Tools Tutorials
Part A

Yunfei Gu
Washington University in St. Louis

Outline

RISC-V Z-scale Architecture

AHB-Lite protocol

Synopsys VCS

RISC-V Z-scale

- What is RISC-V Z-scale?

Z-scale is a tiny 32-bit RISC-V core generator suited for microcontrollers and embedded systems

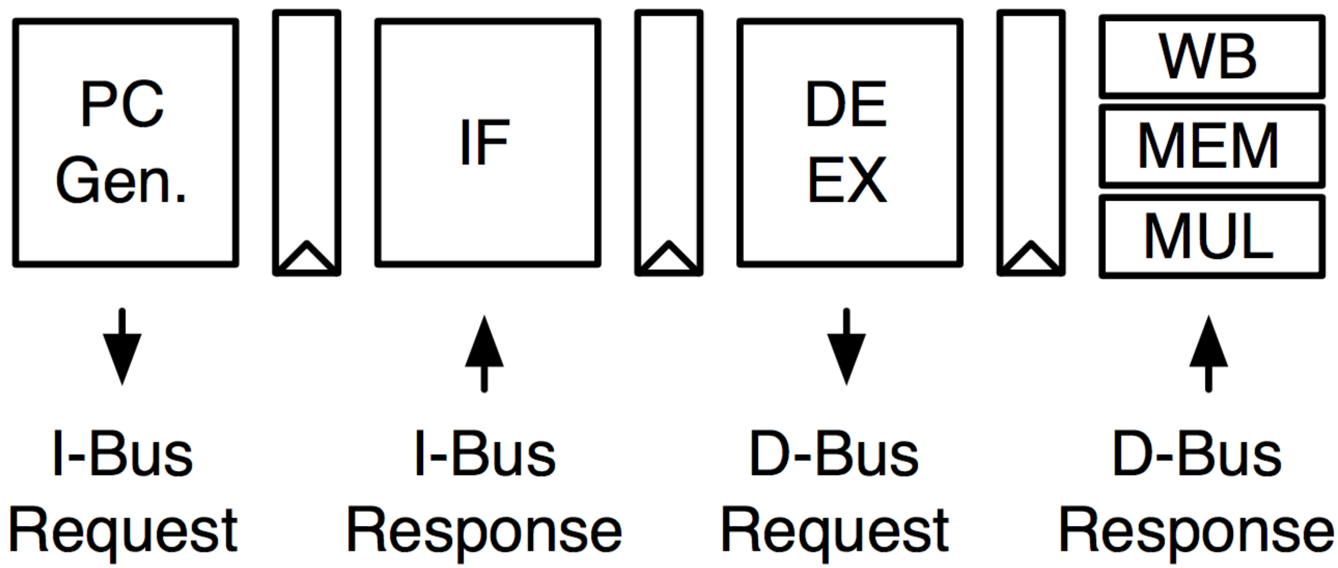
Z-scale is designed to talk to AHB-Lite buses

- plug-compatible with ARM Cortex-M series

Z-scale generator also generates the interconnect between core and devices

- Includes buses, slave muxes, and crossbars

Z-scale Pipelined



- 32-bit 3-stage single-issue in-order pipe
- Executes RV32IM ISA, has M/U privilege modes
- I-bus and D-bus are AHB-Lite and 32-bits wide
- Interrupts are supported

