Constructive Computer Architecture

Tutorial 4

Debugging

Sizhuo Zhang

6.175 TA

BSV Debug -- $display

- See a bug, not sure what causes it
- Add $display statements
- Recompile
- Run
- Still see bug, but you have narrowed it down to a smaller portion of code
- Repeat with more display statements...
- Find bug, fix bug, and remove display statements

BSV Display Statements

- The $display() command is an action that prints statements to the simulation console
- Examples:
  - `$display(“Hello World!”);`
  - `$display(“The value of x is %d”, x);`
  - `$display(“The value of y is “, fshow(y));`

Ways to Display Values
Format Specifiers

- %d – decimal
- %b – binary
- %o – octal
- %h – hexadecimal
- %0d, %0b, %0o, %0h
  - Show value without extra whitespace padding

Ways to Display Values
fshow

- fshow is a function in the FShow typeclass
- It can be derived for enumerations and structures
- Example:

```
typedef emun {Red, Blue} Colors
deriving(FShow);
Color c = Red;
$display("c is ", fshow(c));
```

Prints "c is Red"

BSV Debugging Waveform Viewer

- Simulation executables can dump VCD waveforms
  - `./bsim_dut -V test.vcd`
- Produces test.vcd containing the values of all the signals used in the simulator
  - Not the same as normal BSV signals
- VCD files can be viewed by a waveform viewer
  - Such as gtkwave
- The signal names and values in test.vcd can be hard to understand
  - Especially for structures and enumerations

Bluespec GUI Example

- We use Bluespec GUI + gtkwave to view waveform of VCD files
  - gtkwave Installed in vlsifarm matchines
- SSH with -X
  - ssh –X username@vlsifarm-03.mit.edu
  - Windows may need some tools
    - Cygwin/X
- Use onecycle processor in lab5 as Example

Bluespec project

- We have set SceMi to generate it when building the project
  - project.bld: workstation-project-file

```
[onecycle]
extends-target:              bsim_dut
bsv-define:                  PROC_FILE=OneCycle SIM
workstation-project-file: onecycle.bspec
```

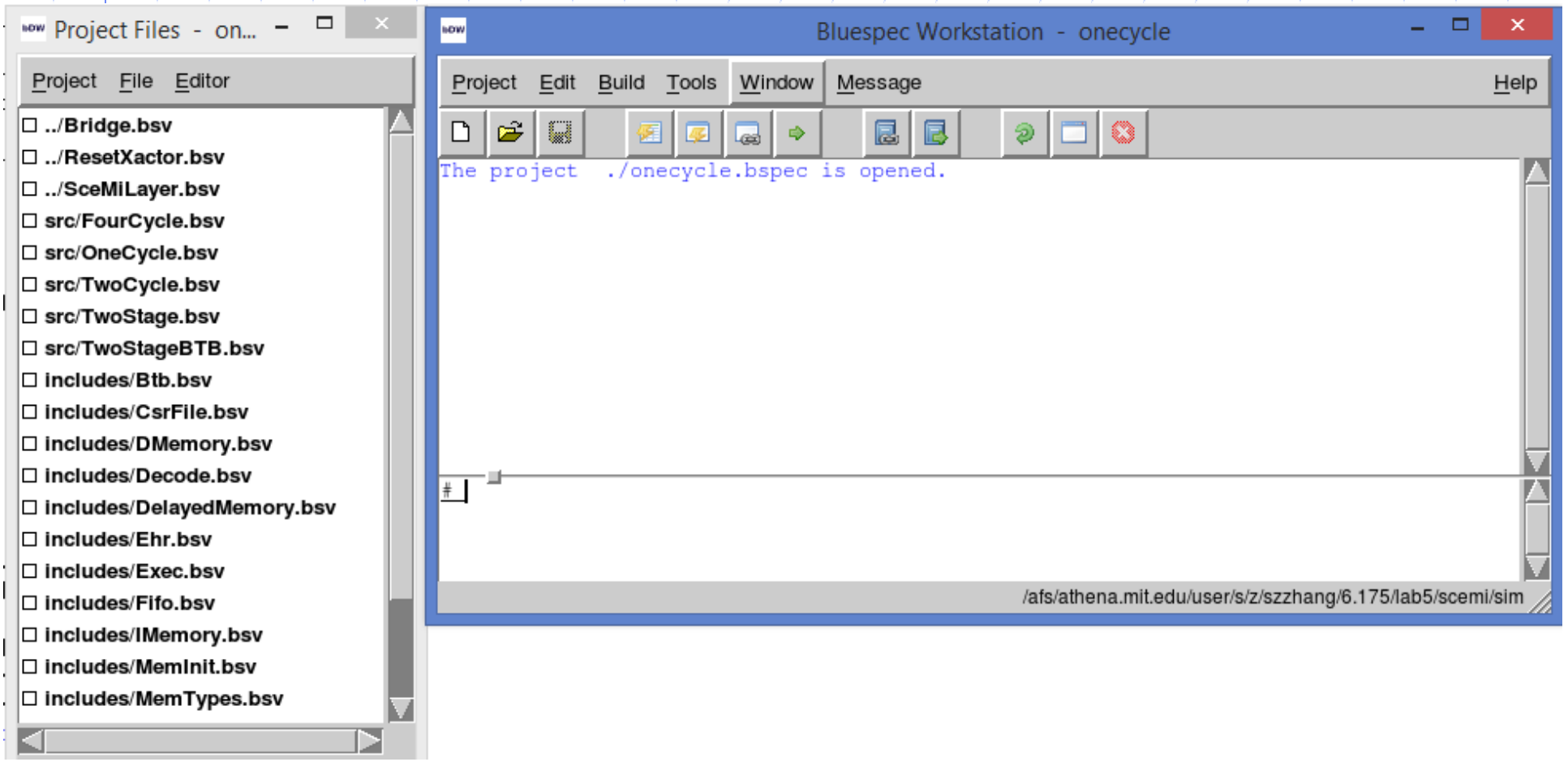
  - $ cd scemi/sim
  - $ build –v onecycle
  - onecycle.bspec is the project file

Simulation: generate VCD file

```
$ cp
../../programs/build/assembly/vmh/simple.riscv.vmh
  mem.vmh
$ ./bsim_dut –V out.vcd > log.txt &
$ ./tb
```

