EMBECOSM®

Hardware Modeling

Jeremy Bennett

Chip Hack, 21 April 2013

EMBECOSM®

Modeling Technologies

- Event driven simulators
  - model all of Verilog/VHDL, not just the synthesizable subset
  - commercially: Modelsim, NC and VCS
  - open source: Icarus Verilog, GHDL
- Cycle accurate modeling tools
  - model synthesizable Verilog/VHDL
  - commercially: Carbon Design Systems SpeedCompiler
  - open source: Verilator
  - hand-writing: SystemC
- High level models
  - transaction level modelling with SystemC
- Wave trace visualization
  - GTKWave

GTKWave

- Open source waveform viewer
  - http://gtkwave.sourceforge.net/
  - works on Verilog Change Dump (VCD) files
- Features
  - reads standard VCD
    - also supports LXT, LXT2, VZT, FST, and GHW formats
  - *twinwave* command for viewing two waveforms alongside
- Example
  - we need to generate VCD files from some sort of model or simulator.

EMBECOSM®

Icarus Verilog

- Open Source Verilog simulator
  - http://iverilog.icarus.com/
- Features
  - all of Verilog 95/2001/2005
  - some support for SystemVerilog 2005/2009
  - no VHDL
  - slower than commercial simulators
- Example

```
$ iverilog -DSIMULATE -o basics basics.v basics_tb.v
$ vvp basics
ctrl-C
> finish
$
```

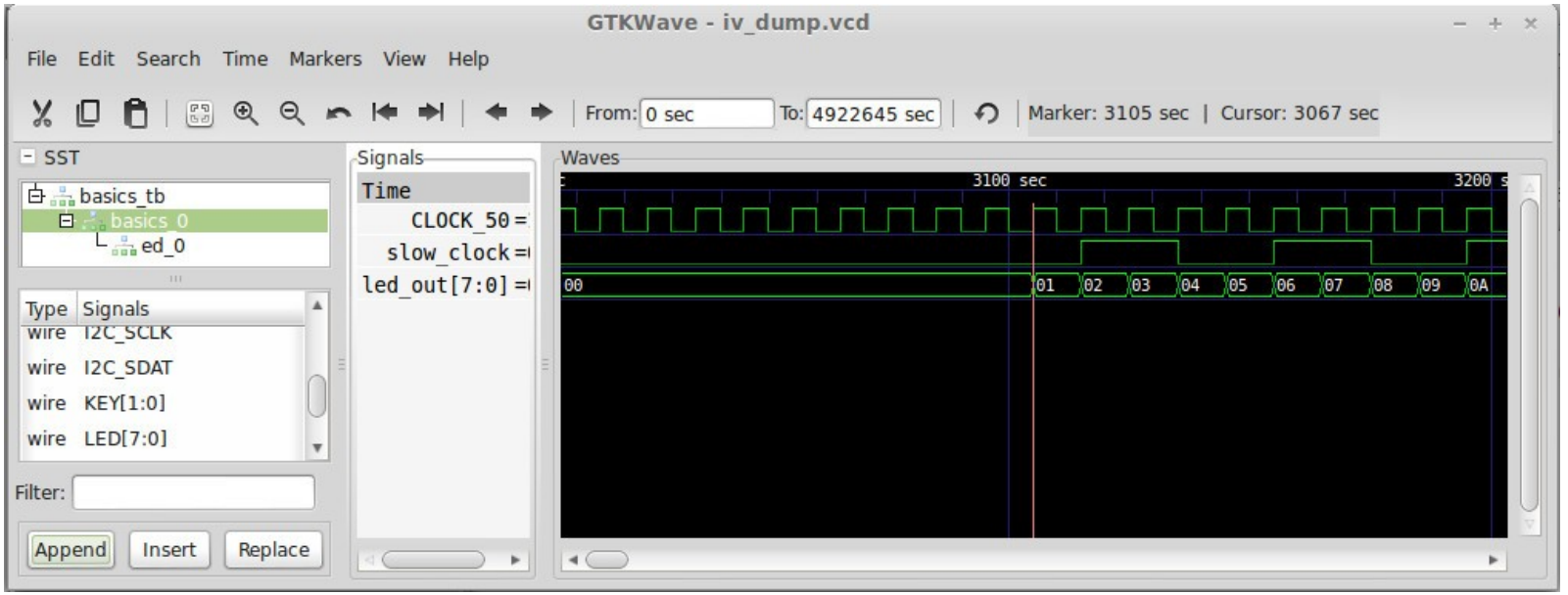
Icarus Verilog Wave Trace

- Simulate

```
$ iverilog -DSIMULATE -DIV_DUMP -o basics basics.v basics_tb.v
$ vvp basics
ctrl-C
> finish
$
```

