National Science Foundation's Industry/University Cooperative Research (I/UCRC) Program

High-Level Synthesis Techniques for In-Circuit Assertion-Based Verification

CHREC NSF Center for High-Performance Reconfigurable Computing

UF UNIVERSITY of FLORIDA

Virginia Tech VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

BYU BRIGHAM YOUNG UNIVERSITY

THE GEORGE WASHINGTON UNIVERSITY WASHINGTON DC

John Curreri

Ph.D. Candidate of ECE, University of Florida

Dr. Greg Stitt

Assistant Professor of ECE, University of Florida

Dr. Alan D. George

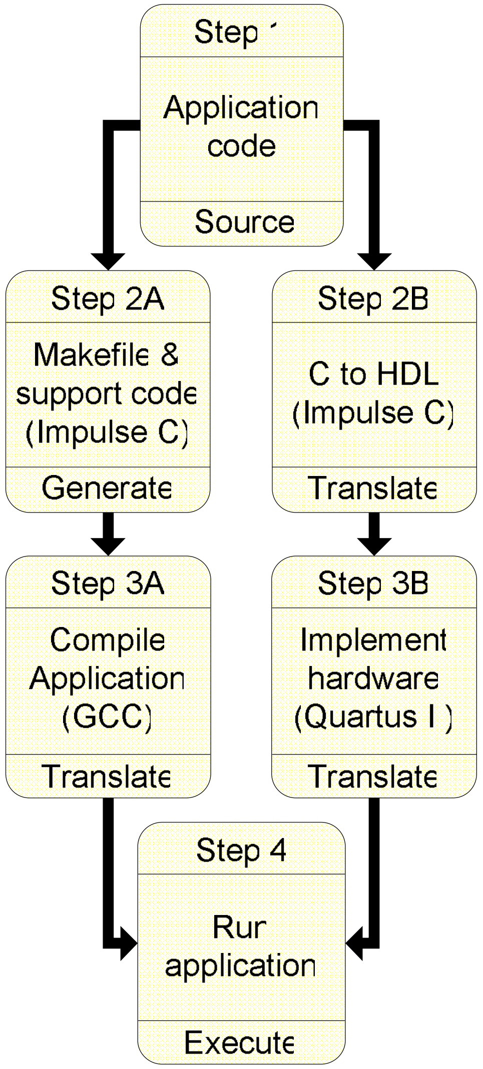
Professor of ECE, University of Florida

Name=speaker

April 19, 2010

High-Level Synthesis

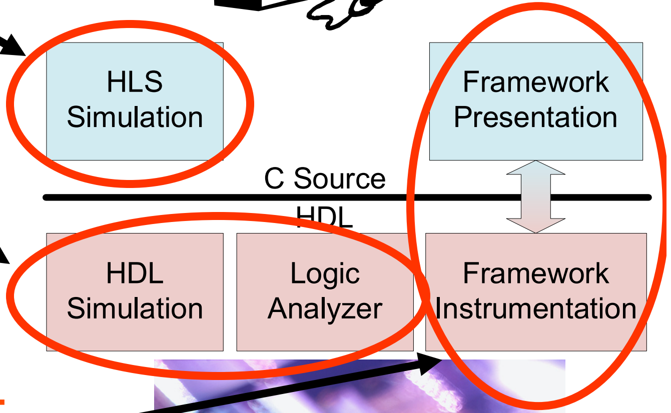
- Ease of programming
  - No HDL coding required for application acceleration
  - Abstraction of communication function
- Provides built-in support
  - Methods to gain speedup
    - Pipelining of loops
    - High-performance library functions
  - Mitigation of race conditions
    - Signals
    - Semaphores
  - Communication
    - Buffered streaming transfers
    - DMA transfers



e.g. Impulse-C tool flow

High-Level Synthesis Verification

- HLS simulation
  - Executes source on CPU
  - Does not provide accurate verification
- HDL simulation and analyzers
  - Provide accurate verification
  - Productivity lowered by the need to understand machine-generated HDL
- Verification framework needed
  - Maintains high abstraction level and provides accurate verification
  - Programmers are not required to understand HDL or HDL tools



