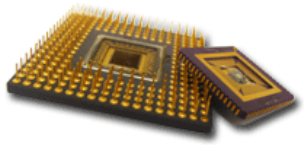
Lab 7: Max-Pooling and Matrix Multiplication Circuit

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

Fall, 2024

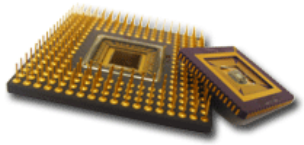
Lab 7: Matrix Calculation

- 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 11/04 by 6:00pm.



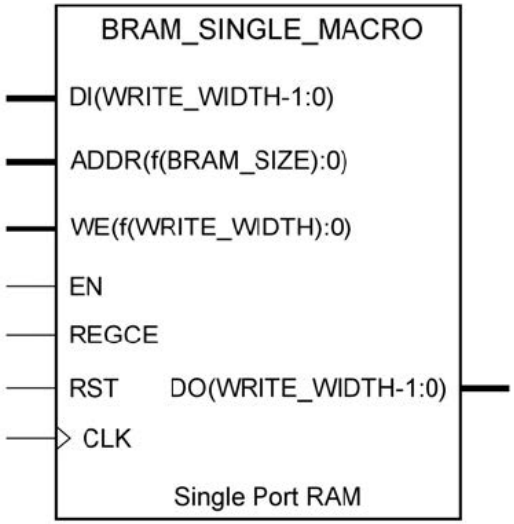
Instantiation of an On-Chip SRAM

- 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.

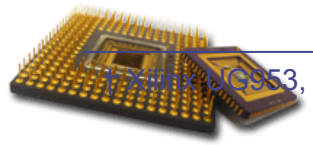


SRAM on FPGAs

- 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.



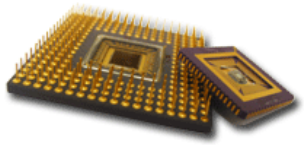
Port	Direction	Width	Function
DO	Output	See Configuration Table below.	Data output bus addressed by ADDR.
DI	Input	See Configuration Table below.	Data input bus addressed by ADDR.
ADDR	Input	See Configuration Table below.	Address input bus.
WE	Input	See Configuration Table below.	Byte-Wide Write enable.
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.



† UG953, *Vivado Design Suite 7 Series FPGA and Zynq-7000 All Programmable SoC Libraries Guide*, page 120.

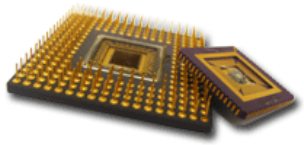
General SRAM Signals (1/2)

- **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.



