**Homework 2: RISC-V core, AMBA Bus and IO**

**Issued: January 3rd (Tuesday), 2023 Due: January 9th (Monday), 2023**

**What to turn in**: Copy the text from your MODIFIED codes and paste it into a document. If a question asks you to plot or display something to the screen, also include the plot and screen output your code generates. Submit either a \*.doc or \*.pdf file.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**WARNING: The homework consists of EIGHT problems in total.**

**Problem 1 (25p): RISC-V**

Design a RISC-V core in Verilog. Please see the description in the lecture note.

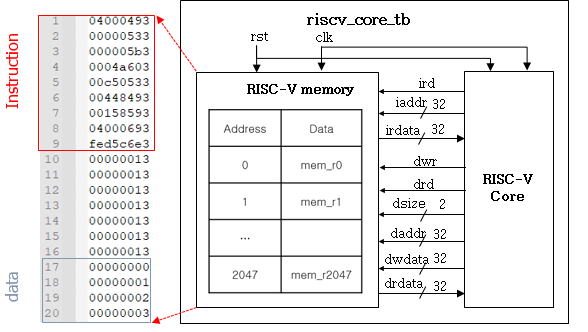


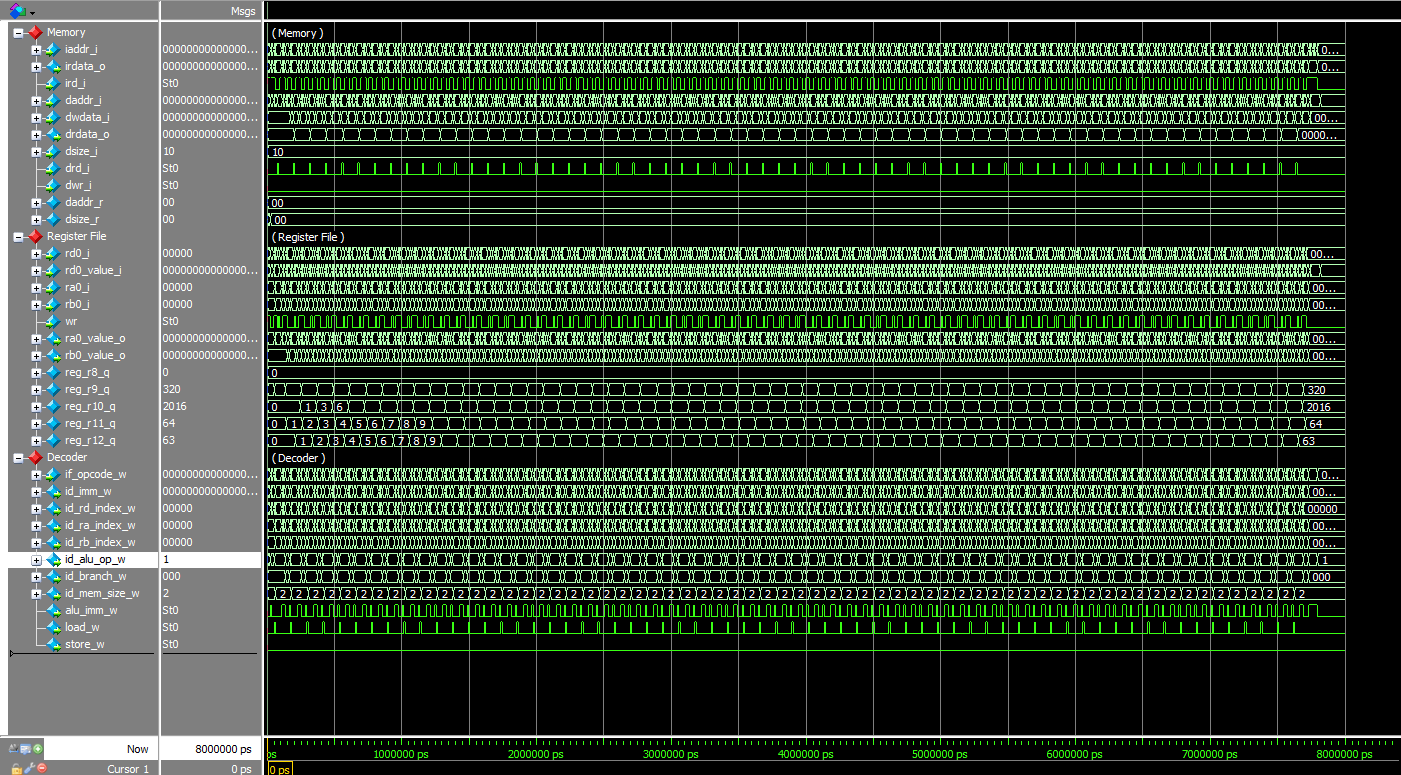
Fig. 1-1: RISC-V core

|  |  |  |
| --- | --- | --- |
| **C code** | **Assembly Code** | |
| 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 | # x9=&A[0]  # sum=0  # i=0  # x12=A[i]  # sum+= A[i]  # &A[i++]  # i++  # x13=64 |

What you have to do:

1. **Baseline (5p)**

* Do a simulation and capture its result, i.e., waveform. (3p)



* Explain why the final values of r9 and r10 are 320 and 2016, respectively. (2p)

Answer: Register 9(x9) and Register 10(x10) represent the index of A[i] or &A[i] and sum number respectively. x9 is initialized by 64 and will be added in the loop for 64 times. Every times it need to add by 4 because of the 32 bit of data, thus x9 = 64 + 64 \* 4 = 320; x10 is initialized by 0 and will be added in the loop for 64 times. Every times, it need to add by A[i], which is x12. In the machine code, the data is initialized by 0,1,2,…,63; Thus the x10 =

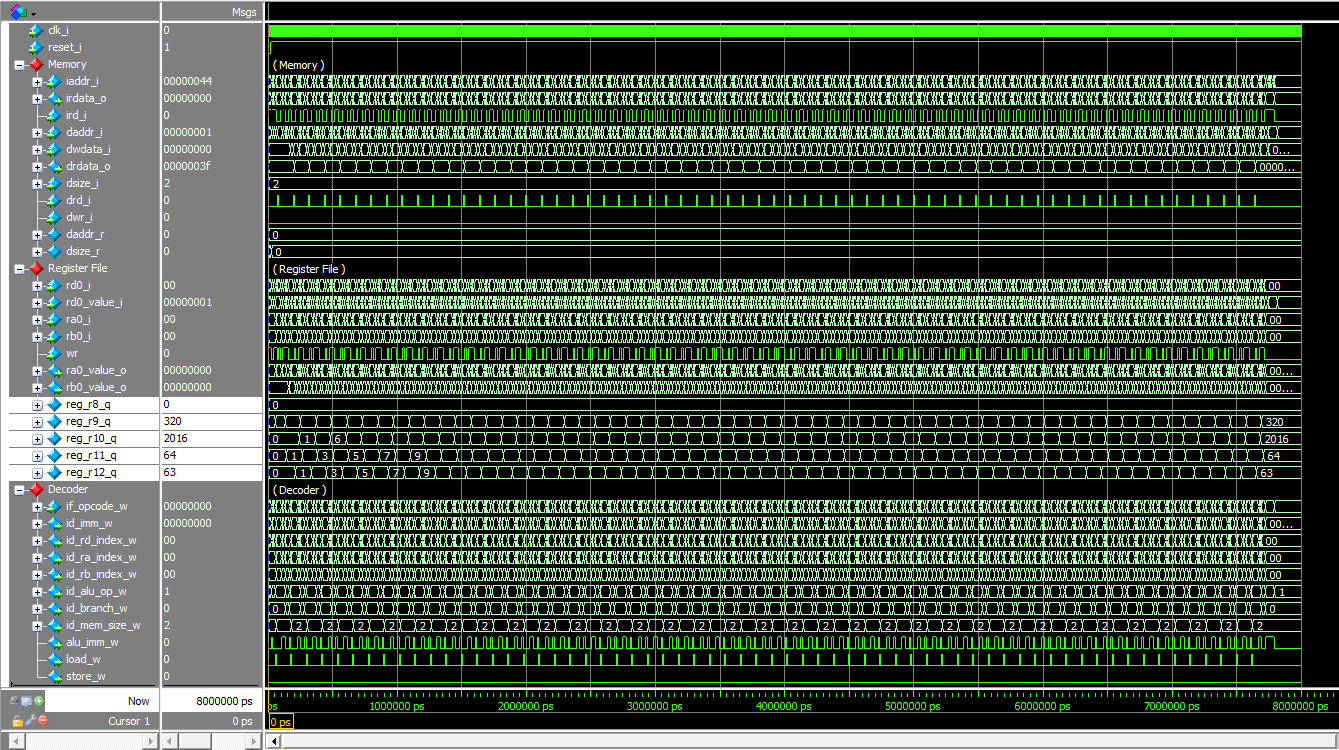
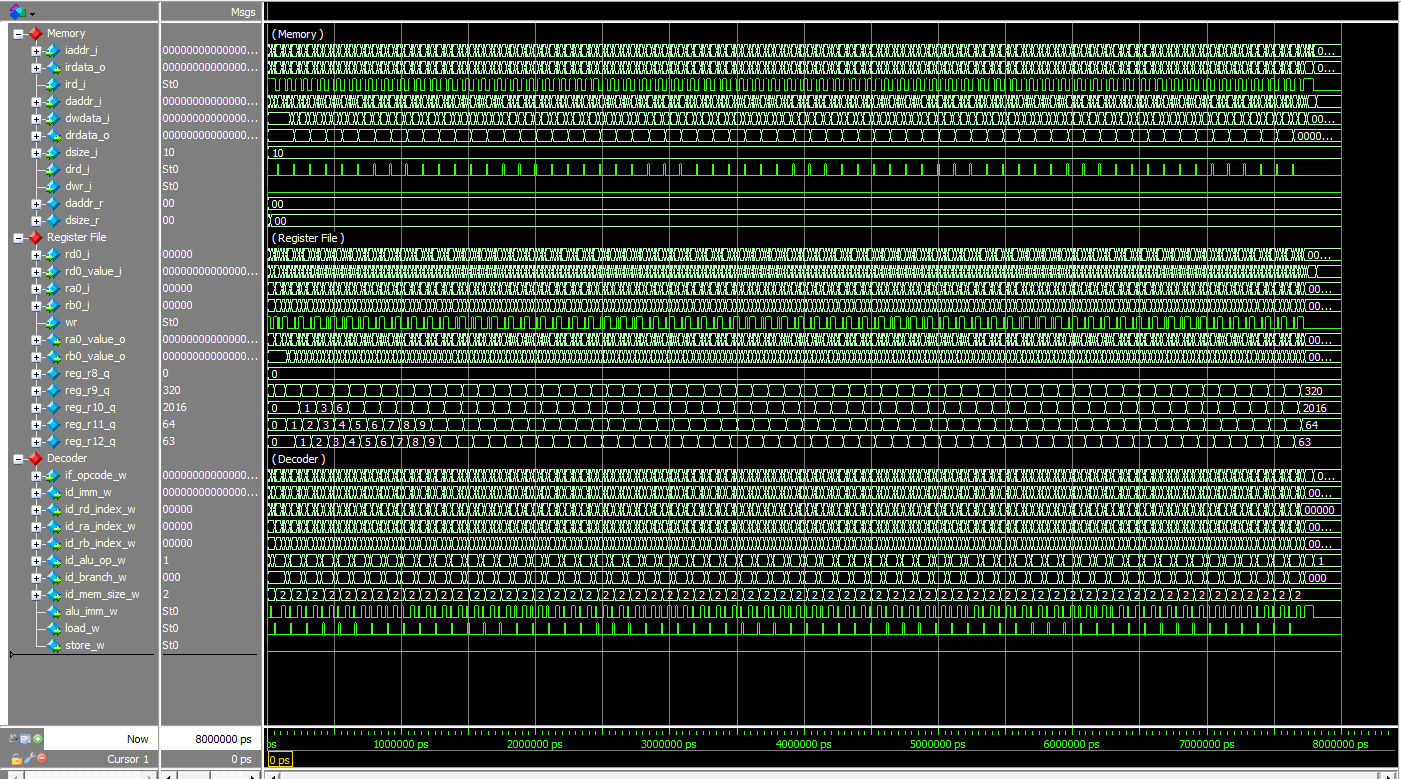