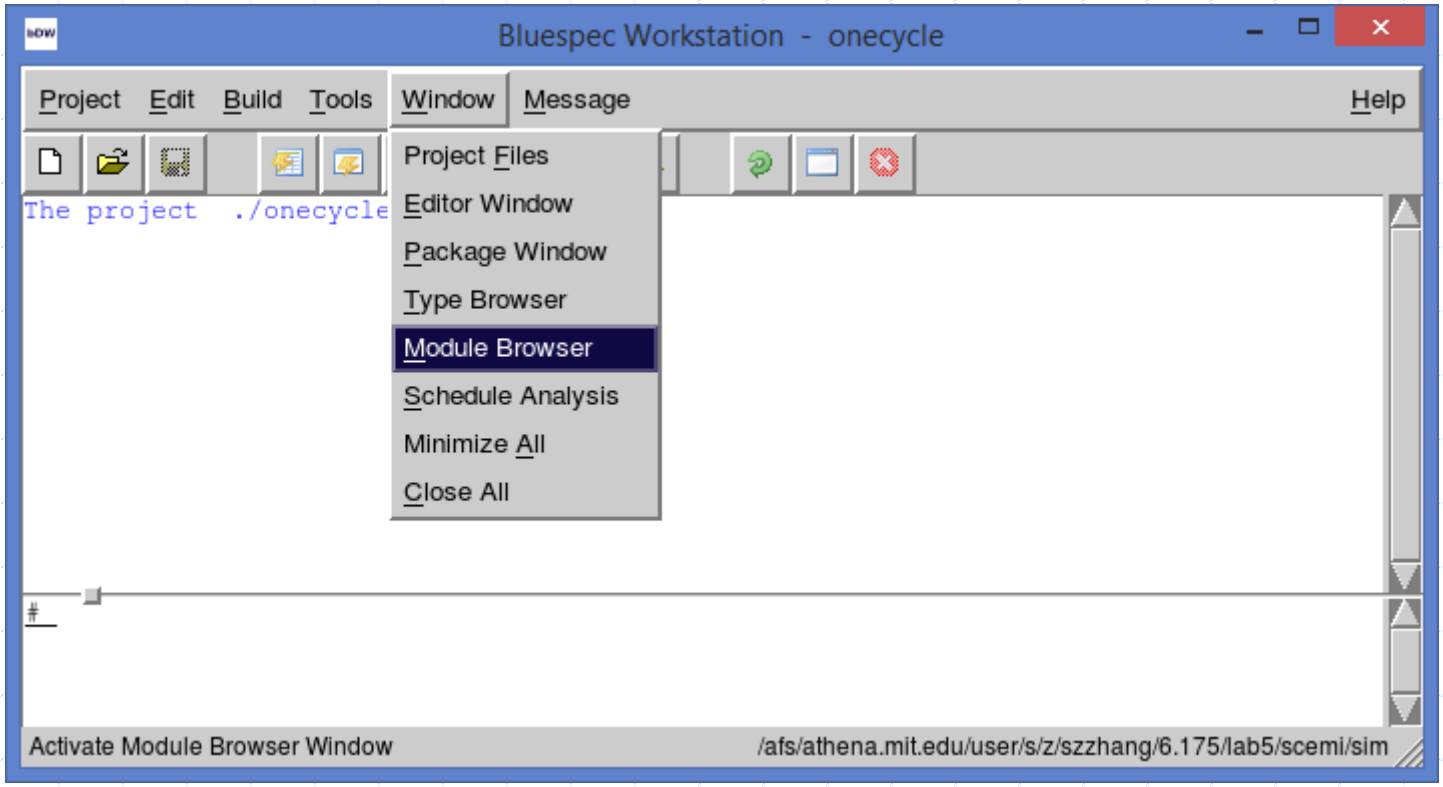
Open Bluespec GUI

$ bluespec onecycle.bspec



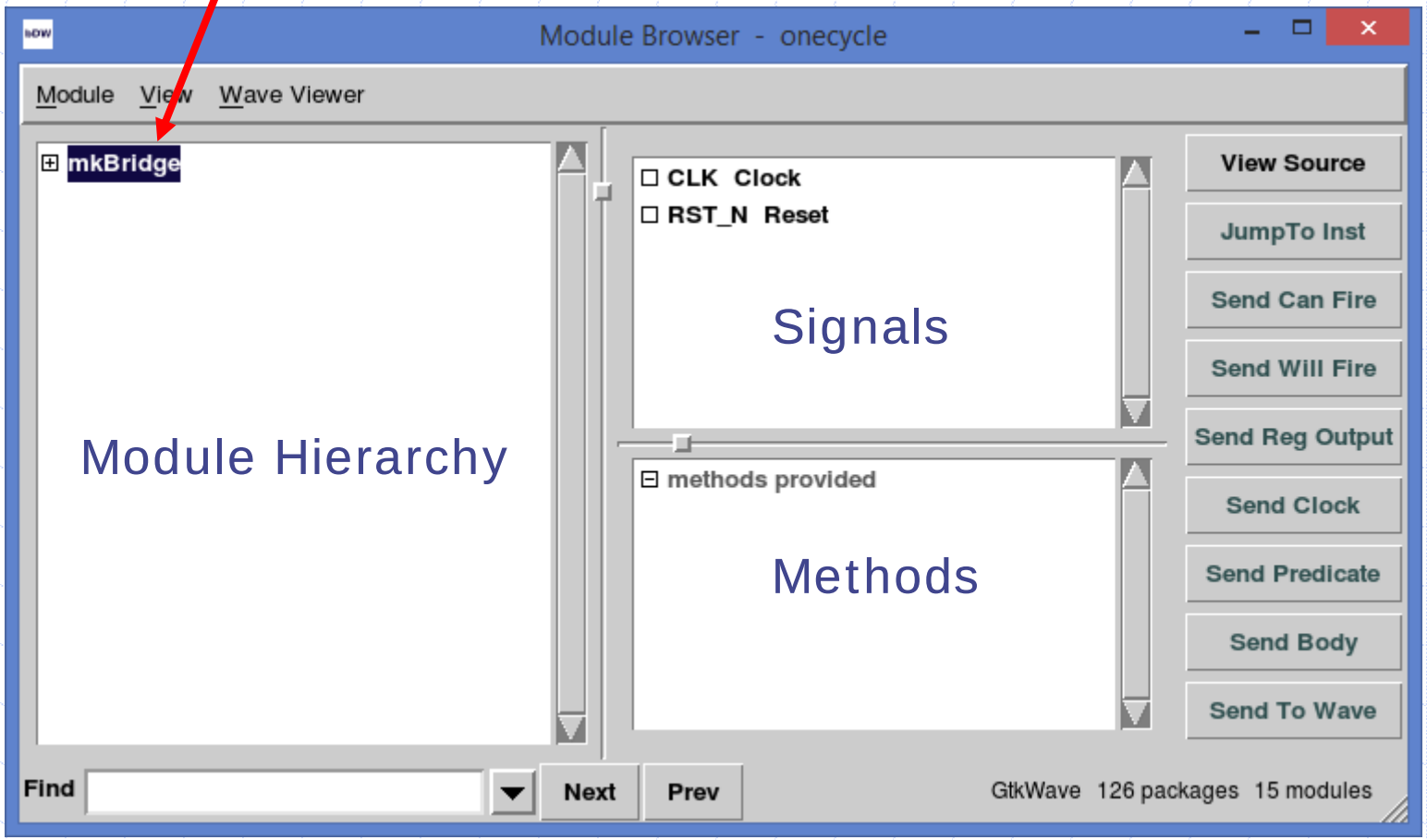
Open Module Browser

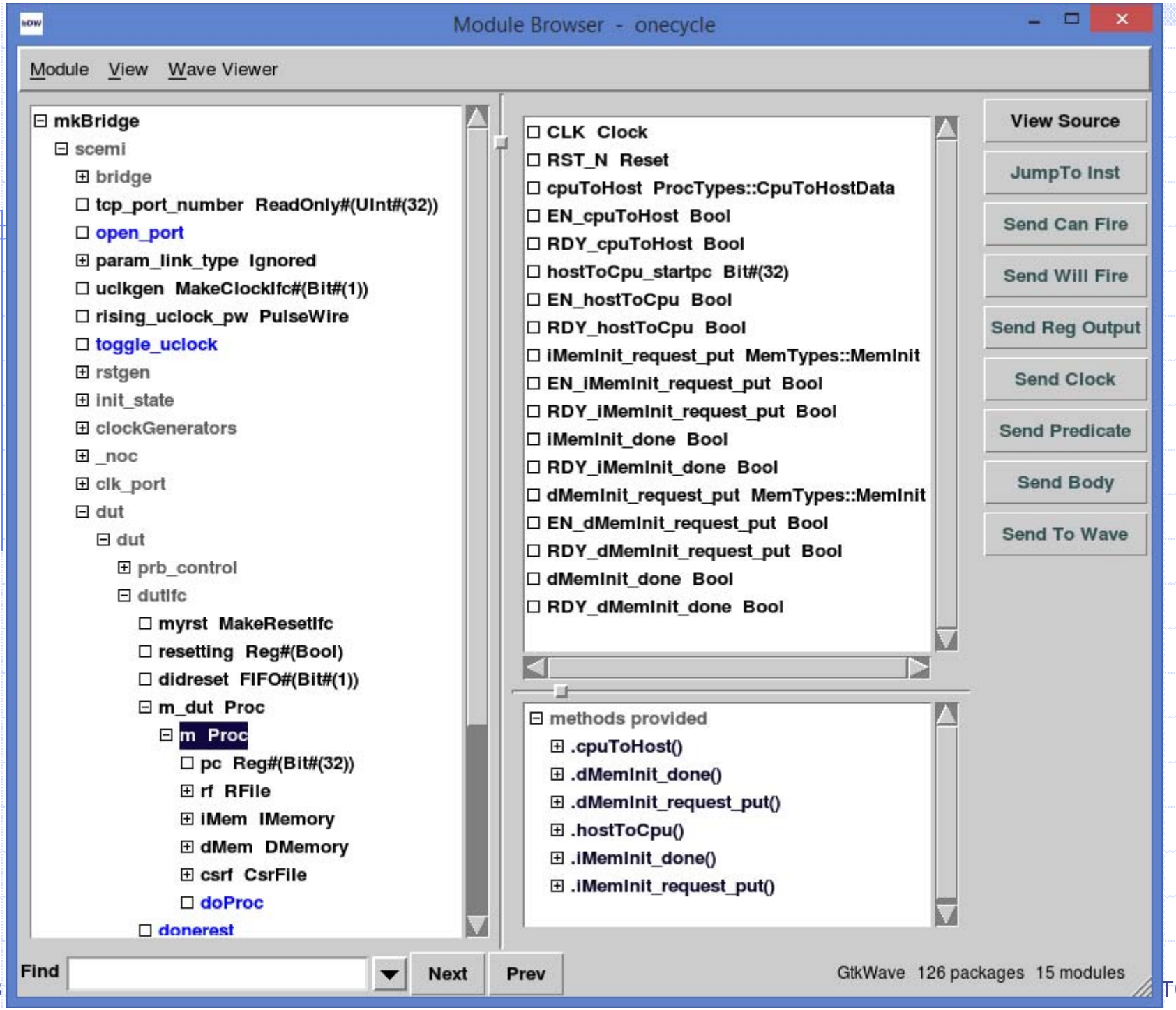
- Window → Module Browser



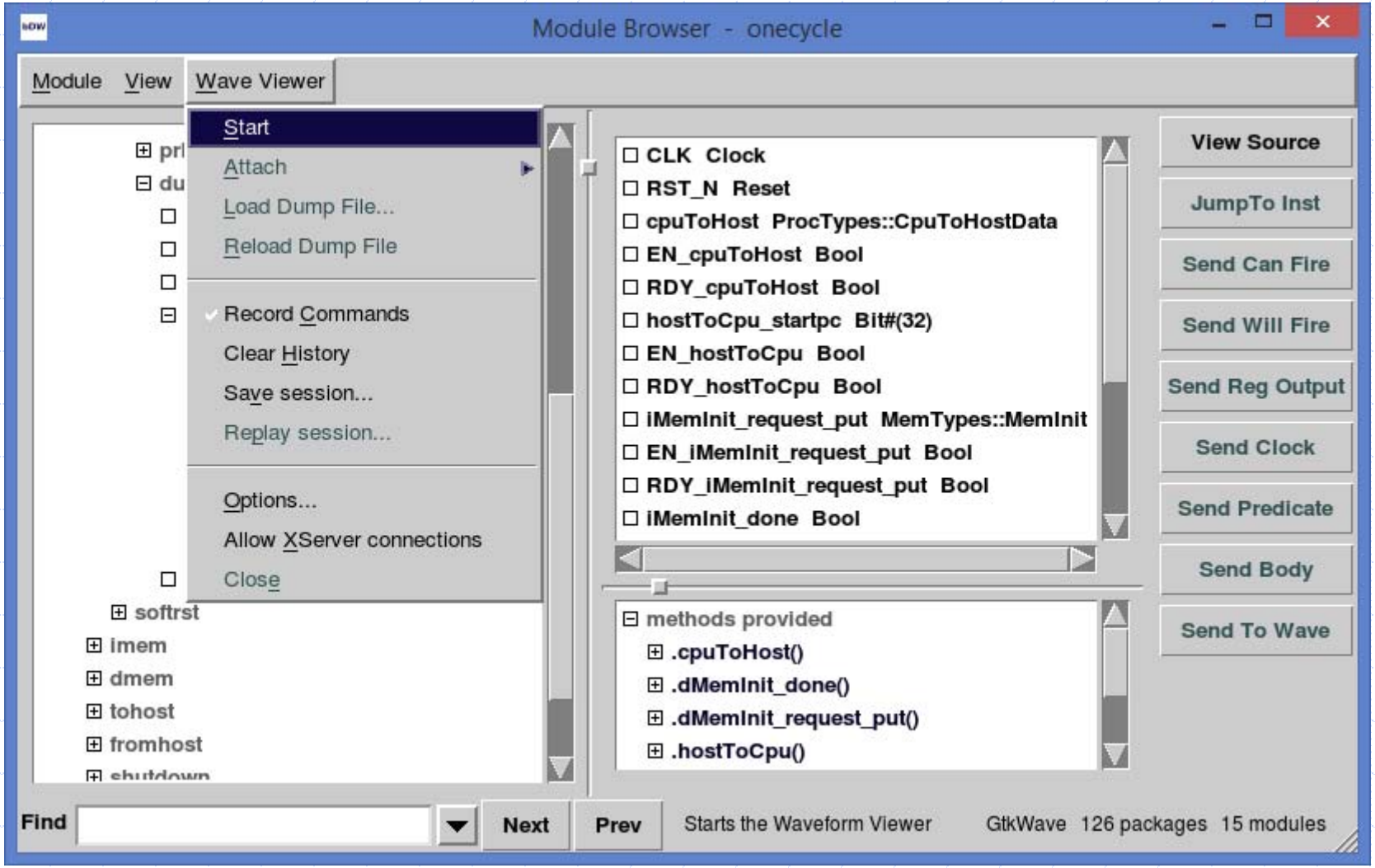
Module Browser

Top level module

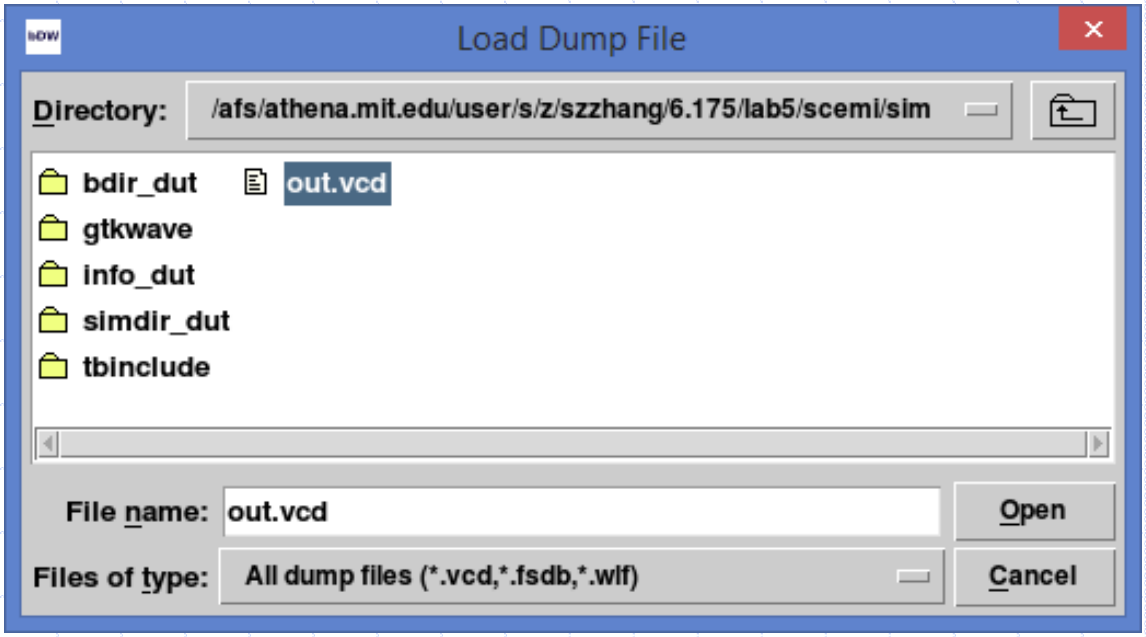



Module Browser - onecycle
Module View Wave Viewer
mkBridge
scemi
bridge
tcp_port_number ReadOnly#(UInt#(32))
open_port
param_link_type Ignored
uclkgen MakeClockIfc#(Bit#(1))
rising_uclock_pw PulseWire
toggle_uclock
rstgen
init_state
clockGenerators
_noc
clk_port
dut
dut
prb_control
dutIfc
myrst MakeResetIfc
resetting Reg#(Bool)
didreset FIFO#(Bit#(1))
m_dut Proc
m Proc
pc Reg#(Bit#(32))
rf RFile
iMem IMemory
dMem DMemory
csrf CsrFile
doProc
donerest
CLK Clock
RST_N Reset
cpuToHost ProcTypes::CpuToHostData
EN_cpuToHost Bool
RDY_cpuToHost Bool
hostToCpu_startpc Bit#(32)
EN_hostToCpu Bool
RDY_hostToCpu Bool
iMemInit_request_put MemTypes::MemInit
EN_iMemInit_request_put Bool
RDY_iMemInit_request_put Bool
iMemInit_done Bool
RDY_iMemInit_done Bool
dMemInit_request_put MemTypes::MemInit
EN_dMemInit_request_put Bool
RDY_dMemInit_request_put Bool
dMemInit_done Bool
RDY_dMemInit_done Bool
methods provided
.cpuToHost()
.dMemInit_done()
.dMemInit_request_put()
.hostToCpu()
.iMemInit_done()
.iMemInit_request_put()
View Source
JumpTo Inst
Send Can Fire
Send Will Fire
Send Reg Output
Send Clock
Send Predicate
Send Body
Send To Wave
Find
Next
Prev
GtkWave 126 packages 15 modules