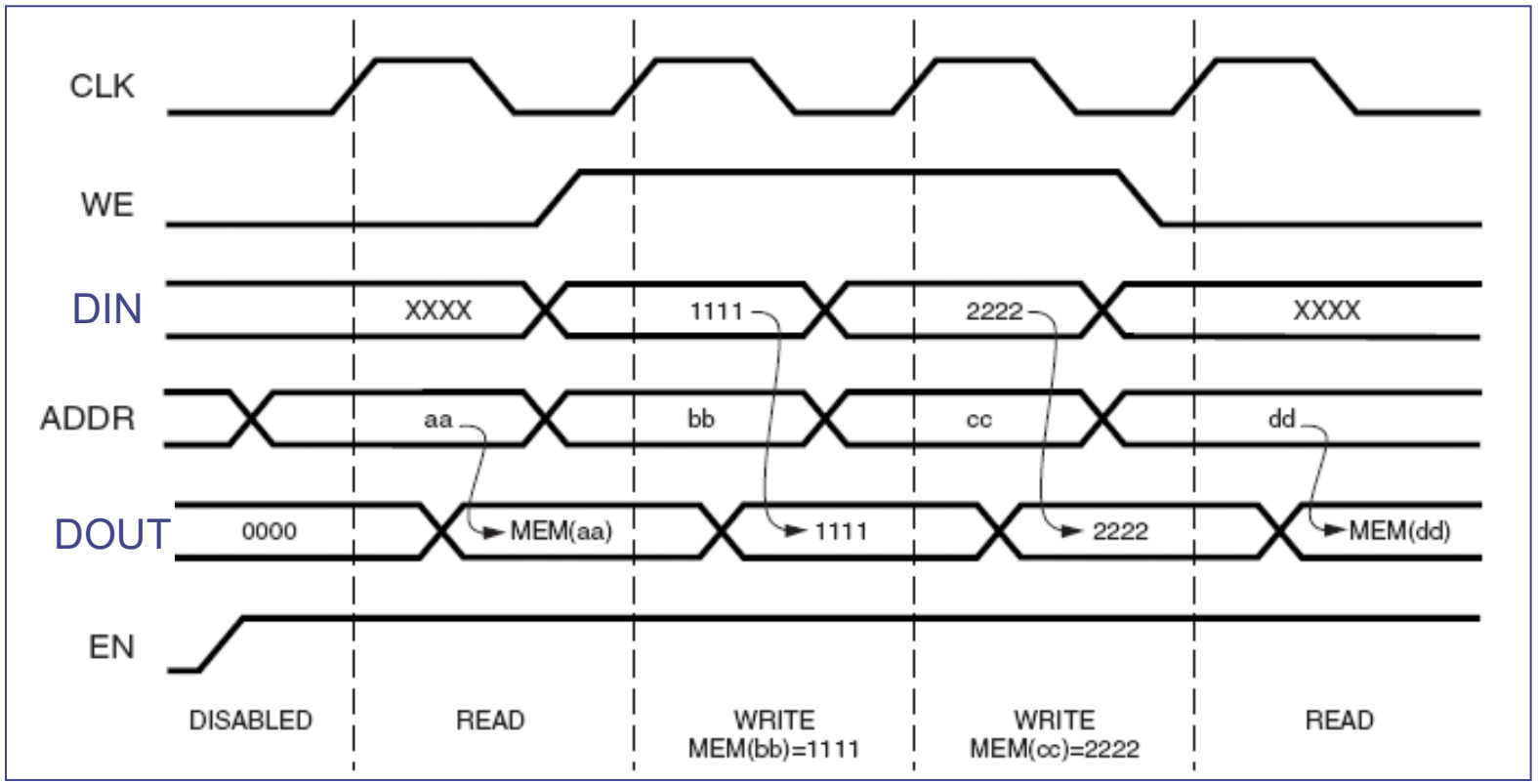
General SRAM Signals (2/2)

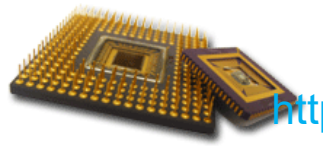
- **DIN** – Data input port
  - The DI port provides the new data to be written into the RAM.
- **DOUT** – Data output port
  - The DOUT port reflects the contents of the memory cells referenced by the address bus at the last active clock edge.
  - During a write operation, the DOUT port reflects the DIN port.



Timing Diagram

- For single-port SRAM:





https://www.xilinx.com/support/documentation/ip_documentation/sp_block_mem.pdf

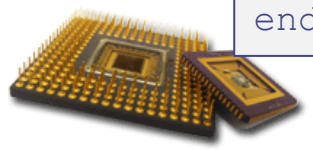
Instantiate an SRAM by Inference

- 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
```



Lan 7 Sample Code (1/2)

- 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.

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

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

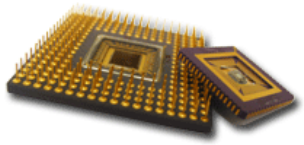
8 bits