Fig. 1-2: Reference waveform for the baseline code.

1. **Branch-Not-Equal (BNE) (5p)**
2. Generate a memory file for the modified assembly code using a NOT-EQUAL operation, called mem\_bne.hex.
3. 04000493
4. 00000533
5. 000005b3
6. 0004a603
7. 00c50533
8. 00448493
9. 00158593
10. 04000693
11. fed596e3
12. Modify a test bench to use a new memory file (riscv\_core\_bne\_tb.v).
13. Make code to calculate a branch enable signal for Branch-Not-Equal (BNE).
14. Do a simulation and capture its result, i.e., waveform.



|  |  |  |
| --- | --- | --- |
| **C code** | **Assembly Code** | |
| 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  bne x11, x13, Loop | # x9=&A[0]  # sum=0  # i=0  # x12=A[i]  # sum+= A[i]  # &A[i++]  # i++  # x13=64 |

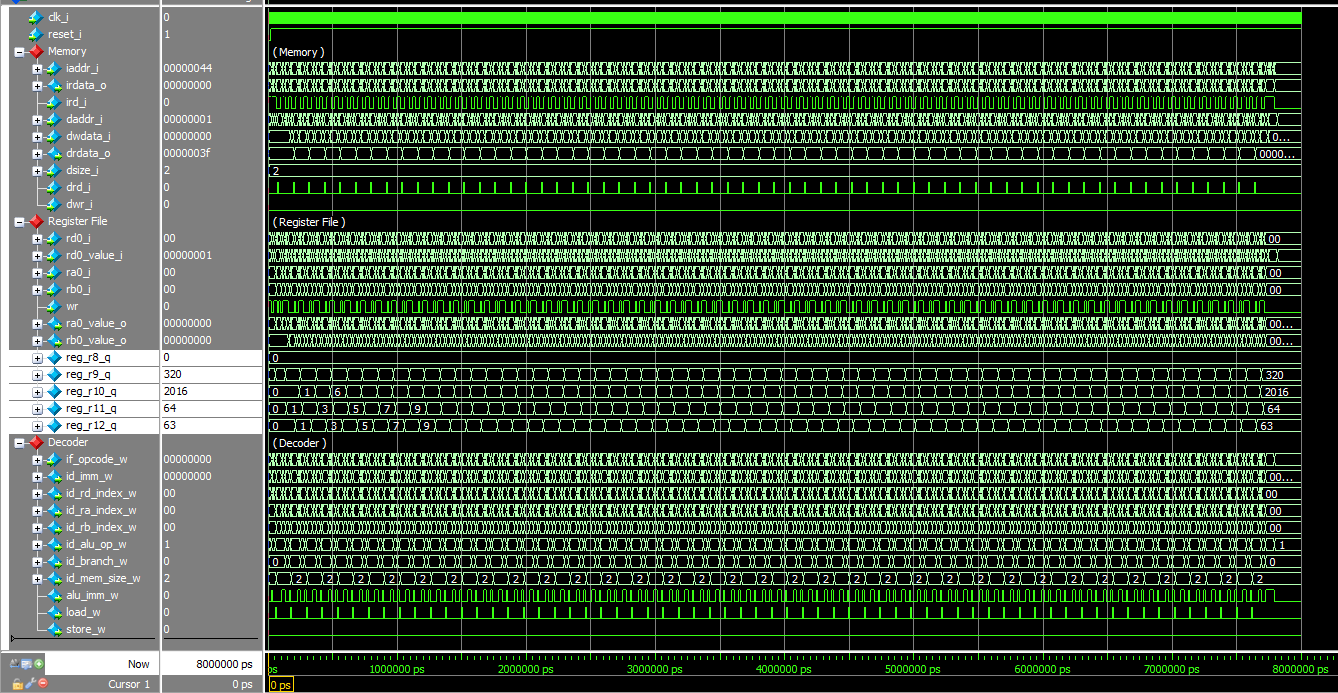
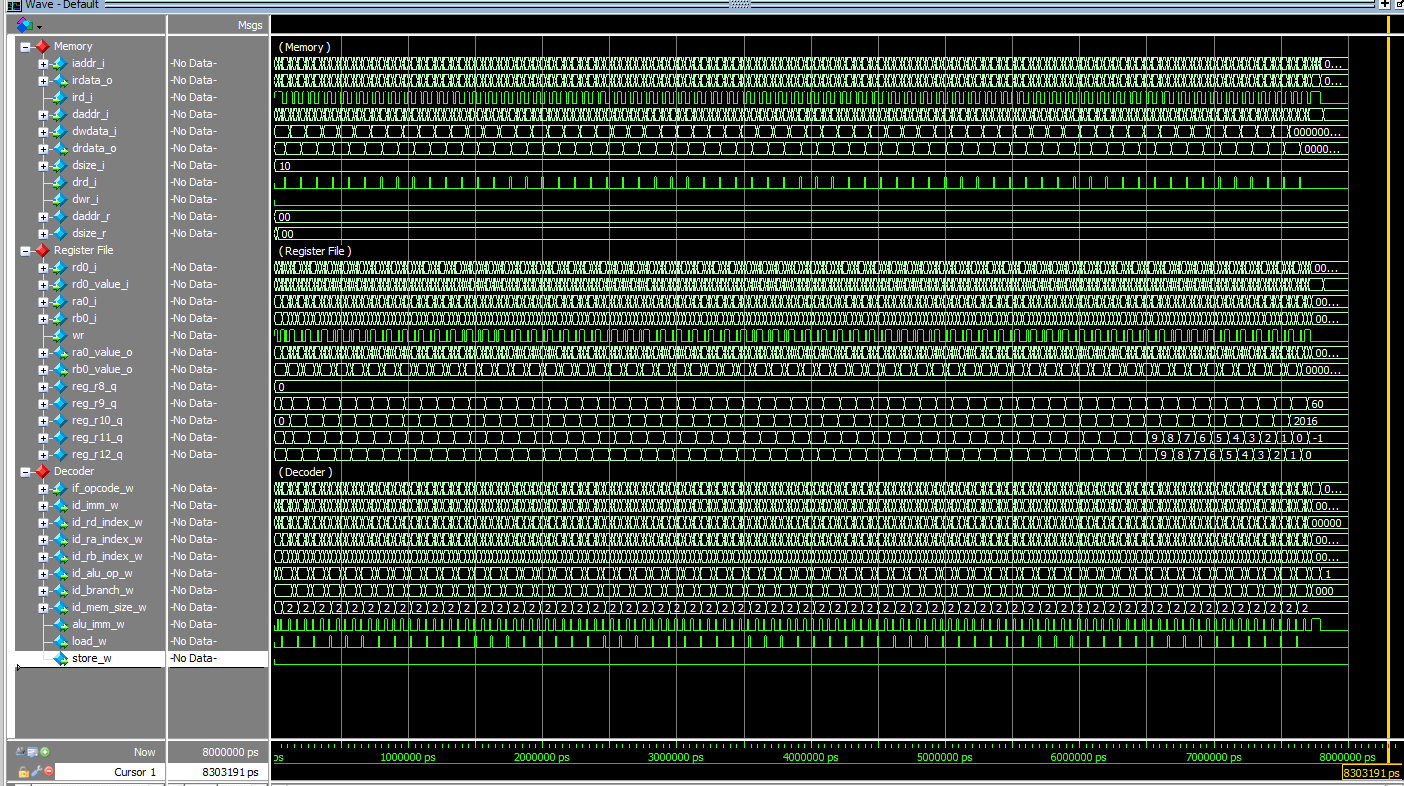