ANSI-C Assertion Debugging

- Error checking
  - Used to check if variables are in an acceptable range
- Example usage

  -
    ```
    int num,i,x[10];
    while(num==0)
    {
      num=x[i];
      i++
      assert(i<10);
    }
    ```
- Failure actions
  - Failure information is printed to stderr
    - assert_test.c:7: main: Assertion `i==0' failed.
    - W:*X*: Y: Assertion ' Z' failed.
    - W = file name; X = line number; Y = function name; Z= *expression*
  - Program terminated using abort()
- Assertion checking switch
  - #define NDEBUG
  - Disables assertion checking

Assertion Code

```
assert(i == 0);
```

Assertion Output

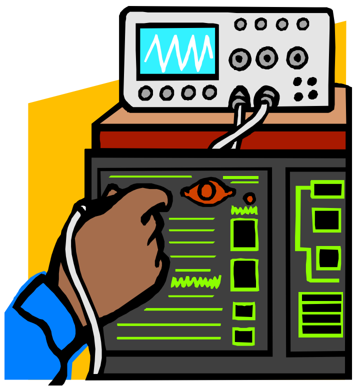
```
assert_test.c:7: main:
Assertion 'i == 0' failed.
```

CHREC NSF Center for High-Performance Reconfigurable Computing

UF UNIVERSITY of FLORIDA · Virginia Tech VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY · BYU BRIGHAM YOUNG UNIVERSITY · THE GEORGE WASHINGTON UNIVERSITY WASHINGTON DC

Related Research

- Assertion languages and libraries
  - VHDL assertion statements
  - SystemVerilog Assertions (SVA)
  - Open Verification Library (OVL)
  - Property Specification Language (PSL)
- Commercial assertion tools
  - Temento's DiaLite
- Academic debugging tools
  - Camera's debugging environment
  - Sea Cucumber synthesizing compiler
- Logic analyzers
  - Xilinx's ChipScope
  - Altera's SignalTap



Verification Framework Overview

- Assertion-based verification usage
  - Document and check for conditions that should never occur during execution
- In-circuit verification process
  - Open application files in GUI
  - Single-click instrumentation
    - Converts assertions to *if* statements
    - Generates communication channels
    - Creates software function to display errors and program abort if failure detected
  - Use standard tool flow to compile/execute
  - Assertion failure output during execution
- Seamlessly transfer assertions from simulation to runtime

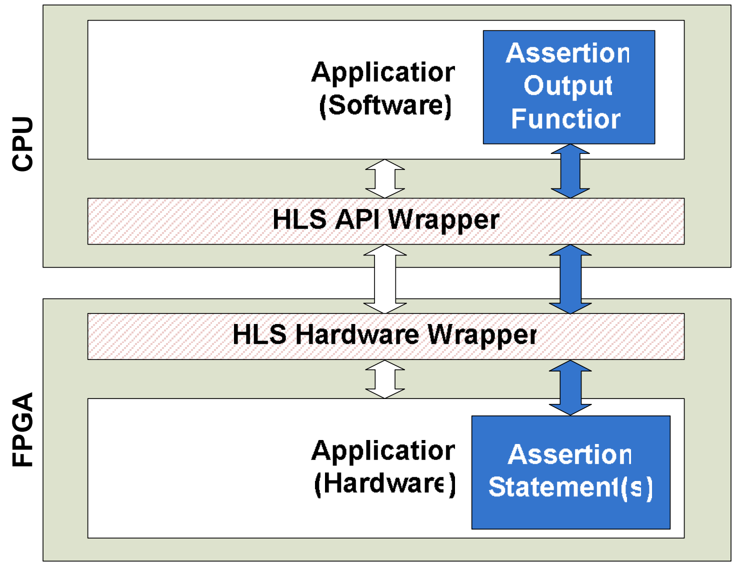
HLS Assert

Assertion Code

