STUDY OF DIGITAL,ANALOG AND MIXED SIGNAL SYSTEMS ACROSS DIFFERENT ARCHITECTURES

Mid-Review-3

GITAM
DEEMED TO BE UNIVERSITY

AY 2021-25

GITAM (Deemed-to-be) University

Major Project
Project ID:
PROJ3999

Project Team:

- BU21EECE0100376
- BU21EECE0100080
- BU21EECE0100294

Department of Electrical Electronics and Communication Engineering

Project Mentor:

- Dr. Prithvi Sekhar Pagala

Project In-charge:

- Dr. Rohan Prasad

Capstone Project 1 : Summary

- This project focused on the building a microprocessor using basic building blocks of a microprocessor.
- It included studying components like logic gates, ALU, which are essential for building circuits.
- It also looked at comparing the technologies like CMOS, MOSFET, and BJT to find ways to make circuits faster, more efficient, and reliable.
- The circuits were tested and analyzed using a simulation tool called Cadence Virtuoso Tool.
- Different system designs were compared based on their power consumption, size, speed, and delay. Important performance factors like power consumption, speed, size and scalability were also studied to understand how they affect system designs.

CMOS

LOGIC GATES	POWER CONSUMPTION (microwatts)	DELAY (picoseconds)	SIZE(Nm)
NOT	100	150	120
AND	110	160	120
OR	105	155	120
NAND	115	170	120
NOR	112	165	120
X-OR	130	180	120
X-NOR	135	190	120

Capstone Project 1 : Summary

BJT

LOGIC GATES	POWER CONSUMPTION (microwatts)	DELAY (picoseconds)	SIZE(Nm)
NOT	500	700	200
AND	550	750	200
OR	520	730	200
NAND	580	770	200
NOR	560	760	200
X-OR	600	800	200
X-NOR	620	820	200

MOSFET

LOGIC GATES	POWER CONSUMPTION (microwatts)	DELAY (picoseconds)	SIZE(Nm)
NOT	200	300	180
AND	220	320	180
OR	210	310	180
NAND	230	330	180
NOR	225	325	180
X-OR	250	350	180
X-NOR	260	370	180

Objective and Goals

Objective

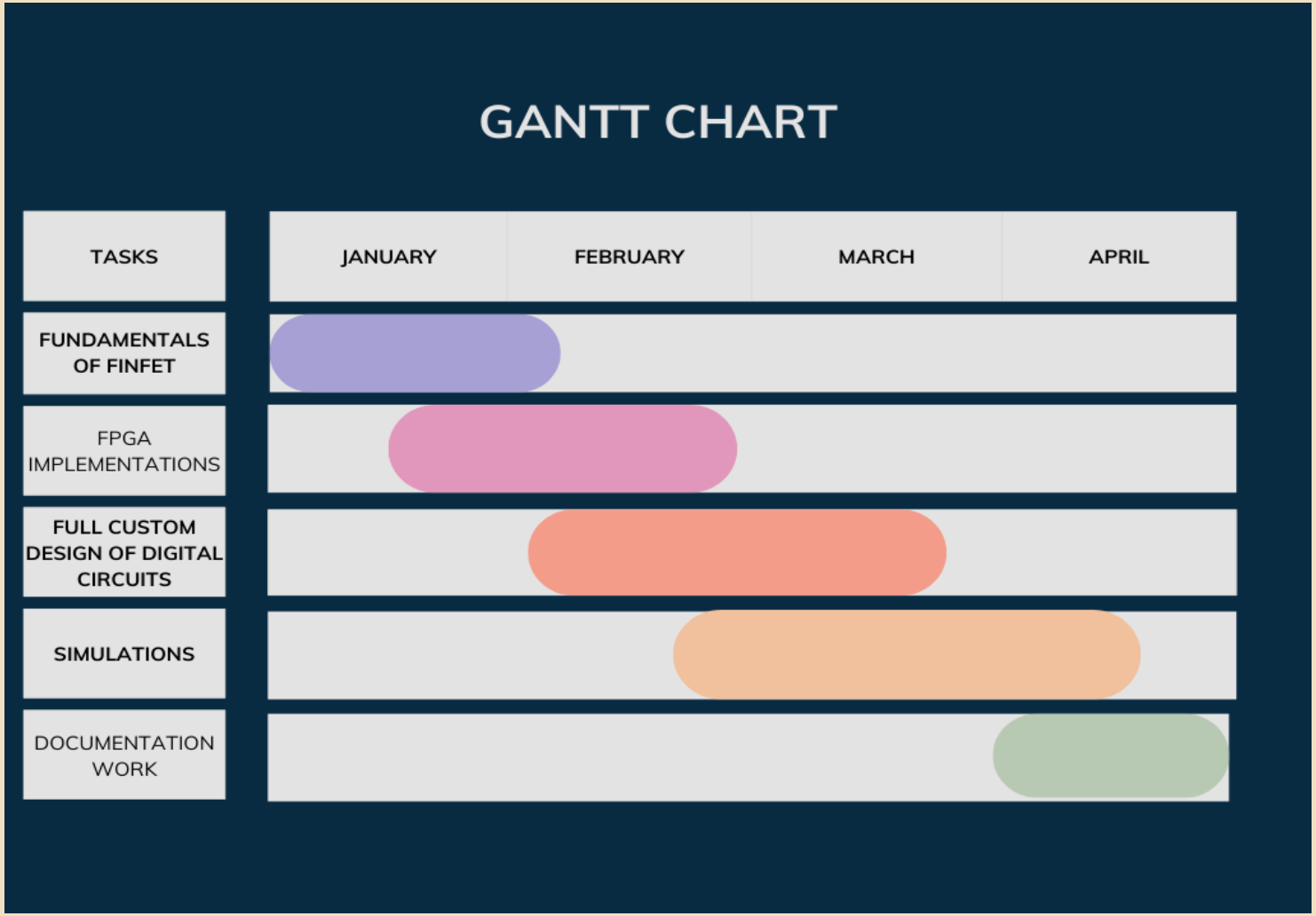
•Study and integrate CMOS,BJT, basic MOSFET technology into the circuit design process.
•Design analog, digital, and mixed-signal subsystems as building blocks for the microprocessor.
•Ensure the microprocessor meets key performance metrics like low power consumption , size , delay and high speed.
•Validate and optimize the designs through simulation and testing.
•Compare the designed microprocessor with traditional architectures to highlight improvements.

