차세대 반도체 디지털 혁신공유대학 사업단

POLARIS

Lecture 6: AMBA AHB Bus Master, Slave, and Interconnect

Xuan-Truong Nguyen

Road map

RISC-V Summary

AMBA AHB Bus
(ALU IP)

Multiplier IP

AHB Decoder

Bus Interconnect

Where we are now?

CPU memory

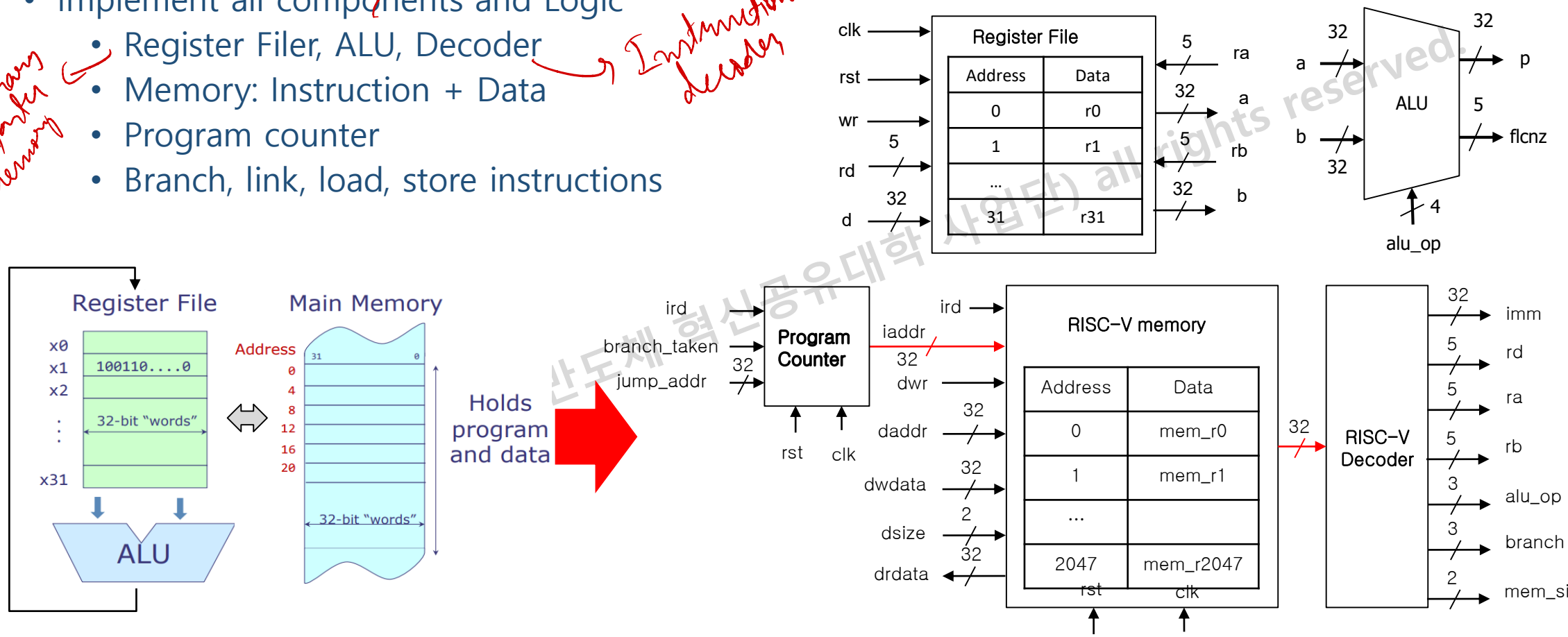
Bus and IO

CNN Accelerator

#	Topics	Labs
1	Introduction, Basic Digital Logic	ModelSim, Encoder, Counter
2	Introduction to RISC-V	Register File, Shifter and ALU
3	Instruction Set Architecture	Instruction memory, decoder
4	Instructions	Program Counter, Branch Instruction
5	RISC-V Compiler, GNU Tool Chain	Load/Store Instruction, RISC-V core
6	AMBA Bus Specification	AHB Master, AHB slave, AHB Bus
7	IO Devices and Display Panel	LCD Drive, Display Panel
8	System integration	LCD interface, memory interface, top system
9	Accelerator, Super resolution networks	Matlab, CNN reference S/W, quantization, BRAM
10	Computing Units of DNN accelerator	MAC, Adder tree, quantization, and activation
11	Datapath for a DNN accelerator I	Sliding windows, weight/bias/scale buffers
12	Datapath for a DNN accelerator II	Buffer access, Super resolution NPU
13	System Integration	BRAM, Function verification, DMA
14	Optimization	Dual buffering, Pipelining
15	Final project and Final Exam	Execution time reduction, Memory buffer reduction

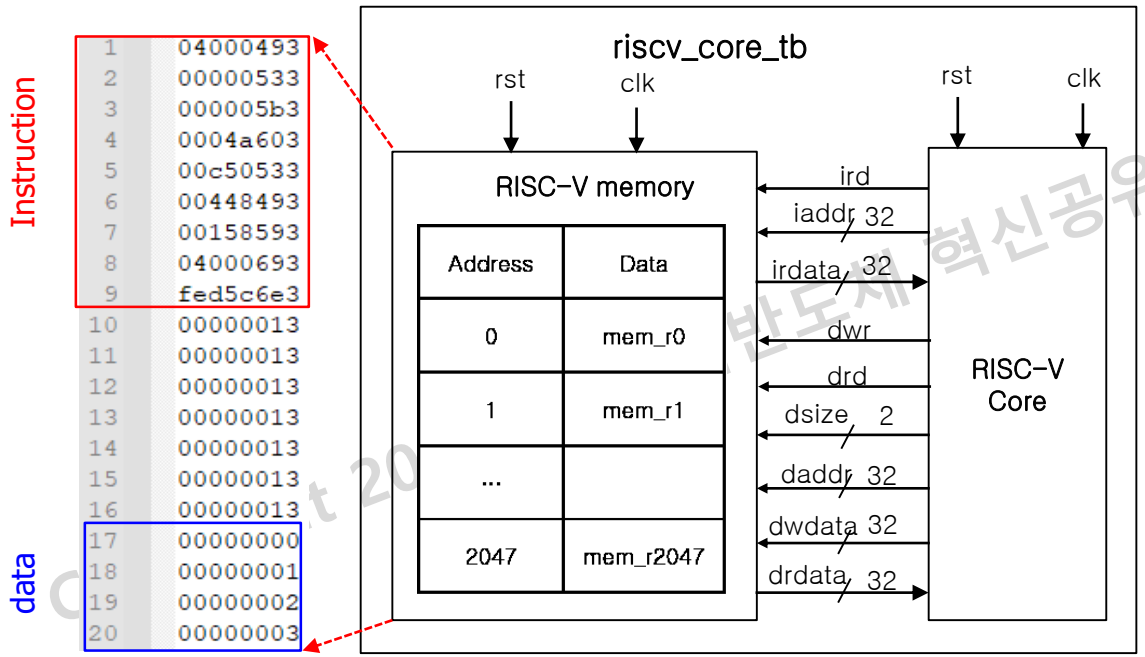
RISC-V

- RISC-V instruction set architecture (ISA)
- Implement all components and Logic
  - Register Filer, ALU, Decoder
  - Memory: Instruction + Data
  - Program counter
  - Branch, link, load, store instructions



RISC-V Core and Memory

- Test bench includes
  - A RISC-V core
  - A memory with initialized instructions and data.




```
riscv_memory #(.FIRMWARE("mem.hex"))
u_riscv_memory
(
  ./*input          */clk_i(clk_i),
  ./*input          */reset_i(reset_i),
  ./*input  [31:0]  */iaddr_i(iaddr),
  ./*output [31:0]  */irdata_o(irdata),
  ./*input          */ird_i(ird),
  ./*input  [31:0]  */daddr_i(daddr),
  ./*input  [31:0]  */dwdata_i(dwdata),
  ./*output [31:0]  */drdata_o(drdata),
  ./*input   [1:0]  */dsize_i(dsize),
  ./*input          */drd_i(drd),
  ./*input          */dwr_i(dwr)
);

riscv_core
u_riscv_core
(
  ./*input          */clk_i(clk_i),
  ./*input          */reset_i(reset_i),
  ./*output         */lock_o(lock),
  ./*output [31:0]  */iaddr_o(iaddr),
  ./*input  [31:0]  */irdata_i(irdata),
  ./*output         */ird_o(ird),
  ./*output [31:0]  */daddr_o(daddr),
  ./*output [31:0]  */dwdata_o(dwdata),
  ./*input  [31:0]  */drdata_i(drdata),
  ./*output  [1:0]  */dsize_o(dsize),
  ./*output         */drd_o(drd),
  ./*output         */dwr_o(dwr)
);
```

Verification

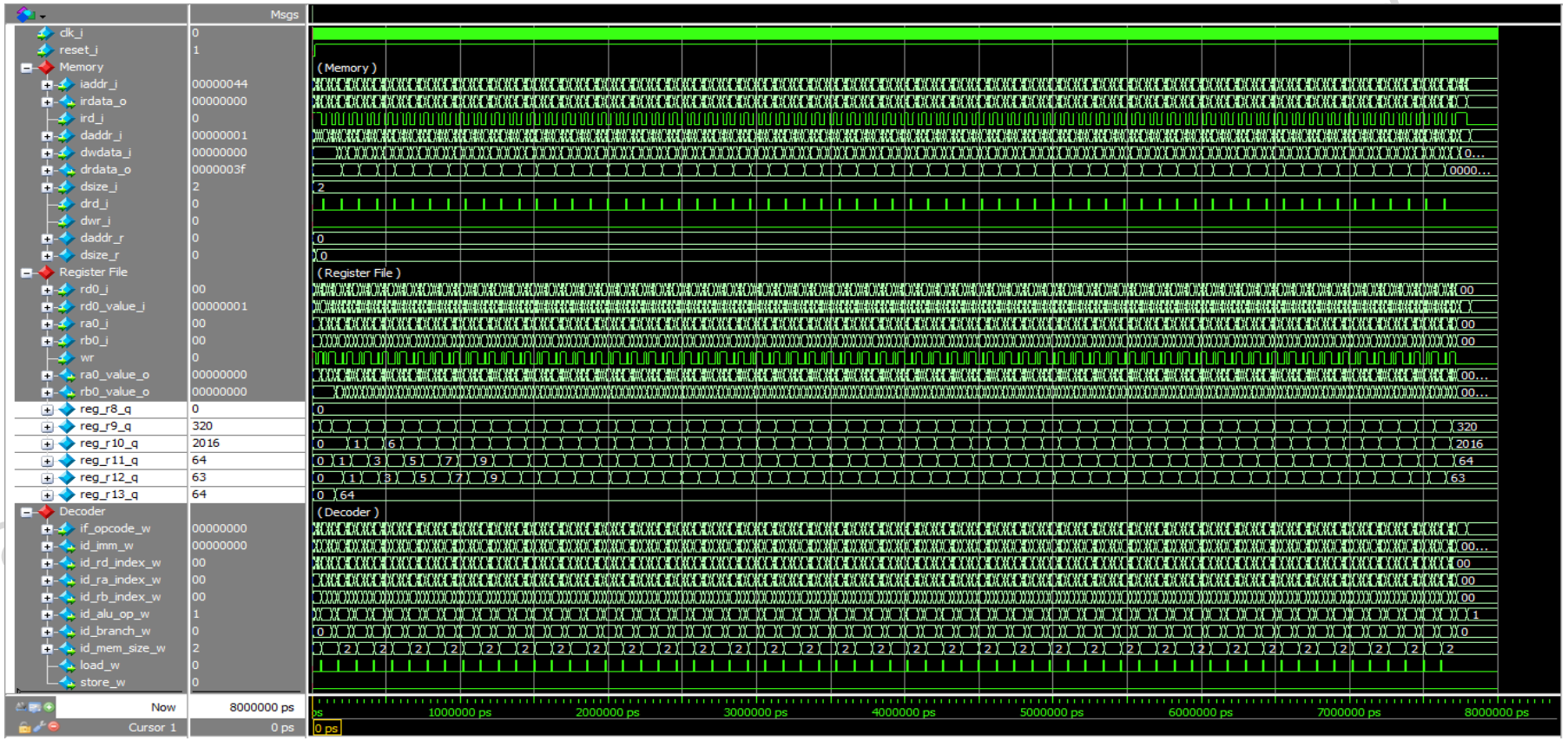
- Compiler and assembler: Generate machine code (mem.hex)
- Do simulation

C code

```
int A[64];
int sum = 0;
for (int i=0; i<64; i++)
  sum += A[i];
```

Assembly

```
addi x9, x0, 64
add x10, x0, x0
add x11, x0, x0
Loop:
  lw x12, 0(x9)
  add x10, x10, x12
  addi x9, x9, 4
  addi x11, x11, 1
  addi x13, x0, 64
  blt x11, x13, Loop
```



Optimization: Instruction Reordering

- RISC-V core runs different programs
- Three assemble versions of a C program
  - Baseline
  - OPT1: reorder instructions 5 and 6.
  - OPT2: reorder instructions 5, 6, and 7.

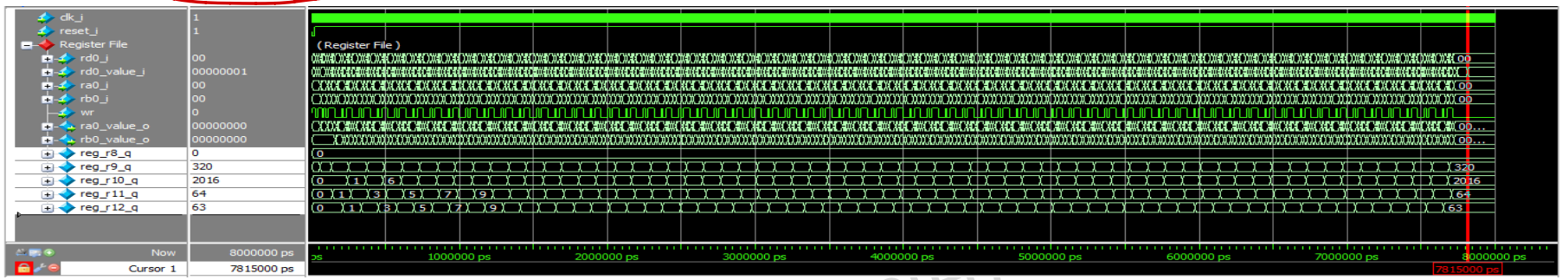
C code	Assembly Code		
	Baseline	Optimized 1 (Opt1)	Optimized 2 (Opt2)

int A[64];
int sum = 0;
for (int i=0; i<64; i++)
  sum += A[i];
	addi x9, x0, 64
	add x10, x0, x0
	add x11, x0, x0
	Loop:
	  lw x12, 0(x9)
	  add x10, x10, x12
	  addi x9, x9, 4
	  addi x11, x11, 1
	  addi x13, x0, 64
	  blt x11, x13, Loop
		addi x9, x0, 64
		add x10, x0, x0
		add x11, x0, x0
		Loop:
		  lw x12, 0(x9)
		  addi x9, x9, 4
		  add x10, x10, x12
		  addi x11, x11, 1
		  addi x13, x0, 64
		  blt x11, x13, Loop
			addi x9, x0, 64
			add x10, x0, x0
			add x11, x0, x0
			Loop:
			  lw x12, 0(x9)
			  addi x9, x9, 4
			  addi x11, x11, 1
			  add x10, x10, x12
			  addi x13, x0, 64
			  blt x11, x13, Loop

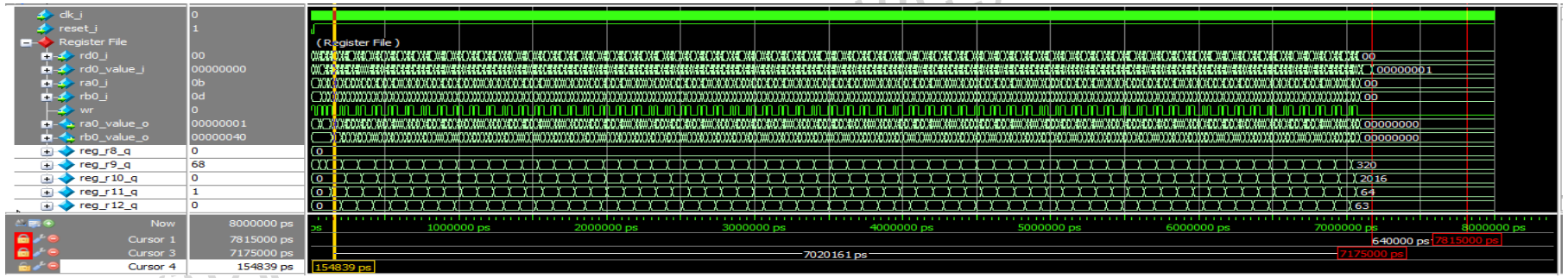
Optimization: Waveform

- Number of cycles: Baseline > OPT1 > OPT2

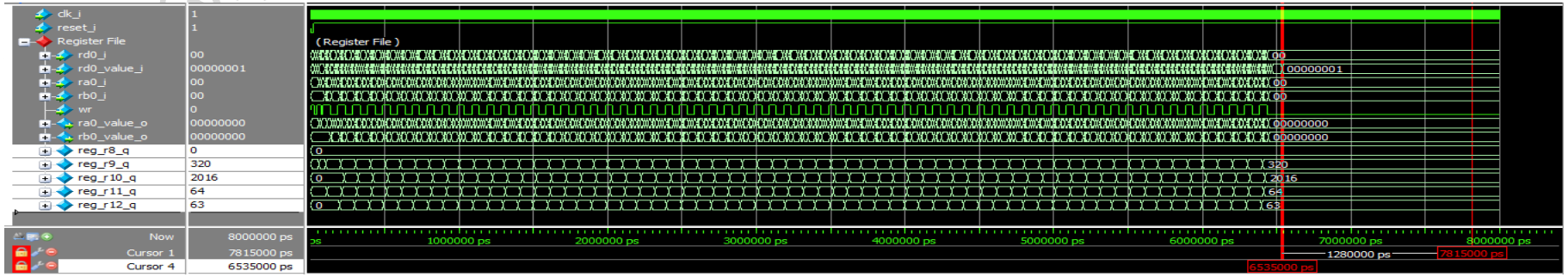
Baseline



OPT 1



OPT 2



Optimization: Instruction reordering

- Three observations:
  - LOAD takes multiple cycles
    - Memory is slower than Register File
  - Data dependency: x12 is the output for LOAD
    - x12 is an input of "add x10, x10, x12" (x10 ← x10 + x12).
  - Pipelining: Hide the latency between two dependent instructions
    - Schedule and execute other **dependency-free** instructions.

Doing the dependent instruction together.

C code	Assembly Code		
	Baseline	Optimized 1 (Opt1)	Optimized 2 (Opt2)

int A[64];
int sum = 0;
for (int i=0; i<64; i++)
  sum += A[i];
	addi x9, x0, 64
	add x10, x0, x0
	add x11, x0, x0
	Loop:
	  lw x12, 0(x9)
	  add x10, x10, x12
	  addi x9, x9, 4
	  addi x11, x11, 1
	  addi x13, x0, 64
	  blt x11, x13, Loop
		addi x9, x0, 64
		add x10, x0, x0
		add x11, x0, x0
		Loop:
		  lw x12, 0(x9)
		  addi x9, x9, 4
		  add x10, x10, x12
		  addi x11, x11, 1
		  addi x13, x0, 64
		  blt x11, x13, Loop
			addi x9, x0, 64
			add x10, x0, x0
			add x11, x0, x0
			Loop:
			  lw x12, 0(x9)
			  addi x9, x9, 4
			  addi x11, x11, 1
			  add x10, x10, x12
			  addi x13, x0, 64
			  blt x11, x13, Loop

Goals

- Understand inefficiency in naïve string matching
- Learn string matching using automata
- Learn Rabin-Karp algorithm
- (Knuth-Morris-Pratt algorithm, Boyer-Moore algorithm)

String Matching

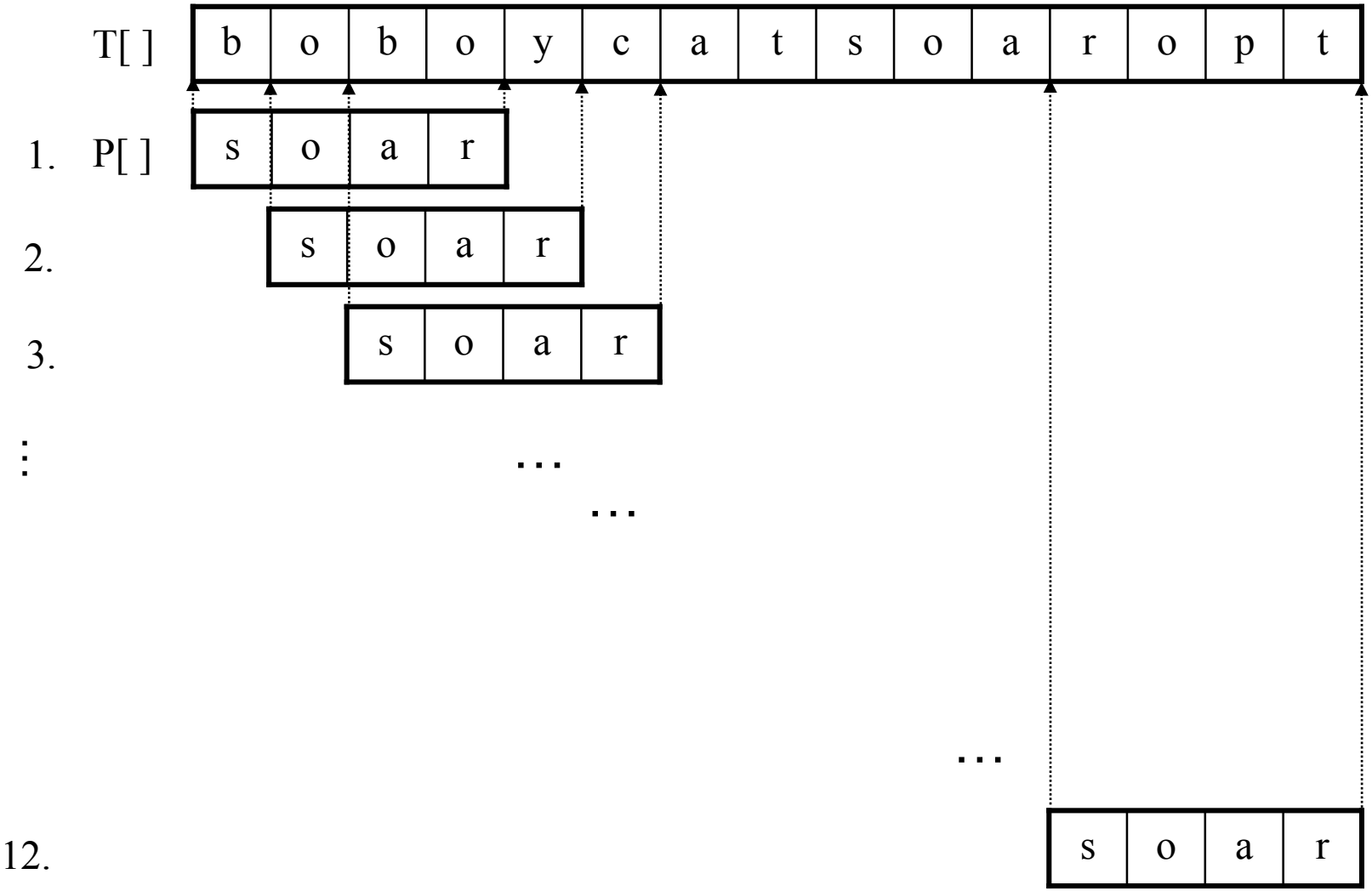
- Input
  - T[1…n]: text string
  - P[1…m]: pattern string
  - $m \ll n$
- String matching problem
  - Find all occurrences of pattern P[1…m] in text T[1…n]

Naïve Algorithm

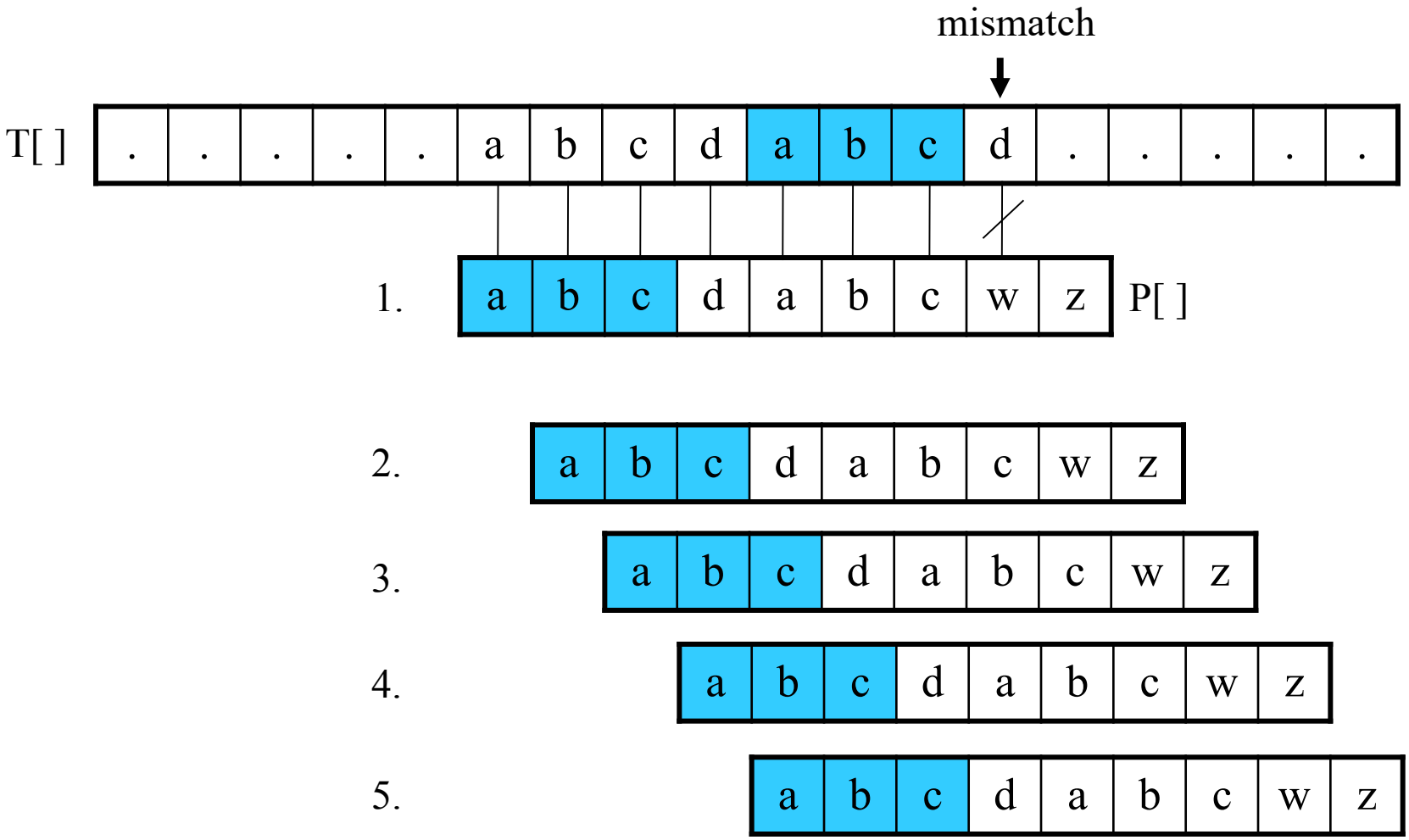
```
naiveMatching(T, P)
    ▷ T[1…n], P[1…m]
    for i ← 1 to n-m+1
        if (P[1…m] = T[i…i+m-1]) then
            output "occurrence at i"
```

