Generating High Coverage Tests for SystemC Designs Using Symbolic Execution

Bin Lin
Department of Computer Science
Portland State University
Agenda

• Introduction
• Related work and Background
• Our Approach
• Evaluation
• Conclusions and Future Work
SystemC

• A hardware description language (HDL) extending C++

• A set of C++ classes and macros for hardware design

• IEEE Standard 1666™-2011
Major SystemC Structures

System

Module

Process

Process

Signals

Port

Port
SystemC Verification

- Find bugs in SystemC designs
- Improve the quality of SystemC designs
Cost of Bugs Increases 10X/Stage

DAC 2004 Verification Panel, Makoto Ishii, SoC Solution Center, Sony.
Agenda

• Introduction
• Related Work and Background
• Our Approach
• Evaluation
• Conclusions and Future Work
Formal Verification of SystemC Designs

• Model Checking SystemC Designs Using Timed Automata. [Herber et al., 2008]

• Proving Transaction and System-level Properties of Untimed SystemC TLM Designs. [Große et al., 2010]

• KRATOS: A Software Model Checker for SystemC. [Cimatti et al., 2011]

• Symbolic Model Checking on SystemC Designs. [Chou et al., 2012]

Limitations: checking limited properties; property formulation is challenging
Dynamic Validation of SystemC Designs

- Code-coverage Based Test Vector Generation for SystemC Designs. [Junior and Cecilio da Silva, 2007]

- Coverage Metrics for Verification of Concurrent SystemC Designs Using Mutation Testing. [Sen and Abadir, 2010]

- Automatic RTL Test Generation from SystemC TLM Specifications. [Chen et al., 2012]
Symbolic Execution

```c
void test(int b) {
    int c;
    if (b < 0)
        c = -b;
    else if (b % 2)
        c = b / 2;
    else
        c = 0;
}
```
Symbolic Execution Engine: KLEE

• Symbolic execution engine
• Built upon the LLVM infrastructure
• Targets on sequential C programs
Agenda

• Introduction
• Related work and Background
• Our Approach
• Evaluation
• Conclusions and Future Work
Our Approach

• Automatic tests generation for SystemC
  – Targets high-level synthesizable subset of SystemC
  – Generates high coverage tests
  – Utilizes symbolic execution
Workflow of Our Approach

1. SystemC Design
2. Test-Harness Generation
3. Test Harness
4. LLVM Bitcode
5. Symbolic Execution
6. Symbolic Expressions
7. Test-Case Generation
8. Test Cases
9. Test-Case Replay
10. Code Coverage
Test-Harness Generation

- Registers SystemC processes
- Initializes shared signals
- Provides synchronization mechanisms
- Constructs symbolic variables
- Handles SystemC library calls

Test harness

SystemC design

Environment inputs

SystemC library calls

Environment outputs
Handling SystemC Concurrency

• SystemC concurrency

```c
sc_signal<int> a;
void T1(){
    wait();
    while(true){
        a = 1;
        wait();
    }
}
void T2() {
    int b;
    wait();
    while(true){
        b = a;
        wait();
    }
}
```

– Simulate by 2 clock cycles

– Execution sequence
  • (T1; T2; T1; T2)
    a: 1, b: 0
  • (T1; T2; T2; T1)
    a: 1, b: 0
  • (T2; T1; T1; T2)
    a: 1, b: 0
  • (T2; T1; T2; T1)
    a: 1, b: 0
Handling SystemC Concurrency (Cont.)

• Scheduler

Runnable queue Q1:

Next_runnable queue Q2:

State:
Handling SystemC Concurrency (Cont.)

• Scheduler

Runnable queue Q1: P2

Next Runnable queue Q2: 

Active process: P1

Diagram:

- S → SP1 → Copy
- SP1 → Executes P1
- SP1 → S'P1
Handling SystemC Concurrency (Cont.)

• Scheduler

Runnable queue Q1:

Next Runnable queue Q2:

Active process: P2
• Scheduler

Runnable queue Q1:

Next_runnable queue Q2:

Active process:
## Technical Challenges and Solutions

<table>
<thead>
<tr>
<th>Challenges</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrency</td>
<td>Scheduler</td>
</tr>
<tr>
<td>Path explosion</td>
<td>Time bound and clock cycle bound</td>
</tr>
<tr>
<td>Hardware data structures</td>
<td>Case by case modeling</td>
</tr>
</tbody>
</table>
Test-Case Generation

• Path constraints
  – \((\text{en}_2 \neq 0) \land (\text{in}_2 < 0) \land (\text{en}_3 \neq 0)\)

• Symbolic expressions
  – \([((\text{Eq} \ false \ (\text{Eq} \ 0 \ (\text{ReadLSB} \ w32 \ 0 \ \text{en}_2))))
    \ (\text{Slt} \ (\text{ReadLSB} \ w32 \ 0 \ \text{in}_2) \ 0)
    \ (\text{Eq} \ false \ (\text{Eq} \ 0 \ (\text{ReadLSB} \ w32 \ 0 \ \text{en}_3))))]\)