Goals

- Exploring High-Performance and Efficient Devices.
- Design a custom microprocessor using knowledge from analog, digital, and mixed-signal systems.
- Implement previously studied circuits using CMOS, BJT, basic MOSFET technology for improved performance
- Develop a system that is efficient in terms of power, speed, size and delay.

Project Plan

Gant Chart - Milestones and Activities



Literature Survey

Key Publications

- VLSI DSP education
- A Mixed-signal Analog-Digital Integrator Design
- A case study of mixed-signal integrated circuit testing: an application of current testing using the upper limit and the lower limit
- A comparative study of advanced mosfet concepts.

Key Resources – Whitepaper| Application Notes | Datasheet| Others

- https://ieeexplore.ieee.org/document/9167646
- https://ieeexplore.ieee.org/document/10592992
- https://ieeexplore.ieee.org/document/6649058

Existing Implementations – Products| Opensource| GitHub etc

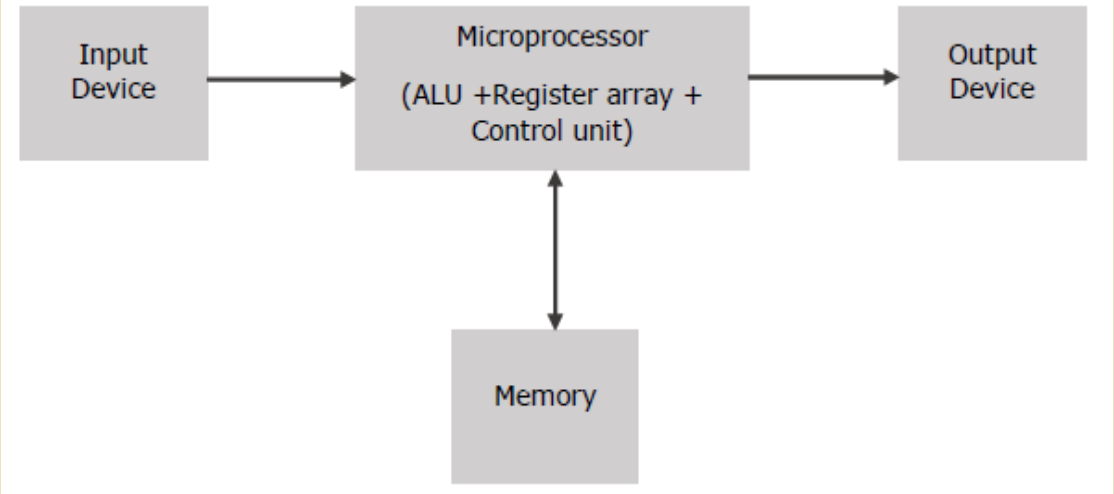
- https://github.com/bernardlee99/4-bit-alu
- https://github.com/microchip-pic-avr-examples/pic18f56q71-led-current-control-opamp
- https://github.com/jbp261/Microcontroller-Projects/blob/master/ADC-DAC/README.md

