NCTU-EE IC LAB – Fall 2019

Midterm Project

Release/Deadline : 2019.12.23 12:00 / 2019.01.08 12:00

Design: Customized ISA Processor

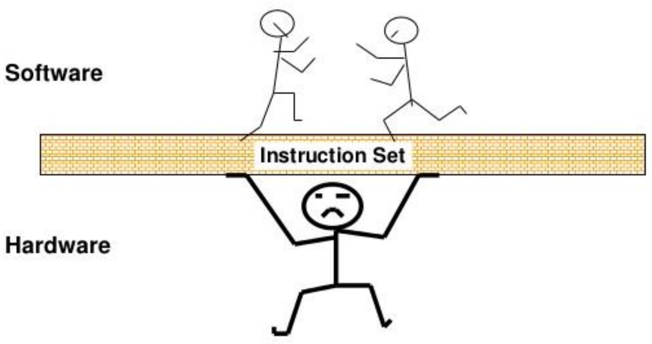
Data Preparation

- ➢ Extract test data from TA's directory:

  % tar xvf ~iclabta01/ICLAB_FP_2019F.tar
- ➢ The extracted Lab director contains:
  a. ICLAB_FP_2019F/ : Final project

Design Description

A reduced instruction set computer (RISC) is a computer instruction set (AKA: the instruction set architecture (ISA)) which allows a computer's microprocessor to have fewer cycles per instruction (CPI) than a complex instruction set computer (CISC) [from wikipedia]. In this exercise, you are asked to design a 16-bits CPU with 8 registers and the CPU should be available for the required instruction set, which shows below (very similar to MIPS).



The following are the four formats for the core instruction set

Type	-15-			- Format – (16-bits)		-0-
R	opcode (3-bits)	rs (3-bits)	rt (3-bits)	rd (3-bits)	func (3-bits)	?
Mult	opcode (3-bits)	rs (3-bits)	rt (3-bits)	rd (3-bits)	rl (3-bits)	?
I	opcode (3-bits)	rs (3-bits)	rt (3-bits)	immediate (7-bits)		
J	opcode (3-bits)	Address (13-bits)				

Register s(rs), Register t(rt), Register d(rd) and Register l(rl) represent the address of registers. Since the instruction takes 3 bits to store the address, it means we have 8 registers, from r0 to r7. Each register reserve 16 bits to store data, e.g. rs = 3’b000 means one of operands is “r0”. rt = 3’b101 means one of operands is “r5”, and so on.

Detailed instructions are shown below

Function Name	**Meaning**	**Type**	**Instruction Binary Encode**
ADD	rd = rs + rt	R	000-rs-rt-rd-000?
SUB	rd = rs – rt	R	000-rs-rt-rd-001?
AND	rd = rs & rt (bit-wise)	R	000-rs-rt-rd-010?
OR	rd = rs | rt (bit-wise)	R	000-rs-rt-rd-011?
NAND	rd = ~(rs & rt) (bit-wise)	R	000-rs-rt-rd-100?
NOR	rd = ~(rs | rt) (bit-wise)	R	000-rs-rt-rd-101?
XOR	rd = rs ^ rt (bit-wise)	R	000-rs-rt-rd-110?
Set less than	if(rs<rt) rd=1 else rd=0	R	000-rs-rt-rd-111?
Mult	{rd, rl} = rs * rt	Mult	001-rs-rt-rd-rl?
ADDI	rt = rs+ immediate (sign)	I	010-rs-rt-iiiiiii
SUBI	rt = rs – immediate (sign)	I	011-rs-rt-iiiiiii
Load	rt = DM[rs+immediate(sign)]	I	100-rs-rt-iiiiiii
Store	DM[rs+immediate(sign)] = rt	I	101-rs-rt-iiiiiii
Branch on equal	if(rs==rt) pc=pc+1+immediate(sign)	I	110-rs-rt-iiiiiii
Jump	pc = address	J	111-address

*pc : program counter

*? : useless bit

Two instruction memory and a data memory are used in this lab. There are two cores in your design, one is used for decoding instruction from instruction memory #1 and the other is used for decoding instruction from instruction memory #2. Because there is only one data memory in this exercise, you need to make sure the data in the memory is newest one. Or you may meet some problems. You are encouraged to use your memory for better performance, e.g. area, latency and power. Just be careful, if you use the pipelining architecture, e.g. basic five-stage pipeline in a RISC machine, you must deal with data hazard problem.

