ECE 128 Final Project: **Calculator**

Alex Hume and Frank Tamburro

Objective

- Design and implement a custom 4-bit calculator on the FPGA board that supports 8 operations:
  - Arithmetic: add, subtract, multiply, and divide
  - Logical: AND, OR, XOR, NOT
- Design the system using a modular hierarchy in Verilog including calculator logic (ALU), binary to BCD converter, and a seven-segment driver to produce real-time output
- Develop a top level system that connects the modules into one functional design
- Create a testbench to simulate and verify the design for all calculator operations and seven-segment display values
- Demonstrate a functional calculator on the FPGA board through hardware testing

Motivation

- Apply and utilize concepts from coursework to implement our calculator design
  - Ex. Combinational multiplier, seven-segment driver, 4-bit ripple carry adder
- Bridge theory and practice by applying digital logic concepts in designing our own modules for specific applications
- Understand how calculators (and processors) execute arithmetic and logical operations
- Strengthen skills in Verilog, testbench creation, and debugging
- Build a practical project that demonstrates proficiency in FPGA implementation and design for future projects, interviews, and much more!

Background

- The calculator operates on two 4-bit inputs (A and B), allowing representation of values from 0 to 15 for each operand
- The 3-bit opcode input designates the operation for the design to calculate, mimicking the hardware of a real calculator
- To produce a readable decimal output on the FPGA board, the results from the calculator must be converted to BCD and driven to the seven-segment display
- The ALU is entirely combinational logic, while the BCD converter and seven-segment display include sequential logic that require multiple CCs for a stable output
- Divider module uses repeated subtraction, where the divisor is subtracted from the dividend in a loop
- The subtractor module relies on a ripple carry adder

Design Methodology

- Use ALU module to contain on calculator logic
  - Arithmetic operations connected to submodules for each operation
  - Use built-in bitwise operators for logical operations
  - Ensure all outputs from ALU are 8-bits (concatenation)
- Design top module to pass I/O from ALU to BCD converter to seven-segment display driver
  - Use enable and ready signals between sequential blocks to ensure timing alignment
- Top module:
  - ALU
    - Combinational Multiplier
    - FourBitSubRippleCarry
      - FourBitFARippleCarry → OneBitFA
    - FourBitFARippleCarry
      - OneBitFA
    - FourBitDivider
  - Bin12to16BCD
  - Multi Segment Driver:
    - Anode Generator
    - BCD to Seven Segment Display

Top Module

```
module Top(clk,A,B,opcode, anode, cathode);
input clk;
input [3:0] A,B;
input [2:0] opcode;
output wire [3:0] anode;
output wire [6:0] cathode;

wire [7:0] result2;
ALU uut1 (.A(A),.B(B),.opcode(opcode),.result(result2));
wire [11:0] result3;
assign result3 = {4'b0000,result2};

reg en1=1'b1;
wire [15:0] bcd_out;
wire rdy;
reg [15:0] bcd_out_ready = 16'b0;
Bin12to16BCD uut2(.clk(clk), .en(en1), .bin_d_in(result3), .bcd_d_out(bcd_out), .rdy(rdy));
multisegDriver uut3 (.clk(clk),.bcd_in(bcd_out_ready),.seg_anode(anode),.seg_cathode(cathode));

always @(posedge clk)
    begin
        if(rdy)
            begin
                bcd_out_ready <= bcd_out;
            end
    end
endmodule
```