✓ Time complexity: $O(mn)$

Naïve Algorithm



Naïve Algorithm



Matching using Automata

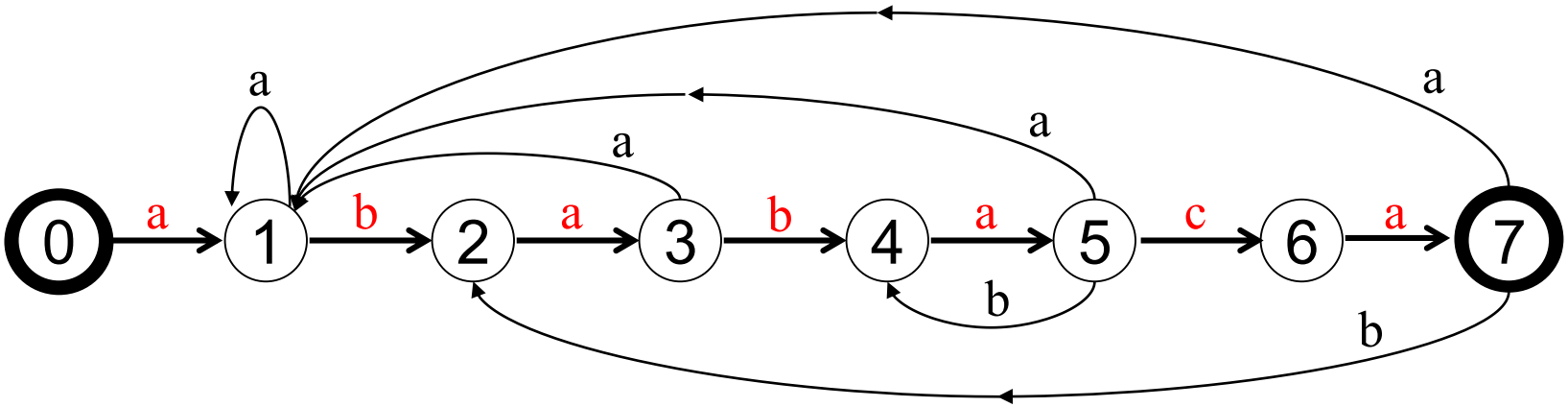
- Automaton: $(Q, q_0, A, \sum, \delta)$
  - Q : set of states
  - q_0 : start state
  - A : set of final states
  - $\sum$: input alphabet
  - δ : state transition function
- Matched characters so far in string matching are represented by states in an automaton

Automaton for Pattern ababaca

$Q = \{0, 1, \ldots, m\}$, $q_0 = 0$, $q_f = m$

State i means that P[1…i] matches text characters so far

$\delta(i, a) = \max\{k : \text{P}[1\ldots k] \text{ is a suffix of P}[1\ldots i]a\}$



Unspecified edges go to state 0

T: dvganbbactababaababacabababacaagbk…

Implementation of Automata

state \ input character	a	b	c	d	e	…	z
0	1	0	0	0	0	…	0
1	1	2	0	0	0	…	0
2	3	0	0	0	0	…	0
3	1	4	0	0	0	…	0
4	5	0	0	0	0	…	0
5	1	4	6	0	0	…	0
6	7	0	0	0	0	…	0
7	1	2	0	0	0	…	0

state \ input character	a	b	c	others
0	1	0	0	0
1	1	2	0	0
2	3	0	0	0
3	1	4	0	0
4	5	0	0	0
5	1	4	6	0
6	7	0	0	0
7	1	2	0	0

Matching using Automata

FA-Matcher (T, δ, f)
▷ T[1…n]
▷ f : final state
    $q \leftarrow 0$
    for $i \leftarrow 1$ **to** n
        $q \leftarrow \delta(q, \mathrm{T}[i])$
        if ($q = f$) **then** output "occurrence at i-m+1"

✓ Time complexity: $\Theta(n + |\sum| m)$

Computing Automata

```
Compute-FA(P)
▷ P[1…m]
    for i ← 0 to m
        for each a ∈ Σ
            k ← min(i+1, m)
            while (P[1…k] is not a suffix of P[1…i]a)
                k ← k - 1
            δ(i, a) ← k
    return δ
```

- ✓Time complexity: $O(m^3|\Sigma|)$
- ✓Can be reduced to $O(m|\Sigma|)$

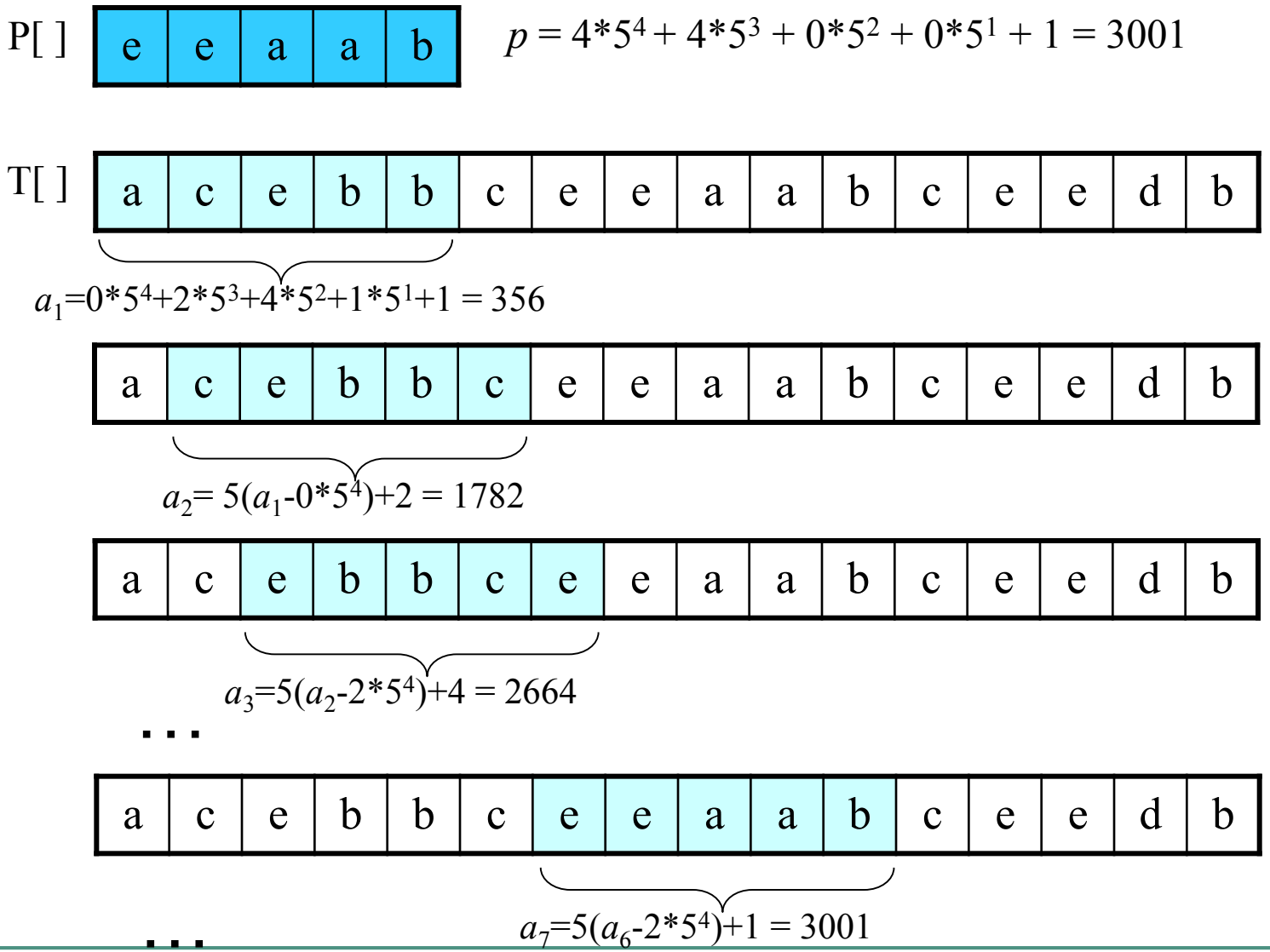
Rabin-Karp Algorithm

- By converting the pattern string to a number, string comparison is replaced by number comparison.
- Conversion
  - Digit system is determined by the size of alphabet $\sum$
  - $\sum$ = {a, b, c, d, e}
    - $|\sum| = 5$
    - a, b, c, d, e correspond to 0, 1, 2, 3, 4, respectively
    - String "cad" is converted to $2*5^2+0*5^1+3*5^0 = 53$

Conversion

- Converting $T[i \ldots i+m-1]$
  - $a_i = T[i+m-1] + d(T[i+m-2] + d(T[i+m-3] + d(\ldots + d(T[i]))\ldots)$
  - $\Theta(m)$ time (Horner's rule)
  - $\Theta(mn)$ for whole text $T[1 \ldots n]$
  - Not better than naïve algorithm
- Successive computations
  - $a_i = d(a_{i-1} - d^{m-1}T[i-1]) + T[i+m-1]$
  - d^{m-1} is computed in advance
  - 2 multiplications, 2 additions

Matching with Numbers



Matching with Numbers

basicRabinKarp(P, T, d, q)
▷ P[1…m], T[1…n]
  $p \leftarrow 0$; $a_1 \leftarrow 0$
  for $i \leftarrow 1$ **to** m ▷ compute a_1
    $p \leftarrow dp + \mathrm{P}[i]$
    $a_1 \leftarrow da_1 + \mathrm{T}[i]$
  for $i \leftarrow 1$ **to** n-m+1
    if ($i \neq 1$) **then** $a_i \leftarrow d(a_{i-1} - d^{m-1}\mathrm{T}[i\text{-}1]) + \mathrm{T}[i+m\text{-}1]$
    if ($p = a_i$) **then** output "occurrence at i"

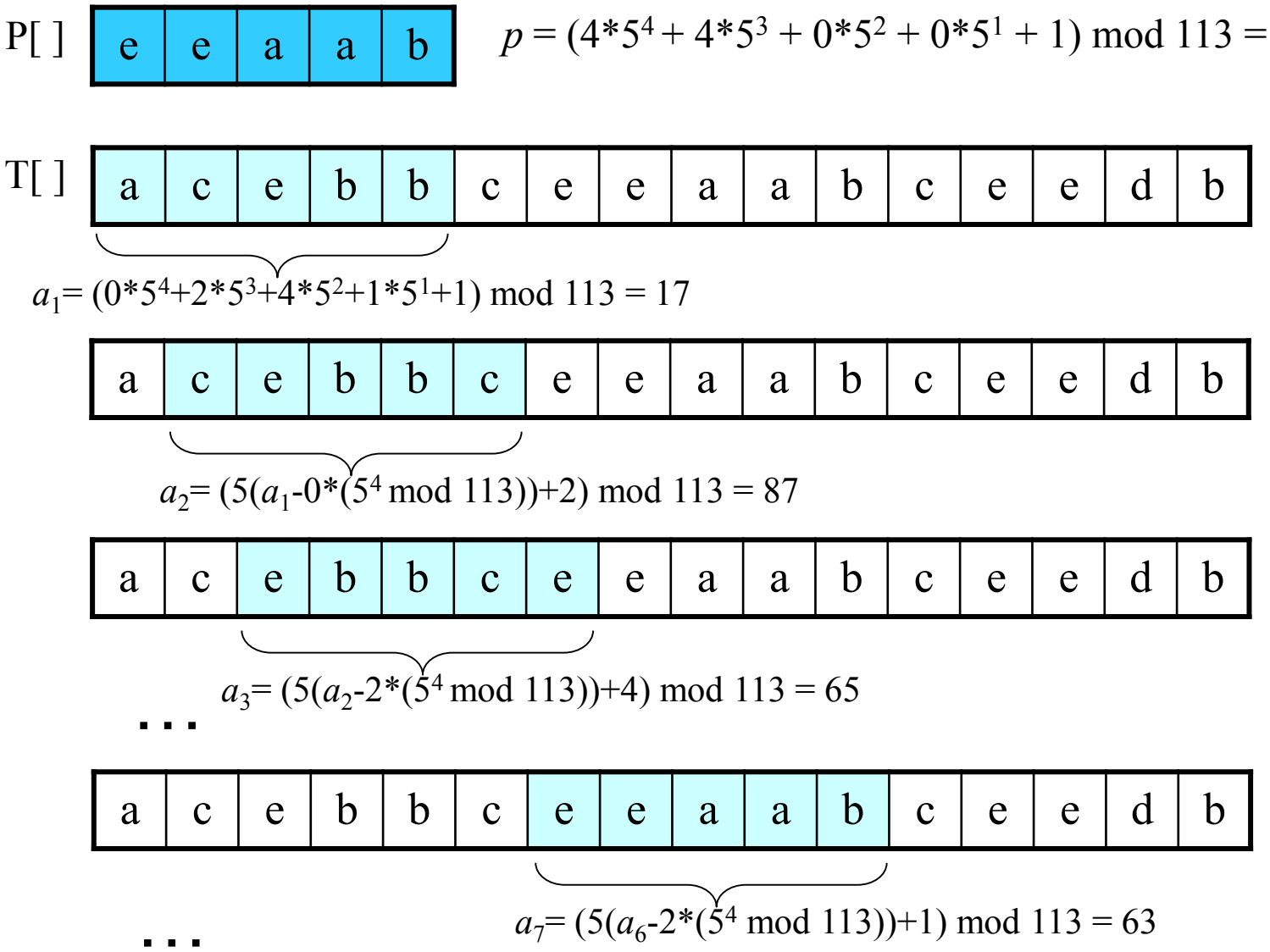
✓ Time complexity: $\Theta(n)$

Too Large Number

- Number a_i may be too large, depending on Σ and m
  - There may be an overflow if it exceeds word size
- Solution
  - Use modulo operation to limit a_i
  - Instead of $a_i = d(a_{i-1} - d^{m-1}\mathrm{T}[i-1]) + \mathrm{T}[i+m-1]$,

    use $b_i = (d(b_{i-1} - (d^{m-1} \bmod q)\,\mathrm{T}[i-1]) + \mathrm{T}[i+m-1]) \bmod q$
  - Choose a big prime as q such that dq fits within one word

Rabin-Karp Algorithm



Rabin-Karp Algorithm

```
RabinKarp(P, T, d, q)
▷ P[1…m], T[1…n]
    p ← 0; b_1 ← 0
    for i ← 1 to m                          ▷ compute b_1
        p ← (dp + P[i]) mod q
        b_1 ← (db_1 + T[i]) mod q
    h ← d^(m-1) mod q
    for i ← 1 to n-m+1
        if (i ≠ 1) then b_i ← (d(b_(i-1) – hT[i-1]) + T[i+m-1]) mod q
        if (p = b_i) then
            if (P[1…m] = T[i…i+m-1]) then
                output "occurrence at i"
```

✓ average time: $\Theta(n)$

Thank you

Lecture plan

Today, we will

- Study how a simple AHB system works
- Make a custom AHB slave IP
- Add a new AHB slave to an AHB system

- Experiments
  - Ex1: AHB master/slave with a single read/write mode (ALU IP)
  - Ex2: Multiplier IP
  - Ex3: AHB decoder
  - Ex4: Bus interconnect

Road map

RISC-V Summary

AMBA AHB Bus
(ALU IP)

Multiplier IP

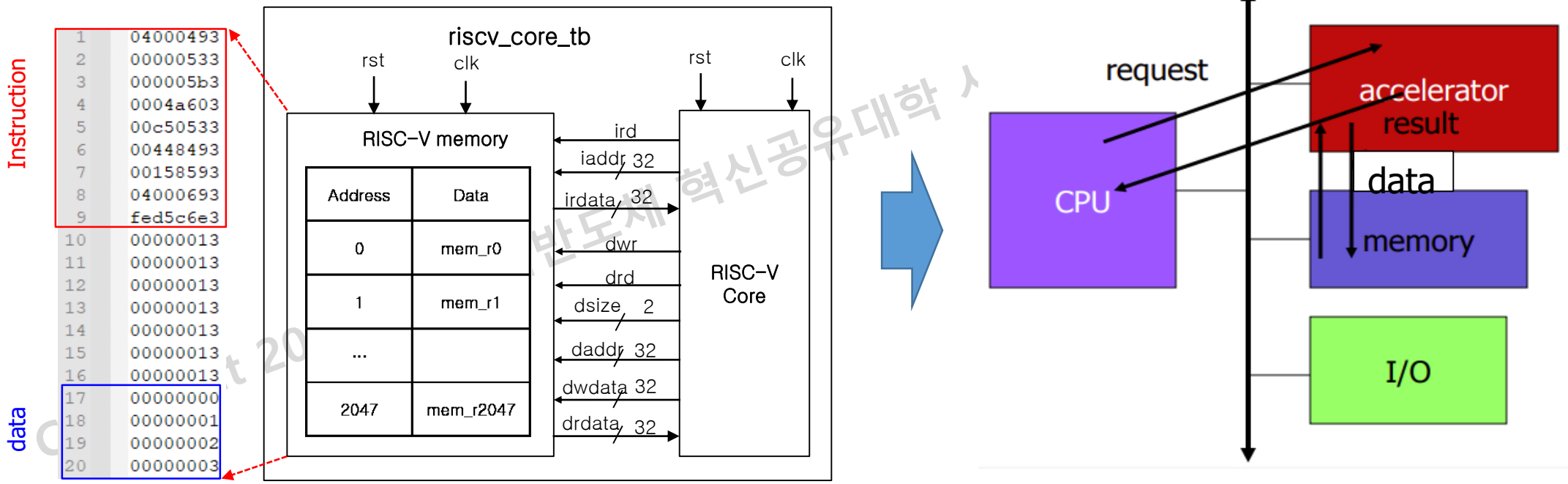
AHB Decoder

Bus Interconnect

Motivation

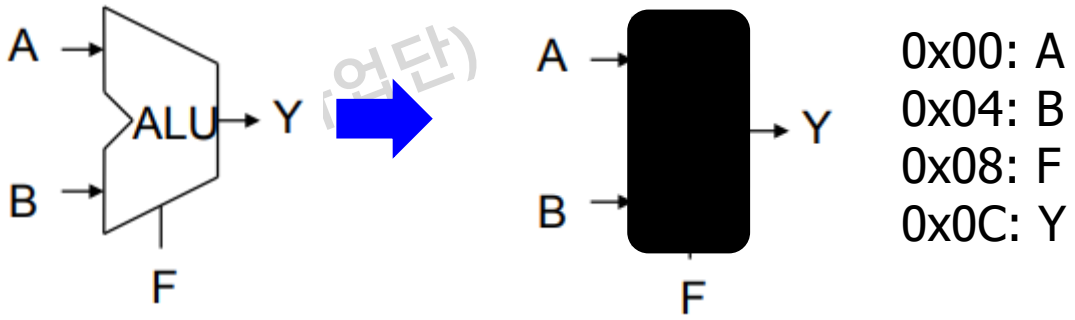
- CPU and Memory are directly connected without using a bus
- For a system with CPU, memory, IO, and other IPs, many connections are required

⇒ Require a protocol to connect all components



Motivation: Why BUS?

- Provide a common way to connect all components in an SoC system
  - **Pack an IP** with a **"standard" interface**
  - **Re-use an IP** for many applications
- Protect the copyright of an IP
  - **Mask** its detailed implementation



- Example: Arithmetic Logic Unit
  - Y = F(A,B)

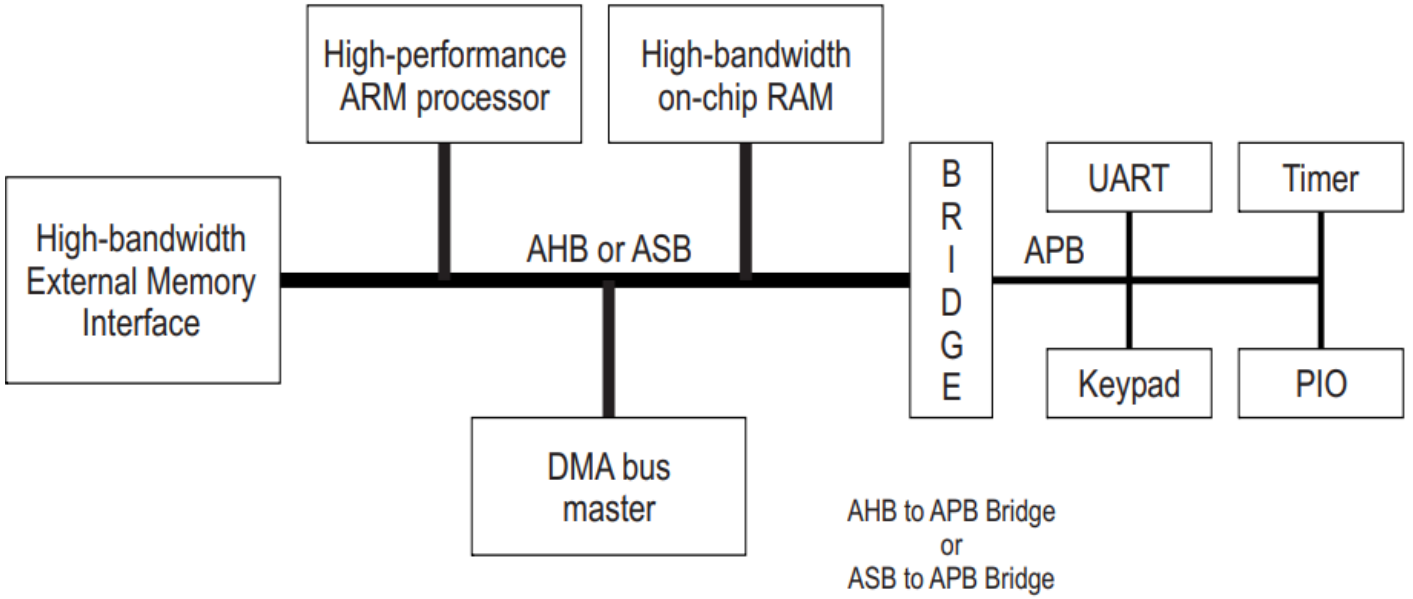
⇒ We can mask all the detail implementation and only provide the register map

About AMBA specification

- The **A**dvanced **M**icrocontroller **B**us **A**rchitecture (AMBA) specification
  - Defines an on-chip communications standard for designing high-performance embedded microcontrollers.

- Three distinct buses are defined within the AMBA specification:
  - The Advanced High-performance Bus (AHB)
  - The Advanced System Bus (ASB)
  - The Advanced Peripheral Bus (APB).

