

CNG 331 – CAD2 Assignment

Fatma Erem Aksoy – 2315075

Muhammad Khizr Shahid – 2413235

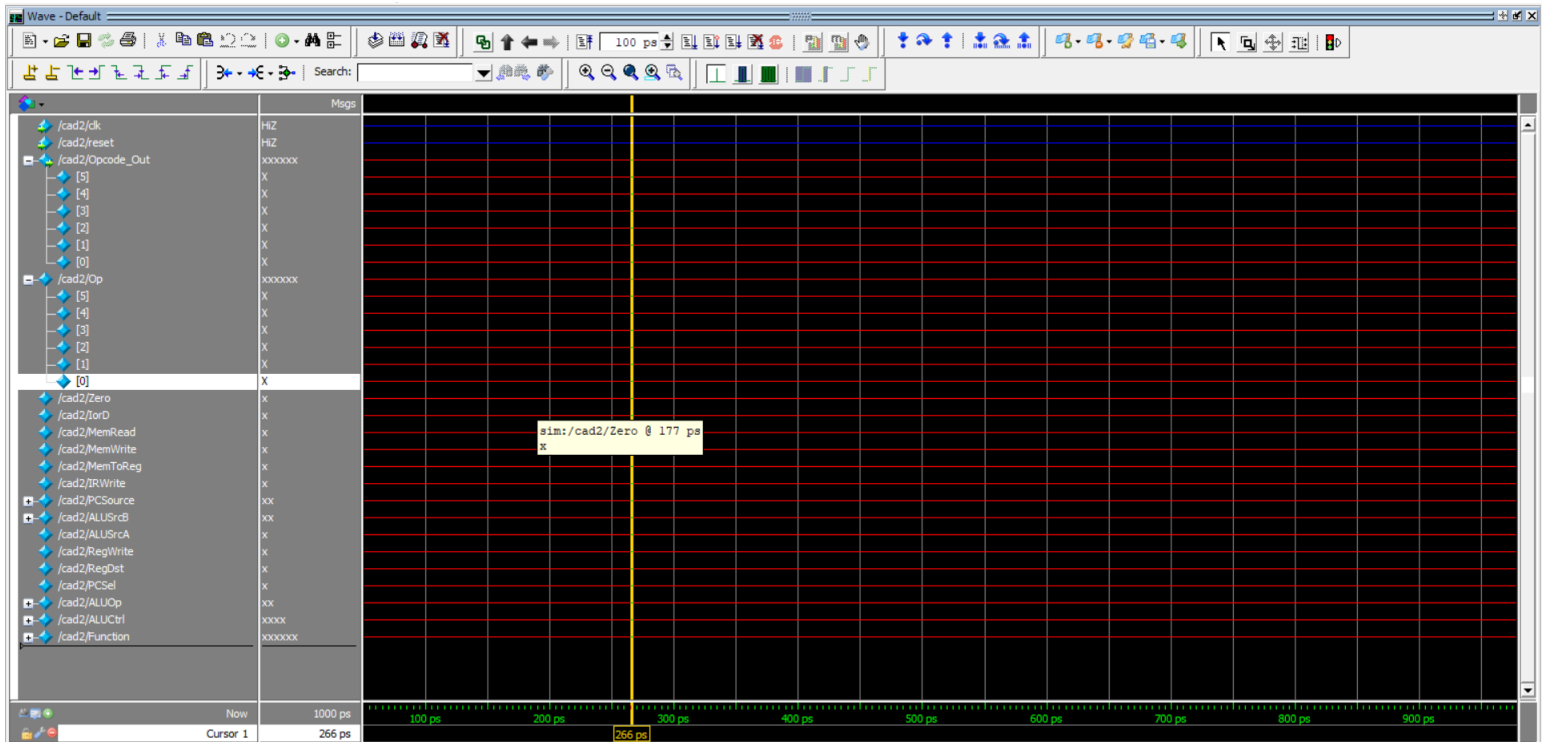
INTRODUCTION:

The goal of this CAD assignment is to design multicycle and pipeline MIPS Processors with the knowledge gained from CAD1. We need to calculate some values such as maximum clock rate (frequency), average CPI, execution time, energy consumption and thermal power dissipation to compare those two designs and decide which one is better when a specific requirement is considered. To achieve that, we also need to simulate some code and compare them based on the simulation results.

