ECE 566A Modern System-on-Chip Design, Spring 2017
Lab 1: Go through ASIC design flow

1. Overview

Lab1 is a tool practice lab in which you will do several works to be familiar with ASIC design flow. In this lab, you need to know how to use ASIC design flow tools including Synopsys VCS, Design Compiler, and Cadence Encounter by following the tutorials that posted on our course website. And you should use these tools to go through ASIC design flow on an open source core named RISC-V v-scale core.

2. Introduction

2.1 What is ASIC design flow?

In ASIC system design phase, the entire chip functionality is broken down to small pieces with clear understanding about the block implementation. For example: for an encryption block, do you use a CPU or a state machine. Some other large blocks need to be divided into subsystems and the relationship between the various blocks has to be defined. In this phase the working environment is documentation.

1) Register Transfer Level (RTL)

For digital ASICs or for digital blocks within a mixed-signal chip, this phase is basically the detailed logic implementation of the entire ASIC. This is where the detailed system specifications is converted into VHDL or Verilog language. In addition to the digital implementation, a functional verification is performed to ensure the RTL design is done according to the specifications.
When all the blocks are implemented and verified the RTL is then converted into a gate level netlist.

2) Synthesis

In this phase the hardware description (RTL) is converted to a gate level netlist. This process is performed by a synthesis tool that takes a standard cell library, constraints and the RTL code and produces a gate-level netlist.

Synthesis tools are running different implementations to provide best gate level netlist that meets the constraints. It takes into account power, speed, size and therefore the results can vary much from each other. To verify whether the synthesis tool has correctly generated the gate-level netlist a verification should be done.

3) Layout

In this stage, the gate level netlist is converted to a complete physical geometric representation. The first step is floorplanning which is a process of placing the various blocks and the I/O pads across the chip area based on the design constraints. Then placement of physical elements within each block and integration of analog blocks or external IP cores is performed. When all the elements are placed, a global and detailed routing is running to connect all the elements together. Also after this phase a complete simulation is required to ensure the layout phase is properly done.
The file produced at the output of the layout is the GDSII (GDS2) file which is the file used by the foundry to fabricate the silicon. The layout should be done according the silicon foundry design rules.

2.2 What is RISC-V v-sclae core?

RISC-V is a new instruction set architecture (ISA) that was originally designed to support computer architecture research and education and is now set to become a standard open architecture for industry implementations under the governance of the RISC-V Foundation. The RISC-V ISA was originally developed in the Computer Science Division of the EECS Department at the University of California, Berkeley.

And V-scale, an implementation of an RV32IM core. This core implements a simple, Z-scale-class pipeline, and is designed for integration with either existing microcontroller-class bus interconnects. Fig.1 shows the z-scale pipeline diagram. Z-scale pipeline is 32-bit 3-stage single-issue in-order pipe. It Executes RV32IM ISA, has M/U privilege modes I-bus and D-bus are AHB-Lite and 32-bits wide. Also, Fig.2 show the Z-scale system block diagram.

