Lab 2: Matrix Multiplication Simulation

Chun-Jen Tsai and Lan-Da Van

Department of Computer Science

National Yang Ming Chiao Tung University

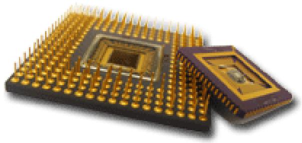
Taiwan, R.O.C.

Fall, 2024

Lab 2: Matrix Multiplication Simulation

- In this lab, you will design a circuit to do 3×3 matrix multiplications on Vivado Simulator.
  - Two register arrays of 3×3 matrices will be given to you in the sample Verilog simulation testbench.
  - You must design a Verilog module to compute their multiplication, and print the result from the testbench.
  - You must use no more than 9 multipliers to implement your circuit.

- The lab file submission deadline is on 09/23 by 6:00pm.



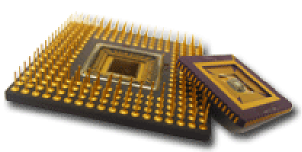
Input Matrix Format

- Each input matrix has 9 unsigned 8-bit elements of values between 0 ~ 255. Matrices A and B are declared in Verilog as follows:

```
reg [0:9*8-1]  A = 72'h_4F_7E_57_0F_14_7B_21_4C_54;
reg [0:9*8-1]  B = 72'h_17_28_3A_40_2F_33_6C_22_77;
```

  - Each matrix is stored in a 72-bit register, and each number in the matrix has 8 bits.
  - The matrix is stored in row-major format.

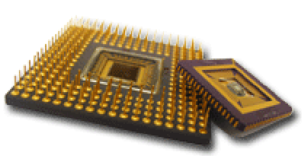
- The output matrix has 9 unsigned 18-bit elements.



Specification of the Multiplier

- The matrix multiplier module is defined as follows:
  - You must follow this declaration to design your matrix multiplication module in order to use the sample simulation testbench.

```
module mmult(
  input   clk,                      // Clock signal.
  input   reset_n,                  // Reset signal (negative logic).
  input   enable,                   // Activation signal for matrix
                                    //   multiplication (tells the circuit
                                    //   that A and B are ready for use).
  input   [0:9*8-1] A_mat,          // A matrix.
  input   [0:9*8-1] B_mat,          // B matrix.

  output valid,                     // Signals that the output is valid
                                    //   to read.
  output reg [0:9*18-1] C_mat // The result of A x B.
);
```

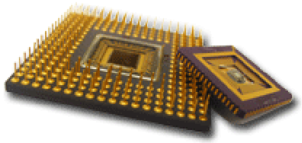


Computation of $A_{3\times3} \times B_{3\times3}$

- A 3×3 matrix multiplication is composed of 9 inner products:

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{pmatrix}$$

- You can compute the outputs in each column of the C matrix in parallel in one clock cycle.
  - At each clock cycle, you use nine multipliers.
  - Three columns of the C matrix takes three cycles to compute!



Testbench of the mmult() Module

- We provide a testbench for you to test the mmult() module.

- The testbench is composed of three parts:
  - Simulation of the clock and reset signals
  - Instantiation of the mmult() module and generation of its input signals
  - Print the output matrix to the console window

Simulation of Cock and Reset Signals

- Digital systems usually requires clock and reset signals.

```
reg clk = 1;       // Clock signal
reg reset_n = 1;   // Reset signal

// 100MHz clock generator
always
  #5 clk = !clk;

// Declare the event
event reset_trigger;

initial begin
  forever begin
// Check whether the event
//"reset_trigger" is triggered or not
    @ (reset_trigger);
    @ (negedge clk);
    reset_n = 0;
    @ (negedge clk);
    reset_n = 1;
  end
end
```

```
// To issue a reset, you must
// trigger a reset event by the
// following code:

initial begin
  // Wait for 10 ns and then trigger
  // the event "reset_trigger".
  #10 -> reset_trigger;
end
```

Instantiation & Invocation of mmult()


```
reg  [0:9*8-1]  A, B;   // 3x3 matrices
wire [0:9*18-1] C;
reg  enable;
wire valid;

// Instiantiates a 3x3 matrix multiplier
mmult uut(
  .clk(clk), .reset_n(reset_n), .enable(enable),
  .A_mat(A), .B_mat(B), .valid(valid), .C_mat(C)
);

initial begin
  // Add stimulus here
  A = 72'h4F_7E_57_0F_14_7B_21_4C_54;
  B = 72'h17_28_3A_40_2F_33_6C_22_77;

  // Wait for 10 ns and then trigger the event “reset_trigger”.
  #10 -> reset_trigger;

  // Wait for 100 ns for global reset to finish
  #100 enable = 1;
end
```

