Constructive Computer Architecture

Tutorial 4: Running and Debugging SMIPS

Andy Wright
6.175 TA

Introduction

- In lab 5, you will be making modifications to an existing, functional, SMIPS processor
- How do you know if your processor is working?
  - You will run an existing suite of C and assembly software test benches on your processor
- What could go wrong?
  - Software and Hardware
- How will you debug this?

Running SMIPS

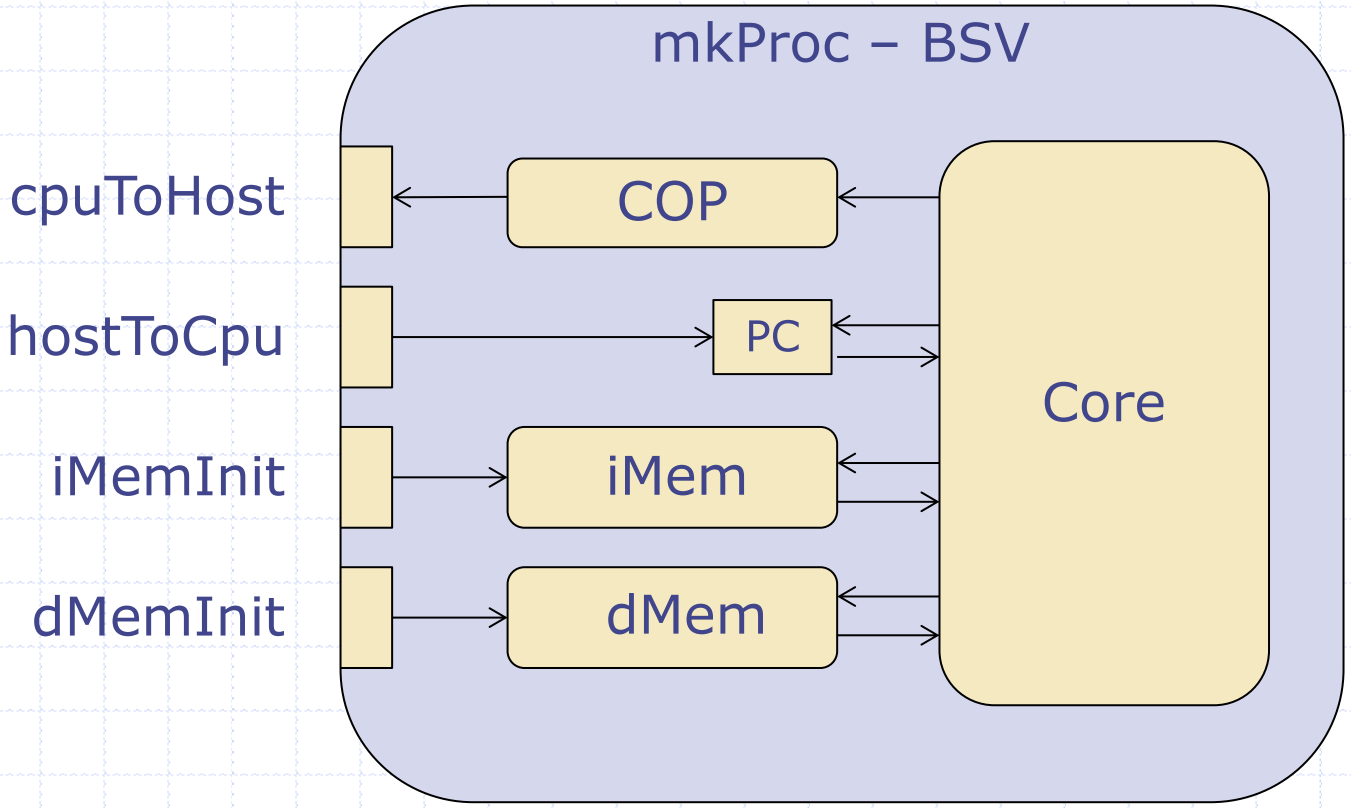
SMIPS Interface

```
interface Proc;
    method ActionValue#(Tuple2#(RIndx,
Data)) cpuToHost;
    method Action hostToCpu(Addr startpc);
    interface MemInitIfc iMemInit;
    interface MemInitIfc dMemInit;
endinterface
```

Sub-interfaces

Slightly different than the interface shown in lecture

Real SMIPS Interface



SMIPS Interface cpuToHost

- mtc0 moves data to coprocessor registers
  - This instruction really just writes data to a FIFO in the form (cop_reg, data)
  - cpuToHost dequeues from that FIFO
- COP Registers:
  - 18: Print data as integer
  - 19: Print data as char
  - 21: Send finish code
    - 0 is passed, all other codes signal failure

