



Lab 7: Max-Pooling and Matrix Multiplication Circuit

Lan-Da Van and Chun-Jen Tsai

Department of Computer Science

National Yang Ming Chiao Tung University

Taiwan, R.O.C.

Fall, 2025

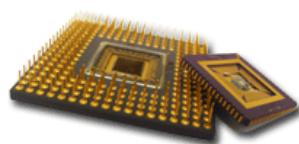


Lab 7: Matrix Calculation

Lab 7

- ◆ In this lab, you will design a circuit to do some matrix operations.
 - Your circuit has a Block RAM (BRAM) that stores two 7×7 matrices.
 - The user presses BTN1 to start the circuit.
 - The circuit reads the matrices, performs the some operations, and prints the output matrix through the UART to terminal window.

- ◆ The lab file submission deadline is on 11/03 by 6:00pm.



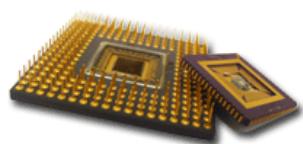


Instantiation of an On-Chip SRAM

Lab 7

- ◆ In this lab, we need to create a single-port static RAM (SRAM) circuit module to store the input matrix.
 - Unlike dynamic RAM (DRAM), an on-chip SRAM can sustain a sequence of random single-cycle read/write requests.
 - Unlike register arrays, a single-port SRAM only outputs one data item per clock cycle.

- ◆ On FPGAs, there are many high speed small memory devices that can be used to synthesize SRAM blocks.
 - On 7th-generation Xilinx FPGA's, there are two devices for SRAM synthesis: distributed RAMs and block RAMs (BRAMs).
 - On Artix-7 35T, there are 313 kbits of distributed RAMs and 50 blocks of 36-kbit BRAMs.

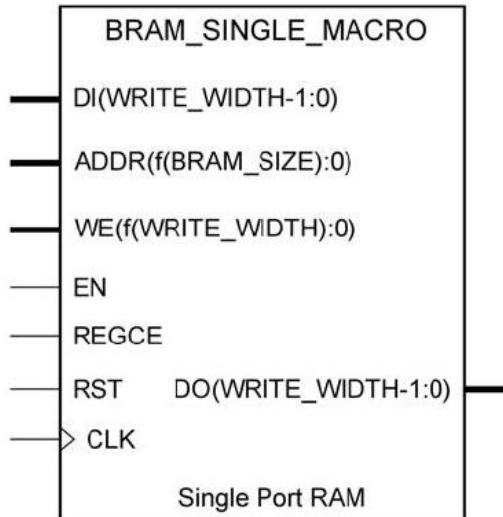




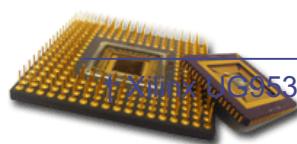
SRAM on FPGAs

Lab 7

- ◆ In Verilog, we can instantiate an SRAM module using explicit declaration[†] or implicit inferencing.
 - For example, a single-port SRAM can be instantiated using the module BRAM_SINGLE_MACRO in Vivado.



Port	Direction	Width	Function
DO	Output	See Configuration Table below.	Data output bus addressed by ADDR.
DI	Input	See Configuration Table below.	Data input bus addressed by ADDR.
ADDR	Input	See Configuration Table below.	Address input bus.
WE	Input	See Configuration Table below.	Byte-Wide Write enable.
EN	Input	1	Write/Read enables.
RST	Input	1	Output registers synchronous reset.
REGCE	Input	1	Output register clock enable input (valid only when DO_REG=1).
CLK	Input	1	Clock input.





General SRAM Signals (1/2)

Lab 7

◆ **CLK** – Clock

- Independent clock pins for synchronous operations

◆ **EN** – Enable

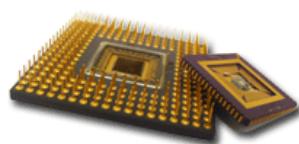
- The read, write and reset functionality of the port is only active when this signal is enabled.

◆ **WE** – Write enable

- When active, the contents of the data input bus are written to the RAM, and the new data also reflects on the data out bus.
- When inactive, a read operation occurs and the contents of the memory cells reflect on the data out bus.

◆ **ADDR** – Address

- The address bus selects the memory cells for read or write.





General SRAM Signals (2/2)

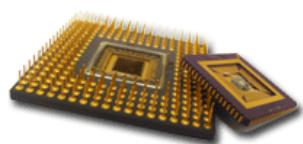
Lab 7

◆ **DIN** – Data input port

- The DI port provides the new data to be written into the RAM.

◆ **DOUT** – Data output port

- The DOUT port reflects the contents of the memory cells referenced by the address bus at the last active clock edge.
- During a write operation, the DOUT port reflects the DIN port.