Fig. 1-3: Reference waveform for the “BNE” code

1. **Branch-Greater-or-Equal (BGE) (5p)**
2. Generate a memory file for the modified assembly code using a Greater-or-Equal operation, called mem\_bge.hex.
3. 13c00493
4. 00000533
5. 03f00593
6. 0004a603
7. 00c50533
8. ffc48493
9. fff58593
10. 000006b3
11. fed5d6e3
12. Modify a test bench to use a new memory file (riscv\_core\_bge\_tb.v).
13. Make code to calculate a branch enable signal for Branch-Greater-or-Equal (BGE).
14. Do a simulation and capture its result, i.e., waveform.



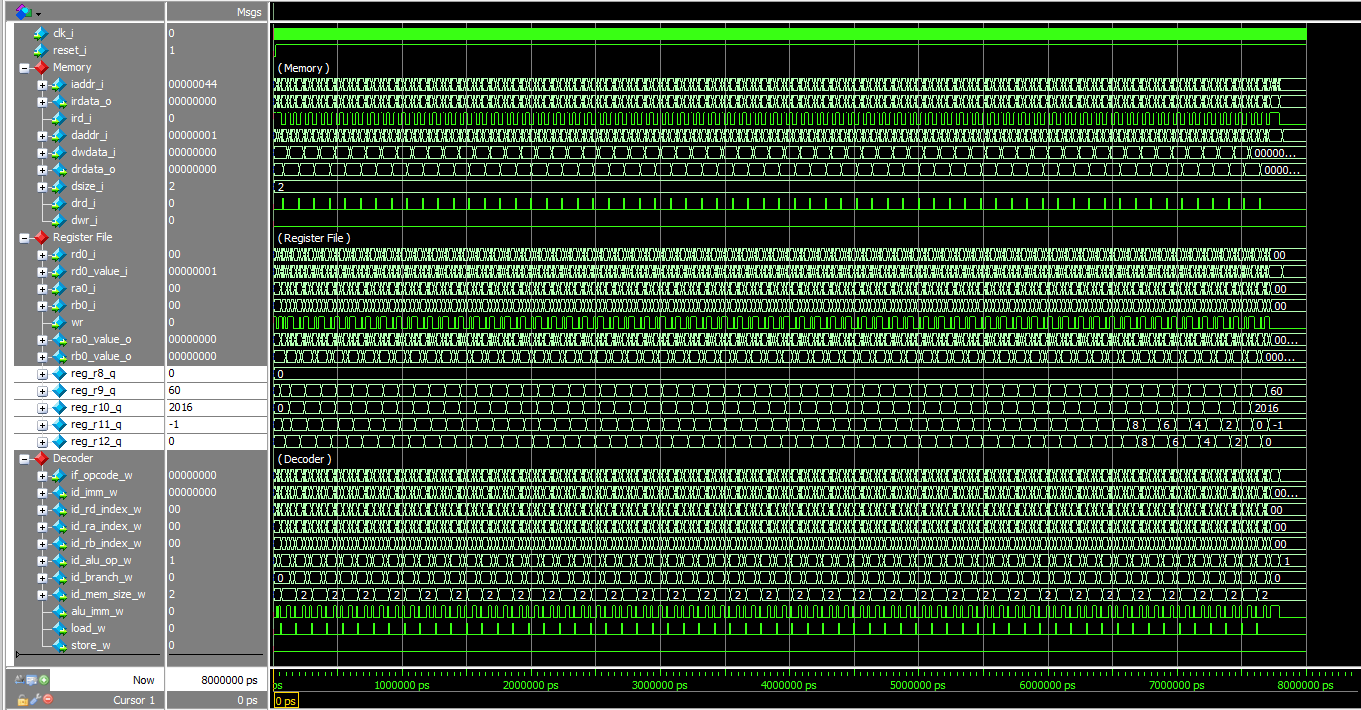


Fig. 1-4: Reference waveform for the “BGE” code

|  |  |  |
| --- | --- | --- |
| **C code** | **Assembly Code** | |
| int A[64];  int sum = 0;  for (int i=63; i>=0; i--)  sum += A[i]; | addi x9, x0, 316  add x10, x0, x0  addi x11, x0, 63  Loop:  lw x12, 0(x9)  add x10, x10, x12  addi x9, x9, -4  addi x11, x11, -1  add x13, x0, x0  bge x11, x13, Loop | # x9=&A[0]  # sum=0  # i=63  # x12=A[i]  # sum+= A[i]  # &A[i--]  # i--  # x13=0 |

1. **Instruction Reordering (10p)**

There are three versions of assembly codes as shown in Table.

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

Fig. 1-5 shows the waveform of the baseline version which executes the program in 7,815 ns.

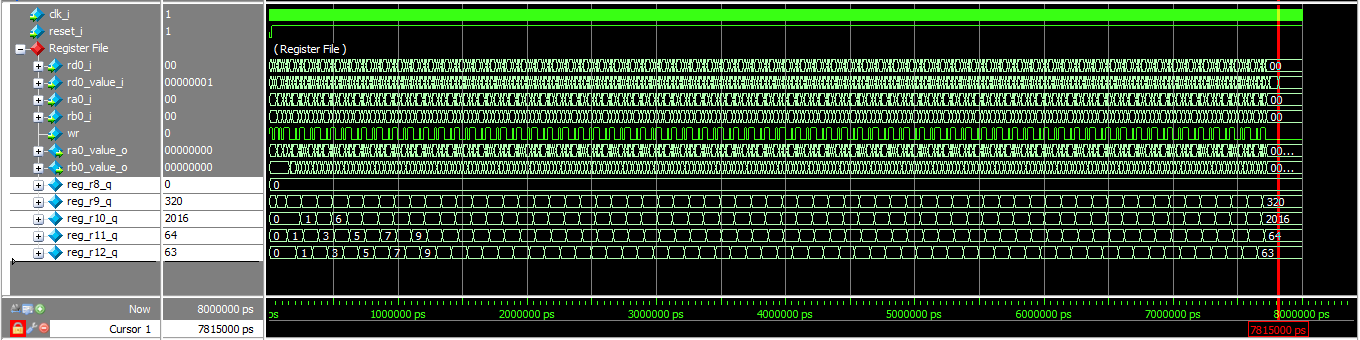


Fig. 1-5: Waveform of the **baseline** version (mem.hex)

What you have to do:

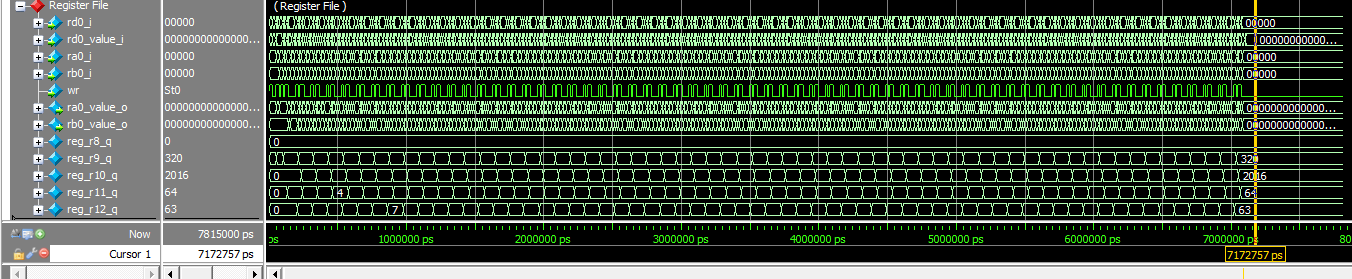
1. Create memory files (mem\_opt1.hex and mem\_opt2.hex).