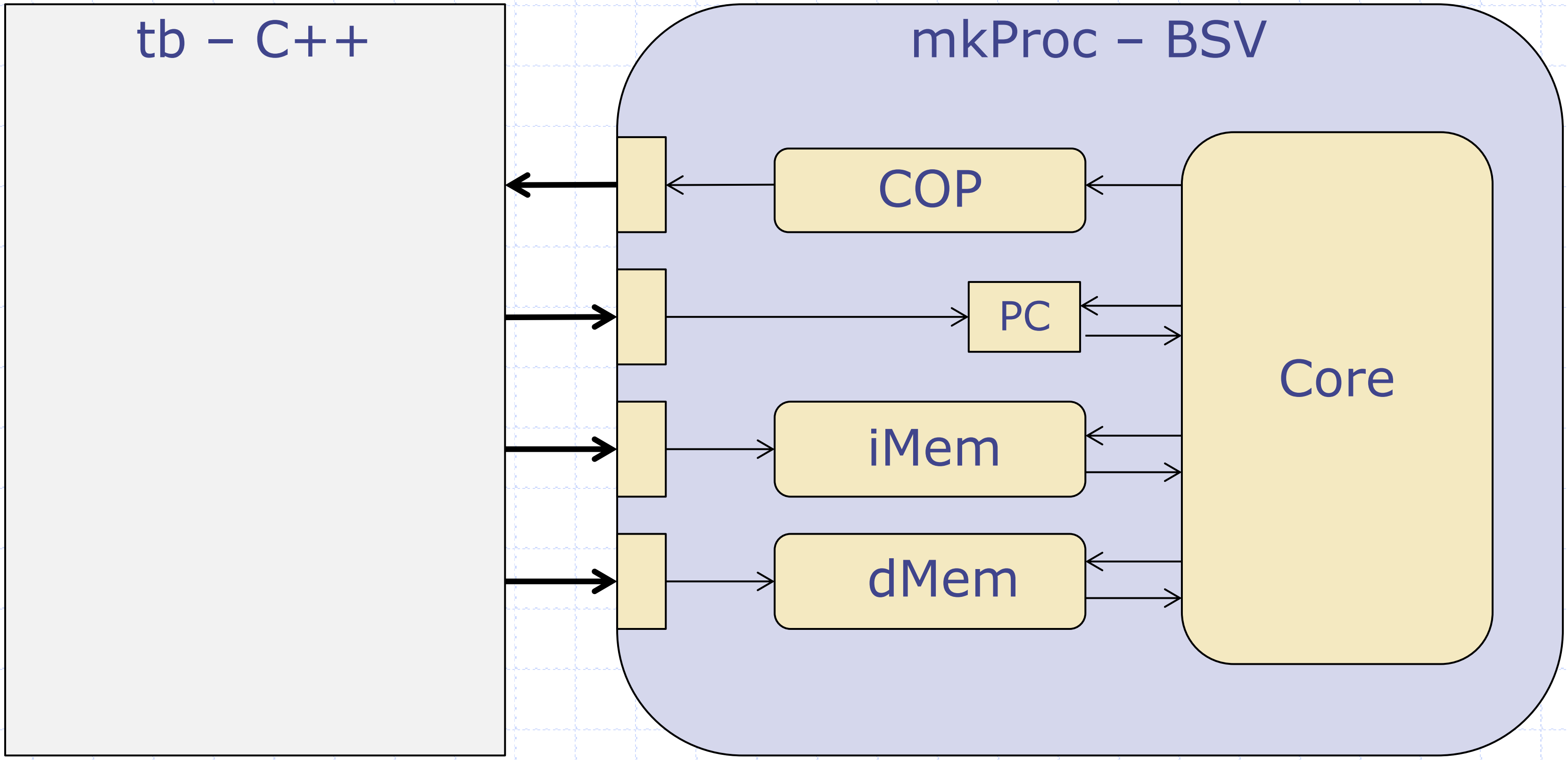
SMIPS Interface
Other Methods/Subinterfaces

- hostToCpu
  - Tells the processor to start running from the given address
- iMemInit/dMemInit
  - Used to initialize iMem and dMem
  - Can also be used to check when initialization is done
  - Defined in MemInit.bsv

