# Lab 1: Sequential Multiplier

Lan-Da Van and Chun-Jen Tsai
Department of Computer Science
National Yang Ming Chiao Tung University
Taiwan, R.O.C.

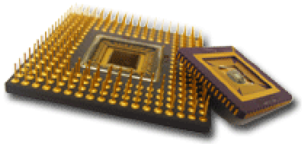
*Fall, 2023*

# Lab 1 Goal: Simulate an 8-bit Multiplier



- In this lab, you must simulate the operations of a sequential binary multiplier using the Vivado Simulator.
  - You should review your textbook on Digital Circuit Design by Mano. Some design guideline of the sequential binary multiplier is in Section 8.10 of Mano's book.
  - The multiplier is designed using only adder, shifter, multiplexor, and gate-level operators. You cannot use the multiplication operator of Verilog.

- The lab file submission deadline is on 9/25 by 6:00pm.



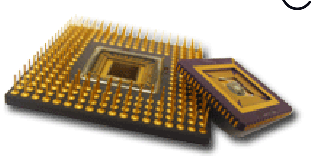
# Write Simulation for a Multiplier



- The input/output ports of the 8-bit multiplier is as follows:

```
module SeqMultiplier(
    input wire clk,
    input wire enable,
    input wire [7:0] A,
    input wire [7:0] B,
    output wire [15:0] C
    );
```

'`clk`' is the system clock,
'`enable`' activates the multiplication operation,
'`A`' is the 8-bit unsigned multiplicand input,
'`B`' is the 8-bit unsigned multiplier input, and
'`C`' is the 16-bit unsigned product output.



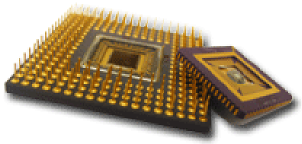
# Sequential Binary Multiplier Behavior



- Sequential binary multiplication of 10111 and 10011:

$23 \times 19 = (437)_{10}$

```
        00010111   → multiplicand
×       00010011   → multiplier
-------------------
        00000000
         00000000
          00000000
           00010111
            00000000
             00000000
              00010111
+              00010111
-------------------
           000110110101   → product
```

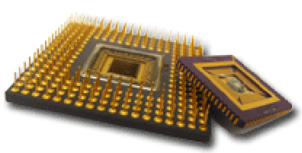


# C Model of the Multiplier



- The C code that performs the sequential multiplication:

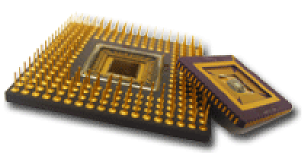
```
typedef unsigned char Byte;
typedef unsigned short Word;

SeqMultiplier(Byte A, Byte B, Word *C)
{
    int idx;

    *C = 0;
    for (idx = 8; idx != 0; idx--)
    {
        *C = *C << 1;
        if ((B & 0x80) == 0x80)
        {
            *C += A;
        }
        B = B << 1;
    }
}
```



# Verilog Module of the Multiplier



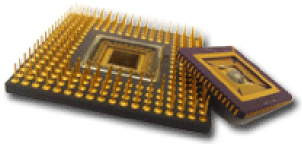
```verilog
module SeqMultiplier(input wire clk, input wire enable,
    input wire [7:0] A, input wire [7:0] B,
    output wire [15:0] C);

reg  [15:0] prod;
reg  [7:0]  mult;
reg  [3:0]  counter;
wire        shift;

assign C = prod;
assign shift = |(counter^7);

always @(posedge clk) begin
  if (!enable) begin
    mult <= B;
    prod <= 0;
    counter <= 0;
  end
  else begin
    mult <= mult << 1;
    prod <= (prod + (A & {8{mult[7]}})) << shift;
    counter <= counter + shift;
  end
end

endmodule
```



# Verilog Module of the Multiplier with Comment (1/2)



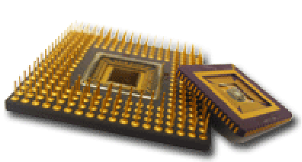
```
module SeqMultiplier(
                 input wire clk,
                 input wire enable,
                 input wire [7:0] A,
                 input wire [7:0] B,
                 output wire [15:0] C
        );
reg [15:0] prod;
reg [7:0] mult;
reg [3:0] counter;
wire shift;
assign C = prod;
assign shift = |(counter^7); //1 if counter<7; 0 if counter==7
/*
Every bit of counter exclusive or 7(4'b0111), and or together
ex: When counter = 4'd2 = 4'b0010, then counter^7 = 4'b0101, shift = 0|1|0|1 = 1
That is, only when counter^7 == 4'b0000 --> counter == 7 will shift be 0.
*/
```