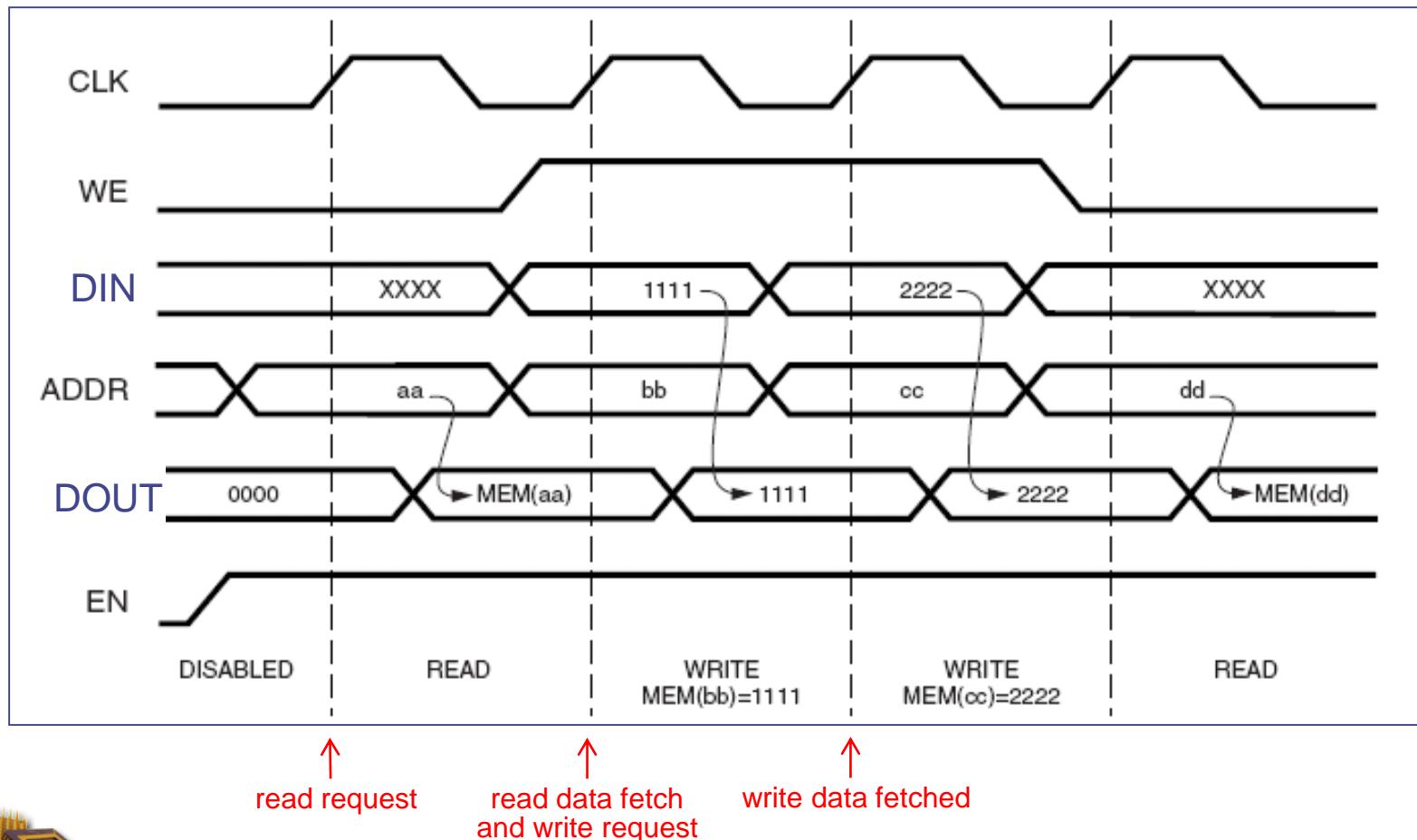




Timing Diagram

Lab 7

- ◆ For single-port SRAM:





Instantiate an SRAM by Inference

Lab 7

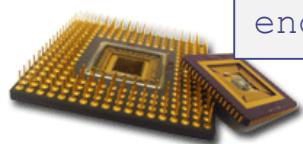
- ◆ The following Verilog code infers an SRAM block:

- The allocation unit size of SRAM on Artix-7s is 18-kbit.
(If you allocate an 8-kbit memory, it will still use an 18-kbit memory block to synthesize it.)

```
reg [7:0] sram[511:0];
wire      sram_we, sram_en;
reg [7:0] data_out;
wire [7:0] data_in;
wire [8:0] sram_addr;

always @(posedge clk) begin // Write data into the SRAM block
    if (sram_en && sram_we) begin
        sram[sram_addr] <= data_in;
    end
end

always @(posedge clk) begin // Read data from the SRAM block
    if (sram_en && sram_we) // If data is being written into SRAM,
        data_out <= data_in; // forward the data to the read port
    else
        data_out <= sram[sram_addr]; // Send data to the read port
end
```





Lab 7 Sample Code (1/3)

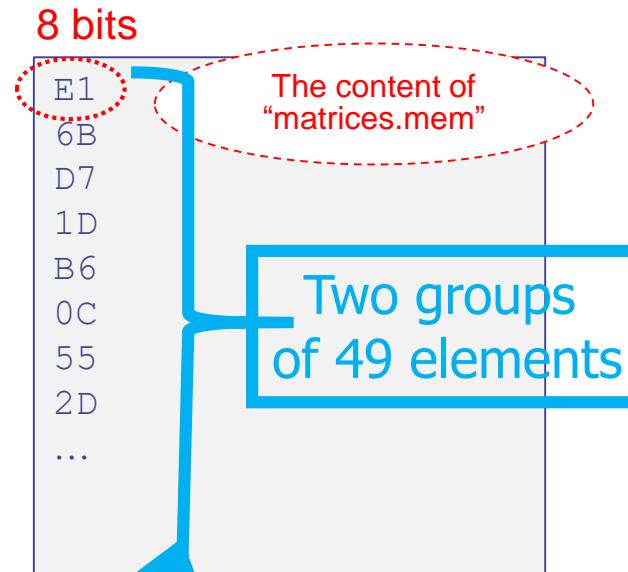
Lab 7

- ◆ The sample code of Lab 7 shows you how to create a SRAM block in FPGA with some data pre-stored in it.
 - The data for the two matrices are pre-stored in SRAM.
 - Initialization of an SRAM block can be done as follows:

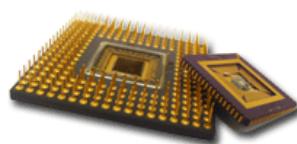
```
// This is a code segment from the sram module.

// Declaration of the memory cells
reg [DATA_WIDTH-1 : 0] RAM [RAM_SIZE - 1:0];

// -----
// SRAM cell initialization
// -----
initial begin
    $readmemh("matrices.mem", RAM);
end
```



\$readmemh() is only synthesizable for FPGAs.
You cannot use this for ASIC design!