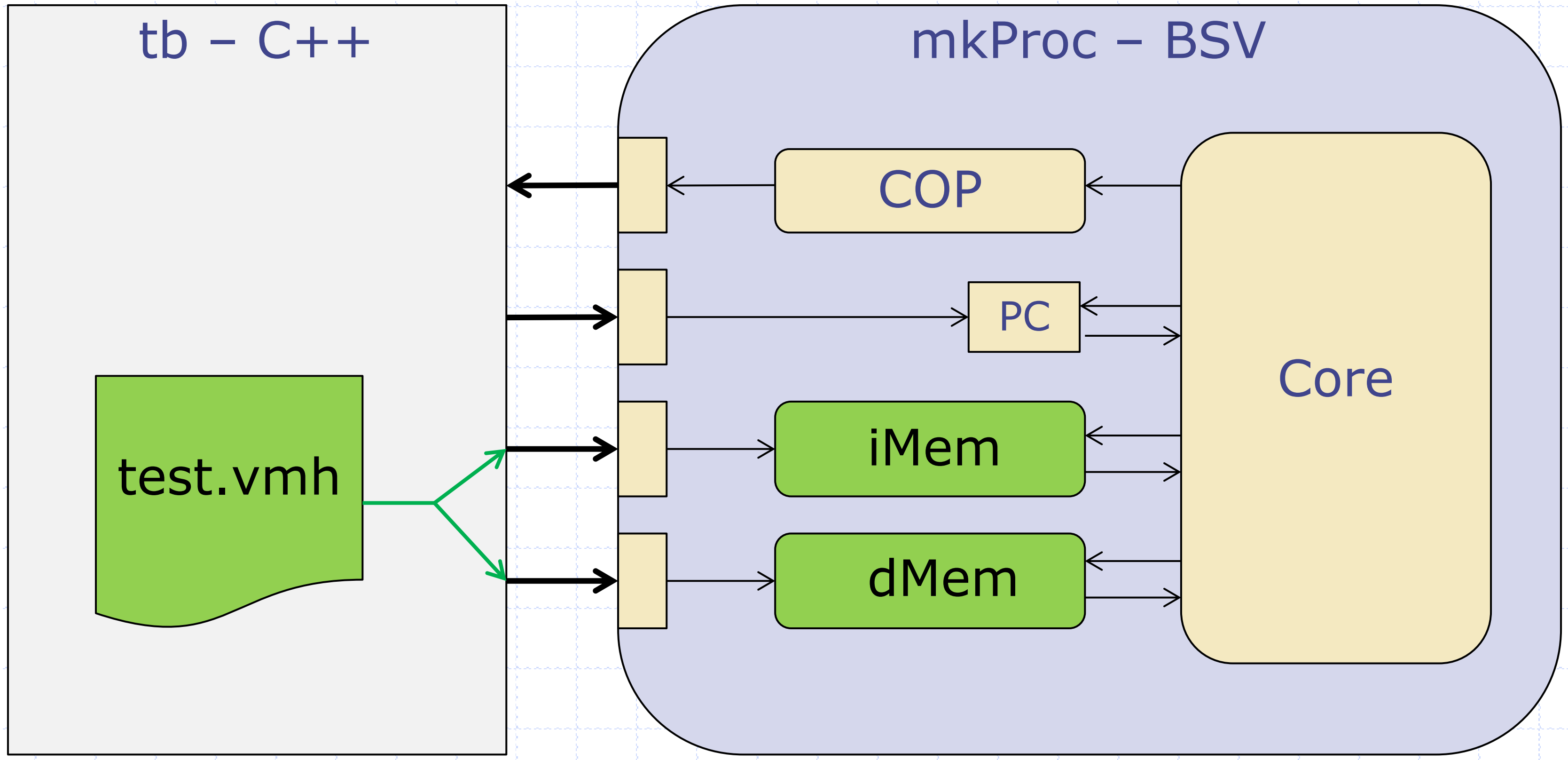
Connecting to the SMIPS interface

- Previous labs have used testbenches written in BSV to connect with modules we wanted to test.
- Now we want a more advanced program testing the processor
  - Want to be able to load multiple files from the user and display printed output
- How do we do this?
  - Use a SceMi interface to connect a testbench written in C++ with a module written in BSV