```
E1
6B
D7
1D
B6
0C
55
2D
...
```

The content of "matrices.mem"

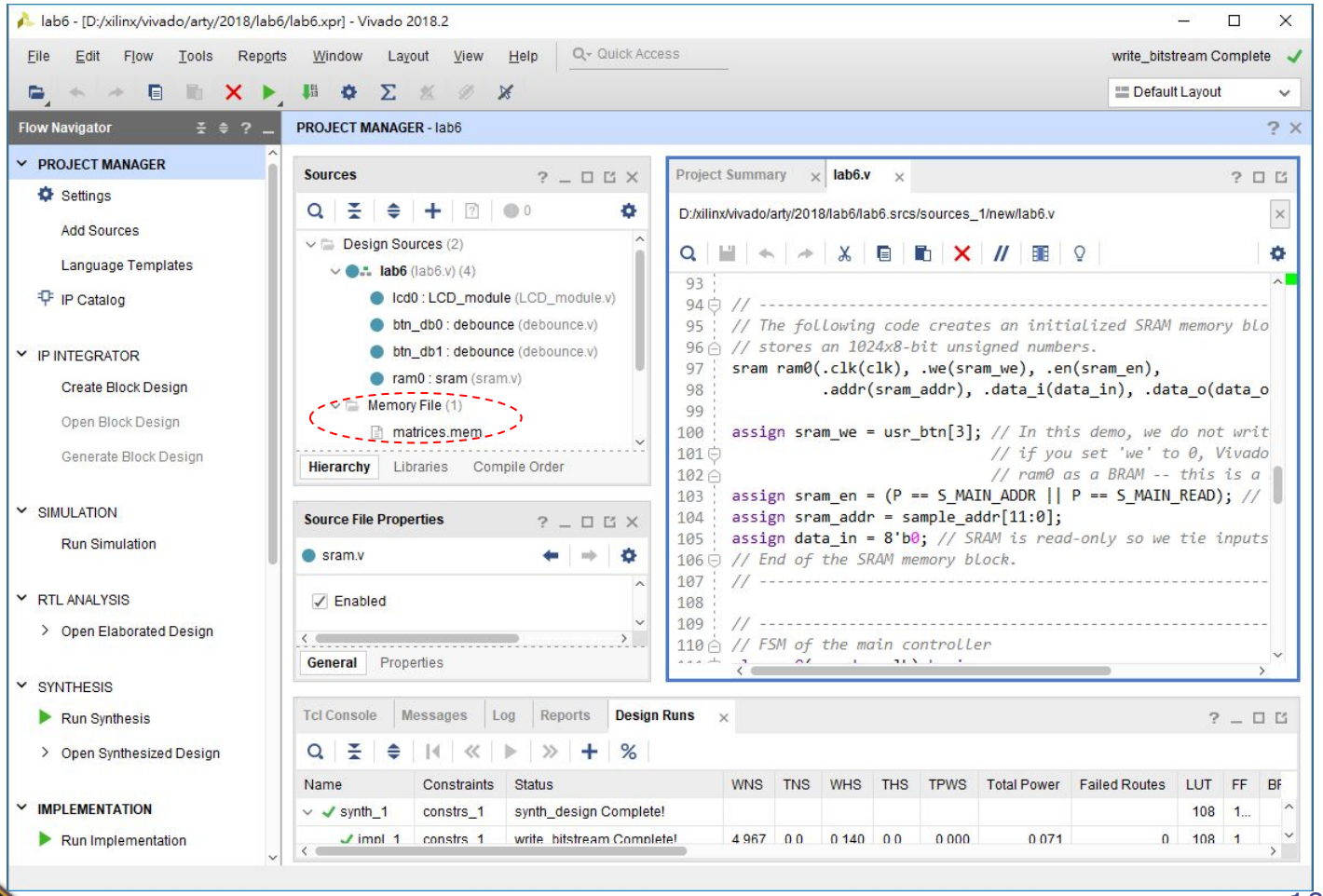
Two groups of 49 elements

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



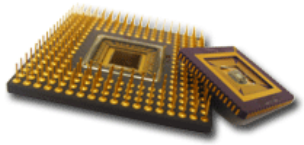
Lab 7 Sample Code (2/2)

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



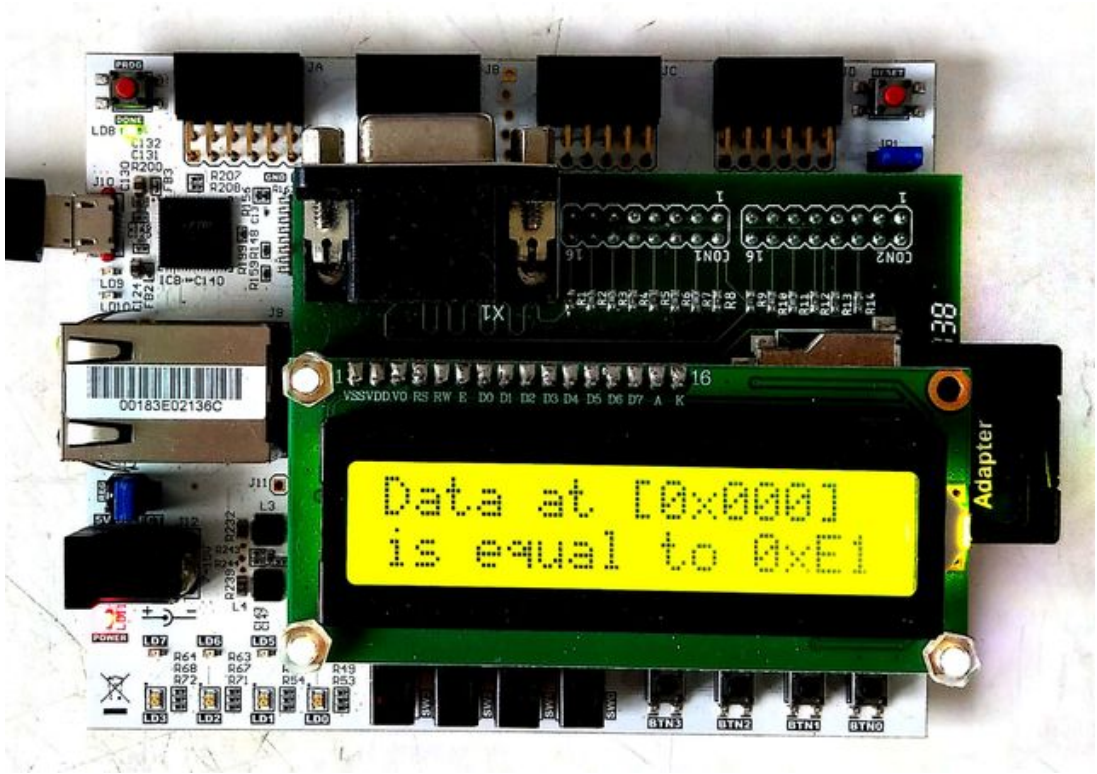
Input Matrix Format (MEM File)

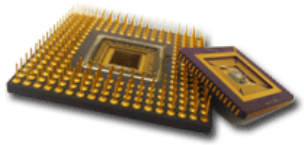
- Each input matrix has 49 unsigned 8-bit elements of values between 0 ~ 255 in the column-major format.
- The staring address of the first matrix in the on-chip SRAM memory is at 0x0000, and the second matrix is at 0x0031.
