2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
OCTOBER 15-16, 2024

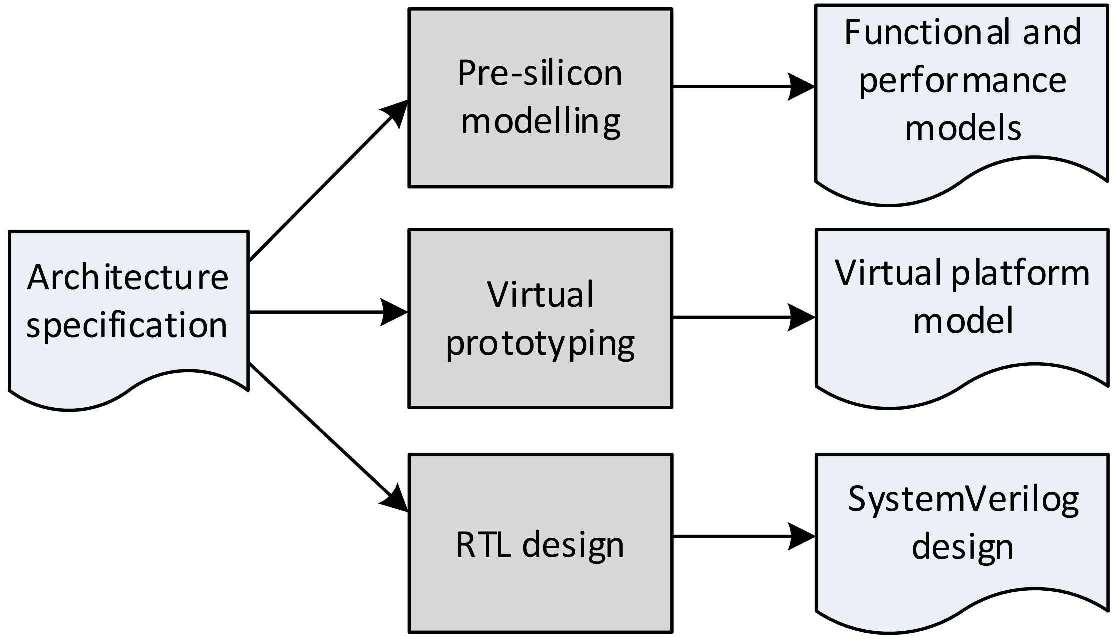
Single Source library for high-level modelling and hardware synthesis

