PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

100-ft Ring Road, Bengaluru – 560 085, Karnataka, India

LOW POWER VLSI PROJECT

LOW POWER ALU DESIGN using cadence genus

Submitted by-

Harshitha R -PES1UG22EC088

Jan - May 2025

under the guidance of

Prof. Babitha S Ullal
Department of Electronics and Communication Engineering
PES University

PROGRAM B.TECH

Introduction:

The Arithmetic Logic Unit (ALU) is a core component in processors responsible for performing arithmetic and logical operations. In digital systems, especially in embedded and battery-powered applications, power efficiency is a key design parameter. This project implements a 4-bit ALU capable of performing 8 operations (ADD, SUB, AND, OR, XOR, SHL, SHR, NOT), focusing on optimizing power consumption through clock gating.

Motivation:

Conventional ALU designs consume dynamic power continuously, even when inactive. To enhance power efficiency, this project explores a design methodology that disables clock propagation to parts of the circuit when not in use. By integrating clock gating, we reduce unnecessary switching activity and conserve power, making the design suitable for low-power systems such as IoT devices, wearable electronics, and embedded controllers.

Unoptimized ALU design –

Design code –

```
`timescale 1ns/1ps
module alu (
  input wire clk,
  input wire rst,
  input wire enable,
  input wire [3:0] A,
  input wire [3:0] B,
  input wire [2:0] opcode,
  output reg [3:0] result,
  output reg carry,
  output reg zero
);

  // Registered inputs
  reg [3:0] A_reg, B_reg;
  reg [2:0] opcode_reg;
  always @(posedge clk or posedge rst) begin
    if (rst) begin
      A_reg <= 4'd0;
      B_reg <= 4'd0;
      opcode_reg <= 3'd0;
```

```
    end else if (enable) begin
      A_reg <= A;
      B_reg <= B;
      opcode_reg <= opcode;
    end
  end

  // ALU Operation
  always @(posedge clk or posedge rst) begin
    if (rst) begin
      result <= 4'd0;
      carry <= 1'b0;
      zero <= 1'b0;
    end else if (enable) begin
      carry <= 1'b0;
      case (opcode_reg)
        3'b000: {carry, result} <= A_reg + B_reg;   // ADD
        3'b001: {carry, result} <= A_reg - B_reg;   // SUB
        3'b010: result <= A_reg & B_reg;   // AND
        3'b011: result <= A_reg | B_reg;   // OR
        3'b100: result <= A_reg ^ B_reg;   // XOR
        3'b101: result <= A_reg << 1;   // SHL
        3'b110: result <= A_reg >> 1;   // SHR
        3'b111: result <= ~A_reg;   // NOT
        default: result <= 4'd0;
      endcase
      zero <= (result == 4'd0);
    end
  end

endmodule
```

Testbench code -

```
`timescale 1ns/1ps
module alu_tb;
  reg clk, rst, enable;
  reg [3:0] A, B;
  reg [2:0] opcode;
  wire [3:0] result;
  wire carry, zero;

  // Instantiate DUT
  alu_top uut (
    .clk(clk),
    .rst(rst),
    .enable(enable),
```

```
    .A(A),
    .B(B),
    .opcode(opcode),
    .result(result),
    .carry(carry),
    .zero(zero)
  );

  // Clock generation
  initial begin
    clk = 0;
    forever #5 clk = ~clk;  // 10ns clock
  end

  // Stimulus
  initial begin
    $dumpfile("alu_waveform.vcd");
    $dumpvars(0, alu_tb);

    rst = 1; enable = 0; A = 0; B = 0; opcode = 3'b000;
    #12 rst = 0;

    enable = 1;
    A = 4'd5; B = 4'd3; opcode = 3'b000; #20;  // ADD
    A = 4'd7; B = 4'd2; opcode = 3'b001; #20;  // SUB
    A = 4'd12; B = 4'd10; opcode = 3'b010; #20; // AND
    A = 4'd6; B = 4'd9; opcode = 3'b011; #20;  // OR
    A = 4'd7; B = 4'd5; opcode = 3'b100; #20;  // XOR
    A = 4'd3; opcode = 3'b101; #20;  // SHL
    A = 4'd8; opcode = 3'b110; #20;  // SHR
    A = 4'd4; opcode = 3'b111; #20;  // NOT

    // Test with enable OFF
    enable = 0;
    A = 4'd9; B = 4'd9; opcode = 3'b000; #20;

    #20 $finish;
  end

endmodule
```

