Generating High Coverage Tests for SystemC Designs Using Symbolic Execution

Bin Lin

Department of Computer Science

Portland State University

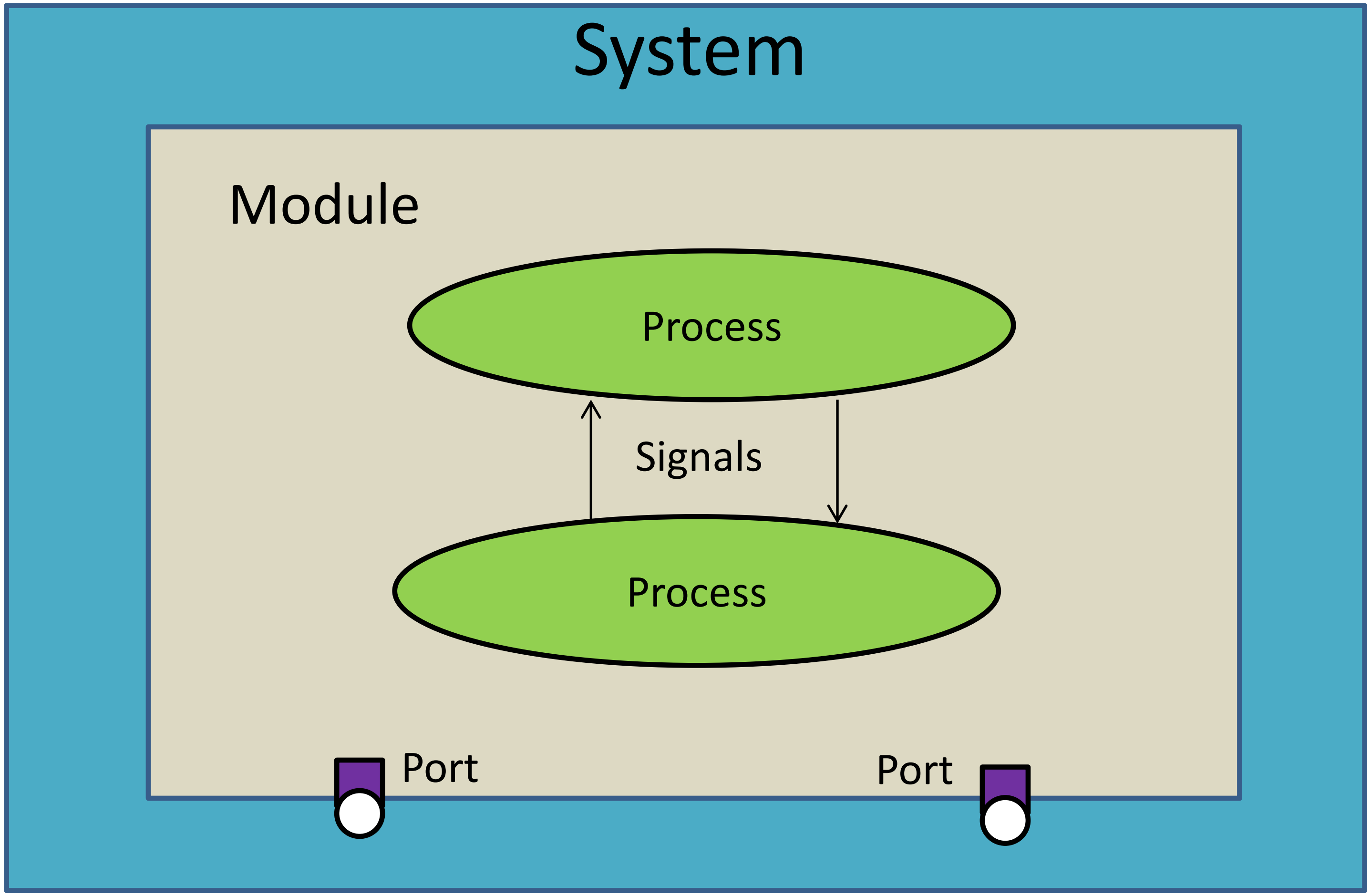
Agenda

- 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

