A Project Report

on

16 Bit RISC Processor

Created By -

Anirban Nath
anirbannath3@gmail.com

16 Bit RISC Processor

Design

ALU

```
`timescale 1ns / 1ps

//module declarations
module alu(

   //input declarartion
   input I_clk,
 input I_en,
 input [4:0]  I_aluop,
 input [15:0] I_dataA,
 input [15:0] I_dataB,
 input [7:0]  I_Imm,

 //output declarartion
 output [15:0] O_dataResult,
 output reg O_shldBranch
);

 //reg/wire declarartion
   reg [17:0] int_result;
 wire [3:0 ] opcode;
 wire op_lsb

//initial block
 initial
    int_result <=0;

 //assign values
   assign op_lsb = I_aluop[0];
   assign opcode =I_aluop[4:1];

 //parameter declaration
 localparam add=0,
          sub=1,
    or1 = 2,
    and1 =3,
    xor1 =4,
    not1 =5,
    load=8,
    cmp=9,   //compare
    shl=10,  //shift by left
    shr=11,  //shift by right
    jmpa=12, //jump address
    jmpr=13; //jump resgiter
```

```
//ALU operations
   always @(negedge I_clk) begin

     if(I_en) begin

  case(opcode)

   add : begin
      int_result <=(op_lsb? ($signed(I_dataA)+ $signed(I_dataB))
:(I_dataA)+(I_dataB));
    O_shldBranch <=0;
   end

   sub : begin
      int_result <=(op_lsb? ($signed(I_dataA)- $signed(I_dataB))
:(I_dataA)-(I_dataB));
    O_shldBranch <=0;
   end

   or1 : begin
      int_result <= I_dataA | I_dataB;
    O_shldBranch <=0;
   end

   and1 : begin
      int_result <= I_dataA & I_dataB;
    O_shldBranch <=0;
   end

   xor1 : begin
      int_result <= I_dataA ^ I_dataB;
    O_shldBranch <=0;
   end

   not1 : begin
      int_result <= ~I_dataA ;
    O_shldBranch <=0;
   end

   load : begin
      int_result <= (op_lsb ? {I_Imm, 8'h00} : {8'h00, I_Imm});
    O_shldBranch <=0;
   end
```

```
cmp : begin

    if(op_lsb) begin
      int_result[0] <=($signed(I_dataA)==$signed(I_dataA))? 1:0;
      int_result[1] <=($signed(I_dataA)==0)? 1:0;
      int_result[2] <=($signed(I_dataB)==0)? 1:0;
      int_result[3] <=($signed(I_dataA) > $signed(I_dataB))? 1:0;
      int_result[4] <=($signed(I_dataA) < $signed(I_dataB))? 1:0;
      O_shldBranch <=0;
    end

    else begin
      int_result[0] <=($signed(I_dataA)==$signed(I_dataA))? 1:0;
      int_result[1] <=($signed(I_dataA)==0)? 1:0;
      int_result[2] <=($signed(I_dataB)==0)? 1:0;
      int_result[3] <=($signed(I_dataA) > $signed(I_dataB))? 1:0;
      int_result[4] <=($signed(I_dataA) < $signed(I_dataB))? 1:0;
      O_shldBranch <=0;
    end
     end

          shl : begin
        int_result <= I_dataA << (I_dataB[3:0]);
    O_shldBranch <=0;

            end

   shl : begin
        int_result <= I_dataA >> (I_dataB[3:0]);
    O_shldBranch <=0;

            end

   jmpa :begin
      int_result <= (op_lsb ? I_dataA : I_Imm );
              O_shldBranch <=1;
            end

            /*jmpr :begin
     int_result <= I_dataA;
              O_shldBranch <= I_dataB [op_lsb,I_Imm[1:0]];


            end */

         endcase

  end

   end

endmodule
```

Instruction Decoder

```
`timescale 1ns/1ps

//module instruction decoder
module Inst_Dec(

  //input declaration
  input I_clk,
  input I_en,
  input [15:0] I_inst,

  //output declarartion
  output reg [4:0] O_aluop,
  output reg [3:0] O_selA,
  output reg [3:0] O_selB,
  output reg [3:0] O_selD,
  output reg O_regwe,
  output reg O_Imm

 );

  //initial Block
  initial begin
      O_aluop <=0;
 O_selA  <=0;
 O_selB  <=0;
 O_selD  <=0;
 O_Imm   <=0;
 O_regwe <=0;
   end

 always @(negedge I_clk) begin
   if(I_en) begin
      O_aluop <= I_inst [15:11];  //OPCODE
    O_selA  <= I_inst [10:8];  //REG A
    O_selB  <= I_inst [7:5];   //REG B
    O_selD  <= I_inst [4:2];   //REG D
    O_Imm   <= I_inst [7:0];   // IMM DATA

      case (I_inst [15:12])

   4'b0111 : O_regwe<=0;
   4'b1100 : O_regwe<=0;
   4'b1101 : O_regwe<=0;
   default : O_regwe<=1;
    endcase
   end
 end

endmodule
```