ALU Module

```
module ALU(A,B,opcode,result);
input [3:0] A,B;
input [2:0] opcode;
output reg [7:0] result;

wire [7:0] product; //wire to connect output of mult
//combinational multiplier instantiation
CombinationalMult uut1 (.a(A),.b(B),.product(product));
//4-bit adder
wire [3:0] partial_sum;
wire c_out;
FourBitFARippleCarry uut2 (.A1(A),.B1(B),.CI(1'b0),.S1(partial_sum),.CO(c_out));
wire [7:0] sum;
assign sum = {3'b000, c_out, partial_sum};
//4-bit subtractor
wire[3:0] partial_diff;
wire c_out2;
FourBitSubRipple uut3(.A1(A),.B1(B),.Diff(partial_diff),.No_Borrow(c_out2));
wire [7:0] diff;
assign diff = {4'b0000, partial_diff};
//divider
wire [3:0] partial_Q;
FourBitDivider uut4(.A1(A),.B1(B), .Q(partial_Q));
wire [7:0] Q;
assign Q = {4'b0000, partial_Q};
```

```
always @(*)
    begin
    case(opcode)

        3'b000: result = sum; //add
        3'b001: result = diff; //subtract
        3'b010: result = product; //multiply
        3'b011: result = Q; //divide
        3'b100: result = A & B; //and
        3'b101: result = A | B; //or
        3'b110: result = A^B; //xor
        3'b111: result = {4'b0000,~A}; //not

        endcase
    end
endmodule
```

