**VLSI Physical Design**

**8 BIT Microprogrammed Processor**

**Project Report**

**Dr. Vandana Khanna**

**School of Engineering & Technology**

**Department**

**of**

**Electrical, Electronics and Communication Engineering**

Name: Vasu Singh and Riya Jain Roll No. 20ECU022 and 20ECU008 Branch: E E C E Section

**The NorthCap University**

**Gurugram, Haryana**

**INDEX Roll No:** ...20ECU022.....

|  |  |  |  |
| --- | --- | --- | --- |
| **Exp.**  **No.** | **Name of Experiment**  (In the sequence of performance) | **Page**  **No** | **Faculty’s Signature with Date** |
| 1 | Declaration | 3 |  |
| 2 | Acknowledgement | 4 |  |
| 3 | Introduction | 5 |  |
| 4 | Main Code | 5-12 |  |
| 5 | RC script and SDC file | 12-13 |  |
| 6 | Step of Physical Design | 14-33 |  |
| Logic Synthesis |  |
| Floor Planning and Power Planning |
| Pre-placement |
| Placement (Pre-CTS) |
| Clock Tree (including Post-CTS) |
| Routing |
| 7 | Results | 33-35 |  |

1. **DECLARATION**

I, **Riya(20ecu008)** and **Vasu(20ecu022)** hereby declare that the project and the report presented to you is the record of authentic work carried out by all of us. And we also confirm that it has not been submitted to any other university or institute for the award of any kind of certification/degree.

We take the full responsibility of not providing this project report or the project to anyone but to you. If found anywhere else, we take the responsibility for the same.

1. **ACKNOWLEDGEMENT**

I would like to express my gratitude and appreciation to all those who gave me the possibility to complete this report. Special thanks are due to my teacher **Dr. Vandana Khanna** whose help, stimulating suggestions and encouragement helped me in all times of research process and in writing this report.

I also sincerely thanks for the time spent proofreading and correcting my many mistakes.

I would also like to thank my university for providing us systems in the lab to complete our project and give us spare time whenever required by us. I would also like to thank our lab assistant and my classmates to support us throughout the project and help us whenever needed.

Any emission in this brief acknowledgement does not mean lack of gratitude.

1. **INTRODUCTION**

* A microprogrammed control unit is a complex programmable unit that outputs control signals to data path according to its "microprogram".
* A microprogrammed control unit can be regarded as a simple computer.  In this view, a processor has another simple processor inside it which is its control unit.

Controlling a data path is described by its microprogram.

1. **MAIN CODE**

**module** MCPU8\_1(**input** clk , **input** rst , **output** [4:0] PC\_OUT , **output** [4:0] MAR\_OUT , **output** [3:0] IR\_OUT1 , **output** [4:0] IR\_OUT2 , **output**[8:0] DATA\_OUT1 , **output**[4:0] ADDR\_OUT1 , **output** [4:0] COUNT\_OUT , **output** [8:0] ACCUMULATOR\_OUT , **output** [8:0] DATA\_OUTPUT , **output** [8:0] B\_REG , **output** [8:0] ALU\_OUT , **output** [8:0] OR\_out , **output** [16:0] CW, **output** EP , **output** CP , **output** LM , **output** CE ,**output** LI , **output** EI , **output** CS , **output** LOAD , **output** CLR , **output** INC , **output** LA , **output** EA , **output** LB , **output** SU , **output** AD , **output** EU , **output** LO);

**wire** [4:0] MAR\_OUT\_w , IR\_OUT\_2\_w , MAR\_IN\_w , MUX\_IN;

**wire** [3:0] IR\_OUT\_1\_w;

**wire** [16:0] CW\_w;

**wire** [8:0] DATA\_IN\_w;

**wire**[4:0] bus\_5;

**wire** [4:0] bus\_5\_1;

**wire** [8:0] bus\_9 , bus\_9\_1 , bus\_9\_2 , ACC\_IN\_w;

**wire** [8:0] ALU\_A\_w;

**wire** [8:0] ALU\_B\_w;

**wire** EP\_w , CP\_w , CS\_w , CE\_w , LOAD\_w , LI\_w , LM\_w , EI\_w , CLR\_w , INC\_w, LA\_w , EA\_w , LB\_w , SU\_w , AD\_w , EU\_w , LO\_w;