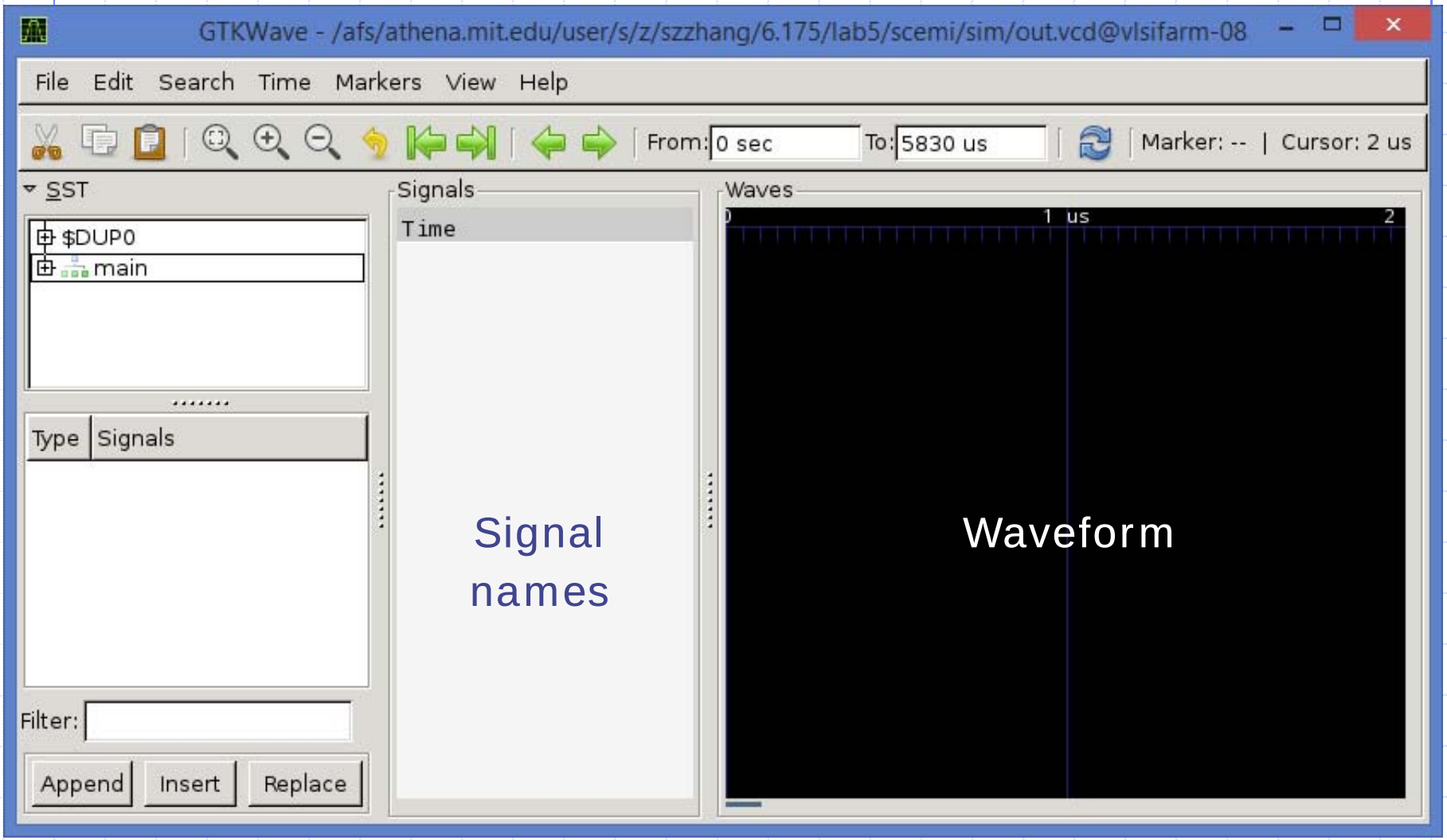
Open Waveform Viewer



Load VCD

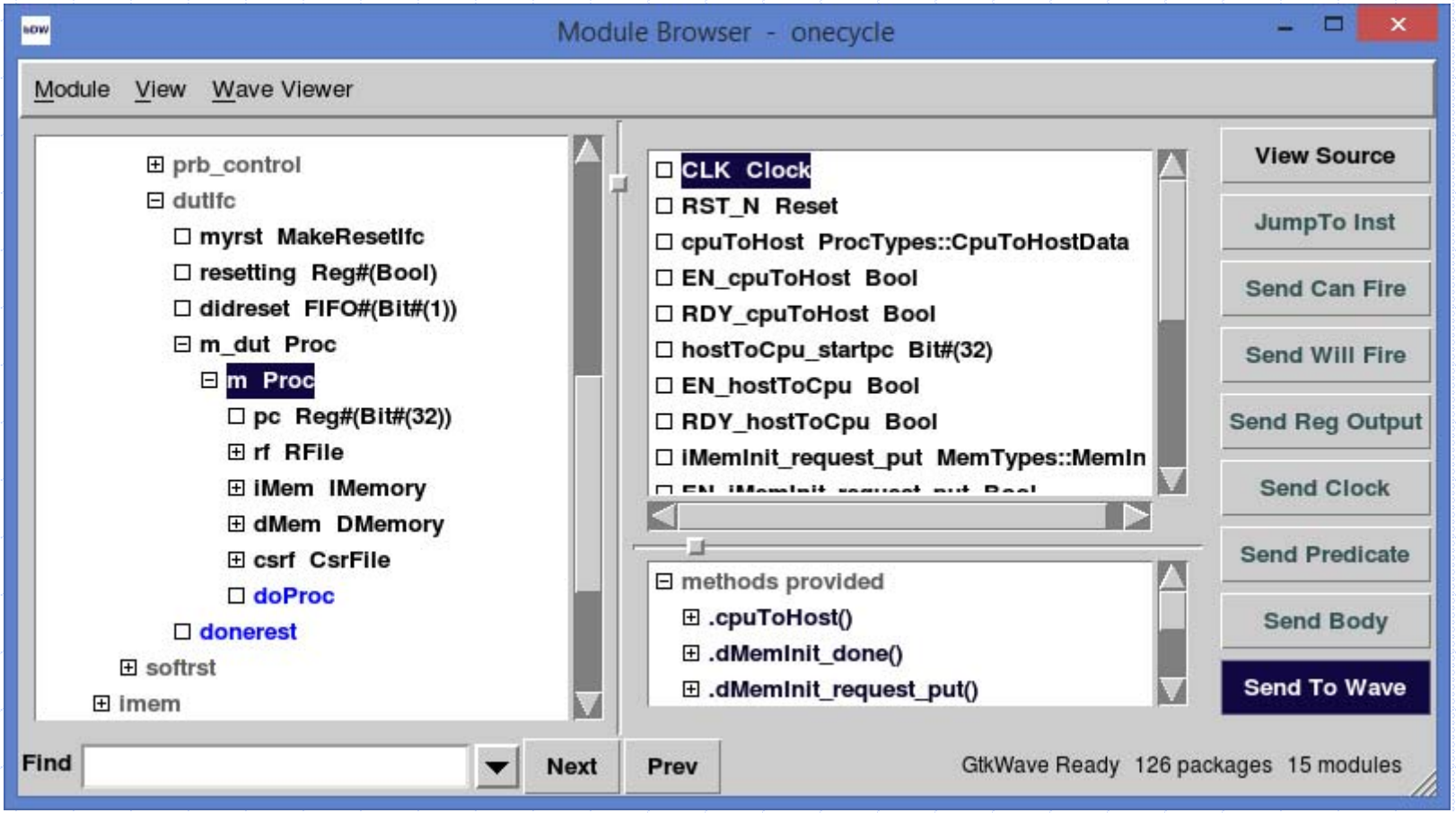

Load Dump File
Directory:
/afs/athena.mit.edu/user/s/z/szzhang/6.175/lab5/scemi/sim
bdir_dut
out.vcd
gtkwave
info_dut
simdir_dut
tbinclude
File name:
out.vcd
Open
Files of type:
All dump files (*.vcd,*.fsdb,*.wlf)
Cancel