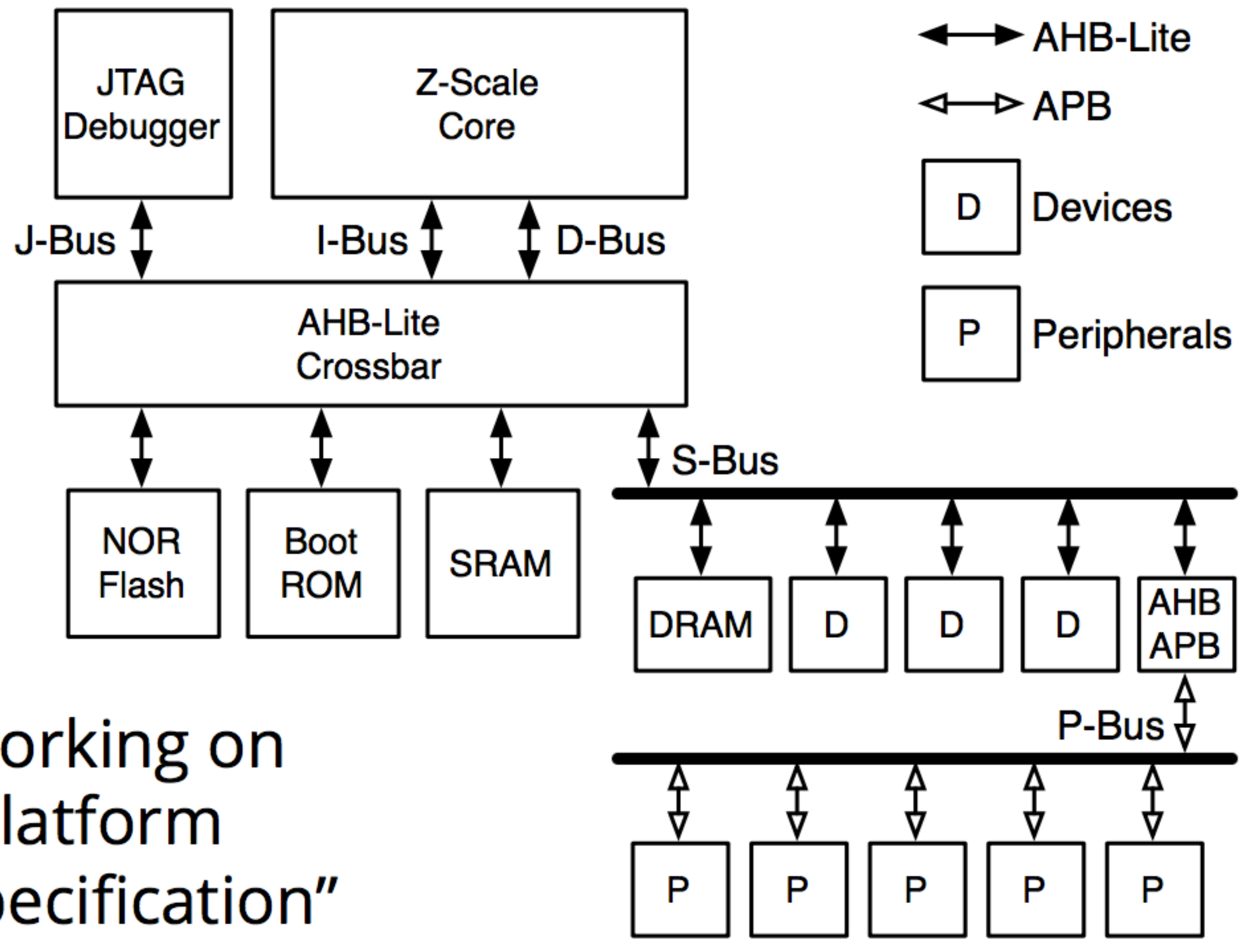
Z-scale

- ARM Cortex-M0 vs. Z-scale

Category	ARM Cortex-M0	RISC-V Zscale
ISA	32-bit ARM v6	32-bit RISC-V (RV32IM)
Architecture	Single-Issue In-Order 3-stage	Single-Issue In-Order 3-stage
Performance	0.87 DMIPS/MHz	1.35 DMIPS/MHz
Process	TSMC 40LP	TSMC 40GPLUS
Area w/o Caches	0.0070 mm^2	0.0098 mm^2
Area Efficiency	124 DMIPS/MHz/mm^2	138 DMIPS/MHz/mm^2
Frequency	≤50 MHz	~500 MHz
Voltage (RTV)	1.1 V	0.99 V
Dynamic Power	5.1 µW/MHz	1.8 µW/MHz

Z-scale

- Building a Z-scale System



- Working on "platform specification"