Mikhail Moiseev, Nanda Kalavai

Intel

accellera SYSTEMS INITIATIVE

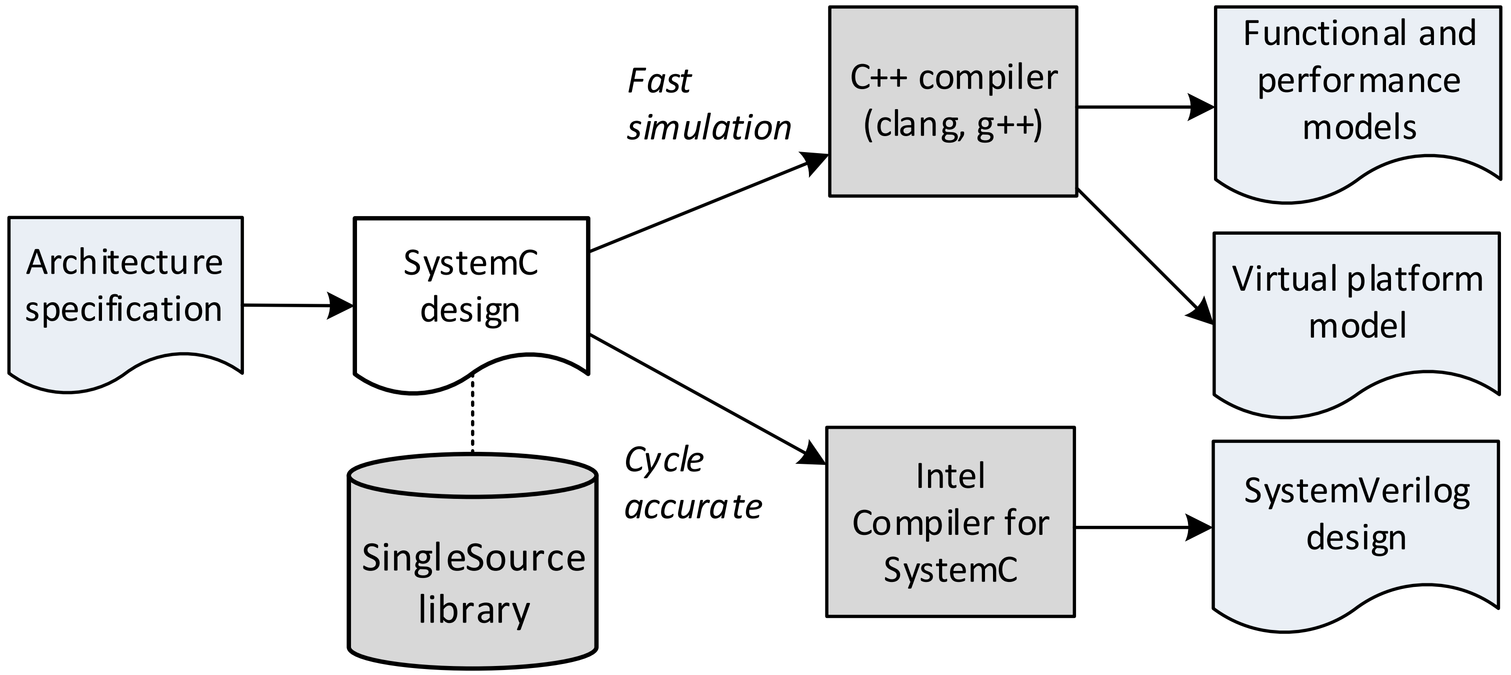
Problem statement



- Multiple independent models require
  - Proving equivalence and sharing updates between the models and RTL
  - Extra resources for multiple models' development and verification

Single Source design flow

- Single Source design flow uses the same SystemC code for architecture exploration, performance evaluation, hardware synthesis and virtual prototyping