Waveform Viewer



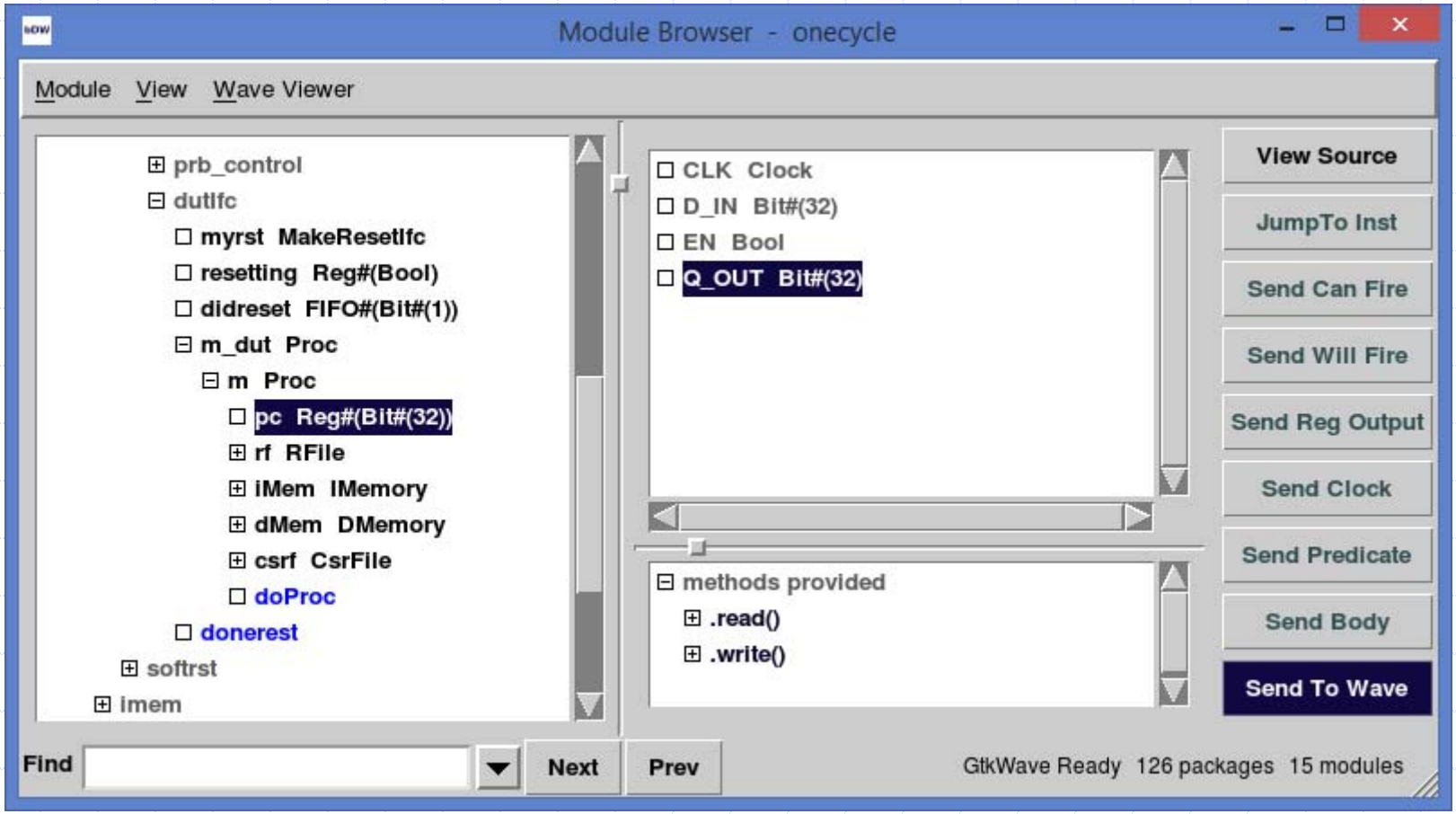
Add Signals to View

- Add clock



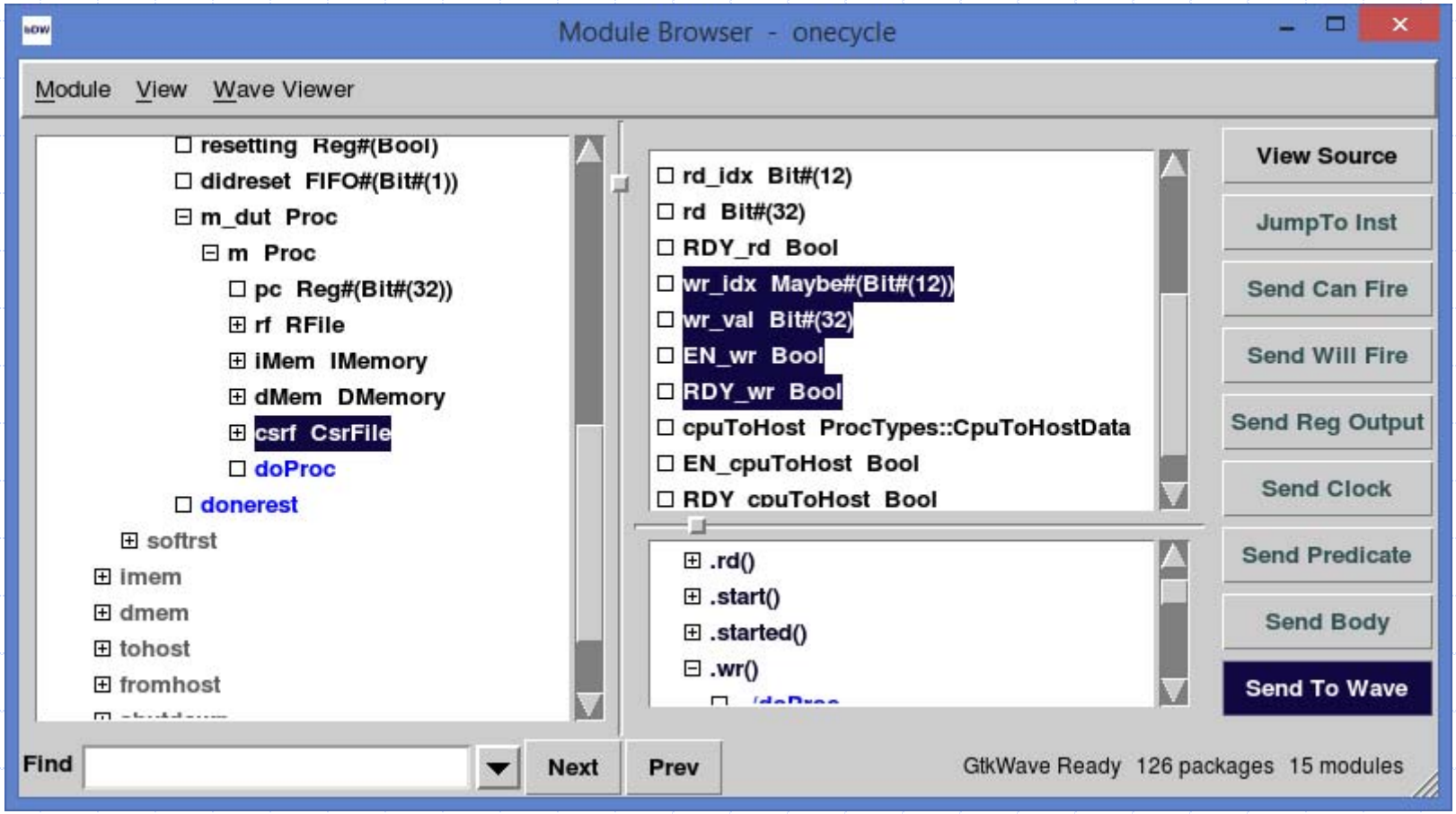
Add Signals to View

- Add PC



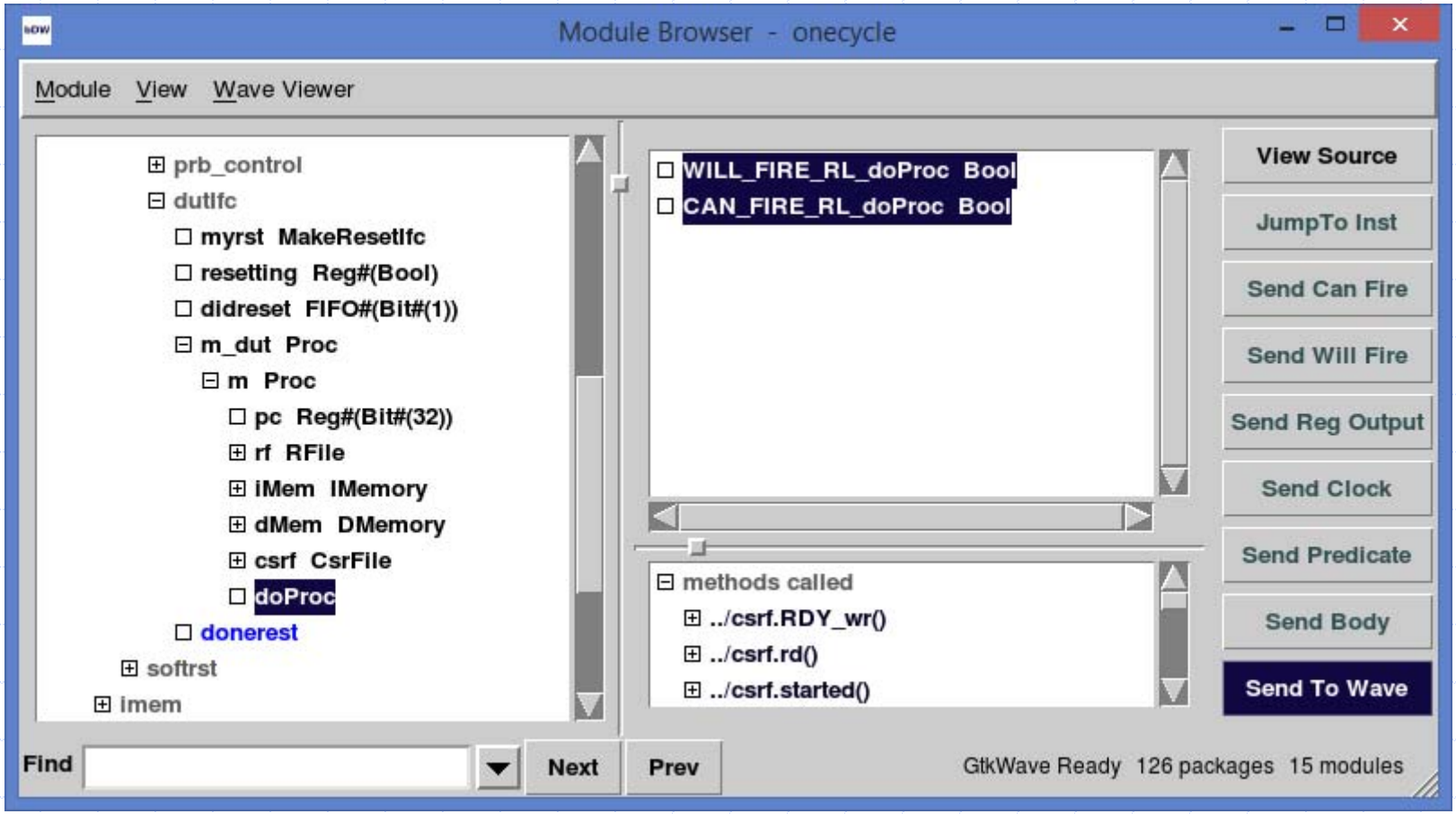
Add Signals to View

- Add csrf.wr method

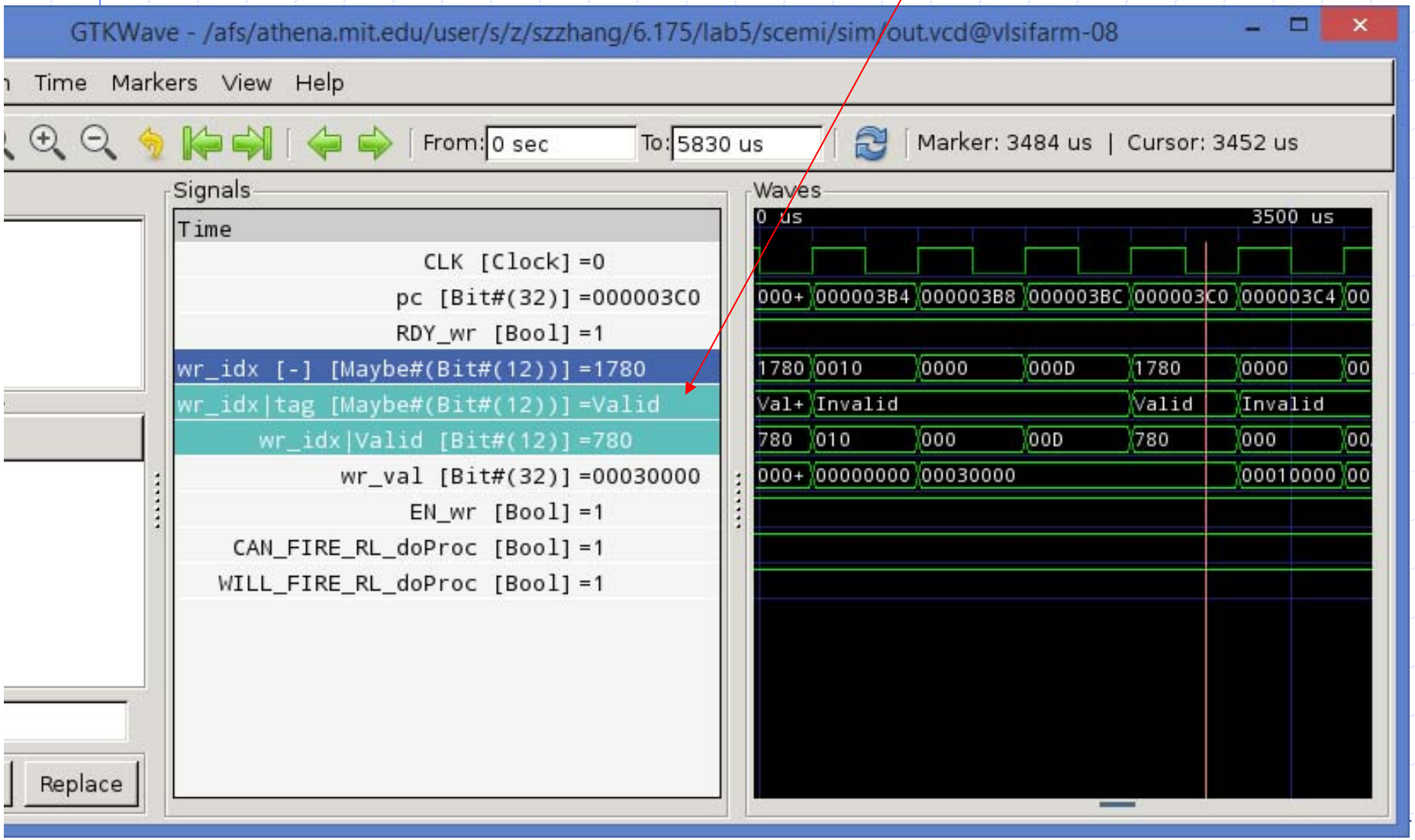


Add Signals to View

- Add can_fire, will_filre for rule doProc



Waveforms


Readable value
GTKWave - /afs/athena.mit.edu/user/s/z/szzhang/6.175/lab5/scemi/sim/out.vcd@vlsifarm-08
Time Markers View Help
From: 0 sec
To: 5830 us
Marker: 3484 us | Cursor: 3452 us
Signals
Waves
Time
CLK [Clock] =0