Inputs & Output

I/O	Signal Name	Description
Input	clk	Positive edge trigger clock within clock period 20.0ns
Input	rst_n	Asynchronous reset active low reset
Input	interrupt_1, interrupt_2	The core 1 should be stopped when interrupt_1 asserts. The core 2 should be stopped when interrupt_2 asserts.
Output	stall_core1	Pull high when core 1 is busy Cannot be continuous high for 1500 cycles
Output	stall_core2	Pull high when core 2 is busy Cannot be continuous high for 1500 cycles

- All inputs will be changed at clock ***negative*** edge.
- There is ***only 1 reset*** before the first pattern, thus, your design must be able to reset automatically.
- The test pattern will check the value in all registers are correct or not at clock ***negative*** edge if stall is low.
- All the registers should be zero after the reset signal is asserted.
- The value in all registers should be **unchanged** when stall is low.
- TA will check the value in data memory after both interrupt signals assert 1500 cycles. Please refer appendix.

Pattern

- Update DRAM data
  00_TESTBED/DRAM/dram_file.dat
  Hint: You can refer lecture note for Lab03 about file input.
- DRAM Raad/Write Latency
  Modify **DRAM_R_LAT**, **DRAM_W_LAT** in the DRAM. You can set DRAM_R_LAT = 0 & DRAM_W_LAT = 0 to speed up the simulation time. But for demo, TA will set **DRAM_R_LAT = 90** and **DRAM_W_LAT = 100.**

AXI 4 Interface (Connected with DRAM in Pattern)

AXI-4 signal name (lower case) + _m_inf (suffix)

Write Address Channel

Signal	Source	Description
AWID[3:0]	Master	Write address ID. This signal is the identification tag for the write address group of signals. (In this project, we only use this to recognize master, reordering method is not supported)
AWADDR[31:0]	Master	Write address. The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst transaction.
AWLEN[7:0]	Master	Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
AWSIZE[2:0]	Master	Burst size. This signal indicates the size of each transfer in the burst. (We only support 3b'001 which is 2 Bytes (matched with Data Bus-width) in each transfer)
AWBURST[1:0]	Master	Burst type. The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated. (only INCR support in this Project)
AWVALID	Master	Write address valid. This signal indicates that valid write address and control information are available: 1 = address and control information available 0 = address and control information not available. The address and control information remain stable until the address acknowledge signal, **AWREADY**, goes HIGH.
AWREADY	Slave	Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals: 1 = slave ready 0 = slave not ready.

Write Data Channel

Signal	Source	Description
WDATA	Master	Write data. The write data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide. (This project only support 16 bit data width: WDATA[15:0])
WLAST	Master	Write last. This signal indicates the last transfer in a write burst.
WVALID	Master	Write valid. This signal indicates that valid write data and strobes are available: 1 = write data and strobes available 0 = write data and strobes not available.
WREADY	Slave	Write ready. This signal indicates that the slave can accept the write data: 1 = slave ready 0 = slave not ready.

Write Response Channel

Signal	Source	Description
BID[3:0]	Slave	Response ID. The identification tag of the write response. The **BID** value must match the **AWID** value of the write transaction to which the slave is responding.
BRESP[1:0]	Slave	Write response. This signal indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR. (In this project we only issue OKAY)
BVALID	Slave	Write response valid. This signal indicates that a valid write response is available: 1 = write response available. 0 = write response not available.
BREADY	Master	Response ready. This signal indicates that the master can accept the response information. 1 = master ready. 0 = master not ready.

Read Address Channel

Signal	Source	Description
ARID[3:0]	Master	Read address ID. This signal is the identification tag for the read address group of signals. (In this project, we only use this to recognize master, reordering method is not supported)
ARADDR[31:0]	Master	Read address. The read address gives the address of the first transfer in a read burst transaction.
ARLEN[7:0]	Master	Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
ARSIZE[2:0]	Master	Burst size. This signal indicates the size of each transfer in the burst. (We only support 3b'001 which is 2 Bytes (matched with Data Bus-width) in each transfer)
ARBURST[1:0]	Master	Burst type. The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated. (only INCR: 2b'01 support in this Project)
ARVALID	Master	Read address valid. This signal indicates, when HIGH, that the read address and control information is valid and will remain stable until the address acknowledge signal, **ARREADY**, is high. 1 = address and control information valid 0 = address and control information not valid.
ARREADY	Slave	Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals: 1 = slave ready 0 = slave not ready.