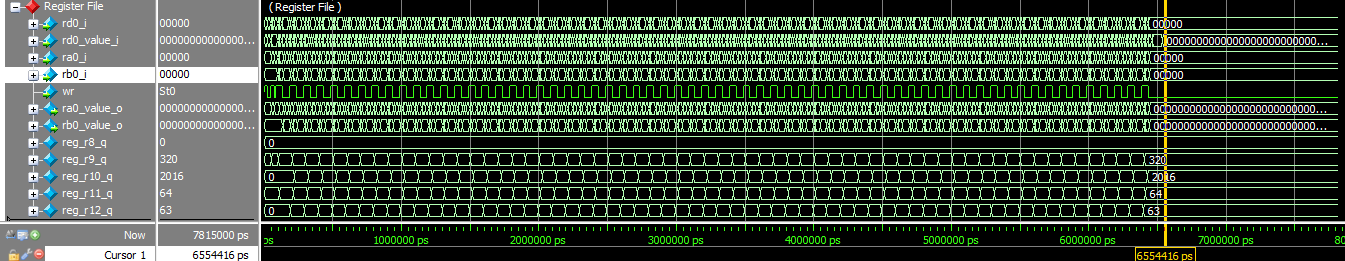
Mem\_opt1.hex

1. 04000493
2. 00000533
3. 000005b3
4. 0004a603
5. 00448493
6. 00c50533
7. 00158593
8. 04000693
9. fed5c6e3

Mem\_opt2.hex

1. 04000493
2. 00000533
3. 000005b3
4. 0004a603
5. 00448493
6. 00158593
7. 00c50533
8. 04000693
9. fed5c6e3
10. Modify a test bench with new memory files (riscv\_core\_tb.v)
11. Do a simulation and captures the waveform.





1. Compare the running times of three versions: baseline, opt1, and opt2. Explain the experimental results.

Answer: The difference is the `wr` signal. In the best option, the `wr` signal are more clear than the others. Because the option2 has moved the two `addi` before the command `add` thus the `x10 = x10 + x12` could get the `x12` immediately instead of reading the data.

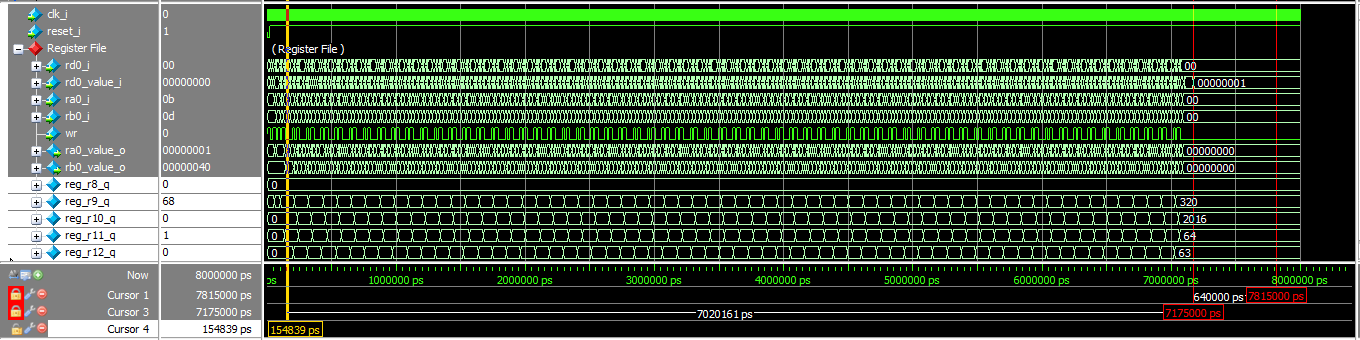


Fig. 1-6: Waveform of the **optimized version 1** (mem\_opt1.hex)

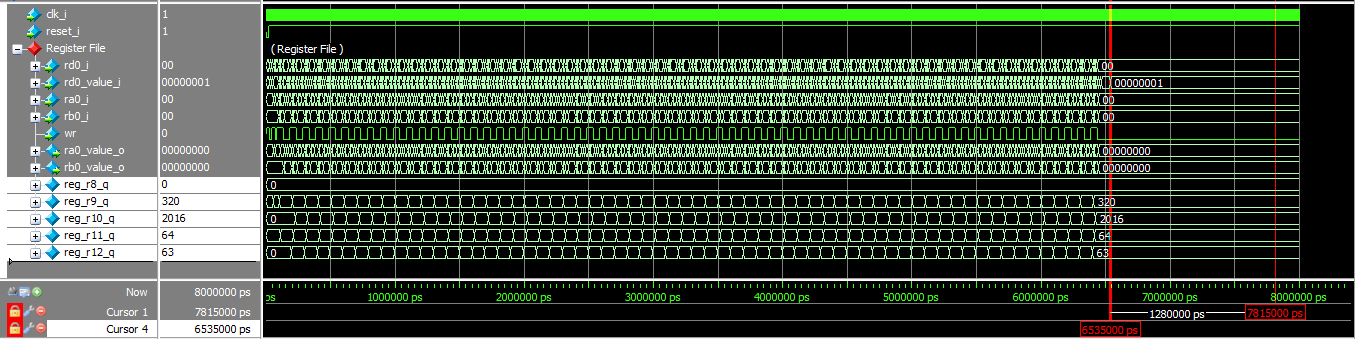


Fig. 1-7: Waveform of the **optimized version 2** (mem\_opt2.hex)

**Problem 2 (15p): ALU IP**

Implement an AHB slave interface of ALU IP. Please see the description in the lecture note for details.



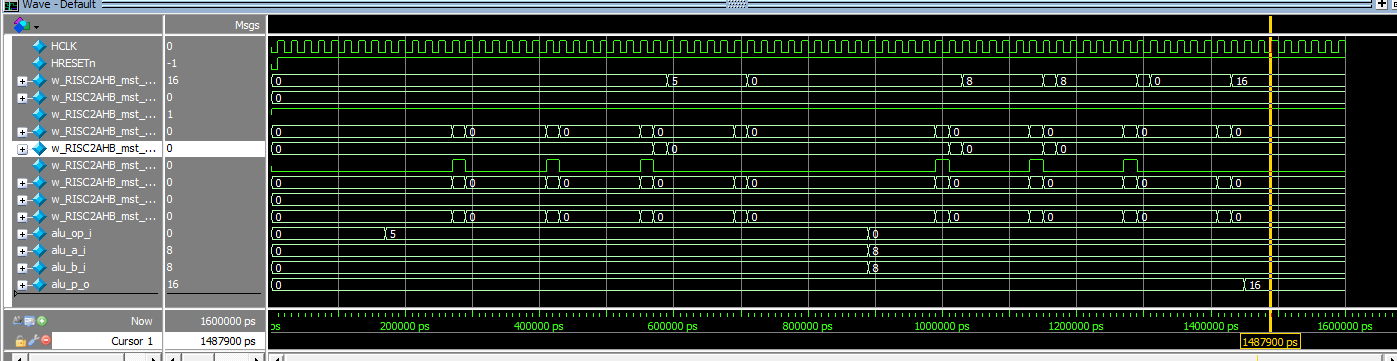
Fig. 2-1: An AHB wrapper of RISC-V ALU.

What you have to do:

* Design an AHB slave interface of ALU IP based on the baseline code.
* Submit your RTL files.

1. localparam REG\_ALU\_P\_O   = 3/\*Insert your code\*/;   //0x0c  ==> READ ONLY
2. begin
3. q\_sel\_sl\_reg <= sl\_HADDR[W\_REGS+W\_WB\_DATA-1:W\_WB\_DATA]/\*Insert your code\*/;
4. q\_ld\_sl\_reg  <= sl\_HWRITE/\*Insert your code\*/;
5. end
6. **case**(q\_sel\_sl\_reg)
7. REG\_ALU\_OP\_I:
8. alu\_op\_i <= sl\_HWDATA[3:0];
9. REG\_ALU\_A\_I:
10. alu\_a\_i <= sl\_HWDATA/\*Insert your code\*/;
11. REG\_ALU\_B\_I:
12. alu\_b\_i <= sl\_HWDATA/\*Insert your code\*/;
13. endcase
14. **case**(q\_sel\_sl\_reg)
15. REG\_ALU\_OP\_I:
16. out\_sl\_HRDATA = alu\_op\_i;
17. REG\_ALU\_A\_I:
18. out\_sl\_HRDATA = alu\_a\_i/\*Insert your code\*/;
19. REG\_ALU\_B\_I:
20. out\_sl\_HRDATA = alu\_b\_i/\*Insert your code\*/;
21. REG\_ALU\_P\_O:
22. out\_sl\_HRDATA = alu\_p\_o/\*Insert your code\*/;
23. endcase

* Capture the waveform.



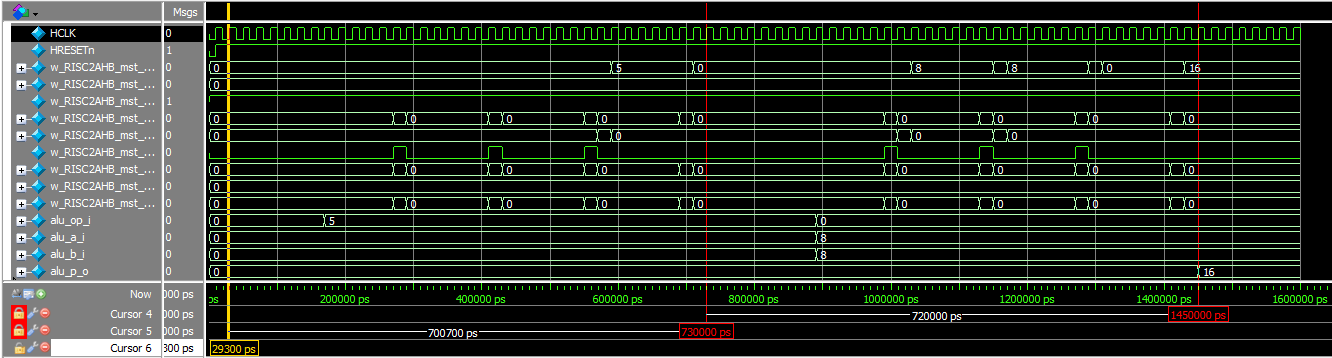


Fig. 2-2: Waveform

**Problem 3 (15p): Multi-AHB-Slave system**

Implement a multi-AHB-slave system. Please see the description in the lecture note for details.