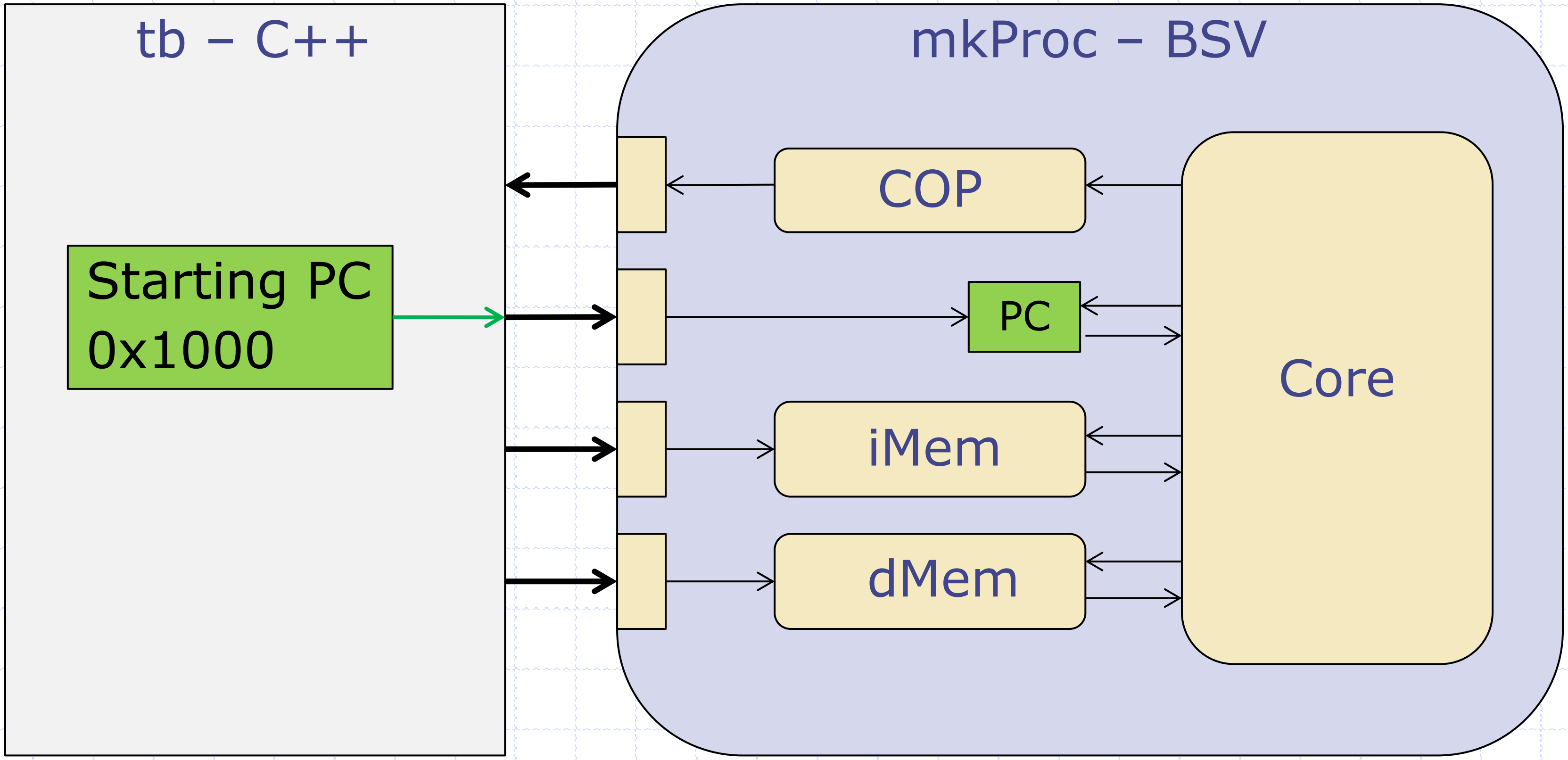
SceMi Testbench Interface



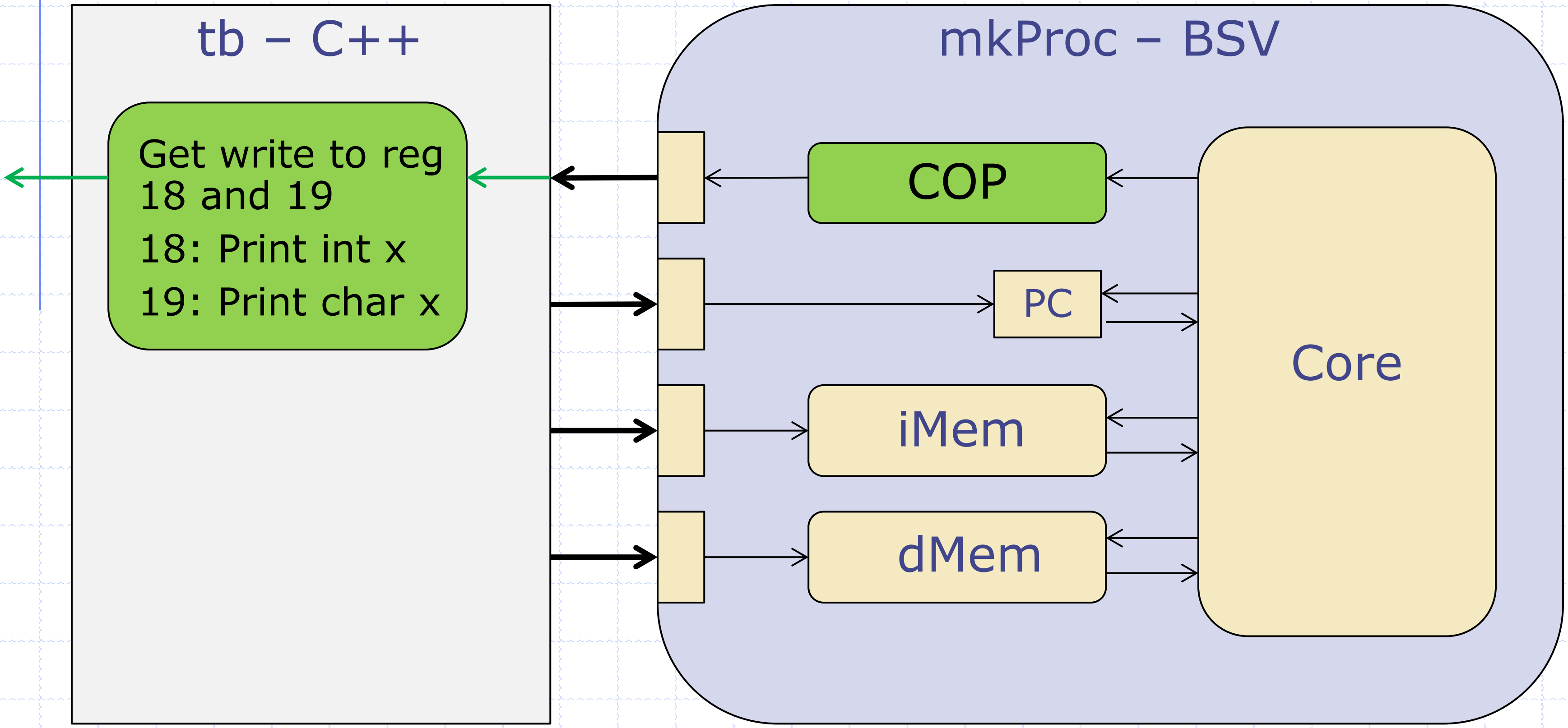
Loading Programs



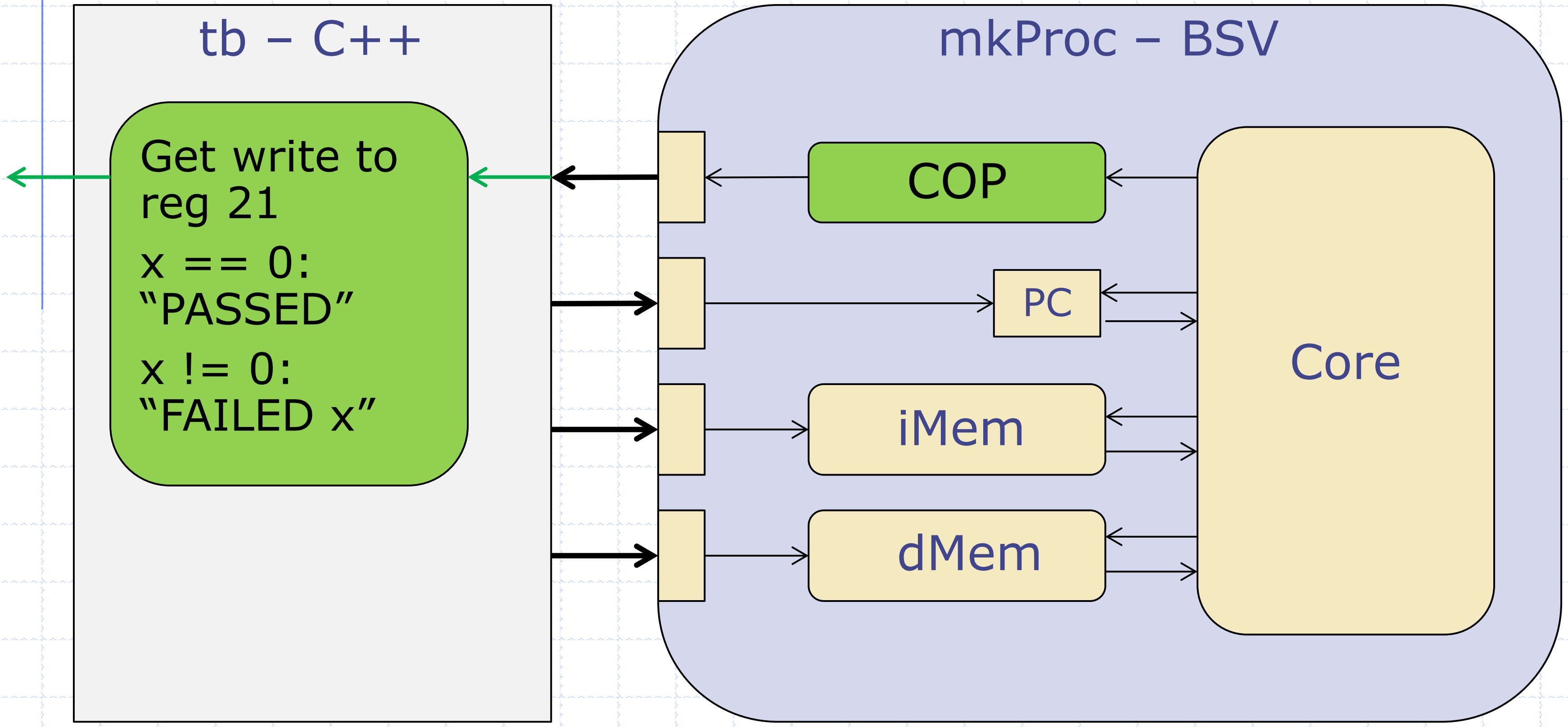