Z-scale

- Z-scale use cases

    Microcontrollers

        -Implement your simple control loops

        -If code density matters

    Embedded Systems

        -Build your system around Z-scale

    Validation of Tiny 32-bit RISC-V Systems

    Verilog versions of Z-scale is open-sourced under the BSD license

        https://github.com/ucb-bar/zscale

        https://github.com/ucb-bar/fpga-spartan6

Outline

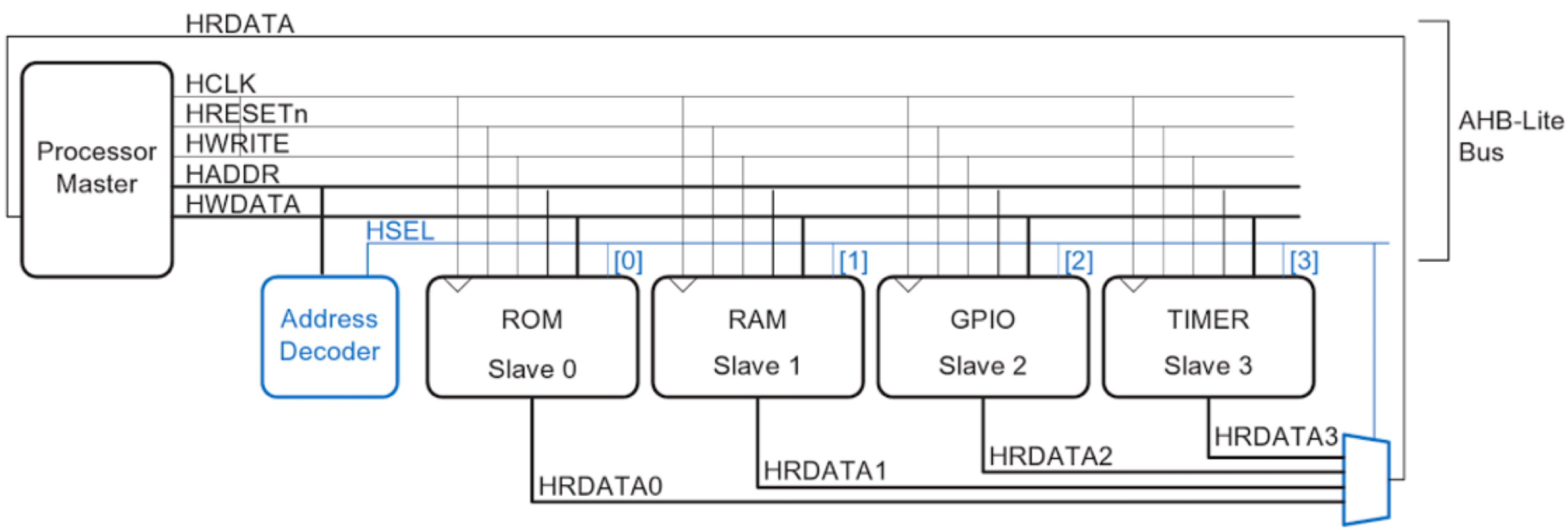
RISC-V Z-scale Architecture

AHB-Lite protocol

Synopsys VCS

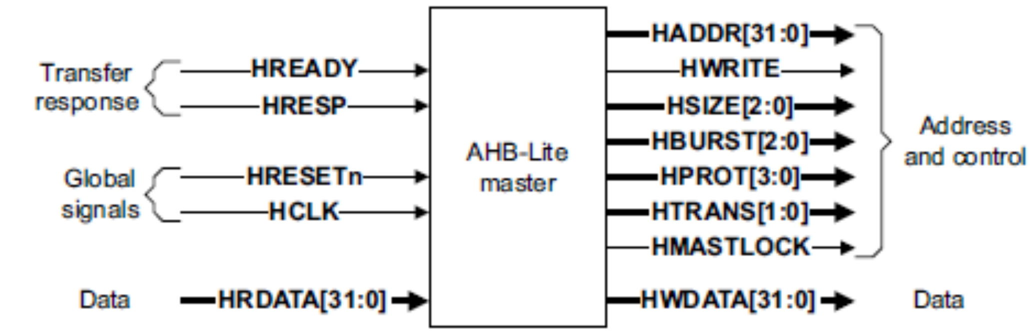
AHB-Lite System

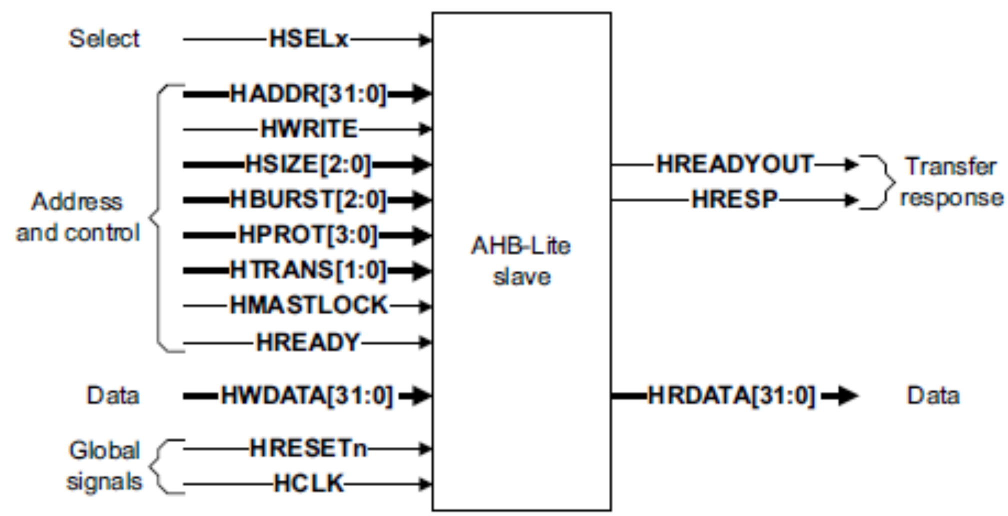
- components of AHB-Lite system
  - Master
  - Slaves
  - Address Decoder and
  - Multiplexer



AHB-Lite bus Master/Slave interface

- Global signals
  - HCLK
  - HRESETn
- Master out/slave in
  - HADDR (address)
  - HWDATA (write data)
  - Control
    - HWRITE
    - HSIZE
    - HBURST
    - HPROT
    - HTRANS
    - HMASTLOCK
- Slave out/master in
  - HRDATA (read data)
  - HREADY
  - HRESP





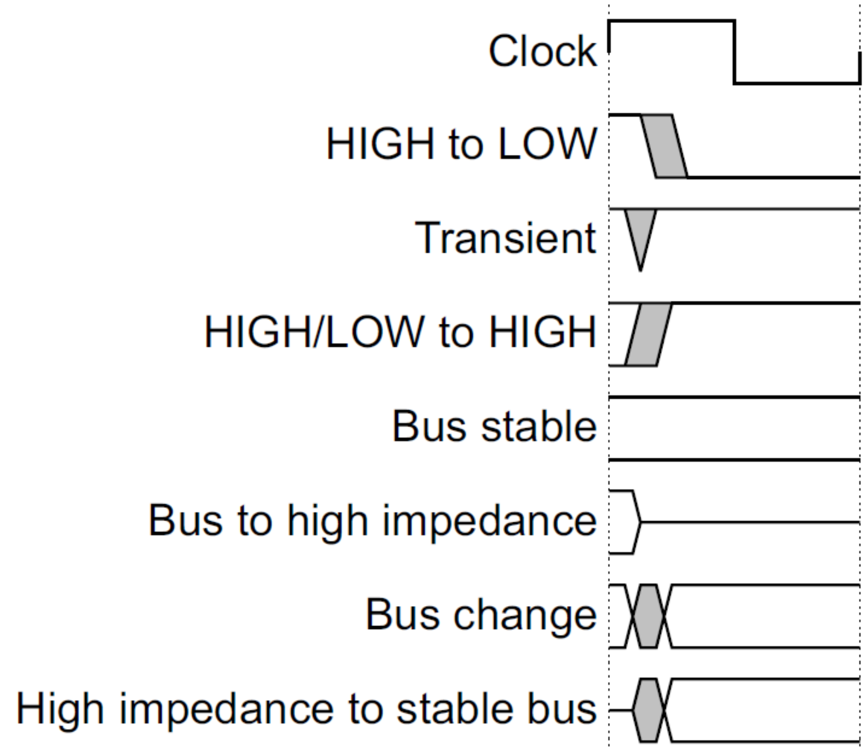
AHB-Lite signal definitions

- Global signals
  - HCLK: the bus clock source (rising-edge triggered)
  - HRESETn: the bus (and system) reset signal (active low)
- Master out/slave in
  - HADDR[31:0]: the 32-bit system address bus
  - HWDATA[31:0]: the system write data bus
  - Control
    - HWRITE: indicates transfer direction (Write=1, Read=0)
    - HSIZE[2:0]: indicates size of transfer (byte, halfword, or word)
    - HBURST[2:0]: burst transfer size/order (1, 4, 8, 16 beats or undefined)
    - HPROT[3:0]: provides protection information (e.g. I or D; user or handler)
    - HTRANS: indicates current transfer type (e.g. idle, busy, nonseq, seq)
    - HMASTLOCK: indicates a locked (atomic) transfer sequence
- Slave out/master in
  - HRDATA[31:0]: the slave read data bus
  - HREADY: indicates previous transfer is complete
  - HRESP: the transfer response (OKAY=0, ERROR=1)