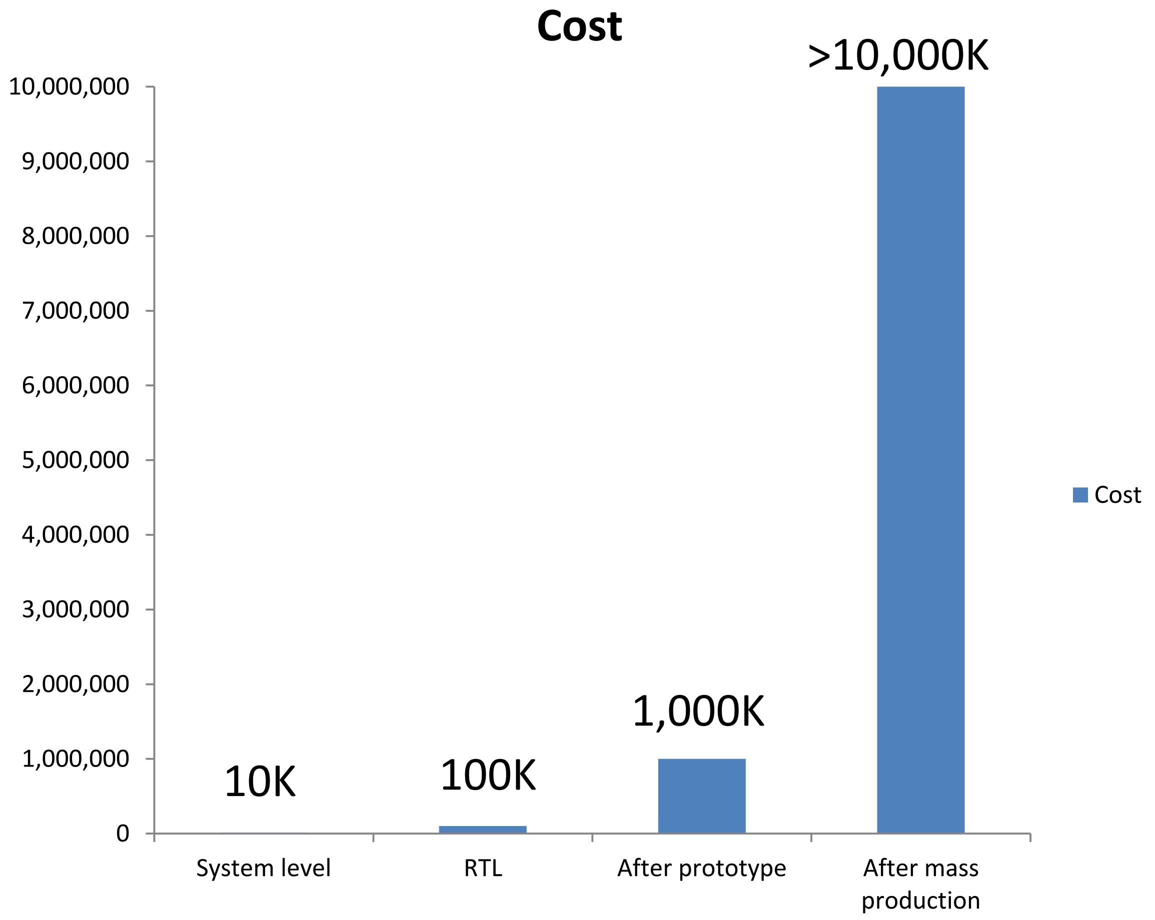


SystemC Verification

- 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]

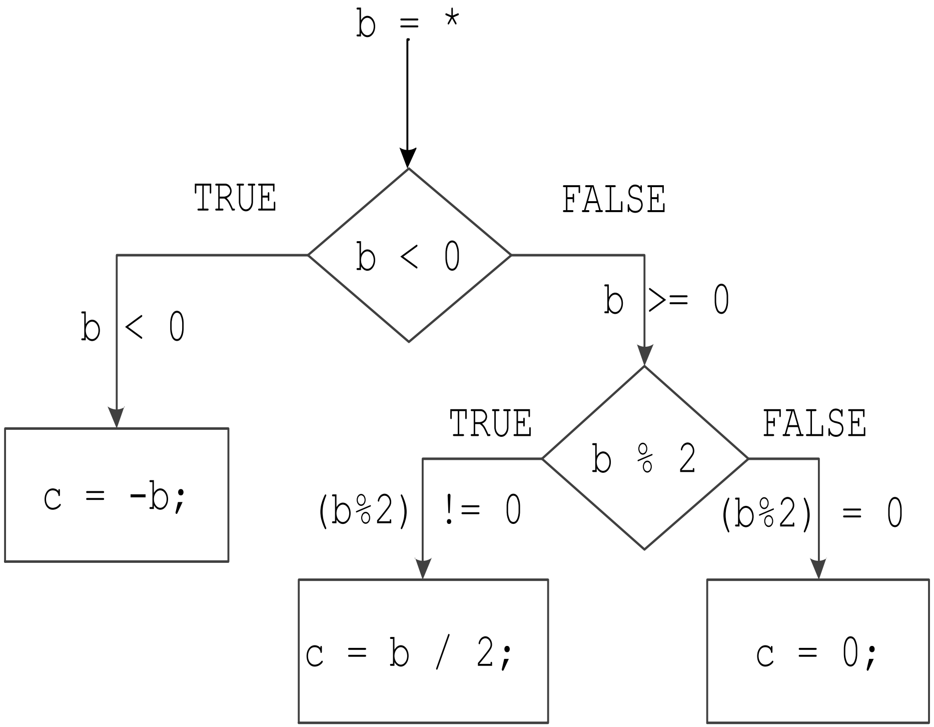
Limitations: checking limited properties; property formulation is challenging