Fig. 3-1: An AHB wrapper of RISC-V Multiplier.



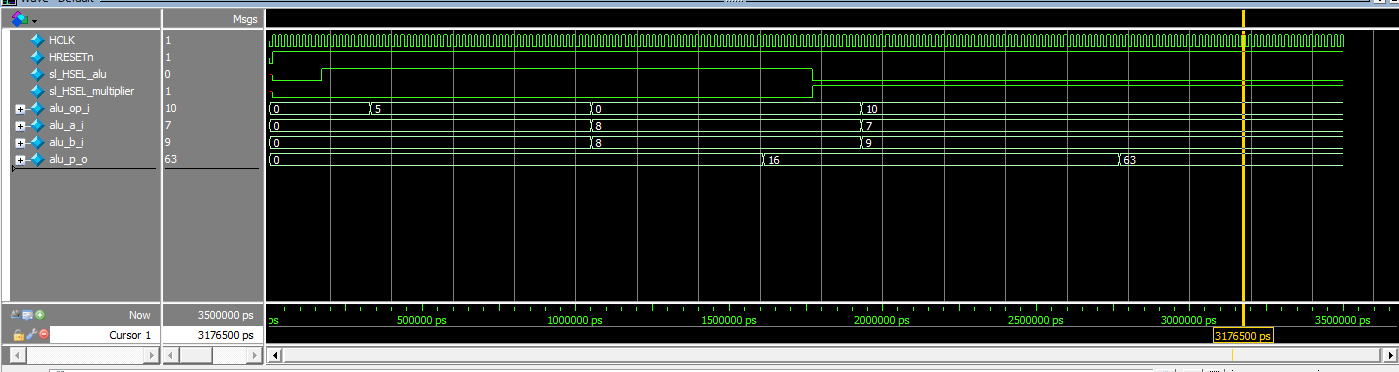
Fig. 3-2: Top module.

What you have to do:

* Design an AHB a system that includes a master and two slaves, i.e. ALU and multiplier.
* Reuse riscv\_alu\_if.v from Problem 2
* Complete the missing codes in riscv\_multiplier\_if.v and top\_system.v.
* Submit your RTL files.

1. /\*Insert your code\*/
2. localparam N\_REGS = 8;  // Number of registers
3. localparam W\_REGS = 3;  //log2(N\_REGS)
4. localparam REG\_MUL\_OP\_I     = 0;    //0x00
5. localparam REG\_MUL\_A\_SIGNED = 1;    //0x04
6. localparam REG\_MUL\_B\_SIGNED = 2;    //0x08
7. localparam REG\_MUL\_A\_I      = 3;    //0x0c
8. localparam REG\_MUL\_B\_I      = 4;    //0x08
9. localparam REG\_MUL\_P\_O\_LO   = 5;    //0x0c  <READ-ONLY>
10. localparam REG\_MUL\_P\_O\_HI   = 6;    //0x08  <READ-ONLY>
11. localparam REG\_MUL\_EX\_STALL = 7;    //0x08  <READ-ONLY>
12. begin
13. q\_sel\_sl\_reg <= sl\_HADDR[W\_REGS+W\_WB\_DATA-1:W\_WB\_DATA]/\*Insert your code\*/;
14. q\_ld\_sl\_reg  <= sl\_HWRITE/\*Insert your code\*/;
15. end
16. **case**(q\_sel\_sl\_reg)
17. REG\_MUL\_OP\_I: alu\_op\_i <= sl\_HWDATA;
18. REG\_MUL\_A\_I: alu\_a\_i <= sl\_HWDATA;
19. REG\_MUL\_B\_I: alu\_b\_i <= sl\_HWDATA/\*Insert your code\*/;
20. REG\_MUL\_A\_SIGNED: a\_signed <= sl\_HWDATA;
21. REG\_MUL\_B\_SIGNED: b\_signed <= sl\_HWDATA/\*Insert your code\*/;
22. endcase
23. **case**(q\_sel\_sl\_reg)
24. REG\_MUL\_OP\_I: out\_sl\_HRDATA = alu\_op\_i;
25. REG\_MUL\_A\_I: out\_sl\_HRDATA = alu\_a\_i;
26. REG\_MUL\_B\_I: out\_sl\_HRDATA = alu\_b\_i/\*Insert your code\*/;
27. REG\_MUL\_A\_SIGNED: out\_sl\_HRDATA = a\_signed;
28. REG\_MUL\_B\_SIGNED: out\_sl\_HRDATA = b\_signed/\*Insert your code\*/;
29. REG\_MUL\_P\_O\_LO: out\_sl\_HRDATA = alu\_p\_o[31:0];
30. REG\_MUL\_P\_O\_HI: out\_sl\_HRDATA =  alu\_p\_o[63:32];
31. REG\_MUL\_EX\_STALL: out\_sl\_HRDATA = ex\_stall\_mul\_w;
32. endcase

* Capture the waveform.



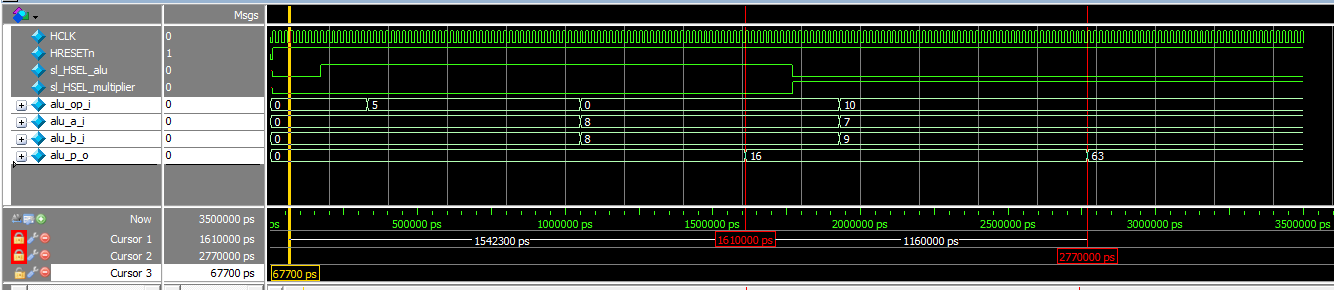


Fig. 3-3: Waveform.

**Problem 4 (10p): AHB Decoder**

1. AHB Decoder (5p)

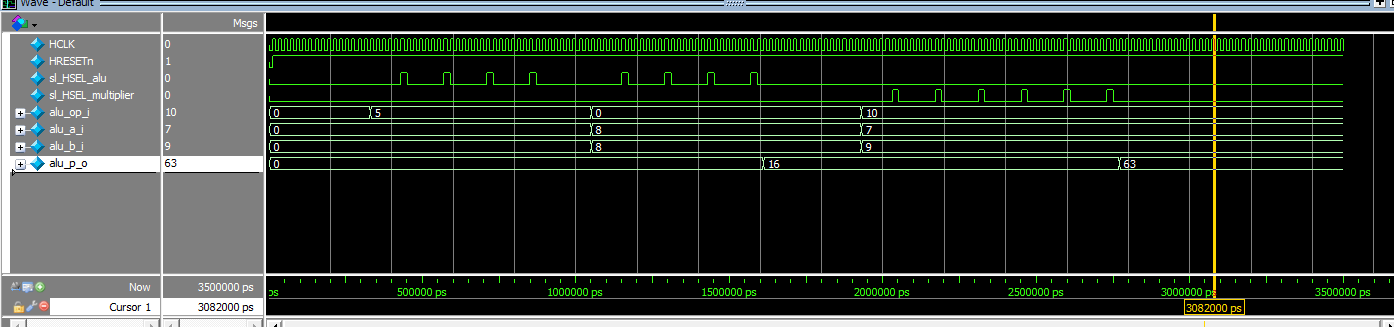
Implement an AHB decoder. Please see the description in the lecture note for details.

What you have to do:

* Design a simple AHB decoder.
* Reuse riscv\_alu\_if.v, riscv\_multiplier\_if.v from Problem 1 and 2.
* Complete the missing codes in top\_system.v.
* Submit your RTL files.

1. //Insert your code
2. **if**(w\_RISC2AHB\_mst\_HADDR>=`RISCV\_ALU\_BASE\_ADDR && w\_RISC2AHB\_mst\_HADDR < `RISCV\_MULTIPLIER\_BASE\_ADDR)
3. // Master accesses Multiplier
4. **else** begin
5. w\_RISC2AHB\_mst\_HRDATA   = w\_RISC2AHB\_mul\_HRDATA/\*Insert your code\*/  ;
6. w\_RISC2AHB\_mst\_HRESP    = w\_RISC2AHB\_mul\_HRESP       ;
7. w\_RISC2AHB\_mst\_HREADY   = w\_RISC2AHB\_mul\_HREADY      ;
8. end

* Capture the waveform.