Typical AMBA bus system



AMBA AHB

* High performance
* Pipelined operation
* Multiple bus masters
* Burst transfers
* Split transactions

AMBA ASB

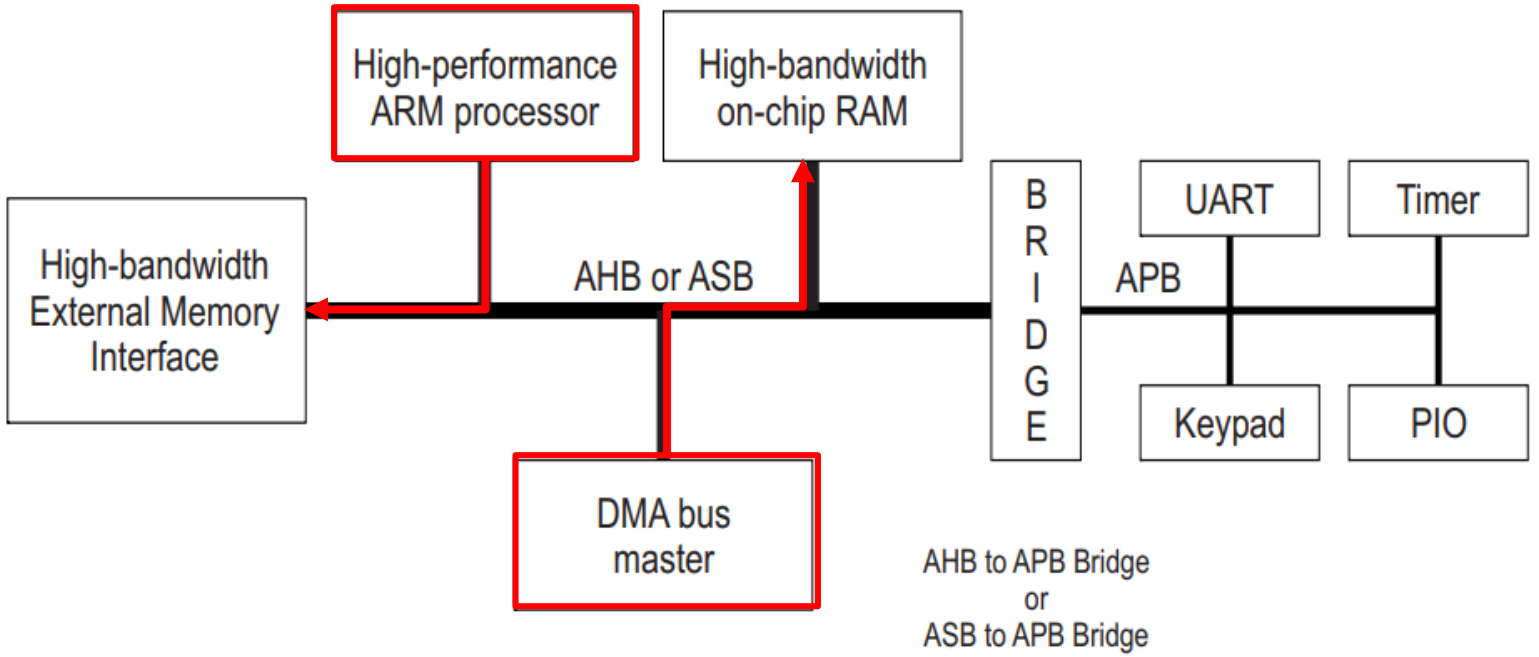
* High performance
* Pipelined operation
* Multiple bus masters

AMBA APB

* Low power
* Latched address and control
* Simple interface
* Suitable for many peripherals

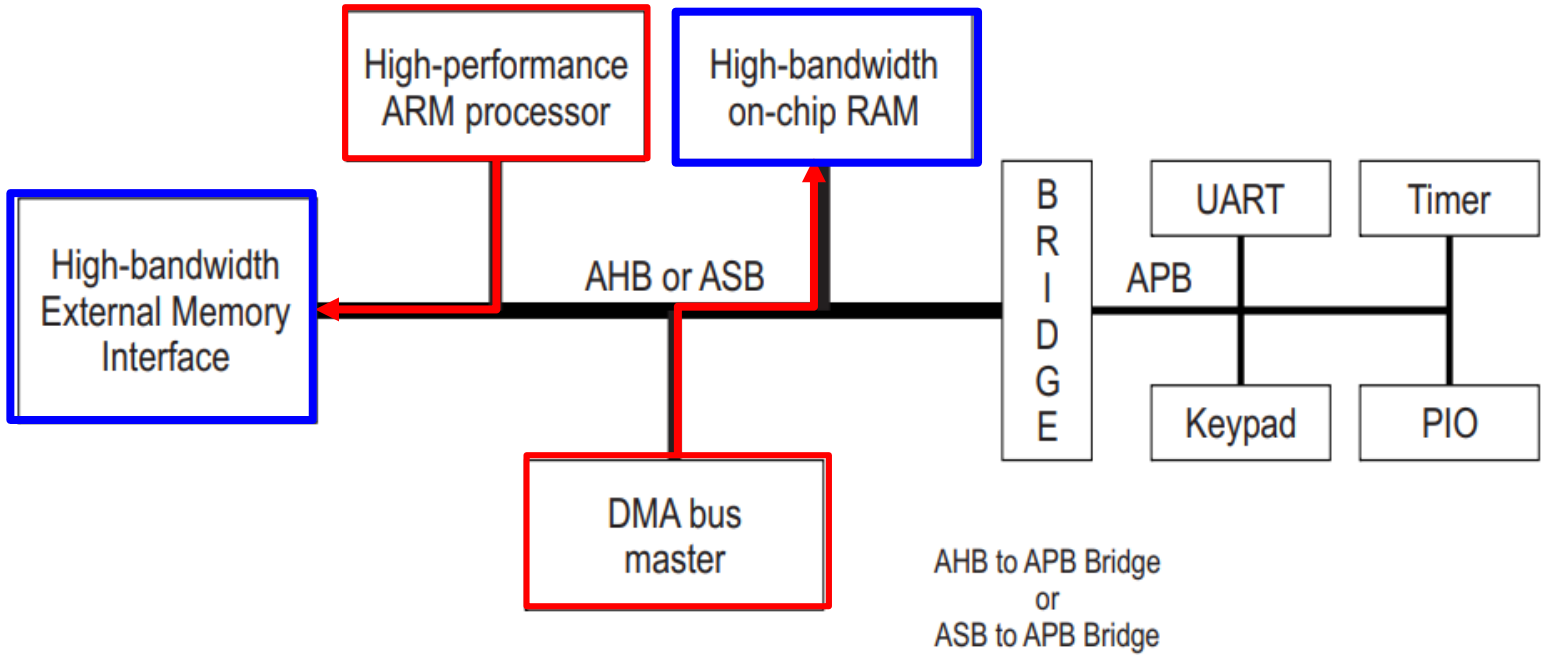
Master

- A Bus master is able to initiate read and write operations by providing an address and control info.
- Only one bus master is allowed to actively use the bus at any one time.
- Example: Processor or Direct Memory Access (DMA)



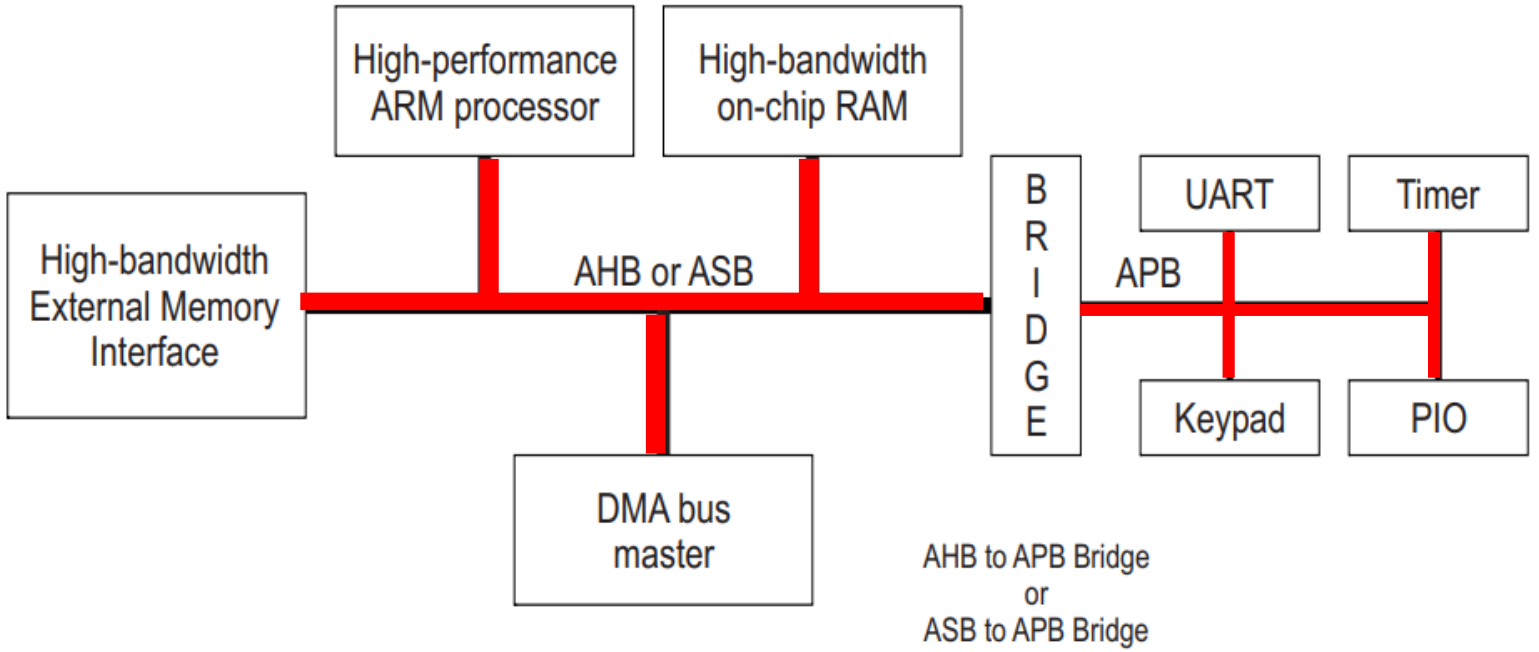
Slave

- A bus slave **responds to a read or write operation** within **a given address-space range**.
- The bus slave signals back to the active master the success, failure or waiting of the data transfer.
- Example: Memory modules



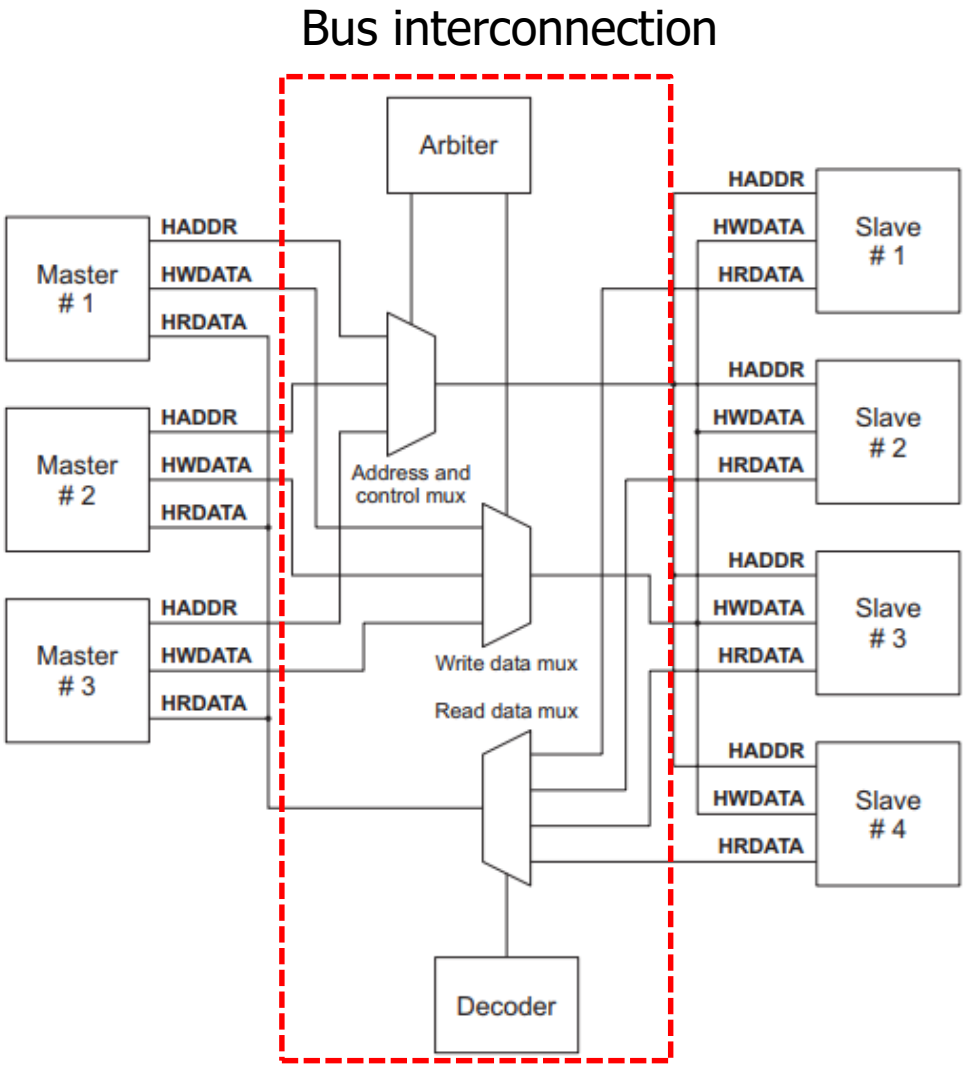
Bus Interconnection

- Bus interconnection: connects multiple masters and multiple slaves.



Bus Interconnection

- Bus interconnection
  - Address and control multiplexer (mux)
  - Write data mux
  - Read data mux
- AHB Decoder: The bus decoder performs the **decoding of the transfer addresses and selects slaves appropriately**.
- AHB Arbiter: The bus arbiter **ensures that only one bus master at a time is allowed to initiate data transfers.**



AMBA Signals (1/3)

- HADDR[31:0]: 32-bit system address bus.
- HWRITE: 0 = READ transfer, 1 = Write transfer
- HTRANS: Type of transfers
  - IDLE, NONSEQ, SEQ, BUSY

Name	Source	Description
HCLK Bus clock	Clock source	This clock times all bus transfers. All signal timings are related to the rising edge of HCLK.
HRESETn Reset	Reset controller	The bus reset signal is active LOW and is used to reset the system and the bus. This is the only active LOW signal.
HADDR[31:0] Address bus	Master	The 32-bit system address bus.
HTRANS[1:0] Transfer type	Master	Indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.
HWRITE Transfer direction	Master	When HIGH this signal indicates a write transfer and when LOW a read transfer.

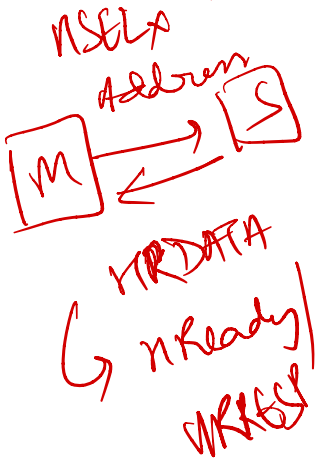
AMBA Signals (2/3)

- HWDATA[31:0]: the ***write data bus*** transfers data from a master to slaves during a write operation.

HPROT[3:0] Protection control	Master	The protection control signals provide additional information about a bus access and are primarily intended for use by any module that wishes to implement some level of protection. The signals indicate if the transfer is an opcode fetch or data access, as well as if the transfer is a privileged mode access or user mode access. For bus masters with a memory management unit these signals also indicate whether the current access is cacheable or bufferable.
HWDATA[31:0] Write data bus	Master	The write data bus is used to transfer data from the master to the bus slaves during write operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.

AMBA Signals (3/3)

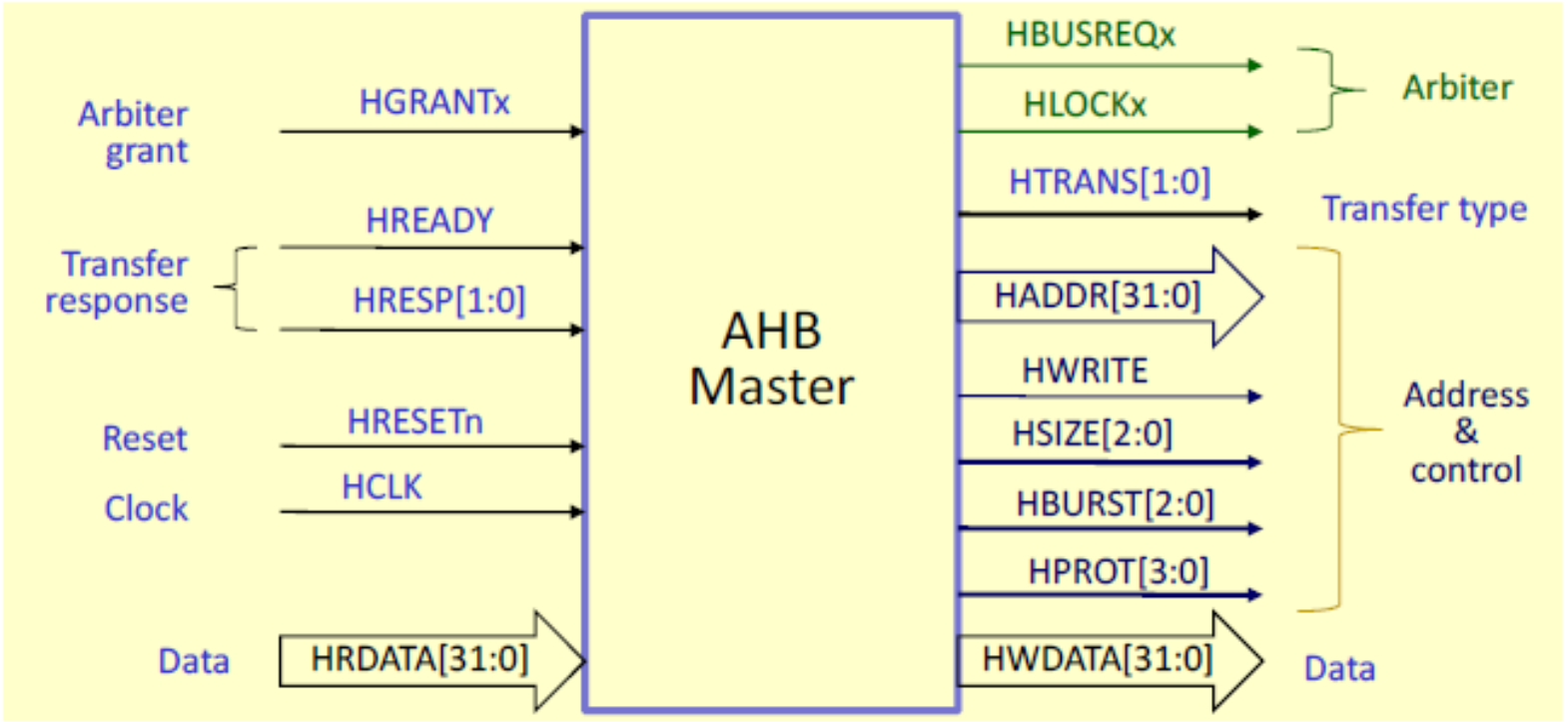
- HSELx: each AHB slave has its own slave select signal, indicating that the current transfer is intended for the selected slave.
- HRDATA[31:0]: The ***read data bus*** from bus slaves to a bus master during a read operation.



Name	Source	Description
HSELx Slave select	Decoder	Each AHB slave has its own slave select signal and this signal indicates that the current transfer is intended for the selected slave. This signal is simply a combinatorial decode of the address bus.
HRDATA[31:0] Read data bus	Slave	The read data bus is used to transfer data from bus slaves to the bus master during read operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HREADY Transfer done	Slave	When HIGH the **HREADY** signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer. Note: Slaves on the bus require **HREADY** as both an input and an output signal.
HRESP[1:0] Transfer response	Slave	The transfer response provides additional information on the status of a transfer. Four different responses are provided, OKAY, ERROR, RETRY and SPLIT.

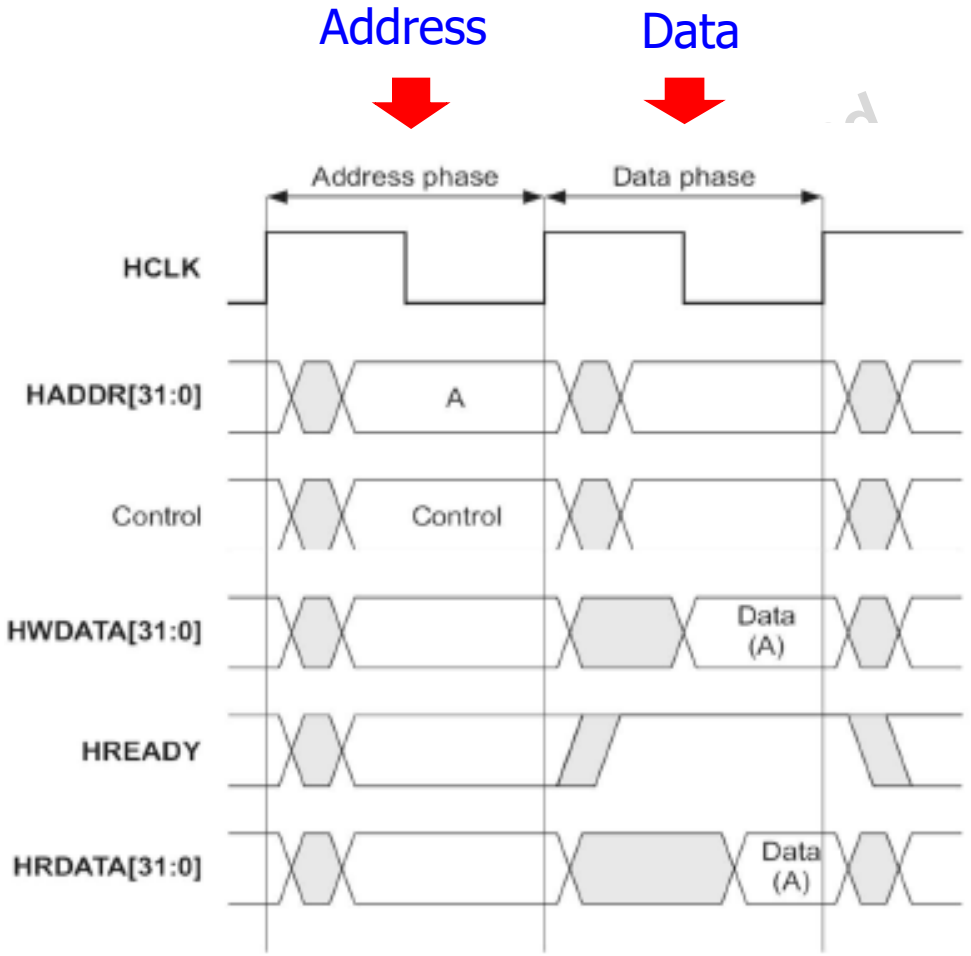
Example: an AHB Master

- AHB Master can start read/write operation
- Read/Write data: HRDATA, HWDATA, address: HADDR
- Control signals: HWRITE, HSIZE, HBURST, HTRANS,…



AMBA AHB transfer

- **An AHB transfer consists of two phases**
  - Address phase: which takes one cycles
    - Determine an address that a master wants to read or write
  - Data phase: requires several cycles
    - HREADY signal is used

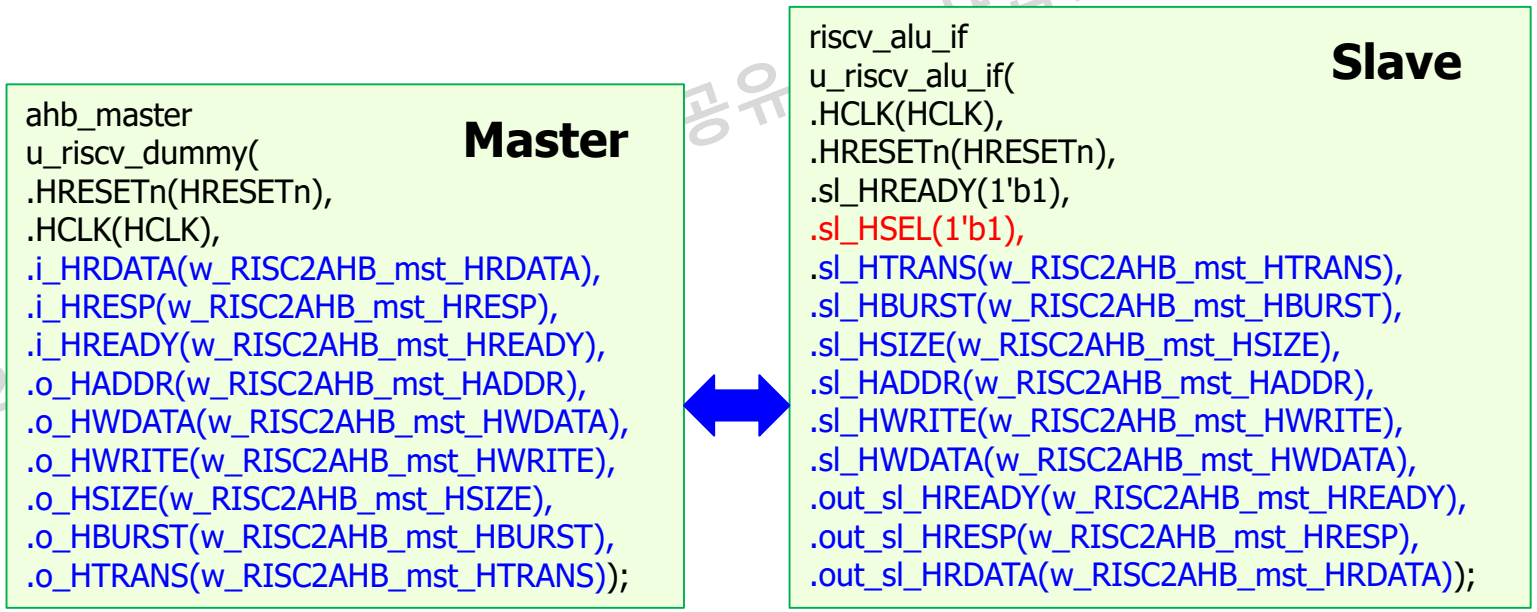


Lab 1: ALU IP

- Lab 1: ALU IP
  - Implement an AHB slave interface
    - **riscv_alu_if.v**
  - Do a simulation with time = 1,600 ns
  - Show the output waveform
  - How to verify the result?
    - Test ALU operations
    - Check a waveform

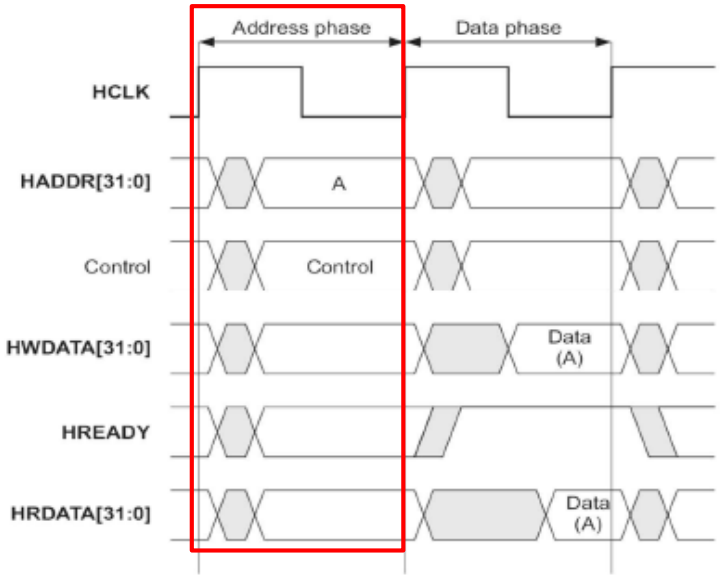
Target system (riscv_alu_if_tb.v)

- A simple system has **one AHB master (RISV-V dummy) and one AHB slave (ALU)**
  - AHB ports of the master and slave are connected directly.
  - The slave is always selected by the master, i.e., sl_HSEL = 1'b1.