**assign** EP = EP\_w;

**assign** CP = CP\_w;

**assign** CW = CW\_w;

**assign** LOAD = LOAD\_w;

**assign** CS = CS\_w;

**assign** LI = LI\_w;

**assign** LM = LM\_w;

**assign** CE = CE\_w;

**assign** EI = EI\_w;

**assign** INC = INC\_w;

**assign** CLR = CLR\_w;

**assign** LA = LA\_w;

**assign** EA = EA\_w;

**assign** LB = LB\_w;

**assign** SU = SU\_w;

**assign** AD = AD\_w;

**assign** EU = EU\_w;

**assign** LO = LO\_w;

**assign** DATA\_OUT1 = bus\_9;

**assign** IR\_OUT1 = IR\_OUT\_1\_w;

**assign** IR\_OUT2 = bus\_5\_1;

**assign** ADDR\_OUT1 = MUX\_IN;

**assign** PC\_OUT = bus\_5;

**assign** MAR\_OUT = MAR\_OUT\_w;

**assign** ACCUMULATOR\_OUT = ALU\_A\_w;

**assign** OR\_out = DATA\_OUTPUT;

**assign** B\_REG = ALU\_B\_w;

**assign** ALU\_OUT = bus\_9\_1;

PC\_4 UUT1 (.clk(clk) , .rst(rst) , .EP(EP\_w) , .CP(CP\_w) , .PC\_OUT(bus\_5));

MAR UUT2(.clk(clk) , .LM(LM\_w) , .MAR\_IN(bus\_5 | bus\_5\_1) , .MAR\_OUT(MAR\_OUT\_w));

SRAM\_8 UUT3(.clk(clk) , .ADDR(MAR\_OUT\_w) , .CE(CE\_w) ,.DATA\_OUT(bus\_9));

IR\_8 UUT4(.clk(clk) , .rst(rst) , .LI(LI\_w) , .EI(EI\_w) ,.DATA\_IN(bus\_9) , .IR\_OUT\_1(IR\_OUT\_1\_w) , .IR\_OUT\_2(bus\_5\_1));

ADDR\_ROM UUT5(.CS(CS\_w) , .ADDR(IR\_OUT\_1\_w) , .ADDR\_OUT(MUX\_IN));

PRESET\_COUNT UUT6(.clk(clk) , .rst(rst) , .LOAD(LOAD\_w) , .CLR(CLR\_w) , .INC(INC\_w) , .COUNT\_IN(MUX\_IN) , .COUNT\_OUT(COUNT\_OUT));

CONTROL\_ROM UUT7(.ADDR\_IN(COUNT\_OUT) , .CW(CW\_w));

MICRO\_DECODER UUT8(.CW(CW\_w) , .EP(EP\_w) , .CP(CP\_w) , .LM(LM\_w) ,.CE(CE\_w), .LI(LI\_w) , .EI(EI\_w) , .CS(CS\_w) , .LOAD(LOAD\_w) , .CLR(CLR\_w) , .INC(INC\_w) , .LA(LA\_w) , .EA(EA\_w) , .LB(LB\_w) , .SU(SU\_w) , .AD(AD\_w) , .EU(EU\_w) , .LO(LO\_w));

ACC\_8 UUT9(.clk(clk) , .ACC\_IN(bus\_9 | bus\_9\_1) , .LA(LA\_w) , .EA(EA\_w) , .ACC\_OUT\_ALU(ALU\_A\_w) , .ACC\_OUT\_BUS(bus\_9\_2));

B\_REG\_8 UUT10(.clk(clk) , .LB(LB\_w) , .B\_IN(bus\_9) , .B\_OUT(ALU\_B\_w));

OUT\_REG\_8 UUT11(.clk(clk) , .OUT\_IN(bus\_9\_2) , .LO(LO\_w) , .OUT\_P(DATA\_OUTPUT));

ALU\_8 UUT12(.ALU\_A(ALU\_A\_w) ,.ALU\_B(ALU\_B\_w) ,.SU(SU\_w) , .AD(AD\_w) , .EU(EU\_w) ,.ALU\_OUT(bus\_9\_1));

**endmodule**

**module** PC\_4(**input** clk , **input** rst , **input** EP , **input** CP , **output** [4:0] PC\_OUT);

**reg** [4:0] PC\_OUT\_r;

**assign** PC\_OUT = EP ? PC\_OUT\_r : 5'h00;

**always**@(**posedge** clk **or** **posedge** rst)

**begin**

**if**(rst)

PC\_OUT\_r <= 5'b00000;