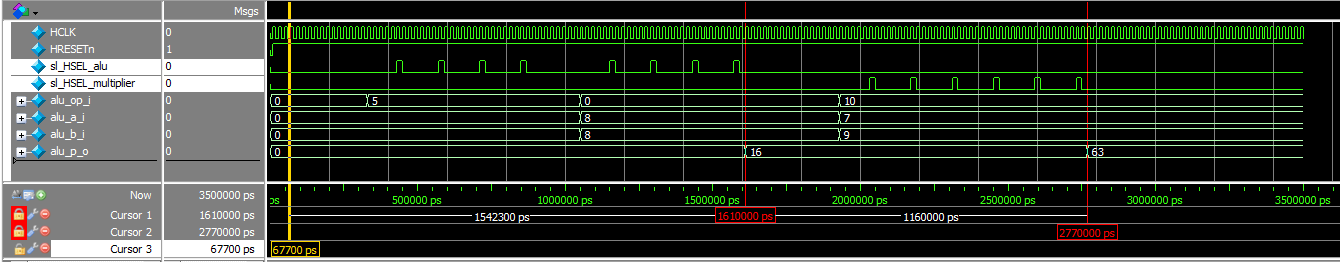


Fig. 4-1: Top module.

1. **Address decoding (5p)**

* Briefly explain the address decoding procedure by using “map.v” when a master sends a read or write request to a slave.

Answer: In the read or write request to a slave, we first indicate the slave to use. This step is completed in top system which could judge the slave address with if statement. And there are two tasks in top system (read and write), both of two tasks need to point out the target address, like `alu\_a\_i` or `alu\_p\_o`. These address are defined in map.v, thus we could read or write it in the relevant address.

* At the address decoding phase in an AHB slave, two registers q\_sel\_sl\_reg and q\_lg\_sl\_reg, are updated, where q\_sel\_sl\_reg and q\_lg\_sl\_reg stores the address of a register that a AHB master accesses, and the READ/WRITE mode, respectively. Briefly explain the codes.

Answer: The `q\_lg\_sl\_eg` is represent the READ/WRITE mode, thus it control the slave to read its address data value or write something into the address. `q\_sel\_sl\_reg` is to select which address to store the data. Thus, these two registers of AHB slave is to control the target address to read or write.

**Problem 5 (10p): AHB Bus Interconnection**

Implement a top system using an AHB Bus. Please see the description in the lecture note for details.



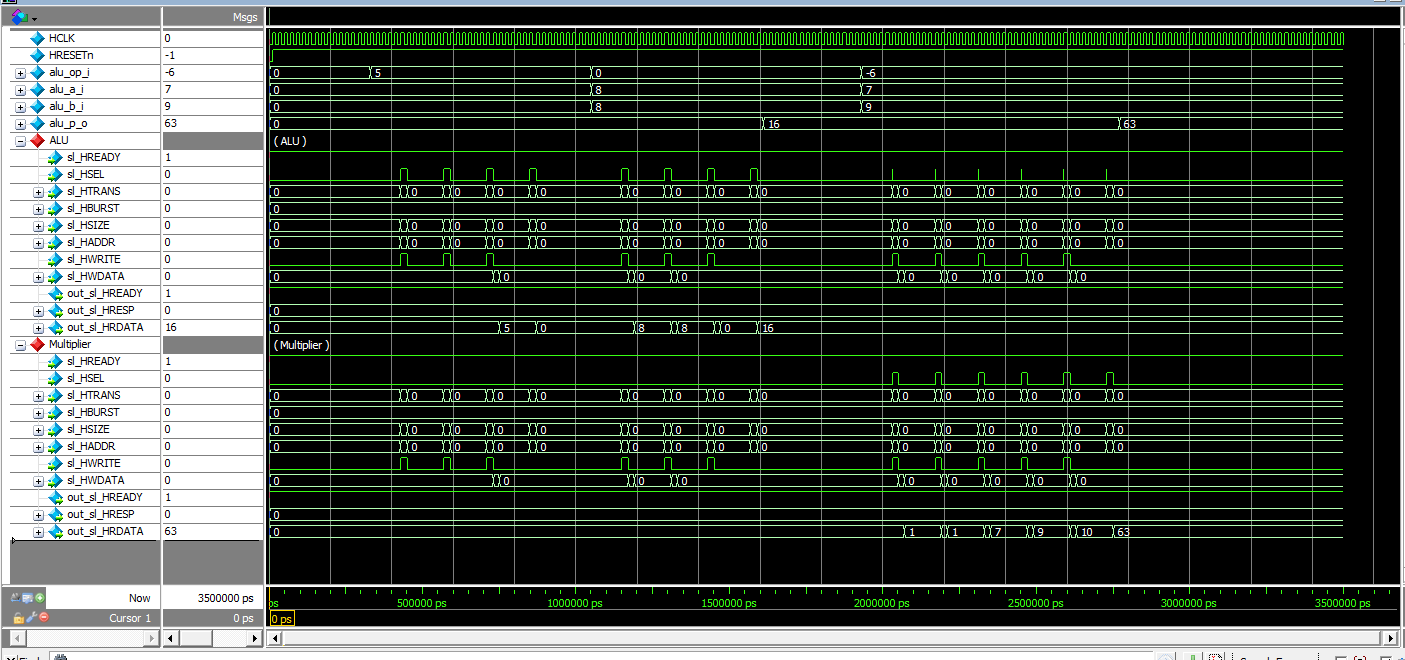
Fig. 5-1: A top system using an AHB bus.

What you have to do:

* Design a top stem using an AHB bus by completing the top module (top\_system.v).
* Submit your RTL files (top\_system.v).

1. parameter ADDR\_START\_MAP = {
2. /\* Insert your code\*/
3. `RISCV\_MULTIPLIER\_BASE\_ADDR,
4. `RISCV\_ALU\_BASE\_ADDR
5. };
6. parameter ADDR\_END\_MAP   = {
7. `RISCV\_MULTIPLIER\_BASE\_ADDR,
8. `RISCV\_ALU\_BASE\_ADDR
9. };
10. parameter ADDR\_MASK = {
11. /\* Insert your code\*/
12. `RISCV\_BASE\_ADDRESS\_MASK,
13. `RISCV\_BASE\_ADDRESS\_MASK
14. };
15. //  2. AHB2MULTIPLIER
16. assign  mul\_sl\_HSEL                     =   w\_AHB\_IC\_out\_sl\_HSEL    [1]/\* Insert your code \*/;
17. assign  mul\_sl\_HADDR                    =   w\_AHB\_IC\_out\_sl\_HADDR   [1\*32+:32]/\* Insert your code \*/;
18. assign  mul\_sl\_HTRANS                   =   w\_AHB\_IC\_out\_sl\_HTRANS  [1\*2+:2]/\* Insert your code \*/;
19. assign  mul\_sl\_HBURST                   =   w\_AHB\_IC\_out\_sl\_HBURST  [1\*`W\_BURST+:`W\_BURST]/\* Insert your code \*/;
20. assign  mul\_sl\_HSIZE                    =   w\_AHB\_IC\_out\_sl\_HSIZE   [1\*3+:3]/\* Insert your code \*/;
21. assign  mul\_sl\_HPROT                    =   w\_AHB\_IC\_out\_sl\_HPROT   [1\*4+:4]/\* Insert your code \*/;
22. assign  mul\_sl\_HWRITE                   =   w\_AHB\_IC\_out\_sl\_HWRITE  [1]/\* Insert your code \*/;
23. assign  mul\_sl\_HWDATA                   =   w\_AHB\_IC\_out\_sl\_HWDATA  [63:32]/\* Insert your code \*/;
24. assign  mul\_sl\_HREADY                   =   w\_AHB\_IC\_out\_sl\_HREADY  [1]/\* Insert your code \*/;
25. assign  w\_AHB\_IC\_sl\_HREADY  [1]         =   out\_mul\_sl\_HREADY/\* Insert your code \*/;
26. assign  w\_AHB\_IC\_sl\_HRESP   [1\*2+:2]    =   out\_mul\_sl\_HRESP/\* Insert your code \*/;
27. assign  w\_AHB\_IC\_sl\_HRDATA  [1\*32+:32]  =   out\_mul\_sl\_HRDATA/\* Insert your code \*/;
29. // ALU
30. riscv\_alu\_if u\_riscv\_alu\_if (
31. .HCLK(HCLK),
32. .HRESETn(HRESETn),
33. .sl\_HREADY(alu\_sl\_HREADY),
34. .sl\_HSEL(alu\_sl\_HSEL),
35. .sl\_HTRANS(alu\_sl\_HTRANS),
36. .sl\_HBURST(alu\_sl\_HBURST),
37. .sl\_HSIZE(alu\_sl\_HSIZE),
38. .sl\_HADDR(alu\_sl\_HADDR),
39. .sl\_HWRITE(alu\_sl\_HWRITE),
40. .sl\_HWDATA(alu\_sl\_HWDATA),
41. .out\_sl\_HREADY(out\_alu\_sl\_HREADY/\* Insert your code\*/),
42. .out\_sl\_HRESP (out\_alu\_sl\_HRESP/\* Insert your code\*/),
43. .out\_sl\_HRDATA(out\_alu\_sl\_HRDATA/\* Insert your code\*/)
44. );
45. // Multiplier
46. riscv\_multiplier\_if u\_riscv\_multiplier\_if (
47. .HCLK(HCLK),
48. .HRESETn(HRESETn),
49. .sl\_HREADY(mul\_sl\_HREADY),
50. .sl\_HSEL(mul\_sl\_HSEL),
51. .sl\_HTRANS(mul\_sl\_HTRANS),
52. .sl\_HBURST(mul\_sl\_HBURST),
53. .sl\_HSIZE(mul\_sl\_HSIZE),
54. .sl\_HADDR(mul\_sl\_HADDR),
55. .sl\_HWRITE(mul\_sl\_HWRITE),
56. .sl\_HWDATA(mul\_sl\_HWDATA),
57. .out\_sl\_HREADY(out\_mul\_sl\_HREADY/\* Insert your code\*/),
58. .out\_sl\_HRESP (out\_mul\_sl\_HRESP/\* Insert your code\*/),
59. .out\_sl\_HRDATA(out\_mul\_sl\_HRDATA/\* Insert your code\*/)
60. );

* Capture the waveform.