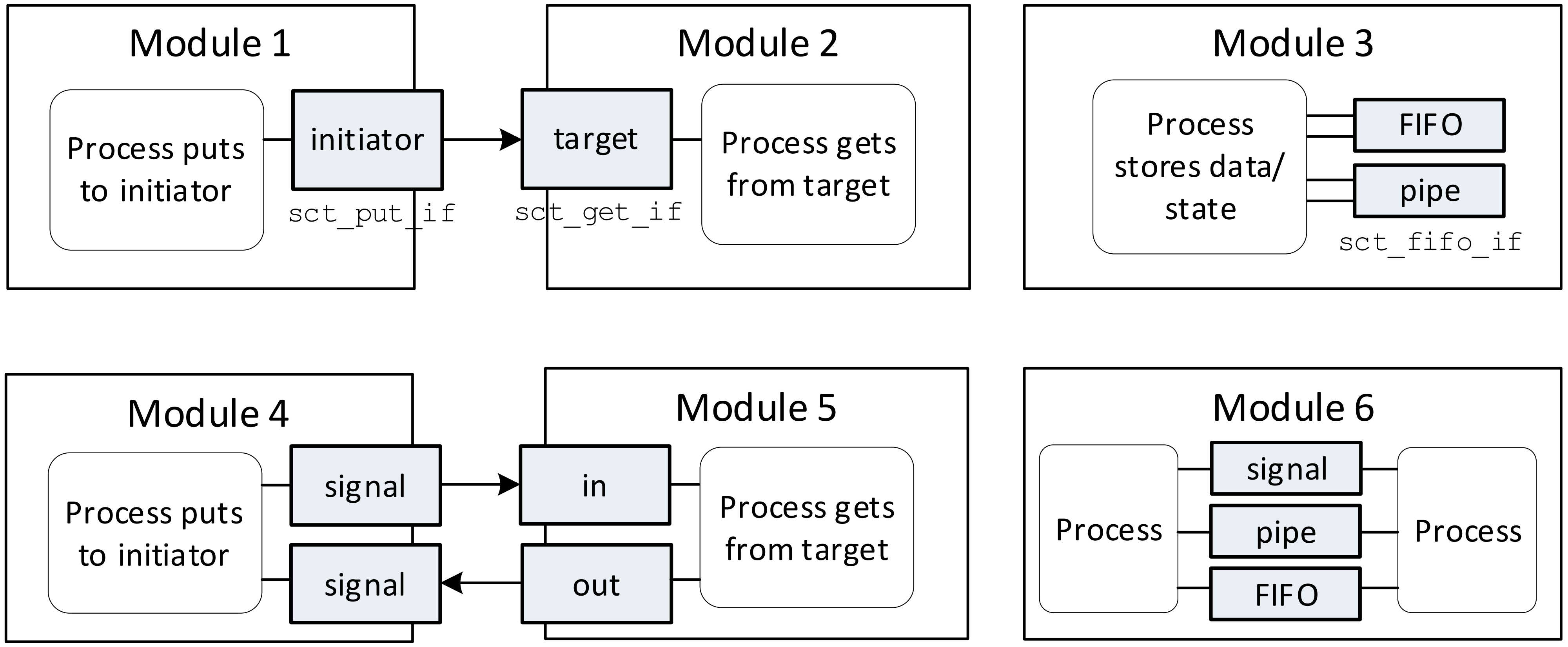


Architecture specification
SystemC design
SingleSource library
Fast simulation
Cycle accurate
C++ compiler (clang, g++)
Intel Compiler for SystemC
Functional and performance models
Virtual platform model
SystemVerilog design

Single Source Library

- Single Source Library of communication channels
  - Functional interfaces to simplify design code and prevent mistakes
  - Support cycle accurate and fast simulation modes
- Cycle accurate mode
  - Cycle accurate clocked design, compatible with SystemC synthesizable standard
  - No extra cost in area, performance and power in synthesis
- Fast simulation mode
  - No clock, request-driven simulation
  - No extra process activation

Single Source channels

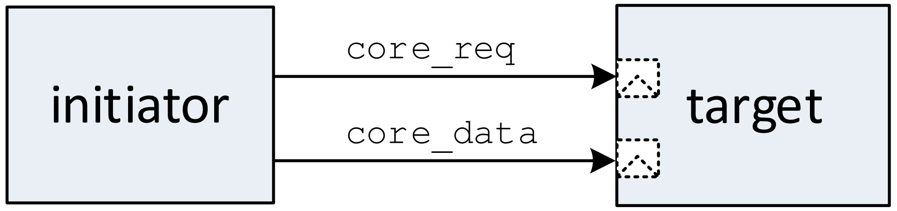


Channel interfaces

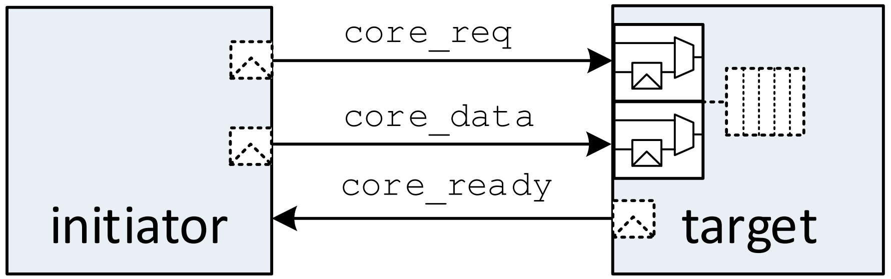
`sct_put_if`	`bool ready()`	Return true if it is ready to put request
	`void reset_put()`	Reset this channel for put
	`bool put(const T& data)`	Non-blocking put request if channel is ready, return ready
	`void b_put(const T& data)`	May-blocking put request, to use in THREAD process only
`sct_get_if`	`bool request()`	Return true if it has request to get
	`void reset_get()`	Reset this channel for get
	`T peek()`	Peek request, return request data
	`T get()`	Non-blocking get and remove request, return request data
	`T b_get()`	May-blocking get request, to use in THREAD process only
`sct_fifo_if`	`inherits sct_put_if<T> and sct_get_if<T>`	
	`unsigned elem_num()`	Return current number of elements

Initiator and target

Combinational connection



Buffered connection



- Combinational connection
  - Target process is always ready to get
  - Optional registers for request
  - Full throughput
- Buffered connection
  - Optional registers for request and ready
  - Optional FIFO
  - Protection from combinational loop
  - Full throughput