- Each output matrix has 25 unsigned 19-bit elements of values between 0 ~ 2^{19}-1.



Lab 7 Sample Code Demo

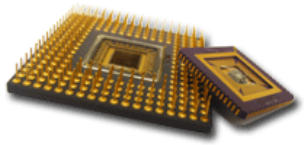
- Once you configured the FPGA, you will see the content of the SRAM on the LCD screen.
  - Use BTN0/BTN1 to browse through the SRAM cells
  - The debounce module isn't performing as expected.





Connecting SRAM to Datapath

- 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.

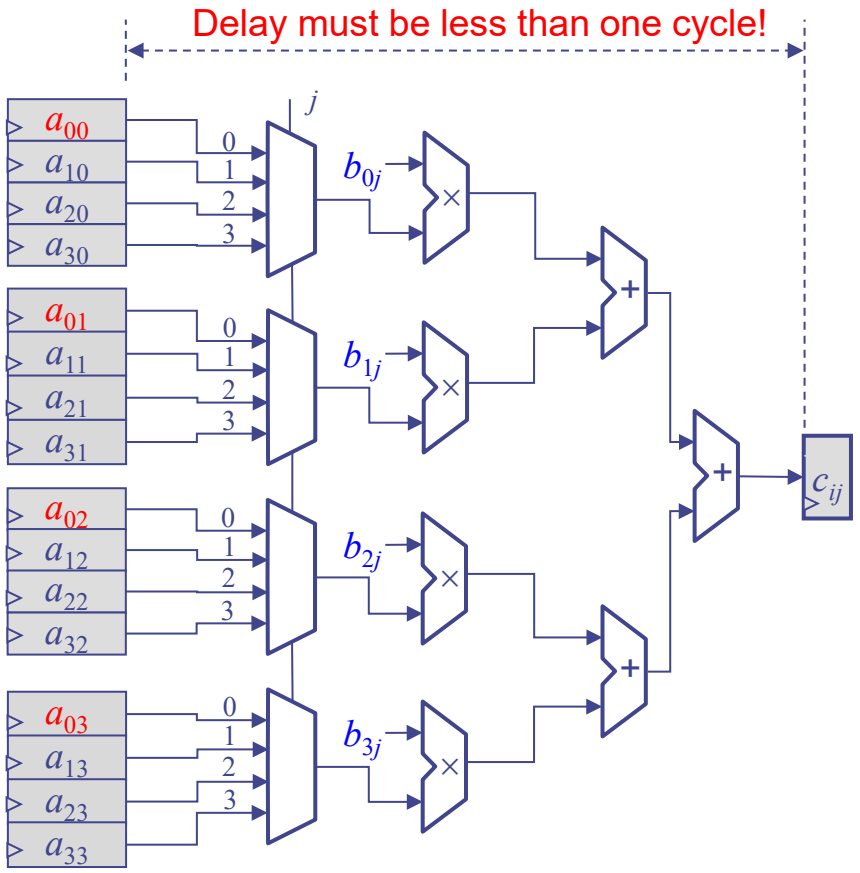


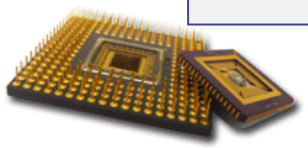
Timing Issues on a Long Combinational Path

- 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
```