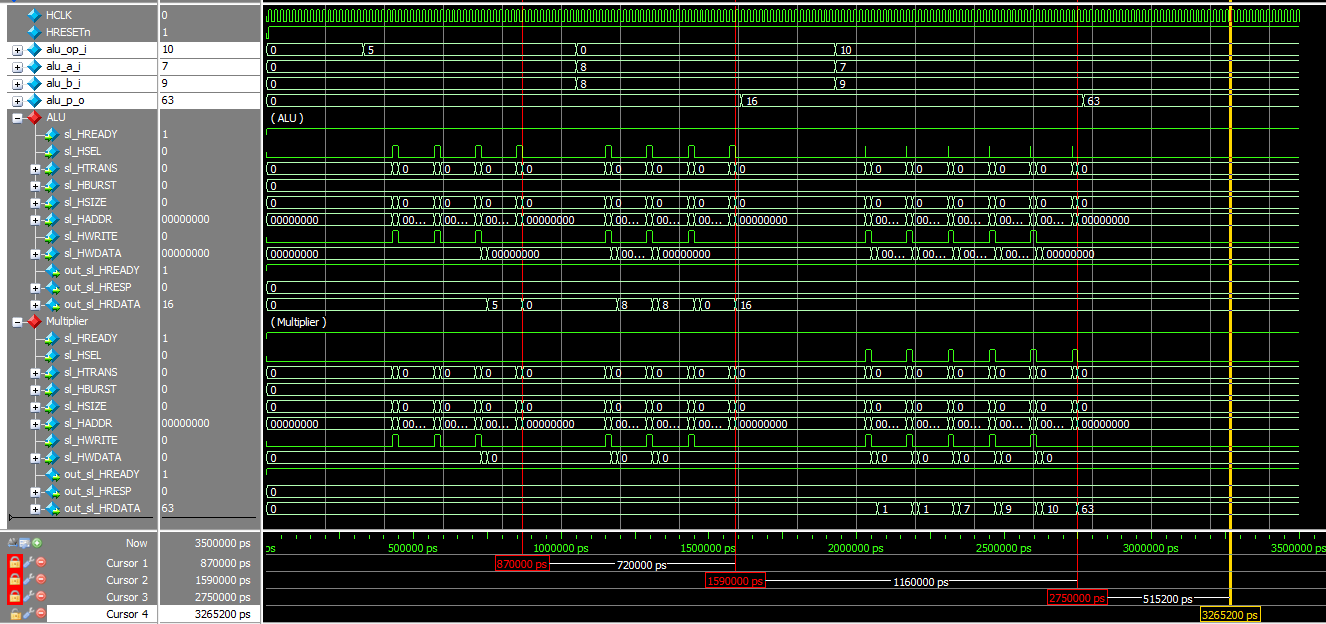


Fig. 5-2: Waveform

**Problem 6 (10p): LCD Drive**

Implement an LCD drive system. Please see the description in the lecture note for details.

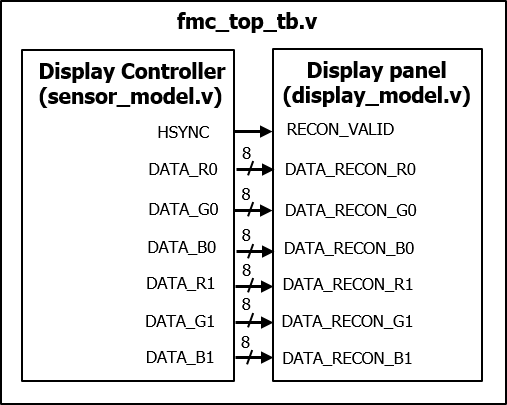


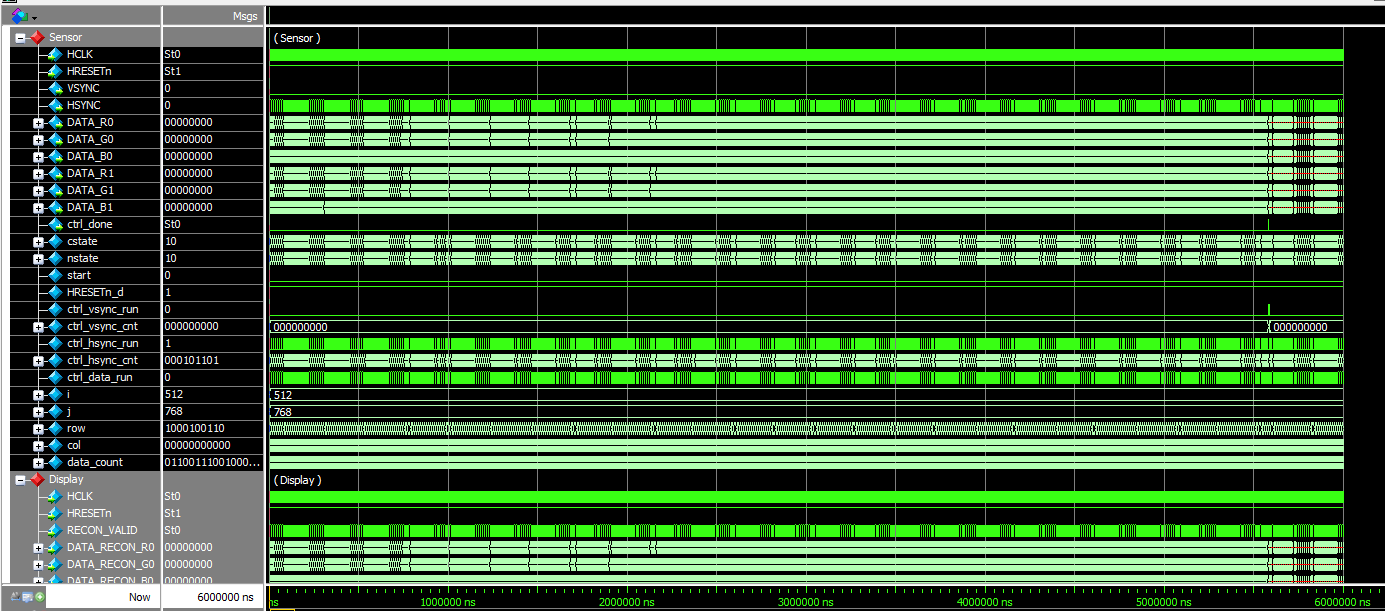
Fig. 6-1: Top test-bench module.

What you have to do:

* Design an LCD drive system by completing the missing codes in the display controller (sensor\_model.v) and the display model (display\_model.v).
* Submit your RTL files.

1. /\*Insert your code\*/
2. out\_BMP[WIDTH\*3\*(HEIGHT-l-1)+6\*m+4] <= DATA\_RECON\_G1;
3. out\_BMP[WIDTH\*3\*(HEIGHT-l-1)+6\*m+3] <= DATA\_RECON\_B1;
4. /\* Insert your code here \*/
5. org\_B[WIDTH\*i+j] = temp\_BMP[WIDTH\*3\*(HEIGHT-i-1)+3\*j+2];
6. ST\_HSYNC: begin
7. **if**(ctrl\_hsync\_cnt == HSYNC\_DELAY/\* Insert your code here \*/)
8. nstate = ST\_DATA/\* Insert your code here \*/;
9. **else**
10. nstate = ST\_HSYNC;
11. end
12. ST\_DATA: begin
13. **if**(ctrl\_done) begin   //end of frame
14. nstate = ST\_VSYNC/\* Insert your code here \*/;
15. end
16. **else** begin
17. **if**(col == WIDTH - 2)    //end of line
18. nstate = ST\_HSYNC;
19. **else**
20. nstate = ST\_DATA;
21. end
22. end
23. **if**(ctrl\_data\_run) begin
24. VSYNC   = 1'b0;
25. HSYNC   = 1'b1;
26. DATA\_R0 = org\_R[WIDTH \* row + col   ];
27. DATA\_G0 = org\_G[WIDTH \* row + col   ];
28. /\* Insert your code here \*/
29. DATA\_B0 = org\_B[WIDTH \* row + col   ];
30. DATA\_R1 = org\_R[WIDTH \* row + col +1];
31. DATA\_G1 = org\_G[WIDTH \* row + col +1];
32. /\* Insert your code here \*/
33. DATA\_B1 = org\_B[WIDTH \* row + col +1];
34. end

* Capture the results, including the waveform and the output image.





**Problem 7 (10p): Brightness Adjustment**

Implement an LCD drive system with a brightness adjustment module. Please see the description in the lecture note for details.