Control Unit

```
`timescale 1ns/1ps

module cntrl_unit(

   //input declaration
   input I_clk,
 input I_reset,

   //output declaration
   output O_enfetch,
 output O_endec,
 output O_enrgrd,
 output O_enrgwr,
 output O_enalu,
 output O_enmem
);

   reg [5:0] state; //reg declaration

 initial begin
   state <=6'b000001;
 end

 always @(posedge I_clk) begin
   if(I_reset)
     state <=6'b000001;
  else begin
    case (state)
       6'b000001 : state<=6'b000010;
     6'b000010 : state<=6'b000100;
     6'b000100 : state<=6'b001000;
     6'b001000 : state<=6'b010000;
     6'b010000 : state<=6'b100000;
     endcase
  end
 end

 assign O_enfetch =state [0];
 assign O_endec   =state [1];
 assign O_enrgrd  =state [2];
 assign O_enalu   =state [3];
 assign O_enrgwr  =state [4];
 assign o_enmem   =state [5];

endmodule
```

RAM

```
`timescale 1ns/1ps

//module declarations
module fake_RAM(
  //input declarations
  input I_clk,
  input I_we,
  input [15:0] I_addr,
  input [15:0] I_data,

  //output
  output [15:0] O_data
);

  //memeory declarations
  reg [15:0] mem [8:0];
  reg [15:0] O_data;

  //initialize regsiters
  initial begin

   mem[0]=16'b1000000011111110;
   mem[1]=16'b1000100111101101;
   mem[2]=16'b0010001000100000;
   mem[3]=16'b1000000011111111;
   mem[4]=16'b1100000000111110;
   mem[5]=16'b1010000011100110;
   mem[6]=16'b1000001111111110;
   mem[7]=0;
   mem[8]=0;

   O_data =16'b0000000000000000;
 end

 //RAM operations
 always @(negedge I_clk) begin

  if(I_we) begin
    mem[I_addr [15:0]] <=I_data;
  end

      O_data<= mem[I_addr [15:0]];
 end

endmodule
```

Program Counter

```
`timescale 1ns/1ps

//module defination
module pc_unit(

   //input
 input I_clk,
 input [1:0] I_opcode,
 input [15:0] I_pc,

 //output
 output reg [15:0] O_pc
);

   //initial block
 initial begin
   O_pc<=0;
 end

 //Program counter state
 always @(negedge I_clk) begin

  case (I_opcode)
     2'b00 : O_pc <=O_pc;
   2'b01 : O_pc <=O_pc+1;
   2'b10 : O_pc <=I_pc;
   2'b11 : O_pc <=0;
  endcase

   end

endmodule
```

Register Handler

```
`timescale 1ns/1ps

// Register File Module
module reg_file(
  input I_clk,
  input I_en,
  input I_we,
  input [2:0] I_selA,
  input [2:0] I_selB,
  input [2:0] I_selD,
  input [15:0] I_dataD,
  output reg [15:0] O_dataA,
  output reg [15:0] O_dataB
);

  reg [15:0] regs [7:0];
  integer count;

  initial begin
    O_dataA = 0;
    O_dataB = 0;
     for (count = 0; count < 8; count = count + 1) begin
      regs[count] = 0;
    end
  end

  always @(negedge I_clk) begin
    if (I_en) begin
      if (I_we)
        regs[I_selD] <= I_dataD;
      O_dataA <= regs[I_selA];
      O_dataB <= regs[I_selB];
    end
  end

endmodule
```