- Visualize

```
$ gtkwave iv_dump.vcd
```

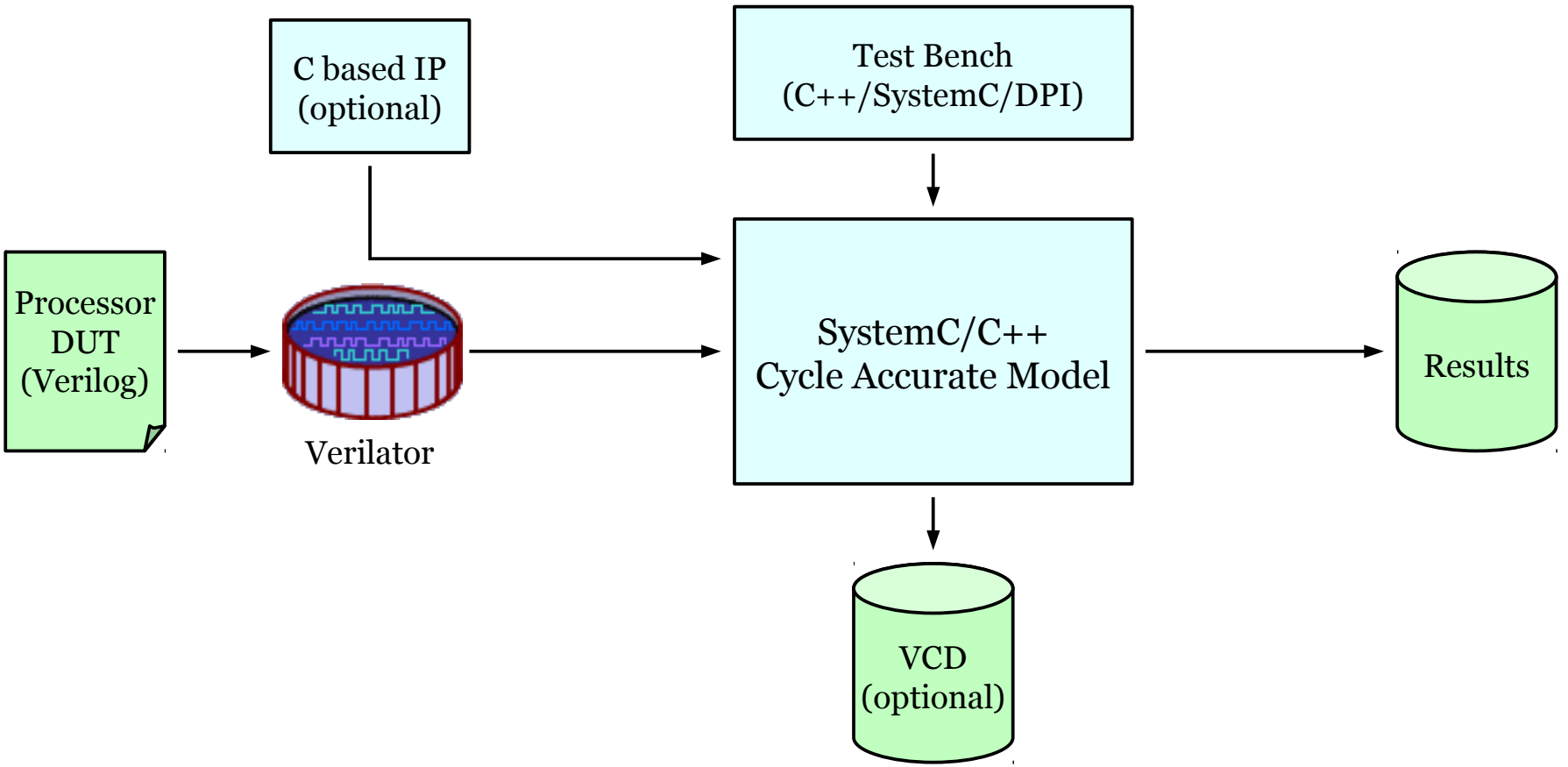


EMBECOSM®

About Verilator

- What is Verilator?
  - open source tool synthesizing Verilog/SystemVerilog to C++/SystemC
    - **Note.** synthesis only
  - models are 2-state, zero delay and cycle accurate.
- What is Verilator used for?
  - system verification: fast cycle accurate models of BIG systems
  - regression testing on server farms: no license fees to worry about
  - processor models for firmware development: e.g Atmel Studio
- How is Verilator developed?
  - originated by DEC in 1994, open sourced in 1998
  - taken over by Wilson Snyder in 2001, complete rewrite in C++
  - Embecosm provide commercial support & development

Modeling Process



Verilator C++ Example

- Need C++ test bench

```
#include "Vbasics.h"
#include "verilated.h"

#include <iostream>

int main (int   argc,
  char *argv[],
  char *env[])
{
  Verilated::commandArgs (argc, argv);
  Vbasics *top = new Vbasics;

  int  old_led;
  vluint64_t main_time = 0;

  top->KEY = 0x3;
  top->CLOCK_50 = 0;
  old_led = top->LED - 1;

  do
    {
      top->eval ();

      if (top->LED != old_led)
        {
          std::cout << "led = " <<
            std::hex << (int) top->LED
            << std::endl;
          old_led = top->LED;
        }

      top->CLOCK_50 =
          1 - top->CLOCK_50;
      main_time += 1;
    }
  while (top->LED != 0xfc);

  top->final();

}
```

Building a Verilator Model

- Convert the Verilog to C++

```
$ verilator -DSIMULATE --cc basics.v --exe basics_main.cpp
%Warning-IMPLICIT: basics.v:77: Signal definition not found, creating
implicitly: slow_clock
%Warning-IMPLICIT: Use "/* verilator lint_off IMPLICIT */" and lint_on around