Starting the Processor



Printing to Console

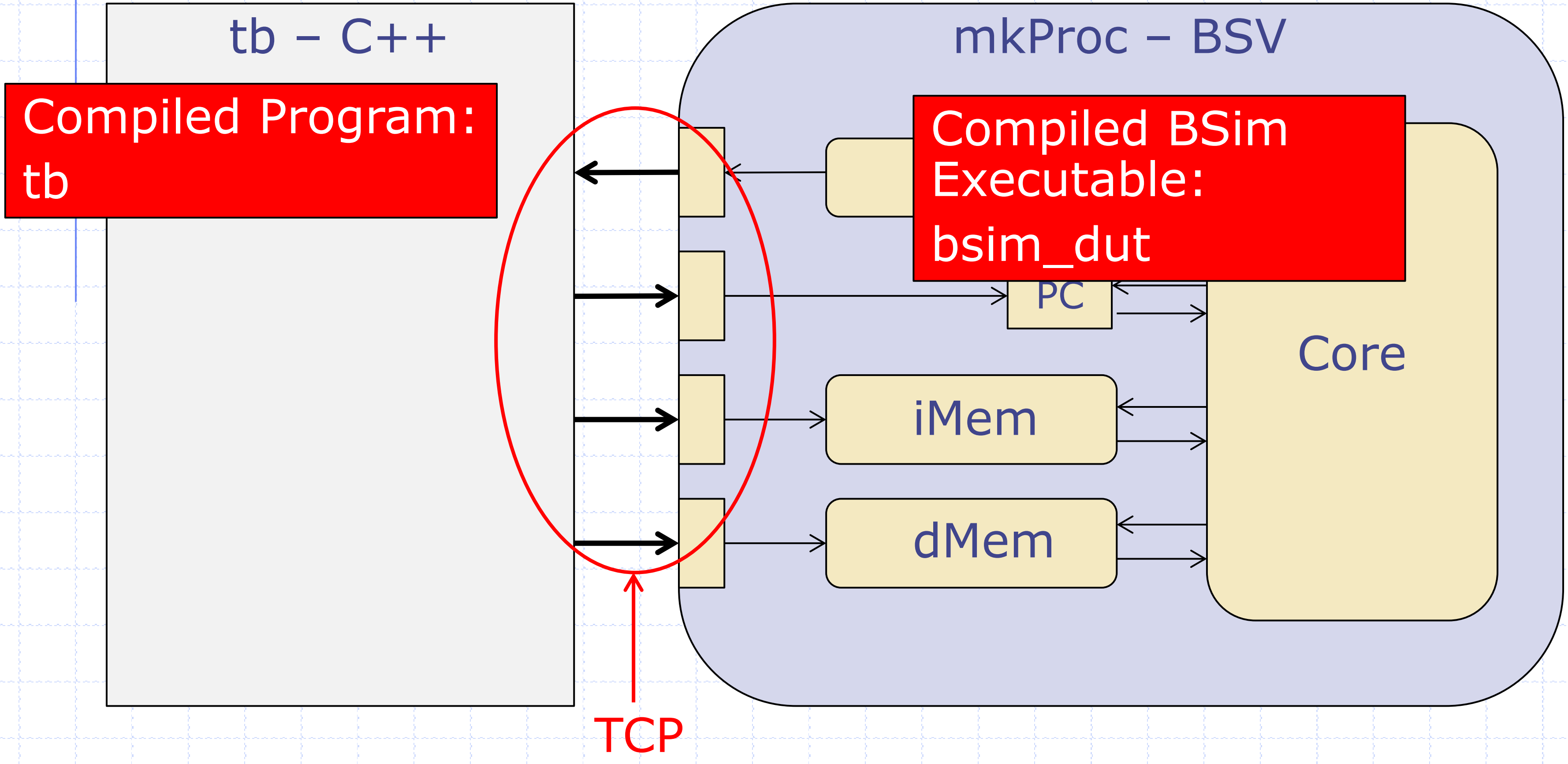


Finishing Program



SceMi Testbench Interface

Simulation



Building the SMIPS processor

```
$ cd scemi/sim
$ build -v onecycle
building target onecycle...
… lots of compiler messages …
done building target onecycle.
```