Initiator and target example

```
SC_MODULE(Producer) {
   sct_initiator<T>    init{"init"};
   SC_CTOR(Producer) {
      init.clk_nrst(clk, nrst);
      SC_METHOD(initProc); sensitive << init;
}};
SC_MODULE(Consumer) {
   sct_target<T>       targ{"targ"};
   SC_CTOR(Consumer) {
      targ.clk_nrst(clk, nrst);
      SC_THREAD(targProc); sensitive << targ;
      async_reset_signal_is(nrst, false);
}};
SC_MODULE(Top) {
   Producer   prod{"prod"};
   Consumer   cons{"cons"};
   SC_CTOR(Top) {
      prod.init.bind(cons.targ);
}};
```

```
void Producer::initProc() {
    init.reset_put();
    if (init.ready())
        init.put(getSomeValue());
}

void Consumer::targProc() {
    targ.reset_get();
    wait();
    while(true) {
        if (targ.request())
            doSomething(targ.get());
        wait();
}}
```

FIFO example

```
SC_MODULE(Top) {
    sct_fifo<T, N>   fifo{"fifo"};
    SC_CTOR(Top) {
        fifo.clk_nrst(clk, nrst);

        SC_THREAD(producerProc);
        sensitive << fifo.PUT;
        async_reset_signal_is(nrst, false);

        SC_METHOD(consumerProc);
        sensitive << fifo.GET;
}}
```

```
void Top::producerProc() {
    fifo.reset_put();
    wait();
    while (true) {
        fifo.b_put(getSomeValue());
        wait();
}}

void Top::consumerProc() {
    fifo.reset_get();
    if (fifo.request())
      doSomething(fifo.get());
}}
```

Pipeline example

```
SC_MODULE(Top) {
    sct_target<T>       targ{"targ"};
    sct_initiator<T>    init{"init"};
    sct_pipe<T, N>      pipe{"pipe"};
    SC_CTOR(Top) {
        targ.clk_nrst(clk, nrst);
        init.clk_nrst(clk, nrst);
        pipe.clk_nrst(clk, nrst);

        SC_METHOD(pipeProc);
        sensitive << targ << init << pipe;
}}
```

```
void Top::pipeProc() {
    targ.reset_get();
    init.reset_put();
    pipe.reset();
    if (pipe.ready() && targ.request()) {
        // Heavy computation to be pipelined
        T data = doSomething(targ.get());
        pipe.put(data);
    }
    if (pipe.request() && init.ready())
        init.put(pipe.get());
}}
```

Experiments

Design	Original Line of Code	SingleSource Line of Code	Simulation speed up
A	360	191	7.9
B	517	287	6.0
C	1624	886	4.3
D	1411	1203	6.2
E	690	465	8.1
F	-	-	6.6

Comparison with existing solutions

	MatchLib	SingleSource
Vendor	Nvidia	Intel
License	Apache 2.0	Apache 2.0 with LLVM exceptions
Compatibility	Catapult C HLS	Intel Compiler for SystemC
Interfaces	Mostly blocking	Mostly non-blocking
Content	Target/initiator, channels, FIFOs	Target/initiator, FIFO, pipe, ports
Processes	Clocked thread	All
Modules	Latency 1+	All

Other channels and features

- Other channels: Register, Buffer, Clock, Clock gate cell, Clock signal
- Ports of Target/Initiator and other channels (`sc_port<>`)
- Array/vector of channels
- C++ class as channel payload
- Various clock edge and reset levels
- Connection between modular interfaces with FIFO
- Cycle accurate and SingleSource code mix
- Examples https://github.com/intel/systemc-compiler/wiki/SingleSource-library

Conclusion

- SingleSource library provides communication channels to be used for
  - High-level models for architecture exploration
  - Hardware models to integrate into virtual platform for software development
  - Cycle-accurate hardware models for SystemVerilog high-level synthesis
- SingleSource library increases design efficiency
  - Reduce IP design time in ~2x
  - Speed up simulation in TLM mode in ~4x-8x
- SingleSource library channels and Intel Compiler for SystemC
  - GitHub repo: https://github.com/intel/systemc-compiler

Questions