```
assert(address >= 0);
```

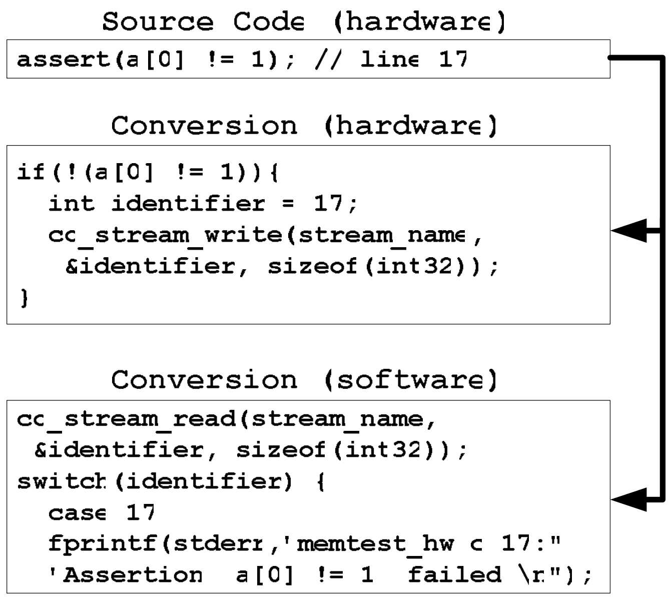
Assertion Output

```
memtest_hw.c:14: Assertion
 'address >= 0' failed.
```



Standard Assertion

- Assertion conversion
  - FPGA side
  - *Assert* statement changed to *if* statement
- False evaluation
  - FPGA side
  - Sends a message with a unique identifier
- Assertion notification
  - CPU side
  - Function to receive, decode, and display failed assertions
  - ANSI-C output format



Source Code (hardware)

```
assert(a[0] != 1); // line 17
```

Conversion (hardware)

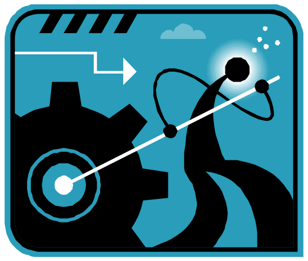
```
if(!(a[0] != 1)){
  int identifier = 17;
  cc_stream_write(stream_name,
   &identifier, sizeof(int32));
}
```

Conversion (software)