Architecture

Microprocessor

❑ **BASIC COMPONENTS :**

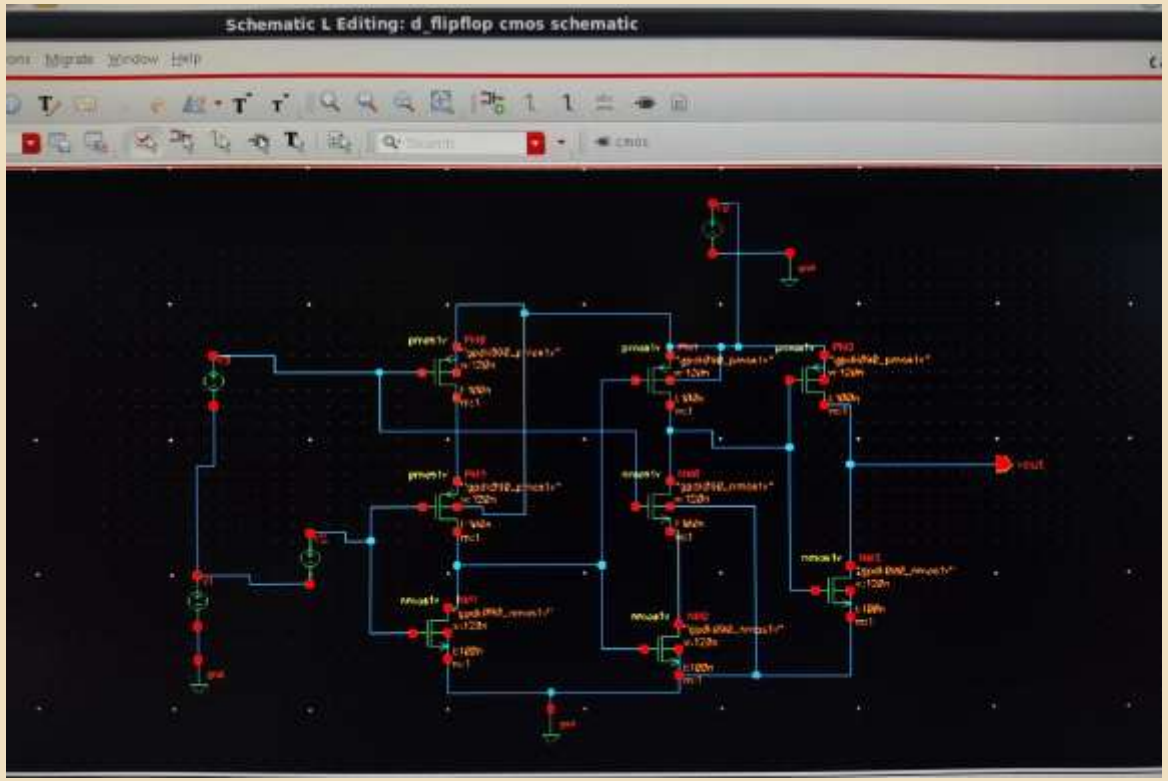
- Input device
- Output device
- ALU
- Memory (flip-flops)