source to disable this message.
%Warning-IMPLICIT: basics.v:81: Signal definition not found, creating
implicitly: reset
%Warning-IMPLICIT: basics.v:82: Signal definition not found, creating
implicitly: button
%Error: Exiting due to 3 warning(s)
%Error: Command Failed /home/jeremy/gittrees/verilator/verilator_bin
-DSIMULATE --cc basics.v --exe basics_main.cpp
```

- Disable that warning!

```
verilator -Wno-IMPLICIT -DSIMULATE --cc basics.v --exe basics_main.cpp
```

- To find all the warnings:

```
verilator -Wall -DSIMULATE --cc basics.v --exe basics_main.cpp
```

Running a Verilator Model

- Change to the model directory

```
cd obj_dir
```

- Compile the model

```
make -f Vbasics.mk Vbasics
```

- Run the model

```
$ ./Vbasics
led = 0
led = 4
led = 8
led = c
led = 10
...
led = f4
led = f8
led = fc
$
```

- Exercise: Print out the clock count as well.
- Exercise: Add a stimulus for the reset button

Tracing with Verilator

- Add header to test bench C++ program

```
#if VM_TRACE
# include <verilated_vcd_c.h> // Trace file format header
#endif
```

- Set up tracing before the main loop

```
#if VM_TRACE
  // Set if called with --trace
  Verilated::traceEverOn(true);
  VerilatedVcdC *tracer = new VerilatedVcdC;
  top->trace (tracer, 99);
  tracer->open ("basics_dump.vcd");
#endif
```

- Dump values during each step in the loop after **`eval ()`**

```
#if VM_TRACE
      tracer->dump (main_time);
#endif
```

- Tidy up at the end

```
#if VM_TRACE
  tracer->close ();
#endif
```

Generating Trace with Verilator

- Generate the C++ with tracing enabled

```
$ verilator --trace -Wno-IMPLICIT -DSIMULATE --cc basics.v --exe
basics_main.cpp
```

