CSE433

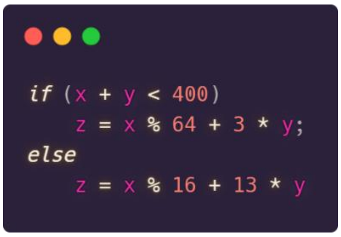
Embedded Systems

Assignment 1 Report

OĞUZHAN AGKUŞ
161044003

Objective

Design a circuit which will take two unsigned 8-bit numbers x and y as inputs and perform following C code in Verilog:



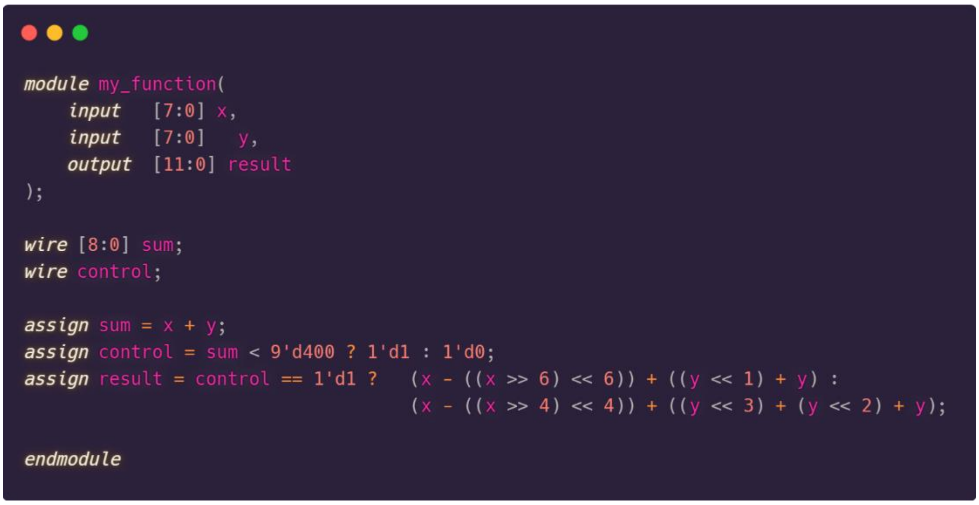
```
if (x + y < 400)
    z = x % 64 + 3 * y;
else
    z = x % 16 + 13 * y
```

Restrictions

- Use structural or dataflow model Verilog, do not use behavioral Verilog
- Do not use any sequential component like registers and flip-flops
- Do not use any multiplier or * sign
- Do not use any divisor or / sign