Key to timing diagram conventions

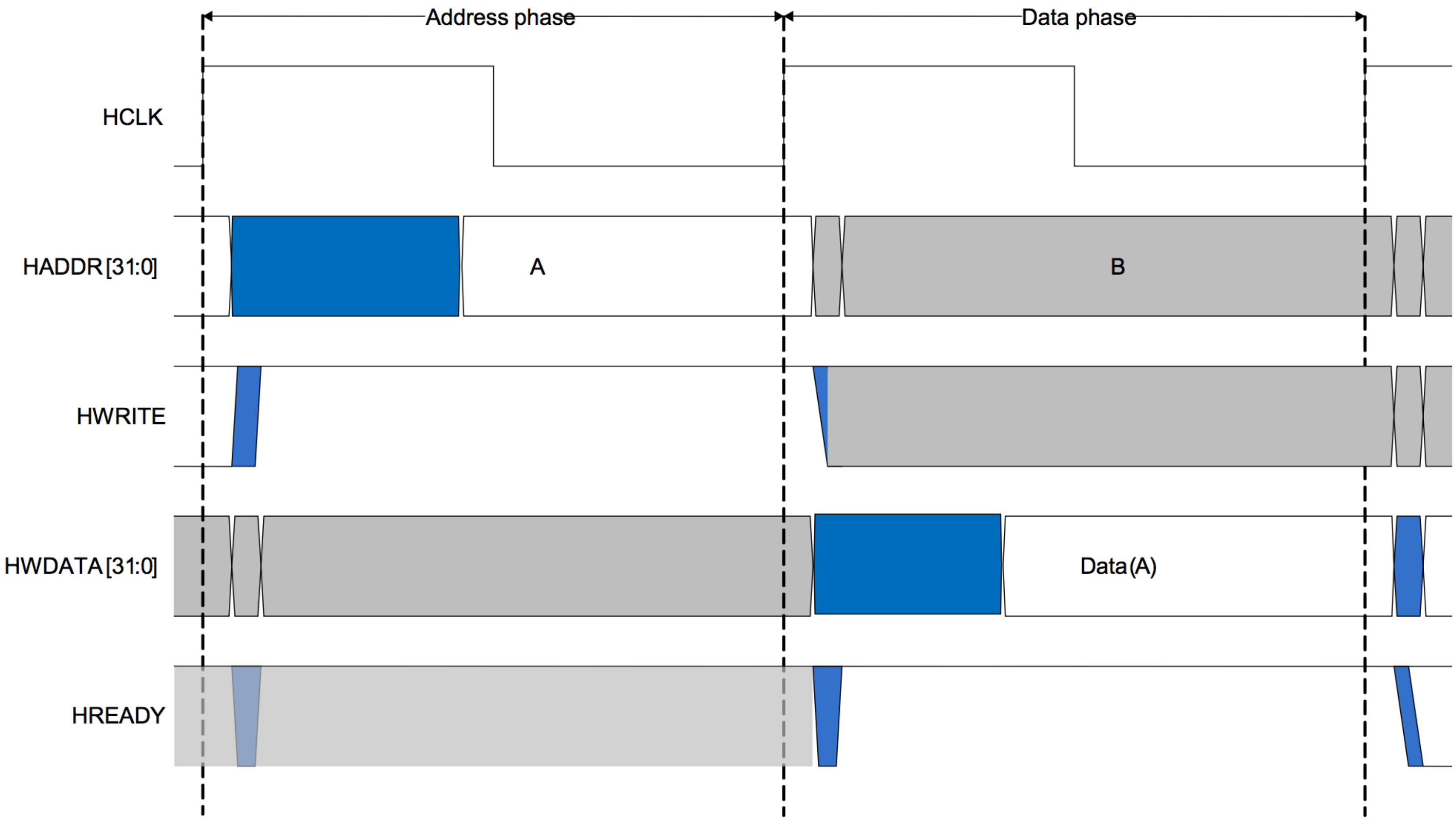
- Timing diagrams
  - Clock
  - Stable values
  - Transitions
  - High-impedance

- Signal conventions
  - Lower case ‘n’ denote active low (e.g. RESETn)
  - Prefix ‘H’ denotes AHB
  - Prefix ‘P’ denotes APB