Optimized ALU design –

Design cde –

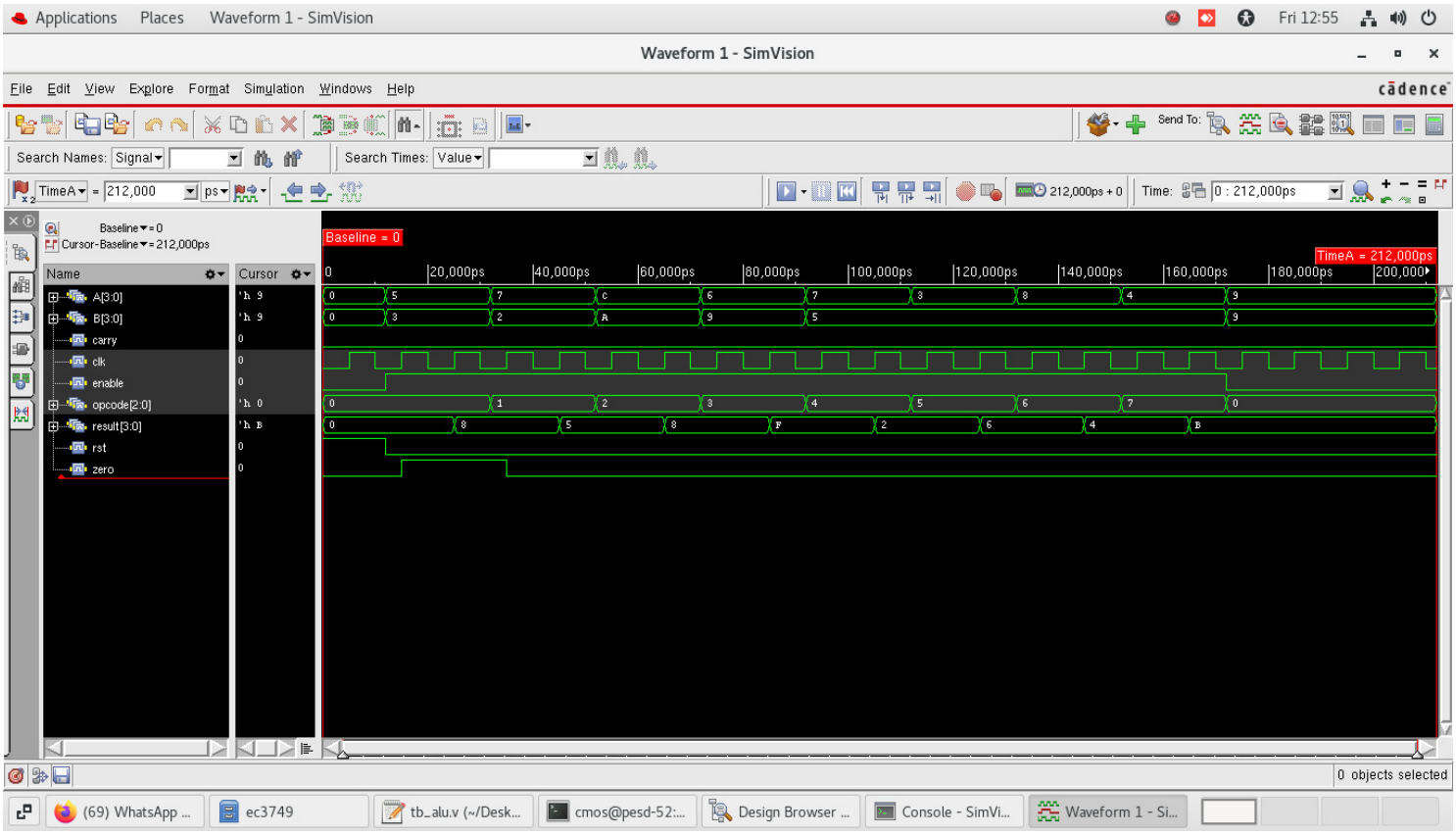
```
`timescale 1ns/1ps

module alu_top (
   input wire clk,
   input wire rst,
   input wire enable,
   input wire [3:0] A,
   input wire [3:0] B,
   input wire [2:0] opcode,
   output reg [3:0] result,
   output reg carry,
   output reg zero
);

   // Clock gating logic
   wire gated_clk;
   reg enable_reg;

   always @(posedge clk or posedge rst) begin
      if (rst)
         enable_reg <= 1'b0;
      else
         enable_reg <= enable;
   end

   assign gated_clk = clk & enable_reg;

   // Registered inputs
   reg [3:0] A_reg, B_reg;
   reg [2:0] opcode_reg;

   always @(posedge gated_clk or posedge rst) begin
      if (rst) begin
         A_reg <= 4'd0;
         B_reg <= 4'd0;
         opcode_reg <= 3'd0;
      end else begin
         A_reg <= A;
         B_reg <= B;
         opcode_reg <= opcode;
      end
   end

   // ALU Operation
```

```
    always @(posedge gated_clk or posedge rst) begin
        if (rst) begin
            result <= 4'd0;
            carry <= 1'b0;
            zero <= 1'b0;
        end else begin
            carry <= 1'b0;

            case (opcode_reg)
                3'b000: {carry, result} <= A_reg + B_reg;   // ADD
                3'b001: {carry, result} <= A_reg - B_reg;   // SUB
                3'b010: result <= A_reg & B_reg;            // AND
                3'b011: result <= A_reg | B_reg;            // OR
                3'b100: result <= A_reg ^ B_reg;            // XOR
                3'b101: result <= A_reg << 1;               // SHL
                3'b110: result <= A_reg >> 1;               // SHR
                3'b111: result <= ~A_reg;                   // NOT
                default: result <= 4'd0;
            endcase
            zero <= (result == 4'd0);
        end
    end

endmodule
```