pc [Bit#(32)] =000003C0
RDY_wr [Bool] =1
wr_idx [-] [Maybe#(Bit#(12))] =1780
wr_idx|tag [Maybe#(Bit#(12))] =Valid
wr_idx|Valid [Bit#(12)] =780
wr_val [Bit#(32)] =00030000
EN_wr [Bool] =1
CAN_FIRE_RL_doProc [Bool] =1
WILL_FIRE_RL_doProc [Bool] =1
0 us
3500 us
Replace

Problem

- No internal signals
  - Optimized away during compilation
  - But I want to see them for dubuggin
    - For example: dInst

Solution -- Probe

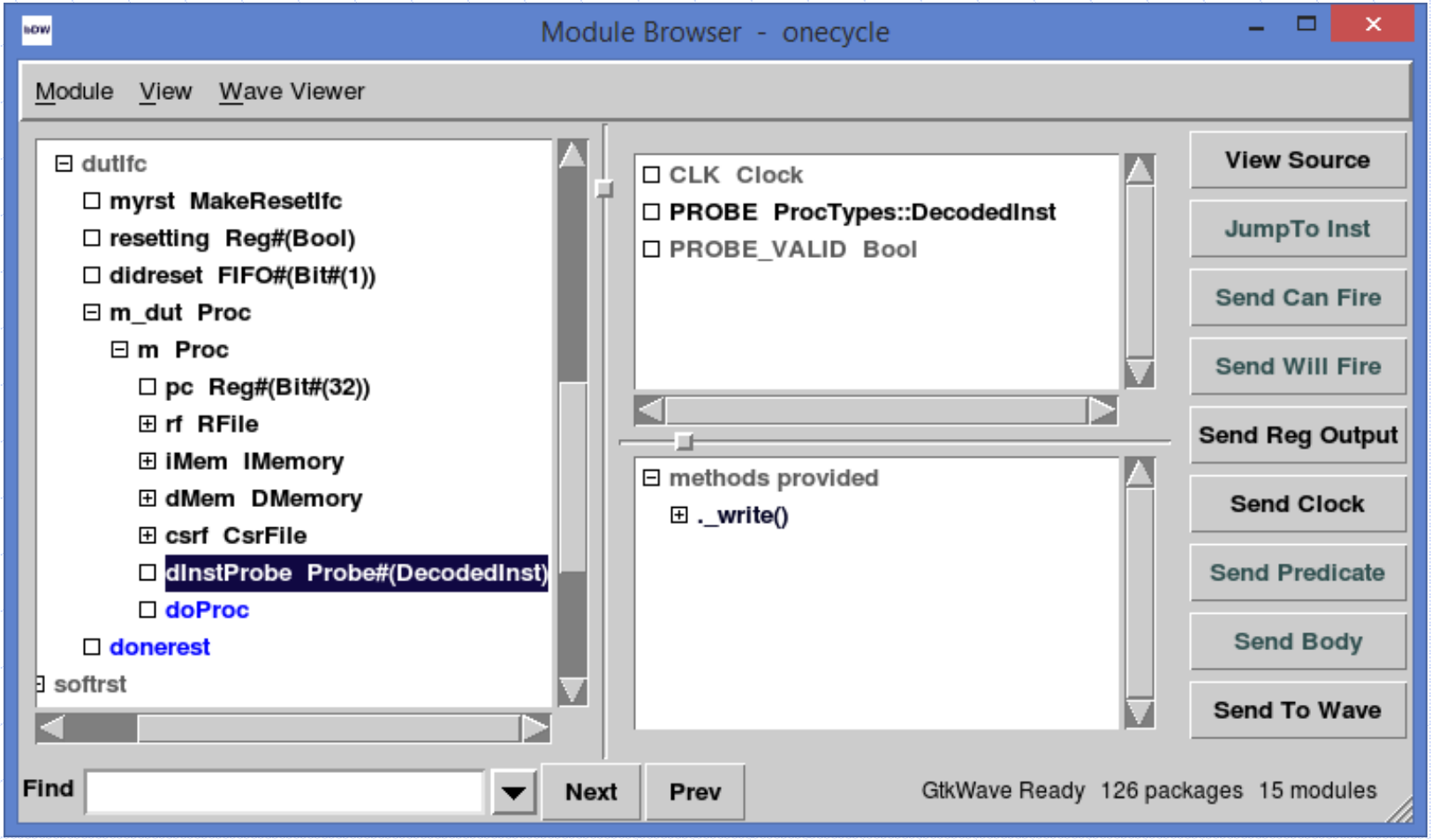
```
interface Probe#(type t);
    method Action _write(t x);
endinterface
module mkProbe(Probe#(t)) provisos(
    Bits#(t, sz));
```