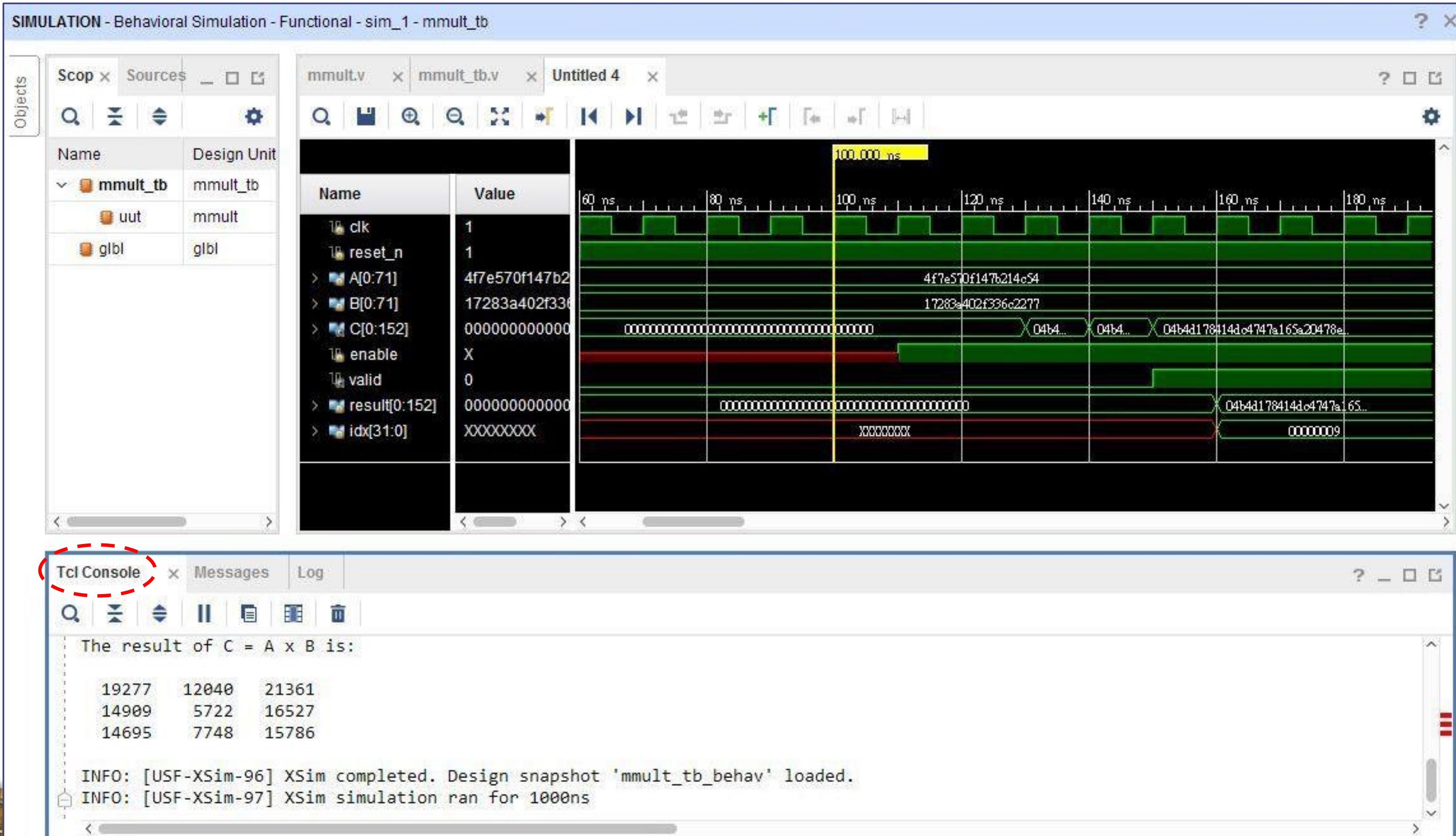
Print the Output Matrix

- In the simulator, you can print the output to console:

```
always @(*) begin // @(*) means all the inputs are included.
  @(posedge valid);

  // Wait one clock cycle so that the output is saved in result[].
  #10 $display("\nThe result of C = A x B is:\n");

  for (idx = 0; idx < 9; idx = idx+1)     // E.g.
  begin                                   // for idx = 0
    $write(" %d ", result[idx*18 +: 18]);// result[0+:18] = result[0:17]
    if (idx%3 == 2) $write("\n");         // for idx = 1
  end                                     // result[18+:18] = result[18:35]
  $write("\n");                           // for idx = 2
End                                       // result[35+:18] = result[36:53]

always @(posedge clk) begin
  if (~reset_n) result <= 0;    // Reset the register “result”
  else if (valid) result <= C; // Write the output “C” into the register
                               // “result”
  else result <= result;       // Keep the value in the register “result”
end
```

The Simulation Output

- The $display() function sends output to "Tcl Console".



Lab 2 Demo Guide

- You can download the sample testbench file mmult_tb.v from E3, and create a Vivado project for it.
- You should upload your lab2 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.