Testbench code –

```
`timescale 1ns/1ps

module tb_alu;

    reg clk, rst, enable;
    reg [3:0] A, B;
    reg [2:0] opcode;
    wire [3:0] result;
    wire carry, zero;

    // Instantiate DUT
    alu_top uut (
        .clk(clk),
        .rst(rst),
        .enable(enable),
```

```
    .A(A),
    .B(B),
    .opcode(opcode),
    .result(result),
    .carry(carry),
    .zero(zero)
  );

  // Clock generation
  initial begin
    clk = 0;
    forever #5 clk = ~clk;  // 10ns clock
  end

  // Stimulus
  initial begin
    $dumpfile("alu_waveform.vcd");
    $dumpvars(0, tb_alu);

    rst = 1; enable = 0; A = 0; B = 0; opcode = 3'b000;
    #12 rst = 0;

    enable = 1;
    A = 4'd5; B = 4'd3; opcode = 3'b000; #20;  // ADD
    A = 4'd7; B = 4'd2; opcode = 3'b001; #20;  // SUB
    A = 4'd12; B = 4'd10; opcode = 3'b010; #20; // AND
    A = 4'd6; B = 4'd9; opcode = 3'b011; #20;  // OR
    A = 4'd7; B = 4'd5; opcode = 3'b100; #20;  // XOR
    A = 4'd3; opcode = 3'b101; #20;  // SHL
    A = 4'd8; opcode = 3'b110; #20;  // SHR
    A = 4'd4; opcode = 3'b111; #20;  // NOT

    // Test with enable OFF
    enable = 0;
    A = 4'd9; B = 4'd9; opcode = 3'b000; #20;

    #20 $finish;
  end

endmodule
```



Working

The ALU accepts two 4-bit inputs (A and B) and a 3-bit opcode to determine the operation. It also has control signals: clk, rst, and enable.