Testbench

Decoder UUT

```
`timescale 1ns/1ps

// Instruction Decoder Unit Testbench
module inst_dec_unittest();
  reg I_clk;
  reg I_en;
  reg [15:0] I_inst;

  wire [4:0] O_aluop;
  wire [2:0] O_selA;
  wire [2:0] O_selB;
  wire [2:0] O_selD;
  wire O_regwe;
  wire [15:0] O_Imm;

  Inst_Dec uut (.I_clk(I_clk), .I_en(I_en), .I_inst(I_inst), .O_aluop(O_aluop),
.O_selA(O_selA), .O_selB(O_selB), .O_selD(O_selD), .O_Imm(O_Imm),
.O_regwe(O_regwe));

  initial begin
    I_clk = 0;
    I_en = 0;
    I_inst = 0;

    #10; I_inst = 16'b0001011100000100;
    #10; I_en = 1;
  end

  always #5 I_clk = ~I_clk;

endmodule
```

Register UUT

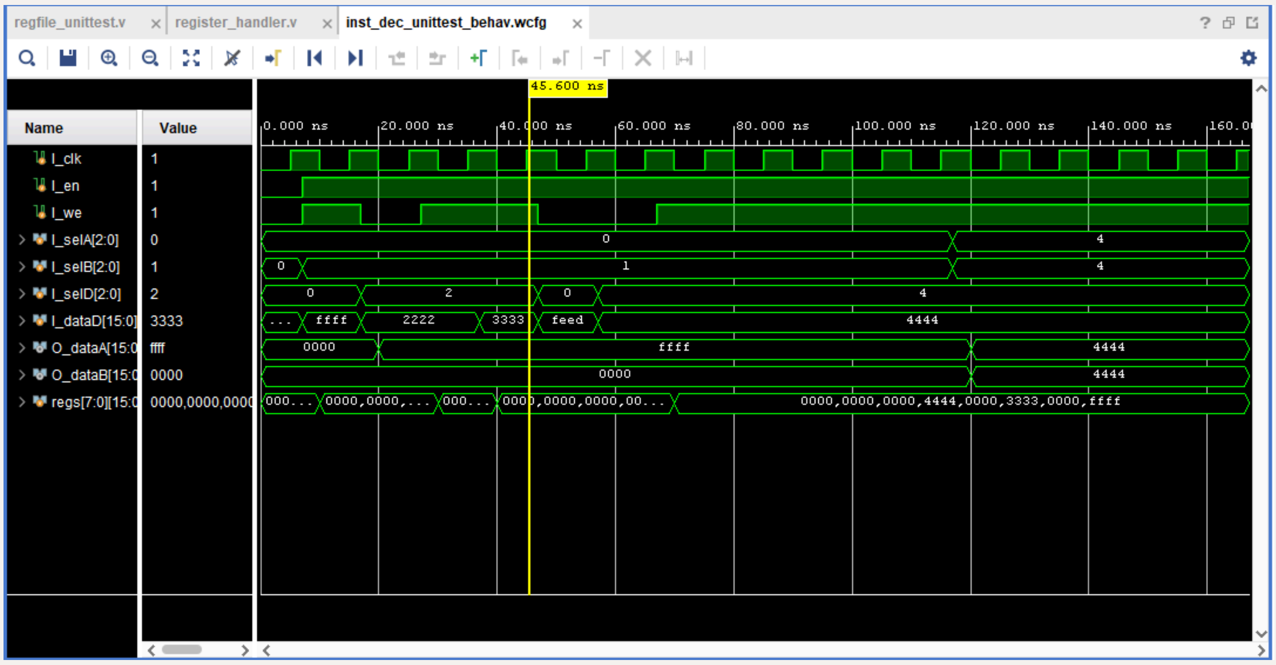
```
`timescale 1ns/1ps

module regfile_unittest();
  reg I_clk;
  reg I_en;
  reg I_we;
  reg [2:0] I_selA;
  reg [2:0] I_selB;
  reg [2:0] I_selD;
  reg [15:0] I_dataD;

  wire [15:0] O_dataA;
  wire [15:0] O_dataB;

  reg_file uut (.I_clk(I_clk), .I_en(I_en), .I_we(I_we), .I_selA(I_selA), .I_selB(I_selB), .I_selD(I_selD),
.I_dataD(I_dataD), .O_dataA(O_dataA), .O_dataB(O_dataB));

  initial begin
    I_clk = 0;
    I_dataD = 0;
    I_en = 0;
    I_selA = 0;
    I_selB = 0;
    I_selD = 0;
    I_we = 0;

    #7; I_en = 1; I_selA = 3'b000; I_selB = 3'b001; I_selD = 3'b000; I_dataD = 16'hFFFF; I_we = 1;
    #10; I_selD = 3'b010; I_we = 0; I_dataD = 16'h2222;
    #10; I_we = 1;
    #10; I_dataD = 16'h3333;
    #10; I_selD = 3'b000; I_we = 0; I_dataD = 16'hFEED;
    #10; I_selD = 3'b100; I_dataD = 16'h4444;
    #10; I_we = 1;
    #50; I_selA = 3'b100; I_selB = 3'b100;

    #50 $stop;
  end

  always #5 I_clk = ~I_clk;


endmodule
```