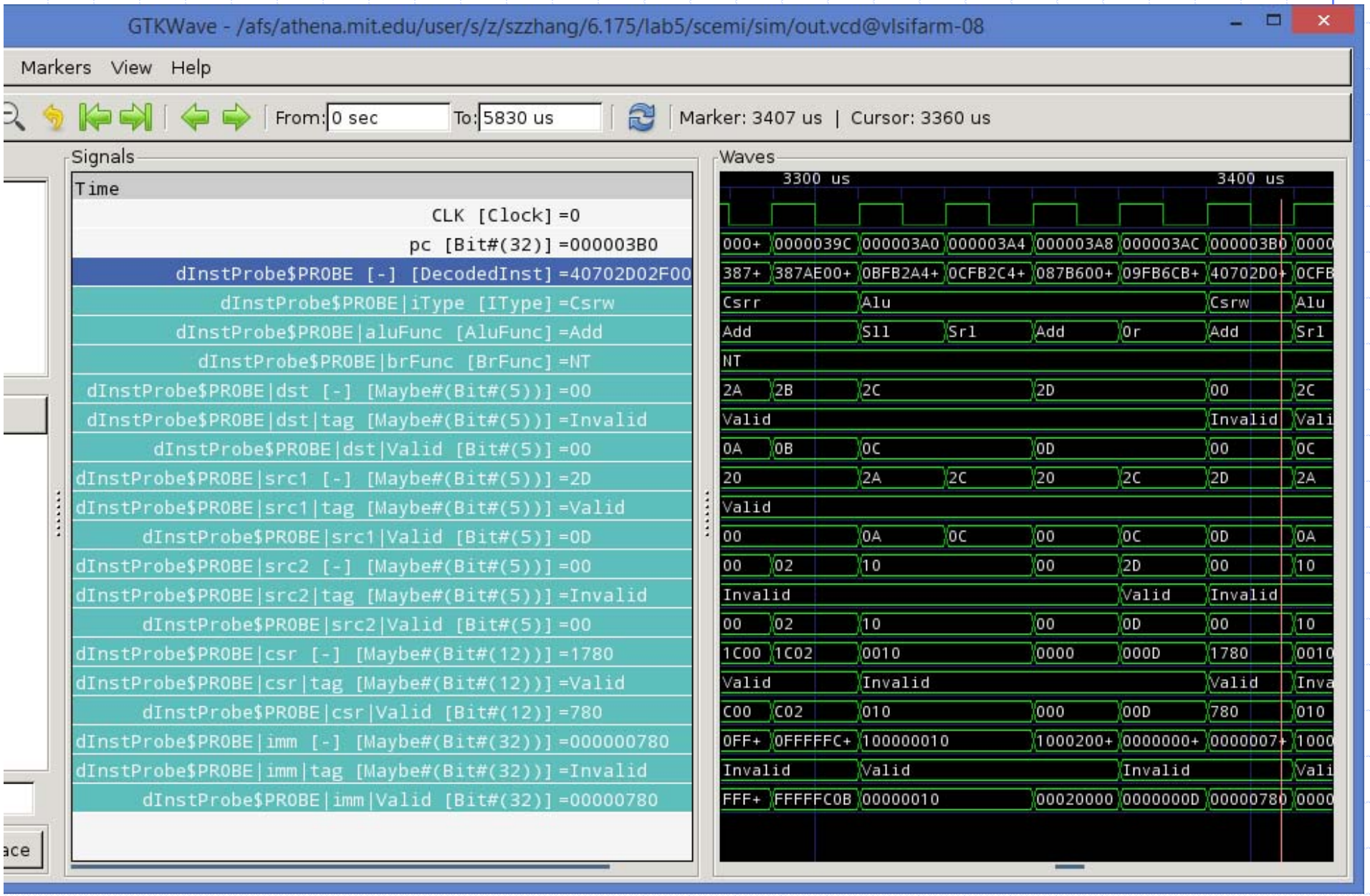
- Write to mkProbe will be recorded in VCD file
  - Instantiate mkProbe for each signal that we want to record

Probe Example

```
import Probe::*;
...
module mkProc(Proc);
    mkProbe#(DecodedInst) dInstProbe <- mkProbe;
    ...
    rule doProc;
        ...
        let dInst = decode(inst);
        dInstProbe <= dInst;
        ...
    endrule
    ...
endmodule
```

After Adding Probe



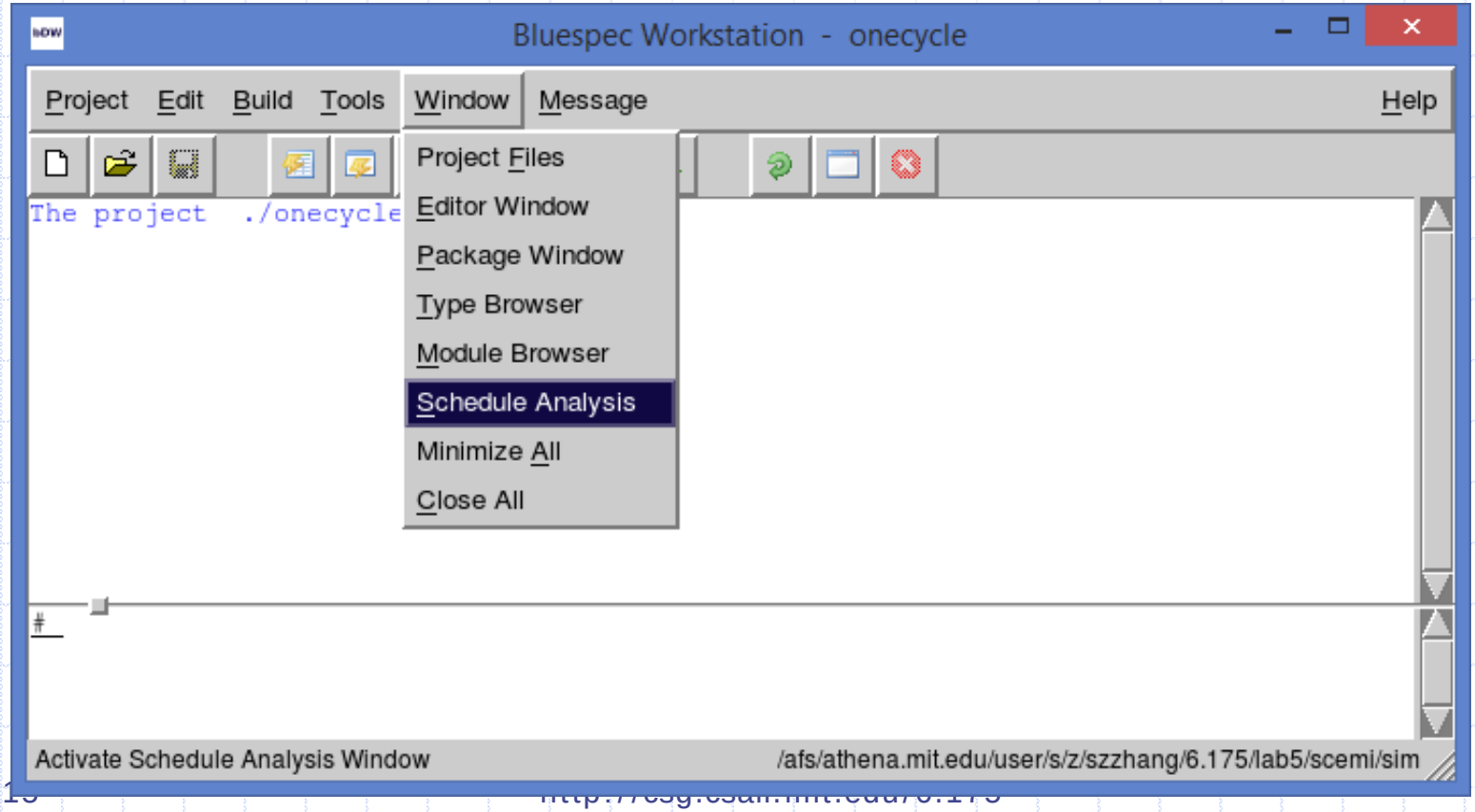

GTKWave - /afs/athena.mit.edu/user/s/z/szzhang/6.175/lab5/scemi/sim/out.vcd@vlsifarm-08
Markers View Help
From: 0 sec
To: 5830 us
Marker: 3407 us | Cursor: 3360 us
Signals
Waves
Time
CLK [Clock] =0
pc [Bit#(32)] =000003B0
dInstProbe$PROBE [-] [DecodedInst] =40702D02F00
dInstProbe$PROBE|iType [IType] =Csrw
dInstProbe$PROBE|aluFunc [AluFunc] =Add
dInstProbe$PROBE|brFunc [BrFunc] =NT
dInstProbe$PROBE|dst [-] [Maybe#(Bit#(5))] =00
dInstProbe$PROBE|dst|tag [Maybe#(Bit#(5))] =Invalid
dInstProbe$PROBE|dst|Valid [Bit#(5)] =00
dInstProbe$PROBE|src1 [-] [Maybe#(Bit#(5))] =2D
dInstProbe$PROBE|src1|tag [Maybe#(Bit#(5))] =Valid
dInstProbe$PROBE|src1|Valid [Bit#(5)] =0D
dInstProbe$PROBE|src2 [-] [Maybe#(Bit#(5))] =00
dInstProbe$PROBE|src2|tag [Maybe#(Bit#(5))] =Invalid
dInstProbe$PROBE|src2|Valid [Bit#(5)] =00
dInstProbe$PROBE|csr [-] [Maybe#(Bit#(12))] =1780
dInstProbe$PROBE|csr|tag [Maybe#(Bit#(12))] =Valid
dInstProbe$PROBE|csr|Valid [Bit#(12)] =780
dInstProbe$PROBE|imm [-] [Maybe#(Bit#(32))] =000000780
dInstProbe$PROBE|imm|tag [Maybe#(Bit#(32))] =Invalid
dInstProbe$PROBE|imm|Valid [Bit#(32)] =00000780
3300 us
3400 us