Opcode	Operation	Description
000	ADD	Adds A and B
001	SUB	Subtracts B from A

010	AND	Bitwise AND
011	OR	Bitwise OR
100	XOR	Bitwise XOR
101	SHL	Shift A left by 1
110	SHR	Shift A right by 1
111	NOT	Bitwise NOT of A

Enable Signal:

- When enable is high, the ALU performs the operation.
- When enable is low, inputs and outputs remain unchanged, avoiding unnecessary computation.

Clock Gating (Optimized Design):

- Instead of allowing the entire ALU to receive the system clock continuously, a gated clock (gated_clk) is generated by combining the main clk with a registered version of enable.
- The ALU logic and input latching are only triggered when enable is high, reducing dynamic power.

Applications Places Text Editor Fri 12:49

Open alu_report.rpt ~/Desktop/ec3749 Save

alu_report.rpt alu_top_report.rpt

```
Instance: /alu
Power Unit: W
PDB Frames: /stim#0/frame#0
  ------------------------------------------------------------------------
    Category         Leakage     Internal    Switching        Total    Row%
  ------------------------------------------------------------------------
       memory     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
     register     1.04242e-08  5.96233e-06  7.56410e-07  6.72916e-06  68.62%
        latch     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
        logic     6.06388e-09  2.33563e-06  7.36108e-07  3.07780e-06  31.38%
         bbox     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
        clock     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
          pad     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
           pm     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
  ------------------------------------------------------------------------
     Subtotal     1.64880e-08  8.29796e-06  1.49252e-06  9.80696e-06 100.00%
   Percentage           0.17%       84.61%       15.22%      100.00% 100.00%
  ------------------------------------------------------------------------
```

Plain Text Tab Width: 8 Ln 1, Col 15 INS

(69) WhatsApp — Mozilla Firefox ec3749 alu_report.rpt (~/Desktop/ec3749) ...

Applications Places Text Editor Fri 12:50

Open alu_top_report.rpt ~/Desktop/ec3749 Save

alu_report.rpt alu_top_report.rpt

```
Instance: /alu_top
Power Unit: W
PDB Frames: /stim#0/frame#0
  ------------------------------------------------------------------------
    Category         Leakage     Internal    Switching        Total    Row%
  ------------------------------------------------------------------------
       memory     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
     register     1.03254e-08  3.54489e-06  2.13247e-06  5.68768e-06  55.95%
        latch     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
        logic     6.53524e-09  1.58747e-06  2.88394e-06  4.47794e-06  44.05%
         bbox     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
        clock     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
          pad     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
           pm     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
  ------------------------------------------------------------------------
     Subtotal     1.68607e-08  5.13235e-06  5.01641e-06  1.01656e-05 100.00%
   Percentage           0.17%       50.49%       49.35%      100.00% 100.00%
  ------------------------------------------------------------------------
```

Plain Text Tab Width: 8 Ln 1, Col 19 INS

(69) WhatsApp — Mozilla Firefox Pictures alu_top_report.rpt (~/Desktop/ec3... Screenshot from 2025-04-25 12-4...

Difference Between Unoptimized and Optimized Design

Feature	**Unoptimized ALU**	**Optimized ALU with Clock Gating**
Clock Propagation	Always active	Controlled by enable using gating
Power Consumption	Higher due to continuous switching	Reduced by avoiding unnecessary toggling
Input Register Clocking	On every posedge clk	Only when enable is high
Area Overhead	Low (no gating logic)	Slightly increased (gating circuit added)
Complexity	Simple	Moderate due to additional gating logic
Suitability for Low Power	Poor	Excellent

Conclusion

The project successfully demonstrates the implementation of a 4-bit ALU with low-power optimization through clock gating. While the unoptimized design continuously consumes power regardless of control signals, the optimized version activates the ALU logic only when required. This approach significantly reduces dynamic power usage, making the design ideal for modern low-power embedded systems.

Clock gating proves to be an effective and practical technique in digital design for energy-efficient operation. Future enhancements could include support for more operations, wider data paths, and integration into larger processor systems.

Thank You