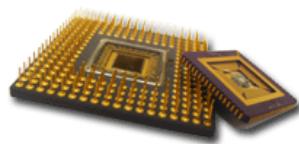




Lab 7 Sample Code (2/3)

Lab 7

- ◆ In MEM file, each input matrix (7x7 matrix) has 49 unsigned 8-bit elements of values between 0 ~ 255 in the column-major format.
- ◆ In MEM file, the starting address of the first matrix in the on-chip SRAM memory is at 0x0000, and the second matrix is at 0x0031.
- ◆ In MEM file, each output matrix has 25 unsigned 19-bit elements of values between 0 ~ $2^{19}-1$.

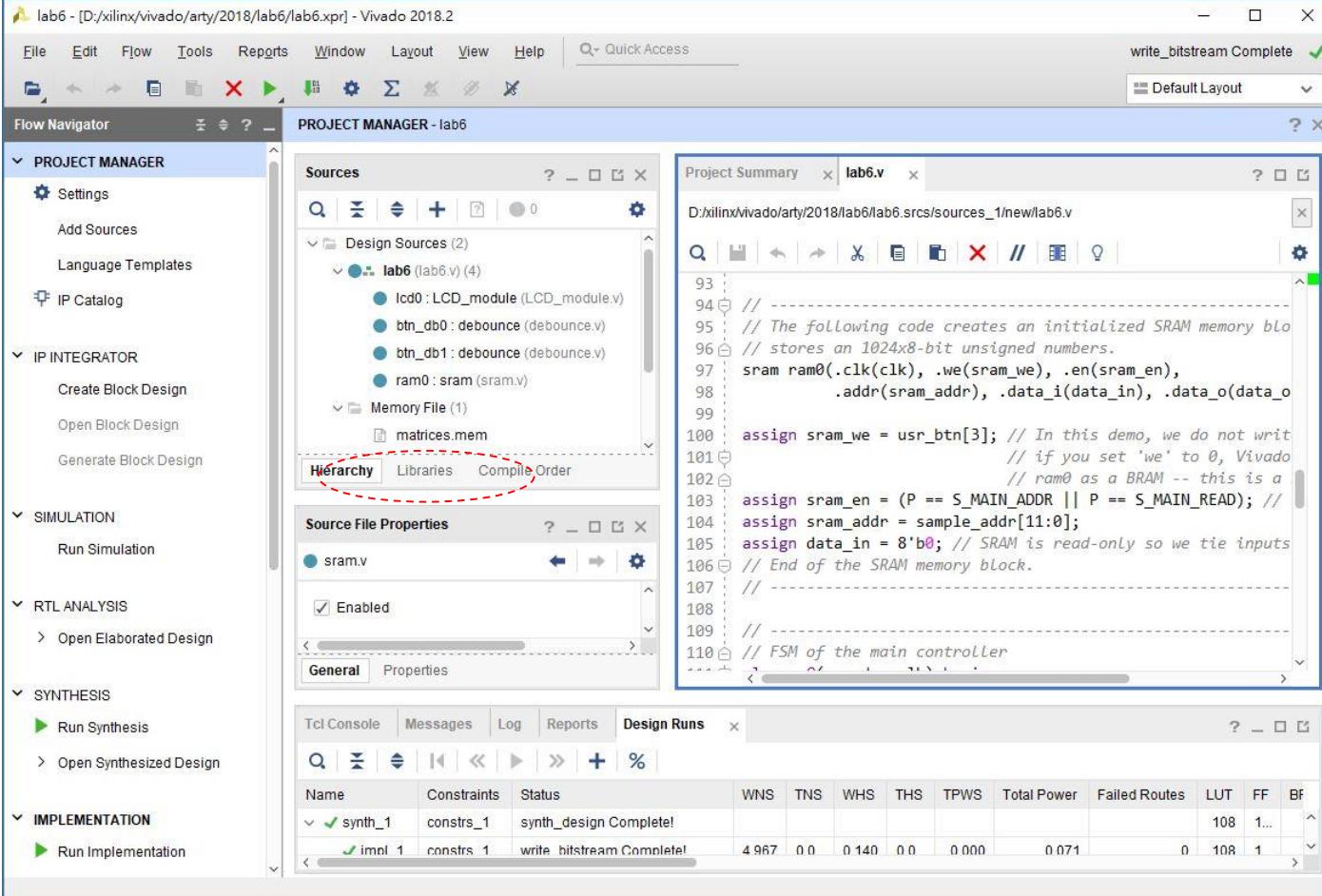




Lab 7 Sample Code (3/3)

Lab 7

- The memory is added to the project as a design source:



The screenshot shows the Vivado 2018.2 Project Manager interface. On the left, the Flow Navigator pane is open, displaying various project management options like Settings, Add Sources, Language Templates, IP Catalog, IP Integrator, Simulation, RTL Analysis, Synthesis, and Implementation. In the center, the PROJECT MANAGER - lab6 window is active. The Sources tab shows two design sources: 'lab6 (lab6.v)' which contains four files (LCD_module, debounce, debounce, sram) and a 'Memory File' named 'matrices.mem'. A red dashed circle highlights the 'Hierarchy' tab in the Source File Properties panel, which is currently showing the properties for 'sram.v'. The right side of the interface displays the code editor for 'lab6.v', containing Verilog code for a SRAM memory block. At the bottom, the Design Runs table shows the status of synthesis and implementation steps, both of which are completed successfully.

```

93 // -----
94 // The following code creates an initialized SRAM memory blo
95 // stores an 1024x8-bit unsigned numbers.
96 ram0(.clk(clk), .we(sram_we), .en(sram_en),
97         .addr(sram_addr), .data_i(data_in), .data_o(data_o
98
99 assign sram_we = usr_btn[3]; // In this demo, we do not writ
100 // if you set 'we' to 0, Vivado
101 // ram0 as a BRAM -- this is a
102 assign sram_en = (P == S_MAIN_ADDR || P == S_MAIN_READ); //
103 assign sram_addr = sample_addr[11:0];
104 assign data_in = 8'b0; // SRAM is read-only so we tie inputs
105 // End of the SRAM memory block.
106
107
108
109
110 // -----
111 // FSM of the main controller