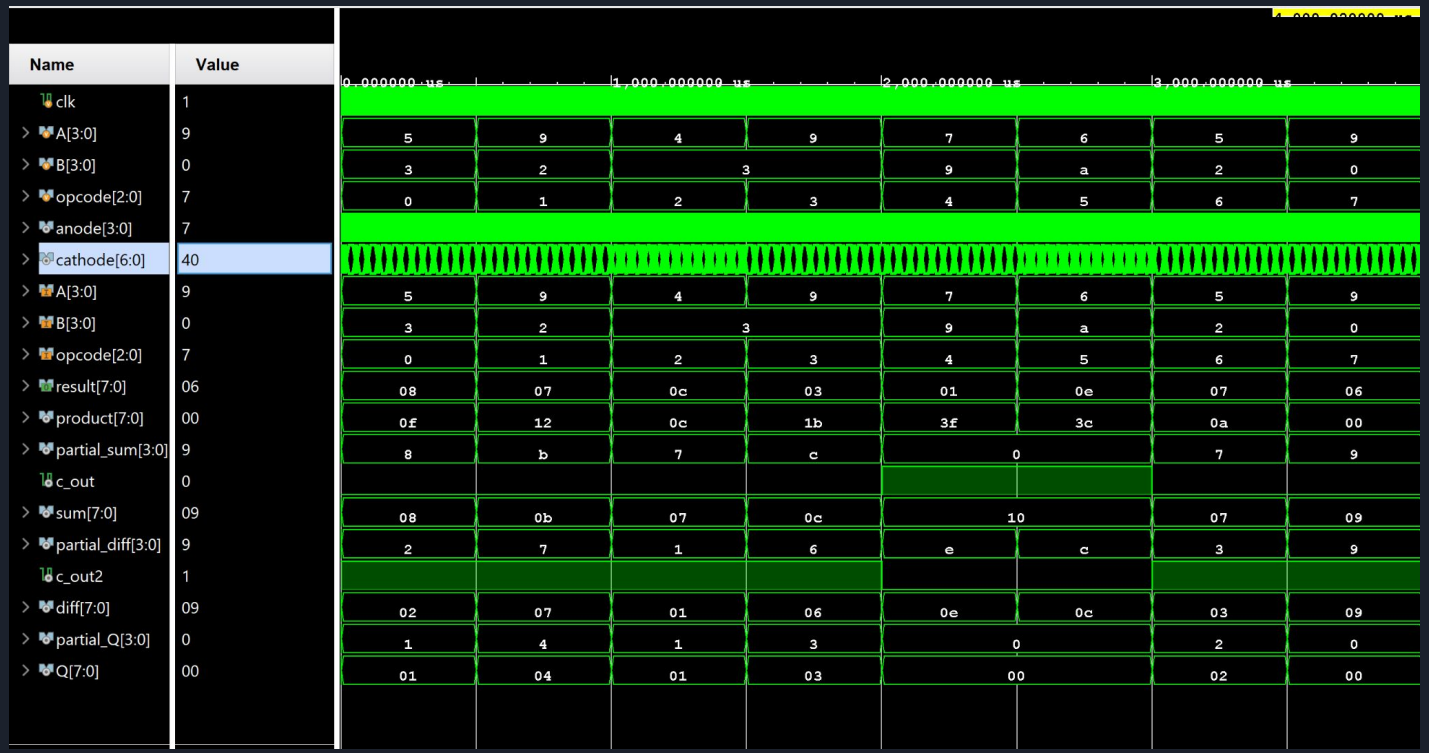
Results and Analysis

Simulation Waveform

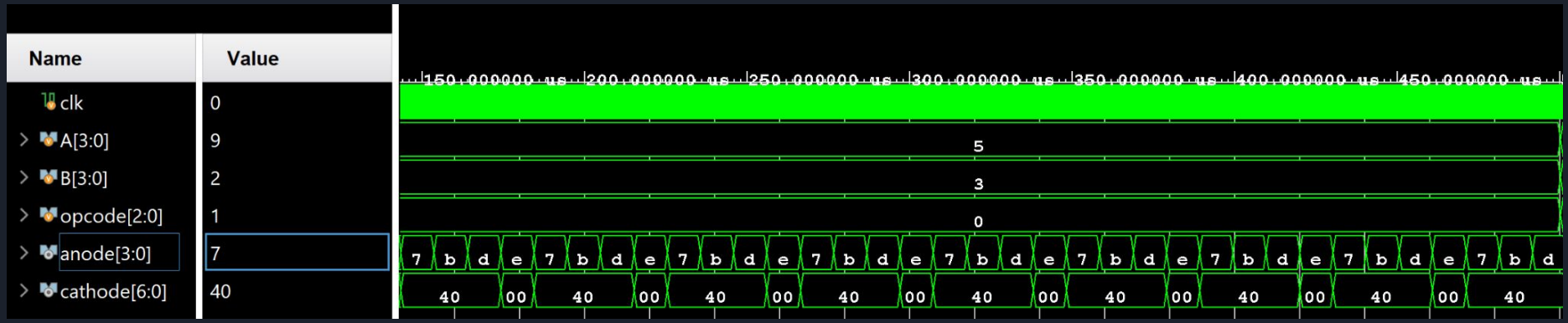
- ALU operations produced results within the same cycle, independent of system clock (Combinational logic)
- BCD conversion and seven-segment driving required multiple clock cycles to produce anode and cathode outputs
    - Produced stable, repeatable outputs
- Testbench tested each opcode value (operation type) with delays appropriate for seven-segment driver outputs

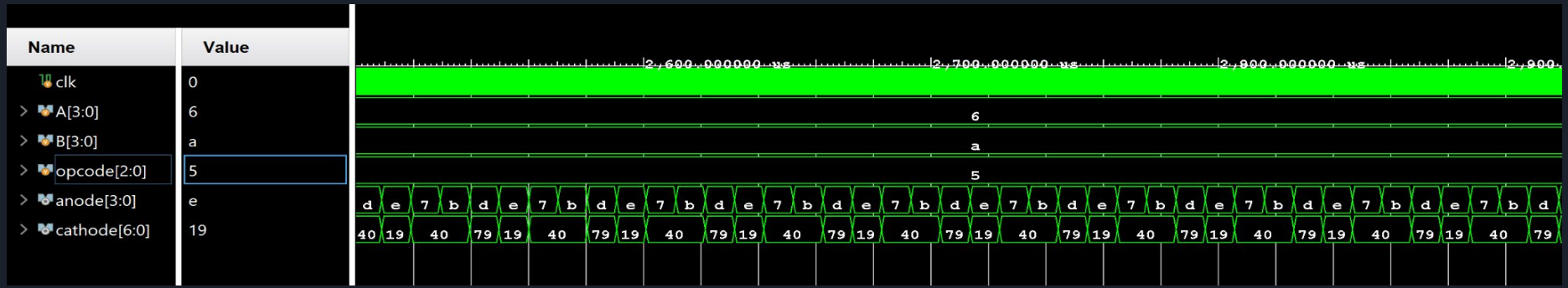
```
initial begin

        A = 0;
        B = 0;
        opcode = 0;
        #20;

        //add
        A = 4'd5;
        B = 4'd3;
        opcode = 3'b000;
        #500000;
        //subtract
        A = 4'd9;
        B = 4'd2;
        opcode = 3'b001;
        #500000;
        //multiply
        A = 4'd4;
        B = 4'd3;
        opcode = 3'b010;
        #500000;
        //divide
        A = 4'd9;
        B = 4'd3;
        opcode = 3'b011;
        #500000;
        //and
        A = 4'd7;
        B = 4'd9;
        opcode = 3'b100;
        #500000;
```