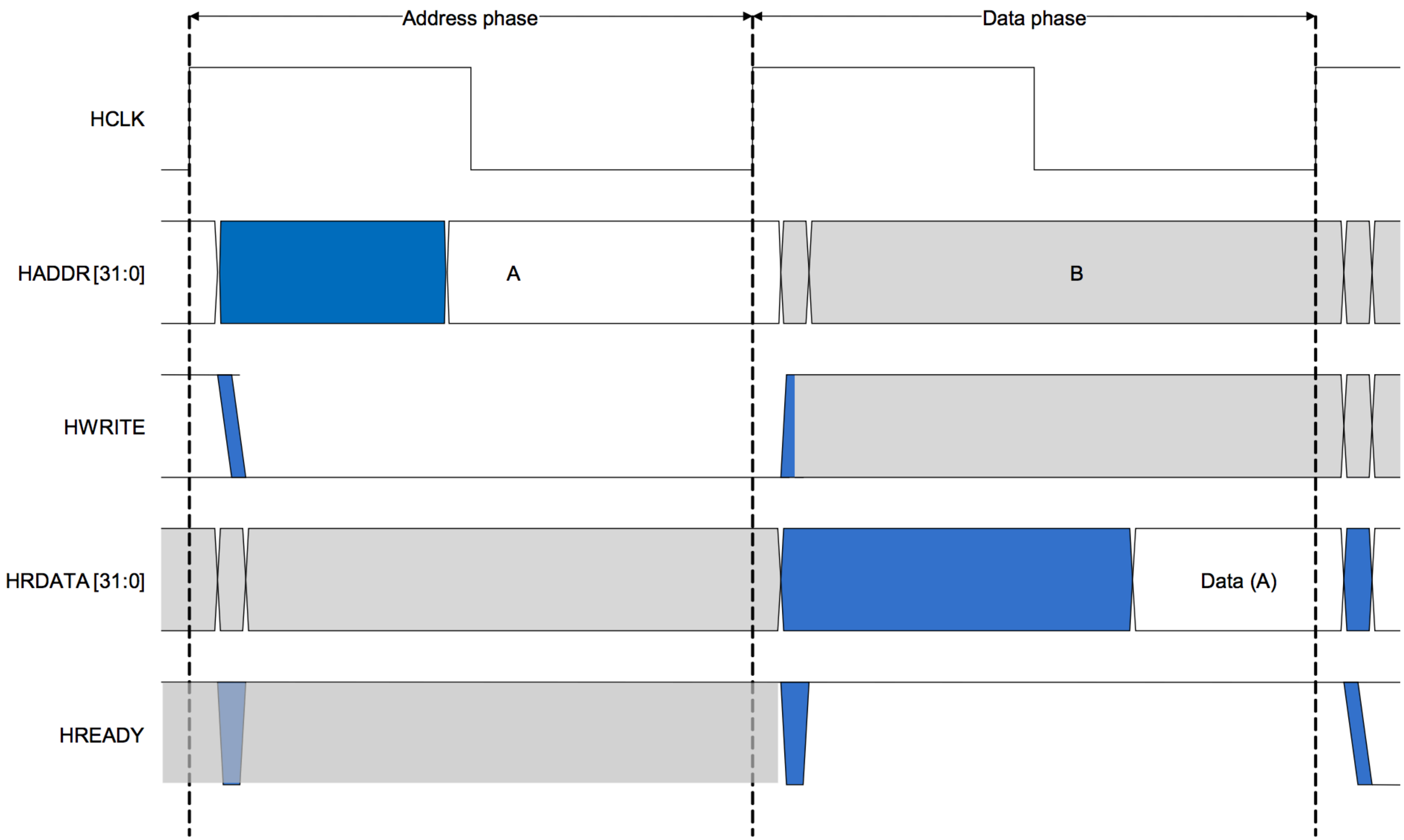
AHB-Lite signal

- Basic transfer - write



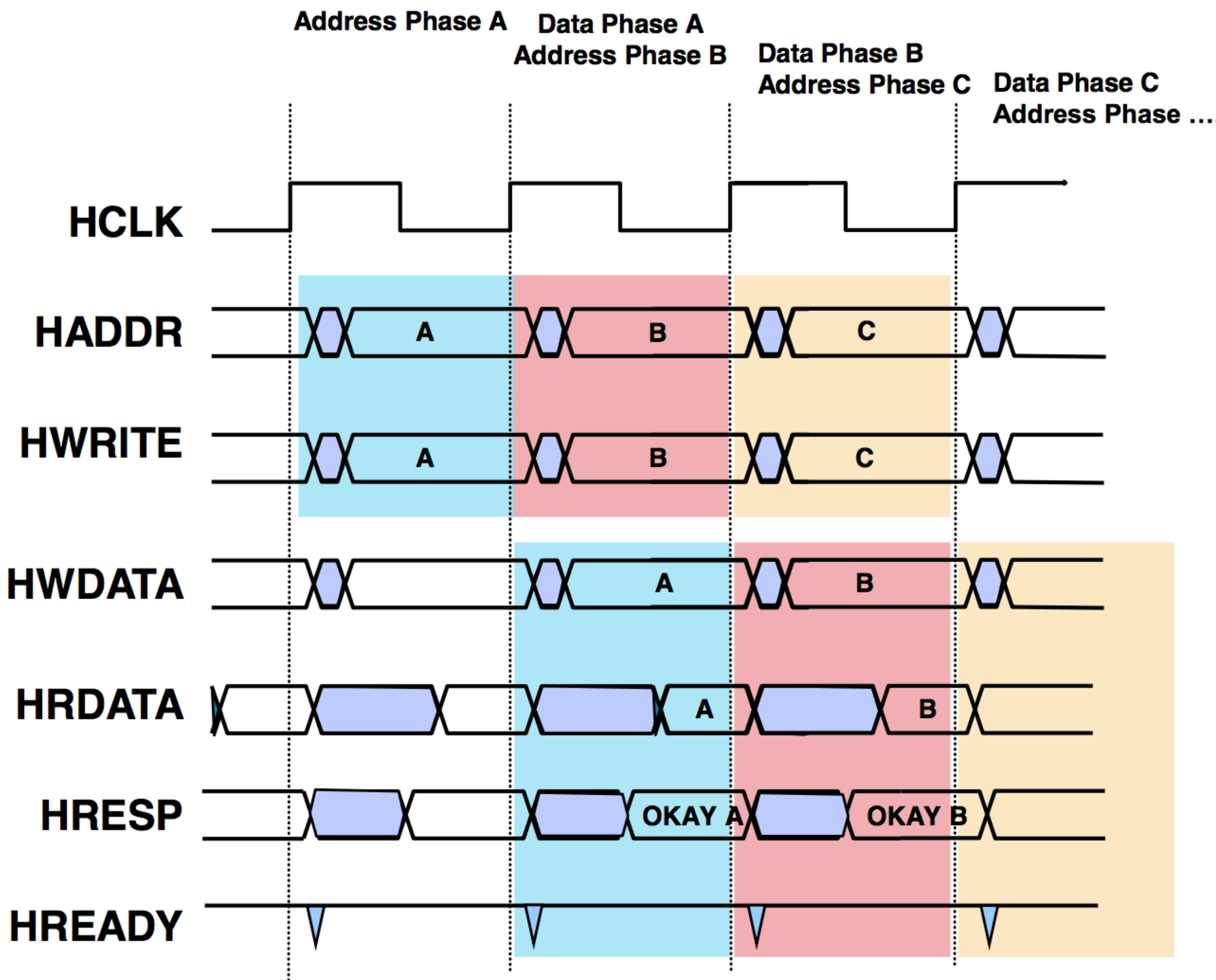
AHB-Lite signal

- Basic transfer - read



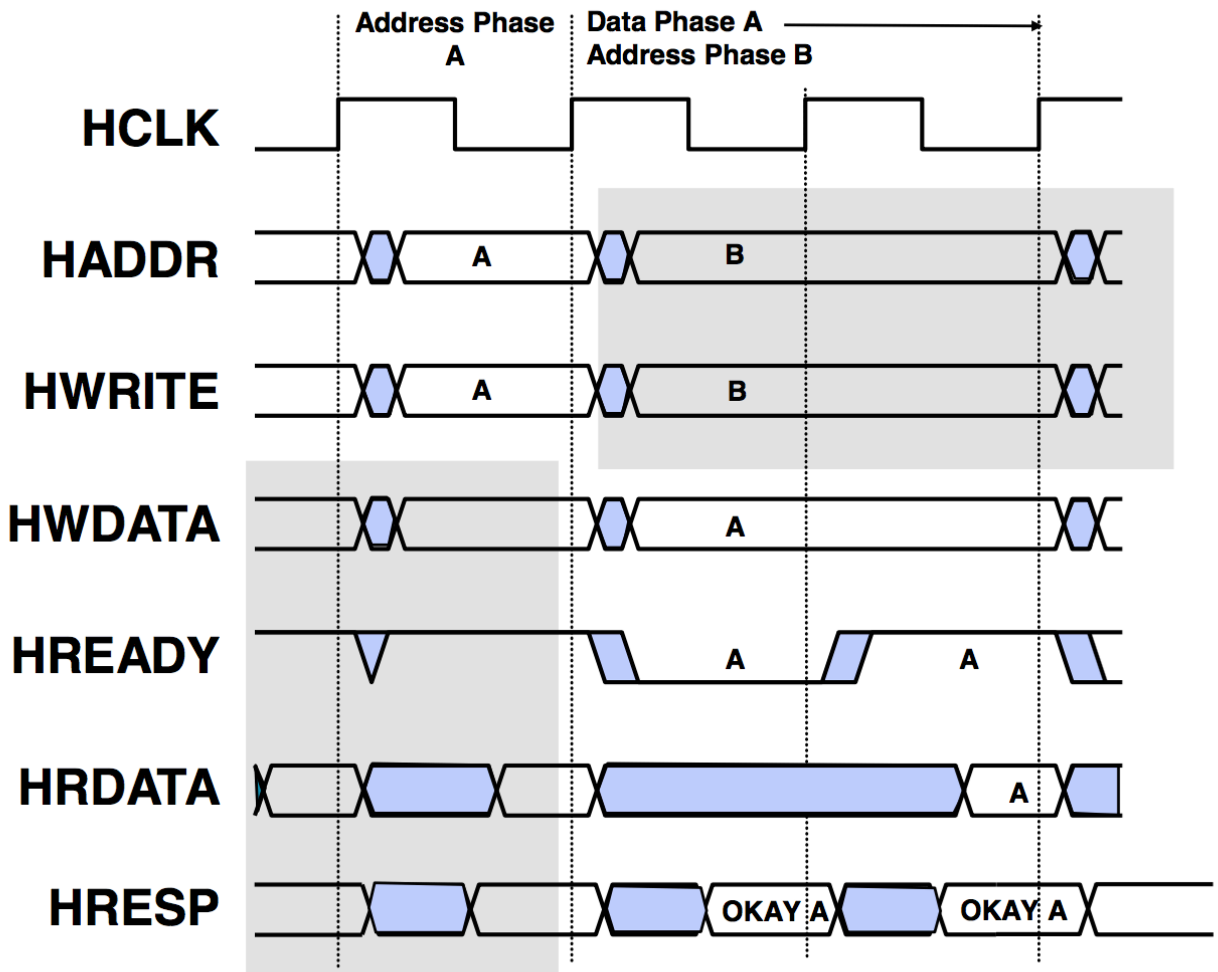
AHB-Lite signal