```

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BF
synth_1	constrs_1	synth_design Complete!								108	1...	
impl_1	constrs_1	write_bitstream Complete!	4.967	0.0	0.140	0.0	0.000	0.071	0	108	1	



Lab 7 Sample Code Demo

Lab 7

- ◆ Once you configured the FPGA, you will see the content of the SRAM on the LCD screen.
 - Use BTN0/BTN1 to browse through the SRAM cells
 - The debounce module isn't performing as expected.



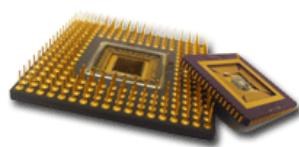
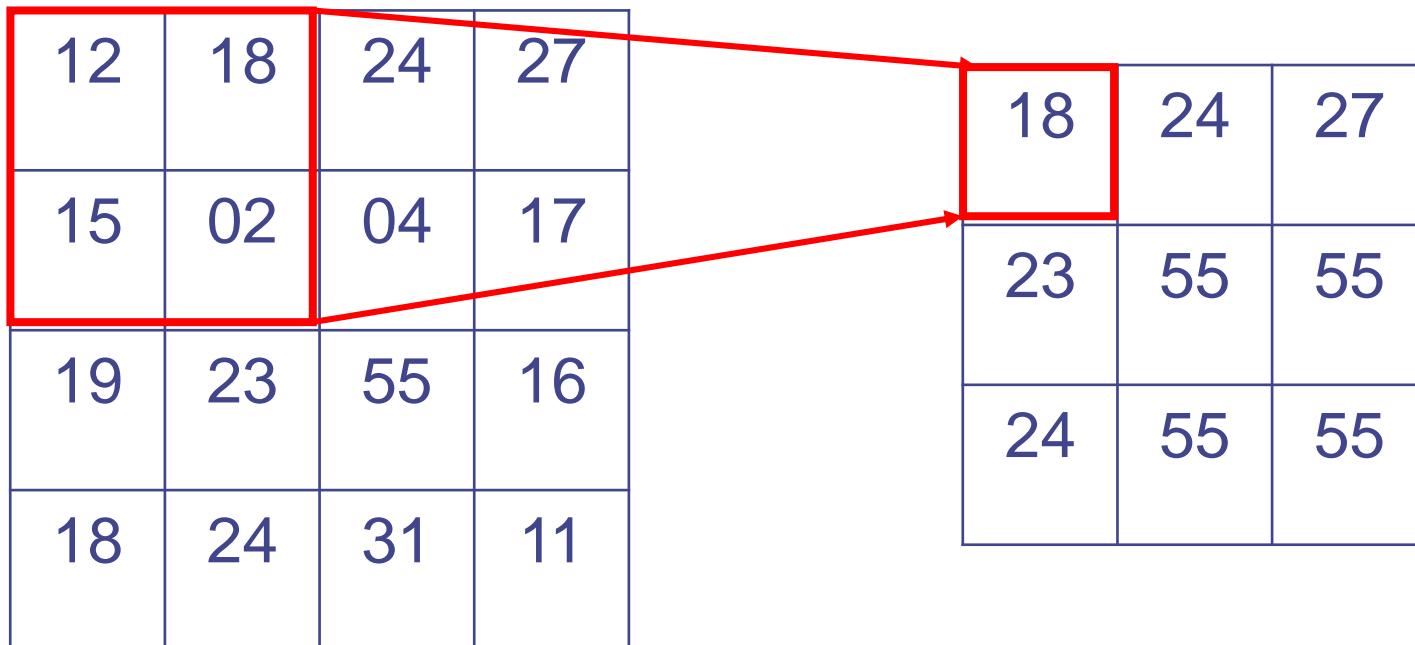


Max-Pooling (1/3)

Lab 7

- ◆ Here is an example for 4*4 matrix applying 2*2 max-pooling.

