..

University of Tehran

Faculty of Engineering

School of ECE

CAD

CA1 - report sheet

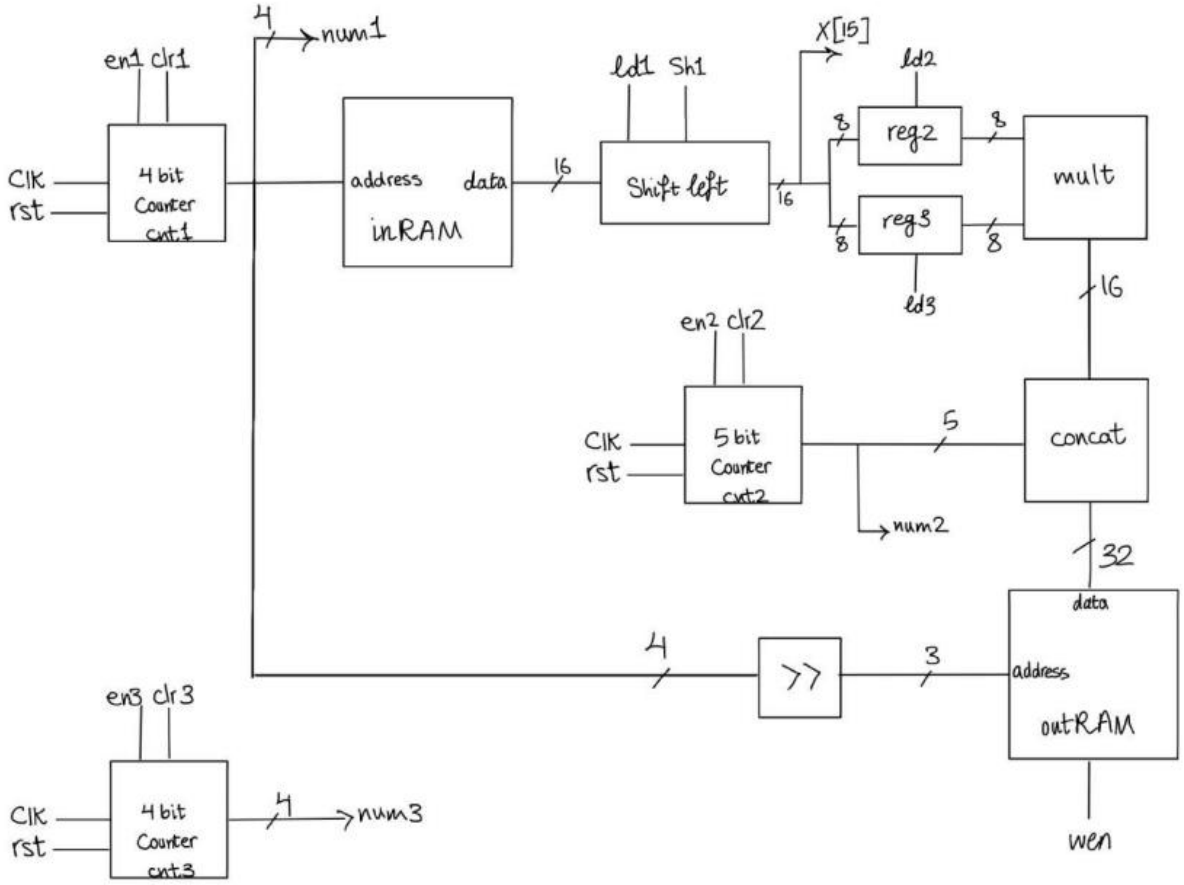
Group project

Niloufar Mortazavi, SID: 220701096

Alborz Mahmoudian, SID: 810101514

Datapath Details

Starting with the datapath design, we have designed the diagram below. All components Verilog codes are available (modules.v).



The input RAM is connected to a 4-bit counter that points to the RAM addresses and would count up to point to the next address in each cycle.

```
module inRAM(input [3:0] address,
             output [15:0] dataout);

        reg [15:0] RAM [0:15];
        initial $readmemb ("data_input.txt" , RAM);
        assign dataout = RAM[address];

endmodule
```

```
module counter #(parameter n)(input clk, rst, clear, enable,
                              output reg [n - 1:0] num);

        always@(posedge clk, posedge rst)begin
                if(rst)
                        num <= {n * 1'b0};
                else if(clear)
                        num <= {n * 1'b0};
                else if(enable)
                        num <= num + 1;
        end
endmodule
```

*We have used this counter module for all counters used as it is written parametrical.