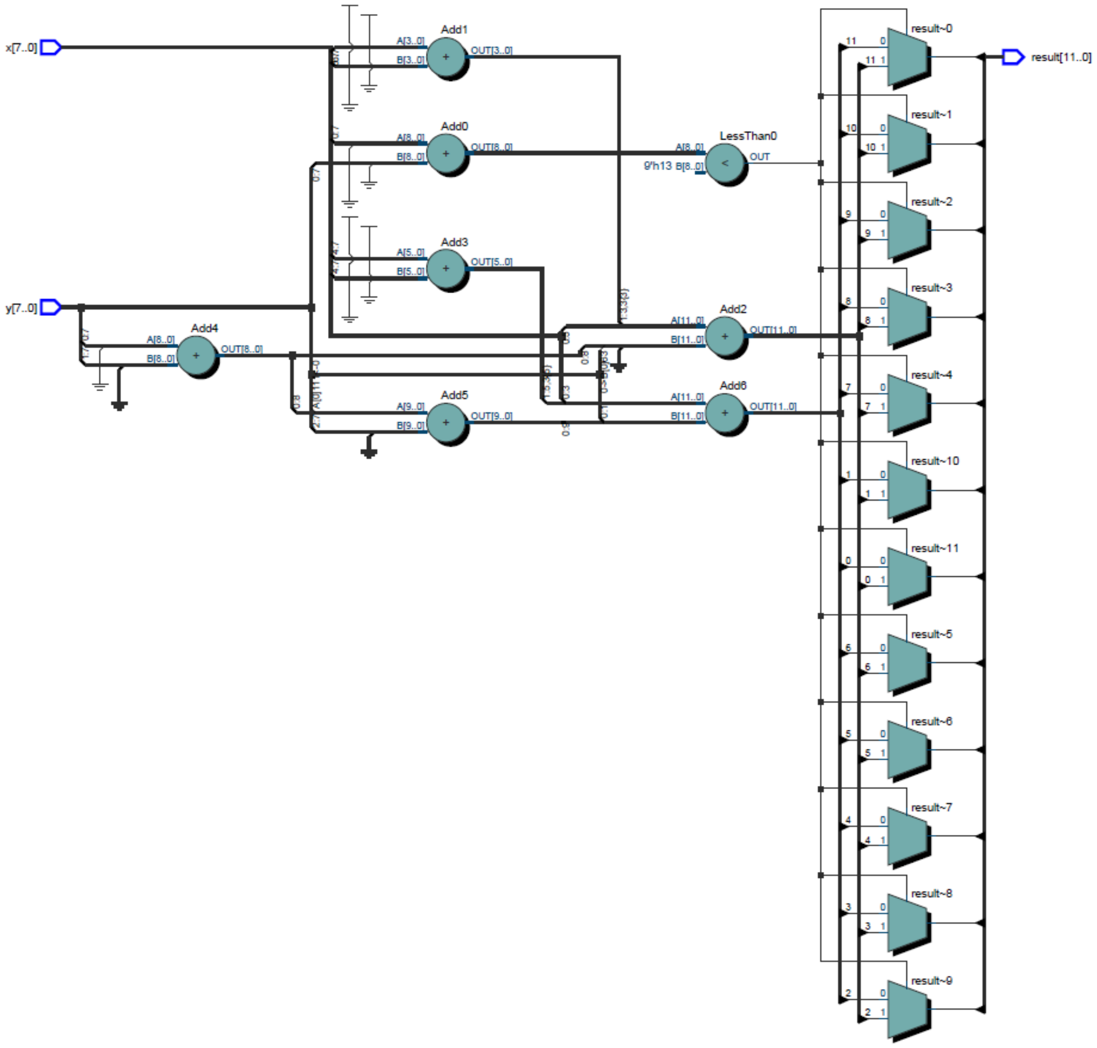
Design Decisions

- I choose to use dataflow model of Verilog. So, I used "assign" statements.
- Since using * or / is not allowed, I used bitwise shift operators. So, they can easily multiply and divide a number by 2 and its powers.
- To perform modulo operation, I combined shift operators and addition (subtraction) operator. For example, to find x % 64 equals (x - ((x >> 6) << 6)). Firstly, shift right 6 bits (6 comes from $\log_2 64$) to divide by 64. Then shift right 6 bits to multiple by 64. Finally, get diffidence of input and result, it will give use the remainder.
- Multiplying by powers of 2 is easy but other power are must be combined from powers of 2. For example, 13 * y = 8 * y + 4 * y + 1 * y = (y << 3) + (y << 2) + y
- By examining given equation, I decided to output should be 12-bit. Biggest result can derive from else condition. Since inputs are 8-bit, y can be 255 at most. So, 13 * 255 = 3315, which can be show as 12 bits.
- I also needed two temporary data. First one is, sum which represents sum of x + y. It is 9-bit because sum of two 8-bit variable can be bigger than 8-bit. Other one is control bit which represents sum is smaller from 400 or not.



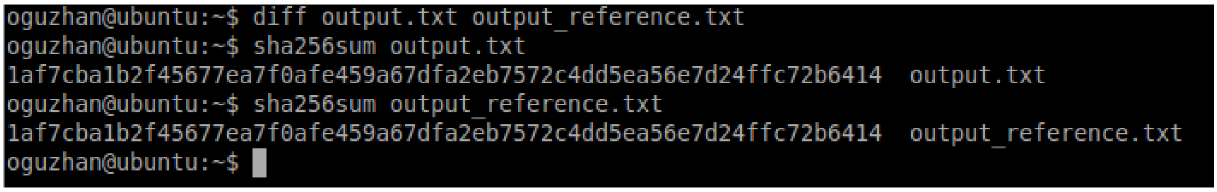
```
module my_function(
    input   [7:0] x,
    input   [7:0]   y,
    output  [11:0] result
);

wire [8:0] sum;
wire control;

assign sum = x + y;
assign control = sum < 9'd400 ? 1'd1 : 1'd0;
assign result = control == 1'd1 ?   (x - ((x >> 6) << 6)) + ((y << 1) + y) :
                                    (x - ((x >> 4) << 4)) + ((y << 3) + (y << 2) + y);

endmodule
```