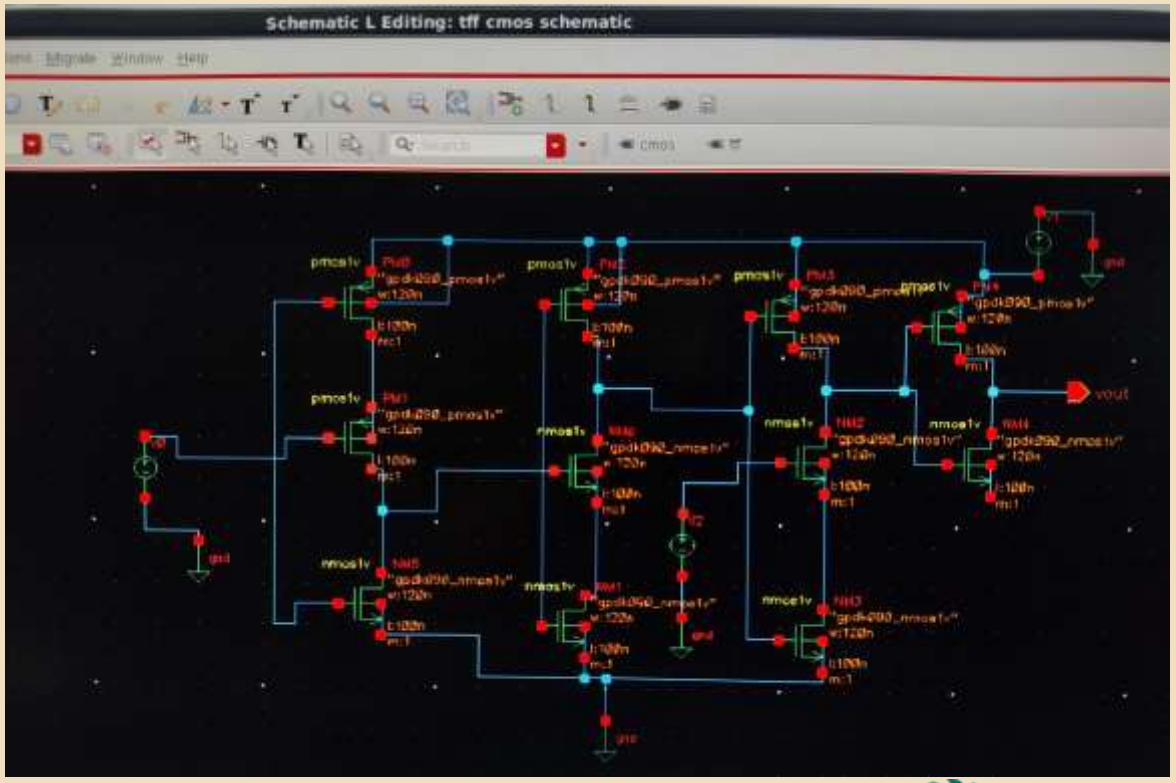
Schematic Diagrams

FLIPFLOPS(MEMORY)