Max{12,18,15,02}



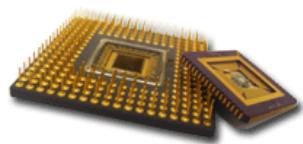
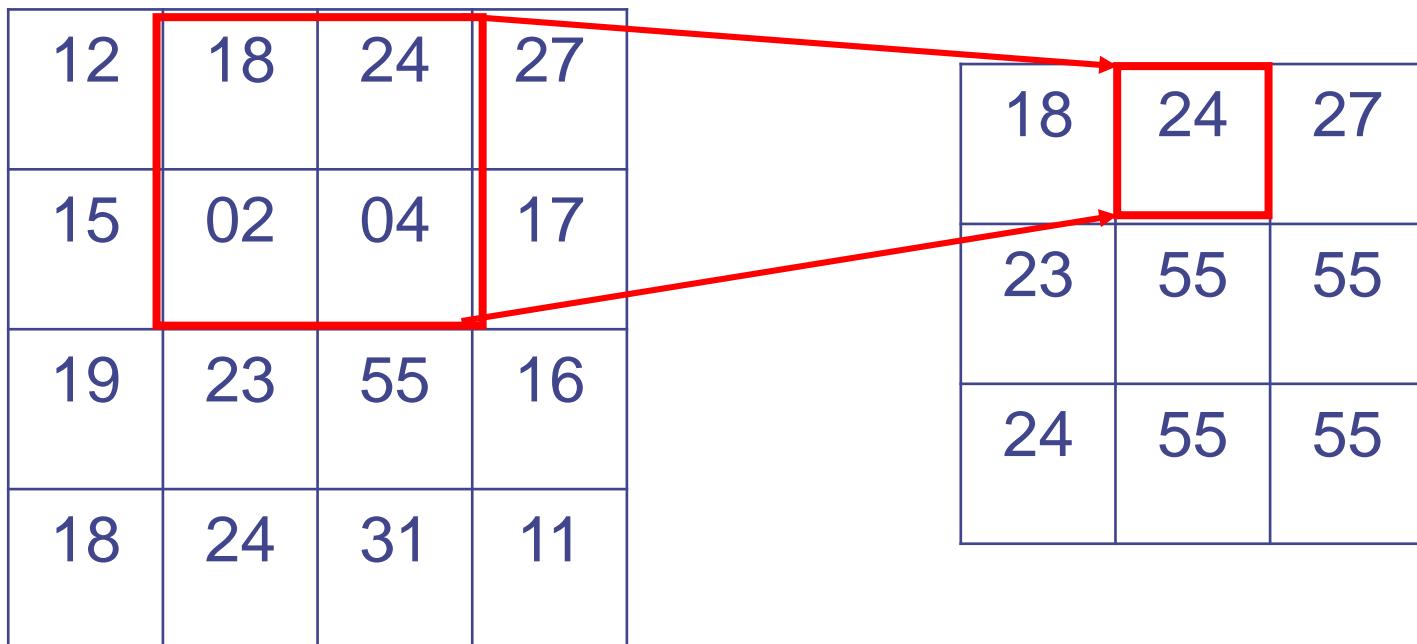


Max-Pooling (2/3)

Lab 7

- ◆ Here is an example for 4*4 matrix applying 2*2 max-pooling.

Max{18,24,02,04}

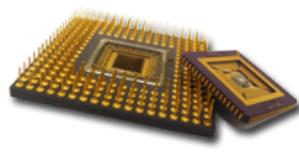
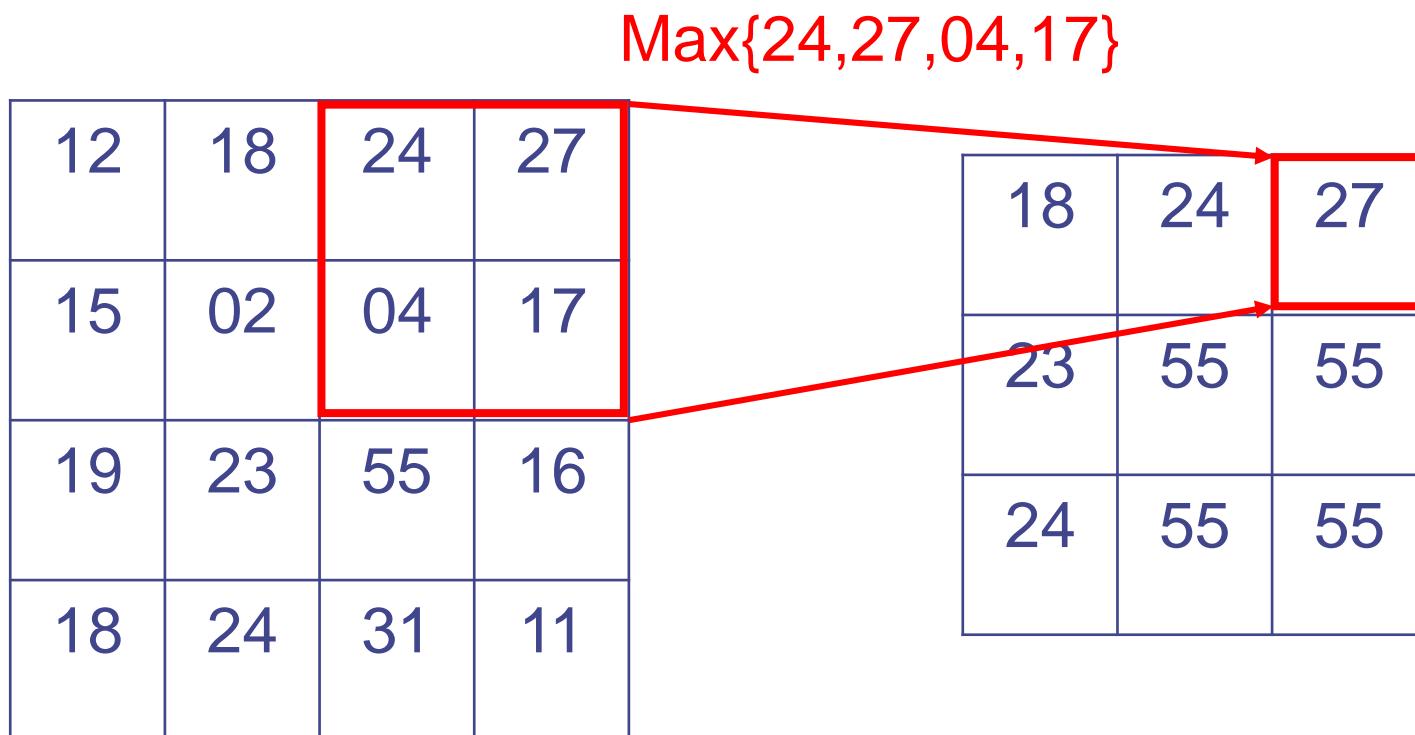




Max-Pooling (3/3)

Lab 7

- ◆ Here is an example for 4*4 matrix applying 2*2 max-pooling.

