Interra Systems, Inc.

Bluespec Testing Results: Comparing RTL Tool Output to Hand-Designed RTL

April 20, 2004

TABLE OF CONTENTS

1. INTRODUCTION **3**

2. PURPOSE **3**

3. DESIGNS **3**

TABLE 1: DESIGN SUMMARIES 3

4. DESIGN METHODOLOGY **4**

5. VERIFICATION METHODOLOGY **4**

6. SYNTHESIS SETTINGS AND RESULTS **4**

TABLE 2: AREA OPTIMIZED 5
TABLE 3: TIMING OPTIMIZED 7

7. CONCLUSIONS **8**

1. Introduction

This document discusses Interra Systems experience with Bluespec tools and the results of its tests comparing the quality of Bluespec's RTL results with those of hand-crafted designs.

2. Purpose

The purpose of this document is to describe Interra Systems observations on Bluespec Compiler (BSC) capabilities. BSC compiles the design specified in the Bluespec SystemVerilog (BSV) high-level synthesis environment into RTL Verilog and a C-model. The following sections discuss the methodology adopted to analyze the quality of BSC tools.

3. Designs

Interra developed 25 small and medium sized designs to test the capabilities of BSC and results are encouraging. The different categories of designs are state machines, pipelined datapath components and control logic. The complete list of designs along with its description is given below in Table 1.

TABLE 1: Design Summaries

S. No.	Designs	Description
1	gray_code_converter	Design converts a binary number into its equivalent gray code.
2	priorityencoder	A priority encoder encodes input data into a coded form such that if two or more single bit inputs are at logic 1 then the input with the highest priority will take precedence and its particular coded value will be output.
3	paritychecker	A parity checker gives the odd even parity of the data passed to it.
4	readWriteFsm	FSM designs having three states read, write and delay.
5	barrel_shifter	The barrel-shifter is a combinational block used to shift data left or right.
6	speedFSM	FSM design having four states. This is a cyclic FSM where changes in state occur in either direction (forward or backward) depending upon input signals which guide the direction of state change.
7	rippleadder	The rippleadder module implements an 8-bit ripple carry adder (r = a + b + cin) implemented through a series of full adders.
8	angularfsm	FSM design having eight states. The states change in a clockwise or anti-clockwise direction depending upon the input signals.
9	one_hot_encoded_fsm	7 state one-hot FSM implementation of a Bus arbiter.
10	pattern_detector	The pattern detector module takes in an 8-bit wide data bus as input and sets the output high for one clock cycle, as and when the following pattern is detected: F6 F6 F6 28 28 28 using a FSM.
11	wallace_multiplier	This is a Wallace tree implementation of a 4 X 4 multiplier using full adder and half adder.
12	handshakeprotocol	Simple handshake protocol implemented using a 3-state state-machine.
13	traffic_light_controller	A traffic light controller between a highway and a farm road with highway road given priority over farm road.Farm road light becomes green only after its determined that a car is present on that road. Implemented using state-machines for state change of farm and highway light and timers are used for setting duration of the signals red, yellow, green.
14	rotorscontrol	Simple control machine for two stepper rotors which can not have the same position at any one time. They are moved in the clockwise or counter clockwise direction by control signals sent from outside.
15	SequentialMultipler	Simple sequential multiplier using the shift add method. The multiplier and multiplicand are assumed to be in signed numbers.

16	shift_add	Simple shift-add implementation of a sequential 4X4 multiplier. It requires 4 cycles to compute the product.
17	threewayroundrobinarbiter	This is a round robin arbiter, with initial preference given to Process A, then B then C. There is a time out counter which is reset when the time for a particular grant is over
18	divider	This implements the division operation using subtraction. The design generates a quotient, remainder and overflow.
19	cache coherence	This design models a three processor subsystem, each with its own cache. The cache maintains coherence by using a three state (DSI) snooping protocol. The write policy assumed is write-through, no-write-allocate.
20	boothMultiplier	Implements a booth multiplication alogorithm for two 16 bit numbers.
21	fibonacci	The fibonacci series module takes in a 16-bit wide data input as an index to output the corresponding number in the fibonacci series.
22	lifo	The lifo (Last In First Out) module implements a simple single clock stack, with explicit underflow and overflow flags.
23	fifo1	Implements a simple single clock fifo with explicit signals generated for full and empty.
24	factorial	The factorial module takes in a 16-bit wide data input and outputs its corresponding factorial.
25	random_number_generator	It generates a 32 bit random number based on a few arithmetic operations performed on predefined parameters.