D-FlipFlop



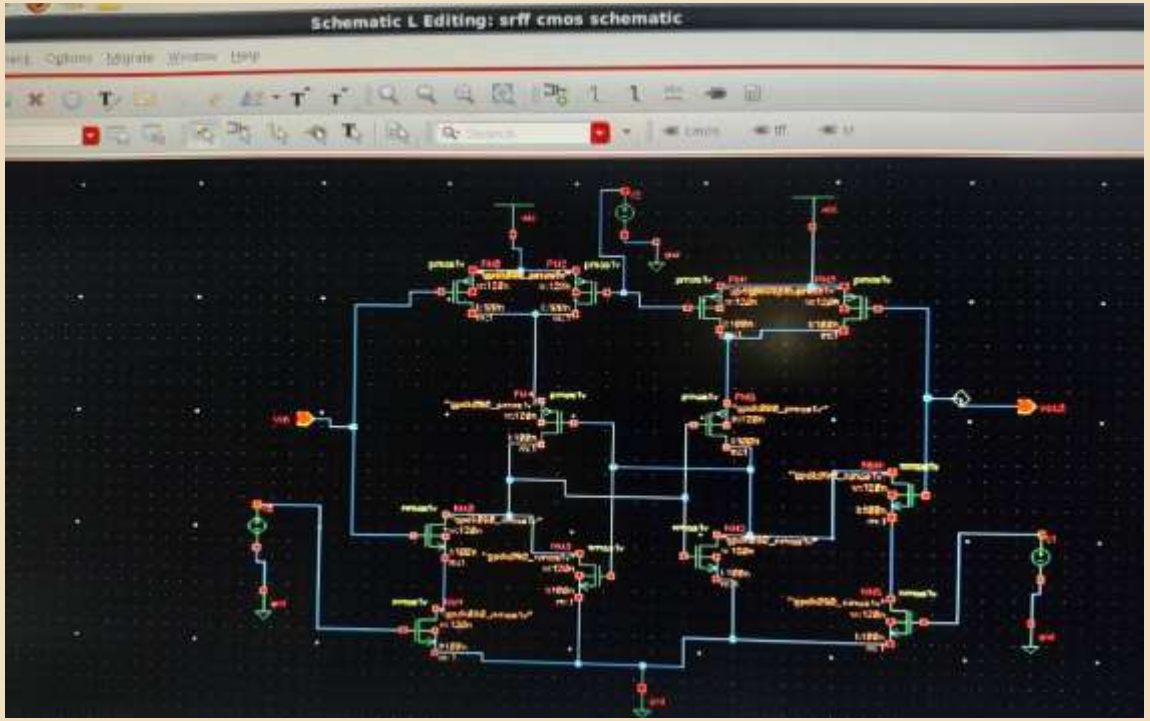
T-FlipFlop



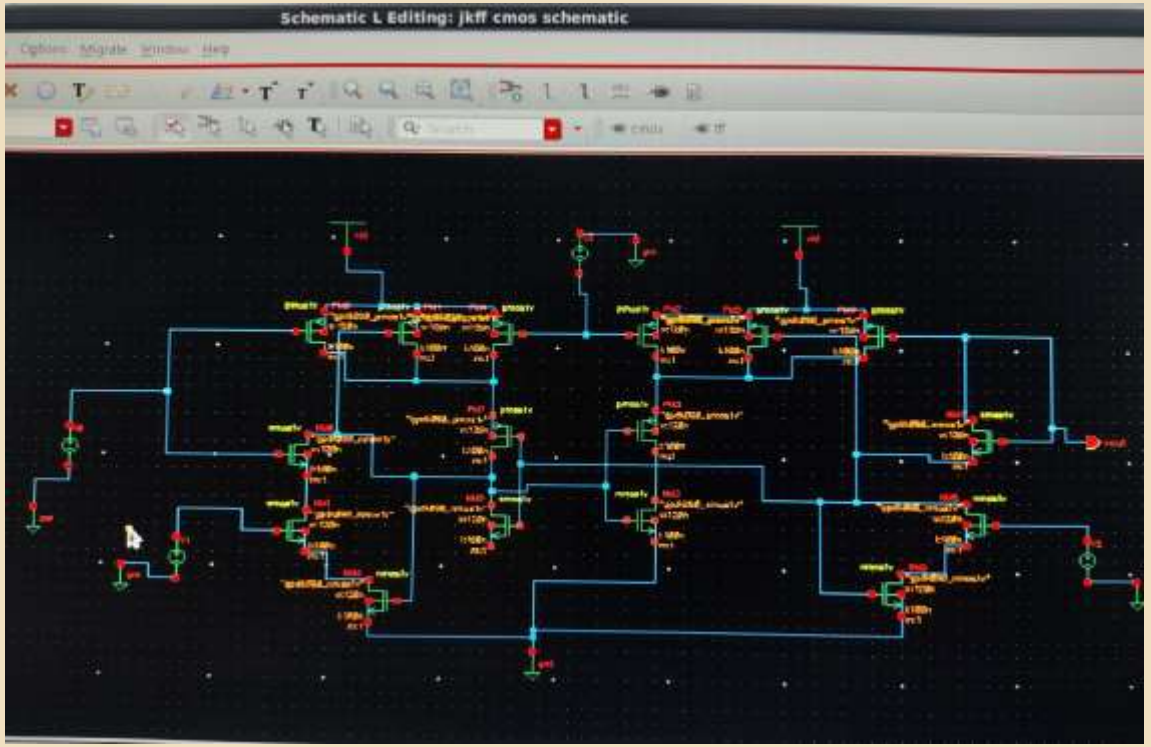
Schematic Diagrams

FLIPFLOPS(MEMORY)

SR-FlipFlop



JK-FlipFlop



Parameters

BJT

FLIP-FLOPS	POWER CONSUMPTION (microwatts)	DELAY (picoseconds)	SIZE(Nm)
SR FLIPFLOP	800	900	200
JK FLIPFLOP	850	950	200
T FLIPFLOP	900	1000	200
D FLIPFLOP	880	980	200