**else** **if**(CP)

PC\_OUT\_r <= PC\_OUT\_r + 1'b1;

**end**

**endmodule**

**module** MAR(**input** clk , **input** LM , **input** [4:0] MAR\_IN , **output** [4:0] MAR\_OUT);

**reg**[4:0]MAR\_r;

**assign** MAR\_OUT = MAR\_r;

**always**@(**posedge** clk)

**begin**

**if**(~LM)

MAR\_r <= MAR\_IN;

**end**

**endmodule**

**module** SRAM\_8(**input** clk , **input** [4:0] ADDR , **input** CE , **output**[8:0] DATA\_OUT);

**reg** [8:0] SRAM [15:0] ;*// A SRAM with 16 locations with each location capable of holding 8-bit of DATA*

**assign** DATA\_OUT = (~CE) ? SRAM[ADDR] : 9'h000;

**always**@(**posedge** clk)

**begin**

SRAM[0] <= 9'b000001001; *//LDA 09 instruction - Load the contents of memory location 09 into the accumulator*

SRAM[1] <= 9'b000101010; *// ADD 0A instruction - Add the contents of Accumulator with the contents of memory location 0A and store the result in Accumulator*

SRAM[2] <= 9'b001001011; *// SUB 0B - Add the contents of Accumulator that is obtained from previous instruction with the contents of memory location 0B and store the result in Accumulator*

SRAM[3] <= 9'b0011xxxxx; *// OUT instruction - To output the content of accumulator that is obtained from previous addition operation*

SRAM[4] <= 9'b111111111; *// HLT instruction - To stop the execution of instructions*

SRAM[5] <= 9'b111111111; *// Unused memory locations are filled with FF*

SRAM[6] <= 9'b111111111; *// Unused memory locations are filled with FF*

SRAM[7] <= 9'b111111111; *//Unused memory locations are filled with FF*

SRAM[8] <= 9'b111111111; *// Unused memory locations are filled with FF*

SRAM[9] <= 9'b000000001; *// 01H is the 8-bit value stored in the location 09*

SRAM[10]<= 9'b000000010; *// 02H is the 8-bit value stored in the location 0A*

SRAM[11]<= 9'b000000001; *// 01H is the 8-bit value stored in the location 0B*

SRAM[12]<= 9'b111111111; *// Unused memory locations are filled with FF*

SRAM[13]<= 9'b111111111; *// Unused memory locations are filled with FF*

SRAM[14]<= 9'b111111111; *// Unused memory locations are filled with FF*

SRAM[15]<= 9'b111111111; *// Unused memory locations are filled with FF*

**end**

**endmodule**

**module** IR\_8 (**input** clk , **input** rst , **input** LI , **input** EI, **input** [8:0] DATA\_IN , **output**[3:0] IR\_OUT\_1 , **output** [4:0] IR\_OUT\_2);

**reg** [8:0] IR\_r;

**assign** IR\_OUT\_1 = IR\_r[8:5];

**assign** IR\_OUT\_2 = EI ? IR\_r[4:0] : 5'h00;

**always**@(**posedge** clk **or** **posedge** rst)

**begin**

**if**(rst)

IR\_r <= 9'h000;

**else** **if**(~LI)

IR\_r <= DATA\_IN;

**end**

**endmodule**

**module** ADDR\_ROM(**input** CS , **input**[3:0] ADDR , **output**[4:0] ADDR\_OUT);

**reg** [4:0] AR [15:0];

**assign** ADDR\_OUT = CS ? AR[ADDR] : 5'h00;

**always** @(ADDR)

**begin**

AR[0] <= 5'b00100; *// LDA Routine Address*

AR[1] <= 5'b00111; *// ADD Routine Address*

AR[2] <= 5'b01100; *// SUB Routine Address*

AR[3] <= 5'b10001; *// OUT Routine Address*

AR[4] <= 5'bxxxxx;

AR[5] <= 5'bxxxxx;

AR[6] <= 5'bxxxxx;

AR[7] <= 5'bxxxxx;

AR[8] <= 5'bxxxxx;

AR[9] <= 5'bxxxxx;

AR[10] <= 5'bxxxxx;

AR[11] <= 5'bxxxxx;

AR[12] <= 5'bxxxxx;

AR[13] <= 5'bxxxxx;

AR[14] <= 5'bxxxxx;