```
cc_stream_read(stream_name,
 &identifier, sizeof(int32));
switch(identifier) {
  case 17
  fprintf(stderr,'memtest_hw c 17:"
  'Assertion  a[0] != 1  failed \r");
```

CHREC NSF Center for High-Performance Reconfigurable Computing

Assertion Optimizations



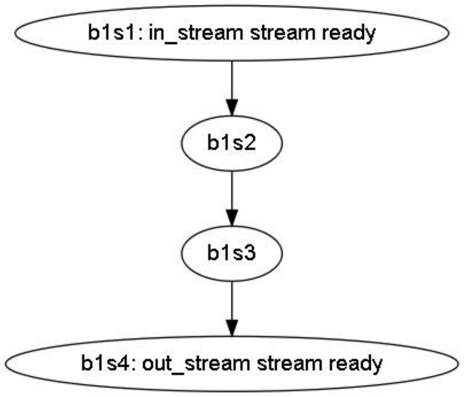
- Parallelization
  - Assertion checking can slow down the application
    - Move assertion checkers to a separate parallel process
  - Communication can slow down pipelined assertions
    - Move communication calls to a third process

Clock cycles	Standard		Optimized	
1	App. line 1		App. line 1	
2	Check assertion		App. line 2	Check assertion
3	Failure communication		App. line 3	Failure communication
4	App. line 2		App. line 4	

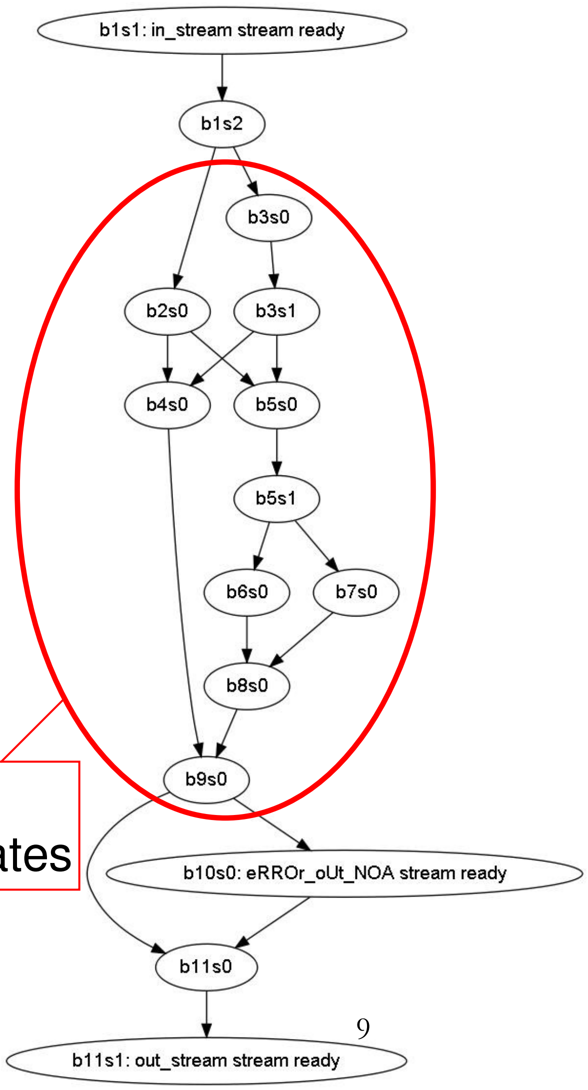
State-machine Comparison

assert((j <= 0 || a[0] == i) && (b[0] == 2 || i > 0));

Original

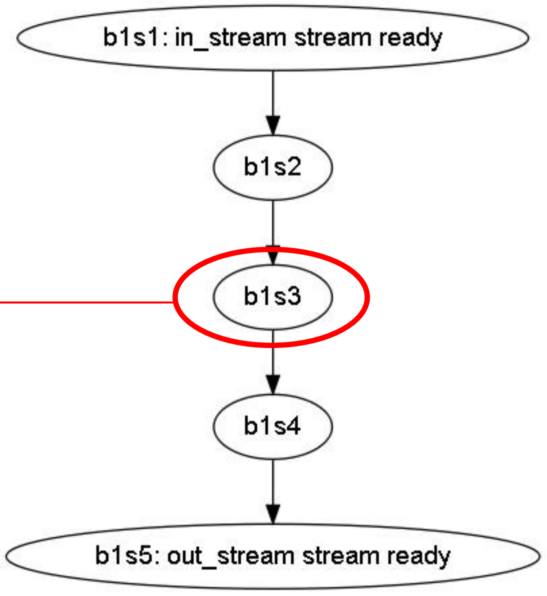


Standard assertion



b1s1: in_stream stream ready
b1s2
b3s0
b2s0
b3s1
b4s0
b5s0
b5s1
b6s0
b7s0
b8s0
b9s0
b10s0: eRROr_oUt_NOA stream ready
b11s0
b11s1: out_stream stream ready

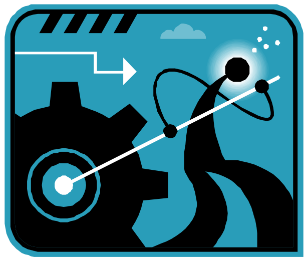
Optimized assertion



Boolean operators require many additional states

Array data retrieval requires an extra state