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

```
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

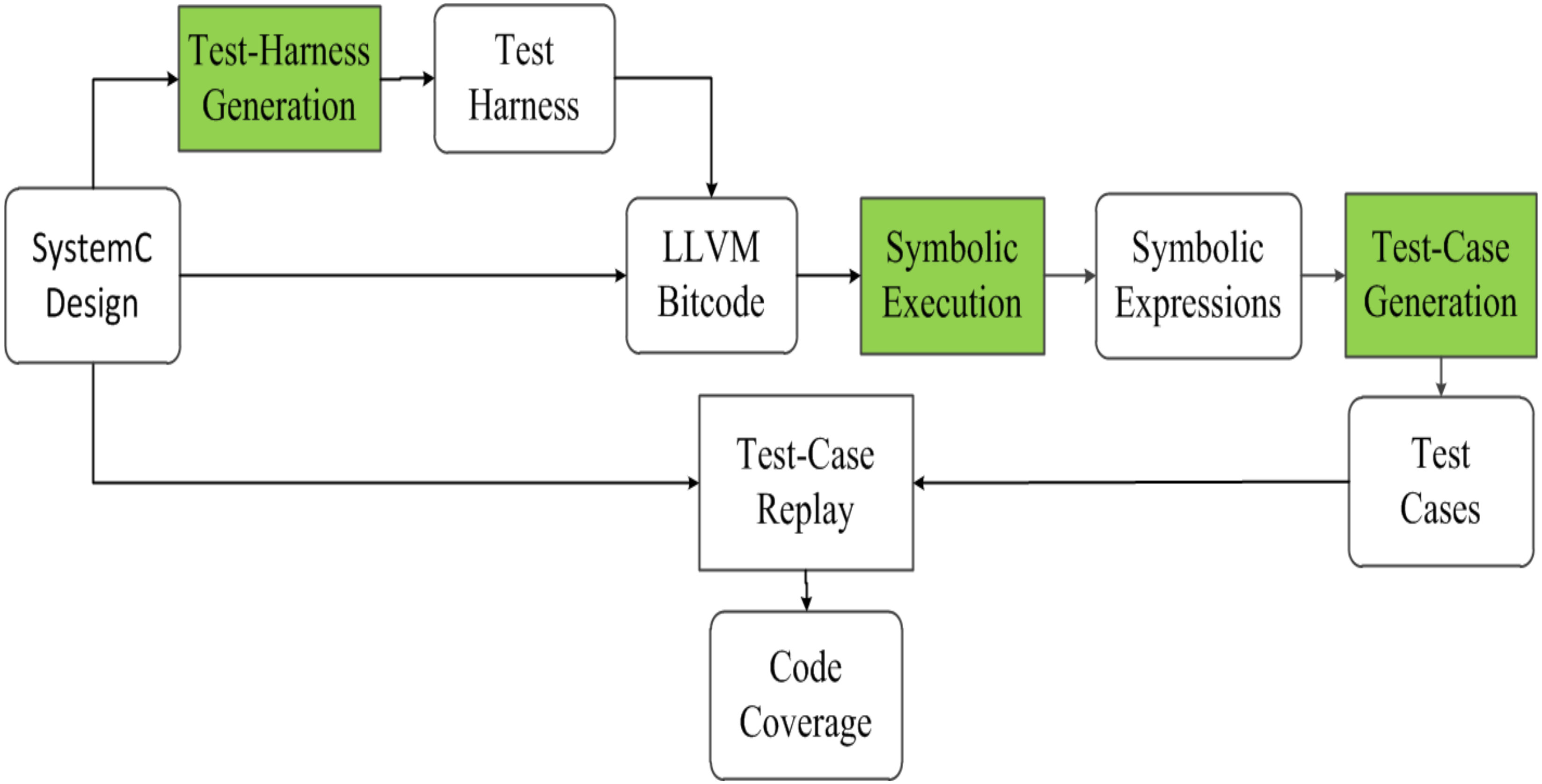
Agenda

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

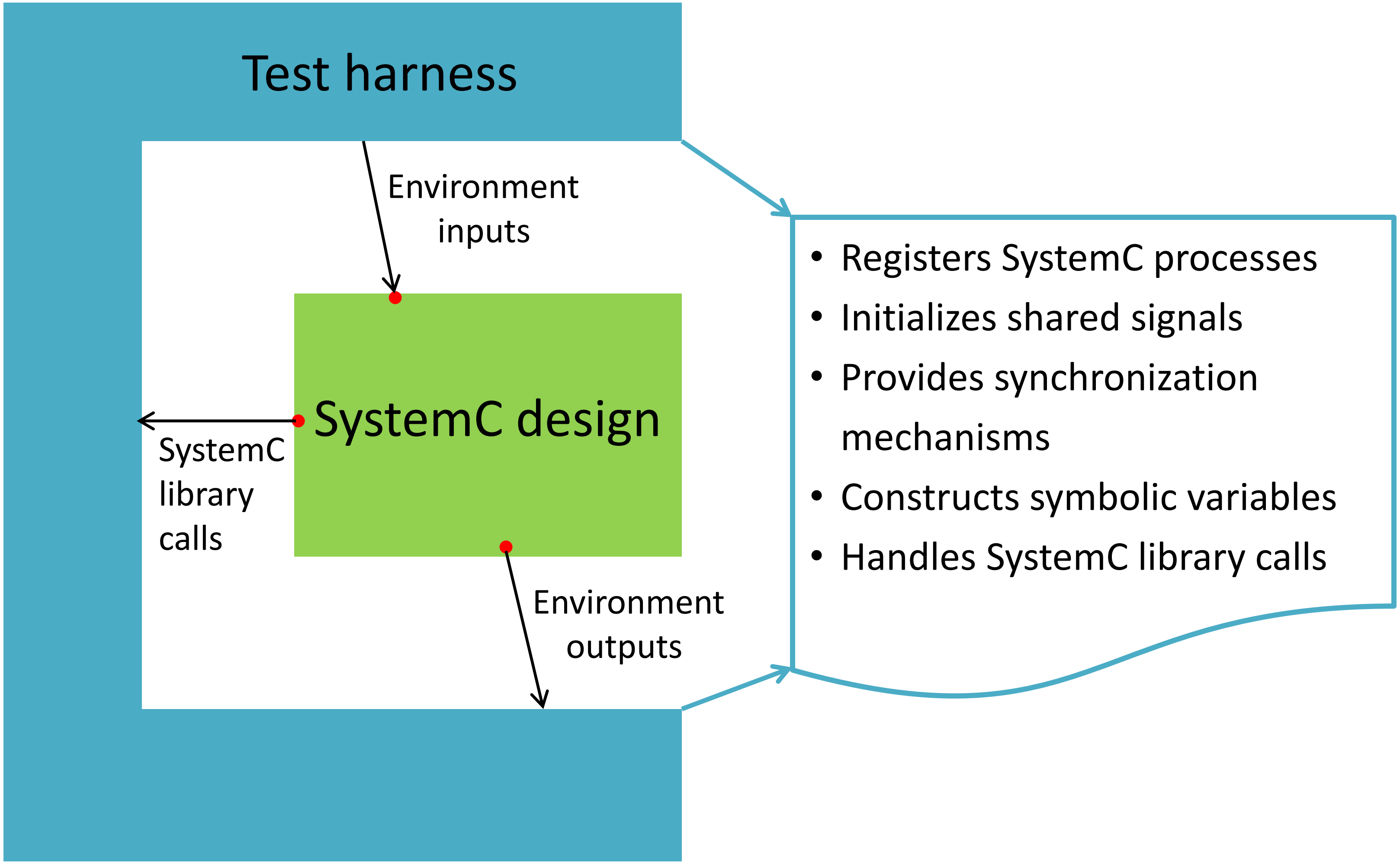
Our Approach

- Automatic tests generation for SystemC
  - Targets high-level synthesizable subset of SystemC
  - Generates high coverage tests
  - Utilizes symbolic execution

Workflow of Our Approach



Test-Harness Generation



Test harness

Environment inputs

SystemC design

SystemC library calls

Environment outputs

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

Handling SystemC Concurrency

- SystemC concurrency