AHB Master (ahb_master.v): Write operation

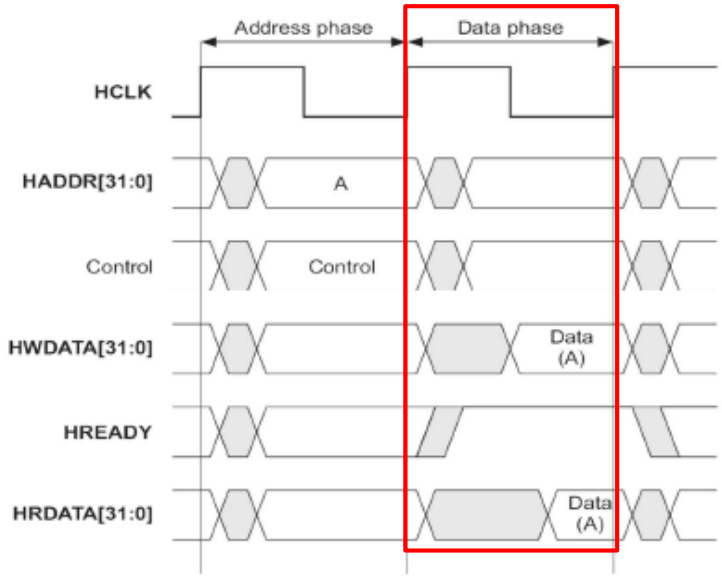
- Address phase
  - Set an output address (o_HADDR)
  - Set o_HTRANS to NONSEQUENTIAL
  - Set o_HBUSRT to SINGLE
  - Set o_HSIZE to 32 bit mode (WORD)
  - Set o_HWRITE to ONE




```
task task_AHBwrite;
    input   [31:0]  i_addr;
    input   [31:0]  i_wData;

    begin
        // Address phase
        @(posedge HCLK);    //#p;
        while(!w_HREADY)    @(posedge HCLK);
        @(posedge HCLK);    //#p;

        o_HADDR     = i_addr;
        o_HTRANS    = TRANS_NONSEQ;
        o_HSIZE     = SZ_WORD;
        o_HBURST    = BUR_SINGLE;
        o_HWRITE    = 1'b1;

        // Data phase
        while(!w_HREADY)    @(posedge HCLK);
        @(posedge HCLK);    //#p;

        o_HADDR     =   32'h00000000;
        o_HWDATA    =   i_wData;
        o_HTRANS    =   2'b00;
        o_HSIZE     =   3'b000;
        o_HBURST    =   3'b000;
        o_HWRITE    =   1'b0;

        @(posedge HCLK);    //#p;
        while(!w_HREADY)    @(posedge HCLK)
        @(posedge HCLK);    //#p;

        o_HWDATA    =   32'h00000000;
    end
endtask
```

AHB Master (ahb_master.v): Write operation

- Data phase
  - Reset an output address (o_HADDR)
  - Reset o_HTRANS
  - Reset o_HBUSRT
  - Reset o_HSIZE
  - Reset o_HWRITE
  - **Set the output data (o_HWDATA)**




```
task task_AHBwrite;
    input   [31:0]  i_addr;
    input   [31:0]  i_wData;

    begin
        // Address phase
        @(posedge HCLK);     //#p;
        while(!w_HREADY)     @(posedge HCLK);
        @(posedge HCLK);     //#p;

        o_HADDR     = i_addr;
        o_HTRANS    = TRANS_NONSEQ;
        o_HSIZE     = SZ_WORD;
        o_HBURST    = BUR_SINGLE;
        o_HWRITE    = 1'b1;

        // Data phase
        while(!w_HREADY)     @(posedge HCLK);
        @(posedge HCLK);     //#p;

        o_HADDR     =   32'h00000000;
        o_HWDATA    =   i_wData;
        o_HTRANS    =   2'b00;
        o_HSIZE     =   3'b000;
        o_HBURST    =   3'b000;
        o_HWRITE    =   1'b0;

        @(posedge HCLK);     //#p;
        while(!w_HREADY)     @(posedge HCLK)
        @(posedge HCLK);     //#p;

        o_HWDATA    =   32'h00000000;
    end
endtask
```

AHB Master (ahb_master.v): Read operation

- Address phase
  - Set an output address (o_HADDR)
  - Set o_HTRANS to NONSEQUENTIAL
  - Set o_HBUSRT to SINGLE
  - Set o_HSIZE to 32 bit mode (WORD)
  - Set o_HWRITE to ZERO

```
task task_AHBread;
    input   [31:0]  i_addr;
    output  [31:0]  o_rData;

    begin
        @(posedge HCLK);    //#p;
        while(!w_HREADY)    @(posedge HCLK);
        @(posedge HCLK);    //#p;

        o_HADDR     =   i_addr;
        o_HTRANS    =   TRANS_NONSEQ;
        o_HSIZE     =   SZ_WORD;
        o_HBURST    =   BUR_SINGLE;
        o_HWRITE    =   1'b0;

        while(!w_HREADY)    @(posedge HCLK);
        @(posedge HCLK);    //#p;

        o_HADDR     =   32'h00000000;
        o_HTRANS    =   2'b00;
        o_HSIZE     =   3'b000;
        o_HBURST    =   3'b000;

        @(posedge HCLK);    //#p;
        while(!w_HREADY)    @(posedge HCLK);
        o_rData     =   w_HRDATA;
    end
endtask
```

AHB Master (ahb_master.v): Read operation

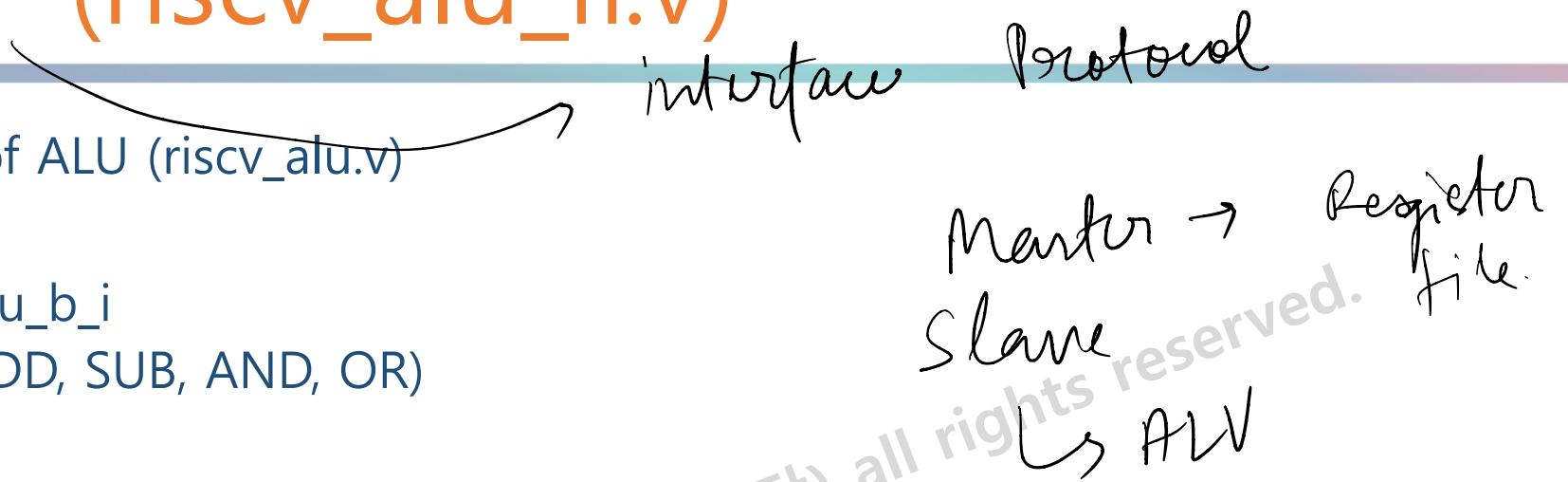
- Data phase
  - Reset an output address (o_HADDR)
  - Reset o_HTRANS
  - Reset o_HBUSRT
  - Reset o_HSIZE
  - Reset o_HWRITE
  - **Get the output data (o_HRDATA)**

```
task task_AHBread;
    input   [31:0]  i_addr;
    output  [31:0]  o_rData;

    begin
        @(posedge HCLK);    //#p;
        while(!w_HREADY)    @(posedge HCLK);
        @(posedge HCLK);    //#p;

        o_HADDR     =   i_addr;
        o_HTRANS    =   TRANS_NONSEQ;
        o_HSIZE     =   SZ_WORD;
        o_HBURST    =   BUR_SINGLE;
        o_HWRITE    =   1'b0;

        while(!w_HREADY)    @(posedge HCLK);
        @(posedge HCLK);    //#p;

        o_HADDR     =   32'h00000000;
        o_HTRANS    =   2'b00;
        o_HSIZE     =   3'b000;
        o_HBURST    =   3'b000;

        @(posedge HCLK);    //#p;
        while(!w_HREADY)    @(posedge HCLK);
        o_rData     =   w_HRDATA;
    end
endtask
```

↳ for the register

AHB slave IP (riscv_alu_if.v)

- Let's consider the case of ALU (riscv_alu.v)
- Inputs
  - Operands: alu_a_i, alu_b_i
  - Opcode: alu_op_i (ADD, SUB, AND, OR)
- Output
  - Result: alu_p_o
- We will **make an AHB interface for ALU**



AHB Slave (riscv_alu_if.v)

- Step 1: Build a register map
  - Define registers by addresses
  - Those addresses are available for an AHB master

```
localparam REG_ALU_OP_I   = 0;                    //0x00
localparam REG_ALU_A_I    = 1;                    //0x04
localparam REG_ALU_B_I    = 2;                    //0x08
localparam REG_ALU_P_O    = /*Insert your code*/; //0x0c  ==> READ ONLY
reg [3:0] alu_op_i;
reg [31:0] alu_a_i, alu_b_i;
wire [31:0] alu_p_o;
```

TODO

- For example
  - Inputs: We can WRITE a value to a register by using its address
    - Operands: alu_a_i (REG_ALU_A_I) , alu_b_i (REG_ALU_B_I)
    - Opcode: alu_op_i (REG_ALU_OP_I)
  - Output: We can READ a value from a register by using its address
    - Result: alu_p_o (REG_ALU_P_O)

AHB Slave (riscv_alu_if.v): Address phase

- Decode stage: Use two registers
  - q_sel_sl_reg: **The address of a register** requested by an AHB master.
  - q_ld_sl_reg: **The READ/WRITE mode** (READ = 0, WRITE= 1)

```
//-----------------------------------------
// Decode Stage: Address Phase
//-----------------------------------------
always @(posedge HCLK or negedge HRESETn)
begin
    if(~HRESETn)
    begin
        //control
        q_sel_sl_reg <= 0;
        q_ld_sl_reg <= 1'b0;
    end
    else begin
        if(sl_HSEL && sl_HREADY && ((sl_HTRANS == `TRANS_NONSEQ) || (sl_HTRANS == `TRANS_SEQ)))
        begin
            q_sel_sl_reg <= sl_HADDR[W_REGS+W_WB_DATA-1:W_WB_DATA];
            q_ld_sl_reg <= sl_HWRITE;
        end
        else begin
            q_ld_sl_reg <= 1'b0;
        end
    end
end
```

Insert code

- Two registers are updated when all the following conditions are satisfied:
  - This AHB Slave is selected by the bus
  - This AHB slave is ready
  - It is a start of a transaction

AHB Slave: Data phase, Write operation

- Write operation
  - q_ld_sl_reg = 1 → write operation
  - Select a target register for update by its address (q_sel_sl_reg)
- For example

If q_sel_sl_reg == REG_ALU_A_I

  - Copy an input data (sl_HWDATA) into alu_a_i

Insert code

```
always @(posedge HCLK or negedge HRESETn)
begin
    if(~HRESETn)
    begin
        //control
        alu_op_i <= 4'h0;
        alu_a_i <= 32'h0;
        alu_b_i <= 32'h0;
    end
    else begin
        //data-transfer state(data phase)
        if(q_ld_sl_reg)
        begin
            case(q_sel_sl_reg)
                REG_ALU_OP_I:
                    alu_op_i <= sl_HWDATA[3:0];
                REG_ALU_A_I:
                    alu_a_i <= sl_HWDATA;
                REG_ALU_B_I:
                    alu_b_i <= sl_HWDATA;
            endcase
        end
    end
end
```

AHB Slave: Data phase, Read operation

- Read operation
  - Select a target register for update by its address (q_sel_sl_reg)
- For example
  - If q_sel_sl_reg == REG_ALU_P_O (output result)
    - Copy alu_p_o to out_sl_HRDATA

```
always @*
begin:rdata
    case(q_sel_sl_reg)
        REG_ALU_OP_I:
            out_sl_HRDATA = alu_op_i;
        REG_ALU_A_I:
            out_sl_HRDATA = alu_a_i;
        REG_ALU_B_I:
            out_sl_HRDATA = alu_b_i;
        REG_ALU_P_O:
            out_sl_HRDATA = alu_p_o;
    endcase
end
```

Insert code

Test bench

```
`timescale 1ns / 100ps
module riscv_alu_if_tb;
        wire [31:0]         w_RISC2AHB_mst_HRDATA;      //**** module output
        wire [1:0]          w_RISC2AHB_mst_HRESP;
        wire                w_RISC2AHB_mst_HREADY;
        wire [31:0]         w_RISC2AHB_mst_HADDR;
        wire [31:0]         w_RISC2AHB_mst_HWDATA;
        wire                w_RISC2AHB_mst_HWRITE;
        wire [2:0]          w_RISC2AHB_mst_HSIZE;
        wire [`W_BURST-1:0] w_RISC2AHB_mst_HBURST;
        wire [1:0]          w_RISC2AHB_mst_HTRANS;

        reg HCLK, HRESETn;                              //**** module input
```

```
ahb_master u_riscv_dummy
(       .HRESETn(HRESETn),
        .HCLK(HCLK),
        .i_HRDATA(w_RISC2AHB_mst_HRDATA),
        .i_HRESP(w_RISC2AHB_mst_HRESP),
        .i_HREADY(w_RISC2AHB_mst_HREADY),
        .o_HADDR(w_RISC2AHB_mst_HADDR),
        .o_HWDATA(w_RISC2AHB_mst_HWDATA),
        .o_HWRITE(w_RISC2AHB_mst_HWRITE),
        .o_HSIZE(w_RISC2AHB_mst_HSIZE),
        .o_HBURST(w_RISC2AHB_mst_HBURST),
        .o_HTRANS(w_RISC2AHB_mst_HTRANS),
);
```

Master module instance for test bench

Test bench

Slave module instance for test bench

```
riscv_alu_if u_riscv_alu_if
(
        .HCLK(HCLK),
        .HRESETn(HRESETn),
        .sl_HREADY(1'b1),
        .sl_HSEL(1'b1),
        .sl_HTRANS(w_RISC2AHB_mst_HTRANS),
        .sl_HBURST(w_RISC2AHB_mst_HBURST),
        .sl_HSIZE(w_RISC2AHB_mst_HSIZE),
        .sl_HADDR(w_RISC2AHB_mst_HADDR),
        .sl_HWRITE(w_RISC2AHB_mst_HWRITE),
        .sl_HWDATA(w_RISC2AHB_mst_HWDATA),
        .out_sl_HREADY(w_RISC2AHB_mst_HREADY),
        .out_sl_HRESP(w_RISC2AHB_mst_HRESP),
        .out_sl_HRDATA(w_RISC2AHB_mst_HRDATA)
);
```

Initialize Inputs

```
initial begin
        HRESETn = 0;

        #(p/2) HRESETn = 1;

        alu_a_i = 0;
        alu_b_i = 0;
        alu_p_o = 0;
```

Address Mapping (riscv_alu_if_tb.v)

- Access internal registers in ALU by
  - Base address (RISCV_ALU_BASE_ADDR)
  - Offset addresses, i.e. 0x00, 0x04, 0x08 and 0x0C
    - ALU opcode at 0x00
    - ALU operands at 0x04, 0x08
    - ALU result at 0x0C

```
//------------------------------------------------------------
// Address mapping
//------------------------------------------------------------
`define RISCV_ALU_BASE_ADDR             32'hE0000000


`define RISCV_ALU_REG_ALU_OP_I   (`RISCV_ALU_BASE_ADDR + 32'h00)          //0x00
`define RISCV_ALU_REG_ALU_A_I    (`RISCV_ALU_BASE_ADDR + 32'h04)          //0x04
`define RISCV_ALU_REG_ALU_B_I    (`RISCV_ALU_BASE_ADDR + 32'h08)          //0x08
`define RISCV_ALU_REG_ALU_P_O    (`RISCV_ALU_BASE_ADDR + 32'h0C)          //0x0c  ==> READ ONLY'
```

Test cases

- Compare two numbers

```
#(8*p)
        alu_a_i = 32'h0;
        alu_b_i = 32'h0;
        alu_op_i = `ALU_SLT;

#(4*p) u_riscv_dummy.task_AHBwrite(`RISCV_ALU_REG_ALU_A_I, alu_a_i);      // Write the first operand
#(4*p) u_riscv_dummy.task_AHBwrite(`RISCV_ALU_REG_ALU_B_I, alu_b_i);      // Write the second operand
#(4*p) u_riscv_dummy.task_AHBwrite(`RISCV_ALU_REG_ALU_OP_I, alu_op_i);  // Write the operation
#(4*p) u_riscv_dummy.task_AHBread(`RISCV_ALU_REG_ALU_P_O, alu_p_o);      // Read the result
```

AHB read and write

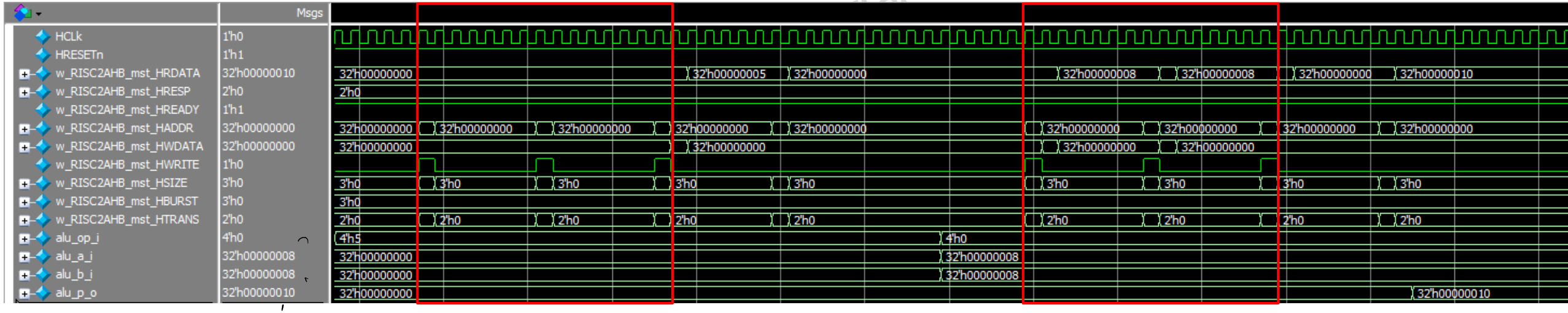
- Add two numbers

```
 #(8*p)
        alu_a_i = 32'h8;
        alu_b_i = 32'h8;
        alu_op_i = `ALU_ADD;
#(4*p) u_riscv_dummy.task_AHBwrite(`RISCV_ALU_REG_ALU_A_I, alu_a_i);      // Write the first operand
#(4*p) u_riscv_dummy.task_AHBwrite(`RISCV_ALU_REG_ALU_B_I, alu_b_i);      // Write the second operand
#(4*p) u_riscv_dummy.task_AHBwrite(`RISCV_ALU_REG_ALU_OP_I, alu_op_i);  // Write the operation
#(4*p) u_riscv_dummy.task_AHBread(`RISCV_ALU_REG_ALU_P_O, alu_p_o);      // Read the result
```

AHB read and write

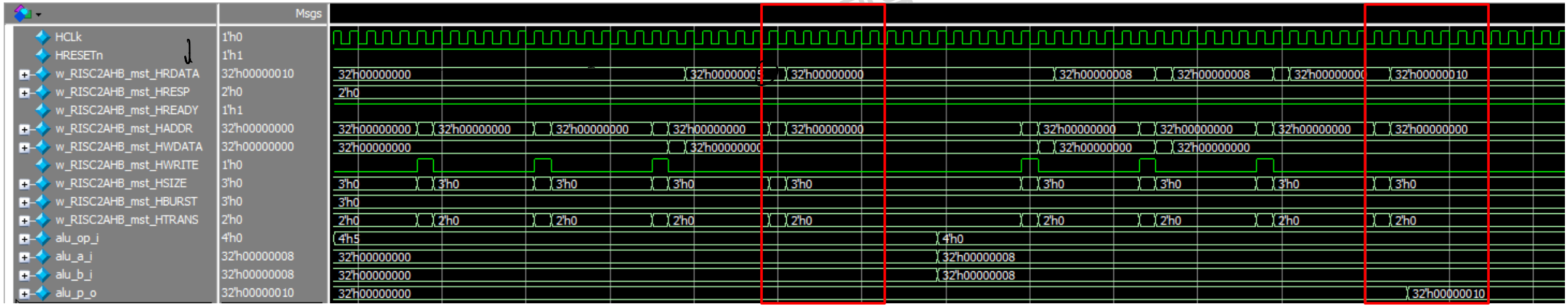
Waveform: Write operations

- SLT or ADD operation
  - Write to alu_a_i, alu_b_i, alu_op_i
  - Read the result to alu_p_o



Waveform: Read operations

- SLT or ADD operation
  - Write to alu_a_i, alu_b_i, alu_op_i
  - Read the result to alu_p_o



To do …

- Implement an AHB slave interface
  - **riscv_alu_if.v**
- Do simulation with time = 1,600ns
- Show the output waveform
- How to verify the result?
  - Test ALU operations
  - Check a waveform

Road map

RISC-V Summary

AMBA AHB Bus
(ALU IP)

Multiplier IP

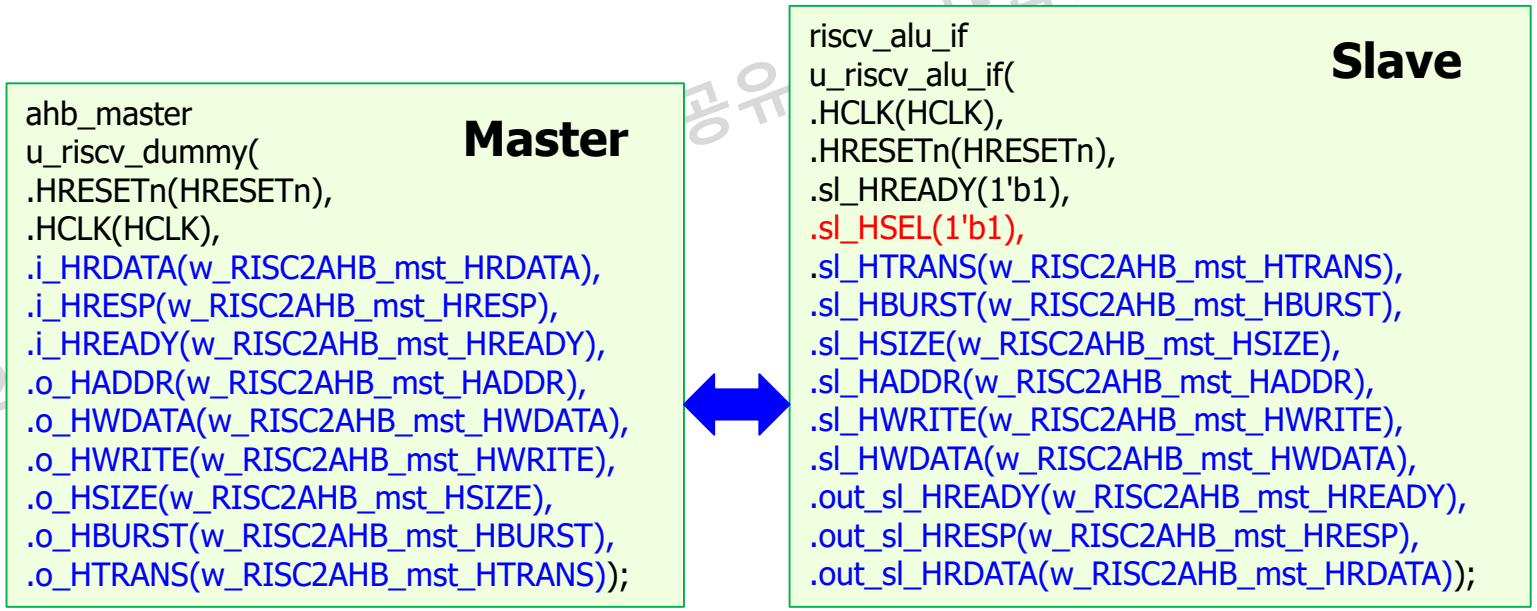
AHB Decoder

Bus Interconnect

Motivation

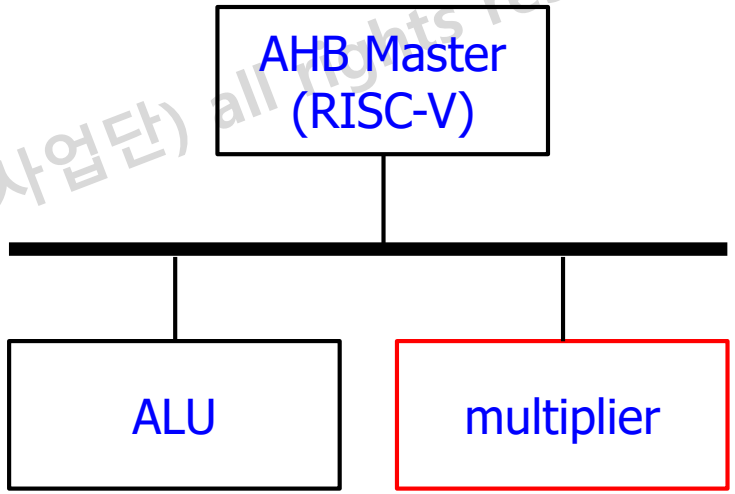
- A simple system has **one AHB master (RISV-V dummy) and one AHB slave (ALU)**
  - AHB ports of the master and slave are connected directly.
  - The slave is always selected by the master, i.e., sl_HSEL = 1'b1.

⇒ In practice, an SoC system usually consists of multiple slaves



Target system

- Build a target system that consists of an AHB master and two AHB slaves
  - Master: RISC-V dummy
  - Slaves: ALU and multiplier
- Two questions:
  - How to make your own AHB slave IP, i.e. multiplier?
  - How to add your own IP into a system?