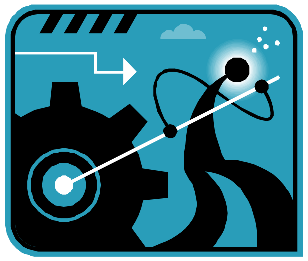
Assertion Optimizations



- Resource replication
  - Application and assertion are competing for data access
    - Replicate data structure (e.g., duplicated block RAM that is dedicated for assertion read access)

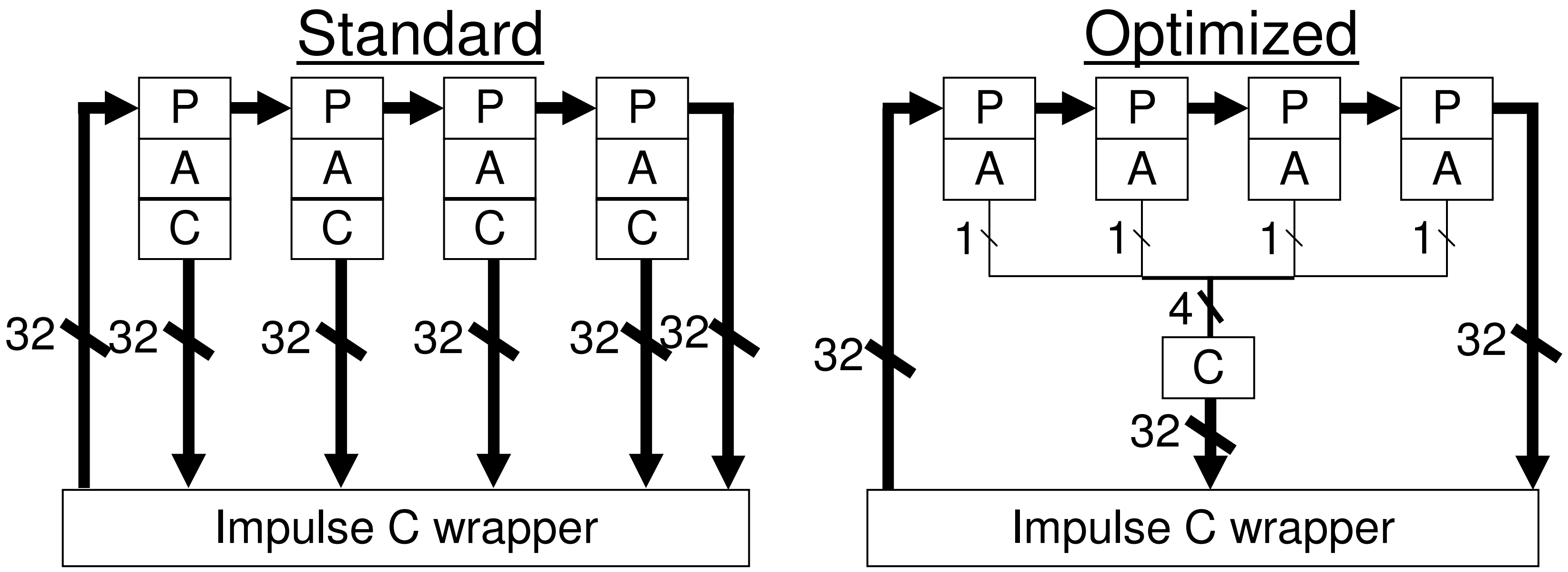
Clock cycles	Standard		Optimized	
1	App. read a[0]		App. read a[0]	
2		Assert read a[1]	App. read a[1]	Assert read a[1]
3	App. read a[1]	Communication	Application	Communication
4	Application		Application	

Assertion Optimizations



- Resource sharing
  - Minimize FPGA resources usage of assertions
    - Reuse assertion checking and communication resources amongst all assertion calls

P	Application process
A	Assertion checker
C	Failure communication



In-Circuit Verification Case Study

- Assertion in Line 6 shows Impulse-C translation mistake
  - Simulation
    - 64-bit comparison of 4294967286 > 4294967296 evaluates to false
  - Execution on target platform
    - 5-bit comparison of 22 > 0 evaluates to true
- Assertion in Line 8 shows user translation mistake
  - Impulse-C simulation requires C code for HDL function
  - Behaviors of C code and HDL may not be the same
  - Assertions can be used to check that behaviors match