4. Design Methodology

Interra Systems selected 25 designs from its design database. These designs are synthesizable through Synopsys Design Compiler and have a complete verification environment. These designs have been previously done and are part of a test suite that is used to test other EDA tools.

A team of 3 designers worked on the 25 designs to develop BSV-version of the designs and also created a BSV testbench for C-verification. These designers did not have prior experience working on these designs. It took 3 weeks to gain working knowledge of BSV language and environment by each of the designer (each had prior Verilog experience) and 4 to 5 weeks time to code these 25 designs in BSV. The BSV testbench development took 2 weeks for the 25 designs.

5. Verification methodology

The BSC generates the RTL verilog and cycle-accurate C-model for a design.

The generated RTL Verilog was verified using existing Verilog testbench/testvectors and results were compared against golden RTL Verilog results.

The generated C-model was verified by writing the BSV testbench using golden Verilog test vectors.

6. Synthesis Settings and Results

Synopsys Design Compiler (DC) tool was used for synthesis. Both Timing and Area constraints were applied. All Synopsys DC compile efforts were also explored.

Both hand-designed Verilog and BSC generated RTL verilog for 25 designs were synthesized using the same constraints and results were compared. The reports are below.

For the first set of tests, the timing constraints were not aggressively set, resulting in more aggressive area results by the synthesizer. The following test profile will be referred to as "Area Optimized":

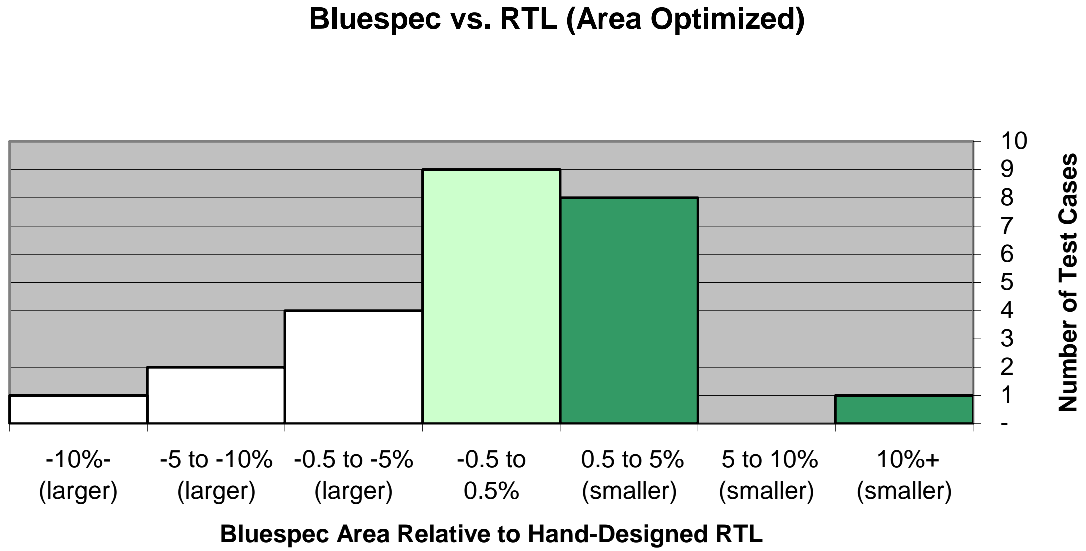
TABLE 2: Area Optimized

Design Compiler Report (20th April 2004)

DC version:	Version 2002.05 for linux -- May 03, 2002
Total number of designs analyzed:	25
Number of combinatorial designs:	7
DC constraints used:	DC compile was run with medium effort, timing constraints are given in table
BSC Version and build used:	Bluespec Compiler, version 3.8.12 (build 4315, 2004-04-18)