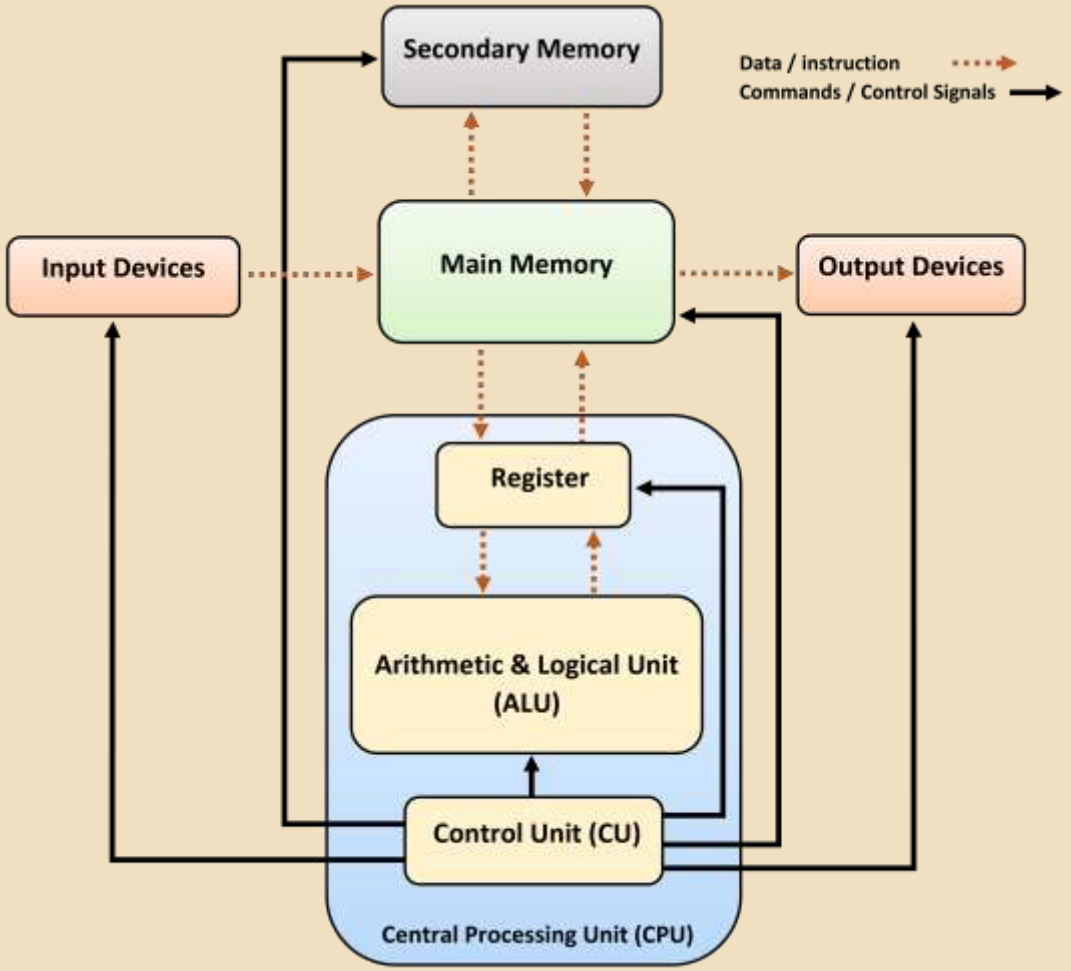
Basic MOSFET

FLIPFLOPS	POWER CONSUMPTION (microwatts)	DELAY (picoseconds)	SIZE(Nm)
SR FLIPFLOP	150	250	180
JK FLIPFLOP	160	280	180
T FLIPFLOP	180	320	180
D FLIPFLOP	170	300	180

CMOS

FLIPFLOPS	POWER CONSUMPTION (microwatts)	DELAY (picoseconds)	SIZE(Nm)
SR FLIPFLOP	120	180	120
JK FLIPFLOP	130	200	120
T FLIPFLOP	140	220	120
D FLIPFLOP	135	210	120

CPU[RISC Microprocessor]



- low power consumption
- fast execution
- simpler design
- cost-effective
- pipelining

Implementation[CODE]

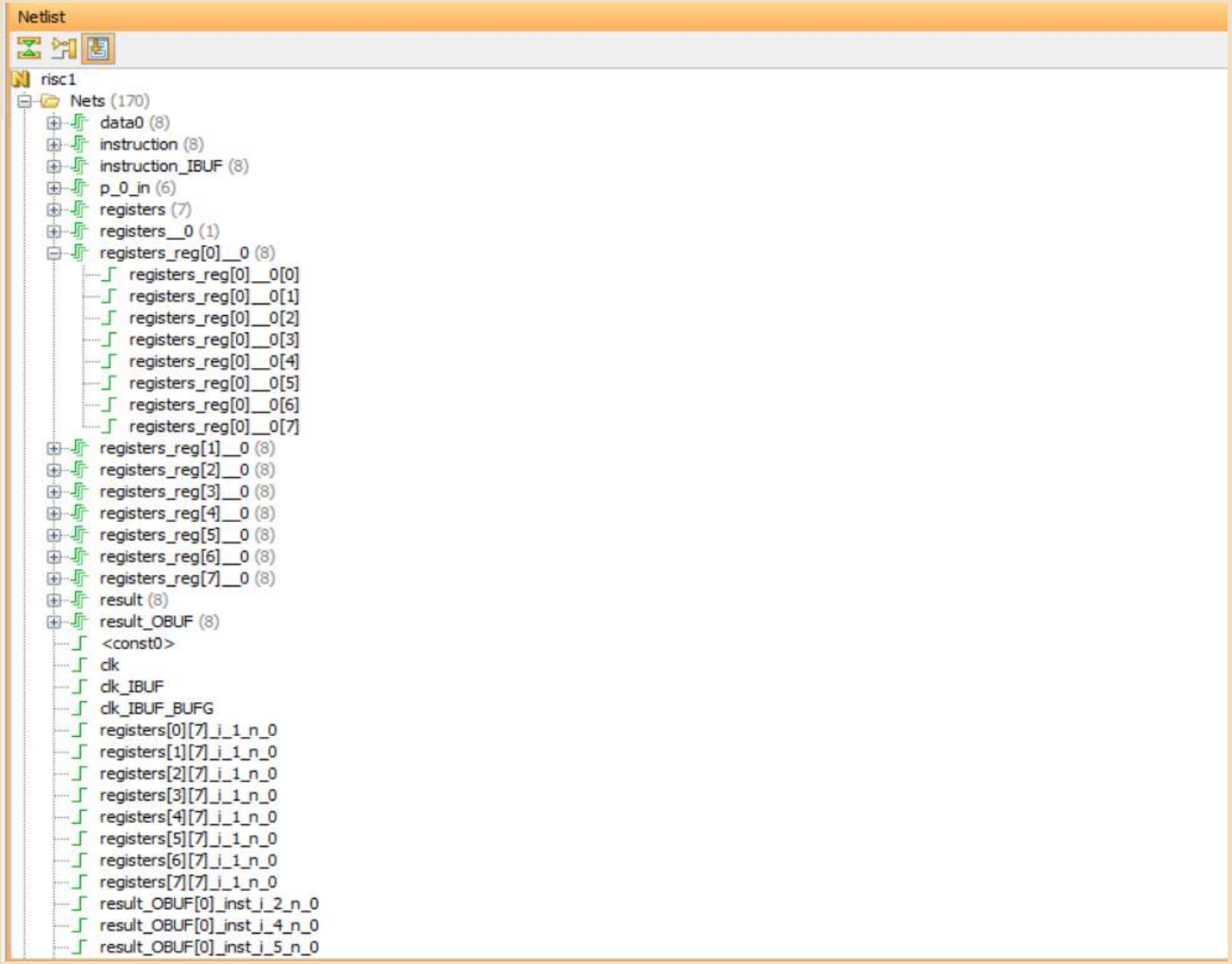
```
module risc1(
    input clk,                  // Clock input
    input rst,                  // Reset input
    input [7:0] instruction,    // 8-bit instruction input
    output [7:0] result         // 8-bit result output
);

    // Internal signals
    wire [7:0] reg1_data, reg2_data;   // Data from registers
    reg [7:0] alu_out;                 // ALU output (must be reg
type)
    wire zero_flag;                    // Zero flag from ALU
    wire write_enable;                 // Enable writing to
register file
    wire [2:0] opcode;                 // ALU operation code
    wire [2:0] reg1, reg2, reg_dest;  // Register addresses

    // Register File (8 registers, each 8 bits wide)
    reg [7:0] registers [7:0];   // 8 registers, each 8 bits wide
```

```
// ALU Logic (Arithmetic Logic Unit)
    always @(*) begin
        case (opcode)
            3'b000: alu_out = reg1_data + reg2_data;  // ADD
            3'b001: alu_out = reg1_data - reg2_data;  // SUB
            3'b010: alu_out = reg1_data & reg2_data;  // AND
            3'b011: alu_out = reg1_data | reg2_data;  // OR
            3'b100: alu_out = reg1_data;              // MOV
            default: alu_out = 8'b00000000;           // Default to
zero
        endcase
    end
    // Zero Flag Logic (zero flag is 1 when ALU result is
zero)
    assign zero_flag = (alu_out == 8'b00000000) ? 1 : 0;
    // Register File Read Logic (Read data from the
registers)
    assign reg1_data = registers[reg1];  // Read data from
register 1
    assign reg2_data = registers[reg2];  // Read data from
register 2
```