```
1   co_uint64 c2, c1;
2   co_int32 address, array[20], out;

3   c2 = 4294967286; c1 = 4294967296;
4   if (c2 > c1) address = c2 - c1;
5   else address = 0;
6   assert(address >= 0);

7   out = user_hdl(address);
8   assert((30 > out) && (out > 20));
9   array[address] = out;
```

Impulse-C design, XD1000, Stratix-II (EP2S180)

Debugging Case Study

- assert(0);
  - Used to “trace” execution
  - To find when an application fails to complete (hangs)
  - Positive indicator rather than negative indicator
- NABORT
  - Stops application from aborting
- Output comparison
  - Line numbers of the failed assertions
    - Software simulation vs. platform execution
    - Hang occurred at a memory read at end of loop
- Solution
  - Memory read replaced with memory write
  - Correction allowed the process to complete execution

```
263   for ( i = 1; i < status; i++ ) {
264     IF_SIM(printf("HW:DE:%i\n",i);)
265     assert(0);
…
392     assert(0);
393     co_memory_readblock(…);
394     assert(0);
395   }
396   assert(0);
297   co_signal_post(done, status);
```

Impulse-C design, XD1000, Stratix-II (EP2S180)

Application Case Studies

- Triple-DES
  - Optimized assertions
    - No latency overhead
    - FPGA overhead to the right
  - Standard assertions
    - More ALUT (+0.125%)
    - Higher freq. (144.74MHz)

EP2S180	Original	Assert	Overhead
Logic Used (out of 143520)	13677 (9.53%)	13851 (9.65%)	+174 (+0.12%)
Comb. ALUT (out of 143520)	7929 (5.52%)	8025 (5.59%)	+96 (+0.07%)
Registers (out of 143520)	10019 (6.98%)	10055 (7.01%)	+36 (+0.03%)
Block RAM (9383040 bits)	222912 (2.37%)	223488 (2.38%)	+221 (+0.04%)
Frequency (MHz)	145.71	141.98	-3.73 (-2.56%)

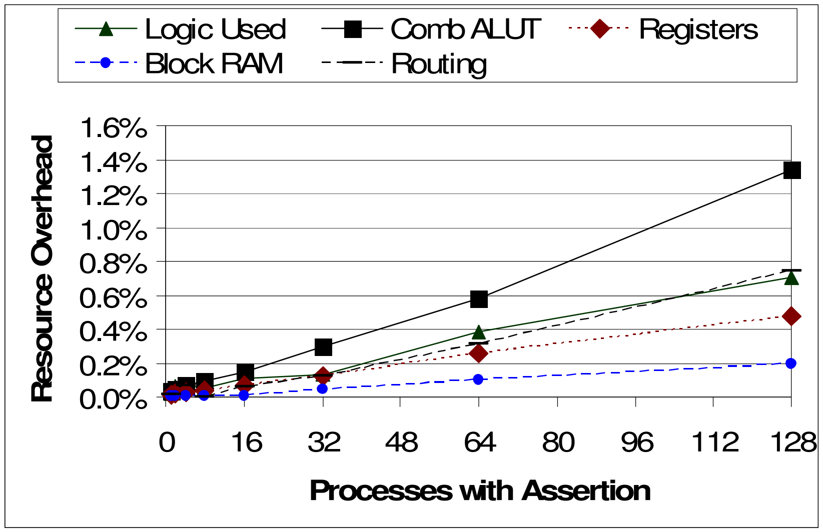
- Edge detection
  - Optimized assertions
    - No performance overhead
    - FPGA overhead to the right
  - Standard assertions
    - Less ALUTs (+0.03%)

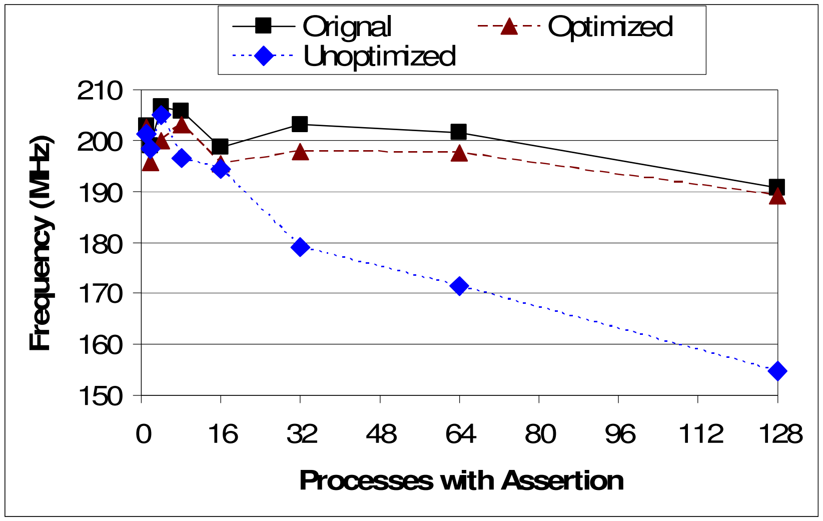
EP2S180	Original	Assert	Overhead
Logic Used (out of 143520)	12250 (8.54%)	12273 (8.56%)	+23 (+0.02%)
Comb. ALUT (out of 143520)	6726 (4.69%)	6809 (4.75%)	+83 (+0.06%)
Registers (out of 143520)	9371 (6.53%)	9417 (6.56%)	+46 (+0.03%)
Block RAM (9383040 bits)	141120 (1.50%)	141696 (1.51%)	+576 (+0.01%)