# Verilog Module of the Multiplier with Comment (2/2)



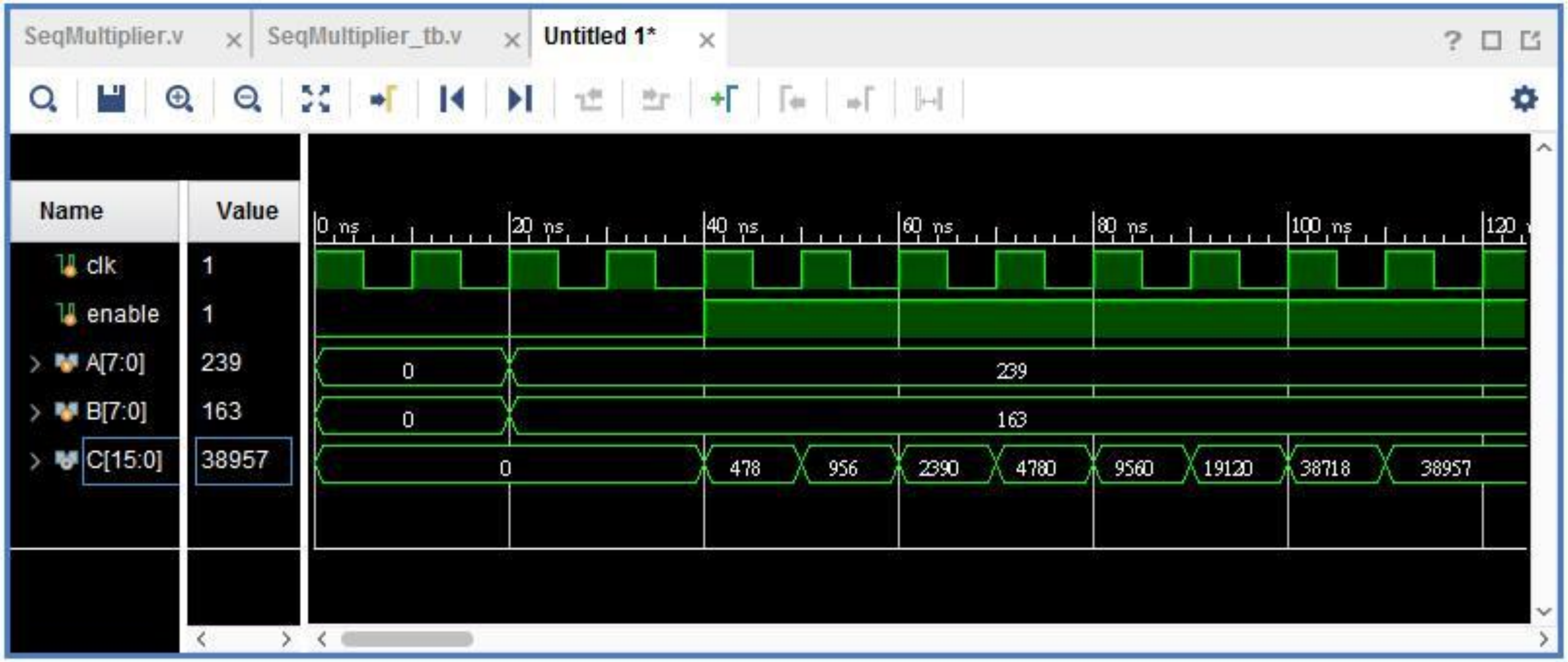
```
always @(posedge clk) begin
        if (!enable) begin //Reset
                mult <= B; //We will change the value of it, so we put it in another register.
                prod <= 0;
                counter <= 0;
        end
        else begin
                mult <= mult << 1; //shift left
                prod <= (prod + (A & {8{mult[7]}})) << shift;
/*
Replication, {8{mult[7]}} == {mult[7], mult[7], mult[7], mult[7], mult[7], mult[7], mult[7], mult[7]}
Every bit of A will '&'(bit-wise operator) the highest bit of mult,
ex: 01000101 & 11111111 = 01000101 / 01000101 & 00000000 = 00000000
add the result to prod(product) and prod then shifts left.
Can take page 31 of lab1's ppt as example:
                            00010111  → multiplicand
                   x        00010011  → multiplier
                ------------------------
                                 00000000   <-- |         counter = 0: prod = 0000000000000000
                                  00000000    . |         counter = 1: prod = 0000000000000000
                                   00000000   . |         counter = 2: prod = 0000000000000000
                                    00010111  . |         counter = 3: prod = 0000000000010111
                                     00000000   |         counter = 4: prod = 0000000000101110
                                      00000000  |         counter = 5: prod = 0000000001011100
                                       00010111 |         counter = 6: prod = 0000000011001111
                   +                    00010111|         counter = 7: prod = 0000000110110101
                            -------------------
                                 0000000110110101| → product
*/
                counter <= counter + shift; //counter+1 when counter < 7
        end
end
endmodule
```



# Example of Simulated Waveforms



- An example of the timing diagram of 239×163 = 38957



# Lab 1 Demo Guide



- You can design one 8X8 sequential multiplier in Verilog. You have to finish the design by one-bit by one-bit in shift and add way.
- You should complete the simulation and show some multiplication result waveforms.
- You should upload your Lab 1 solution to E3 before the deadline.
- During the demo time, TA will ask you to modify the testbench to show different results.
  - You can download your code from E3 during demo.