```
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)

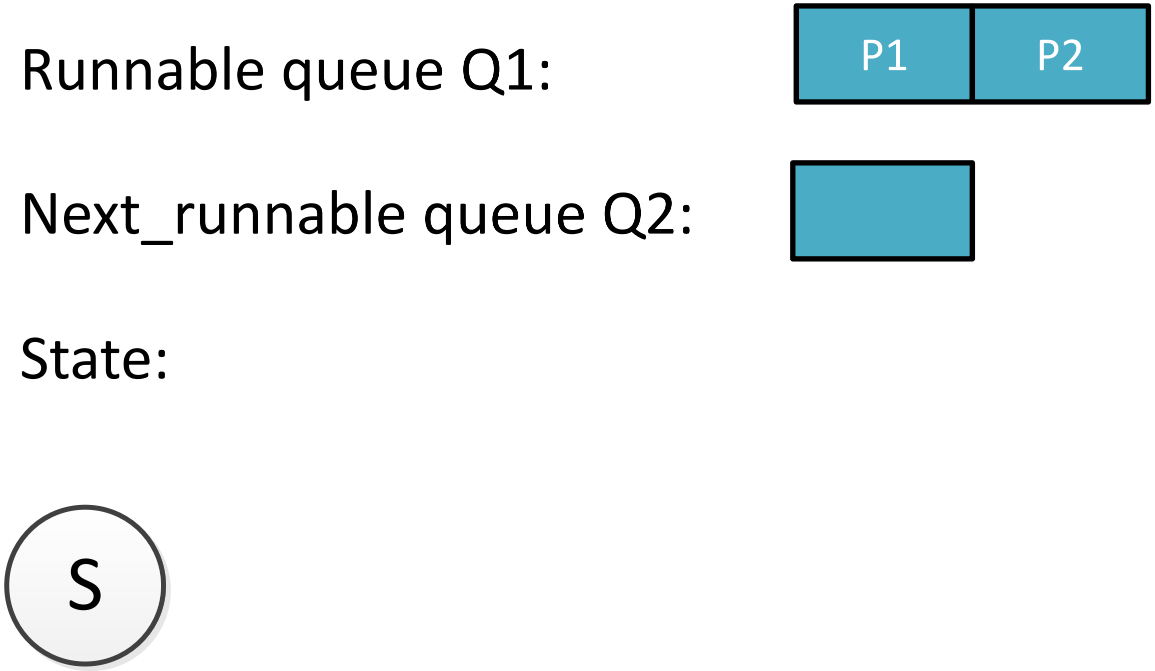
      a: 1, b: 0
    - (T2; T1; T1; T2)

      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:

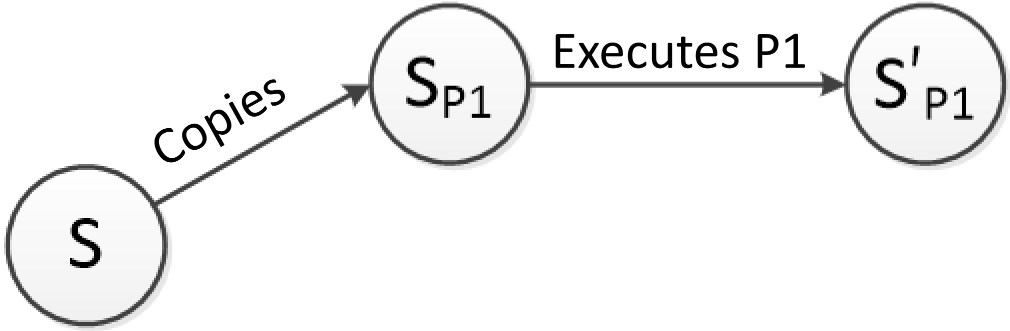
Handling SystemC Concurrency (Cont.)

- Scheduler

Runnable queue Q1: P2

Next_runnable queue Q2:

Active process: **P1**



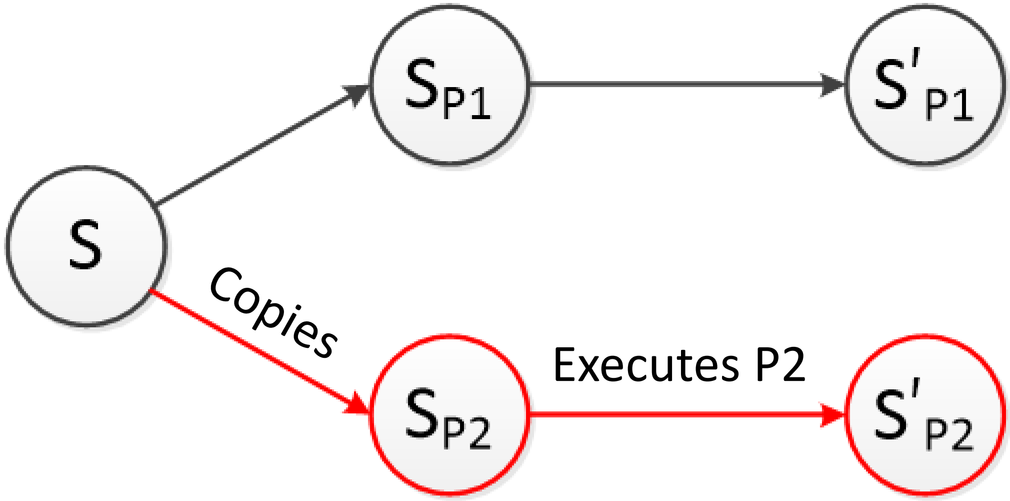
Handling SystemC Concurrency (Cont.)