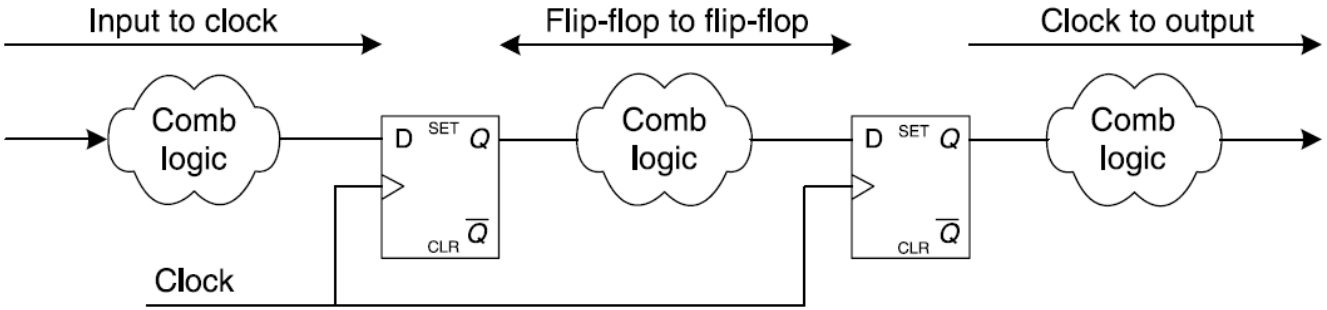


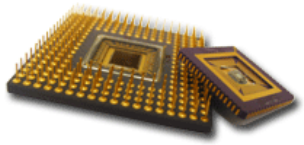
Setup Time and Hold Time

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

$$T_{min} = T_{path\_delay} + T_{setup}$$

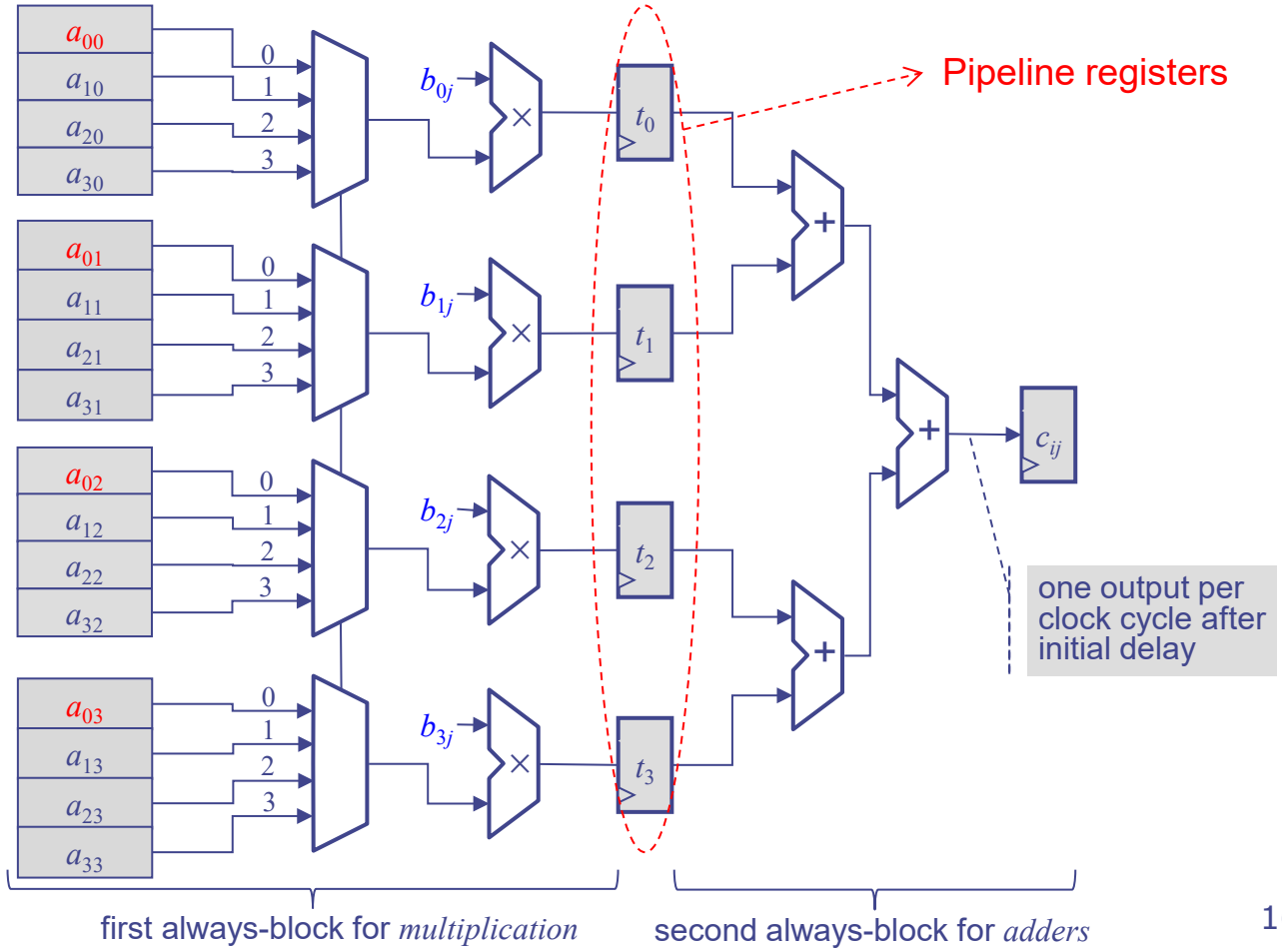
  - T_{path_delay} is the propagation delay through logics and wires.
  - T_{setup} is the minimum time data must arrive at D before the next rising edge of clock (setup time).





Breaking a Long Combinational Path

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



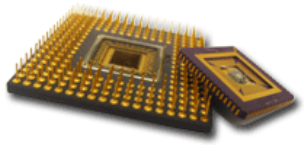
Max-Pooling (1/3)

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

Max{12,18,15,02}

12	18	24	27
15	02	04	17
19	23	55	16
18	24	31	11

18	24	27
23	55	55
24	55	55



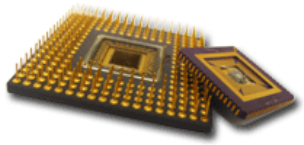
Max-Pooling (2/3)

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

Max{18,24,02,04}

12	18	24	27
15	02	04	17
19	23	55	16
18	24	31	11

18	24	27
23	55	55
24	55	55



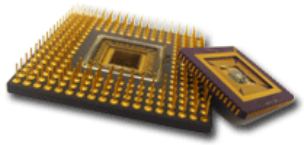
Max-Pooling (3/3)

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

Max{24,27,04,17}

12	18	24	27
15	02	04	17
19	23	55	16
18	24	31	11

18	24	27
23	55	55
24	55	55



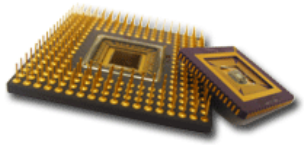
Things to do in Lab 7 (1/3)

- 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.