METHOD:

This CAD assignment was really challenging for us as it's a practical assignment and even though we are familiar and confident with the theoretical concepts, applying them to practice was an arduous process for both of us. We did not really split the task between each other to make sure that we both worked together in each step to learn new things from each other's knowledge. We worked together during the whole process, but still it was hard for us to complete both parts multicycle and pipeline design. We only managed to do multicycle design. Our code works properly on Quartus, but it does not work on Modelsim for some reason that we could not detect. It gives red lines, and we could not find the issue there to fix it to get the required waveforms. We did loads of debugging, and it was fun to analyze the code step by step with the hope that we could fix the waveforms, but it did not work, so it was very tiring and time-consuming. As Modelsim did not work as desired, we mostly worked on Quartus to get values using Timequest Analysis and Powerplay Analysis tools. We checked the summary after simulating the analysis, and we were able to get logical results from there.

FUNCTIONAL VERIFICATION:



Project Navigator

- Files
 - datapath.v
 - control.v
 - alucontrol.v
 - mips_mc.v

Tasks

Flow: Compilation

Task

- Compile Design
- Analysis & Synthesis
 - Edit Settings
 - View Report
 - Analysis & Elaboration
 - Partition Merge

Status

Module	% Progress
PowerPlay Power Analyzer	100%

Table of Contents

- Flow Summary
- Flow Settings
- Flow Non-Default Global Settings
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
- Fitter
- Assembler
- TimeQuest Timing Analyzer
- EDA Netlist Writer
- PowerPlay Power Analyzer
 - Summary
 - Settings
 - Indeterminate Toggle Rates
 - Operating Conditions Used
 - Thermal Power Dissipation by
 - Thermal Power Dissipation by
 - Thermal Power Dissipation by
 - Core Dynamic Thermal Power
 - Current Drawn from Voltage S
 - Confidence Metric Details
 - Signal Activities
 - Messages
 - Flow Messages
 - Flow Suppressed Messages

PowerPlay Power Analyzer Summary

PowerPlay Power Analyzer Status: Successful - Sat Jan 15 19:00:17 2022

Quartus II 64-Bit Version: 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition

Revision Name: cad2

Top-level Entity Name: mips_mc

Family: Cyclone II

Device: EP2C70F896C6

Power Models: Final

Total Thermal Power Dissipation: 194.15 mW

Core Dynamic Thermal Power Dissipation: 0.00 mW

Core Static Thermal Power Dissipation: 154.95 mW

I/O Thermal Power Dissipation: 39.20 mW

Power Estimation Confidence: Low: user provided insufficient toggle rate data

Messages

Type	ID	Message
Information	215049	Average toggle rate for this design is 0.000 millions of transitions / sec
Information	215031	Total thermal power estimate for the design is 194.15 mW
Information	>	Quartus II 64-Bit PowerPlay Power Analyzer was successful. 0 errors, 14 warnings

Operating Conditions Used		
	Setting	Value
1	Device power characteristics	Typical
2		
3	▼ Voltages	
1	VCCINT	1.20 V
2	3.3-V LVTTTL I/O Standard	3.3 V
4		
5	▼ Auto computed junction temperature	25.8 degrees Celsius
1	Ambient temperature	25.0 degrees Celsius
2	Junction-to-Case thermal resistance	2.10 degrees Celsius/Watt
3	Case-to-Heat Sink thermal resistance	0.10 degrees Celsius/Watt
4	Heat Sink-to-Ambient thermal resistance	2.10 degrees Celsius/Watt
6		
7	Board model used	None

RESULTS:

Parameter	Multi Cycle MIPS
F_{max} (TimeQuest Timing Analyzer)	92.8 MHz
CPI_{avg}	
Execution Time (50MHz)	(CPI * 10) / 50MHz
Core Total Thermal Power Dissipation (PowerPlay Power Analyzer)	194.15 mW
Energy Consumption (50MHz)	194.15 mW * Exec.Time

CPI (avg) = # of clock cycles/# of instructions

Execution Time = (CPI * # of instructions) / Clock rate = (CPI * 10) / 50MHz

Energy Consumption = Power * Execution Time = 194.15 mW * Exec.Time

CONCLUSION:

Basically we were able to complete almost all of multi cycle design, but unfortunately we were not able to do the pipeline design due to the complexity and difficulty level of that part. We were able to compile the multi cycle design successfully, however, we weren't getting the desired simulation outputs. We were unable to calculate the CPI for the multi cycle design so we were unable to calculate the rest of the parts such as Execution Time and Energy consumption but we had an idea on how to calculate them so we have shown the formulas and available calculations above in the table. As we were unable to do the pipeline design, we couldn't compare are results from multi cycle design to the pipeline design.