Lab 2: Multiplier IP

- Lab 2: Multiplier IP
- Build an AHB wrapper of a multiplier and add it to the top system.
  - Complete the missing code
    - **top_system.v & multiplier_if.v & map.v**
  - Do a simulation with time = 3,500ns
  - Show the output waveform

Multiplier IP

- The multiplier wrapper has the same AHB slave ports as the ALU wrapper

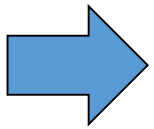
```
module riscv_alu_if #(
    parameter W_ADDR = 32,
    parameter W_DATA = 32,
    parameter WB_DATA = 4,
    parameter W_WB_DATA = 2,
    parameter W_CNT = 16,
    parameter DEF_HPROT = {`PROT_NOTCACHE,
    `PROT_UNBUF, `PROT_USER, `PROT_DATA},
    parameter W_PIX = 8)
(
    //CLOCK
    HCLK,
    HRESETn,
    //input signals of control port(slave)
    sl_HREADY,
    sl_HSEL,
    sl_HTRANS,
    sl_HBURST,
    sl_HSIZE,
    sl_HADDR,
    sl_HWRITE,
    sl_HWDATA,
    //output signals of control port(slave)
    out_sl_HREADY,
    out_sl_HRESP,
    out_sl_HRDATA
);
```

```
module riscv_multiplier_if #(
    parameter W_ADDR = 32,
    parameter W_DATA = 32,
    parameter WB_DATA = 4,
    parameter W_WB_DATA = 2,
    parameter W_CNT = 16,
    parameter DEF_HPROT = {`PROT_NOTCACHE,
    `PROT_UNBUF, `PROT_USER, `PROT_DATA},
    parameter W_PIX = 8)
(
    //CLOCK
    HCLK,
    HRESETn,
    //input signals of control port(slave)
    sl_HREADY,
    sl_HSEL,
    sl_HTRANS,
    sl_HBURST,
    sl_HSIZE,
    sl_HADDR,
    sl_HWRITE,
    sl_HWDATA,
    //output signals of control port(slave)
    out_sl_HREADY,
    out_sl_HRESP,
    out_sl_HRDATA
);
```

Multiplier IP (riscv_multiplier_if.v)

- Step 1: Define a register map
  - Registers correspond to input and output ports of a customized IP.
  - 5 inputs and 3 outputs are mapped to 8 registers.

```
riscv_multiplier
u_multiplier(
./*input */clk_i(HCLK),
./*input */reset_i(HRESETn),
./*input [3:0]  */id_alu_op_r(alu_op_i),
./*input        */id_a_signed_r(a_signed),
./*input        */id_b_signed_r(b_signed),
./*input [31:0] */id_ra_value_r(alu_a_i),
./*input [31:0] */id_rb_value_r(alu_b_i),
./*output [63:0] */mul_res_w(alu_p_o),
./*output       */ex_stall_mul_w(ex_stall_mul_w)
);
```



```
localparam N_REGS = 8;  // Number of registers
localparam W_REGS = 3;  //log2(N_REGS)

localparam REG_MUL_OP_I      = 0;    //0x00
localparam REG_MUL_A_SIGNED  = 1;    //0x04
localparam REG_MUL_B_SIGNED  = 2;    //0x08
localparam REG_MUL_A_I       = 3;    //0x0c
localparam REG_MUL_B_I       = 4;    //0x08
localparam REG_MUL_P_O_LO    = 5;    //0x0c  <READ-ONLY>
localparam REG_MUL_P_O_HI    = 6;    //0x08  <READ-ONLY>
localparam REG_MUL_EX_STALL  = 7;    //0x08  <READ-ONLY>
```

Address phase (riscv_multiplier_if.v)

- Step 2: Decode an access address given by a master
  - Which register does a master want to access?

```
always @(posedge HCLK or negedge HRESETn)
begin
    if(~HRESETn)
    begin
        //control
        q_sel_sl_reg <= 0;
        q_ld_sl_reg <= 1'b0;
    end
    else begin
        if(sl_HSEL && sl_HREADY && ((sl_HTRANS == `TRANS_NONSEQ) || (sl_HTRANS == `TRANS_SEQ)))
        begin
            q_sel_sl_reg <= sl_HADDR[/*Insert your code*/];
            q_ld_sl_reg <= /*Insert your code*/;
        end
        else begin
            q_ld_sl_reg <= 1'b0;
        end
    end
end
```

TODO

Data phase (riscv_multiplier_if.v)

- Step 3: Write operation
  - This IP receives data from a Master
  - Registers are updated by using a given address from a master

```
always @(posedge HCLK or negedge HRESETn)
begin
    if(~HRESETn)
    begin
        //control
        alu_op_i <= 4'h0;
        alu_a_i <= 32'h0;
        alu_b_i <= 32'h0;
        a_signed <= 1'b0;
        b_signed <= 1'b0;
    end
    else begin
        //data-transfer state(data phase)
        if(q_ld_sl_reg == 1)
        begin
            case(q_sel_sl_reg)
                REG_MUL_OP_I: alu_op_i <= sl_HWDATA;
                REG_MUL_A_I: alu_a_i <= sl_HWDATA;
                REG_MUL_B_I: alu_b_i <= /*Insert your code*/;
                REG_MUL_A_SIGNED: a_signed <= sl_HWDATA;
                REG_MUL_B_SIGNED: b_signed <= /*Insert your code*/;
            endcase
        end
    end
end
```

TODO

Data phase (riscv_multiplier_if.v)

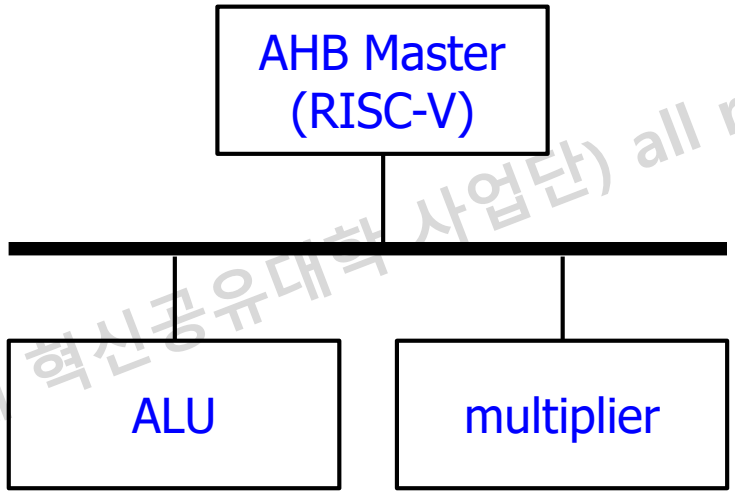
- Step 4: Read operation
  - The slave returns data to the Master.
  - Make an output data based on its request address.

```
always @*
begin:rdata
    out_sl_HRDATA = 32'b0;
    case(q_sel_sl_reg)
        REG_MUL_OP_I: out_sl_HRDATA = alu_op_i;
        REG_MUL_A_I: out_sl_HRDATA = alu_a_i;
        REG_MUL_B_I: out_sl_HRDATA = /*Insert your code*/;
        REG_MUL_A_SIGNED: out_sl_HRDATA = a_signed;
        REG_MUL_B_SIGNED: out_sl_HRDATA = /*Insert your code*/;
        REG_MUL_P_O_LO: out_sl_HRDATA = alu_p_o[31:0];
        REG_MUL_P_O_HI: out_sl_HRDATA =  alu_p_o[63:32];
        REG_MUL_EX_STALL: out_sl_HRDATA = ex_stall_mul_w;
    endcase
end
```

TODO

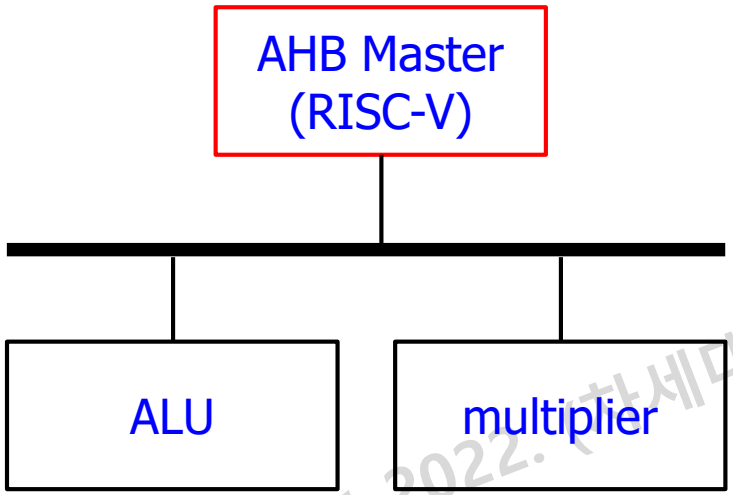
Build a top module (top_system.v)

- How to add your own IP into the system?
- A top module consists of:
  - 1 master:
  - 2 slaves: alu, multiplier



AHB Interface for Master

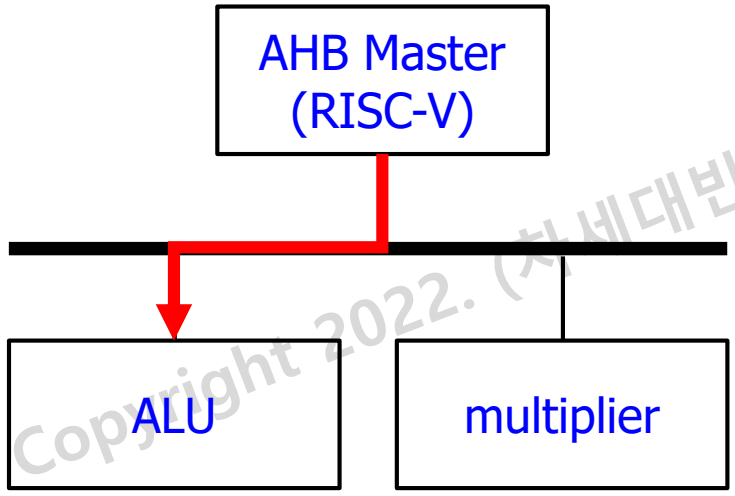
- Master is as that of the previous example.




```
//------------------------------------------------------
// Master
//------------------------------------------------------
ahb_master u_riscv_dummy(
     .HRESETn          (HRESETn                )
    ,.HCLK             (HCLK                   )
    ,.i_HRDATA         (w_RISC2AHB_mst_HRDATA      )
    ,.i_HRESP          (w_RISC2AHB_mst_HRESP       )
    ,.i_HREADY         (w_RISC2AHB_mst_HREADY      )
    ,.o_HADDR          (w_RISC2AHB_mst_HADDR       )
    ,.o_HWDATA         (w_RISC2AHB_mst_HWDATA      )
    ,.o_HWRITE         (w_RISC2AHB_mst_HWRITE      )
    ,.o_HSIZE          (w_RISC2AHB_mst_HSIZE       )
    ,.o_HBURST         (w_RISC2AHB_mst_HBURST      )
    ,.o_HTRANS         (w_RISC2AHB_mst_HTRANS      )
    );
//
```

ALU IP

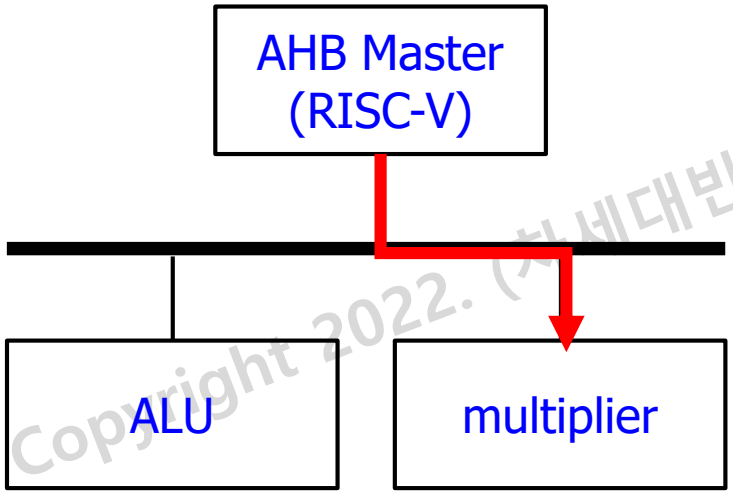
- The input ports of ALU_if are directly connected to Master.
- But ALU has its own output ports
- There is a separate select signal for an interface
  - Determine when ALU slave is accessed by a Master




```
riscv_alu_if u_riscv_alu_if (
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .sl_HREADY(1'b1),
    .sl_HSEL(sl_HSEL_alu),
    .sl_HTRANS(w_RISC2AHB_mst_HTRANS),
    .sl_HBURST(w_RISC2AHB_mst_HBURST),
    .sl_HSIZE(w_RISC2AHB_mst_HSIZE),
    .sl_HADDR(w_RISC2AHB_mst_HADDR),
    .sl_HWRITE(w_RISC2AHB_mst_HWRITE),
    .sl_HWDATA(w_RISC2AHB_mst_HWDATA),
    .out_sl_HREADY(w_RISC2AHB_alu_HREADY),
    .out_sl_HRESP( w_RISC2AHB_alu_HRESP),
    .out_sl_HRDATA(w_RISC2AHB_alu_HRDATA)
    );
```

Multiplier IP

- The input ports of the Multiplier interface are directly connected to the Master.
- But Multiplier has its output ports
- There is a separate select signal for an interface
  - Determine when the Multiplier slave is accessed by a Master




```
multiplier_if u_multiplier_if (
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .sl_HREADY(1'b1),
    .sl_HSEL(sl_HSEL_multiplier),
    .sl_HTRANS(w_RISC2AHB_mst_HTRANS),
    .sl_HBURST(w_RISC2AHB_mst_HBURST),
    .sl_HSIZE(w_RISC2AHB_mst_HSIZE),
    .sl_HADDR(w_RISC2AHB_mst_HADDR),
    .sl_HWRITE(w_RISC2AHB_mst_HWRITE),
    .sl_HWDATA(w_RISC2AHB_mst_HWDATA),
    .out_sl_HREADY(w_RISC2AHB_mul_HREADY),
    .out_sl_HRESP( w_RISC2AHB_mul_HRESP),
    .out_sl_HRDATA(w_RISC2AHB_mul_HRDATA)
  );
```

AHB Interface for ALU/Multiplier

- The input ports of ALU and Multiplier are directly connected to Master.
- But ALU and Multiplier have their own output ports
  - Why?

```
// ALU
riscv_alu_if u_riscv_alu_if (
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .sl_HREADY(1'b1),
    .sl_HSEL(sl_HSEL_alu),
    .sl_HTRANS(w_RISC2AHB_mst_HTRANS),
    .sl_HBURST(w_RISC2AHB_mst_HBURST),
    .sl_HSIZE(w_RISC2AHB_mst_HSIZE),
    .sl_HADDR(w_RISC2AHB_mst_HADDR),
    .sl_HWRITE(w_RISC2AHB_mst_HWRITE),
    .sl_HWDATA(w_RISC2AHB_mst_HWDATA),
    .out_sl_HREADY(w_RISC2AHB_alu_HREADY),
    .out_sl_HRESP( w_RISC2AHB_alu_HRESP),
    .out_sl_HRDATA(w_RISC2AHB_alu_HRDATA)
    );
```

```
// Multiplier
riscv_multiplier_if u_riscv_multiplier_if (
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .sl_HREADY(1'b1),
    .sl_HSEL(sl_HSEL_multiplier),
    .sl_HTRANS(w_RISC2AHB_mst_HTRANS),
    .sl_HBURST(w_RISC2AHB_mst_HBURST),
    .sl_HSIZE(w_RISC2AHB_mst_HSIZE),
    .sl_HADDR(w_RISC2AHB_mst_HADDR),
    .sl_HWRITE(w_RISC2AHB_mst_HWRITE),
    .sl_HWDATA(w_RISC2AHB_mst_HWDATA),
    .out_sl_HREADY(w_RISC2AHB_mul_HREADY),
    .out_sl_HRESP( w_RISC2AHB_mul_HRESP),
    .out_sl_HRDATA(w_RISC2AHB_mul_HRDATA)
    );
```

AHB Interface for ALU/Multiplier

- The input ports of ALU and Multiplier are directly connected to Master.
- But ALU and Multiplier have their own output ports

  ⇒ A master can only access one of them, i.e. ALU or multiplier.

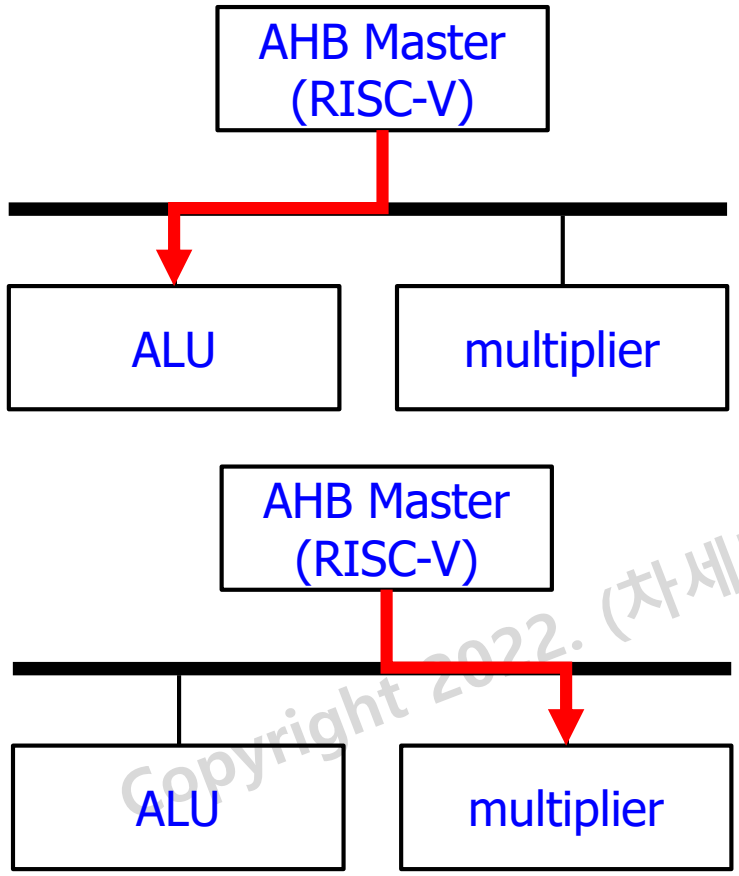
Master communicates with everyone but slave only communicate with the master

```
// ALU
riscv_alu_if u_riscv_alu_if (
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .sl_HREADY(1'b1),
    .sl_HSEL(sl_HSEL_alu),
    .sl_HTRANS(w_RISC2AHB_mst_HTRANS),
    .sl_HBURST(w_RISC2AHB_mst_HBURST),
    .sl_HSIZE(w_RISC2AHB_mst_HSIZE),
    .sl_HADDR(w_RISC2AHB_mst_HADDR),
    .sl_HWRITE(w_RISC2AHB_mst_HWRITE),
    .sl_HWDATA(w_RISC2AHB_mst_HWDATA),
    .out_sl_HREADY(w_RISC2AHB_alu_HREADY),
    .out_sl_HRESP( w_RISC2AHB_alu_HRESP),
    .out_sl_HRDATA(w_RISC2AHB_alu_HRDATA)
    );
```

```
// Multiplier
riscv_multiplier_if u_riscv_multiplier_if (
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .sl_HREADY(1'b1),
    .sl_HSEL(sl_HSEL_multiplier),
    .sl_HTRANS(w_RISC2AHB_mst_HTRANS),
    .sl_HBURST(w_RISC2AHB_mst_HBURST),
    .sl_HSIZE(w_RISC2AHB_mst_HSIZE),
    .sl_HADDR(w_RISC2AHB_mst_HADDR),
    .sl_HWRITE(w_RISC2AHB_mst_HWRITE),
    .sl_HWDATA(w_RISC2AHB_mst_HWDATA),
    .out_sl_HREADY(w_RISC2AHB_mul_HREADY),
    .out_sl_HRESP( w_RISC2AHB_mul_HRESP),
    .out_sl_HRDATA(w_RISC2AHB_mul_HRDATA)
    );
```

Select signals

- With 2 slaves, it must select the output signals from one of them.




```
always@(*) begin
    // Master accesses ALU
    if(sl_HSEL_alu == 1'b1) begin
        w_RISC2AHB_mst_HRDATA   = w_RISC2AHB_alu_HRDATA     ;
        w_RISC2AHB_mst_HRESP    = w_RISC2AHB_alu_HRESP      ;
        w_RISC2AHB_mst_HREADY   = w_RISC2AHB_alu_HREADY     ;
    end
    // Master accesses Multiplier
    else begin
        w_RISC2AHB_mst_HRDATA   = /* Insert your code */    ;
        w_RISC2AHB_mst_HRESP    = w_RISC2AHB_mul_HRESP      ;
        w_RISC2AHB_mst_HREADY   = w_RISC2AHB_mul_HREADY     ;
    end
end
```

TODO

Map (map.v): Address mapping

- Two steps to define a mapping address
  - Define a base address for Multiplier
  - Define a register address map for Multiplier.

```
//------------------------------------------------------------
// Base Address
//------------------------------------------------------------
`define RISCV_ALU_BASE_ADDR         32'hE0000000
`define RISCV_MULTIPLIER_BASE_ADDR  32'hE0001000

//------------------------------------------------------------
// ALU
//------------------------------------------------------------
`define RISCV_REG_ALU_OP_I   (`RISCV_ALU_BASE_ADDR + 32'h00)
`define RISCV_REG_ALU_A_I    (`RISCV_ALU_BASE_ADDR + 32'h04)
`define RISCV_REG_ALU_B_I    (`RISCV_ALU_BASE_ADDR + 32'h08)
`define RISCV_REG_ALU_P_O    (`RISCV_ALU_BASE_ADDR + 32'h0C)

//------------------------------------------------------------
// Multipler
//------------------------------------------------------------
`define RISCV_REG_MUL_OP_I       (`RISCV_MULTIPLIER_BASE_ADDR + 32'h00)
`define RISCV_REG_MUL_A_I        (`RISCV_MULTIPLIER_BASE_ADDR + 32'h04)
`define RISCV_REG_MUL_B_I        (`RISCV_MULTIPLIER_BASE_ADDR + 32'h08)
`define RISCV_REG_MUL_A_SIGNED   (`RISCV_MULTIPLIER_BASE_ADDR + 32'h0C)
`define RISCV_REG_MUL_B_SIGNED   (`RISCV_MULTIPLIER_BASE_ADDR + 32'h10)
`define RISCV_REG_MUL_P_O_LOW    (`RISCV_MULTIPLIER_BASE_ADDR + 32'h14)
`define RISCV_REG_MUL_P_O_HIGH   (`RISCV_MULTIPLIER_BASE_ADDR + 32'h18)
`define RISCV_REG_MUL_STALL_W    (`RISCV_MULTIPLIER_BASE_ADDR + 32'h1C)
```

hexadecimal

4 address

Test bench

```
`timescale 1ns / 100ps
module top_system_tb;
        reg HCLK, HRESETn;                          //**** module input

        reg sl_HSEL_alu;                            //**** select signals
        reg sl_HSEL_multipliers;

        reg [3:0]       alu_op_i;                   //**** control signals
        reg [31:0]      alu_a_i, alu_b_i;
        reg [31:0]      alu_p_o;
```

```
top_system u_top_system(                            module instance for test bench
        .HRESETn(HRESETn),
        .HCLK(HCLK),
        .sl_HSEL_alu (sl_HSEL_alu),
        .sl_HSEL_multiplier(sl_HSEL_multiplier)
);
```

```
initial begin                                       Initialize Inputs
        HCLK = 0;
        HRESETn = 0;
        #(p/2) HRESETn = 1;

        alu_a_i = 0;
        alu_b_i = 0;
        alu_p_o = 0;
```

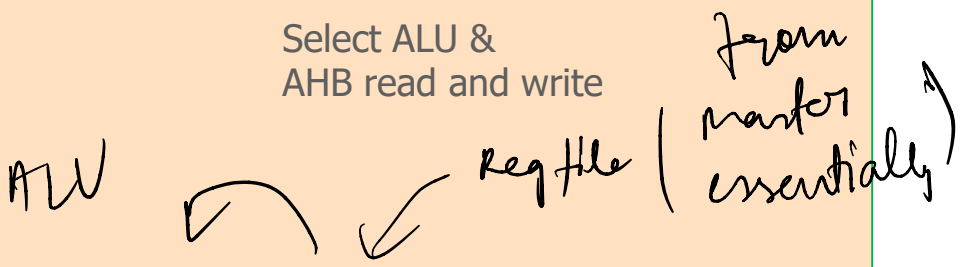
Test cases

- Master accesses ALU
  - Comparison and addition requests.

```
sl_HSEL_alu = 1'b0;                                          // No slave is selected
sl_HSEL_multiplier = 1'b0;
```

```
#(8*p)
        sl_HSEL_alu = 1'b1;
        sl_HSEL_multiplier = 1'b0;
#(8*p)
        alu_a_i = 32'h0;
        alu_b_i = 32'h0;
        alu_op_i = `ALU_SLT;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_A_I, alu_a_i);    // Write the first operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_B_I, alu_b_i);    // Write the second operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_OP_I, alu_op_i);  // Write the operation
#(4*p) u_top_system.u_riscv_dummy.task_AHBread (`RISCV_REG_ALU_P_O, alu_p_o);     // Read the result

#(8*p)
        alu_a_i = 32'h8;
        alu_b_i = 32'h8;
        alu_op_i = `ALU_ADD;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_A_I, alu_a_i);    // Write the first operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_B_I, alu_b_i);    // Write the second operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_OP_I, alu_op_i);  // Write the operation
#(4*p) u_top_system.u_riscv_dummy.task_AHBread( `RISCV_REG_ALU_P_O, alu_p_o);     // Read the result
```

Select ALU &
AHB read and write

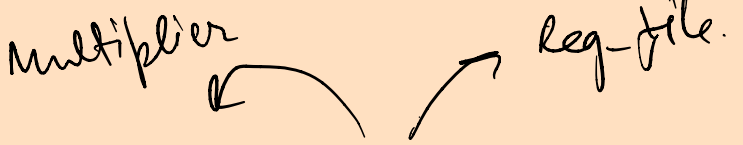


Test cases

- Master accesses Multiplier
  - A multiplication request.