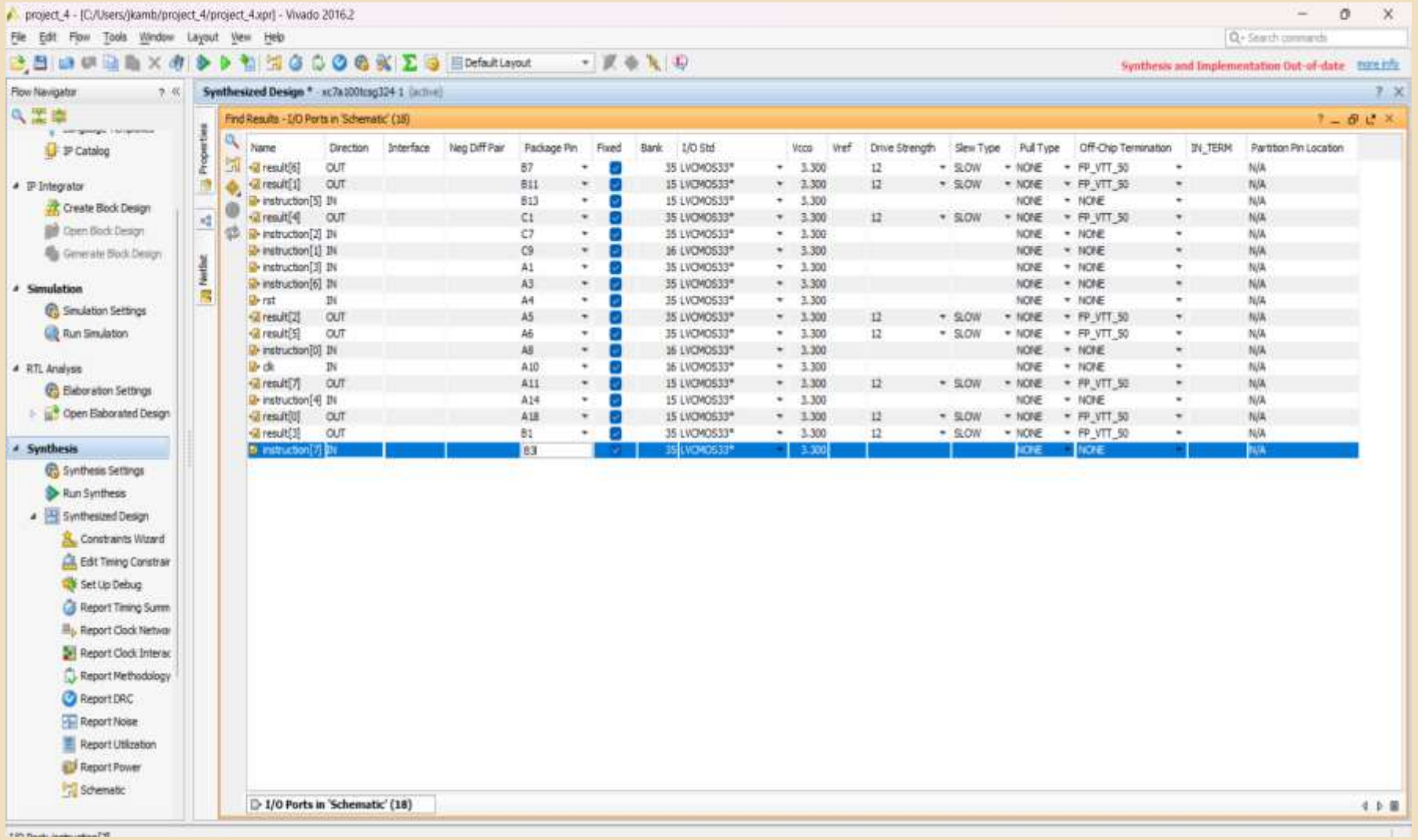
Implementation[CODE]

```
// Instruction Decoding (Extract opcode and register addresses from the instruction)
    assign opcode = instruction[7:5];   // Opcode is the upper 3 bits
    assign reg1 = instruction[4:2];     // Register 1 is bits 4 to 2
    assign reg2 = instruction[1:0];     // Register 2 is bits 1 to 0
    assign reg_dest = instruction[4:2]; // Destination register (same as reg1 for now)
    assign write_enable = 1'b1;         // Always enable write (simplified control)

    // Register File Write Logic (Write ALU output to the destination register)
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            // Reset all registers to 0 on reset signal
            registers[0] <= 8'd0;
            registers[1] <= 8'd0;
            registers[2] <= 8'd0;
            registers[3] <= 8'd0;
            registers[4] <= 8'd0;
            registers[5] <= 8'd0;
            registers[6] <= 8'd0;
            registers[7] <= 8'd0;
        end else if (write_enable) begin
                // Write ALU output to the destination register
                registers[reg_dest] <= alu_out;
            end
        end

        // Final result output (assign ALU output to result)
        assign result = alu_out;

endmodule
```