APPENDICIES:

datapath.sv:

```
1 module datapath(input logic clk, reset, IorD, MemRead, MemWrite, MemToReg, input logic IRWrite, ALUSrcA, RegWrite, RegDst, PCSel,
2   input logic [1:0] PCSource, input logic [1:0] ALUSrcB, input logic [3:0] ALUCtrl, output logic Zero, output logic [5:0] Op,
3   output logic [5:0] Function);
4
5   parameter PCSTART = 128; //starting address of instruction memory
6   logic [7:0] PC;
7   logic [31:0] ALUOut;
8   logic [31:0] ALUResult;
9   logic [31:0] OpA;
10  logic [31:0] OpB;
11  logic [31:0] A;
12  logic [31:0] B;
13  logic [31:0] Instruction;
14  logic [31:0] MDR;
15  logic [31:0] dA;
16  logic [31:0] dB;
17  logic [7:0] address;
18  logic [31:0] MemData;
19  logic [31:0] RF_WriteData;
20  logic [4:0] RF_rd;
21  reg[31:0] mem[255:0];
22  reg[31:0] registers[31:0];
23
24
25  assign Op = Instruction[31:26];
26  assign Function = Instruction[5:0];
27
28  assign address = (IorD) ? ALUOut : PC;
29  initial $readmemh("unified_memory.dat", mem);
30
31
32  always @(posedge clk) begin
33    if(MemWrite)
34      mem[address] <= B;
35  end
36
37  assign MemData = (MemRead)?mem[address] : 32'bx;
38
39  always @(posedge clk)begin
40    if(reset)
41      PC <= PCSTART;
42    else
43      if(PCSel)begin
44        case (PCSource)
45          1'b0: PC <= ALUResult;
46          1'b1: PC <= ALUOut;
47        endcase
48      end
49  end
50
51
52  always @(posedge clk) begin
53    if (IRWrite)
54      Instruction <= MemData;
55  end
56
57
58  always @(posedge clk) begin
59    MDR <= MemData;
60  end
61
62  //Register File
63  //$r0 is always 0
64  assign dA = (Instruction[25:21]!=0) ? registers[Instruction[25:21]] : 0;
65  assign dB = (Instruction[20:16]!=0) ? registers[Instruction[20:16]] : 0;
66  assign RF_WriteData = (MemToReg) ? MDR : ALUOut;
67  assign RF_rd = (RegDst) ? Instruction[15:11] : Instruction[20:16];
```

```

70  always @(posedge clk) begin
71      if (RegWrite) begin
72          registers[RF_rd] <= RF_WriteData;
73      end
74  end
75
76
77  //A and B registers
78  always @(posedge clk) begin
79      if (reset)
80          A <= 0;
81      else
82          A<=dA;
83      end
84
85  always @(posedge clk) begin
86      if (reset)
87          B <= 0;
88      else
89          B<=dB;
90      end
91
92
93  //ALU
94  assign OpA = (ALUSrcA) ? A : PC;
95
96  always_comb
97  begin
98      case(ALUSrcB)
99          2'b00: OpB = B;
100         2'b01: OpB = 4;
101         2'b10: OpB = {{(16){Instruction[15]}}, Instruction[15:0]};
102         2'b11: OpB = {{(14){Instruction[15]}}, Instruction[15:0],2'b00};
103     endcase
104 end
105
106 assign Zero = (ALUResult==0); //Zero == 1 when ALUResult is 0 (for branch)
107
108 always_comb
109 begin
110     case(ALUCtrl)
111         4'b0000: ALUResult = OpA & OpB; //and
112         4'b0001: ALUResult = OpA | OpB; //or
113         4'b0010: ALUResult = OpA ^ OpB; //add
114         4'b0011: ALUResult = ~(OpA | OpB);
115         4'b0110: ALUResult = OpA + OpB; //branch
116         4'b1110: ALUResult = OpA - OpB;
117         4'b1111: ALUResult = OpA < OpB?1:0; //ALUResult is 1 when OpA<Opb, otherwise 0
118         default: ALUResult = OpA + OpB;
119     endcase
120 end
121
122
123 always@(posedge clk) begin
124     ALUOut <= ALUResult;
125 end
126
127 endmodule