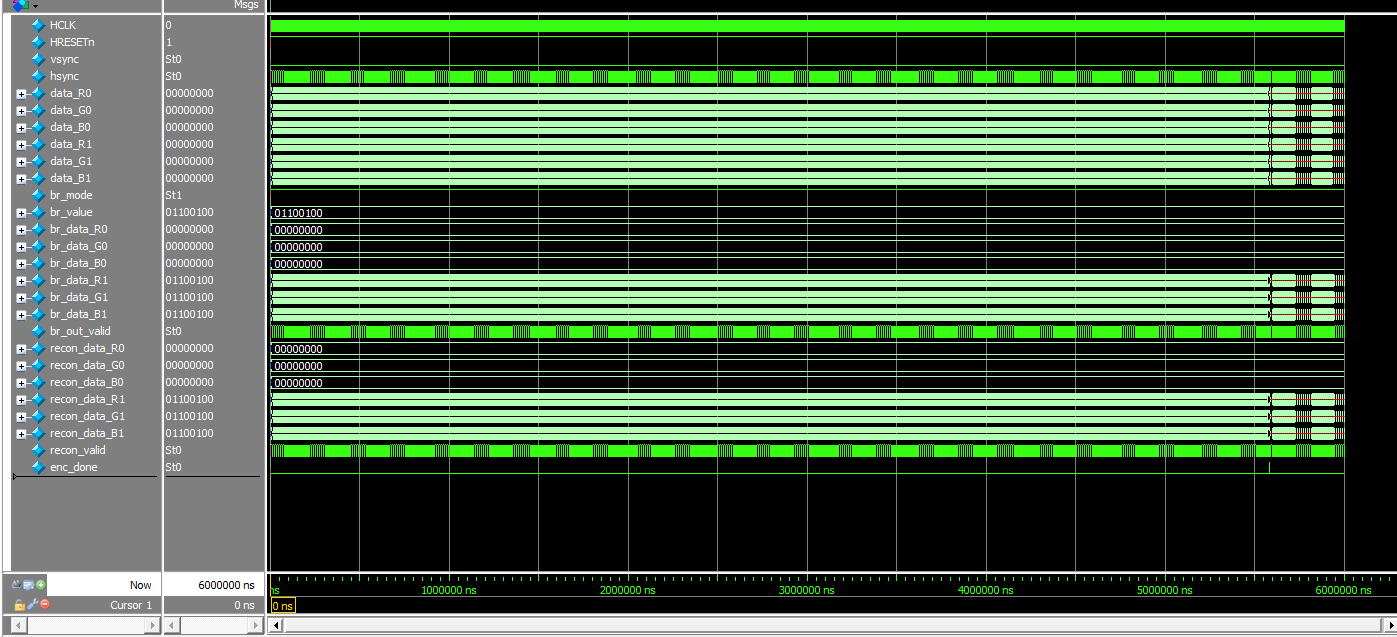
Fig. 7-1: Top test-bench module.

What you have to do:

* Design an LCD drive system with a brightness adjustment module by:
  + Reusing the controller (sensor\_model.v) and the display model (display\_model.v) in Problem 6.
  + Completing the missing code in brightness\_adjustment.v.
* Submit your RTL files.

1. out\_r1 <= ({1'b0, r1} + {1'b0, value} > 255) ? 255 : r1 + value/\* Insert your code here\*/;
2. out\_g1 <= ({1'b0, g1} + {1'b0, value} > 255) ? 255 : g1 + value/\* Insert your code here\*/;
3. out\_b1 <= ({1'b0, b1} + {1'b0, value} > 255) ? 255 : b1 + value/\* Insert your code here\*/;
4. out\_r1 <= (r1 < value) ? 0 : r1 - value/\* Insert your code here\*/;
5. out\_g1 <= (g1 < value) ? 0 : g1 - value/\* Insert your code here\*/;
6. out\_b1 <= (b1 < value) ? 0 : b1 - value/\* Insert your code here\*/;

* Capture the results, including the waveform and the output images.



|  |  |
| --- | --- |
| C:\Users\BRL\Desktop\ModelSim\Homework2\Brightness_Adjustment\out\kodim03.bmp | C:\Users\BRL\Desktop\ModelSim\Homework2\Brightness_Adjustment\out\kodim03.bmp |
| C:\Users\BRL\Desktop\ModelSim\Homework2\Brightness_Adjustment\out\kodim03.bmp | C:\Users\BRL\Desktop\ModelSim\Homework2\Brightness_Adjustment\out\kodim03.bmp |



Fig. 7-2: Simulation results.

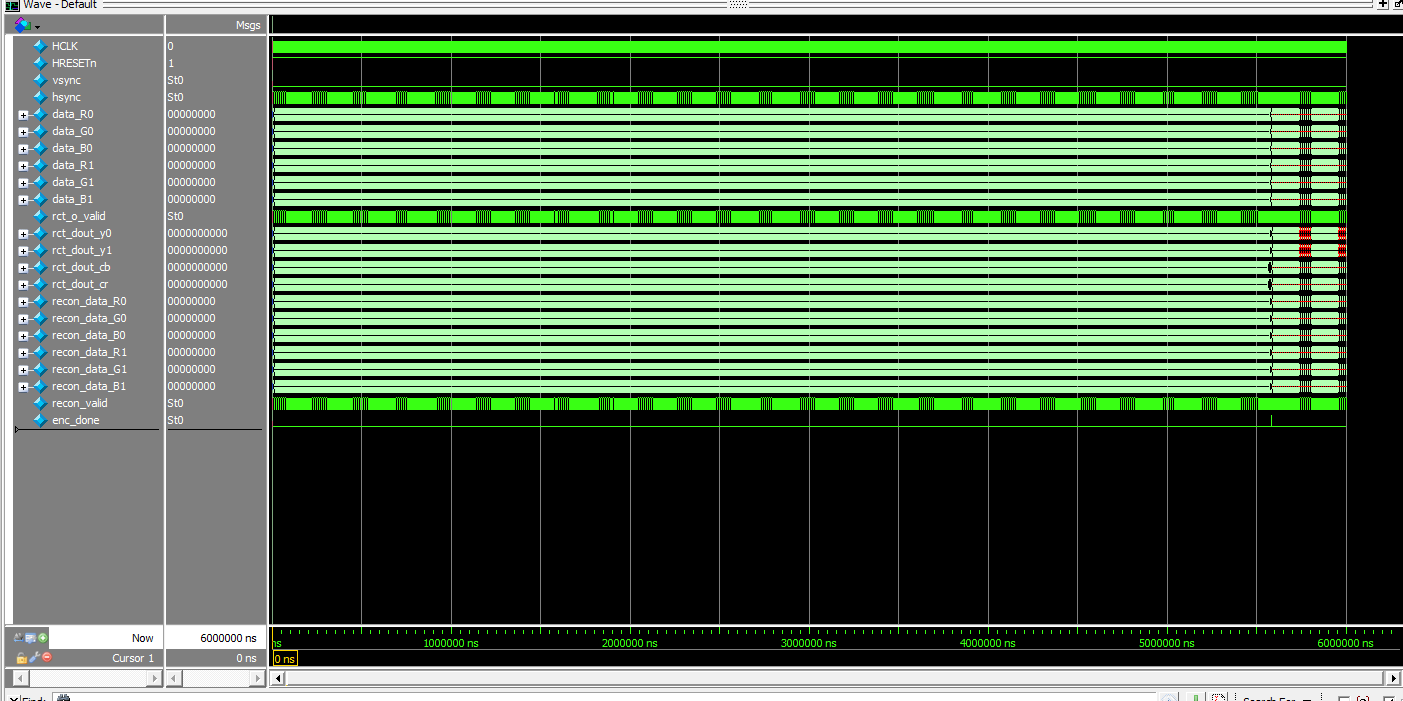
**Problem 8 (10p): Frame memory compression**

What you have to do:

* Implement RCT forward and backward modules by completing the missing codes.
* Submit your RTL files.

1. y0  <= {2'b0,tmp\_y0[9:2]};
2. y1  <= {2'b0,tmp\_y1[9:2]}/\*Insert your code here\*/;
3. cb0 <= tmp\_cb;
4. cr0 <= tmp\_cr/\*Insert your code here\*/;
5. out\_valid <= 1;
7. r0\_tmp <= cr0 + tmp0;
8. g0\_tmp <= tmp0;
9. b0\_tmp <= cb0 + tmp0;
10. r1\_tmp <= cr1 + tmp1/\*Insert your code here\*/;
11. g1\_tmp <= tmp1/\*Insert your code here\*/;
12. b1\_tmp <= cb1 + tmp1/\*Insert your code here\*/;
13. out\_valid <= 1;

* Capture the results including the waveform and the output images.



|  |  |
| --- | --- |
| C:\Users\BRL\Desktop\ModelSim\Homework2\RCT\out\kodim03.bmp | C:\Users\BRL\Desktop\ModelSim\Homework2\RCT\out\kodim03_ycbcr.bmp |

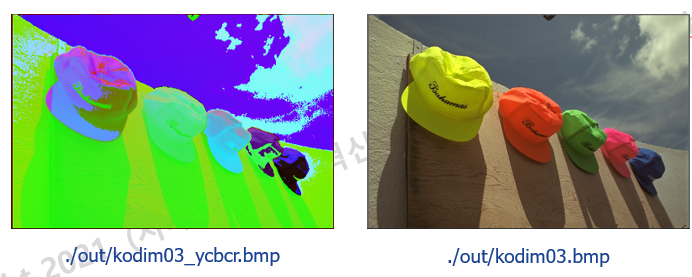


Fig. 8-1: Simulation results.