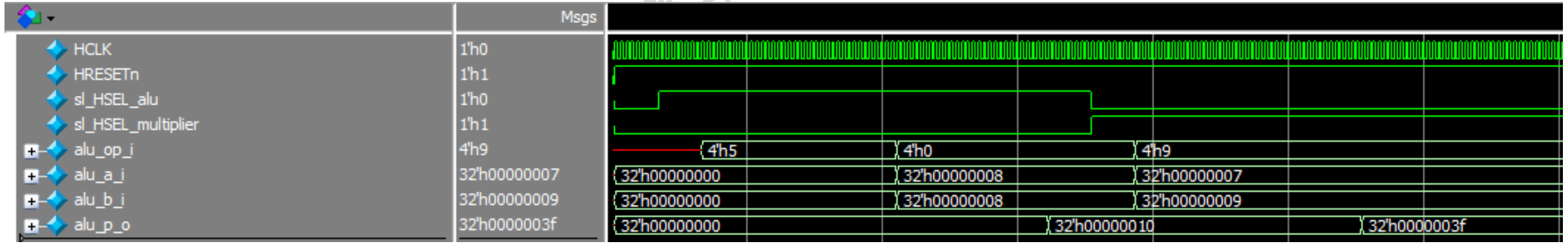
```
#(8*p)
        sl_HSEL_alu = 1'b0;
        sl_HSEL_multiplier = 1'b1;
#(8*p)
        alu_a_i = 32'h7;
        alu_b_i = 32'h9;
        alu_op_i = `ALU_MULL;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_A_I, alu_a_i     ); // Write the first operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_B_I, alu_b_i     ); // Write the second operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_A_SIGNED, alu_a_i); // Write the first operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_B_SIGNED, alu_b_i); // Write the second operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_OP_I, alu_op_i   );    // Write the operation
#(4*p) u_top_system.u_riscv_dummy.task_AHBread( `RISCV_REG_MUL_P_O_LOW, alu_p_o );  // Read the result

end
```

Select multiplier &
AHB read and write



Waveform

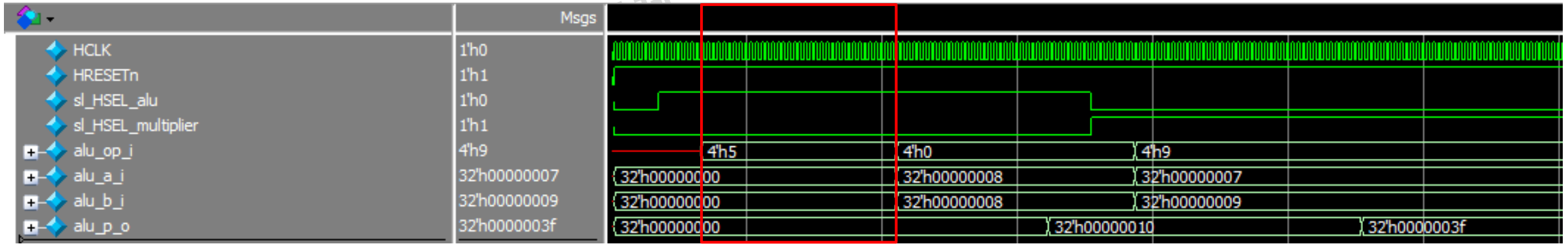
- ALU Slave
  - SLT operation
  - ADD operation
- Multiplier Slave
  - Multiplier operation



Waveform

- ALU Slave
  - SLT operation
  - ADD operation
- Multiplier Slave
  - Multiplier operation

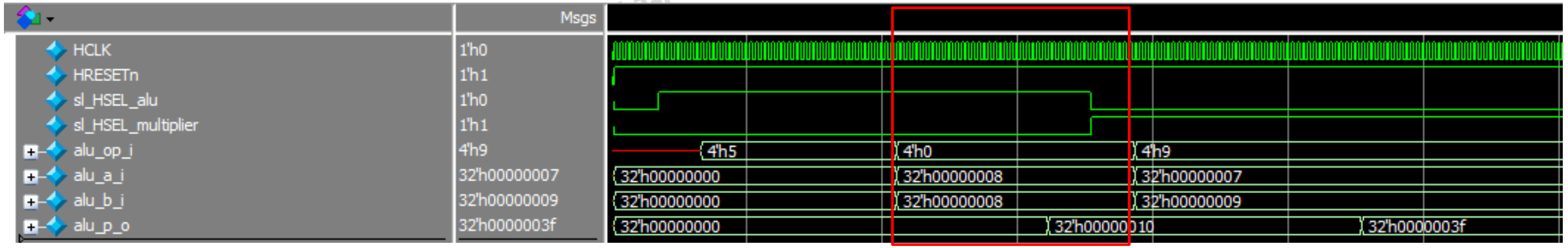
```
#(8*p)
        sl_HSEL_alu = 1'b1;
        sl_HSEL_multiplier = 1'b0;
#(8*p)
        alu_a_i = 32'h0;
        alu_b_i = 32'h0;
        alu_op_i = `ALU_SLT;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_A_I, alu_a_i);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_B_I, alu_b_i);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_OP_I, alu_op_i);
#(4*p) u_top_system.u_riscv_dummy.task_AHBread (`RISCV_REG_ALU_P_O, alu_p_o);
```



Waveform

- ALU Slave
  - SLT operation
  - ADD operation
- Multiplier Slave
  - Multiplier operation

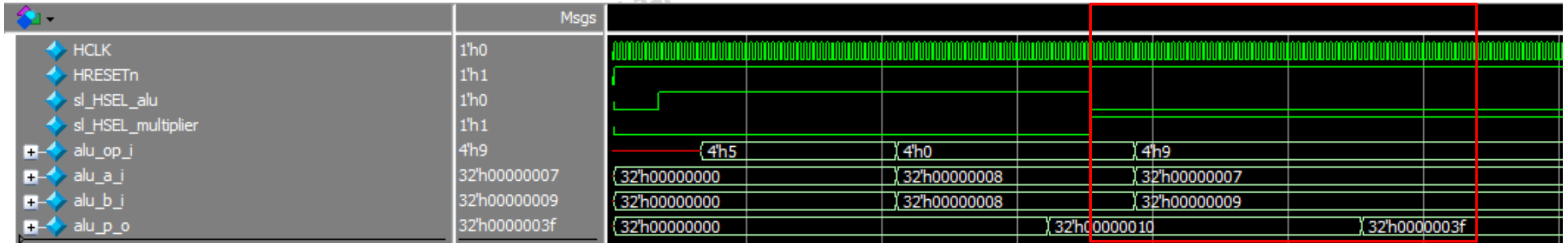
```
#(8*p)
        alu_a_i = 32'h8;
        alu_b_i = 32'h8;
        alu_op_i = `ALU_ADD;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_A_I, alu_a_i);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_B_I, alu_b_i);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_OP_I,
alu_op_i#(4*p) u_top_system.u_riscv_dummy.task_AHBread( `RISCV_REG_ALU_P_O,
alu_p_o);
```



Waveform

- ALU Slave
  - SLT operation
  - ADD operation
- Multiplier Slave
  - Multiplier operation

```
#(8*p)
            sl_HSEL_alu = 1'b0;
            sl_HSEL_multiplier = 1'b1;
    #(8*p)
            alu_a_i = 32'h7;
            alu_b_i = 32'h9;
            alu_op_i = `ALU_MULL;
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_A_I, alu_a_i );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_B_I, alu_b_i  );
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_A_SIGNED, alu_a_i);
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_B_SIGNED, alu_b_i);
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_OP_I, alu_op_i   );
    #(4*p) u_top_system.u_riscv_dummy.task_AHBread( `RISCV_REG_MUL_P_O_LOW, alu_p_o );
```



To do …

- Build an AHB wrapper of a multiplier and add it into a top system.
  - Complete the missing codes
    - **top_system.v, multiplier_if.v, map.v**
  - Do a simulation with time = 3,500ns
  - Show the output waveform

Road map

RISC-V Summary

AMBA AHB Bus
(ALU IP)

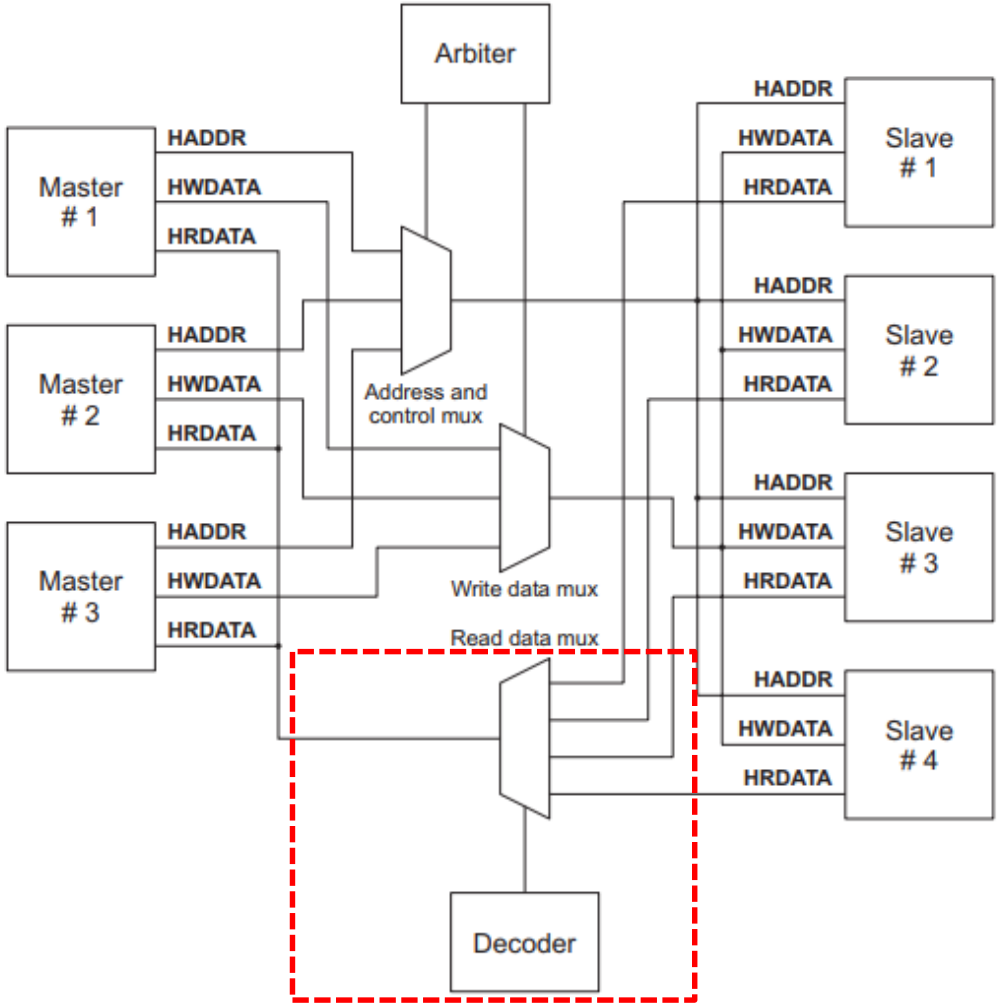
Multiplier IP

AHB Decoder

Bus Interconnect

AHB Decoder

- AHB Decoder: The bus decoder performs the **decoding of the transfer addresses** and **selects slaves appropriately.**
- How?
  - Each slave has an unique base address



Lab 3: AHB Decoder

- Lab 3: Implement decoding function
  - Generate internal select signals in the top module
    - Complete the missing codes
      - **top_system.v**
    - Do simulation with time = 3,500ns
    - Show the output waveform

Map (map.v)

- ALU and Multiplier have different base addresses

```
//------------------------------------------------------------
// Base Address
//------------------------------------------------------------
`define RISCV_ALU_BASE_ADDR          32'hE000_0000
`define RISCV_MULTIPLIER_BASE_ADDR   32'hE000_1000
`define RISCV_BASE_ADDRESS_MASK      32'hFFFF_F000
//------------------------------------------------------------
// ALU
//------------------------------------------------------------
`define RISCV_REG_ALU_OP_I   (`RISCV_ALU_BASE_ADDR + 32'h00)
`define RISCV_REG_ALU_A_I    (`RISCV_ALU_BASE_ADDR + 32'h04)
`define RISCV_REG_ALU_B_I    (`RISCV_ALU_BASE_ADDR + 32'h08)
`define RISCV_REG_ALU_P_O    (`RISCV_ALU_BASE_ADDR + 32'h0C)

//------------------------------------------------------------
// Multipler
//------------------------------------------------------------
`define RISCV_REG_MUL_OP_I        (`RISCV_MULTIPLIER_BASE_ADDR + 32'h00)
`define RISCV_REG_MUL_A_I         (`RISCV_MULTIPLIER_BASE_ADDR + 32'h04)
`define RISCV_REG_MUL_B_I         (`RISCV_MULTIPLIER_BASE_ADDR + 32'h08)
`define RISCV_REG_MUL_A_SIGNED    (`RISCV_MULTIPLIER_BASE_ADDR + 32'h0C)
`define RISCV_REG_MUL_B_SIGNED    (`RISCV_MULTIPLIER_BASE_ADDR + 32'h10)
`define RISCV_REG_MUL_P_O_LOW     (`RISCV_MULTIPLIER_BASE_ADDR + 32'h14)
`define RISCV_REG_MUL_P_O_HIGH    (`RISCV_MULTIPLIER_BASE_ADDR + 32'h18)
`define RISCV_REG_MUL_STALL_W     (`RISCV_MULTIPLIER_BASE_ADDR + 32'h1C)
```

Motivation

- Observation
  - Read/Write commands from a master are valid

  ⇒ Select signals are not necessary

```
`define RISCV_ALU_BASE_ADDR              32'hE0000000
`define RISCV_MULTIPLIER_BASE_ADDR       32'hE0001000
```

```
#(8*p)
    //sl_HSEL_alu = 1'b1;
    //sl_HSEL_multiplier = 1'b0;
#(8*p)
        alu_a_i = 32'h0;
        alu_b_i = 32'h0;
        alu_op_i = `ALU_SLT;
#(4*p) u_top_system.RISC_model_u0.task_AHBwrite(`RISCV_REG_ALU_A_I, alu_a_i);
#(4*p) u_top_system.RISC_model_u0.task_AHBwrite(`RISCV_REG_ALU_B_I, alu_b_i);
#(4*p) u_top_system.RISC_model_u0.task_AHBwrite(`RISCV_REG_ALU_OP_I, alu_op_i);
#(4*p) u_top_system.RISC_model_u0.task_AHBread (`RISCV_REG_ALU_P_O, alu_p_o);
```

Top module (top_system.v)

- Modified top module (top_system.v)
  - External select signals are eliminated
  - Two corresponding internal signals are added.

```
module top_system(
// Inputs
input HCLK,
input HRESETn
//// Select signals
//input sl_HSEL_alu,
//input sl_HSEL_multiplier
);

reg sl_HSEL_alu;
reg sl_HSEL_multiplier;
```

To do …

- Generate select signals from based addresses

```
always@(*) begin
    sl_HSEL_alu = 1'b0;
    sl_HSEL_multiplier =1'b0;
    //Insert your code
    //{{{

    //}}}
end
always@(*) begin
    // Master accesses ALU
    if(sl_HSEL_alu == 1'b1) begin
        w_RISC2AHB_mst_HRDATA   = w_RISC2AHB_alu_HRDATA      ;
        w_RISC2AHB_mst_HRESP    = w_RISC2AHB_alu_HRESP       ;
        w_RISC2AHB_mst_HREADY   = w_RISC2AHB_alu_HREADY      ;
    end
    // Master accesses Multiplier
    else begin
        //w_RISC2AHB_mst_HRDATA   = /*Insert your code*/      ;
        w_RISC2AHB_mst_HRESP    = w_RISC2AHB_mul_HRESP       ;
        w_RISC2AHB_mst_HREADY   = w_RISC2AHB_mul_HREADY      ;
    end
end
```

Test bench

```
`timescale 1ns / 100ps
module top_system_tb;
        reg HCLK, HRESETn;                          //**** module input

        //reg sl_HSEL_alu;                          //**** select signals
        //reg sl_HSEL_multipliers;

        reg [3:0]       alu_op_i;                   //**** control signals
        reg [31:0]      alu_a_i, alu_b_i;
        reg [31:0]      alu_p_o;
```

```
top_system u_top_system(                            module instance for test bench
        .HRESETn(HRESETn)
        ,.HCLK(HCLK)
        //,.sl_HSEL_alu         (sl_HSEL_alu)
        //,.sl_HSEL_multiplier(sl_HSEL_multiplier)
);
```

```
initial begin                                       Initialize Inputs
        HCLK = 0;
        HRESETn = 0;
        #(p/2) HRESETn = 1;

        alu_a_i = 0;
        alu_b_i = 0;
        alu_p_o = 0;
```

Test cases

- Master accesses ALU
  - Comparison and addition requests.

```
//sl_HSEL_alu = 1'b0;                                          // No slave is selected
//sl_HSEL_multiplier = 1'b0;
```

```
#(8*p)
        //sl_HSEL_alu = 1'b1;                                  Select ALU &
        //sl_HSEL_multiplier = 1'b0;                           AHB read and write
#(8*p)
        alu_a_i = 32'h0;
        alu_b_i = 32'h0;
        alu_op_i = `ALU_SLT;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_A_I, alu_a_i);    // Write the first operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_B_I, alu_b_i);    // Write the second operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_OP_I, alu_op_i);  // Write the operation
#(4*p) u_top_system.u_riscv_dummy.task_AHBread (`RISCV_REG_ALU_P_O, alu_p_o);     // Read the result
#(8*p)
        alu_a_i = 32'h8;
        alu_b_i = 32'h8;
        alu_op_i = `ALU_ADD;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_A_I, alu_a_i);    // Write the first operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_B_I, alu_b_i);    // Write the second operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_OP_I, alu_op_i);  // Write the operation
#(4*p) u_top_system.u_riscv_dummy.task_AHBread( `RISCV_REG_ALU_P_O, alu_p_o);     // Read the result
```

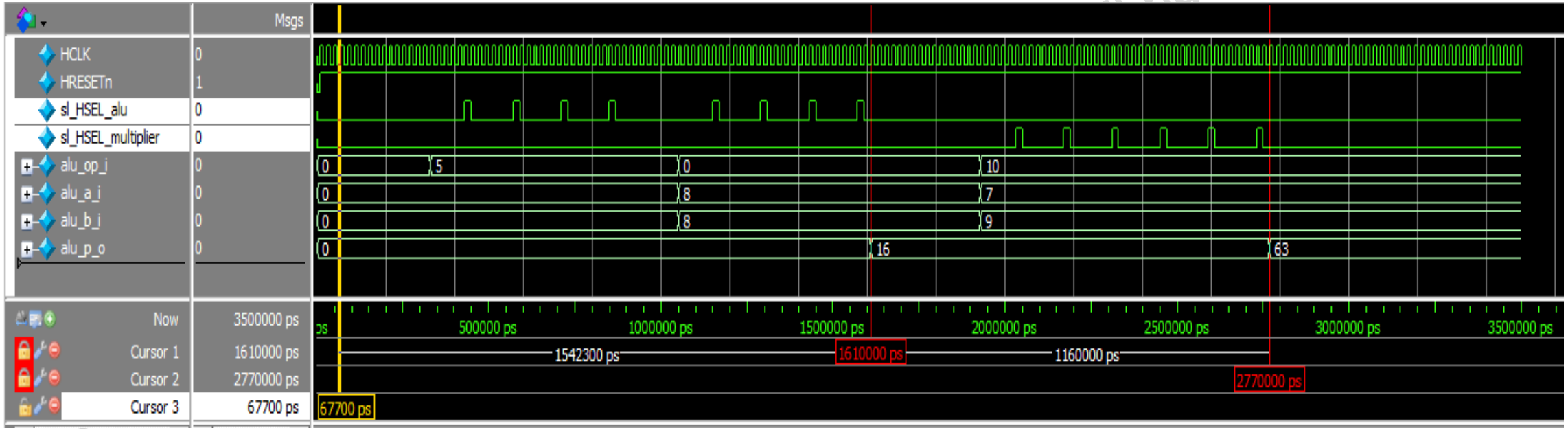
Test cases

- Master accesses Multiplier
  - A multiplication request.

```
    #(8*p)
            //sl_HSEL_alu = 1'b0;                                    Select multiplier &
            //sl_HSEL_multiplier = 1'b1;                             AHB read and write
    #(8*p)
            alu_a_i = 32'h7;
            alu_b_i = 32'h9;
            alu_op_i = `ALU_MULL;
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_A_I, alu_a_i     ); // Write the first operand
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_B_I, alu_b_i     ); // Write the second operand
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_A_SIGNED, alu_a_i); // Write the first operand
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_B_SIGNED, alu_b_i); // Write the second operand
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_OP_I, alu_op_i   );    // Write the operation
    #(4*p) u_top_system.u_riscv_dummy.task_AHBread( `RISCV_REG_MUL_P_O_LOW, alu_p_o ); // Read the result

  end
```

Waveform

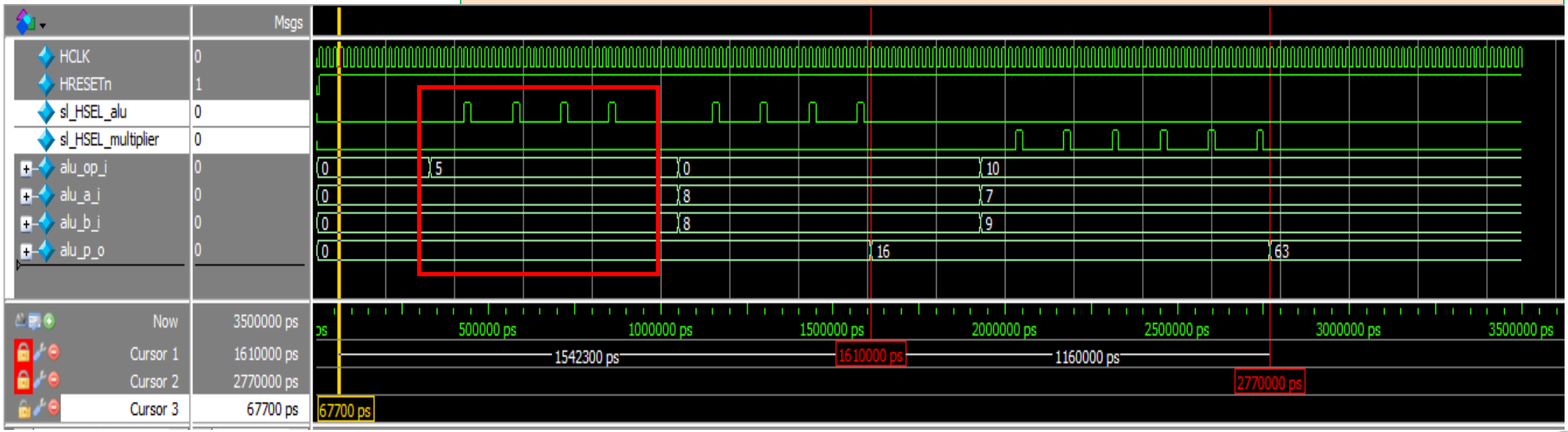
- ALU Slave: Comparison, addition operations
- Multiplier Slave: multiplication



Waveform

- ALU Slave
  - SLT operation
  - ADD operation
- Multiplier Slave
  - Multiplier operation

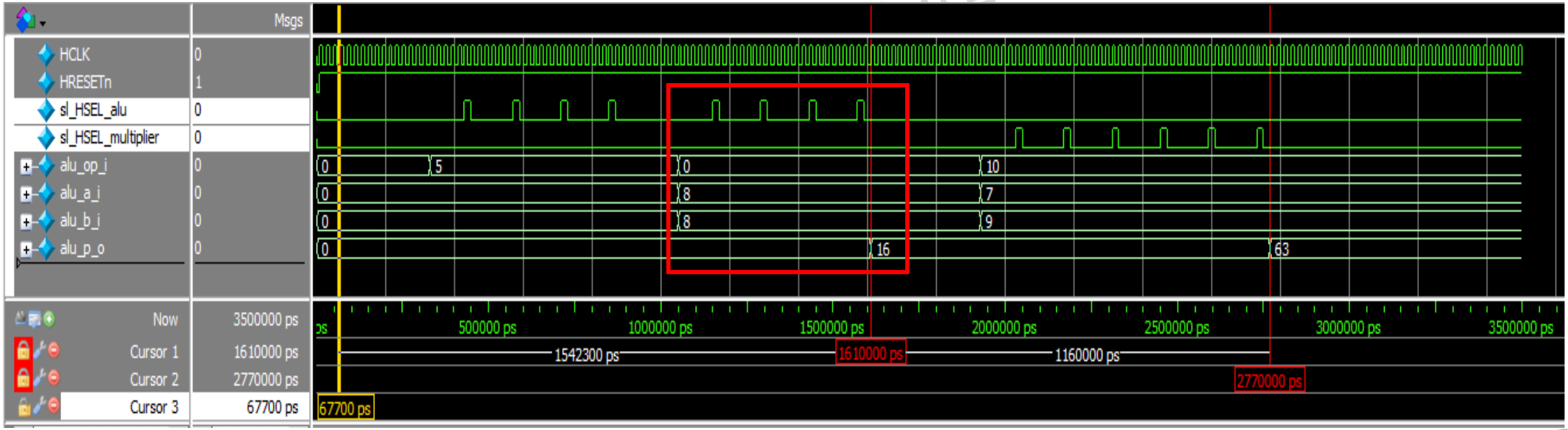
```
#(8*p)
        //sl_HSEL_alu = 1'b1;
        //sl_HSEL_multiplier = 1'b0;
#(8*p)
        alu_a_i = 32'h0;
        alu_b_i = 32'h0;
        alu_op_i = `ALU_SLT;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_A_I, alu_a_i);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_B_I, alu_b_i);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_OP_I, alu_op_i);
#(4*p) u_top_system.u_riscv_dummy.task_AHBread (`RISCV_REG_ALU_P_O, alu_p_o);
```



Waveform

- ALU Slave
  - SLT operation
  - ADD operation
- Multiplier Slave
  - Multiplier operation

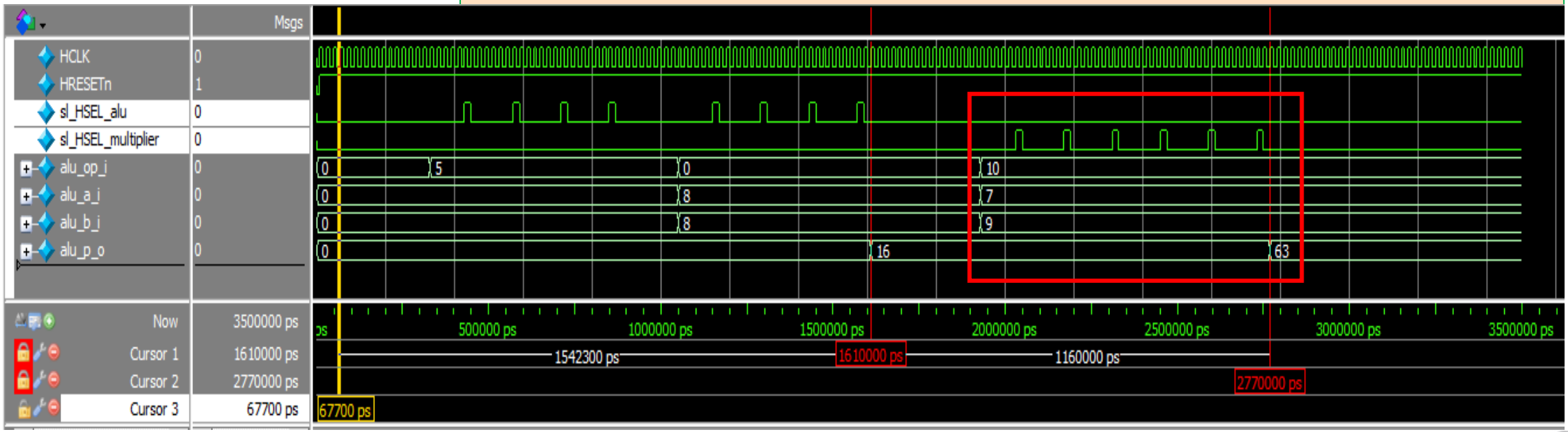
```
#(8*p)
        alu_a_i = 32'h8;
        alu_b_i = 32'h8;
        alu_op_i = `ALU_ADD;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_A_I, alu_a_i);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_B_I, alu_b_i);
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_OP_I,
alu_op_i#(4*p) u_top_system.u_riscv_dummy.task_AHBread( `RISCV_REG_ALU_P_O,
alu_p_o);
```