- Scheduler

Runnable queue Q1:

Next_runnable queue Q2: P1

Active process: **P2**



Handling SystemC Concurrency (Cont.)

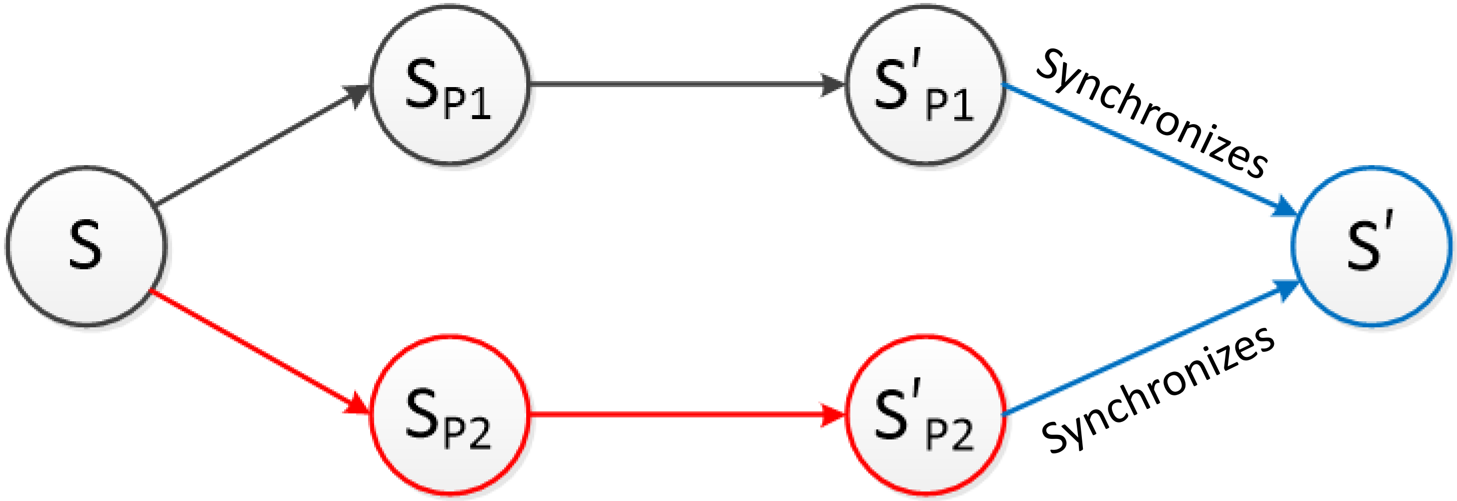
- Scheduler

Runnable queue Q1:

Next_runnable queue Q2:

P1	P2

Active process:



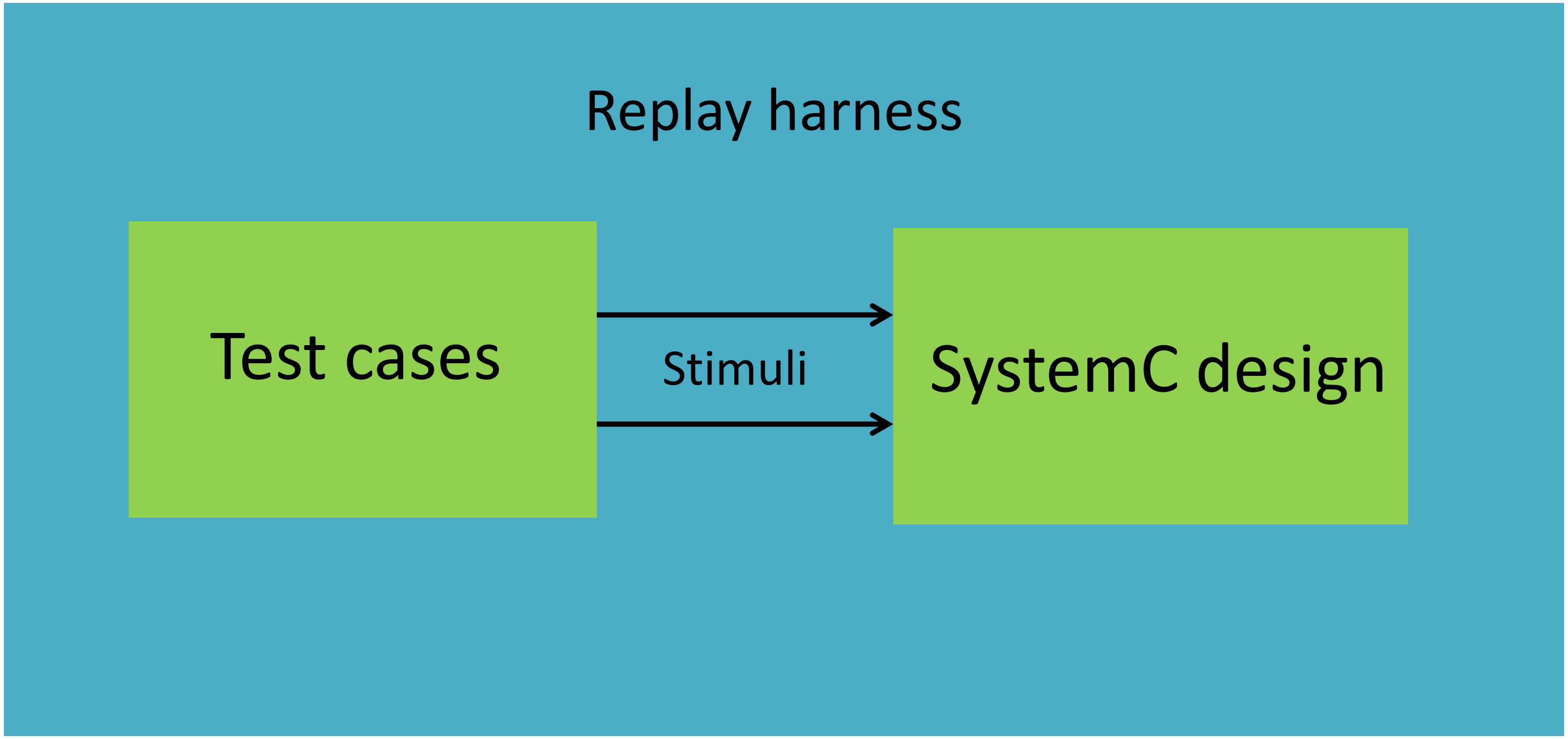
Technical Challenges and Solutions

Challenges	**Solutions**
Concurrency	Scheduler
Path explosion	Time bound and clock cycle bound
Hardware data structures	Case by case modeling

Test-Case Generation

- Path constraints
  - $(en_2 \neq 0)$ ^ $(in_2 < 0)$ ^ $(en_3 \neq 0)$
- Symbolic expressions
  - [(Eq false (Eq 0 (ReadLSB w32 0 en_2)))
    (Slt (ReadLSB w32 0 in_2) 0)
    (Eq false (Eq 0 (ReadLSB w32 0 en_3)))]
- Concrete test case
  - $en_2 = 0$, $in_2 = -1$, $en_3 = 1$

Test-Case Replay



Replay harness

Test cases

Stimuli

SystemC design

Agenda

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

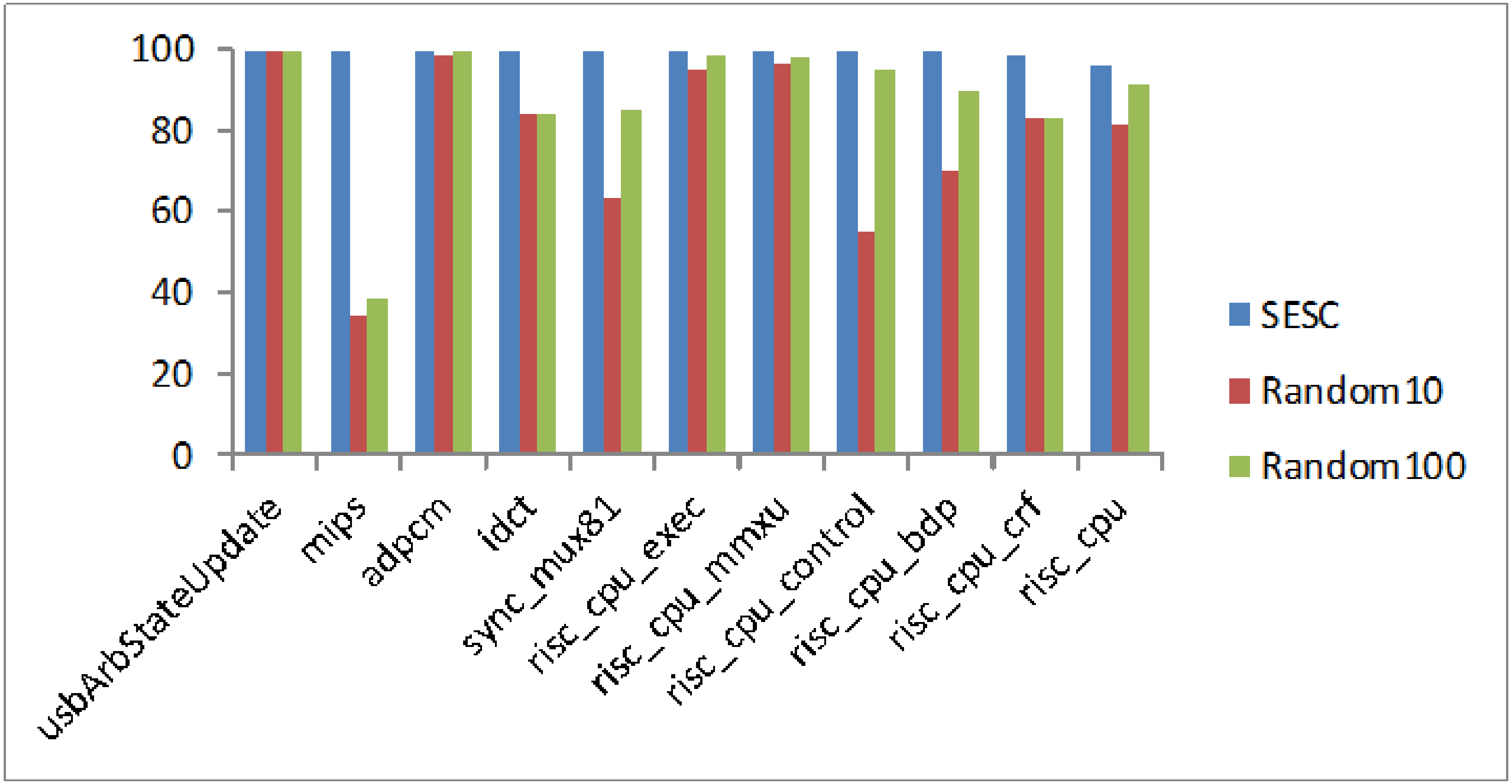
Code Coverage Results

Designs	LoC	Line Coverage (%)	Branch Coverage (%)
usbArbStateUpdate	85	100	100
mips	255	100	97.9
adpcm	134	100	100
idct	244	100	100
Sync_mux81	52	100	100
risc_cpu_exec	126	100	100
risc_cpu_mmxu	187	99.4	97.9
risc_cpu_control	826	100	100
risc_cpu_bdp	148	100	100
risc_cpu_crf	927	98.2	95.7
risc_cpu	2056	96.3	93.2