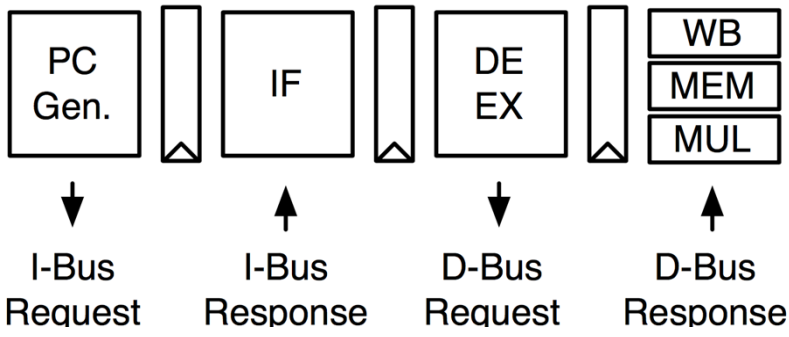

Fig.1 Z-scale pipeline

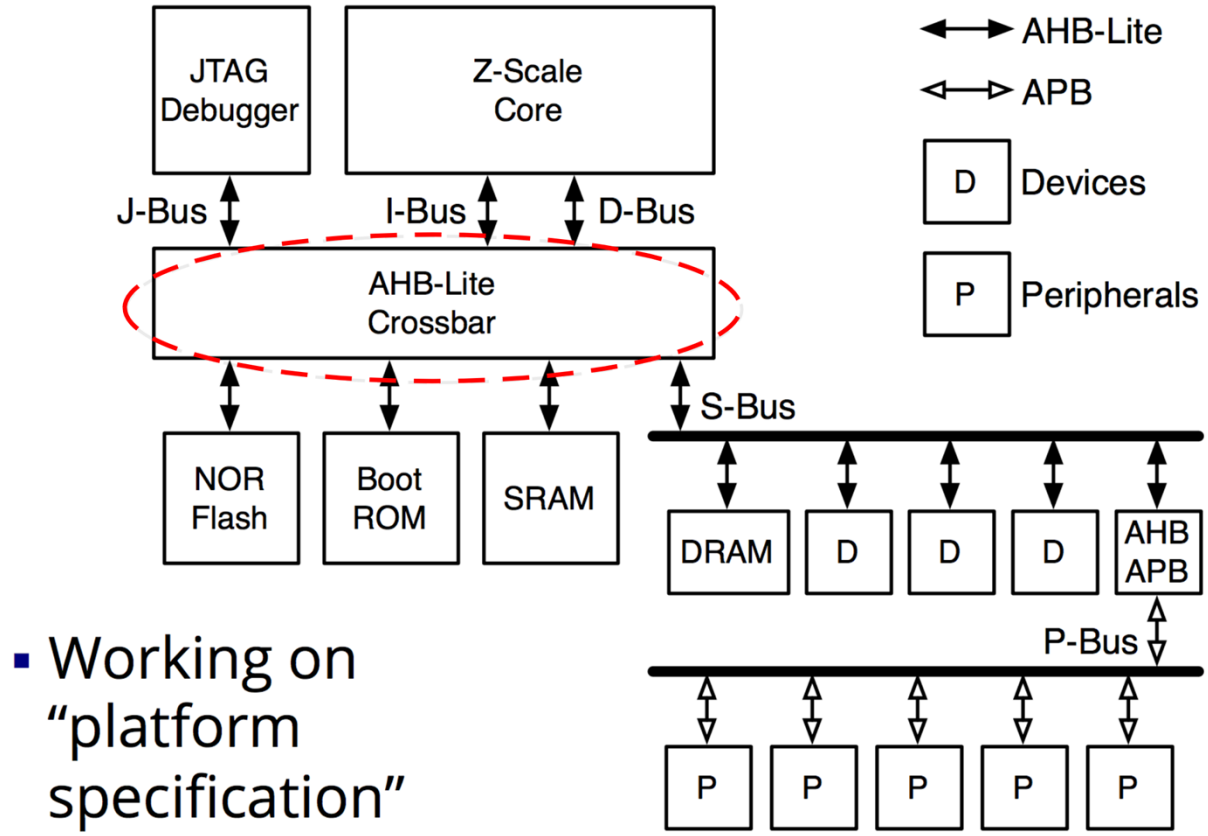

Fig.2 Z-scale system

If you want more information of section 2.2 you can read the material in
https://riscv.org/wp-content/uploads/2015/06/riscv-zscale-workshop-june2015.pdf

2.3 What is AHB-Lite?

In the Section 2.2 Fig.2, we can see that AHB-Lite Crossbar plays an important role when Z-scale core interconnects with external device. However, what is AHB-Lite protocol?

The main components of AHB-Lite system contain Master, Slaves, Address Decoder and Multiplexor. In RISC-V z-scale architecture, what we should concern about is Master and Slaves and the timing diagram of communication transitions. Fig.3 shows the AHB-Lite master and slave interfaces.