- Build the model as before

```
$ cd obj_dir
$ make -f Vbasics.mk Vbasics
```

- Run the model

```
$ ./Vbasics
```

- The VCD is generated and can be viewed with GTKWave

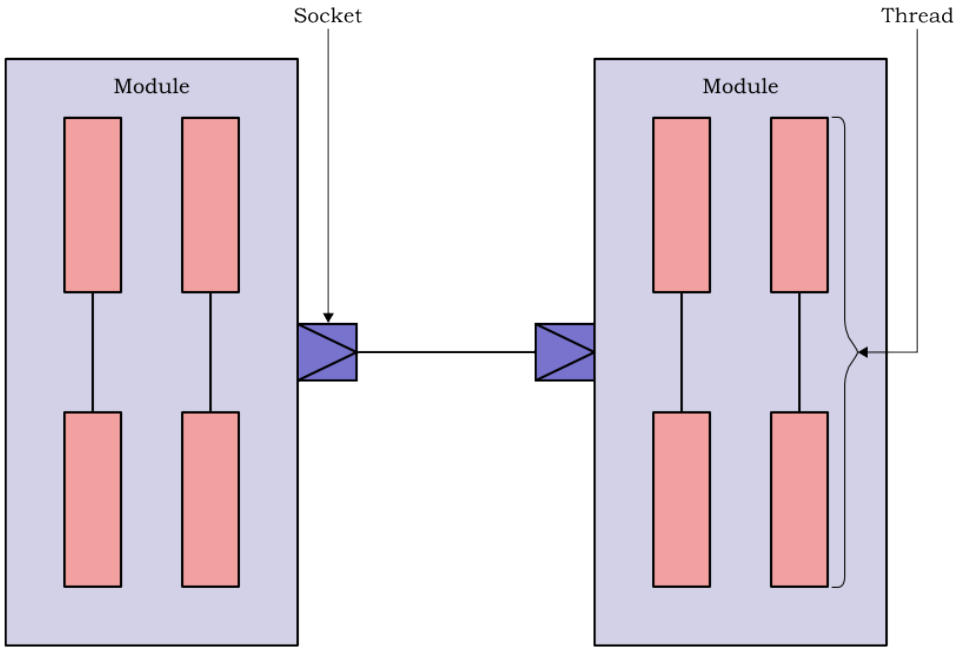
```
$ ls *.vcd
basics_dump.vcd
gtkwave basics_dump.vcd
```

- Exercise: use *twinwave* to compare with the Icarus VCD

- Template library for C++ to support modeling
  - concept of modules
  - modules connected via ports
    - set of C++ types corresponding to different types of connection
  - modules have processes
    - threads which run forever and can be pre-empted
    - methods with run once and cannot be pre-empted
    - sensitive to module data
- No main, instead **`sc_main ()`**
  - called once from the kernel
  - instantiate and connect all the modules
  - then call **`sc_start ()`**

Transaction Level Modeling

- What is TLM?
  - conventionally advance clock, all components advance their state
  - TLM instead have set of processes, communicating via messages
- TLM is more efficient
  - components only compute when they have something to do
  - mirrors way components talk across buses



EMBECOSM®

Further Reading

- More on GTKWave, Icarus Verilog and Verilator
  - http://iverilog.icarus.com/
  - http://www.veripool.org/wiki/verilator
  - http://gtkwave.sourceforge.net/
  - Embecosm App Note #6: High Performance SoC Modeling with Verilator
    - http://www.embecosm.com/appnotes/ean6/embecosm-or1k-verilator-tutorial-ean6-issue-1.html
- More on SystemC:
  - IEEE 1666 (it's free!)
    - http://standards.ieee.org/findstds/standard/1666-2011.html
  - www.systemc.org
- More on TLM:
  - Embecosm App Note #1: Building a Loosely Timed SoC Model with OSCI TLM 2.0
    - http://www.embecosm.com/appnotes/ean1/ean1-tlm2-or1ksim-2.0.html

Thank You

Hope you enjoyed Chip Hack

www.embecosm.com