```

alucontrol.sv

```
1 module alucontrol(ALUOp,Function,ALUCtrl);
2
3     input [1:0] ALUOp;
4     input [5:0] Function;
5
6     output reg [3:0] ALUCtrl; //for R-type instructions
7
8     always@(ALUOp or Function)begin
9         casex({ALUOp,Function})
10             8'b00_XXXXXX : ALUCtrl=4'b0010; //lw or sw
11             8'b01_XXXXXX : ALUCtrl=4'b0110; //beq
12             8'b1x_XX0000 : ALUCtrl=4'b0010; //add
13             8'b1x_XX0010 : ALUCtrl=4'b0110; //sub
14             8'b1x_XX0100 : ALUCtrl=4'b0000; //and
15             8'b1x_XX0101 : ALUCtrl=4'b0001; //or
16             8'b1x_XX1010 : ALUCtrl=4'b0111; //slt
17         endcase
18     end
19
20 endmodule
```

mips_mc.sv:

```
1 module mips_mc(input logic clk,
2                 input logic reset,
3                 output logic [5:0] Opcode_Out);
4
5     logic [5:0] Op;
6     logic Zero;
7     logic IorD;
8     logic MemRead;
9     logic MemWrite;
10    logic MemToReg;
11    logic IRWrite;
12    logic [1:0] PCSource;
13    logic [1:0] ALUSrcB;
14    logic ALUSrcA;
15    logic RegWrite;
16    logic RegDst;
17    logic PCSel;
18    logic [1:0] ALUOp;
19    logic [3:0] ALUCtrl;
20    logic [5:0] Function;
21    assign Opcode_Out = Op[5:0];
22
23    control u0(.clk(clk), .reset(reset), .Zero(Zero), .Op(Op), .IorD(IorD), .MemRead(MemRead),
24              .MemWrite(MemWrite), .MemToReg(MemToReg), .IRWrite(IRWrite), .ALUSrcA(ALUSrcA),
25              .RegWrite(RegWrite), .RegDst(RegDst), .PCSel(PCSel), .PCSource(PCSource),
26              .ALUSrcB(ALUSrcB), .ALUOp(ALUOp));
27
28    alucontrol u1(.ALUOp(ALUOp), .Function(Function), .ALUCtrl(ALUCtrl));
29
30    datapath u2(.clk(clk), .reset(reset), .IorD(IorD), .MemRead(MemRead), .MemWrite(MemWrite),
31              .MemToReg(MemToReg), .IRWrite(IRWrite), .ALUSrcA(ALUSrcA), .RegWrite(RegWrite),
32              .RegDst(RegDst), .PCSel(PCSel), .PCSource(PCSource), .ALUSrcB(ALUSrcB),
33              .ALUCtrl(ALUCtrl), .Zero(Zero), .Op(Op), .Function(Function));
34 endmodule
```

tb_mips_mc.sv:

```
1 module tb_mips_mc;
2     logic clk;
3     logic reset;
4     logic [5:0] Opcode_Out;
5
6     mips_mc utb(.clk(clk), .reset(reset), .Opcode_Out(Opcode_Out));
7
8     initial
9         forever #10000 clk=~clk;
10    initial begin
11        clk = 0; #20000;
12        reset = 1;
13        reset = 0;
14        #1000000 $finish;
15    end
16
17 endmodule
```