AR[15] <= 5'bxxxxx;

**end**

**endmodule**

**module** PRESET\_COUNT(**input** clk , **input** rst , **input** INC , **input** CLR , **input** LOAD , **input** [4:0] COUNT\_IN , **output** [4:0] COUNT\_OUT); *// Micro-Routine Program Counter*

**reg** [4:0] COUNT\_OUT\_r;

**assign** COUNT\_OUT = COUNT\_OUT\_r;

**always**@(**posedge** clk **or** **posedge** rst)

**begin**

**if**(rst)

COUNT\_OUT\_r <= 5'b0000;

**else** **if**(LOAD)

COUNT\_OUT\_r <= COUNT\_IN;

**else** **if**(INC) *// Always associate a signal that must occur along with CLOCK EDGE , for an operation to be done. This is the correct way.*

COUNT\_OUT\_r <= COUNT\_OUT\_r + 1'b1;

**else** **if**(CLR)

COUNT\_OUT\_r <= COUNT\_IN;

**end**

**endmodule**

**module** CONTROL\_ROM(**input** [4:0] ADDR\_IN , **output** [16:0] CW);

**reg** [16:0] CR [19:0];

**assign** CW = CR[ADDR\_IN];

**always**@(ADDR\_IN)

*// EP CP LM\* CE\* LI\* EI CS LOAD CLR INC LA\* EA LB\* SU AD EU LO\**

*// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FETCH ROUTINE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*// 1 0 0 1 1 0 0 0 0 1 1 0 1 0 0 0 1*

*// 0 1 1 1 1 0 0 0 0 1 1 0 1 0 0 0 1*

*// 0 0 1 0 0 0 0 0 0 1 1 0 1 0 0 0 1*

*// 0 0 1 1 1 0 1 1 0 0 1 0 1 0 0 0 1*

*// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LDA ROUTINE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*// 0 0 0 1 1 1 0 0 0 1 1 0 1 0 0 0 1*

*// 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0 1*

*// 0 0 1 1 1 0 0 0 1 0 1 0 1 0 0 0 1*

*// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ADD ROUTINE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*// 0 0 0 1 1 1 0 0 0 1 1 0 1 0 0 0 1*

*// 0 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 1*

*// 0 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 1*

*// 0 0 1 1 1 0 0 0 0 1 0 0 1 0 0 1 1*

*// 0 0 1 1 1 0 0 0 1 0 1 0 1 0 0 0 1*

*// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SUB ROUTINE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*// 0 0 0 1 1 1 0 0 0 1 1 0 1 0 0 0 1*

*// 0 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 1*

*// 0 0 1 1 1 0 0 0 0 1 1 0 1 1 0 0 1*

*// 0 0 1 1 1 0 0 0 0 1 0 0 1 0 0 1 1*

*// 0 0 1 1 1 0 0 0 1 0 1 0 1 0 0 1 1*

*// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* OUT ROUTINE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*// 0 0 0 1 1 1 0 0 0 1 1 0 1 0 0 1 1*

*// 0 0 1 1 1 0 0 0 0 1 1 1 1 0 0 1 0*

*// 0 0 1 1 1 0 0 0 1 0 1 0 1 0 0 1 1*

**begin**

*// Fetch Micro-routine*

CR[0] <= 17'b10011000011010001; *// EP and LM signals activated and Preset Counter Incremented*

CR[1] <= 17'b01111000011010001; *// CP signal activated and Preset Counter Incremented*

CR[2] <= 17'b00100000011010001; *// CE and LI signals activated and Preset Counter Incremented*

CR[3] <= 17'b00111011001010001; *// CS and Preset LOAD signal activated.*

*// LDA Micro-routine*

CR[4] <= 17'b00011100011010001 ; *// LM , EI signals activated and Preset Counter Incremented*

CR[5] <= 17'b00101000010010001; *// CE and LA signals activated and Preset Counter Incremented*

CR[6] <= 17'b00111000101000001; *// Preset CLR signal activated*

*// ADD Micro-routine*

CR[7] <= 17'b00011100011010001; *// LM , EI signals activated and Preset Counter Incremented*

CR[8] <= 17'b00101000011000001; *// CE and LB signals activated and Preset Counter Incremented*

CR[9] <= 17'b00111000011010101; *// AD signal activated and Preset Counter Incremented*

CR[10] <= 17'b00111000010010011; *// EU and LA signals activated and Preset Counter Incremented*