Read Data Channel

Signal	Source	Description
RID[3:0]	Slave	Read ID tag. This signal is the ID tag of the read data group of signals. The **RID** value is generated by the slave and must match the **ARID** value of the read transaction to which it is responding.
RDATA	Slave	Read data. (This project only support 16 bit data width: RDATA[15:0])
RRESP[1:0]	Slave	Read response. This signal indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR. (In this project we only issue OKAY)
RLAST	Slave	Read last. This signal indicates the last transfer in a read burst.
RVALID	Slave	Read valid. This signal indicates that the required read data is available and the read transfer can complete: 1 = read data available 0 = read data not available.
RREADY	Master	Read ready. This signal indicates that the master can accept the read data and response information: 1= master ready 0 = master not ready.

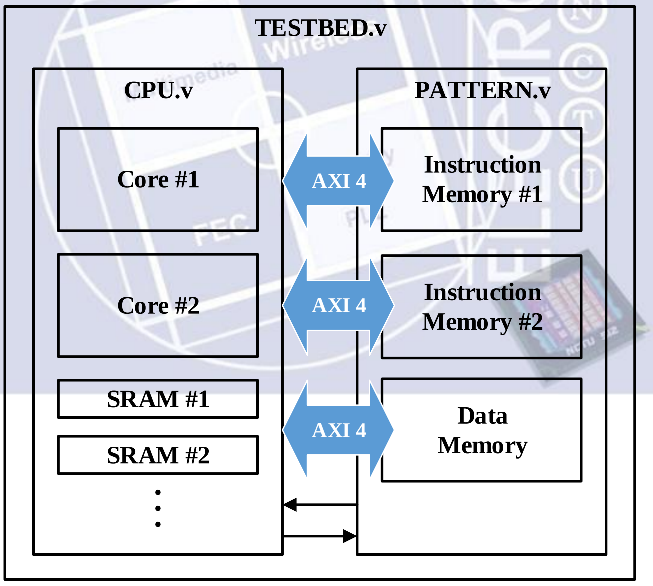
Specifications

1. Top module name: **CPU** (top file name: **CPU.v**)
2. It is **asynchronous** reset and **active-low** architecture. **rst_n** would active once at the beginning. All the registers should be zero after the reset signal asserts.
3. All the data are in **signed** format in all registers.
4. The maximum clock period is set to **20ns**, and **you can determine the clock period by yourself**.
5. The input delay and the output delay should be **half** of the clock period. For example, if clock period is 10ns, the input delay and the output delay will be 5ns.
6. The output loading is set to **0.05**. The wire load is **top** and target library is **slow.db**
7. The synthesis result of data type cannot include any **LATCH** (in syn.log).
8. In this project, you should **modify the syn.tcl (e.g. link library for your memory.db) by yourself**. **compile_ultra** will be used to synthesis. Since memories are used in this project, the syn.tcl in Lab05 may be a good reference one.
9. You **CANNOT PASS** the demo if there are **timing violation** messages in the gate

level simulation WITHOUT notimingchecks option

10. No **ERROR** is allowed in every simulation/synthesis.
11. The **size of SRAM Memory Block** should between **256~2048 Bytes**, the **number of SRAM is not limited.**
12. **You must use memory in your design or you wouldn't get any point in this project.**
13. The maximum area should be smaller than **2,000,000.**
14. **You cannot change the register name from the original file.**
15. You may meet the overflow problem when doing arithmetic operation. You do not need to handle this problem.
16. The program counter will not over access range of memory when the jump instruction asserts. So, you need to think how to write your pattern by yourself.
17. For achieving the data dependence in real case, the next load/store address will be constrained in certain range for data memory and the address for jump/branch instruction will also be constrained in certain range. Please refer appendix.

Block Diagram



Grading Policy

1st Demo (100%), 2nd Demo (70% of total)

- Synthesis and Gate Level Simulation Correctness: 30%
- APR and Post Level Simulation Correctness: 30%
  You can only get this score with correct synthesis and gate-level simulation.
- Performance : 40%

1. **Performance: (Total Cycle)$^{1.5}$ x Core Area x Clock Period**

2. You can only get performance score with correct APR and post simulation result.

Note

1. Please upload the following files on new e3 platform before **12:00 noon** on **Jan. 08**.
2. Upload Format: ICLAB_2019FALL_FP_iclabXX.tar
   - *iclabXX_GATE_??ns_POST_??ns.txt*
   - 01_RTL/
     CPU.v, file_list.f
   - 02_SYN/
     syn.tcl
   - 02_SYN/Netlist/
     CPU_SYN.sdf, CPU_SYN.v
   - 04_MEM/
     All your memory (*.lib, .vclef, .v, .db*)
   - 05_APR/
     CHIP.inn.dat, CHIP.sdc, CHIP.io, CHIP.sdf, CHIP.v, CHIP.inn

   *Hint : Use " tar -cvf " to generate tar file on workstation.