Waveform

- ALU Slave
  - SLT operation
  - ADD operation
- Multiplier Slave
  - Multiplier operation

```
#(8*p)      //sl_HSEL_alu = 1'b0;
             //sl_HSEL_multiplier = 1'b1;
#(8*p)      alu_a_i = 32'h7;
            alu_b_i = 32'h9;
            alu_op_i = `ALU_MULL;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_A_I, alu_a_i );
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_B_I, alu_b_i  );
      #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_A_SIGNED, alu_a_i);
      #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_B_SIGNED, alu_b_i);
      #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_OP_I, alu_op_i   );
      #(4*p) u_top_system.u_riscv_dummy.task_AHBread( `RISCV_REG_MUL_P_O_LOW, alu_p_o );
```



To do …

- Generate internal select signals in the top module
  - Complete the missing codes
    - **top_system.v**
  - Do a simulation with time = 3,500ns
  - Show the output waveform

Road map

RISC-V Summary

AMBA AHB Bus
(ALU IP)

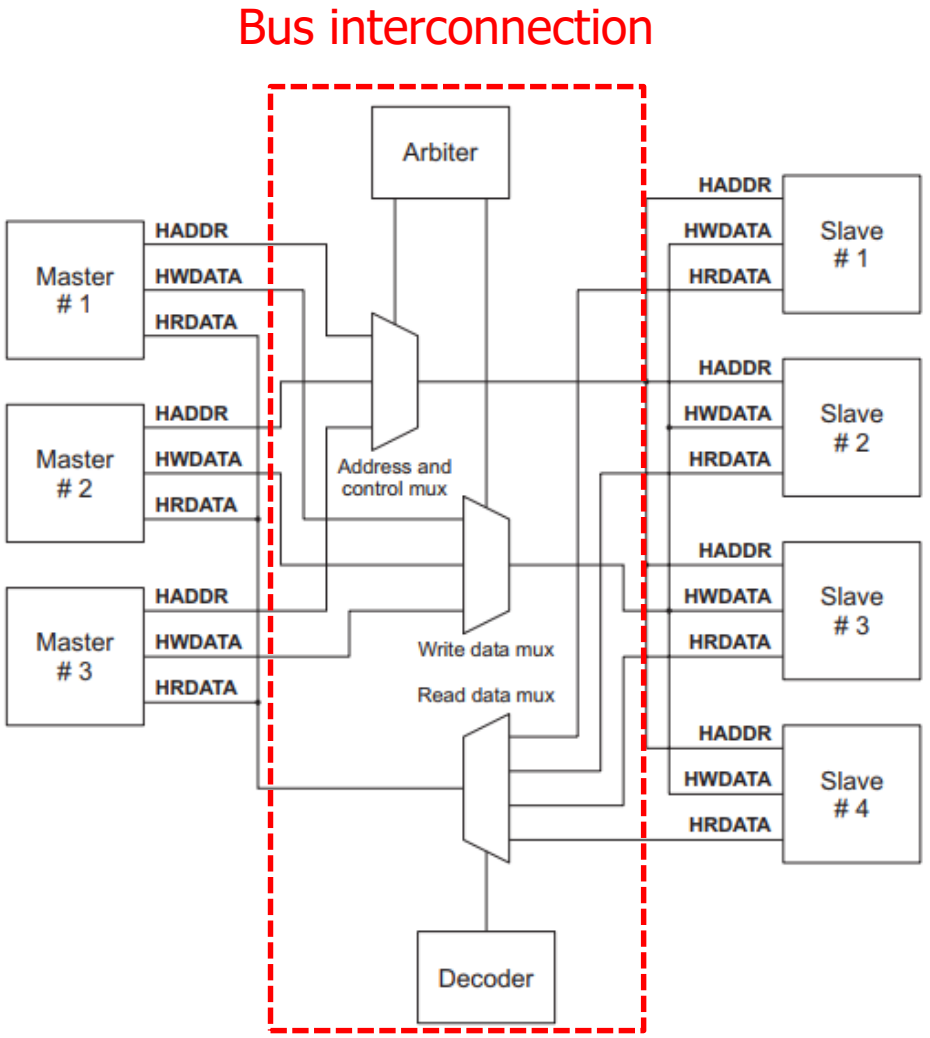
Multiplier IP

AHB Decoder

Bus Interconnect

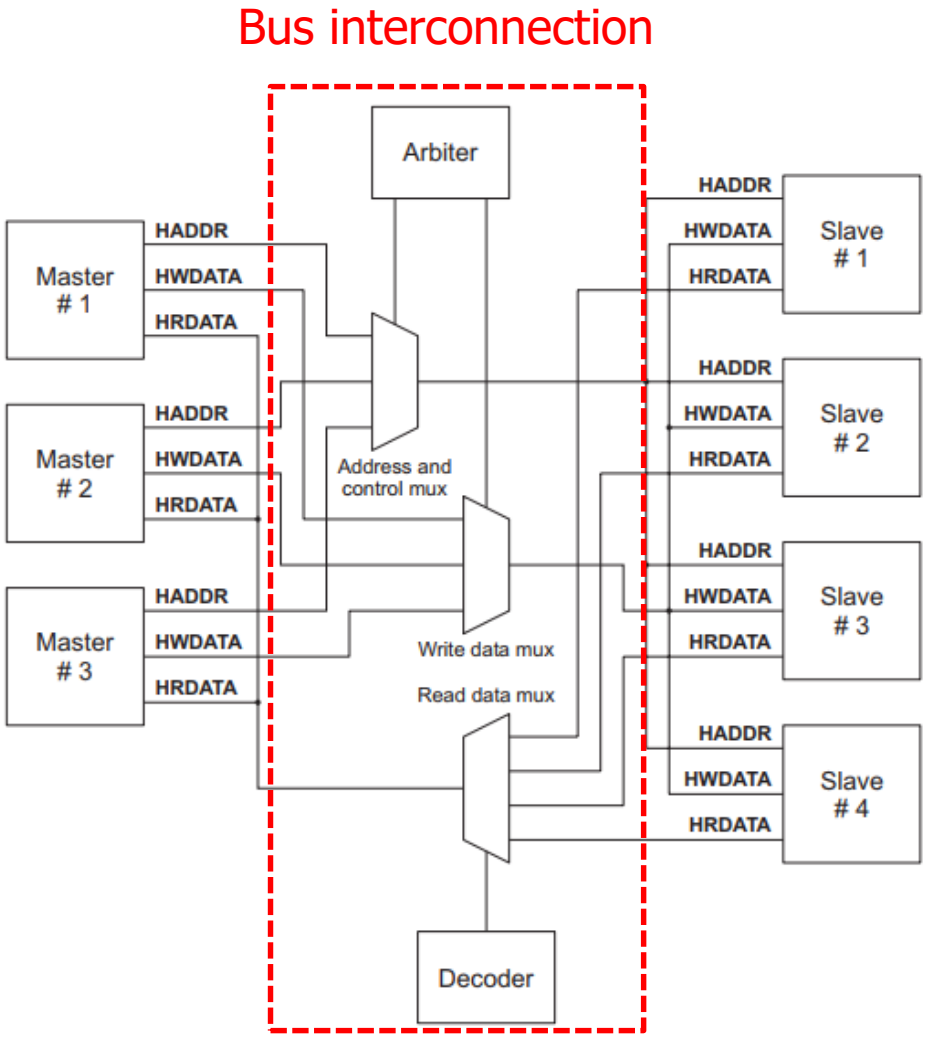
AHB Interconnection

- Bus interconnection: connects multiple masters and multiple slaves.
- Bus interconnection
  - Address and control multiplexer (mux)
  - Write data mux
  - Read data mux
- What signals are required for each IP?



AHB Interconnection

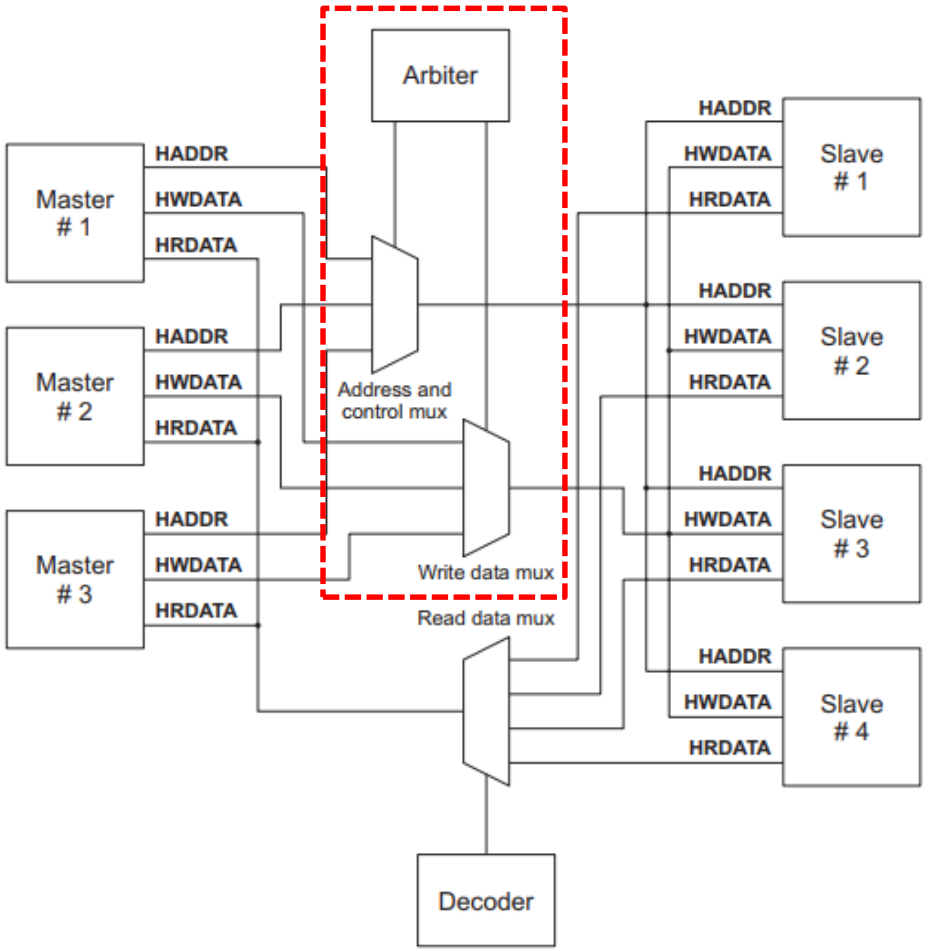
- Bus interconnection: connects multiple masters and multiple slaves.
- Bus interconnection
  - Address and control multiplexer (mux)
  - Write data mux
  - Read data mux
- What signals are required for each IP?
  - HADDR: Address (base + offset)
  - HWDATA: Write data
  - HRDATA: Read data
  - Other control signals



AHB Arbiter

- AHB Arbiter: Control which master has accessed to Bus.
- Every bus master has a REQUEST/GRANT interface to the arbiter
- The arbiter uses a **prioritization scheme** to decide which bus master is currently **the highest priority master** requesting the bus.
- The prioritization scheme is not fixed, i.e. Round-Robin scheme.

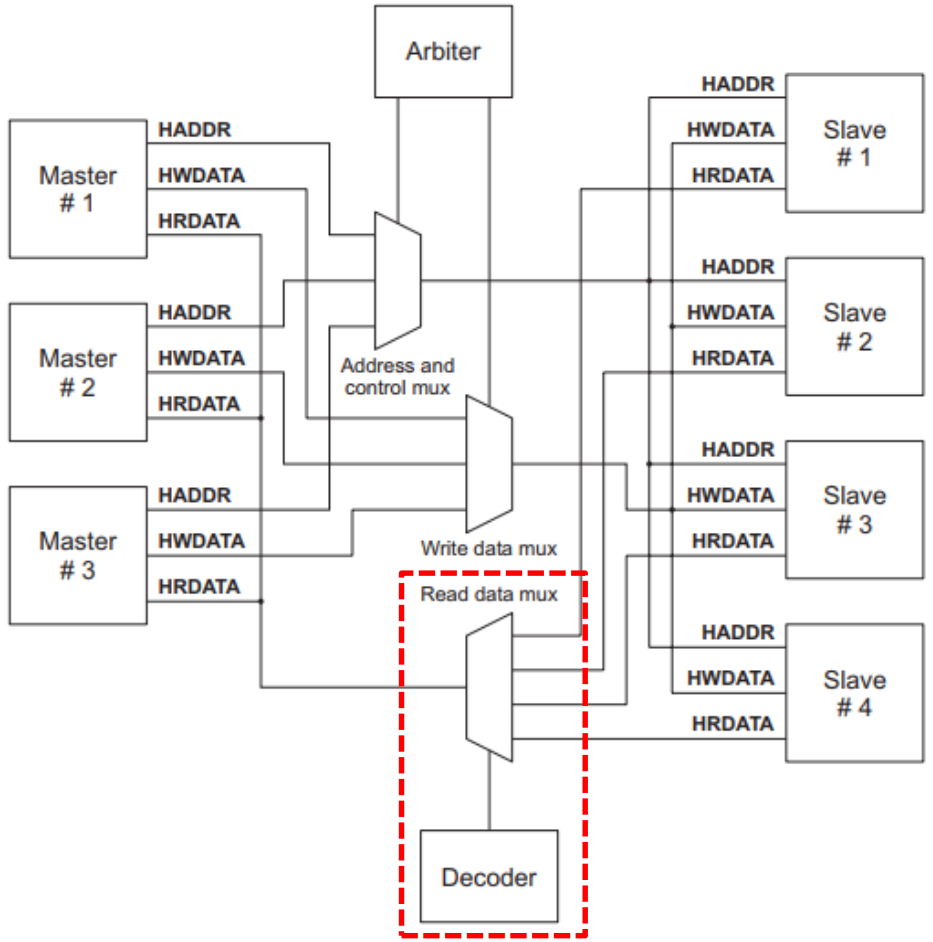
Bus interconnection



AHB Decoder

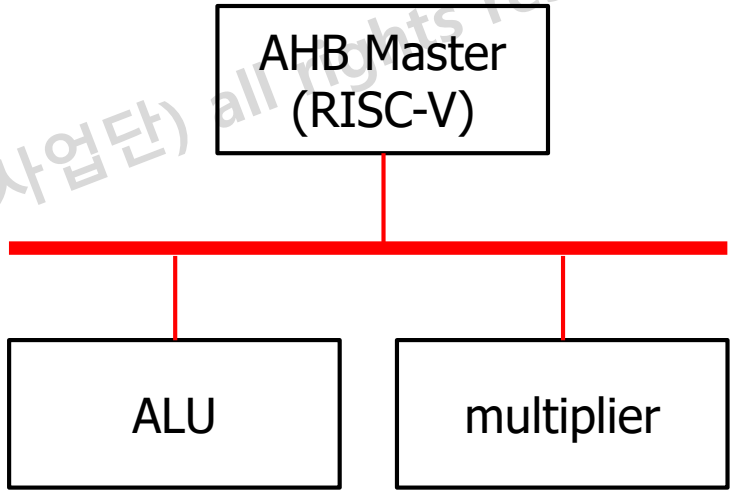
- AHB Decoder: Performs the decoding of the transfer addresses and selects slaves appropriately.

Bus interconnection



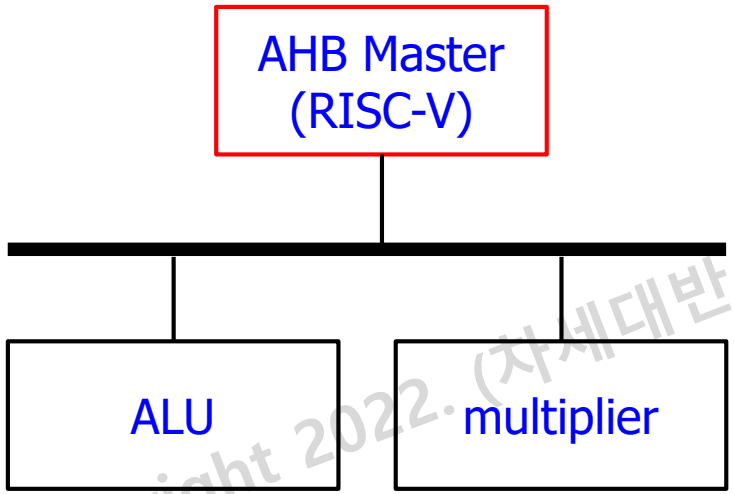
Lab 4: Bus Interconnection

- Lab 1: Build a top module (top_system.v)
- How to add your own IP into the system?
- We make a top module
  - 1 master: RISC-V Dummy
  - 2 slaves: ALU, multiplier
- This example focuses on **Bus Interconnection**.
- Do simulation and show the waveform



Master

- We reuse RISC-V Dummy as a master
  - Issues READ/WRITE commands to slaves




```
//------------------------------------------------------
// Master
//------------------------------------------------------
ahb_master u_riscv_dummy(
      .HRESETn          (HRESETn                )
     ,.HCLK             (HCLK                   )
     ,.i_HRDATA         (w_RISC2AHB_mst_HRDATA      )
     ,.i_HRESP          (w_RISC2AHB_mst_HRESP       )
     ,.i_HREADY         (w_RISC2AHB_mst_HREADY      )
     ,.o_HADDR          (w_RISC2AHB_mst_HADDR       )
     ,.o_HWDATA         (w_RISC2AHB_mst_HWDATA      )
     ,.o_HWRITE         (w_RISC2AHB_mst_HWRITE      )
     ,.o_HSIZE          (w_RISC2AHB_mst_HSIZE       )
     ,.o_HBURST         (w_RISC2AHB_mst_HBURST      )
     ,.o_HTRANS         (w_RISC2AHB_mst_HTRANS      )
     );
//
```

AHB Slaves

- Two AHB slaves are ALU and Multiplier
- What are changed in the AHB slaves' ports?

Your code

```
// ALU
riscv_alu_if u_riscv_alu_if (
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .sl_HREADY(alu_sl_HREADY),
    .sl_HSEL(alu_sl_HSEL),
    .sl_HTRANS(alu_sl_HTRANS),
    .sl_HBURST(alu_sl_HBURST),
    .sl_HSIZE(alu_sl_HSIZE),
    .sl_HADDR(alu_sl_HADDR),
    .sl_HWRITE(alu_sl_HWRITE),
    .sl_HWDATA(alu_sl_HWDATA),
    .out_sl_HREADY(/* Insert your code*/),
    .out_sl_HRESP (/* Insert your code*/),
    .out_sl_HRDATA(/* Insert your code*/)
    );
```

```
Multiplier
scv_multiplier_if u_riscv_multiplier_if (
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .sl_HREADY(mul_sl_HREADY),
    .sl_HSEL(mul_sl_HSEL),
    .sl_HTRANS(mul_sl_HTRANS),
    .sl_HBURST(mul_sl_HBURST),
    .sl_HSIZE(mul_sl_HSIZE),
    .sl_HADDR(mul_sl_HADDR),
    .sl_HWRITE(mul_sl_HWRITE),
    .sl_HWDATA(mul_sl_HWDATA),
    .out_sl_HREADY(/* Insert your code*/),
    .out_sl_HRESP (/* Insert your code*/),
    .out_sl_HRDATA(/* Insert your code*/)
    );
```

AHB Slaves

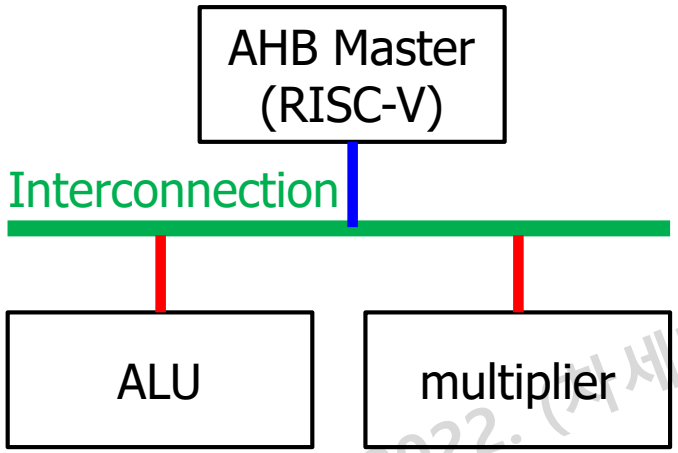
- What are changed in the AHB slaves' ports?
  - A Master's ports are NOT connected directly Slave's ports

```
// ALU
riscv_alu_if u_riscv_alu_if (
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .sl_HREADY(1'b1),
    .sl_HSEL(sl_HSEL_alu),
    .sl_HTRANS(w_RISC2AHB_mst_HTRANS),
    .sl_HBURST(w_RISC2AHB_mst_HBURST),
    .sl_HSIZE(w_RISC2AHB_mst_HSIZE),
    .sl_HADDR(w_RISC2AHB_mst_HADDR),
    .sl_HWRITE(w_RISC2AHB_mst_HWRITE),
    .sl_HWDATA(w_RISC2AHB_mst_HWDATA),
    .out_sl_HREADY(w_RISC2AHB_alu_HREADY),
    .out_sl_HRESP( w_RISC2AHB_alu_HRESP),
    .out_sl_HRDATA(w_RISC2AHB_alu_HRDATA)
);
```

```
// ALU
riscv_alu_if u_riscv_alu_if (
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .sl_HREADY(alu_sl_HREADY),
    .sl_HSEL(alu_sl_HSEL),
    .sl_HTRANS(alu_sl_HTRANS),
    .sl_HBURST(alu_sl_HBURST),
    .sl_HSIZE(alu_sl_HSIZE),
    .sl_HADDR(alu_sl_HADDR),
    .sl_HWRITE(alu_sl_HWRITE),
    .sl_HWDATA(alu_sl_HWDATA),
    .out_sl_HREADY(/* Insert your code*/),
    .out_sl_HRESP (/* Insert your code*/),
    .out_sl_HRDATA(/* Insert your code*/)
);
```

AHB interconnection

- AHB interconnection gathers all connections between masters and slaves
  - Master ports and Slave ports



Master

Slave

```
u_ahb_lite_interconnect(
    ./*input                                */HCLK                  (HCLK                  )
    ./*input                                */HRESETn               (HRESETn               )
    ,./*input   [N_MASTER-1:0]              */ma_HREADY             (w_AHB_IC_ma_HREADY    )
    ,./*input   [N_MASTER-1:0]              */ma_HSEL               (w_AHB_IC_ma_HSEL      )
    ,./*input   [N_MASTER*2-1:0]            */ma_HTRANS             (w_AHB_IC_ma_HTRANS    )
    ,./*input   [N_MASTER*3-1:0]            */ma_HBURST             (w_AHB_IC_ma_HBURST    )
    ,./*input   [N_MASTER*3-1:0]            */ma_HSIZE              (w_AHB_IC_ma_HSIZE     )
    ,./*input   [N_MASTER*4-1:0]            */ma_HPROT              (w_AHB_IC_ma_HPROT     )
    ,./*input   [N_MASTER-1:0]              */ma_HMASTLOCK          (w_AHB_IC_ma_HMASTLOCK )
    ,./*input   [N_MASTER*W_ADDR-1:0]       */ma_HADDR              (w_AHB_IC_ma_HADDR     )
    ,./*input   [N_MASTER-1:0]              */ma_HWRITE             (w_AHB_IC_ma_HWRITE    )
    ,./*input   [N_MASTER*W_DATA-1:0]       */ma_HWDATA             (w_AHB_IC_ma_HWDATA    )
    ,./*output  [N_MASTER-1:0]              */out_ma_HREADY         (w_AHB_IC_out_ma_HREADY)
    ,./*output  [N_MASTER*2-1:0]            */out_ma_HRESP          (w_AHB_IC_out_ma_HRESP )
    ,./*output  [N_MASTER*W_DATA-1:0]       */out_ma_HRDATA         (w_AHB_IC_out_ma_HRDATA)

    ,./*output  [N_SLAVE-1:0]               */out_sl_HREADY         (w_AHB_IC_out_sl_HREADY    )
    ,./*output  [N_SLAVE-1:0]               */out_sl_HSEL           (w_AHB_IC_out_sl_HSEL      )
    ,./*output  [N_SLAVE*2-1:0]             */out_sl_HTRANS         (w_AHB_IC_out_sl_HTRANS    )
    ,./*output  [N_SLAVE*3-1:0]             */out_sl_HBURST         (w_AHB_IC_out_sl_HBURST    )
    ,./*output  [N_SLAVE*3-1:0]             */out_sl_HSIZE          (w_AHB_IC_out_sl_HSIZE     )
    ,./*output  [N_SLAVE*4-1:0]             */out_sl_HPROT          (w_AHB_IC_out_sl_HPROT     )
    ,./*output  [N_SLAVE-1:0]               */out_sl_HMASTLOCK      (w_AHB_IC_out_sl_HMASTLOCK )
    ,./*output  [N_SLAVE*W_ADDR-1:0]        */out_sl_HADDR          (w_AHB_IC_out_sl_HADDR     )
    ,./*output  [N_SLAVE-1:0]               */out_sl_HWRITE         (w_AHB_IC_out_sl_HWRITE    )
    ,./*output  [N_SLAVE*W_DATA-1:0]        */out_sl_HWDATA         (w_AHB_IC_out_sl_HWDATA    )
    ,./*input   [N_SLAVE-1:0]               */sl_HREADY             (w_AHB_IC_sl_HREADY        )
    ,./*input   [N_SLAVE*2-1:0]             */sl_HRESP              (w_AHB_IC_sl_HRESP         )
    ,./*input   [N_SLAVE*W_DATA-1:0]        */sl_HRDATA             (w_AHB_IC_sl_HRDATA        )
```

Addressing

- A system includes a master and two slaves
  - N_MASTER=1
  - N_SLAVE=2

⇒ **If we have to add one IP, what do we have to change?**

```
/*
    RISC model   : master 1
    ALU          : slave 1
    MULT         : slave 2
*/
parameter N_MASTER  =   1;
parameter W_MASTER  =   $clog2(N_MASTER); //GetBitWidth(N_MASTER);
parameter N_SLAVE   =   2;
parameter W_SLAVE   =   $clog2(N_SLAVE); //GetBitWidth(N_SLAVE);

parameter ADDR_START_MAP = {
        `RISCV_ALU_BASE_ADDR,
        `RISCV_MULTIPLIER_BASE_ADDR
    };

parameter ADDR_END_MAP   = {
        `RISCV_ALU_BASE_ADDR,
        `RISCV_MULTIPLIER_BASE_ADDR
    };


parameter ADDR_MASK = {
        `RISCV_BASE_ADDRESS_MASK,
        `RISCV_BASE_ADDRESS_MASK
    };
```

Addressing

- A system includes a master and two slaves
  - N_MASTER=1
  - N_SLAVE=2

⇒ **If we have to add one IP, what do we have to change?**

⇒ Change the number of masters and slaves.

```
/*
    RISC model   : master 1
    ALU          : slave 1
    MULT         : slave 2
*/
parameter N_MASTER  =   1;
parameter W_MASTER  =   $clog2(N_MASTER); //GetBitWidth(N_MASTER
parameter N_SLAVE   =   2;
parameter W_SLAVE   =   $clog2(N_SLAVE); //GetBitWidth(N_SLAVE);

parameter ADDR_START_MAP = {
        /* Insert your code*/
    };

parameter ADDR_END_MAP   = {
        `RISCV_MULTIPLIER_BASE_ADDR,
        `RISCV_ALU_BASE_ADDR
    };


parameter ADDR_MASK = {
        /* Insert your code*/
    };
//}}}
```

64'h000...
64bit
from
32bit
+
32bit

Addressing

- A system includes a master and two slaves
  - N_MASTER=1
  - N_SLAVE=2
- We need to define an address map for all slaves
  - Include BASE addresses and their MASK addresses.