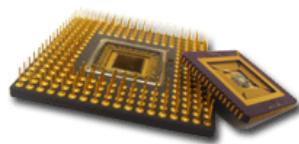




Connecting SRAM to Datapath

Lab 7

- ◆ Since a single-port 8-bit SRAM only outputs one data per clock cycle, you cannot connect an SRAM directly to a parallel-input matrix multiplication datapath.
- ◆ Two possible solutions:
 - Use multiple SRAM blocks, each block has one or two address/data ports.
 - In the FSM, you can design a state to sequentially read the data from the SRAM, and store them in register arrays for parallel computation later.





Timing Issues on a Long Combinational Path

Lab 7

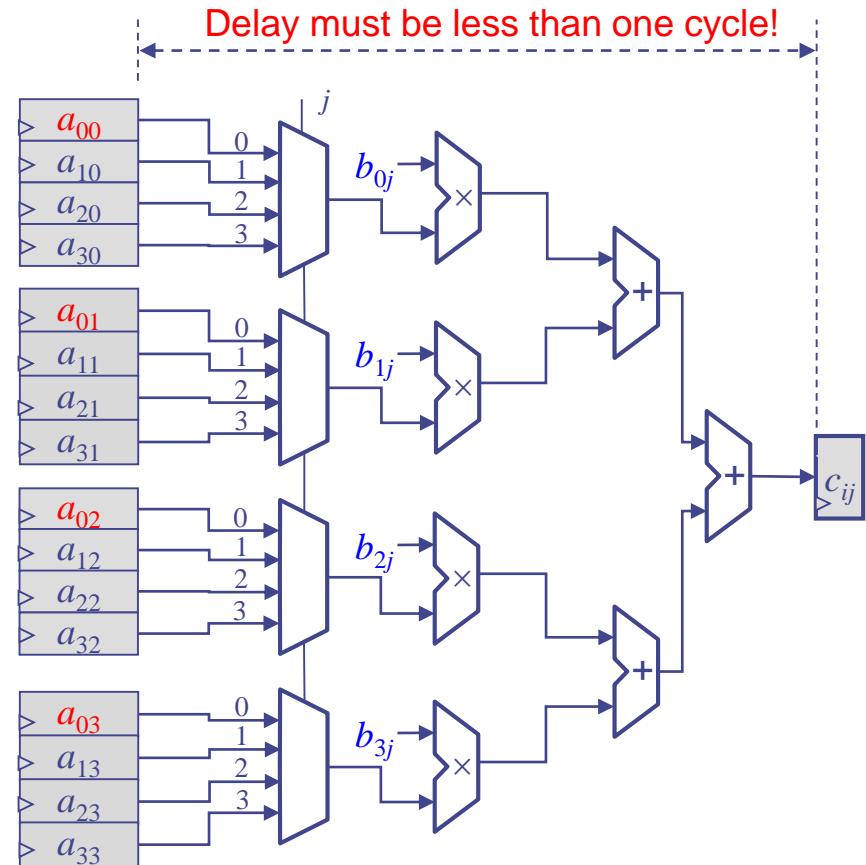
- ◆ A long arithmetic equation will be synthesized into a multi-level combinational circuit path:

```

reg [15:0] a[0:15];
reg [15:0] b[0:15];
reg [31:0] c[0:15];

always @ (posedge clk)
  case (j)
    0: c[i*4] <=
        a[i*4+0]*b[0*4] +
        a[i*4+1]*b[1*4] +
        a[i*4+2]*b[2*4] +
        a[i*4+3]*b[3*4];
    1: . . .
    2: . . .
    3: . . .
  endcase

```





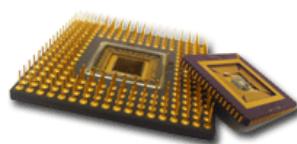
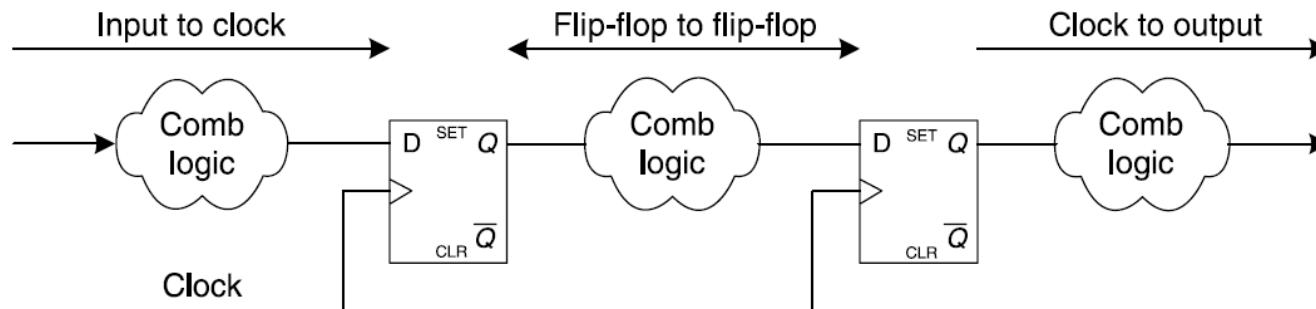
Setup Time and Hold Time