Main Test UUT

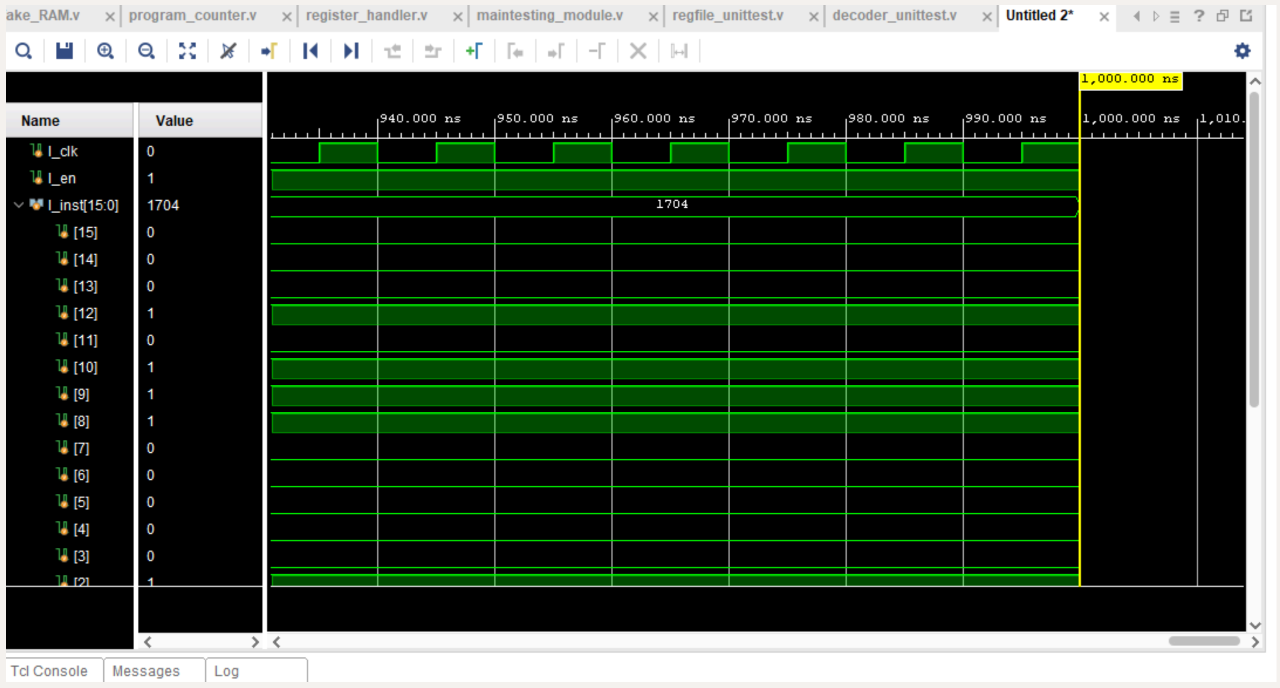
```
`timescale 1ns/1ps

// Main Testbench Module
module main_test();

  // Variable Declarations
  reg clk;
  reg reset;
  reg we = 0;
  reg [15:0] dataI = 0;
  reg en;
  reg [15:0] inst;
  wire [4:0] opcode;
  reg [15:0] addr;
  reg en;

  wire [2:0] selA;
  wire [2:0] selB;
  wire [2:0] selD;
  wire [15:0] dataA;
  wire [15:0] dataB;
  wire [15:0] dataD;
  wire [4:0] aluop;
  wire [7:0] Imm;
  wire [15:0] dataO;
  wire [15:0] pcO;
  wire shldBranch;
  wire enfetch;
  wire endec;
  wire enalu;
  wire enmem;
  wire enrgrd;
  wire enrgwr;
  wire regwe;
  wire [15:0] dataResult;
  wire O_shldBranch;
  wire Opc;

  // Module Instantiations
  reg_file main_reg (.I_clk(clk), .I_en(en), .I_we(we), .I_selA(selA), .I_selB(selB),
.I_selD(selD), .I_dataD(dataD), .O_dataA(dataA), .O_dataB(dataB));

  Inst_Dec main_inst (.I_clk(clk), .I_en(en), .I_inst(inst), .O_aluop(aluop), .O_selA(selA),
.O_selB(selB), .O_selD(selD), .O_Imm(Imm), .O_regwe(regwe));

  alu main_alu (.I_clk(clk), .I_en(en), .I_aluop(aluop), .I_dataB(dataB), .I_Imm(Imm),
.O_dataResult(dataResult), .O_shldBranch(shldBranch));
```

```
  cntrl_unit main_ctrl (.I_clk(clk), .I_reset(reset), .O_enfetch(enfetch),
.O_endec(endec), .O_enrgrd(enrgrd), .O_enalu(enalu), .O_enrgwr(enrgwr),
.O_enmem(enmem));

   pc_unit pc_main (.I_clk(clk), .I_opcode(opcode),.I_pc(pcO),.O_pc(Opc));

   fake_RAM main_ram (.I_clk(clk), .I_we(we), .I_addr(addr), .I_data(dataI),
.I_data(dataO));

   // Opcode Assignment
   assign opcode = (reset) ? 2'b00 : ((shldBranch) ? 2'b10 : ((we) ? 2'b01 : 2'b00));

   // Initial and Clock Generation
   initial begin
      clk = 0;
      reset = 1;
      #20;
      reset = 0;
      #25 $finish;
   end

   always #5 clk = ~clk;

endmodule
```