CR[11] <= 17'b00111000101010001; *// Preset CLR signal activated*

*// SUB Micro-routine*

CR[12] <= 17'b00011100011010001; *// LM , EI signals activated and Preset Counter Incremented*

CR[13] <= 17'b00101000011000001; *// CE and LB signals activated and Preset Counter Incremented*

CR[14] <= 17'b00111000011011001; *// SU signal activated and Preset Counter Incremented*

CR[15] <= 17'b00111000010010011; *// EU and LA signals activated and Preset Counter Incremented*

CR[16] <= 17'b00111000101010011; *// Preset CLR signal activated*

*// Actually this design of CPU requires 5-bit address [i.e. memory with 32 memory locations] in order to accomodate the micro-routines of all instructions. The design has to be accordingly changed.*

*// OUT Micro-routine*

CR[17] <= 17'b00011100011010011;

CR[18] <= 17'b00111000011110010;

CR[19] <= 17'b00111000101010011;

*// NOP Micro-routine*

*//CR[14] <= 17'b00111000001010001;*

**end**

**endmodule**

**module** MICRO\_DECODER(**input** [16:0] CW , **output** EP , CP , LM , CE , LI , EI , CS , LOAD , CLR , INC , LA , EA , LB , SU , AD , EU , LO);

**assign** EP = CW[16];

**assign** CP = CW[15];

**assign** LM = CW[14];

**assign** CE = CW[13];

**assign** LI = CW[12];

**assign** EI = CW[11];

**assign** CS = CW[10];

**assign** LOAD = CW[9];

**assign** CLR = CW[8];

**assign** INC = CW[7];

**assign** LA = CW[6];

**assign** EA = CW[5];

**assign** LB = CW[4];

**assign** SU = CW[3];

**assign** AD = CW[2];

**assign** EU = CW[1];

**assign** LO = CW[0];

**endmodule**

**module** ACC\_8(**input** clk , **input** [8:0] ACC\_IN , **input** LA , **input** EA , **output** [8:0] ACC\_OUT\_ALU , **output**[8:0] ACC\_OUT\_BUS);

**reg** [8:0] ACC\_OUT\_r;

**assign** ACC\_OUT\_ALU = ACC\_OUT\_r;

**assign** ACC\_OUT\_BUS = EA ? ACC\_OUT\_r : 9'h000;

**always**@(**posedge** clk)

**begin**

**if**(~LA)

ACC\_OUT\_r <= ACC\_IN;

**end**

**endmodule**

**module** B\_REG\_8(**input** clk , **input** [8:0] B\_IN , **input** LB , **output** [8:0] B\_OUT);

**reg** [8:0] B\_OUT\_r;

**assign** B\_OUT = B\_OUT\_r;

**always**@(**posedge** clk)

**begin**

**if**(~LB)

B\_OUT\_r <= B\_IN;

**end**

**endmodule**

**module** OUT\_REG\_8(**input** clk , **input** [8:0] OUT\_IN , **input** LO , **output** [8:0]OUT\_P );

**reg** [8:0] OUT\_P\_r;

**assign** OUT\_P = OUT\_P\_r;

**always**@(**posedge** clk)

**begin**

**if**(~LO)

OUT\_P\_r<= OUT\_IN;

**end**

**endmodule**

**module** ALU\_8(**input**[8:0] ALU\_A , **input** [8:0] ALU\_B , **input** SU , **input** AD , **input** EU , **output** [8:0]ALU\_OUT); *// Here addition of AD signal , eliminated the need for a separate multiplexer for the accumulator.*

**wire** [8:0] ALU\_OUT\_w;

**assign** ALU\_OUT\_w = SU ? ALU\_A - ALU\_B : AD ? ALU\_A + ALU\_B : ALU\_OUT\_w;

**assign** ALU\_OUT = EU ? ALU\_OUT\_w : 9'h000;

**endmodule**

1. **RC SCRIPT & CONSTRAINTS-TOP.sdc**

set\_attr lib\_search\_path /home/install/FOUNDRY/digital/90nm/dig/lib

set\_attr hdl\_search\_path /root/Desktop/Vasu\_Main

set\_attr library slow.lib

read\_hdl MP.v

elaborate

read\_sdc /root/Desktop/Vasu\_Main/constraints\_top.sdc

set\_attribute syn\_generic\_effort medium

set\_attribute syn\_map\_effort medium

set\_attribute syn\_opt\_effort medium

syn\_generic

syn\_map

syn\_opt

report timing

report power

report qor

#synthesize -to\_mapped -effort medium

write\_sdf -timescale ns -nonegchecks -recrem split -edges check\_edge > delays.sdf