Lab 7

- ◆ To store values into flip-flops (registers) properly, the minimum allowable clock period T_{min} is computed by

$$T_{min} = T_{path_delay} + T_{setup}$$

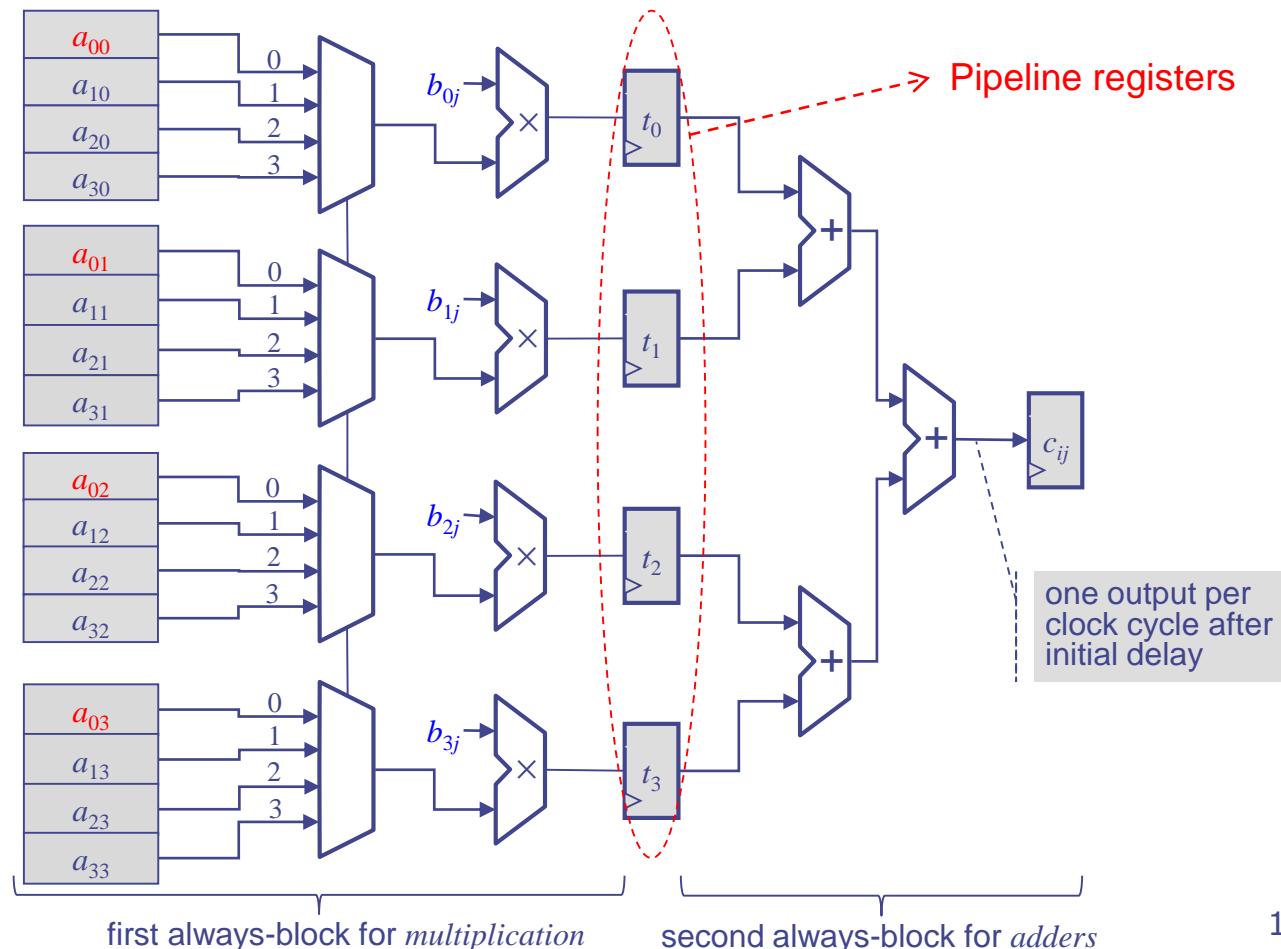
- T_{path_delay} is the propagation delay through logics and wires.
- T_{setup} is the minimum time data must arrive at D before the next rising edge of clock (setup time).





Breaking a Long Combinational Path

- You can divide a long combinational path into two or more always blocks to meet the timing constraint.



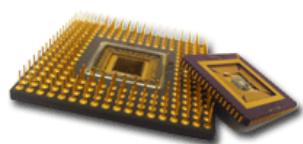


Things to do in Lab 7 (1/3)

Lab 7

- ◆ For Lab 7, you need to read two matrix from BRAM, apply 3x3 max-pooling and transpose one of them before performing matrix multiplication, and then print the result to the terminal via UART.
- ◆ Let “£” be the function of 3*3 max-pooling operation and “T” be the transpose operation. “A” is the first input 7*7 matrix and “B” is the second 7*7 matrix.
- ◆ You should perform the following operations and output 5*5 matrix C through UART:

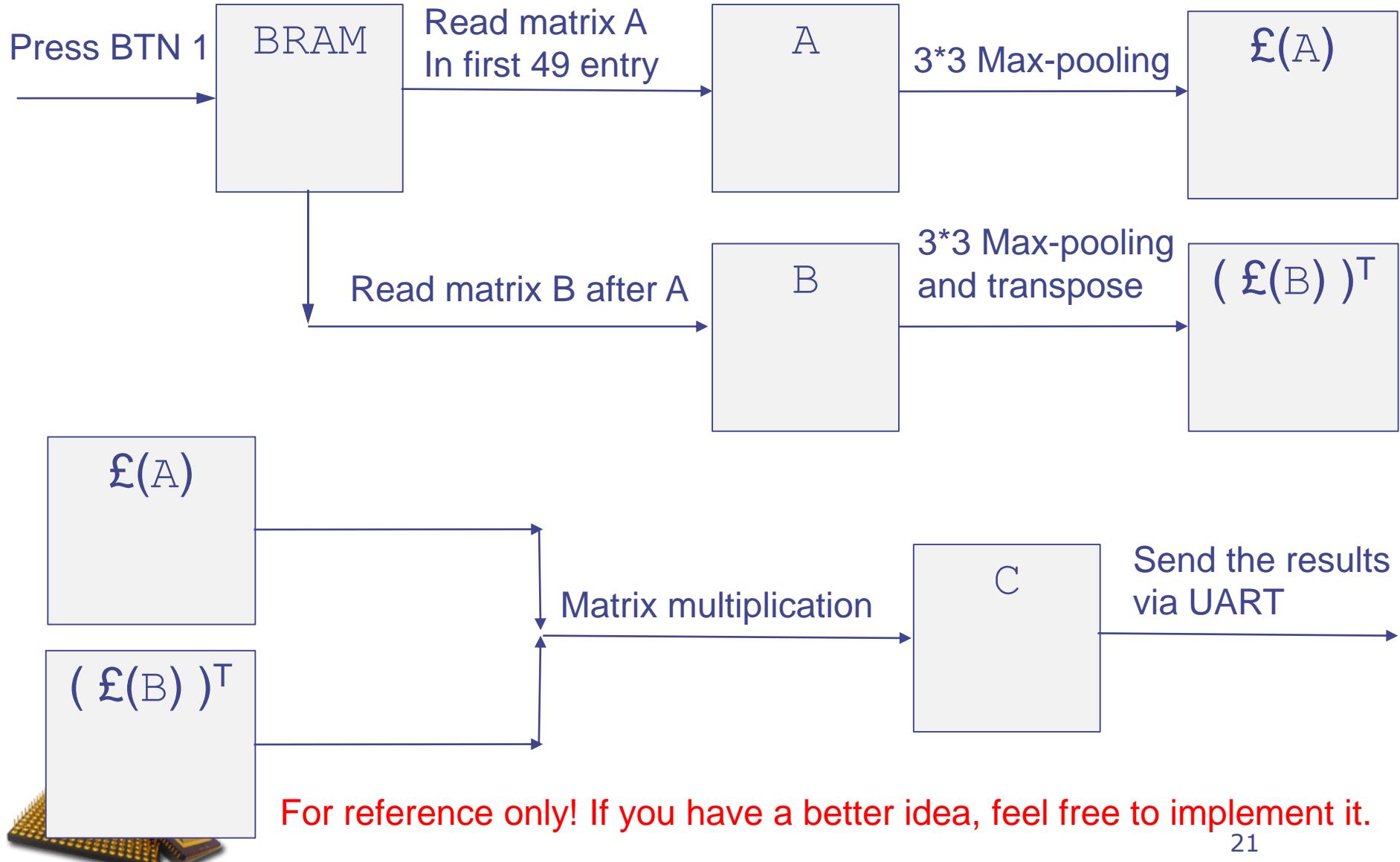
$$C = \mathcal{L}(A) * (\mathcal{L}(B))^T$$





Things to do in Lab 7 (2/3)

Lab 7





Things to do in Lab 7 (3/3)

Lab 7

- ◆ For Lab 7, after those operation, your circuit must print the resulting matrix via the UART as follows:

The matrix operation result is:

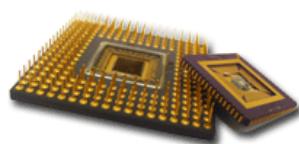
[01347, 012AC, 01211, 01176, 010DB]

[013F6, 01356, 012B6, 01216, 01176]

[014A5, 01400, 0135B, 012B6, 01211]

[01554, 014AA, 01400, 01356, 012AC]

[01603, 01554, 014A5, 013F6, 01347]





Check FPGA Resource Utilization

Lab 7

The screenshot shows the Vivado 2018.2 interface with the project "lab6" open. The Flow Navigator panel on the left has "IMPLEMENTATION" expanded, with "Report Utilization" circled in red. The central workspace shows the "IMPLEMENTED DESIGN - xc7a35tcsg324-1L" tab. The bottom-right corner of the workspace displays a BRAM block diagram with four 32x32-bit blocks labeled X0Y0, X0Y1, X1Y1, and X1Y2. The utilization report in the bottom-right pane shows the following data:

Resource	Utilization	Available	Utilization %
LUT	108	20800	0.52
FF	171	41600	0.41
BRAM	0.50	50	1.00
IO	16	210	7.62

A red dashed arrow points from the highlighted BRAM utilization value to a bar chart below it, which shows the distribution of resources across the utilization scale. The chart indicates that 0.50 units of BRAM are used, which is 1% of the available 50 units.

Used 0.5 block of a 36-kbit BRAM.

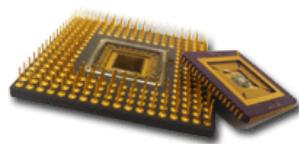


Lab 7 Grading (1/2)

Lab 7

- ◆ Your grade will be based on correctness and efficient use of logic; smaller logic usage is preferred.
 - The “size” of the logic is calculated by the number of physical multipliers, LUTs, Flip-flops (FFs).
 - BRAM blocks are considered as memory resource, not logic resource.

- ◆ You should use **no more** than **25** multipliers.





Lab 7 Grading (2/2)

Lab 7

◆ Functional Correctness (3 hidden testcases) – 50%

◆ Timing Check – 20%

- If WNS > 0 in your design, you will pass this part.
- If you failed any testcases, you will lose these points.

◆ Utilization – 20%

- TA would rank your design by the following formula:

$$0.35 * \text{LUT utilization}(\%) + 0.35 * \text{FF utilization}(\%) + 0.3 * \text{DSP utilization}(\%)$$

- Less is better, the ranked result will be divided into 5 level, the point you get will depend on which level you are. However, if the DSP you used is **more than 25**, you will get **0 grade** in Utilization part.

- If you failed any testcases, you will lose these points.

◆ Question – 10%