Semantics

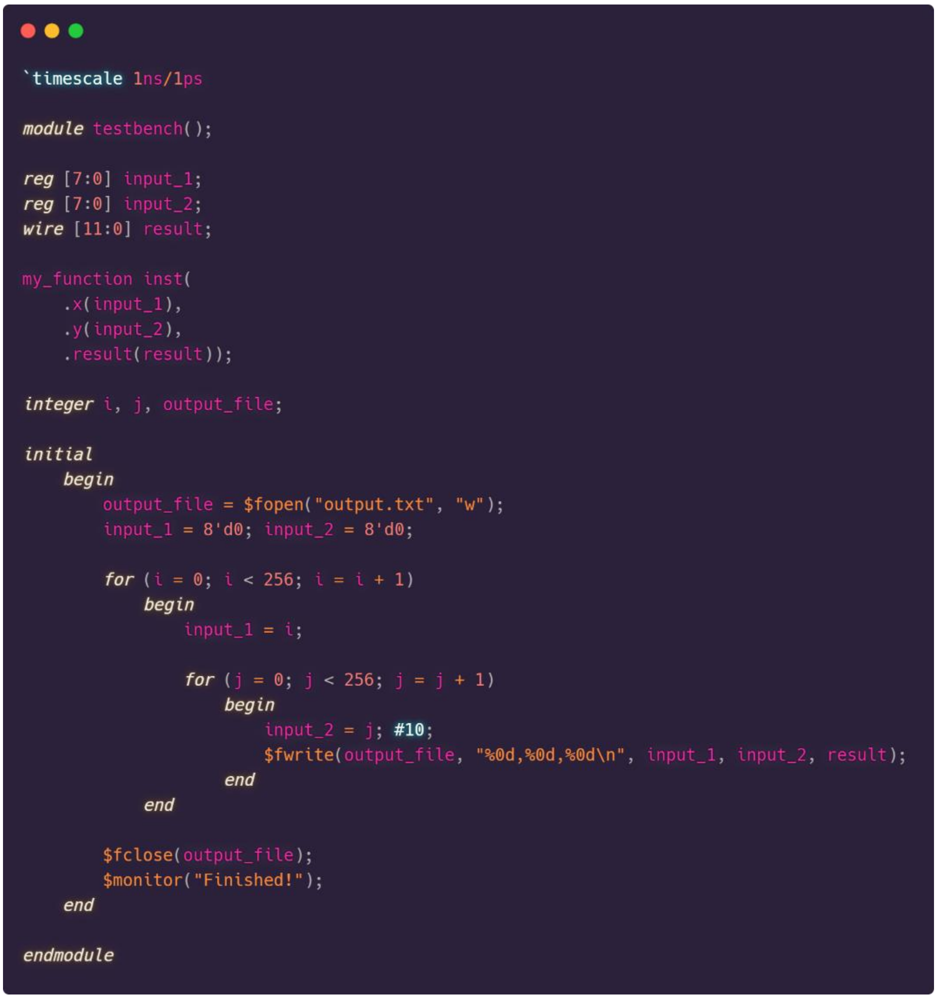


Testbench and Results

- I have written a testbench which tries each possible input.
- Each input is 8-bit, so it can get 256 different values. Since we have 2 inputs, we have totally 256*256 = 65536 possible situations.
- It is hard to follow each of them one by one. So, I decide to print all output to the file and check it automatically.
- I write a Python script. It also calculates and prints the results of all combinations. Then I check both output files with extra tools. I used diff command on Linux and compared hashes of both files. Output is here

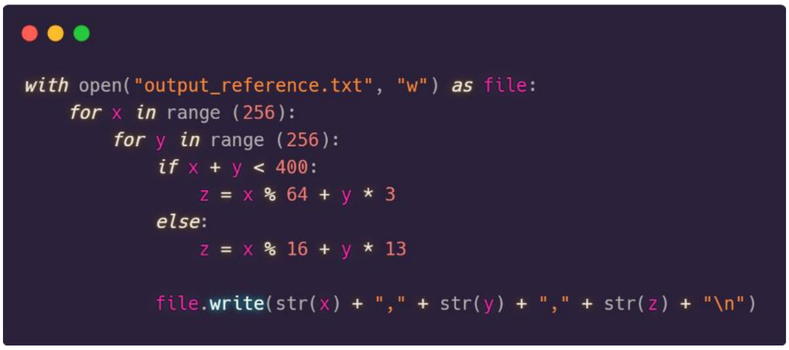


```
oguzhan@ubuntu:~$ diff output.txt output_reference.txt
oguzhan@ubuntu:~$ sha256sum output.txt
1af7cba1b2f45677ea7f0afe459a67dfa2eb7572c4dd5ea56e7d24ffc72b6414  output.txt
oguzhan@ubuntu:~$ sha256sum output_reference.txt
1af7cba1b2f45677ea7f0afe459a67dfa2eb7572c4dd5ea56e7d24ffc72b6414  output_reference.txt
oguzhan@ubuntu:~$ 
```