Time and Memory Usage

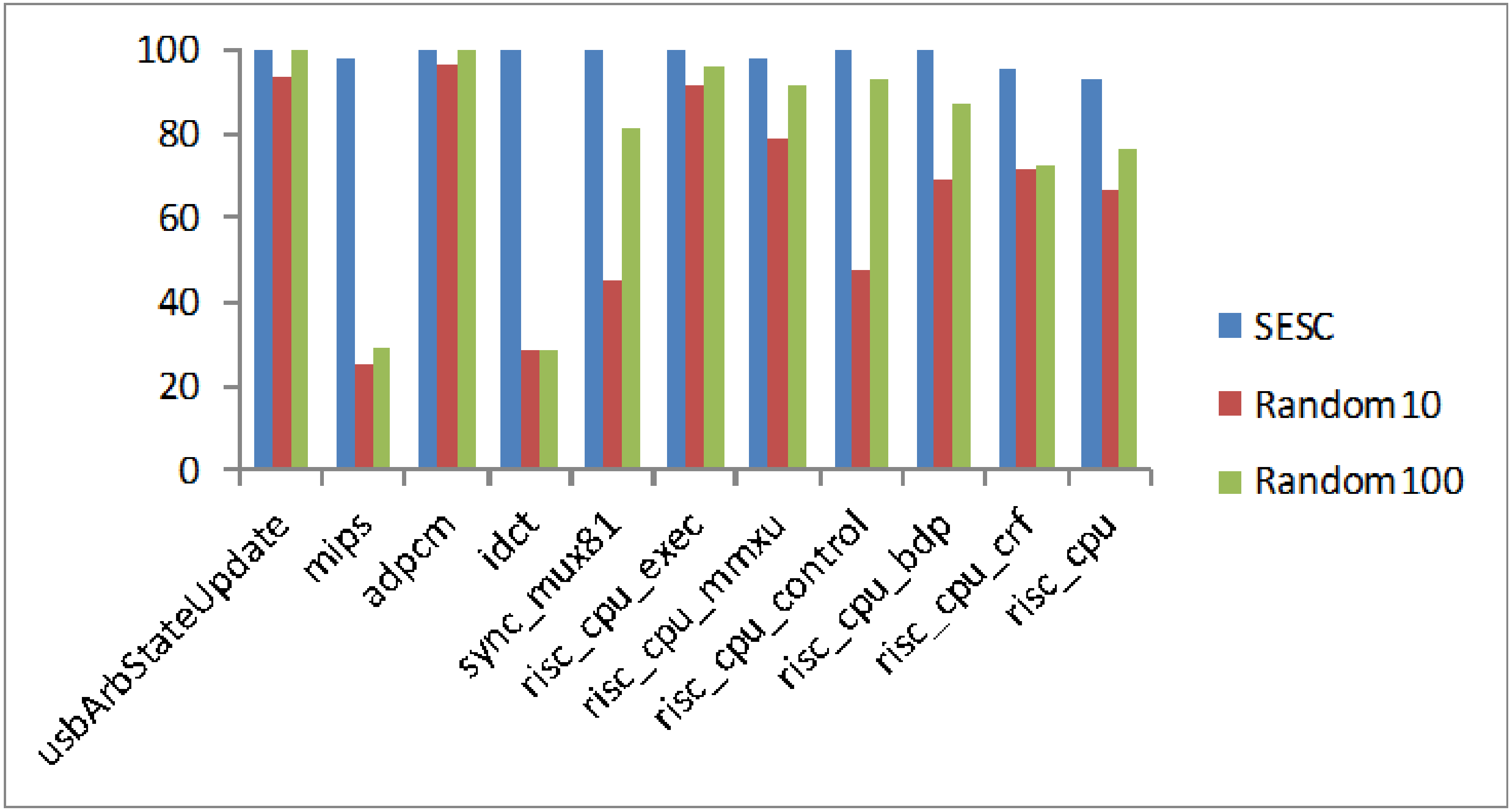
Designs	Time (seconds)	Memory (MB)
usbArbStateUpdate	0.05	13.7
mips	178.23	27.6
adpcm	1.88	16.2
idct	180.00	134.0
Sync_mux81	0.04	13.5
risc_cpu_exec	3.23	46.9
risc_cpu_mmxu	11.38	15.6
risc_cpu_control	0.57	17.8
risc_cpu_bdp	0.15	17.5
risc_cpu_crf	300.00	61.1
risc_cpu	169	264