write\_hdl > MP\_netlist.v

write\_sdc > MP\_sdc.sdc

gui\_show

report timing > MP\_timing.rep

report power > MP\_power.rep

report area > MP\_cell.rep

**Constraints\_top.sdc**

create\_clock -name clk -period 10 -waveform {0 5} [get\_ports "clk"]

set\_clock\_transition -rise 0.1 [get\_clocks "clk"]

set\_clock\_transition -fall 0.1 [get\_clocks "clk"]

set\_clock\_uncertainty 0.01 [get\_ports "clk"]

set\_input\_delay -max 1.0 [get\_ports "rst"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "PC\_OUT"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "MAR\_OUT"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "IR\_OUT1"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "IR\_OUT2"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "DATA\_OUT1"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "ADDR\_OUT1"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "COUNT\_OUT"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "ACCUMULATOR\_OUT"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "DATA\_OUTPUT"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "B\_REG"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports " ALU\_OUT"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "OR\_out"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "CW"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "EP"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "CP"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "LM"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "CE"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "LI"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "EI"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "CS"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "LOAD"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "CLR"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "INC"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "LA"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "EA"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "LB"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "SU"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "AD"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "EU"] -clock [get\_clocks "clk"]

set\_output\_delay -max 1.0 [get\_ports "LO"] -clock [get\_clocks "clk"]

1. **STEPS OF PHYSICAL DESIGN**

* **LOGIC SYNTHESIS**

Logic synthesis is **the process of automatic production of logic components, in particular digital circuits**. It is a subject about how to abstract and represent logic circuits, how to manipulate and transform them, and how to analyze and optimize them.

