A Theoretical and Experimental Review of SystemC Front-ends

Kevin Marquet Matthieu Moy Bageshri Karkare

Verimag (Grenoble INP)
Grenoble
France

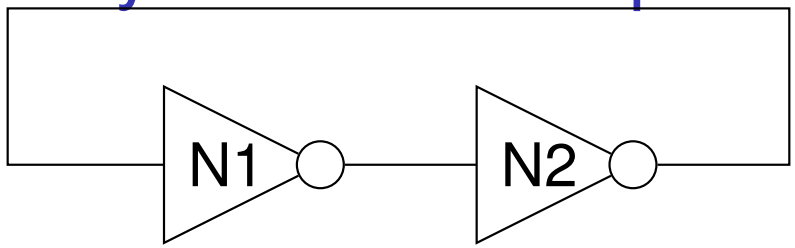
FDL, September 15th 2010

Summary

SystemC

- Industry-standard for high-level modeling (TLM, ...) of Systems-on-a-Chip,
- Library for C++ (compile with `g++ -lsystemc...`)

SystemC: Example




```
SC_MODULE(not_gate) {
    sc_in<bool> in;
    sc_out<bool> out;

    void compute (void) {
        // Behavior
        bool val = in.read();
        out.write(!val);
    }

    SC_CTOR(not_gate) {
        SC_METHOD(compute);
        sensitive << in;
    }
};
```

```
int sc_main(int argc, char **argv) {
    // Elaboration phase (Architecture)
    not_gate n1("N1");
    not_gate n2("N2");
    sc_signal<bool> s1, s2;

    // Binding
    n1.out.bind(s1);
    n2.out.bind(s2);
    n1.in.bind(s2);
    n2.in.bind(s1);

    // Start simulation
    sc_start(100, SC_NS);
    return 0;
}
```

Summary

This section

2 SystemC Front-Ends
- Applications
- Difficulties
- Approaches

When you *don't* need a front-end

- Main application of SystemC: Simulation
  - Just need a C++ compiler + the library
- Testing, run-time verification, monitoring...
  - (Small) modifications of the SystemC library
- IDE integration, Debugging ...
  - Plain C++ front-ends can do most of the job.

When you *don't* need a front-end

- Main application of SystemC: Simulation
  - Just need a C++ compiler + the library
- Testing, run-time verification, monitoring...
  - (Small) modifications of the SystemC library
- IDE integration, Debugging ...
  - Plain C++ front-ends can do most of the job.

No reference front-end available on
`http://www.systemc.org/`

When you *really* need a front-end

- Symbolic formal verification, High-level synthesis
- Visualization
- Introspection

- SystemC-specific Compiler Optimizations
- Advanced debugging features (architecture $\rightarrow$ source code, . . .)

When you *really* need a front-end

- Symbolic formal verification, High-level synthesis
  - Need to extract almost everything about the platform
- Visualization
- Introspection
- SystemC-specific Compiler Optimizations
- Advanced debugging features (architecture $\rightarrow$ source code, . . .)

When you *really* need a front-end

- Symbolic formal verification, High-level synthesis
  - Need to extract almost everything about the platform
- Visualization
  - Need to extract the architecture. Behavior is less important
- Introspection

- SystemC-specific Compiler Optimizations

- Advanced debugging features (architecture $\rightarrow$ source code, . . .)

When you *really* need a front-end

- Symbolic formal verification, High-level synthesis
  - Need to extract almost everything about the platform
- Visualization
  - Need to extract the architecture. Behavior is less important
- Introspection
  - Information about the module (Architecture) + possibly local variables (Behavior)
- SystemC-specific Compiler Optimizations

- Advanced debugging features (architecture $\rightarrow$ source code, . . .)

When you *really* need a front-end

- Symbolic formal verification, High-level synthesis
  - Need to extract almost everything about the platform
- Visualization
  - Need to extract the architecture. Behavior is less important
- Introspection
  - Information about the module (Architecture) + possibly local variables (Behavior)
- SystemC-specific Compiler Optimizations
  - Can use architecture information to optimize the behavior
- Advanced debugging features (architecture $\rightarrow$ source code, . . .)

When you *really* need a front-end

- Symbolic formal verification, High-level synthesis
  - Need to extract almost everything about the platform
- Visualization
  - Need to extract the architecture. Behavior is less important
- Introspection
  - Information about the module (Architecture) + possibly local variables (Behavior)
- SystemC-specific Compiler Optimizations
  - Can use architecture information to optimize the behavior
- Advanced debugging features (architecture $\rightarrow$ source code, . . .)
  - Needs the architecture and behavior

This section

2 SystemC Front-Ends
- Applications
- Difficulties
- Approaches

Difficulties Writing SystemC Front-Ends

1. C++ is complex (e.g. `clang` ≈ 200,000 LOC)
2. Architecture is built at runtime, with C++ code

```
SC_MODULE(not_gate) {
    sc_in<bool> in;
    sc_out<bool> out;

    void compute (void) {
        // Behavior
        bool val = in.read();
        out.write(!val);
    }

    SC_CTOR(not_gate) {
        SC_METHOD(compute);
        sensitive << in;
    }
};
```

```
int sc_main(int argc, char **argv) {
    // Elaboration phase (Architecture)
    not_gate n1("N1");
    not_gate n2("N2");
    sc_signal<bool> s1, s2;

    // Binding
    n1.out.bind(s1);
    n2.out.bind(s2);
    n1.in.bind(s2);
    n2.in.bind(s1);

    // Start simulation
    sc_start(100, SC_NS);
    return 0;
}
```

This section

2 SystemC Front-Ends
- Applications
- Difficulties
- Approaches

Dealing with the complexity of C++

- Write a new C++ front-end (lex+yac, ...)
  - Either huge effort, or many limitations
- Reuse one
  - EDG: Good, expansive C++ front-end
  - GNU g++: Good C++ support, but hard to use as a front-end
  - clang (part of LLVM): Good C++ support (recent), modular
  - ...