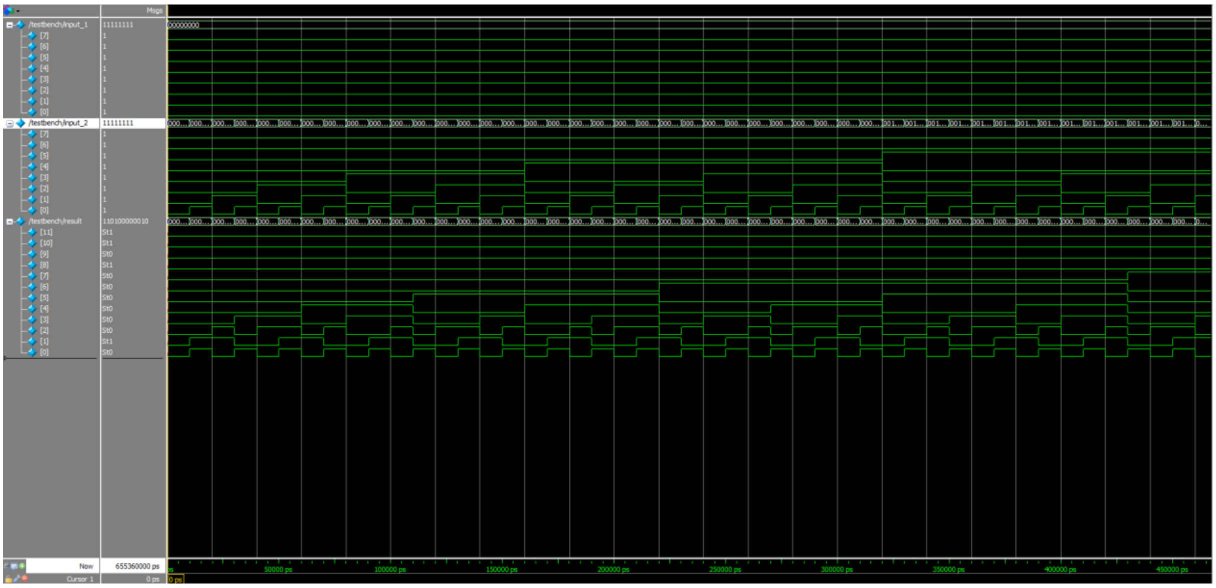
```verilog
`timescale 1ns/1ps

module testbench();

reg [7:0] input_1;
reg [7:0] input_2;
wire [11:0] result;

my_function inst(
    .x(input_1),
    .y(input_2),
    .result(result));

integer i, j, output_file;

initial
    begin
        output_file = $fopen("output.txt", "w");
        input_1 = 8'd0; input_2 = 8'd0;

        for (i = 0; i < 256; i = i + 1)
            begin
                input_1 = i;

                for (j = 0; j < 256; j = j + 1)
                    begin
                        input_2 = j; #10;
                        $fwrite(output_file, "%0d,%0d,%0d\n", input_1, input_2, result);
                    end
            end

        $fclose(output_file);
        $monitor("Finished!");
    end

endmodule
```