**G-Netlist**

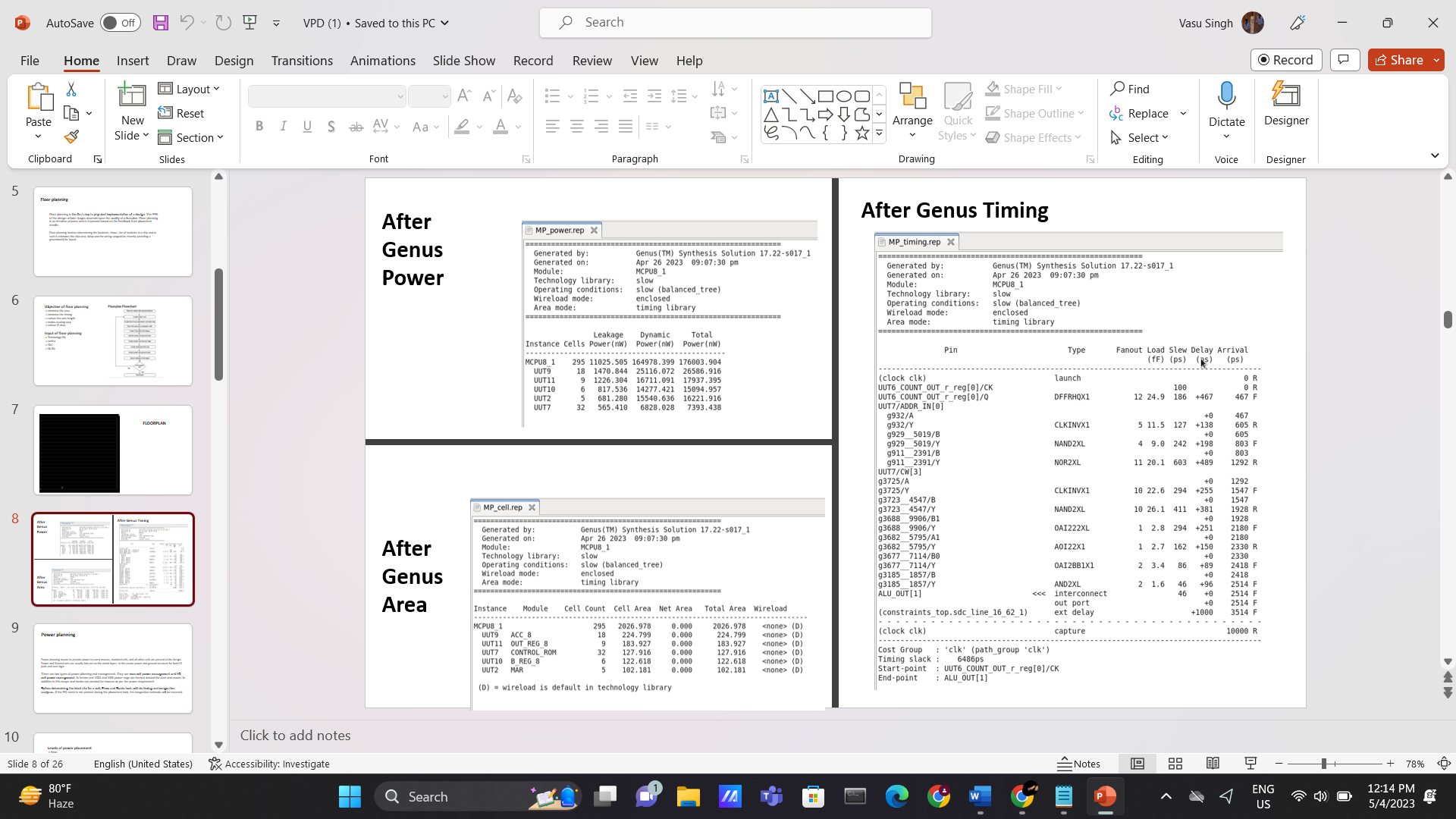
**Graphical user interface, text

Description automatically generated**

**Schematic**

**Graphical user interface

Description automatically generated**



* **FLOOR PLANNING**

Floor planning is **the first step in the physical implementation of a design**. The PPA of the design at later stages depends upon the quality of a floorplan. Floor planning is an iterative process and is improved based on the feedback from placement results.

Floor planning involves determining the locations, shape, size of modules in a chip and as such it estimates the chip area, delay, and the wiring congestion, thereby providing a groundwork for layout.

**Graphical user interface

Description automatically generated**

* **POWER PLANNING**

Power planning means to provide power to every macros, standard cells, and all other cells are present in the design. Power and Ground nets are usually laid out on the metal layers. In this create power and ground structure for both IO pads and core logic.

There are two types of power planning and management. They are **core cell power management and I/O cell power management**. In former one VDD and VSS power rings are formed around the core and macro. In addition to this straps and trunks are created for macros as per the power requirement.

**Before determining the ideal site for a cell, Place and Route tools will do timing and congestion analyses**. If the PG mesh is not present during the placement task, the congestion estimate will be incorrect.

**Levels of power placement**

* 1. Rings
  2. Stripes
  3. Rails
  4. Power vias
  5. Trunks

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

* **PRE-PLACEMENT**

**Physical Only Cells** (Well Taps, End Caps)

**Special Cells** (Spare cells, Decap cells)

**Cell Padding**

-> Cell padding is done to reserve space for avoiding routing congestion.

Pre-Placement Optimization

**Goals:**

Routability

Performance

Power (with cells)

**Optimization before Placement:**

Delay model must be removed

Zero RC optimization

Multi corner multi mode settings before Std. cell placement.

Graphical user interface

Description automatically generated

* **PLACEMENT**

Placement is the process of determining the locations of circuit devices on a die surface. It is an important stage in the VLSI design flow because it affects rout ability, performance, heat distribution, and to a less extent, power consumption of a design. It is the process of placing std cells in the design. The tools determine the location of each cell on the die. The tool places these based on the algorithm which it uses internally.

**A screenshot of a computer

Description automatically generated with medium confidence**

**A screenshot of a computer

Description automatically generated with medium confidence**

* **PRE-CTS**

**Pre CTS Area**

**Graphical user interface, application

Description automatically generated**

**Pre CTS-Timing**

Graphical user interface, text, application, email

Description automatically generated

* **PRE-CTS OPTIMIZATION**

Text

Description automatically generated with medium confidence

A picture containing text

Description automatically generated

* **CLOCK TREE SYNTHESIS (CTS)**

CTS is one of the most important stages in PnR. CTS QoR decides timing convergence and power. In most of the ICs clock consumes 30-40% of total power. So efficient clock architecture, clock gating and clock tree implementation helps to reduce power.