S. No	Testcases	DC Constraints ClockPeriod/ Max Delay(ns)	Total Area (golden)	Total Area (bsv)	Golden RTL Slack	BSV RTL Slack	Num. Flops (golden)	Num Flops (bsv)
1	gray_code_converter	30	9	9	28.94	28.94	0	0
2	priorityencoder	30	21	21	27.01	27.01	0	0
3	paritychecker	30	23	23	25.70	25.70	0	0
4	readWriteFsm	30	20	20	25.79	25.79	2	2
5	barrel_shifter	30	34	34	28.85	28.85	0	0
6	speedFSM	30	33	33	24.95	25.25	2	2
7	rippleadder	30	86	86	14.01	14.01	0	0
8	angularfsm	30	66	63	24.23	23.42	3	3
9	one_hot_encoded_fsm	30	99	116	23.04	22.10	7	7
10	pattern_detector	30	67	67	23.94	23.00	3	3
11	wallace_multiplier	30	142	141	19.73	20.80	0	0
12	handshakeprotocol	30	109	112	22.86	23.39	8	8
13	traffic_light_controller	30	215	211	18.36	17.63	11	11
14	rotorscontrol	30	259	249	19.26	17.51	4	4
15	SequentialMultipler	30	340	361	21.16	21.16	23	23
16	shift_add	30	391	399	15.96	15.30	26	26
17	threewayroundrobinarbiter	30	399	385	18.87	20.94	18	18
18	divider	40	350	347	0.11	0.56	0	0
19	cache_coherence	30	352	382	16.75	14.74	18	18
20	boothMultiplier	35	974	822	2.53	2.54	39	39
21	fibonacciseries	30	914	877	6.57	6.70	82	82
22	lifo	35	1764	1850	2.63	0.85	141	141
23	fifo1	30	1926	2018	9.48	9.44	146	147
24	factorial	50	1611	1605	2.06	1.32	50	50
25	random_number_generator	90	9278	8947	6.00	0.12	96	96

The results of the Area Optimized tests are summarized in the following graph, which compares the % difference between the Bluespec compiled RTL and hand-designed RTL (the comparison is between the areas of each design):

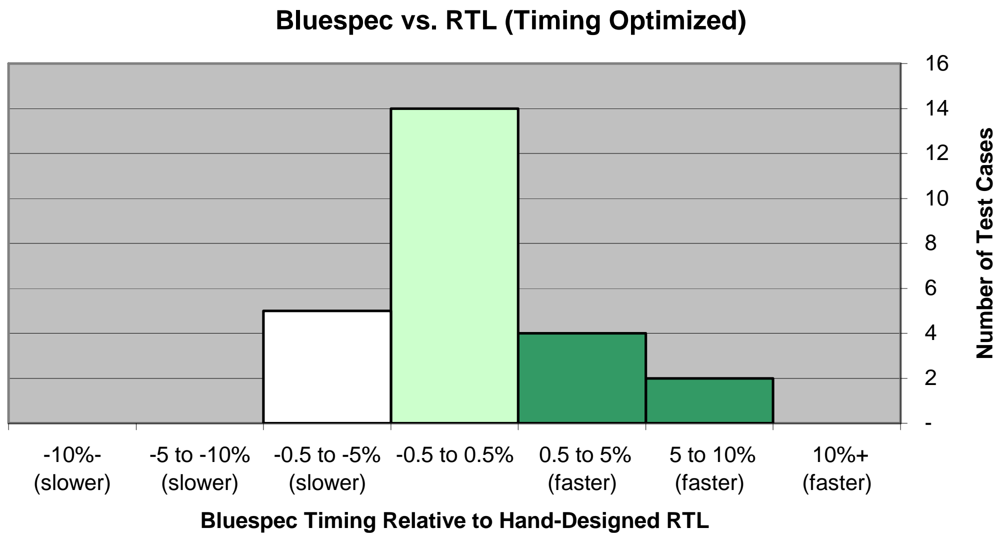


For the second set of tests, the timing constraints were aggressively set, resulting in more aggressive timing results by the synthesizer. The following test profile will be referred to as “Timing Optimized”:

TABLE 3: Timing Optimized

Design Compiler (20th April 2004)								
DC version:						Version 2002.05 for linux -- May 03, 2002		
Total number of designs analyzed:						25		
Number of combinatorial designs:						7		
DC constraints used:						DC compile was run with medium effort, timing constraints are given in table		
BSC Version and build used:						Bluespec Compiler, version 3.8.12 (build 4315, 2004-04-18)		
S. No	**Testcases**	**DC Constraints ClockPeriod/ Max Delay(ns)**	**Total Area (golden)**	**Total Area (bsv)**	**Golden RTL Slack**	**BSV RTL Slack**	**Num. Flops (golden)**	**Num Flops (bsv)**
1	gray_code_converter	2	9	9	0.44	0.44	0	0
2	priorityencoder	2	33	33	0.00	0.00	0	0
3	paritychecker	4	21	21	0.06	0.06	0	0
4	readWriteFsm	4	21	21	0.07	0.07	2	2
5	barrel_shifter	2	34	34	0.35	0.35	0	0
6	speedFSM	5	33	33	0.14	0.25	2	2
7	rippleadder	11	98	98	0.02	0.02	0	0
8	angularfsm	5	71	77	0.03	0.01	3	3
9	one_hot_encoded_fsm	6	101	134	0.24	0.03	7	7
10	pattern_detector	6	68	81	0.02	0.26	3	3
11	wallace_multiplier	10	149	162	0.40	0.09	0	0
12	handshakeprotocol	6	120	119	0.01	0.02	8	8
13	traffic_light_controller	10	232	234	0.07	0.00	11	11
14	rotorscontrol	9	304	299	0.07	0.00	4	4
15	SequentialMultipler	9	341	362	0.07	0.07	23	23
16	shift_add	10	442	479	0.00	0.15	26	26
17	threewayroundrobinarbiter	10	400	385	0.27	0.94	18	18
18	divider	28	406	406	0.35	0.35	0	0
19	cache_coherence	10.5	390	465	0.14	0.00	18	18
20	boothMultiplier	15	1762	1497	0.09	0.08	39	39
21	fibonacciseries	15	1139	947	0.55	1.10	82	82
22	lifo	8	1881	1959	0.01	0.01	141	141
23	fifo1	15	1977	2019	0.02	0.80	146	147
24	factorial	35	1775	1699	0.03	0.05	50	50
25	random_number_generator	80	7733	7848	0.49	0.12	96	96

The results of the Timing Optimized tests are summarized in the following graph, which compares the % difference between the Bluespec compiled RTL and hand-designed RTL (the comparison is between the times through the circuits which is calculated as (CONSTRAINT – SLACK TIME)):



7. Conclusions

- Bluespec's SystemVerilog is strongly typed and easy to learn for user with previous knowledge of Verilog/C.

- The RTL Verilog generated by BSC is often better or equivalent to hand-coded RTL Verilog and is always synthesizable through Synopsys Design Compiler. The analysis of gate level netlists shows that the area and timings are equal or better than hand coded RTL Verilog designs in the vast majority of times.

- The design intent can be abstracted at a higher level in BSV, which means it is closer to system specifications as compared to Verilog, which is at an RTL level. The thinking paradigm is different with BSV. The designer need not think in terms of hardware blocks but, instead, may think in terms of system behavior.

- With BSV design specifications are captured at the system level, some optimizations were observed in the BSC generated RTL Verilog that would be unlikely to be considered if the design were specified at RTL Verilog level directly. For example, BSC generates more synthesis efficient case constructs, by carrying out resource sharing in the generated Verilog and not leaving it to DC to infer.

- Design verification turnaround time is faster through BSC because it generates both the RTL Verilog and C-model for a BSV design. The generated RTL Verilog can be verified with the Verilog test bench directly. Also the designer need not develop a C-model explicitly as the BSV testbench provides the capability to do the C-simulation of the BSV design.

- BSV supports a pre-defined (and, even, user-defined) library of design components like FIFO, RAM, and LOCAL BUS, etc., which can be directly imported into a design. This makes the BSV design concise and reduces the verification time (libraries are already verified). The designs are easy to maintain for enhancements.