   Note that you can provide **multiple** memory specs in this lab. If the uploaded files violating the naming rule, you will get **5 deduct points**.
3. Template folders and reference commands:

   01_RTL/ (for RTL simulation) **./01_run**

   02_SYN/ (for Synthesis) **./01_run_dc**

   Remember modify .tcl to fit your memory db name

   (Check the design which contains **Latch** and **Error** or not in **syn.log**)

   (Check the design's timing in /Report/***CPU.timing*** to see if the slack is **MET**)

   03_GATE_SIM/ (Gate Level simulation) **./01_run**

   You can key in **./09_clean_up** to clear all log files and dump files in each folder

   04_MEM/ (Memory location)

   You should generate your memories and put the required files (.v and .db) here

   05_APR/ (back-end APR) **./01_combine**

   ./02_run_uniquify

   ./09_clean_up (clean all log and command files)

   06_POST_SIM/(Post-layout simulation) **./01_run**

Hint

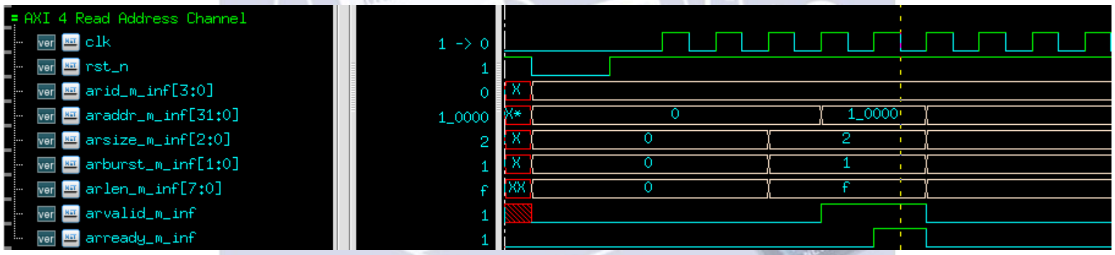
1. In this project you may use multiple kinds of memories, thus you have to provide multiple netlists and libraries for the simulation and the synthesis.

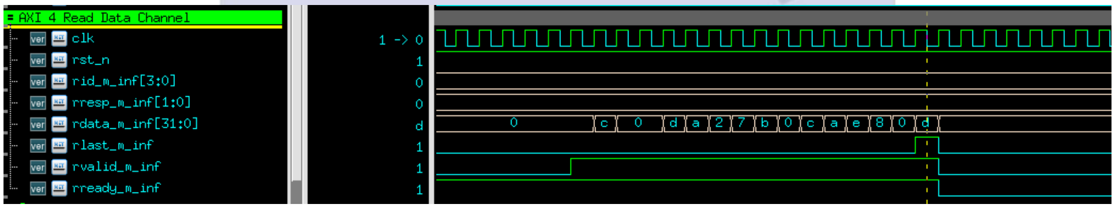
For example, if you use two memories, you have to provide two .v and .db file. Here provides a simple flow to perform:

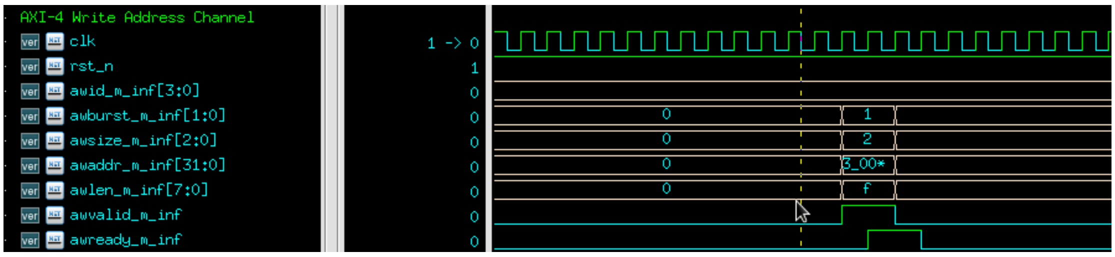
Login to the Memory Compiler Server :
Copy the template memory generate folder :
Go to the folder and execute the shell script.
Argument of script: [name] [number of word] [number of bit] [mux type] [frequency]
Frequency could be any value larger than 0 (We suggest you set to 1)
Example:
Account Name: iclab99
Spec: name: SRAM128W64B number of word:128
number of bit : 64 mux type :4 frequency : 1