The process of distributing the clock and balancing the load is called CTS. Basically, delivering the clock to all sequential elements. CTS is the process of insertion of buffers or inverters along the clock paths of ASIC design to achieve zero/minimum skew or balanced skew. Before CTS, all clock pins are driven by a single clock source. CTS starting point is clock source and CTS ending point is clock pins of sequential cells.

Graphical user interface, chart

Description automatically generated

add\_ndr -width {Metal1 0.12 Metal2 0.14 Metal3 0.14 Metal4 0.14 Metal5 0.14 Metal6 0.14 Metal7 0.14 Metal8 0.14 Metal9 0.14} -spacing {Metal1 0.12 Metal2 0.14 Metal3 0.14 Metal4 0.14 Metal5 0.14 Metal6 0.14 Metal7 0.14 Metal8 0.14 Metal9 0.14} -name 2w2s

create\_route\_type -name clkroute -non\_default\_rule 2w2s -bottom\_preferred\_layer Metal5 -top\_preferred\_layer Metal6

set\_ccopt\_property route\_type clkroute -net\_type trunk

set\_ccopt\_property route\_type clkroute -net\_type leaf

set\_ccopt\_property buffer\_cells {CLKBUFX8 CLKBUFX12}

set\_ccopt\_property inverter\_cells {CLKINVX8 CLKINVX12}

set\_ccopt\_property clock\_gating\_cells TLATNTSCA\*

create\_ccopt\_clock\_tree\_spec -file ccopt.spec

* **POST-CTS**

**Post CTS- Setup Time**

**Graphical user interface, text, application, email

Description automatically generated**

**Post CTS- Hold Time**

**Graphical user interface, text, application, email

Description automatically generated**

**Post CTS- Area**

**Graphical user interface, text, application, email

Description automatically generated**

**Post CTS- Timing**

**Graphical user interface, text, application, email

Description automatically generated**

**Post CTS- Power**

**A picture containing graphical user interface

Description automatically generated**

**Rest all the results/values after Post-CTS, (including Pre-CTS, Logic Synthesis as well) are displayed in the at the end of the document.**

* **ROUTING**

Routing in VLSI is making physical connections between signal pins using metal layers.

**Importance**

* 1. Gate delay decreases
  2. interconnect resistance increases
  3. interconnect capacitance dominates total gate loading.

**Objective**

* 1. skew requirement
  2. Open circuit clean
  3. routed paths must meet setup and hold timing margin.
  4. DRVs max. capacitance/transitions must be under the limit.

**Goals**

* 1. minimize the total wire length.
  2. minimize the critical path delay.
  3. minimize the congestion hotspots.
  4. meeting the timing DRCs and obtaining a good timing QoR.

A screenshot of a computer

Description automatically generated with medium confidence

1. **FINAL RESULTS**

* **Logic Synthesis**
* Hold Path: 121
* Total No of paths: 254
* Total Nets: 121
* Timing Slack(nanosecond): 6.486
* Total Area: 2026.978
* Cell Count: 295
* **Floor Planning and Power Planning**
* Aspect ratio: 0.857
* Core utilization: 0.699(69.9%)
* Distance from core to I/O pins: 8
* Width of stripes: 2
* No of stripes: 3
* Spacing: 0.5
* **Pre-placement**
* Width of Chip/Core Area: 58.392
* End Cap: FILL1
* Well Tap: FILL2
* **Placement and Optimization (Pre-CTS)**
* Total Area: 1986.1056 (**Area Decreases**)
* Timings: **WNS:** 5.689

**TNS:** 0.00

**Optimization:**

* Total Power: 0.1101
* Total Area: 1986.8625(**Area Increases**)
* Timings: **WNS:** 5.673

**TNS:** 0.00

* **Post-CTS**
* Total Area: 1986.8625
* Setup-Timings: **WNS:** 5.673

**TNS:** 0.00

* Hold-Timings: **WNS:** 0.059

**TNS:** 0.00

* Total Power: 0.1101

**Optimization:**

* Total Power: 0.1101
* Total Area: 1986.8625
* Timings: **WNS:** 5.673

**TNS:** 0.00

* **Post-Route**
* Total Area: 1986.8625
* Setup-Timings: **WNS:** 5.578 (**Slack Decreases**)

**TNS:** 0.00

* Total Power: 0.11006 (**Small Change**)

**Optimization:**

* Total Power: 0.11006
* Total Area: 1986.8625
* Timings: **WNS:** 5.578

**TNS:** 0.00