Each data coming from the RAM would then be loaded to a shift-left register that would shift the 16-bit input data if the MSB is 0. This shifting will be repeated till the MSB is 1, then the control signal X[15] would be 1 and the shifting would be stopped.

```
module ShiftLReg2(input clk, rst, ld, shL,
                  input [15:0] datain,
                  output reg[15:0] dataout);

        always@(posedge clk , posedge rst)begin
                  if(rst)
                          dataout <= 16'b0;
                  else if(ld)
                          dataout <= datain;
                  else if(shL)
                          dataout <= {dataout[14:0] , 1'b0};
        end
endmodule
```

The fisrt number now would be loaded to the register number 2 as a multiplicand and then the RAM counter would count up and point to the next position in RAM and all the above steps would be applied to the next number in RAM and the shifted number will be loaded to register number 3 as multiplier.

```
module register (input clk, rst, ld,
                          input [7:0] datain,
                          output reg [7:0] dataout);

        always@(posedge clk, posedge rst) begin
                  if(rst)
                          dataout <= {8'b0};
                  else if(ld)
                          dataout <= datain;
        end
endmodule

register reg2 (clk, rst, ld2, shLout[15:8], reg2out);
register reg3 (clk, rst, ld3, shLout[15:8], reg3out);
```

Now we have the two 8-bit numbers and we calculate the multiplying output.

```
module mult(input [7:0] A, B,
                output [15:0] multout);

        assign multout = A * B;

endmodule
```

Now here we have a 5-bit counter that has counted how many times our two numbers has been shifted overall (that means the number of 0s on the left of both first and second number).now instead of putting x zeroes on the left side of the final output, we shift the multiplier result (16-x) times and the final output would be zero-extended (worthless bits) by itself as being stored as 32-bit datas . These steps are accomplished in the module below.

```
module concat2 (input [4:0] num2,
                input [15:0] multresult,
                output [31:0] dataout);

        assign dataout = multresult << (16 - num2);
endmodule
```

Now here we had to prevent each number from being shifted more than 8 time, so we used a 4-bit counter that counts to 8 and gets reset each time we load a new number in the shift register.

Then we have the final result and should store it in a 8*32 output RAM. Here is the module for it:

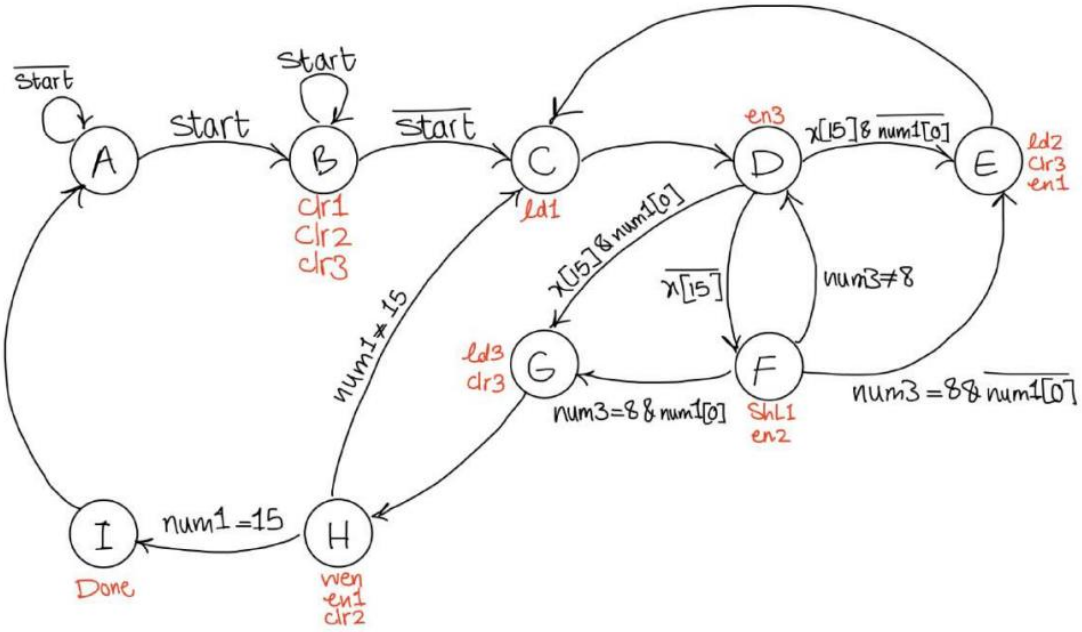
```
module outRAM(input clk, rst, wen,
                input [2:0] address,
                input [31:0] datain);

        reg [31:0] RAM [0:7];
        initial $readmemb ("data_output.txt" , RAM);

        always@(posedge clk) begin
                if(wen)
                        RAM[address] <= datain;
        end
endmodule
```