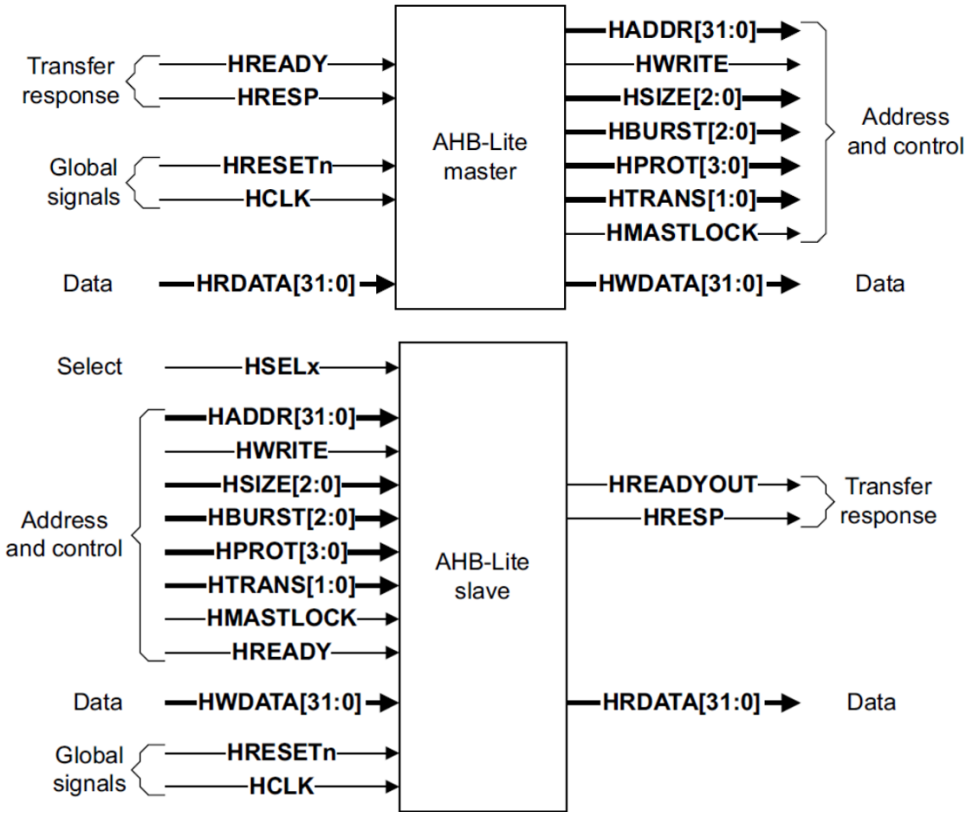

Fig.3 AHB-Lite master and slave interfaces

Here are the definitions of the interfaces shown in the Fig.3

- 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)

Here we give a detailed AHB-Lite implementation example. Fig.4 shows a simple AHB-Lite but connecting a processor (bus master) TO RAM, ROM and two peripherals (slaves). The master provides a asynchronous clock (HCLK) to all of the slaves and can reset the slaves by asserting HRESETn low. The master sends and address. The address decoder uses the most significant bits to generate the HSEL signal selecting which slave to access, and the slaves use the least significant bits to define the memory location or register. The master sends HWDATA for writes. Each slaves reads onto its own HRDATA, and a multiplexer chooses the data from the selected slave.