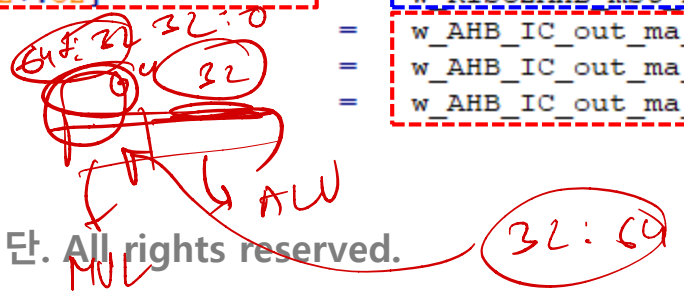
```
/*
    RISC model   : master 1
    ALU          : slave 1
    MULT         : slave 2
*/
parameter N_MASTER  =   1;
parameter W_MASTER  =   $clog2(N_MASTER); //GetBitWidth(N_MASTER
parameter N_SLAVE   =   2;
parameter W_SLAVE   =   $clog2(N_SLAVE); //GetBitWidth(N_SLAVE);

parameter ADDR_START_MAP = {
        /* Insert your code*/
    };

parameter ADDR_END_MAP   = {
        `RISCV_MULTIPLIER_BASE_ADDR,
        `RISCV_ALU_BASE_ADDR
    };

parameter ADDR_MASK = {
        /* Insert your code*/
    };
//}}}
```

Your code

Master

- Connect a master's ports to AHB Interconnect
  - Ports from a master are connected to specified ports of Bus.
  - NOTE: Index 0 indicates the first master

```
// 0. RISCmodel
assign w_AHB_IC_ma_HREADY    [0]                  =  1'b1                      ;
assign w_AHB_IC_ma_HSEL      [0]                  =  |w_RISC2AHB_mst_HTRANS    ;
assign w_AHB_IC_ma_HTRANS    [0*2+:2]             =  w_RISC2AHB_mst_HTRANS     ;
assign w_AHB_IC_ma_HBURST    [0*`W_BURST+:`W_BURST] =  w_RISC2AHB_mst_HBURST   ;
assign w_AHB_IC_ma_HSIZE     [0*3+:3]             =  w_RISC2AHB_mst_HSIZE      ;
assign w_AHB_IC_ma_HPROT     [0*4+:4]             =  4'h0                      ;
assign w_AHB_IC_ma_HMASTLOCK [0]                  =  |w_RISC2AHB_mst_HTRANS    ;
assign w_AHB_IC_ma_HADDR     [0*32+:32]           =  w_RISC2AHB_mst_HADDR      ;
assign w_AHB_IC_ma_HWRITE    [0]                  =  w_RISC2AHB_mst_HWRITE     ;
assign w_AHB_IC_ma_HWDATA    [0*32+:32]           =  w_RISC2AHB_mst_HWDATA     ;
assign w_RISC2AHB_mst_HREADY                      =  w_AHB_IC_out_ma_HREADY    [0];
assign w_RISC2AHB_mst_HRESP                       =  w_AHB_IC_out_ma_HRESP     [0];
assign w_RISC2AHB_mst_HRDATA                      =  w_AHB_IC_out_ma_HRDATA    [0*32+:32]  ;
```



Slave: ALU

- Connect a master's ports to AHB Interconnect
  - Ports from a master are connected to specified ports of Bus.
  - NOTE: Index 0 indicates the first master

```
//  1. AHB2ALU
assign  alu_sl_HSEL      =  w_AHB_IC_out_sl_HSEL     [0]          ;
assign  alu_sl_HADDR     =  w_AHB_IC_out_sl_HADDR    [0*32+:32]   ;
assign  alu_sl_HTRANS    =  w_AHB_IC_out_sl_HTRANS   [0*2+:2]     ;
assign  alu_sl_HBURST    =  w_AHB_IC_out_sl_HBURST   [0*`W_BURST+:`W_BURST] ;
assign  alu_sl_HSIZE     =  w_AHB_IC_out_sl_HSIZE    [0*3+:3]     ;
assign  alu_sl_HPROT     =  w_AHB_IC_out_sl_HPROT    [0*4+:4]     ;
assign  alu_sl_HWRITE    =  w_AHB_IC_out_sl_HWRITE   [0]          ;
assign  alu_sl_HWDATA    =  w_AHB_IC_out_sl_HWDATA   [31:0]       ;
assign  alu_sl_HREADY    =  w_AHB_IC_out_sl_HREADY   [0]          ;
assign  w_AHB_IC_sl_HREADY  [0]          =  out_alu_sl_HREADY;
assign  w_AHB_IC_sl_HRESP   [0*2+:2]     =  out_alu_sl_HRESP;
assign  w_AHB_IC_sl_HRDATA  [0*32+:32]   =  out_alu_sl_HRDATA;
```

To do …

- Connect a slave' ports to AHB Interconnect
  - Ports from a slave are connected to specified ports of Bus.
- Completing the connections between Multiplier and Bus

```
//  2. AHB2MULTIPLIER
//assign    mul_sl_HSEL                =   /* Insert your code */;
//assign    mul_sl_HADDR               =   /* Insert your code */;
//assign    mul_sl_HTRANS              =   /* Insert your code */;
//assign    mul_sl_HBURST              =   /* Insert your code */;
//assign    mul_sl_HSIZE               =   /* Insert your code */;
//assign    mul_sl_HPROT               =   /* Insert your code */;
//assign    mul_sl_HWRITE              =   /* Insert your code */;
//assign    mul_sl_HWDATA              =   /* Insert your code */;
//assign    mul_sl_HREADY              =   /* Insert your code */;
//assign    w_AHB_IC_sl_HREADY  [1]    =   /* Insert your code */;
//assign    w_AHB_IC_sl_HRESP   [1*2+:2]   =   /* Insert your code */;
//assign    w_AHB_IC_sl_HRDATA  [1*32+:32] =   /* Insert your code */;
```

AHB interconnect

- AHB interconnection module
  - Numbers of Masters and Slaves
  - Address maps and Masks.
- Example
  - One master, two slaves

```
//-----------------------------------------
// AHB Interconnect
//-----------------------------------------
ahb_lite_interconnect
#( //amba_ahb_parameter //{{{
.N_MASTER                   (N_MASTER),
.W_MASTER                   (W_MASTER),
.N_SLAVE                    (N_SLAVE),
.W_SLAVE                    (W_SLAVE),
.W_ADDR                     (32),
.W_DATA                     (32),
.WB_DATA                    (4),
.W_WB_DATA                  (2),
.NUM_DEF_MASTER             (0),//the number of
.NUM_DEF_SLAVE              (0),//the number of
//amba_ahb_arbiter_h
.PRIORITY_SCHEME            (1),//amba arbiter
.ROUND_ROBIN_SCHEME         (0),
//amba_ahb_decoder_h
.ADDR_START_MAP             (ADDR_START_MAP),
.ADDR_END_MAP               (ADDR_END_MAP),
.ADDR_MASK                  (ADDR_MASK),
.ADDR_PREVILEGE_MAP     (5'b0000), //{N_SLVE{1
.ADDR_RW_MAP            (8'b11111111),
.DEC_SEL_ACTION         (1'b1)
)
u_ahb_lite_interconnect(
```

AHB interconnect

- Address and data bus widths
  - Address (W_ADDR): 32 bits
  - Data (W_ADDR): 32 bits

```
//-----------------------------------------
// AHB Interconnect
//-----------------------------------------
ahb_lite_interconnect
#( //amba_ahb parameter // {{{
.N_MASTER                   (N_MASTER),
.W_MASTER                   (W_MASTER),
.N_SLAVE                    (N_SLAVE),
.W_SLAVE                    (W_SLAVE),
.W_ADDR                     (32),
.W_DATA                     (32),
.WB_DATA                    (4),
.W_WB_DATA                  (2),
.NUM_DEF_MASTER             (0),//the number of
.NUM_DEF_SLAVE              (0),//the number of
//amba_ahb_arbiter_h
.PRIORITY_SCHEME            (1),//amba arbiter
.ROUND_ROBIN_SCHEME         (0),
//amba_ahb_decoder_h
.ADDR_START_MAP             (ADDR_START_MAP),
.ADDR_END_MAP               (ADDR_END_MAP),
.ADDR_MASK                  (ADDR_MASK),
.ADDR_PREVILEGE_MAP    (5'b0000), //{N_SLVE{1
.ADDR_RW_MAP           (8'b11111111),
.DEC_SEL_ACTION        (1'b1)
)
u_ahb_lite_interconnect(
```

Test bench

```
`timescale 1ns / 100ps
module top_system_tb;
        reg HCLK, HRESETn;                              //**** module input

        //reg sl_HSEL_alu;                              //**** select signals
        //reg sl_HSEL_multipliers;

        reg [3:0]       alu_op_i;                       //**** control signals
        reg [31:0]      alu_a_i, alu_b_i;
        reg [31:0]      alu_p_o;
```

```
top_system u_top_system                                 module instance for test bench
(
        .HRESETn(HRESETn)
        ,.HCLK(HCLK)
        //,.sl_HSEL_alu         (sl_HSEL_alu)
        //,.sl_HSEL_multiplier(sl_HSEL_multiplier)
);
```

```
initial begin                                           Initialize Inputs
        HCLK = 0;
        HRESETn = 0;
        #(p/2) HRESETn = 1;

        alu_a_i = 0;
        alu_b_i = 0;
        alu_p_o = 0;
```

Test cases

- Master accesses ALU
  - Comparison and addition requests.

```
//sl_HSEL_alu = 1'b0;                                          // No slave is selected
//sl_HSEL_multiplier = 1'b0;
```

```
#(8*p)
        //sl_HSEL_alu = 1'b1;                                  Select ALU &
        //sl_HSEL_multiplier = 1'b0;                           AHB read and write
#(8*p)
        alu_a_i = 32'h0;
        alu_b_i = 32'h0;
        alu_op_i = `ALU_SLT;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_A_I, alu_a_i);    // Write the first operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_B_I, alu_b_i);    // Write the second operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_OP_I, alu_op_i);  // Write the operation
#(4*p) u_top_system.u_riscv_dummy.task_AHBread (`RISCV_REG_ALU_P_O, alu_p_o);     // Read the result

#(8*p)
        alu_a_i = 32'h8;
        alu_b_i = 32'h8;
        alu_op_i = `ALU_ADD;
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_A_I, alu_a_i);    // Write the first operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_B_I, alu_b_i);    // Write the second operand
#(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_ALU_OP_I, alu_op_i);  // Write the operation
#(4*p) u_top_system.u_riscv_dummy.task_AHBread( `RISCV_REG_ALU_P_O, alu_p_o);     // Read the result
```

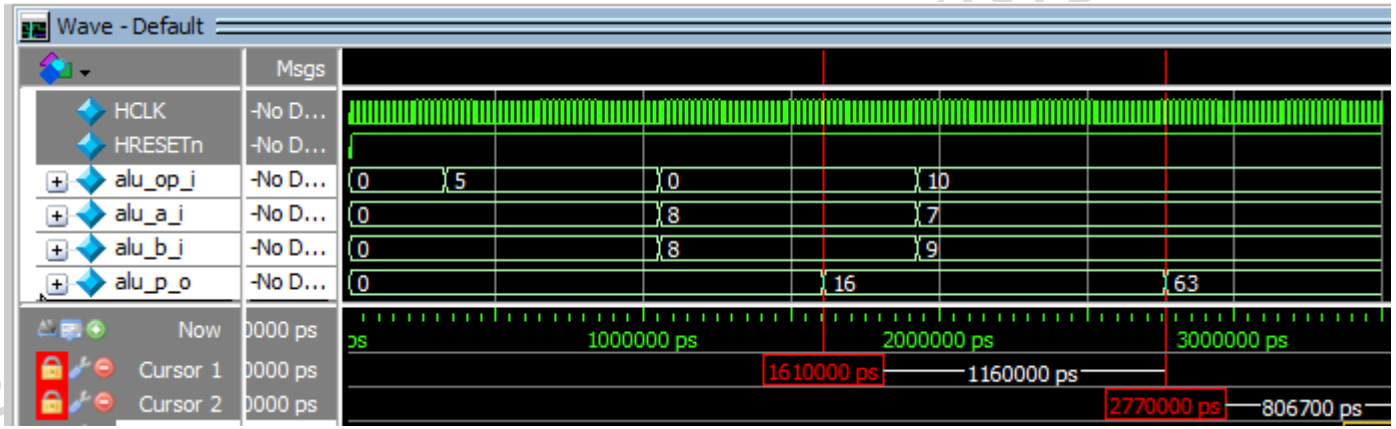
Test cases

- Master accesses Multiplier
  - A multiplication request.

```
    #(8*p)
            //sl_HSEL_alu = 1'b0;                                   Select multiplier &
            //sl_HSEL_multiplier = 1'b1;                            AHB read and write
    #(8*p)
            alu_a_i = 32'h7;
            alu_b_i = 32'h9;
            alu_op_i = `ALU_MULL;
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_A_I, alu_a_i     ); // Write the first operand
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_B_I, alu_b_i     ); // Write the second operand
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_A_SIGNED, alu_a_i); // Write the first operand
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_B_SIGNED, alu_b_i); // Write the second operand
    #(4*p) u_top_system.u_riscv_dummy.task_AHBwrite(`RISCV_REG_MUL_OP_I, alu_op_i   );    // Write the operation
    #(4*p) u_top_system.u_riscv_dummy.task_AHBread( `RISCV_REG_MUL_P_O_LOW, alu_p_o ); // Read the result

  end
```

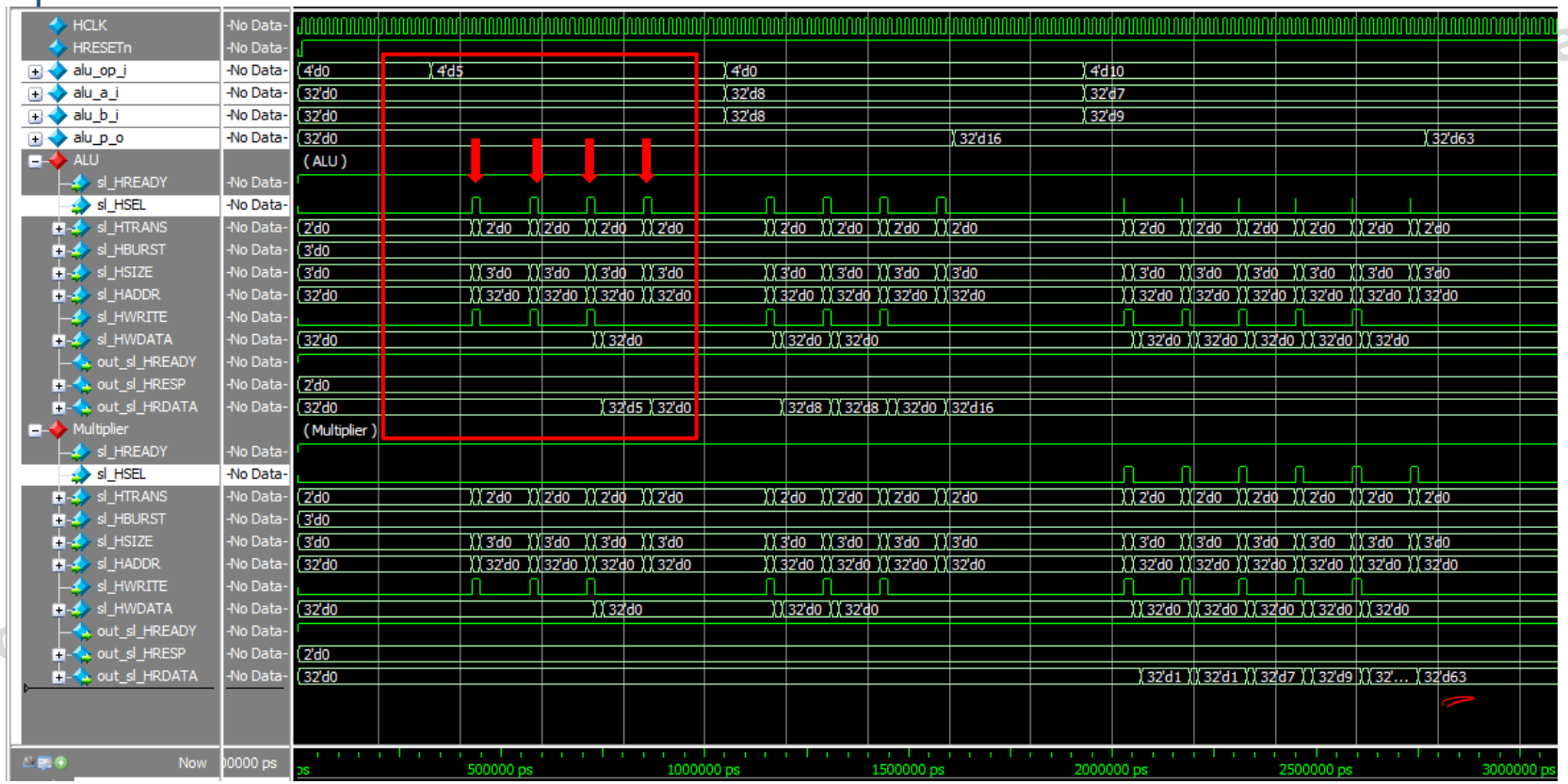
Waveform

- ALU Slave
  - SLT operation
  - ADD operation
- Multiplier Slave
  - Multiplier operation



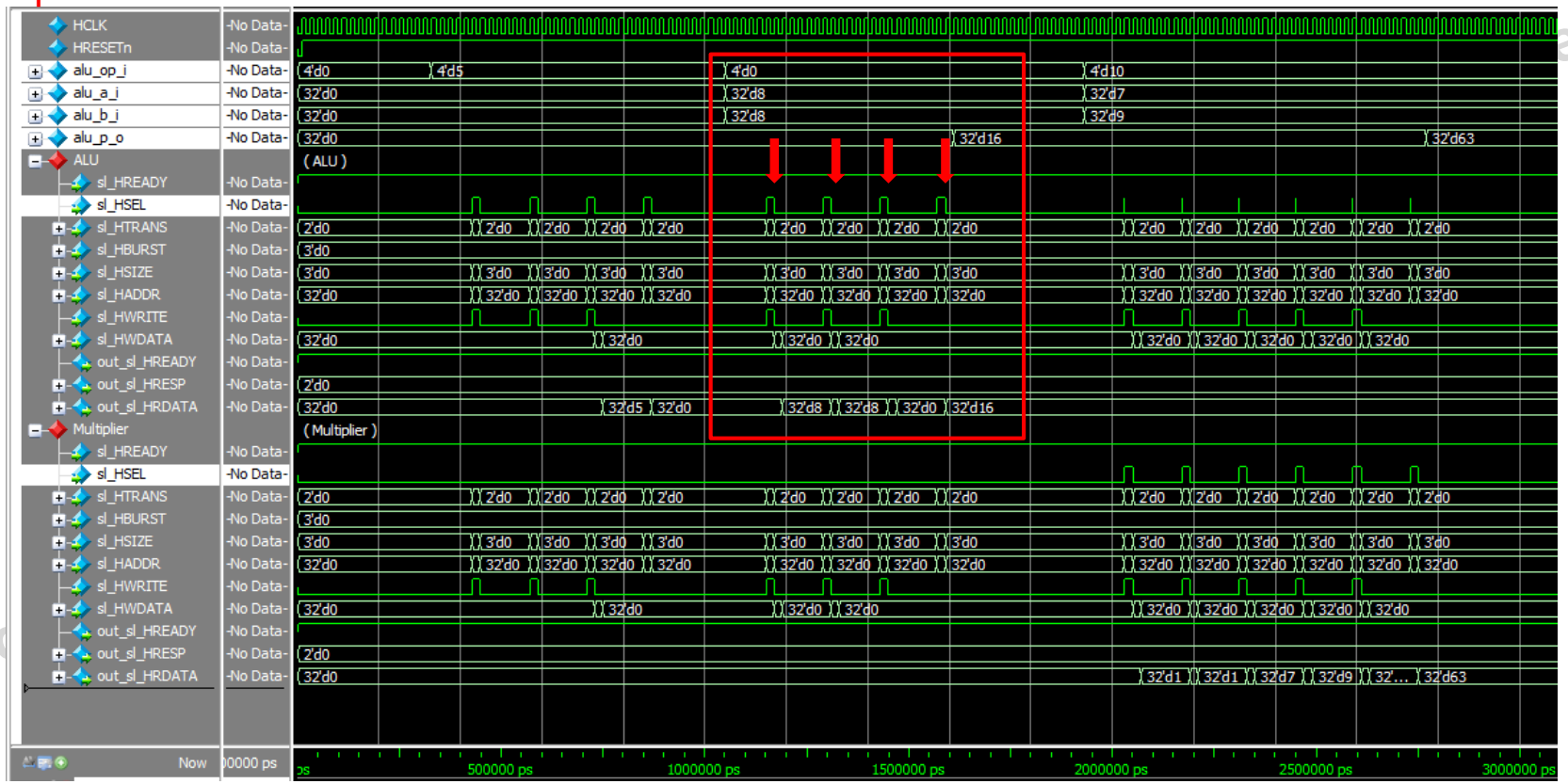
Waveform

- ALU Slave is selected, i.e. sl_HSEL = 1
  - SLT operation
  - ADD operation



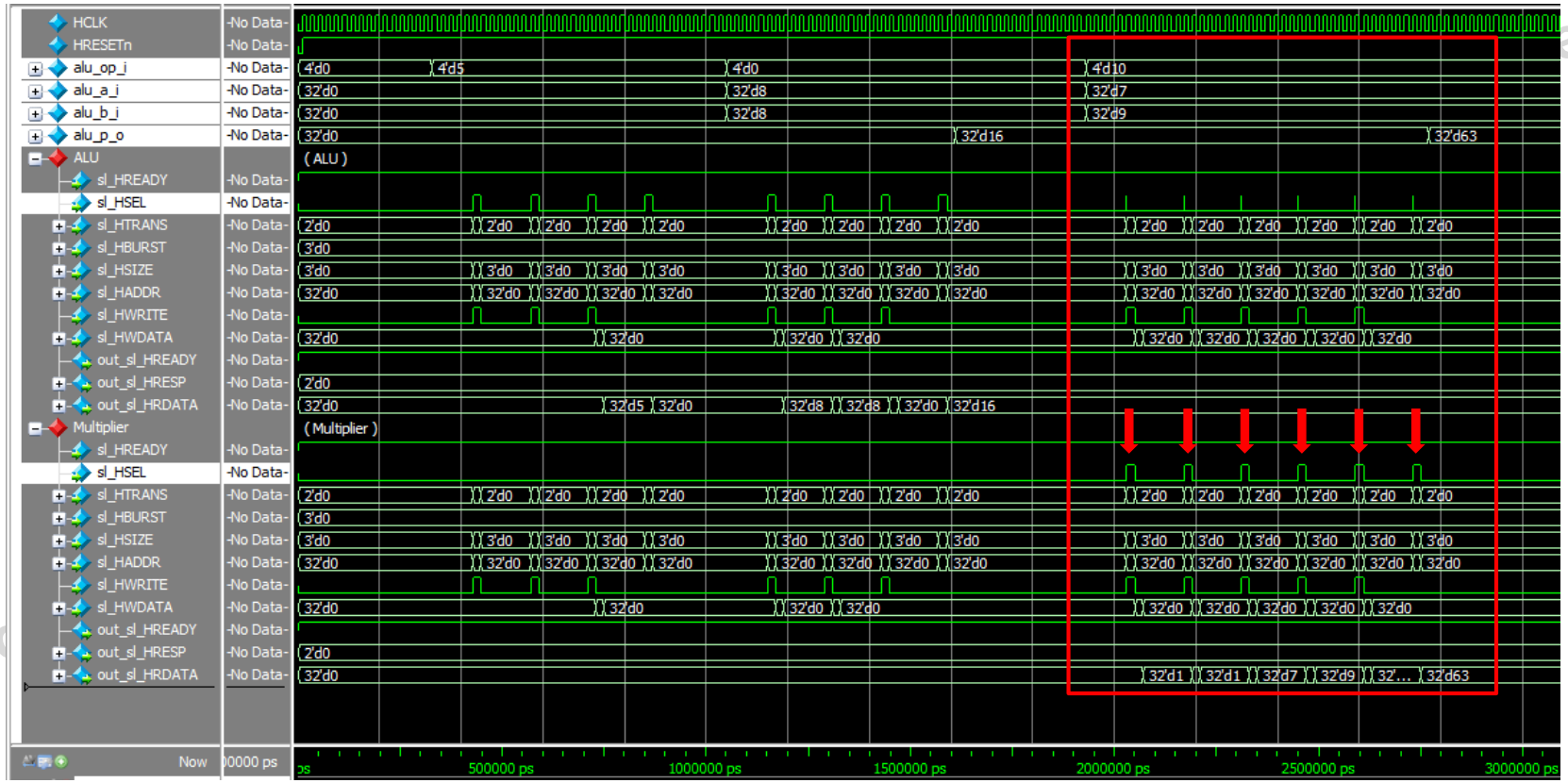
Waveform

- ALU Slave is selected, i.e. sl_HSEL = 1
  - SLT operation
  - ADD operation



Waveform

- Multiplier Slave is selected, i.e. sl_HSEL = 1
  - Multiplier operation



To do …

- AHB Interconnection
  - Complete the missing code : **top_system.v**
  - Do a simulation with time = 3,500ns
  - Show the output waveform
  - How to verify the results?
    - Test multiple operations for each slave
    - Check a waveform