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

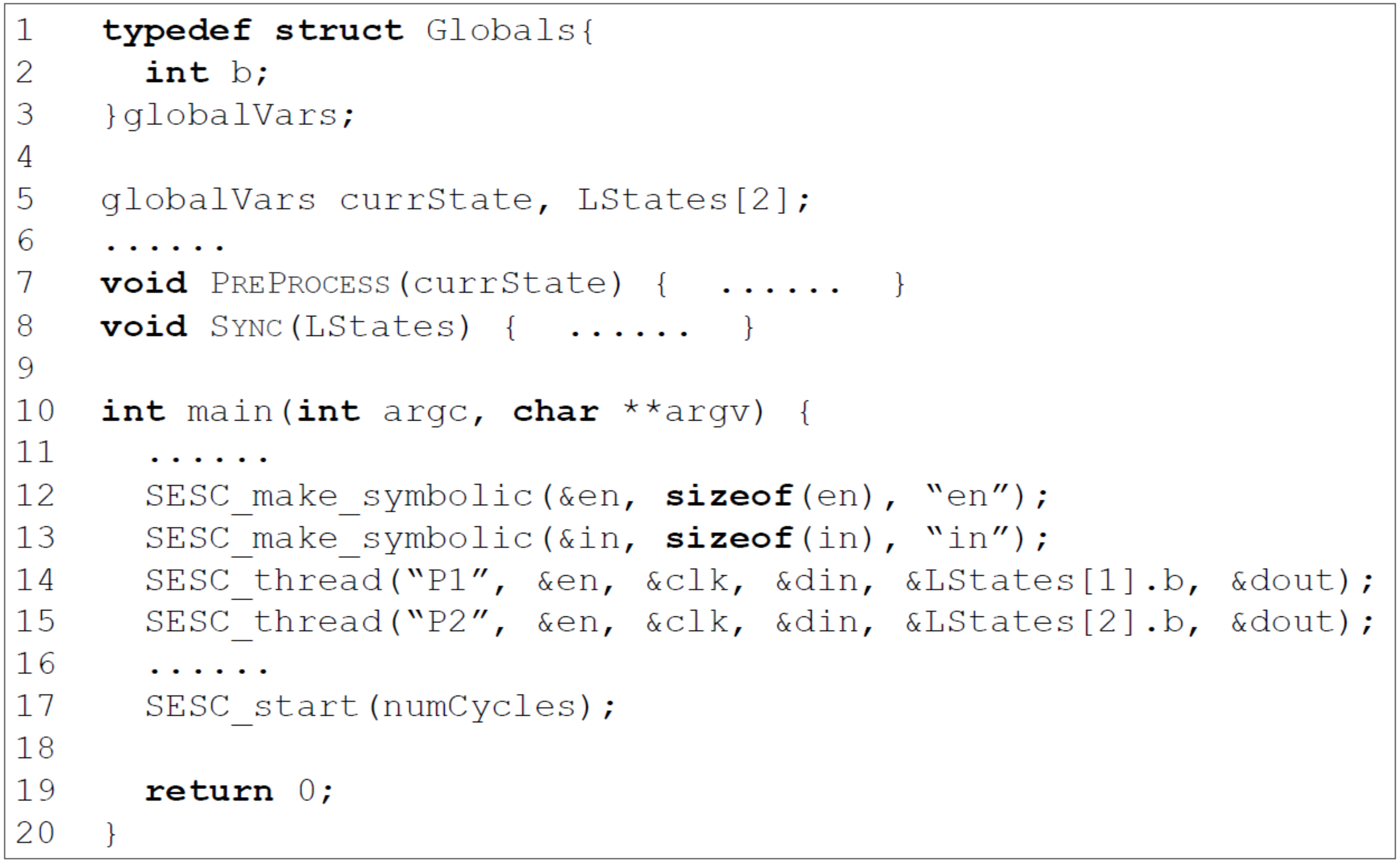
- Support more SystemC structures
- Develop algorithms to detect data race
- Enlarge the set of SystemC designs

Thank you!

SystemC Example

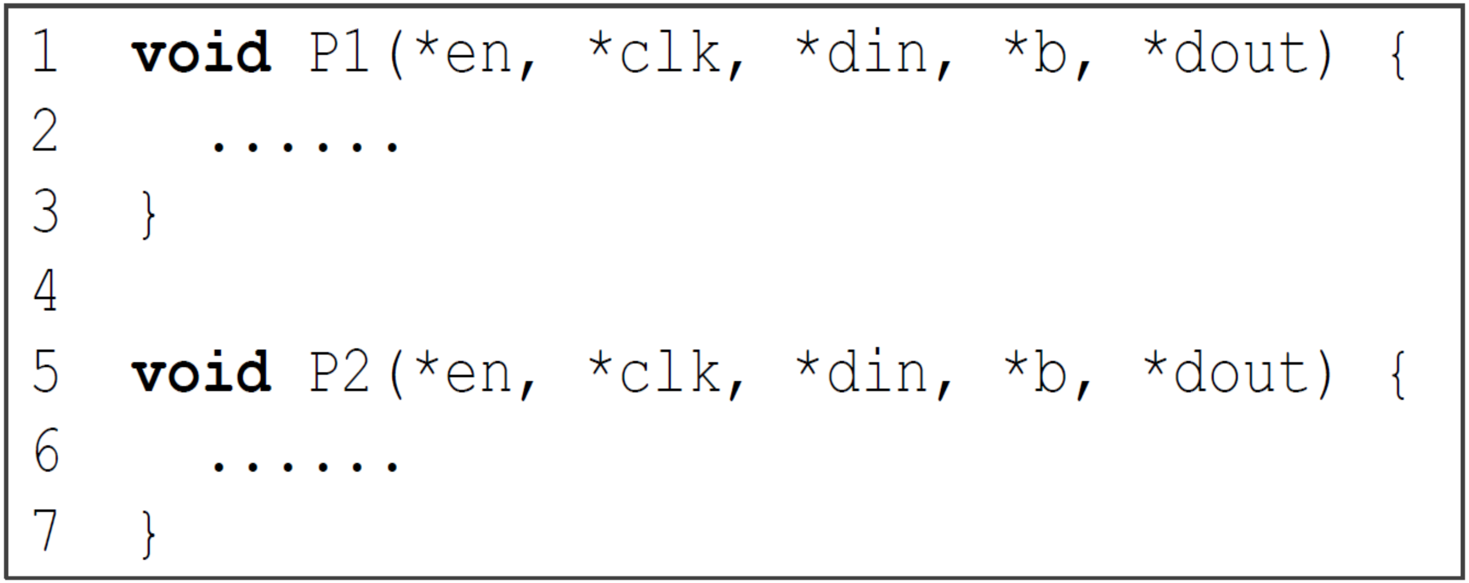
```
SC_MODULE(example){
  sc_in<bool>   en;
  sc_in<bool>  clk;
  sc_in<int>    in;
  sc_out<int>  out;

  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());
  }
};
```

Test Harness



```
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;
}
```

Flattened Result



```
void P1(*en, *clk, *din, *b, *dout) {
  ......
}

void P2(*en, *clk, *din, *b, *dout) {
  ......
}
```

Test-Case Replay Harness

```
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();
  }
}
```