Simulation Waveform (with ALU signals)



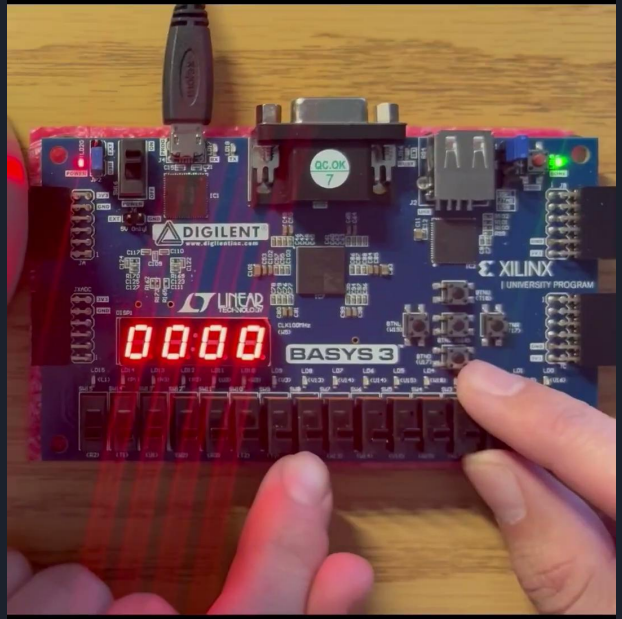
Simulation Waveform (Top module only)





Recorded Demonstration

- Each operation tested with two different sets of inputs
- 4-bit input A (SW[0] - SW[3])
- 4-bit input B (SW[4] - SW[7])
- 3-bit input opcode (SW[8] - SW[10])
- All outputs appear on seven-segment display as decimal
    - Downside for logical operations
- Outputs remained stable when inputs switched at runtime



Report Utilization

Name 1	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)	BUFGCTRL (32)
N Top	121	78	46	121	23	1
uut1 (ALU)	10	0	6	10	0	0
uut2 (Bin12to16BCD)	92	40	37	92	0	0
uut3 (multisegDriver)	20	22	9	20	0	0

- Design only used small fraction of FPGA resources - effective design
  - Used 0.58% LUT, 0.19% FF, and 21.7% IO
- Implementation highlights plenty of room for future expansion
  - Additional calculator operations

Conclusion

- Successfully designed and implemented a fully functional 4-bit calculator on the FPGA board, supporting a total 8 arithmetic and logical operations
- Built a modular hierarchical design in Verilog to produce a cohesive hardware design
  - Opportunities for scalability
- Verified accuracy of design through simulation and a Testbench with multiple input cases
  - Improved understanding of how combinational and sequential logic interact
- Demonstrated real-time functionality with accurate outputs when implemented to FPGA board
- Integrated core digital design concepts from throughout the course, and highlighted the complete engineering design process
  - Designed a service that can provide real-world practicality