- AHB Pipelined transaction



AHB-Lite signal

- Adding wait states



Master will extend Address Phase B

Outline

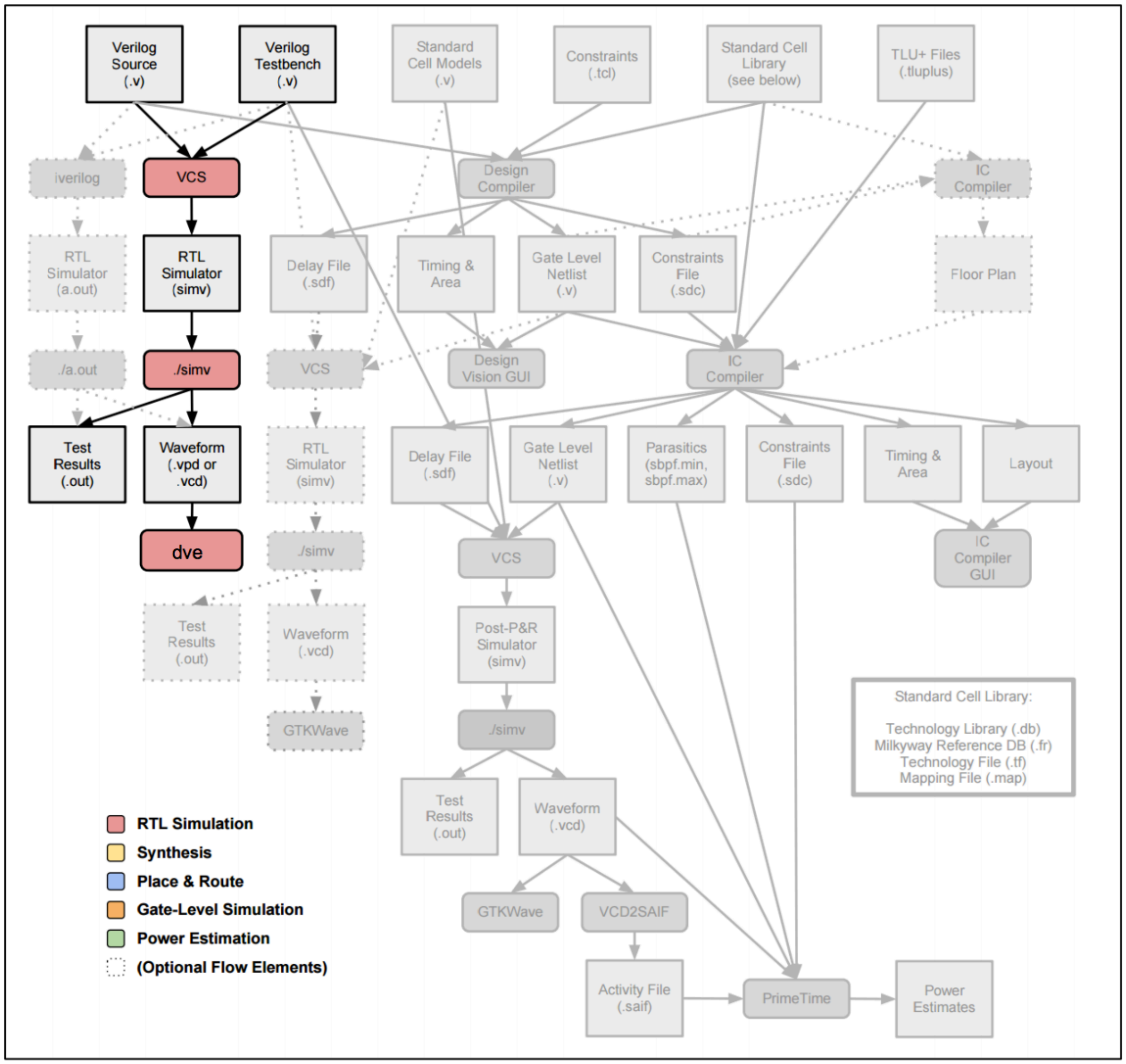
RISC-V Z-scale Architecture

AHB-Lite protocol

Synopsys VCS

VCS

- What is Synopsys VCS?