Waveforms

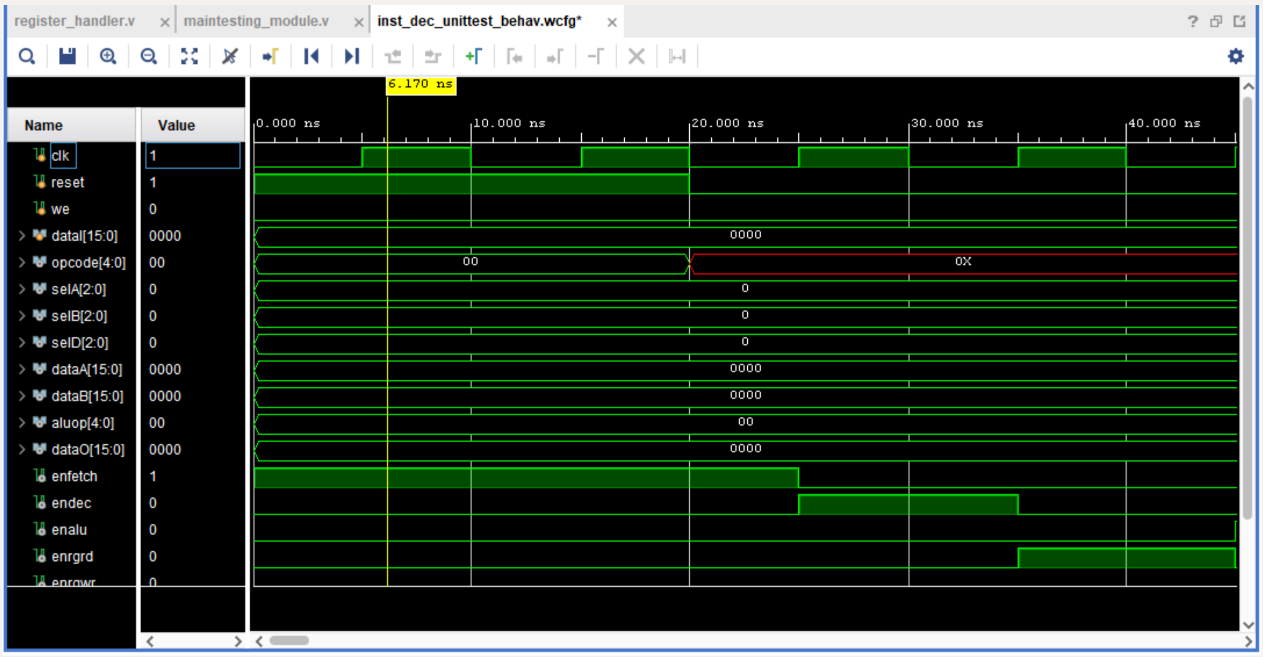


regfile_unittest.v
register_handler.v
inst_dec_unittest_behav.wcfg
45.600 ns
Name
Value
I_clk 1
I_en 1
I_we 1
I_selA[2:0] 0
I_selB[2:0] 1
I_selD[2:0] 2
I_dataD[15:0] 3333
O_dataA[15:0] ffff
O_dataB[15:0] 0000
regs[7:0][15:0] 0000,0000,0000
0.000 ns
20.000 ns
40.000 ns
60.000 ns
80.000 ns
100.000 ns
120.000 ns
140.000 ns
ffff
2222
3333
feed
4444
0000,0000,0000,4444,0000,3333,0000,ffff