Frequency (MHz)	77.52	79.31	+1.79 (+2.31%)

Impulse-C design, XD1000, Stratix-II (EP2S180)

Scalability Case Study

- Resource overhead
  - Optimized shown to right
  - 128 processes
    - 4.07% ALUTs standard
    - 1.34% of ALUTs optimized
      - Over a 3x improvement
- Frequency overhead
  - Shown in graph to right
  - 128 processes
    - 154MHz standard
      - 18.8% overhead
    - 189MHz optimized
      - 18.5% improvement



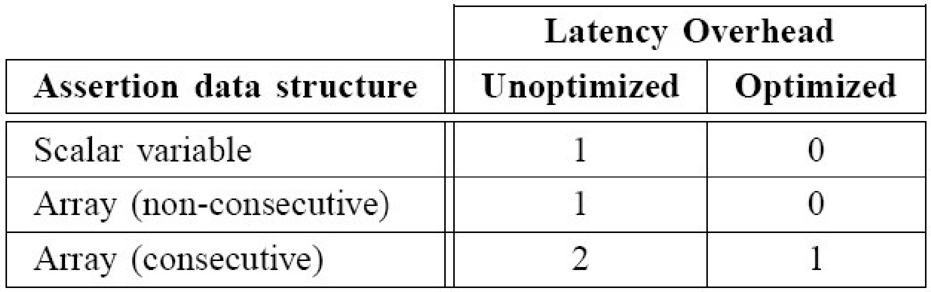


Impulse-C design, XD1000, Stratix-II (EP2S180)

Performance Overhead Case Study

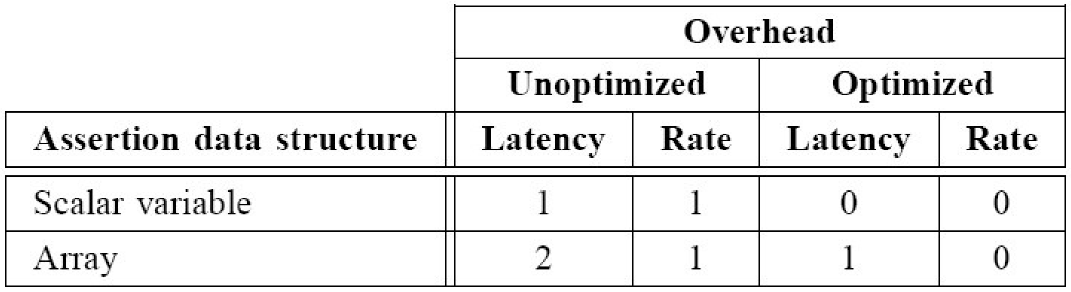
- Single-comparison assertion
  - Lower bound on optimization improvements
- Scalar variable
  - Optimized overhead reduced to zero
- Array
  - Optimized overhead
    - Rate reduced to zero
    - Latency reduced

TABLE 3
SINGLE-COMPARISON ASSERTION



	Latency Overhead	
Assertion data structure	Unoptimized	Optimized
Scalar variable	1	0
Array (non-consecutive)	1	0
Array (consecutive)	2	1

TABLE 4
PIPELINED SINGLE-COMPARISON ASSERTION

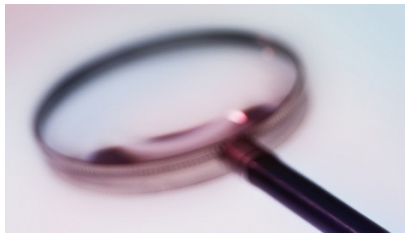


	Overhead			
	Unoptimized		Optimized	
Assertion data structure	Latency	Rate	Latency	Rate
Scalar variable	1	1	0	0
Array	2	1	1	0

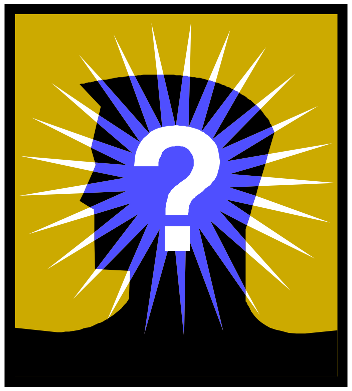
Impulse-C design, XD1000, Stratix-II (EP2S180)

Conclusions

- Created first framework/tool (to our knowledge) for HLS in-circuit assertion-based verification
  - Familiar and easy to use ANSI-C assertions
  - Automated conversion for Impulse C
- Application case studies performed
  - Low area and frequency overhead
  - Highly scalable
  - Minimal to no change of application's state machine
- Future work
  - Fully automate generation of optimized assertions
  - Add capability to check timing via assertions



Questions



References

1. Deepchip, "Mindshare vs. marketshare," http://www.deepchip.com/items/snug07-01.html, March 2008.
2. D. Pellerin and Thibault, *Practical FPGA Programming in C*. Prentice Hall PTR, 2005.
3. D. Poznanovic, "Application development on the SRC Computers, Inc. systems," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, April 2005, pp. 78a–78a.
4. Impulse Accelerated Technologies, "Codeveloper's users guide," 2008.