VCS

- Compile your code

terminal command line:

% vcs -full64 -PP +lint=all,noVCDE +v2k -timescale=1ns/10ps <file>.v <file_tb>.v

```
  _csrc0.so     rmapats_mop.o rmapats.o rmar.o  rmar_llvm_0_1.o rmar_llvm_0_0.o
       /project/linuxlab/synopsys/vcs_mx/amd64/lib/libzerosoft_rt_stubs.so /pr
oject/linuxlab/synopsys/vcs_mx/amd64/lib/libvirsim.so /project/linuxlab/synopsys
/vcs_mx/amd64/lib/liberrorinf.so /project/linuxlab/synopsys/vcs_mx/amd64/lib/lib
snpsmalloc.so    /project/linuxlab/synopsys/vcs_mx/amd64/lib/libvcsnew.so /proje
ct/linuxlab/synopsys/vcs_mx/amd64/lib/libuclinative.so   -Wl,-whole-archive /pro
ject/linuxlab/synopsys/vcs_mx/amd64/lib/libvcsucli.so -Wl,-no-whole-archive
     /project/linuxlab/synopsys/vcs_mx/amd64/lib/vcs_save_restore_new.o -ldl -lm
  -lc -lpthread -ldl
../simv up to date
CPU time: .238 seconds to compile + .294 seconds to elab + .341 seconds to link
[dengxue.yan@linuxlab006 VcsTutorial]$
```

Successfully compiled

```
[dengxue.yan@linuxlab006 VcsTutorial]$ vcs -full64 -PP +lint=all,noVCDE +v2k -ti
mescale=1ns/10ps Counter.v Counter_tb.v
                         Chronologic VCS (TM)
      Version J-2014.12-SP1-1_Full64 -- Tue Jan 24 15:34:39 2017
               Copyright (c) 1991-2014 by Synopsys Inc.

This program is proprietary and confidential information of Synopsys Inc.
and may be used and disclosed only as authorized in a license agreement
controlling such use and disclosure.

The design hasn't changed and need not be recompiled.
If you really want to, delete file simv.daidir/.vcs.timestamp and
run VCS again.

[dengxue.yan@linuxlab006 VcsTutorial]$
```

Don't need to recompile because nothing changed

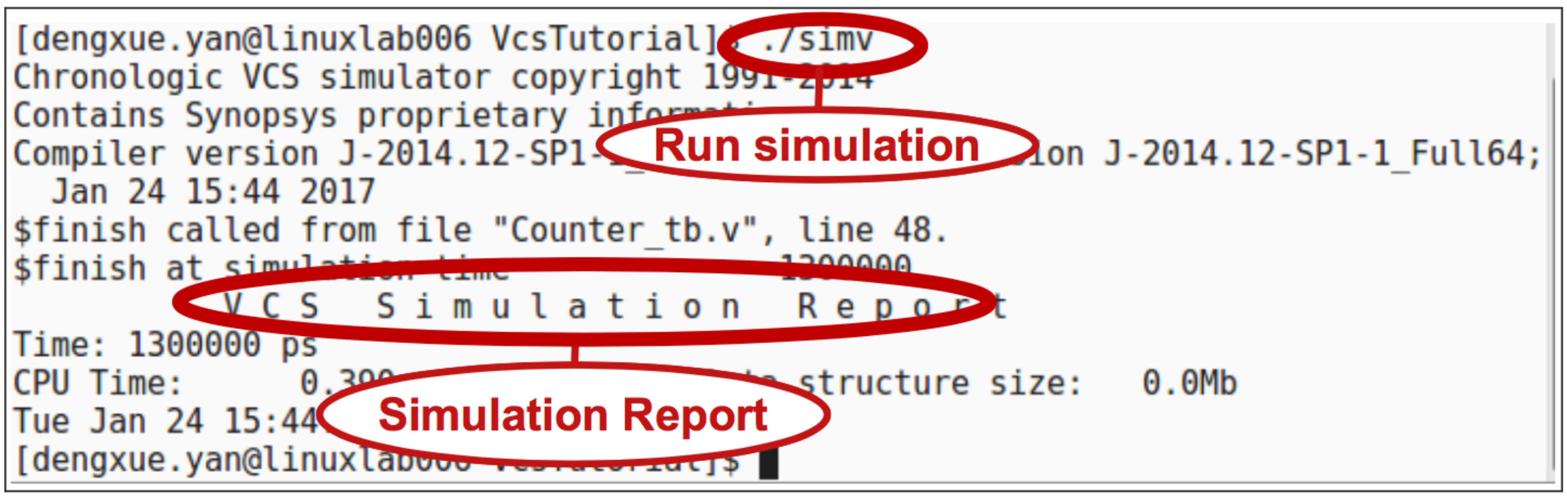
VCS

- simulation your code

A successfully compiling will print out on terminal "../simv up to date". And it should generate an executable file named "simv" in the same folder where your codes are present. Then in the terminal run:

terminal command line:

% ./sim

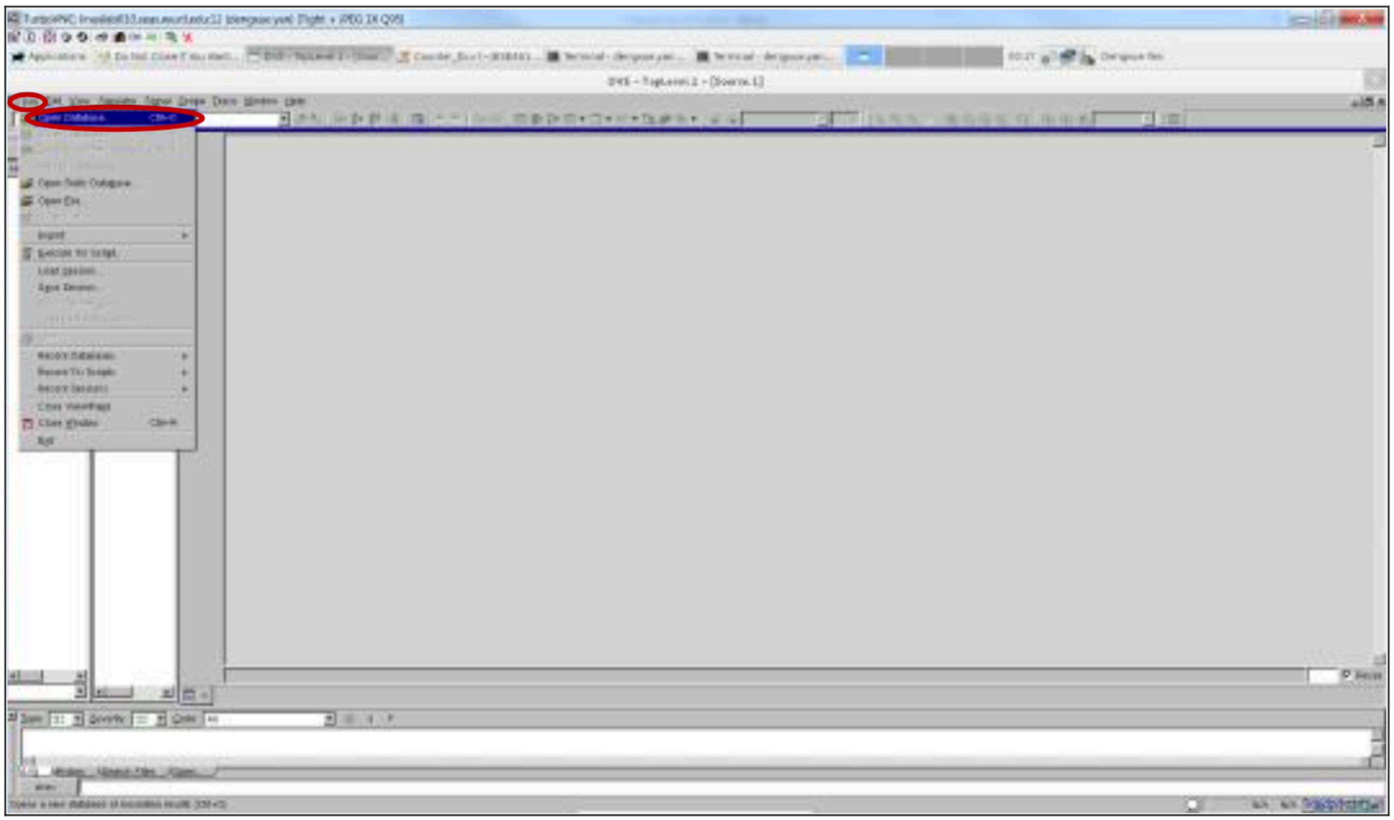


VCS

- View trace output with dve

After simulation report and “<file>.vcd” is generated, now type the following command in the terminal:

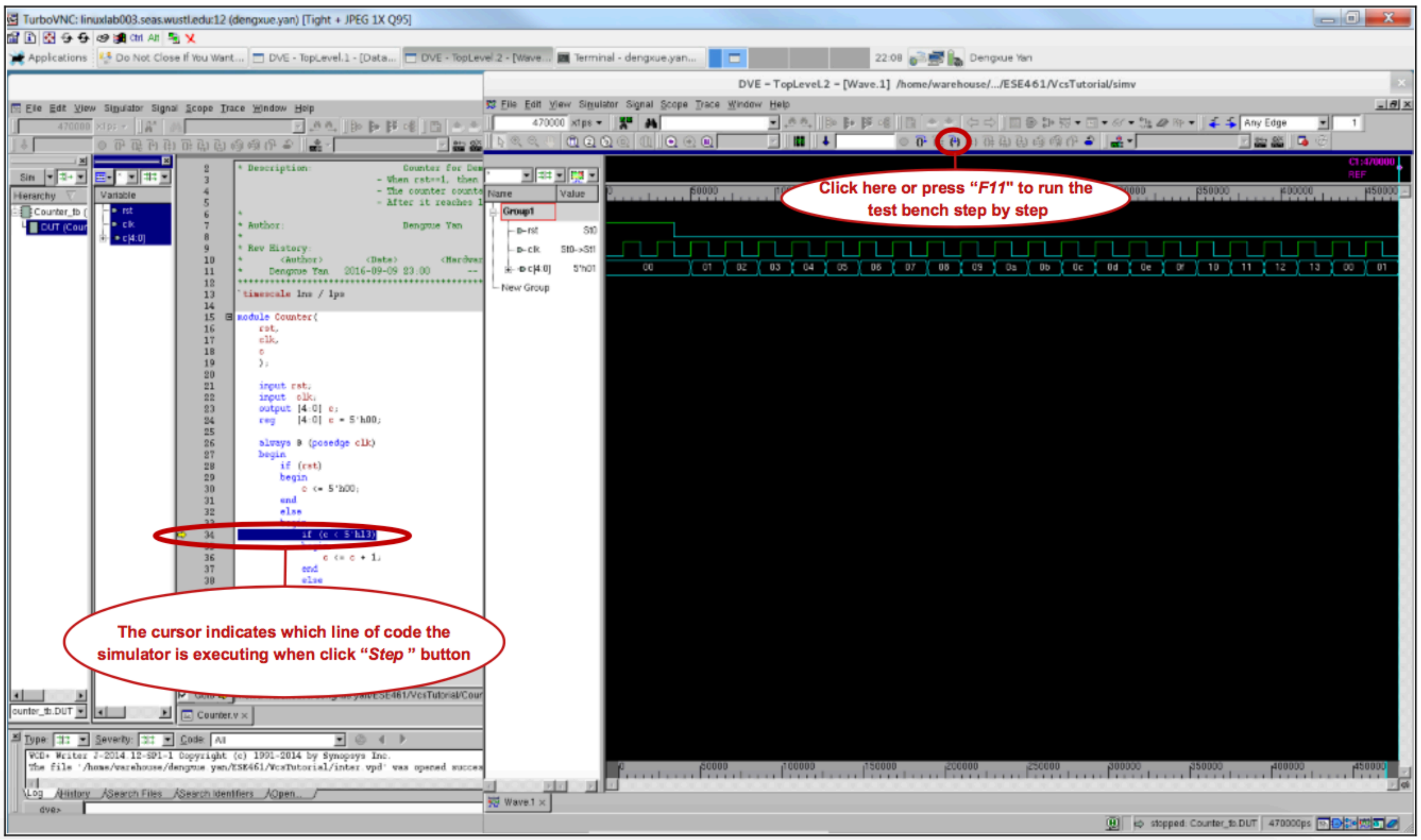
% dve
This is a viewer to plot and verify your results.



VCS

- View trace output with dve

Go to “File->Open Database” and select the “*.vcd*” file from the project folder. Then you will find the name of your test bench model in the Hierarchy box (Counter_tb here). Expand it so that you can find *DUT* in the options.If you click on *DUT*, select the signals listed(all or partial) and right click, you will find an option “Add to Waves”. Also it is easy to debug as shown below:



Acknowledgement

https://riscv.org/wp-content/uploads/2015/06/riscv-zscale-workshop-june2015.pdf

CS250 VLSI Systems Design (2009-2011) - University of California at Berkeley

The architecture for digital world: ARM