Builds ./bsim_dut and ./tb

Running SMIPS Simulations

```
$ ./bsim_dut > sim.out &
$ ./tb ../../programs/build/qsort.bench.vmh
../../programs/build/qsort.bench.vmh
Cycles = 21626
Insts  = 21626
PASSED
SceMi Service thread finished!
```

Writes BSV output to sim.out

Printed by SMIPS mtc0 instructions through tb executable

Running SMIPS Simulations

```
$ ./bsim_dut -V test.vcd > /dev/null &
```

Dumps VCD waveform to test.vcd

Ignores BSV output

```
$ ./tb ../../programs/build/qsort.bench.vmh
../../programs/build/qsort.bench.vmh
Cycles = 21626
Insts  = 21626
PASSED
SceMi Service thread finished!
```

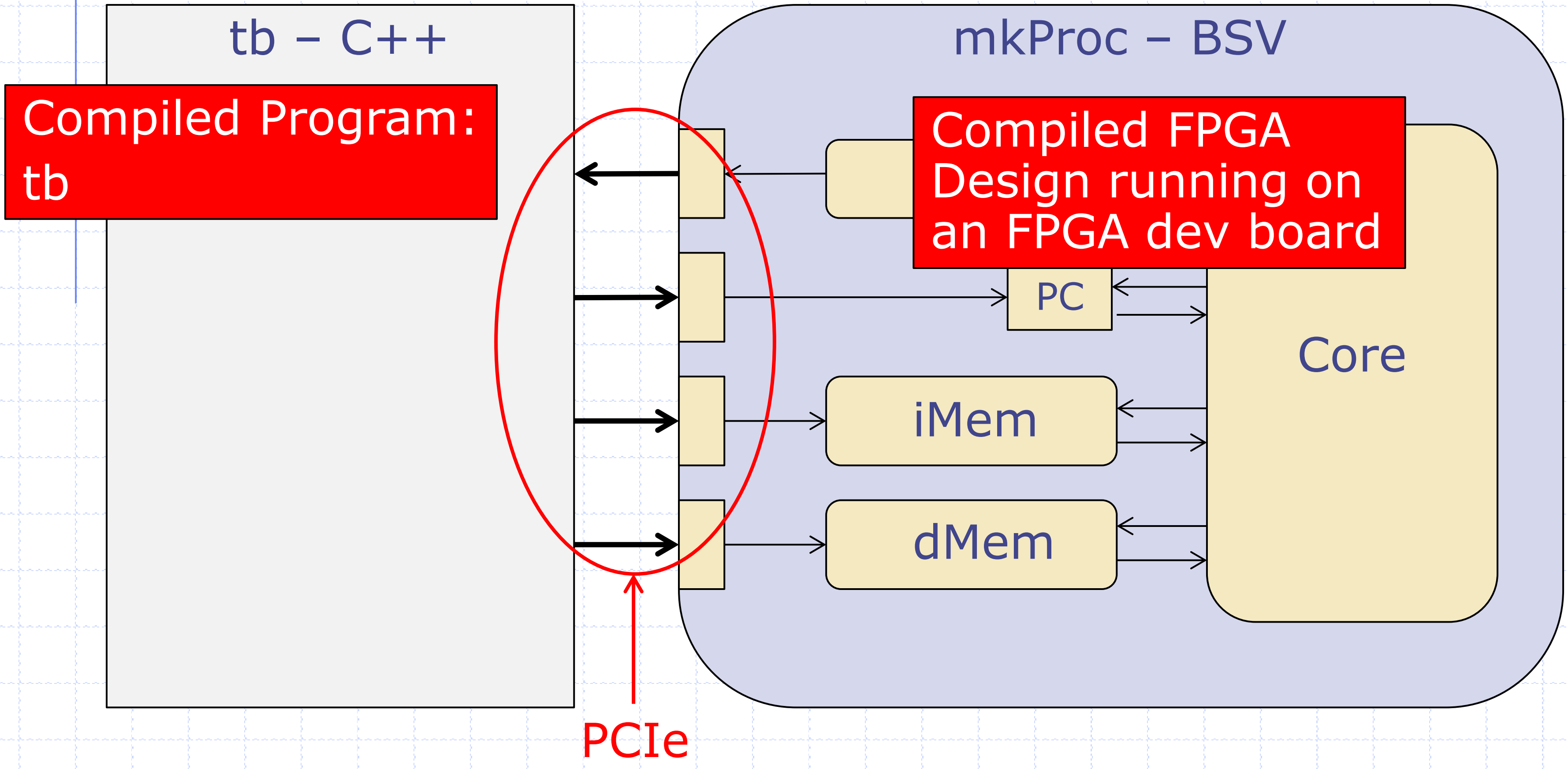
Running all SMIPS Tests

```
$ ./run_assembly
… runs all assembly tests …
$ ./run_benchmarks
../../programs/build/median.bench.vmh
Cycles = 6871
Insts  = 6871
../../programs/build/multiply.bench.vmh
Cycles = 21098
Insts  = 21098
../../programs/build/qsort.bench.vmh
Cycles = 21626
Insts  = 21626
…
SceMi Service thread finished!
```