Dealing with the architecture

- Static approach: Analyze the elaboration, and find out the architecture
  - Usually very limited wrt. complexity of the elaboration code
- Dynamic approach: Execute the elaboration, and see the result
  - Few limitations
  - Main difficulty: link Behavior $\leftrightarrow$ Architecture.

```
int sc_main(int argc, char **argv) {
    // Elaboration phase (Architecture)
    not_gate n1("N1");
    not_gate n2("N2");
    sc_signal<bool> s1, s2;

    // Binding
    n1.out.bind(s1);
    n2.out.bind(s2);
    n1.in.bind(s2);
    n2.in.bind(s1);

    // Start simulation
    sc_start(100, SC_NS);
    return 0;
}
```

Dealing with the architecture

When it becomes tricky...

```
int sc_main(int argc, char **argv) {
    int n = atoi(argv[1]);
    int m = atoi(argv[2]);
    Node array[n][m];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            array[i][j]
                = new Node(...);
            ...
        }
    }

    sc_start(100, SC_NS);
    return 0;
}
```

Dealing with the architecture

When it becomes tricky...

- Static approach: cannot deal with such code
- Dynamic approach: can extract the architecture for individual instances of the system

```
int sc_main(int argc, char **argv) {
    int n = atoi(argv[1]);
    int m = atoi(argv[2]);
    Node array[n][m];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            array[i][j]
                = new Node(...);
            ...
        }
    }

    sc_start(100, SC_NS);
    return 0;
}
```

Dealing with the architecture

When it becomes *very* tricky...

```
void compute(void) {
    for (int i = 0; i < n; i++) {
        ports[i].write(true);
    }
    ...
}
```

Dealing with the architecture

When it becomes *very* tricky...

- One can unroll the loop to let `i` become constant,
- Undecidable in the general case.

```
void compute(void) {
    for (int i = 0; i < n; i++) {
        ports[i].write(true);
    }
    ...
}
```

Summary

Existing SystemC front-ends

An attempt at classification

	Static	Dynamic
Home-made parser	KaSCPar, sc2v, ParSyC, Scoot, SystemPerl...	
Existing parser	SystemCXML	DATE09[1], Pinapa, PinaVM

- Hard to classify: Quiny (purely dynamic approach)
- Commercial tools (closed, not detailed here): Synopsys, Semantic Design, NC-SystemC (Cadence)

[1] Overcoming limitations of the SystemC data introspection, Christian Genz and Rolf Drechsler

Who should read the paper?

- Targeted reader: People in need of a SystemC front-end to build a research tool
  - To use an existing one
    $\Rightarrow$ emphasis on available/open tools
  - To build a new one
    $\Rightarrow$ description of the challenges and approaches

Who should read the paper?

- Targeted reader: People in need of a SystemC front-end to build a research tool
  - To use an existing one
    $\Rightarrow$ emphasis on available/open tools
  - To build a new one
    $\Rightarrow$ description of the challenges and approaches
- Content:
  - Summary and bibliography for each front-end
  - A small but representative testsuite (public)
  - Experimental results

Who should read the paper?

- Targeted reader: People in need of a SystemC front-end to build a research tool
  - To use an existing one
    $\Rightarrow$ emphasis on available/open tools
  - To build a new one
    $\Rightarrow$ description of the challenges and approaches
- Content:
  - Summary and bibliography for each front-end
  - A small but representative testsuite (public)
  - Experimental results

Disclaimer: paper written by authors of 2 front-ends (Pinapa and PinaVM)

Conclusions of the Review

- Very limited front-ends: KaSCPar, sc2v, SystemPerl, SystemCXML, Quiny
- Not available: ParSyC, DATE09
- Remaining candidates for research tools:
  - Scoot: close-source (but authors opened to discussion), good SystemC/C++ support
  - Pinapa: open-source, very few limitations, but painful to install
  - PinaVM (new, not in the paper): open-source, very few theoretical limitations, still at prototype stage.

Scoot

- Static scheduling based on model-checking and partial order reduction
- In-house C++ front-end, but good in our experience

Pinapa

- Front-end of the tool LusSy: connection of SystemC to various model-checkers
- Based on GCC to support C++
- Executes the elaboration, link the result to the syntax tree

PinaVM

- Connection of SystemC to various model-checkers
- Based on the LLVM compiler infrastructure (uses `llvm-g++` as the front-end)
- Relies on Just-In-Time compilation to link the architecture and the behavior

PinaVM

- Connection of SystemC to various model-checkers
- Based on the LLVM compiler infrastructure (uses `llvm-g++` as the front-end)
- Relies on Just-In-Time compilation to link the architecture and the behavior
- See you at Emsoft for the details ;-)

Summary

Conclusion

- Writing a (good) SystemC front-end = surprisingly difficult task
- Experimental and theoretical comparison of existing SystemC front-ends presented
- Lots of candidates, none 100% satisfactory

Questions?