Combine with Assembly

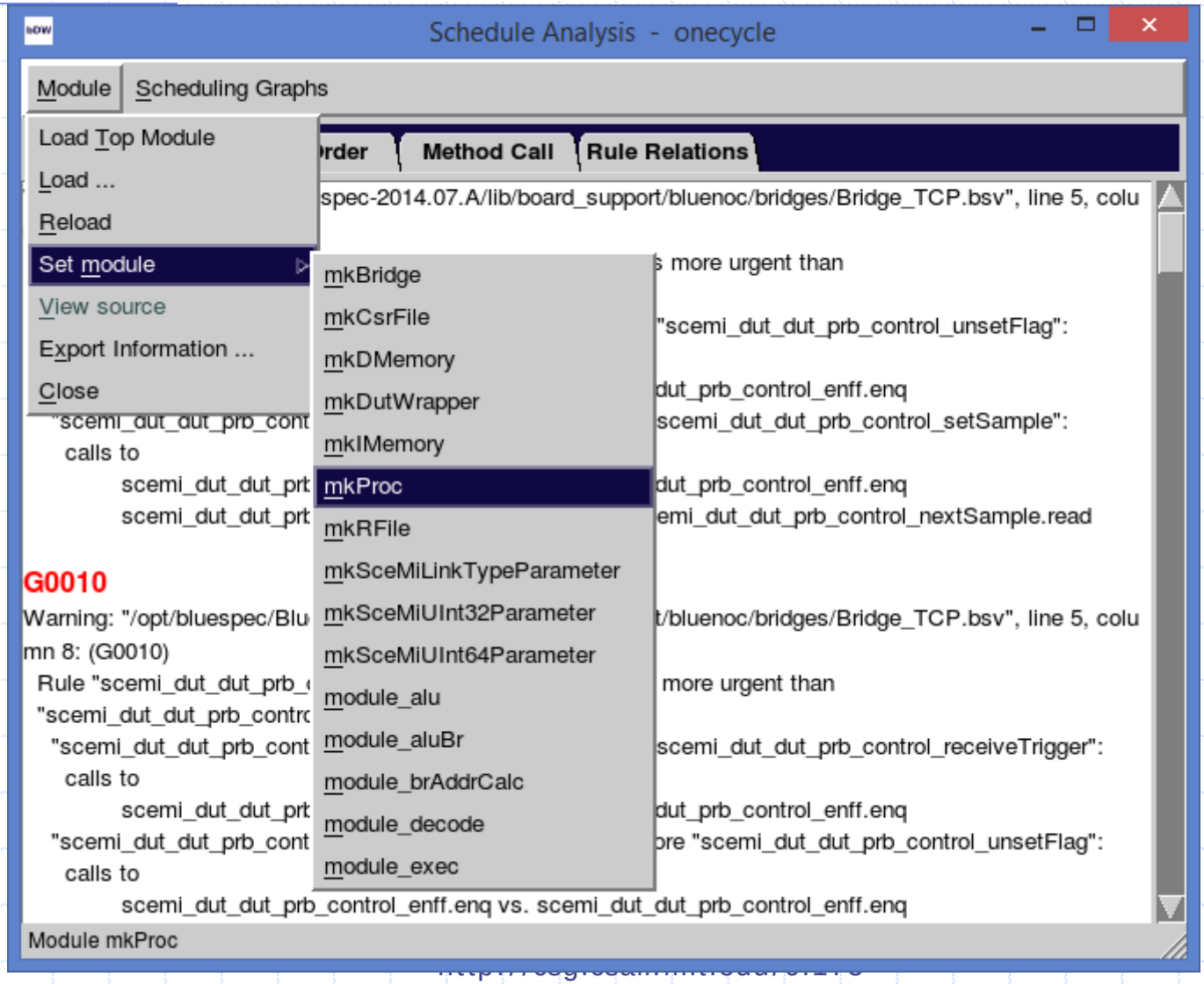
- Assembly code of compiled tests are in
  - programs/build/*/dump
- Refer to the assembly code in debugging

Debugging Scheduling

- GUI: schedule Analysis
  - The same as “-show-shedule” BSC flag

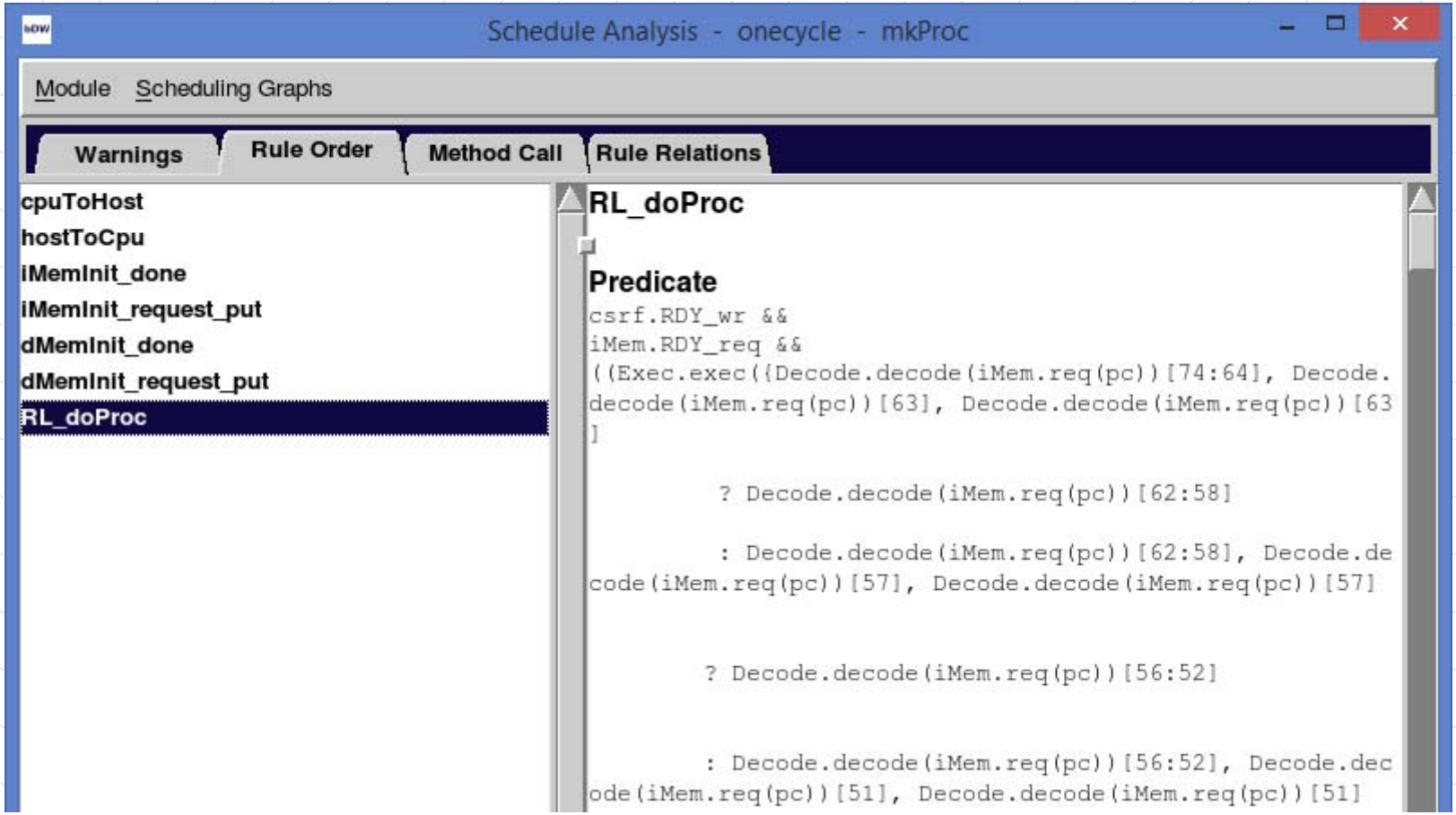


Select mkProc



Output of Schedule Analysis

- Rule order: execution order of rules and methods
  - From top to bottom



Alternative for Schedule Analysis

- Add –show-schedule to project.bld

```
[DEFAULT]
default-targets:     all
bsc-compile-options: -aggressive-conditions -keep-fires -show-schedule
bsc-link-options:    -Xc++ -O0 -keep-fires
```

  - Generate info_dir/*.sched
  - Contain same information as GUI outputs

General Debugging Advice

- Debugging is to guess what goes wrong
- Check warnings in compiler outputs
  - Scheduling warnings
  - Understand all warnings in mkProc
- Stare at your code – prove its correctness
- Add $display – convenient & very useful
- Look at assembly code, locate position
- Dump VCD file and see waveform

Note

- Codes on the slide is a little different from our labs
- Read src/includes to be familiar with out lab setup.