• Concrete test case
  – \(\text{en}_2 = 0, \ \text{in}_2 = -1, \ \text{en}_3 = 1\)
Test-Case Replay

Replay harness

Test cases       Stimuli       SystemC design
Agenda

• Introduction
• Related work and Background
• Our Approach
• Evaluation
• Conclusions and Future Work
## Code Coverage Results

<table>
<thead>
<tr>
<th>Designs</th>
<th>LoC</th>
<th>Line Coverage (%)</th>
<th>Branch Coverage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>usbArbStateUpdate</td>
<td>85</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>mips</td>
<td>255</td>
<td>100</td>
<td>97.9</td>
</tr>
<tr>
<td>adpcm</td>
<td>134</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>idct</td>
<td>244</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Sync_mux81</td>
<td>52</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>risc_cpu_exec</td>
<td>126</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>risc_cpu_mmxu</td>
<td>187</td>
<td>99.4</td>
<td>97.9</td>
</tr>
<tr>
<td>risc_cpu_control</td>
<td>826</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>risc_cpu_bdp</td>
<td>148</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>risc_cpu_crf</td>
<td>927</td>
<td>98.2</td>
<td>95.7</td>
</tr>
<tr>
<td>risc_cpu</td>
<td>2056</td>
<td>96.3</td>
<td>93.2</td>
</tr>
</tbody>
</table>
## Time and Memory Usage

<table>
<thead>
<tr>
<th>Designs</th>
<th>Time (seconds)</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>usbArbStateUpdate</td>
<td>0.05</td>
<td>13.7</td>
</tr>
<tr>
<td>mips</td>
<td>178.23</td>
<td>27.6</td>
</tr>
<tr>
<td>adpcm</td>
<td>1.88</td>
<td>16.2</td>
</tr>
<tr>
<td>idct</td>
<td>180.00</td>
<td>134.0</td>
</tr>
<tr>
<td>Sync_mux81</td>
<td>0.04</td>
<td>13.5</td>
</tr>
<tr>
<td>risc_cpu_exec</td>
<td>3.23</td>
<td>46.9</td>
</tr>
<tr>
<td>risc_cpu_mmxu</td>
<td>11.38</td>
<td>15.6</td>
</tr>
<tr>
<td>risc_cpu_control</td>
<td>0.57</td>
<td>17.8</td>
</tr>
<tr>
<td>risc_cpu_bdp</td>
<td>0.15</td>
<td>17.5</td>
</tr>
<tr>
<td>risc_cpu_crf</td>
<td>300.00</td>
<td>61.1</td>
</tr>
<tr>
<td>risc_cpu</td>
<td>169</td>
<td>264</td>
</tr>
</tbody>
</table>
Comparison with Random Testing

Line Coverage
Comparison with Random Testing

Branch Coverage
Agenda

• Introduction
• Related work and Background
• Our Approach
• Evaluation
• Conclusions and Future Work
Conclusions

• Automatically generates test cases
• Provides high code coverage
• Uses modest time and memory
• Scales to designs of practical sizes
Future Work

• Support more SystemC structures
• Develop algorithms to detect data race
• Enlarge the set of SystemC designs
Thank you!
**SystemC Example**

```c
SC_MODULE(example) {
    sc_in<bool> en;
    sc_in<bool> clk;
    sc_in<int> in;
    sc_out<int> out;
    sc_signal<int> b;

    void P1() {
        wait();
        while(true) {
            if(en.read())
                b = in.read();
            wait();
        }
    }

    void P2() {
        int c;
        wait();
        while(true) {
            if(b < 0)
                c = -b;
            else if(b % 2)
                c = b / 2;
            out.write(c);
            wait();
        }
    }

    SC_CTOR(example) {
        SC_CTHREAD(P1, clk.pos());
        SC_CTHREAD(P2, clk.pos());
    }
};
```
typedef struct Globals{
  int b;
}globalVars;

globalVars currState, LStates[2];

......

void PREPROCESS(currState) {
  ...... }

void SYNC(LStates) {
  ...... }

int main(int argc, char **argv) {
  ......
  SESC_make_symbolic(&en, sizeof(en), “en”);
  SESC_make_symbolic(&in, sizeof(in), “in”);
  SESC_thread(“P1”, &en, &clk, &din, &LStates[1].b, &dout);
  SESC_thread(“P2”, &en, &clk, &din, &LStates[2].b, &dout);
  ......
  SESC_start(numCycles);

  return 0;
}
1 \textbf{void} P1(*en, *clk, *din, *b, *dout) {
2     \ldots.
3 }  

4

5 \textbf{void} P2(*en, *clk, *din, *b, *dout) {
6     \ldots.
7 }
Test-Case Replay Harness

```c
int replayHarness() {
    bool en;
    int in;

    wait();
    while(true) {
        SESC_de_symbolic(&en, sizeof(en), "en");
        SESC_de_symbolic(&in, sizeof(in), "in");
        en_out.write(en);
        in_out.write(in);
        wait();
    }
}
```