

• Main options to be tested
  – functions synthesis
  – arrays of different bit widths
  – different levels of loop nesting
  – extensive use of *define* macros (language support)
Snow3G

• Description
  – stream cipher that produces a key stream that consists of 32-bit blocks using a 128-bit key

• Main options to be tested
  – Support of templates. A variable length multiplication operation is performed in this algorithm, which may be easily simplified using templates
  – Loops, functions and array synthesis
ADPCM - Adaptive Differential Pulse-Code modulation (encoder part only)

• **Description**
  – accepts 16-bit Pulse Code Modulation (PCM) samples as input and converts them into 4-bit samples

• **Main options to be tested**
  – loop unrolling, function synthesis, array synthesis
  – support for structures synthesis
FFT – Fast Fourier Transform (not synthesizable)

- **Description**
  - converts time/space to frequency and vice versa
- **Main options to be tested**
  - floating point operations and trigonometric functions
  - not synthesizable as per latest synthesis draft

Included because most commercial HLS provide tools to deal with floating points and trigonometric functions

Helps evaluation engineers understand how these operations are supported
FIR – Finite Impulse Response Filter

• Description
  – 10-tap FIR filter algorithm designed for 8-bit integer operations.

• Main options to be tested
  – loop unrolling and pipelining
  – automatic array expansion of the I/O ports
  – pointers to functions
Decimation Filter

• Description
  – 5-stage decimation filter. Consists of 5 FIR filters cascaded together where the output of one stage is the input to the next stage.

• Main options to be tested
  – resource sharing of the Multiply Accumulate (MAC) operations across loops
  – generated RTL is able to preserve the sum of product (SoP) construct → logic synthesis tool can optimize the construct further
  – fixed-point data types and its different rounding and saturation modes.
Interpolation Filter

• Description
  – 4-stage interpolation filter

• Main options to be tested
  – automatic polynomial decompositions. Significant area reduction can be obtained if polynomials can be decomposed into terms, so that the total number of arithmetic operations required is reduced → Mathematical optimization of HLS tool
  – fixed-point data types and its different rounding and saturation modes
IDCT - Inverse Discrete Cosine Transform

• Description
  – expresses a finite sequence of data points in terms of, a sum of cosine functions of different frequencies

• Main options to be tested
  – initialization of an array using \#include statement
  – loops, functions, array synthesis
Disparity – Stereoscopic Disparity Estimator

• Description
  – estimates the disparity in a stereoscopic image.
  – It is the largest of all the designs and consists of 4 processes executed in parallel

• Main options to be tested
  – Almost all previously mentioned optimizations
  – Synthesis running time of the HLS tool (main thread contains a large number of loops leading to extreme long synthesis run times)
  – Verification of Multi-process (threads) systems
Detailed Benchmark Characteristics

- Size, complexity, arithmetic operations

<table>
<thead>
<tr>
<th>Program Characteristics</th>
<th>qsort</th>
<th>sobel</th>
<th>aes cipher</th>
<th>kasumi</th>
<th>md5c</th>
<th>snow 3G</th>
<th>adpcm</th>
<th>fft</th>
<th>fir</th>
<th>decm</th>
<th>interp</th>
<th>idct</th>
<th>disparity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code</td>
<td>204</td>
<td>269</td>
<td>429</td>
<td>415</td>
<td>467</td>
<td>522</td>
<td>270</td>
<td>334</td>
<td>176</td>
<td>422</td>
<td>231</td>
<td>450</td>
<td>634</td>
</tr>
<tr>
<td>Processes</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Function</td>
<td>1</td>
<td>2</td>
<td>11</td>
<td>5</td>
<td>7</td>
<td>11</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Number of arrays</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>13</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>5</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>if statement</td>
<td>1</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td>19</td>
<td>0</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>for statement</td>
<td>5</td>
<td>8</td>
<td>20</td>
<td>12</td>
<td>8</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>15</td>
<td>5</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>while statement</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>Test vector</td>
<td>.txt</td>
<td>.bmp</td>
<td>.txt</td>
<td>.txt</td>
<td>.txt</td>
<td>.txt</td>
<td>.txt</td>
<td>.txt</td>
<td>.txt</td>
<td>.txt</td>
<td>.txt</td>
<td>.txt</td>
<td>.bmp</td>
</tr>
<tr>
<td>Operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Addition/subtract</td>
<td>8</td>
<td>26</td>
<td>65</td>
<td>44</td>
<td>284</td>
<td>11</td>
<td>15</td>
<td>17</td>
<td>7</td>
<td>50</td>
<td>14</td>
<td>123</td>
<td>33</td>
</tr>
<tr>
<td>Multiplications</td>
<td>0</td>
<td>2</td>
<td>16</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>33</td>
<td>2</td>
</tr>
<tr>
<td>Divisions</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Logic operations</td>
<td>7</td>
<td>0</td>
<td>22</td>
<td>39</td>
<td>274</td>
<td>67</td>
<td>9</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>33</td>
<td>13</td>
</tr>
<tr>
<td>Comparisons</td>
<td>0</td>
<td>17</td>
<td>29</td>
<td>22</td>
<td>16</td>
<td>10</td>
<td>16</td>
<td>10</td>
<td>0</td>
<td>43</td>
<td>8</td>
<td>36</td>
<td>42</td>
</tr>
</tbody>
</table>
Publicly Available

www.s2cbench.org
http://sourceforge.net/projects/s2cbench/

S2CBENCH

Synthesizable SystemC Benchmark Suite

Home  Download  Contact

Download:

The S2CBench provides 12 programs written in synthesizable SystemC language. Each benchmark is designed for specific domains such as multimedia, digital signal processing, security, image processing, etc. The programs are provided with the objective to enable researchers analyze their innovative algorithms and techniques and help users compare the quality of results of state of the art commercial high level synthesis tools available in industry.
Summary and Conclusions

• A benchmark suite in a common language supported by all major HLS vendors

• Each benchmark tests unique HLS features
  1. Tool language support
  2. Synthesis optimizations
  3. Tool performance

• Benchmarks include testbench with inputs, golden outputs and option to generate VCD file

• Publicly available at www.s2cbench.org or sourceforge.net