5. GNU, "The GNU C library reference manual," http://www.gnu.org/software/libc/manual, March 2009.
6. Accellera, "SystemVerilog 3.1a language reference manual," http://www.eda.org/sv/SystemVerilog 3.1a.pdf, May 2004.
7. Accellera, "OVL open verification library manual, ver. 2.4," http://www.accellera.org/activities/ovl, March 2009.
8. Accellera, "PSL language reference manual, ver. 1.1," http://www.eda.org/vfv/docs/PSL-v1.1.pdf, June 2004.
9. M. Pellauer, M. Lis, D. Baltus, and R. Nikhil, "Synthesis of synchronous assertions with guarded atomic actions," in *Formal Methods and Models for Co-Design, 2005. MEMOCODE '05. Proceedings. Third ACM and IEEE International Conference on*, July 2005, pp. 15–24.
10. M. Boule, J.-S. Chenard, and Z. Zilic, "Assertion checkers in verification, silicon debug and in-field diagnosis," in *Quality Electronic Design, 2007. ISQED '07. 8th International Symposium on*, March 2007, pp. 613–620.

References

11. M. Kakoee, M. Riazati, and S. Mohammadi, “Enhancing the testability of RTL designs using efficiently synthesized assertions,” in *Quality Electronic Design, 2008. ISQED 2008. 9th International Symposium on*, March 2008, pp. 230–235.

12. K. Camera and R. Brodersen, “An integrated debugging environment for fpga computing platforms,” in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, Sept. 2008, pp. 311–316.

13. Xilinx, “ChipScope pro 10.1 software and cores user guide,” http://www.xilinx.com/ise/verification/ chipscope pro sw cores 10 1 ug029.pdf, March 2008.

14. Altera, “Design debugging using the SignalTap ii embedded logic analyzer,” http://www.altera.com/literature/hb/qts/qts qii53009.pdf, March 2009.

15. K. Hemmert, J. Tripp, B. Hutchings, and P. Jackson, “Source level debugger for the sea cucumber synthesizing compiler,” in *Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on*, April 2003, pp. 228–237.

16. G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

17. XtremeData Inc., “XD1000 FPGA coprocessor module for socket 940,” http://www.xtremedatainc.com/pdf/XD1000 Brief.pdf.

18. NIST, “Data encryption standard (DES),” http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf, October 1999.