NETLIST


Netlist
risc1
Nets (170)
data0 (8)
instruction (8)
instruction_IBUF (8)
p_0_in (6)
registers (7)
registers__0 (1)
registers_reg[0]__0 (8)
registers_reg[0]__0[0]
registers_reg[0]__0[1]
registers_reg[0]__0[2]
registers_reg[0]__0[3]
registers_reg[0]__0[4]
registers_reg[0]__0[5]
registers_reg[0]__0[6]
registers_reg[0]__0[7]
registers_reg[1]__0 (8)
registers_reg[2]__0 (8)
registers_reg[3]__0 (8)
registers_reg[4]__0 (8)
registers_reg[5]__0 (8)
registers_reg[6]__0 (8)
registers_reg[7]__0 (8)
result (8)
result_OBUF (8)
<const0>
clk
clk_IBUF
clk_IBUF_BUFG
registers[0][7]_i_1_n_0
registers[1][7]_i_1_n_0
registers[2][7]_i_1_n_0
registers[3][7]_i_1_n_0
registers[4][7]_i_1_n_0
registers[5][7]_i_1_n_0
registers[6][7]_i_1_n_0
registers[7][7]_i_1_n_0
result_OBUF[0]_inst_i_2_n_0
result_OBUF[0]_inst_i_4_n_0
result_OBUF[0]_inst_i_5_n_0

Synthesized design



Contribution

Team Progress and Movement

- ALU
- memory
- RISC processor[CPU]

Individual Contribution

Key contributions: Voruguntla Mounika

- Implementation of circuits using CMOS technology and checking the parameters power consumption, area, size and delay.
- Documentation work (ppt , word document) .
- Implemented a RISC processor.

Key contributions: Polimera Sandeep Reddy

- Implementation of circuits using basic MOSFET technology and checking the parameters power consumption, area, size and delay.
- Updating project in Github.

Key contributions: Yeruva Aditya udaya Reddy

- Literature survey.
- Collecting required information.
- Works on ADC and DAC.

Conclusion & Future Work

Summary and Conclusion

- **RISC Processor in Verilog (Xilinx Vivado):** This method uses coding (Verilog) to design a processor, which can be tested and run on an FPGA. It is faster and easier since you work with digital logic instead of individual transistors.
- **Building with Transistors (Cadence CMOS/BJT/MOSFET):**This approach designs the processor at the transistor level, using CMOS, BJT, MOSFET. It gives more control over power and performance but is much harder and takes more time.
- Our project aims to design a microprocessor by studying and integrating analog, digital, and mixed-signal systems using advanced semiconductor technologies such as BJT, CMOS, and MOSFET.
- The focus is on leveraging the unique characteristics and strengths of each technology to optimize the performance, power consumption, area, and delay of the microprocessor.

Future Work

- This future work plan outlines the key steps and considerations involved in designing a microprocessor using ALU, flip-flops, ADC, DAC, filters, and op-amps using FINFET technology
- By following this plan, we aim to develop a high-performance and reliable microprocessor that meets the specific requirements of our project.

THANK YOU

Have a Great Day !