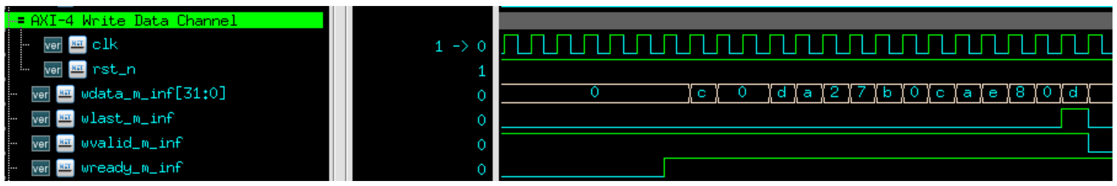
```
ssh mem@ee08.ee.nctu.edu.tw
cp -r mid_mem_template iclab99_mid
cd iclab99_mid
./01_mem_gen.sh SRAM128W64B 128 64 4 1
```

AXI example waveform:
Read Transaction



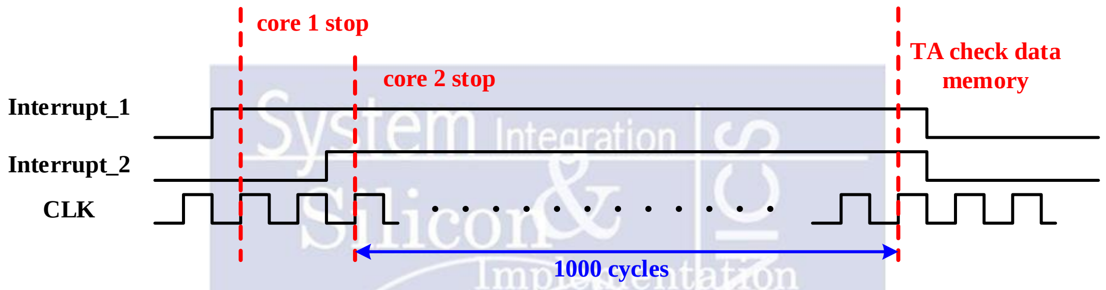




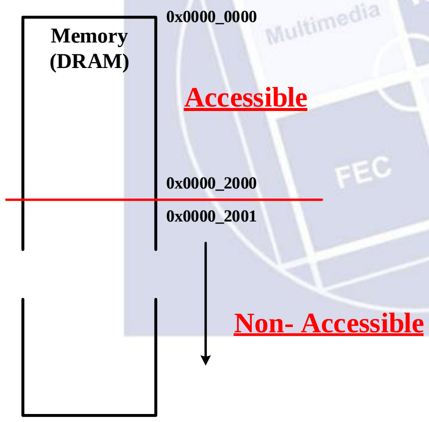


Appendix

- TA will check the value in data memory after both interrupt signals assert 1000 cycles. Please refer appendix.



- Memory available range

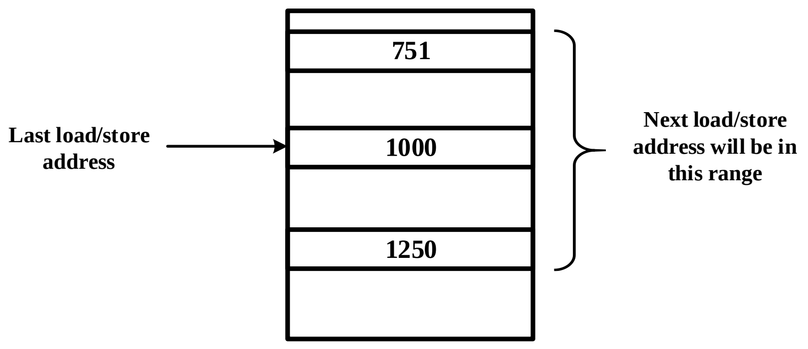


- Instruction

  Ex. 000-001-010-011-000-0 means r1 + r2 and saves the result into r3.

- Data Dependence Prediction

  For achieving the data dependence in real case, the next load/store address will be constrained in certain range for data memory and the address for jump/branch instruction will also be constrained in certain range. The range is from (current address – 250 + 1) to (current address + 250), e.g. if current address is dec(1000), the next load/store address will be from dec(751) to dec(1250).


751
Last load/store address
1000
Next load/store address will be in this range
1250