SceMi Testbench Interface

Hardware

The same SceMi Testbench can be used to test hardware FPGA designs too!



Running SMIPS Hardware

```
$ programfpga
… programs the FPGA …
$ runtb ./tb ../../programs/build/qsort.bench.vmh
../../programs/build/qsort.bench.vmh
Cycles = 21626
Insts  = 21626
PASSED
```

programfpga takes care loading the design, and runtb takes care of communicating with the FPGA board

Your current design can’t run on FPGA boards and won’t be able to run on them until later labs.

Debugging SMIPS

Step 1 – Fail a test

```
$ ./run_assembly
…
../../programs/build/smipsv1_lw.asm.vmh
FAILED 1
…
$ ./run_benchmarks
…
../../programs/build/vvadd.bench.vmh
Executing unsupported instruction at pc: 00000004.
Exiting
…
```

$\leftarrow$ Error Code

BSV Error message

$fwrite(stderr, "Executing unsupported …

We need more feedback from the processor!

Step 2 – Think about it

- Just because you failed smipsv1_lw.asm.vmh doesn’t mean your problem is only with the lw instruction
  - Are other programs failing?

Step 3 – Investigate

```
$ ./bsim_dut -V test.vcd > test.out &
```

Save VCD file		Save BSV output

```
$ ./tb ../../programs/build/smipsv1_lw.asm.vmh
../../programs/build/smipsv1_lw.asm.vmh
FAILED 1
```

- Look at ../../programs/build/smipsv1_lw.asm.vmh to see the program running (the source is in ../../programs/src/assembly/)
- Look at test.out for any BSV messages you outputted
- Look at test.vcd to see the firing rules and the state of the processor
- Also look at *targetname*_compile_for_bluesim.log for compiler warnings

Step 3 – Investigate
Programs

- Test Benches can be written in assembly or in C
  - Assembly examples:
    - baseline.S
    - smipsv2_addu.S
  - C example:
    - vvadd
- The .vmh files are the compiled versions with some human readable assembly as comments

Step 3 – Investigate
BSV Output

- The BSV output may look something like this:

```
…
Cycle     3057 --------------------------------
pc: 00001098 inst: (03e00008) expanded: jr 'h1f

Cycle     3058 --------------------------------
pc: 00000004 inst: (aaaaaaaa) expanded: nop
```

- To debug further, you can add more $display(…) statements

Step 3 – Investigate
Waveforms

- “build –v testname” generates testname.bspec
- Open the bluespec workstation using “bluespec testname.bspec &”
- Open the module viewer to explore the VCD file as mentioned in the previous tutorial
- Can view states and rule signals
  - CAN_FIRE tells you if rule gaurds are true
  - WILL_FIRE tells you if rules are actually firing

Step 3 – Investigate
Compiler Output

- The compiler will output many warnings when compiling the SMIPS processor
  - Most of them are because of the SceMi interface
  - All mkProc warnings will appear before any of the SceMi warnings
- The compiler gives a warning when
  - Two rules conflict
  - A rule never fires
  - Many other useful situations…

Step 4 – Get More Information

- Create new programs
  - New assembly programs can just be added to programs/src/assembly and then build by running “make” in programs/
  - New C programs require adding targets to the makefile
  - ./run_assembly and ./run_benchmarks will find all *.asm.vmh and all *.bench.vmh files that have been built

Step 4 – Get More Information

- Add more debug outputs from the processor
  - New display statements will show up in stdout
  - $fwrite(stderr, "…") messages will show up in stderr
  - Add Probes to your module to get more information in the waveform viewer

Step 4 – Get More Information: Probes

- A Probe has an interface similar to a register, but it can't be read
  - Import Probe::*;
  - Probe#(t) myprobe <- mkProbe();
  - myprobe <= my_value; (only inside rules)
- Probes will be preserved by the compiler and you will be able to see them in the VCD viewer
- Possible probe idea:
  - Create a vector of probes to output the state of a FIFO in logical order (probe[0] is the first element in the fifo, and so on)

Step 5 – Fix the bug

- (hopefully you found it by this point)

Questions?