control.sv :

```
1 module control (input logic clk, reset, Zero, input logic [5:0] Op, output logic IorD,
2     MemRead, MemWrite, MemToReg, IRWrite, output logic ALUSrcA, RegWrite,
3     RegDst, PCSel, output logic [1:0] PCSource, output logic [1:0] ALUSrcB,
4     output logic [1:0] ALUOp);
5
6     reg PCWrite;
7     reg PCWriteCond;
8
9     assign PCSel = (PCWrite | (PCWriteCond & Zero));
10
11     parameter Fetch = 4'b0000; //Stage 0 - fetch
12     parameter Decode = 4'b0001; //Stage 1 - decode
13     parameter LoadORStore = 4'b0010; //Stage 2 - load or store
14     parameter MemAccessLoad = 4'b0011; //Stage 3 - L1
15     parameter WriteBack = 4'b0100; //Stage 4 - Write Back
16     parameter MemAccessStore = 4'b0101; //Stage 5 - Store
17     parameter Execution = 4'b0110; //Stage 6 - R-type
18     parameter RType = 4'b0111; //Stage 7
19     parameter Beq = 4'b1000; //Stage 8 - Branch
20     parameter J = 4'b1001; //Stage 9 - Jump
21
22     reg [3:0] state;
23     reg [3:0] nextstate;
24
25     always@(posedge clk)
26
27         if (reset)
28             state <= Fetch;
29         else
30             state <= nextstate;
31
32     always@(state or Op) begin
33         case (state)
34             Fetch:
```

```

35         nextstate = Decode;
36
37     Decode:
38     case(Op)
39         6'b100011: nextstate = LoadORStore;    //load(lw)
40         6'b101011: nextstate = LoadORStore;    //store(sw)
41         6'b000000: nextstate = Execution;    //r
42         6'b000100: nextstate = Beq;    //branch(beq)
43         6'b000010: nextstate = J;    //jump(j)
44         default: nextstate = Fetch;
45     endcase
46
47     LoadORStore: case(Op)
48         6'b100011: nextstate = MemAccessLoad; //lw
49         6'b101011: nextstate = MemAccessStore; //sw
50         default: nextstate = Fetch;
51     endcase
52
53     MemAccessLoad:    nextstate = WriteBack;
54     WriteBack:        nextstate = Fetch;
55     MemAccessStore:   nextstate = Fetch;
56     Execution:        nextstate = RType;
57     RType:            nextstate = Fetch;
58     Beq:              nextstate = Fetch;
59     J:                nextstate = Fetch;
60     default: nextstate = Fetch;
61 endcase
62 end
63
64 always@(state) begin
65     IorD=1'b0; MemRead=1'b0; MemWrite=1'b0; MemToReg=1'b0; IRWrite=1'b0;
66     PCSource=1'b0; ALUSrcB=2'b00; ALUSrcA=1'b0; RegWrite=1'b0; RegDst=1'b0;
67     PCWrite=1'b0; PCWriteCond=1'b0; ALUOp=2'b00;
68
69     case (state)
70     Fetch:
71     begin
72         MemRead = 1'b1;
73         IRWrite = 1'b1;
74         ALUSrcB = 2'b01;
75         PCWrite = 1'b1;
76     end
77
78     Decode:
79         ALUSrcB = 2'b11;
80
81     LoadORStore:
82     begin
83         ALUSrcA = 1'b1;
84         ALUSrcB = 2'b10;
85     end
86
87     MemAccessLoad:
88     begin
89         MemRead = 1'b1;
90         IorD = 1'b1;
91     end
92
93     WriteBack:
94     begin
95         RegWrite = 1'b1;
96         MemToReg = 1'b1;
97         RegDst = 1'b0;
98     end
99
100    MemAccessStore:

```



```

101     begin
102         MemWrite = 1'b1;
103         IorD      = 1'b1;
104     end
105
106     Execution:
107     begin
108         ALUSrcA = 1'b1;
109         ALUOp   = 2'b10;
110     end
111
112     RType:
113     begin
114         RegDst = 1'b1;
115         RegWrite = 1'b1;
116     end
117
118     Beq:
119     begin
120         ALUSrcA = 1'b1;
121         ALUOp   = 2'b01;
122         PCWriteCond = 1'b1;
123         PCSrc = 2'b01;
124     end
125
126     J:
127     begin
128         PCSrc = 2'b10;
129         PCWrite = 1'b1;
130     end
131     endcase
132 end
133
134 endmodule

```