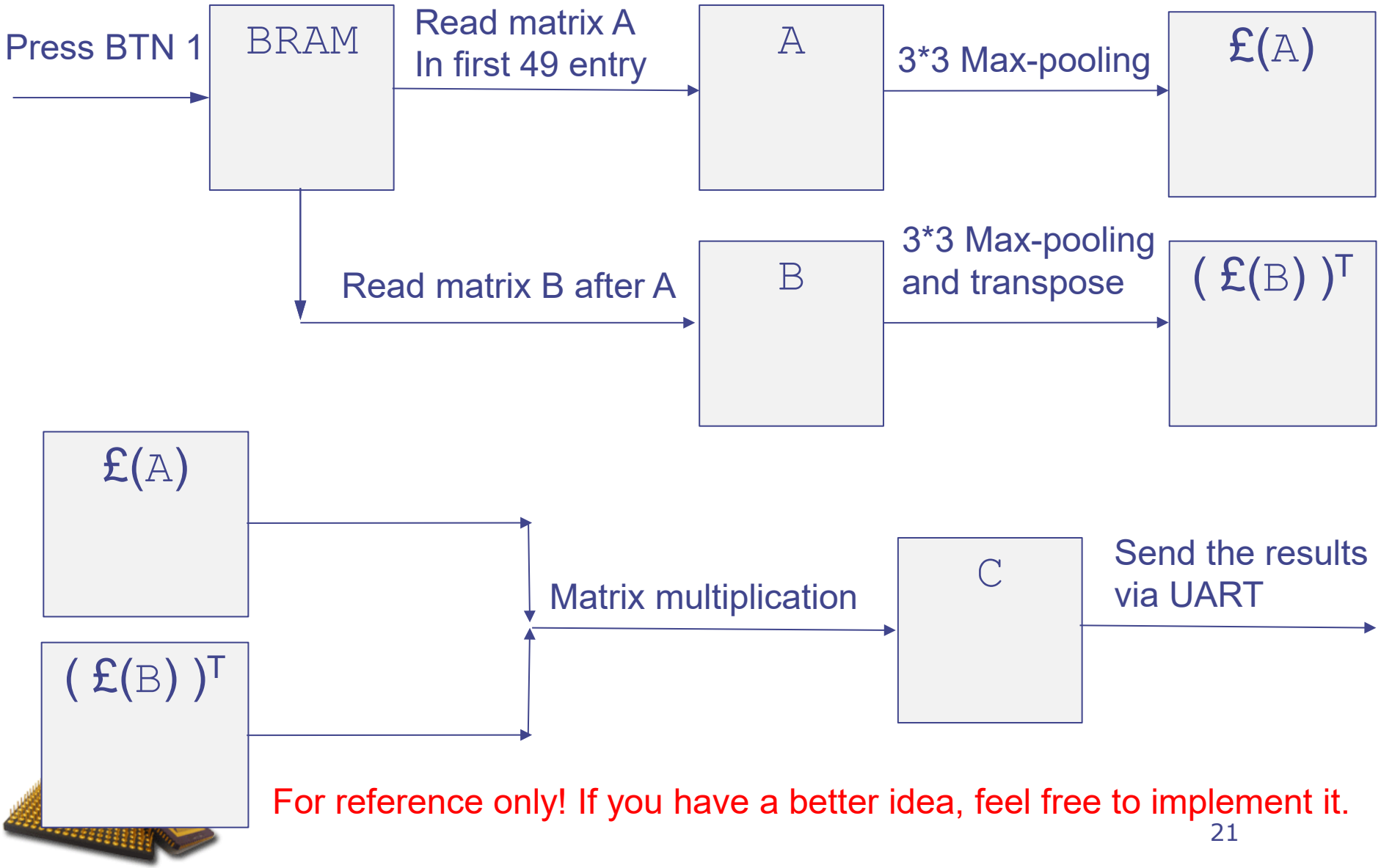
- Let “£” be the function of 3*3 max-pooling operation and “T” be the transpose operation. “A” is the first input 7*7 matrix and “B” is the second 7*7 matrix.

- You should perform the following operations and output 5*5 matrix C through UART:

$$C = £(A) * (£(B))^T$$



Things to do in Lab 7 (2/3)

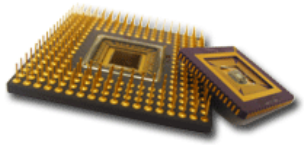


For reference only! If you have a better idea, feel free to implement it.