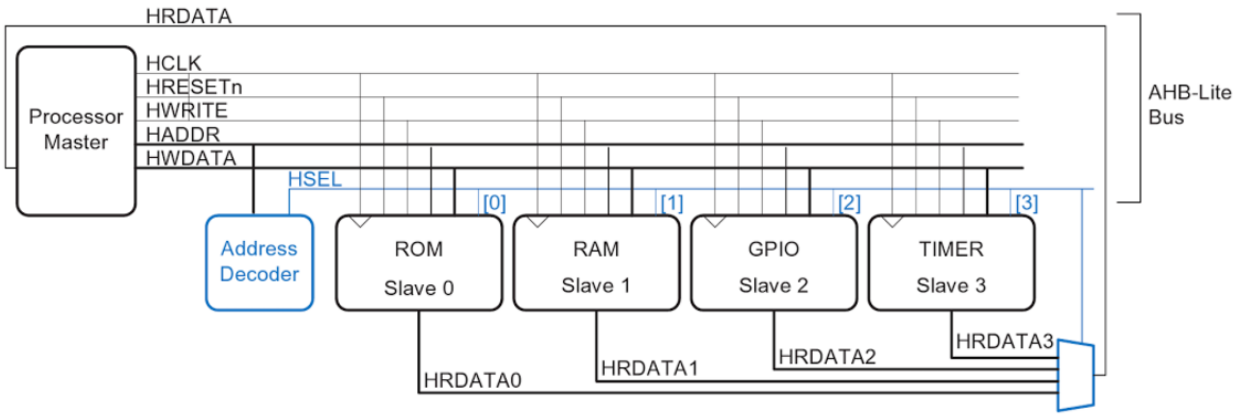

Fig.4 AHB-Lite transfer bus

The master sends a 32-bit address on one cycle and writes or reads data on the subsequent cycle. The write or read is called a “transfer”. For writes (see in Fig.5a), the master raises HWRITE and sends the 32-bit HWDATA to write. For reads (see in Fig.5b), the master lowers HWRITE and the slave responds with 32-bit HRDATA. Transfers can overlap so that the master can send the address of the next transfer while reading or writing data for the current transfer. Fig.5c illustrates the timing of the bus for a write followed immediately by a read. Observe how the data lags one cycle behind the address and how the two transfers partially overlap.