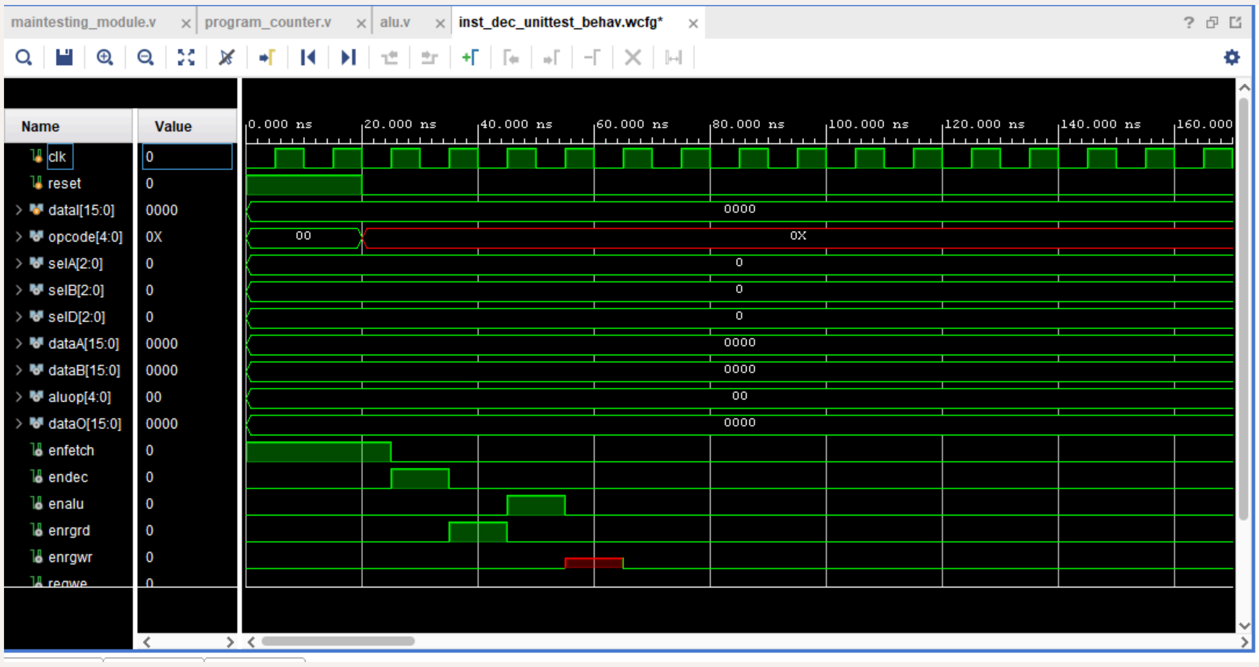


program_counter.v
register_handler.v
maintesting_module.v
regfile_unittest.v
decoder_unittest.v
Untitled 2*
1,000.000 ns
Name
Value
I_clk 0
I_en 1
I_inst[15:0] 1704
[15] 0
[14] 0
[13] 0
[12] 1
[11] 0
[10] 1
[9] 1
[8] 1
[7] 0
[6] 0
[5] 0
[4] 0
[3] 0
940.000 ns
950.000 ns
960.000 ns
970.000 ns
980.000 ns
990.000 ns
1,000.000 ns
1704
Tcl Console
Messages
Log

Waveforms



Name	Value
clk	1
reset	1
we	0
dataI[15:0]	0000
opcode[4:0]	00
selA[2:0]	0
selB[2:0]	0
selD[2:0]	0
dataA[15:0]	0000
dataB[15:0]	0000
aluop[4:0]	00
dataO[15:0]	0000
enfetch	1
endec	0
enalu	0
enrgrd	0
enrgwr	0



Name	Value
clk	0
reset	0
dataI[15:0]	0000
opcode[4:0]	0X
selA[2:0]	0
selB[2:0]	0
selD[2:0]	0
dataA[15:0]	0000
dataB[15:0]	0000
aluop[4:0]	00
dataO[15:0]	0000
enfetch	0
endec	0
enalu	0
enrgrd	0
enrgwr	0
regwe	0

Thank You