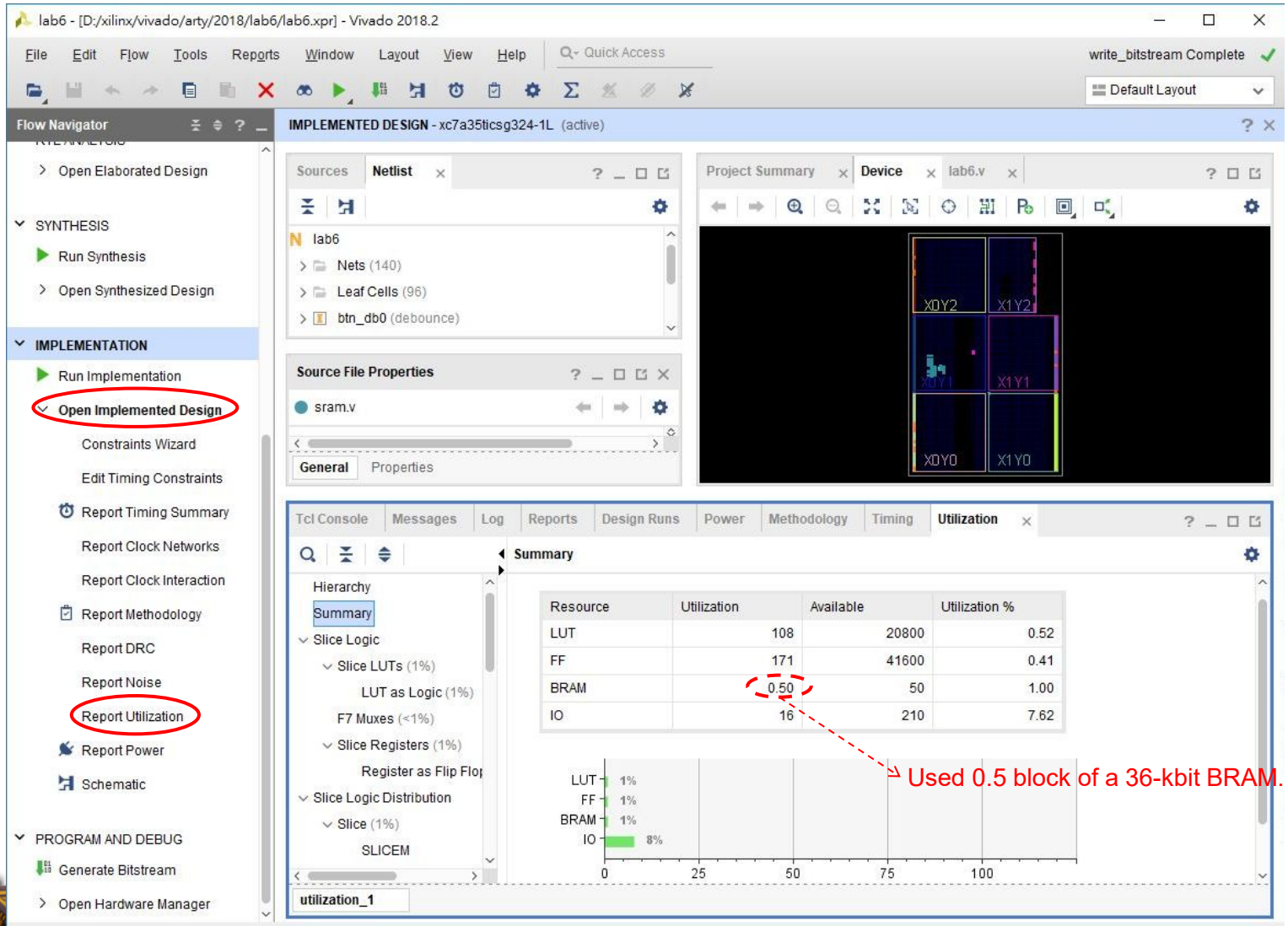
Things to do in Lab 7 (3/3)

- 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]
```



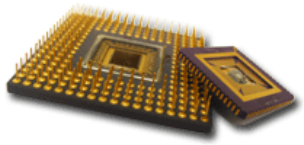
Check FPGA Resource Utilization



Lab 7 Grading (1/2)

- 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)

- 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 * LUT utilization(%) + 0.35 * FF utilization(%) + 0.3 * 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%