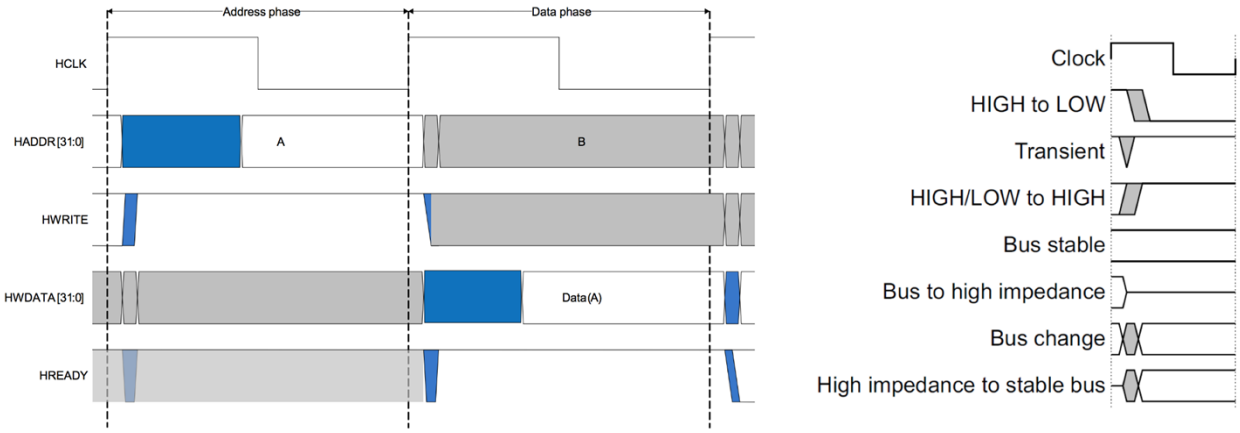

Fig.5a Basic Transfer - Write

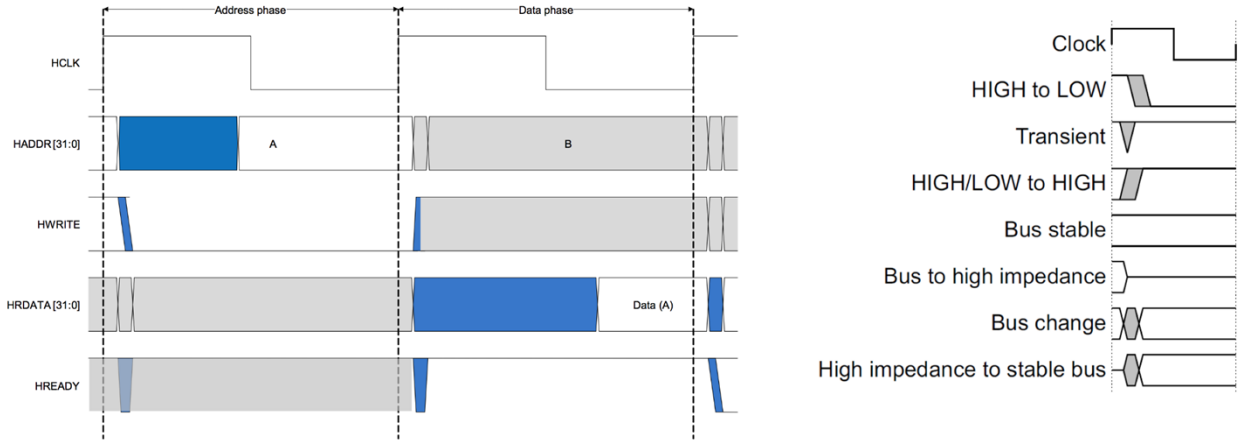

Fig.5b Basic Transfer- Read

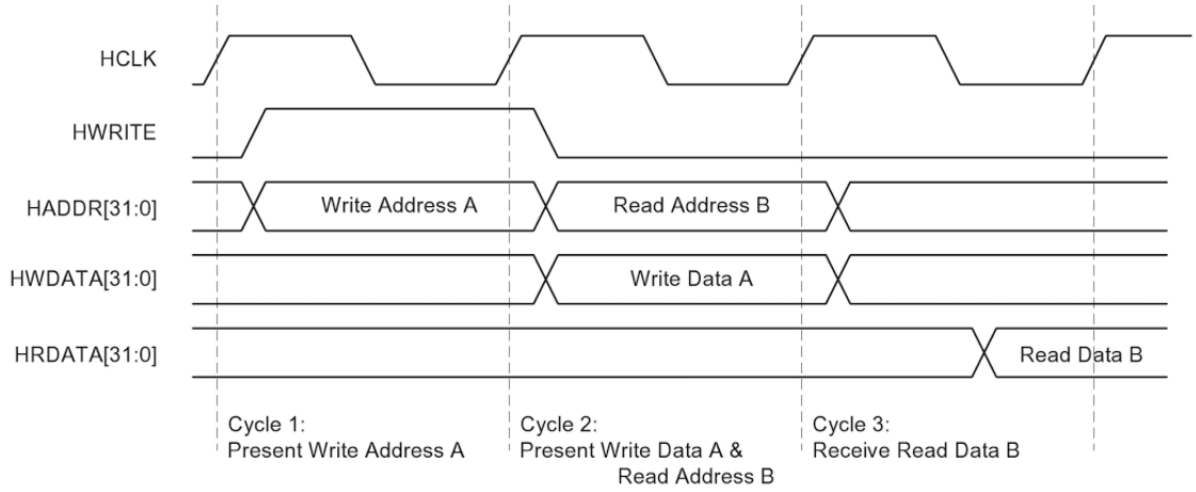

Fig.5c AHB-Lite transfer timing

If you want more information of section 2.3 you can read the material in
http://mazsola.iit.uni-miskolc.hu/~drdani/docs_arm/IHI0033A_AMBA3_AHB_Lite.pdf

3. How to get started

You are expected to accept the lab assignment in the link by click the button “Accept this assignment”.