For the addresses we used the first RAM addresses and shifted by 1 bit to the right to convert the 16 row of addresses to 8 rows.

Pay attention that we did not calculate the number of zeroes(x) that should be on the left side of the final output, because we shifted the approximate answer (16-x) times and the final output would be zero-extended (worthless bits) by itself as being stored as 32-bit datas.

Controller Details

As we get the start signal we go to next state to initialize the calculation. We clear the counters in state B and as the signal start go of we go to state C and load the first number out of the RAM and in the shift register.



From there we go directly to state D and shift upon the zeroes until X[15] be 1. When that happen the path goes to calculation states and load the second number.

The as we mentioned before, the approximate multiplication would be done and stored in output RAM in state H when the signal wen activates.

At the end in the state I all 8, 32-bit numbers would be stored completely and signal Done is activated.

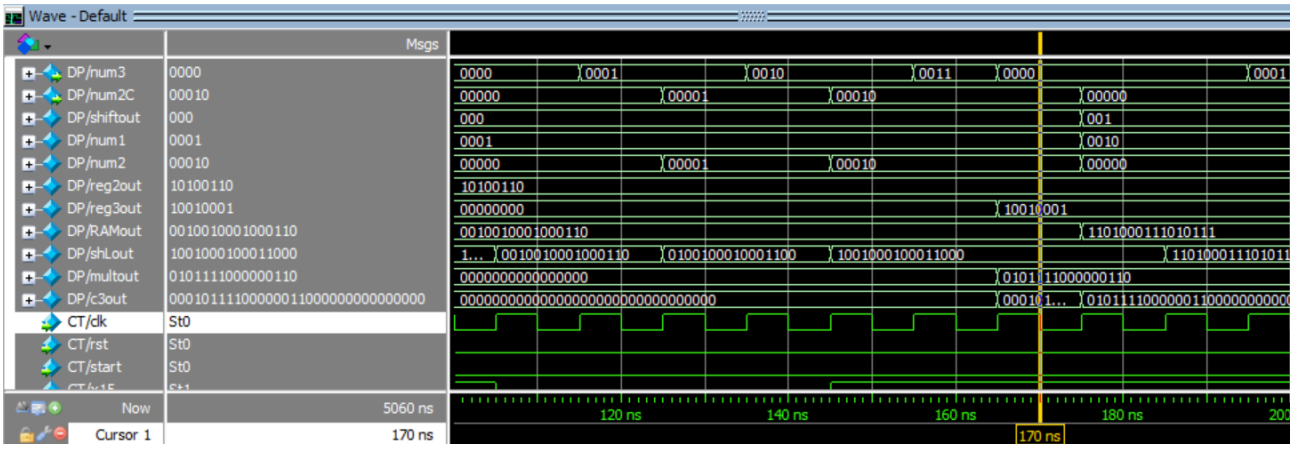
Test Details

The primary goal of this testbench is to:

1. Clock Signal Generation: A 10ns clock cycle is simulated, ensuring that the TopLevel module functions with a regular clock signal.
2. Reset Signal: The testbench initially holds the system in reset for 15ns, then de-asserts it to begin normal operation.
3. Start Signal: After reset, the start signal is activated to initiate the module's operation.
4. Observation: The simulation monitors the done signal to confirm that the TopLevel module completes its tasks.

The simulation runs for a total of 5000 nanoseconds before halting using the $stope task. This setup allows for basic verification of the system's initialization, operation, and completion.

By checking the waveform it is obvious that in each tuple multiplication we will get the correct answer by this code.



Also by looking at the output file and compare it to the output report provided in the project, we can claim that test cases all came true.