Testbench file

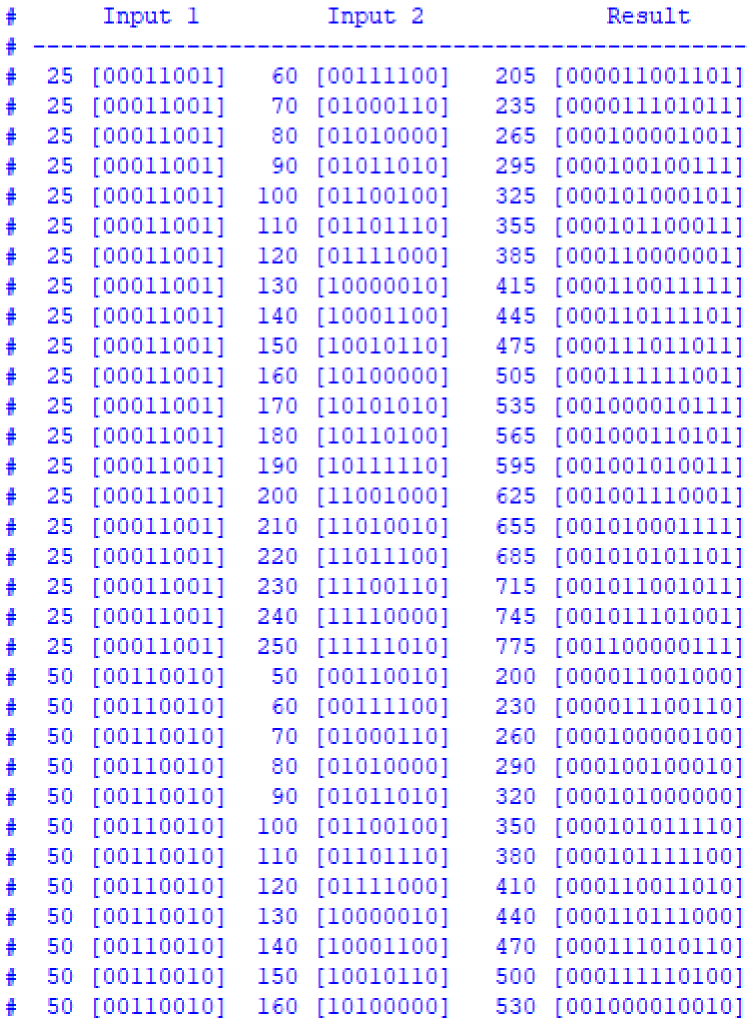


```python
with open("output_reference.txt", "w") as file:
    for x in range (256):
        for y in range (256):
            if x + y < 400:
                z = x % 64 + y * 3
            else:
                z = x % 16 + y * 13

            file.write(str(x) + "," + str(y) + "," + str(z) + "\n")
```

Python script



Simulation screenshot



```
#     Input 1          Input 2            Result
# ------------------------------------------------
#  25 [00011001]    60 [00111100]    205 [000011001101]
#  25 [00011001]    70 [01000110]    235 [000011101011]
#  25 [00011001]    80 [01010000]    265 [000100001001]
#  25 [00011001]    90 [01011010]    295 [000100100111]
#  25 [00011001]   100 [01100100]    325 [000101000101]
#  25 [00011001]   110 [01101110]    355 [000101100011]
#  25 [00011001]   120 [01111000]    385 [000110000001]
#  25 [00011001]   130 [10000010]    415 [000110011111]
#  25 [00011001]   140 [10001100]    445 [000110111101]
#  25 [00011001]   150 [10010110]    475 [000111011011]
#  25 [00011001]   160 [10100000]    505 [000111111001]
#  25 [00011001]   170 [10101010]    535 [001000010111]
#  25 [00011001]   180 [10110100]    565 [001000110101]
#  25 [00011001]   190 [10111110]    595 [001001010011]
#  25 [00011001]   200 [11001000]    625 [001001110001]
#  25 [00011001]   210 [11010010]    655 [001010001111]
#  25 [00011001]   220 [11011100]    685 [001010101101]
#  25 [00011001]   230 [11100110]    715 [001011001011]
#  25 [00011001]   240 [11110000]    745 [001011101001]
#  25 [00011001]   250 [11111010]    775 [001100000111]
#  50 [00110010]    50 [00110010]    200 [000011001000]
#  50 [00110010]    60 [00111100]    230 [000011100110]
#  50 [00110010]    70 [01000110]    260 [000100000100]
#  50 [00110010]    80 [01010000]    290 [000100100010]
#  50 [00110010]    90 [01011010]    320 [000101000000]
#  50 [00110010]   100 [01100100]    350 [000101011110]
#  50 [00110010]   110 [01101110]    380 [000101111100]
#  50 [00110010]   120 [01111000]    410 [000110011010]
#  50 [00110010]   130 [10000010]    440 [000110111000]
#  50 [00110010]   140 [10001100]    470 [000111010110]
#  50 [00110010]   150 [10010110]    500 [000111110100]
#  50 [00110010]   160 [10100000]    530 [001000010010]
```

Terminal output of simulation