https://classroom.github.com/assignment-invitations/2e1a214d45c4490bc17a1620c103639b

If you are the first time to use github, you should generate your own public key and add it to your github account, or you will have permission denied when you git clone the repository form our github classroom. When you add the ssh public key, you can follow the link below:

https://help.github.com/articles/connecting-to-github-with-ssh/

And then create a folder in your own linux server account and git clone your lab assignment repository. There are the command lines you may refer below:

% mkdir name_folder (i.e., mkdir lab1)
% cd name_folder
% git clone git@github.com:wustl-ese566/lab1-your_user_name.git (i.e., git clone git@github.com:wustl-ese566/lab1-YunfeiGu.git)
% cd lab1-your_user_name/ (i.e., cd lab1-YunfeiGu/)

This repository lab1-your_user_name/vscale contains the following files for you to build the project.

- src/Main/Verilog: a folder contains the open source code for RSIC-V cores
- src/Main/Verilog/lab1_dc.tcl: this is the tcl script you should change and submit.
- src/Test: a folder contains SRAM source code, testbench and input files.
- Makefile: a file that integrated instructions to compile and test v-scale core.
- .gitignore: determine which files and directories to ignore, before you make a commit.
- License: a statement that indicates this repo is truly open source.
- README.md: a file that records the instruction how to use this open source core.

Please be sure to source the class setup script using the following command before compiling your source code: module add ese461.

4. Required tasks

- Use Synopsys VCS to compile the Verilog source code in src/main/verilog directory by following the tutorials. Here, you can use your individual command to compile it or use the Makefile to run *make run-asm-tests*. In this process you will know how to compile the Verilog code using Synopsys VCS and how to load

memory to test the ISA on core. In this lab, you should get a folder named output. In this folder, there are several .vpd files. Actually, these files are the VCS simulation data results, you can use dve tool to look at the waveform of the simulation and to verify the logic.

- Use Design Compiler to do the synthesis of the RISC-V v-scale core by following the tutorials. The first thing you have to figured out is that which module is the top module and which are child modules. And then revise the tcl script we provided to synthesize the RISC-V v-scale source code. In this part, you should know how to write lab1_dc.tcl file to define your own constrains and then do the synthesis. When you finish the synthesis, you should get the timing, power, and area report for RISC-V v-scale core. Try to revise the constrains of lab1_dc.tcl file and have a better understanding of each parameter of these constrains, especially how will these parameters influence the timing, power and area results. In the end of this part, you should form a report contains the screenshots of all results when you do the synthesis.
- Use Cadence Encounter following the tutorials to do the place and route for RISC-V v-scale core by following the tutorials. In this part, try to use different parameters to layout (i.e., use different pin distributed form) and have a better understanding of these parameters and how will they influence the timing, power, area report. In the end of this part, you should take the screenshots of all results (i.e., physical layout and timing report) into the report.
- Write the report contains all contents required in above tasks. And copy your report in the directory /lab1-your_user_name/

5. Submission

Please submit your lab assignment on Github. You are expected to submit your report and lab1_dc.tcl file that contains the following contents:

There are the reference command lines to submit lab assignment below:

% cd directory_of_your_lab_assignment/lab1-your_user_name/vscale/src/Main/Verilog/
% git config --global user.name “your_user_name”
% git config --global user.email “your_email_for_github”
% git add lab1_dc.tcl
% git commit –m “your commits”
% cd directory_of_your_lab_assignment/lab1-your_user_name/
% git add lab1_report-your_name-your_student_ID.pdf
% git commit –m “your commits”
% git push origin master

You also should submit anything else you think may help us understand your code and

result.

Please do not summit files like compiling result(*simv*) or simulation data(*.vpd*).

6. Acknowledgement

https://riscv.org/
https://inst.eecs.berkeley.edu/~cs250/fa11/handouts/lab2-riscv-v2-vcs-dc.pdf
https://www.youtube.com/watch?v=iOh_TccRgm4
https://help.github.com/