Lab 6: Matrix Multiplication Circuit for Real

National Chiao Tung University
Chun-Jen Tsai
11/02/2018

Lab 6: Matrix Multiplication

- In this lab, you will design a circuit to do 4×4 matrix multiplications.
  - Your circuit has an SRAM block that stores two 4×4 matrices.
  - The user press BTN1 to start the circuit
  - The circuit reads the matrices, perform the multiplication, and print the output matrix through the UART to a terminal window
- The deadline of the lab is on 11/13

Design Constraint of Lab 6

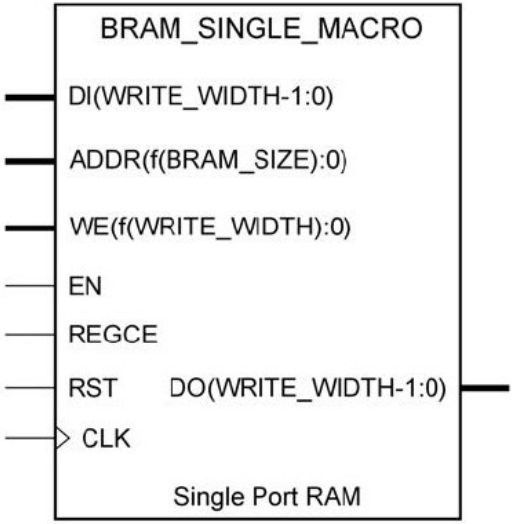
- You must use no more than 16 multipliers to implement your circuit
  - Each Atrix-7 35T FPGA on the Arty board has 90 20×18-bit multipliers
- Your grade will be based on correctness and logic usage; the smaller the logic, the better
  - The “size” of the logic is calculated by the number of physical multipliers, LUTs, Flip-flops (FFs)
  - BRAM blocks are considered as memory resource, not logic resource

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 output 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.

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

General SRAM Signals (1/2)

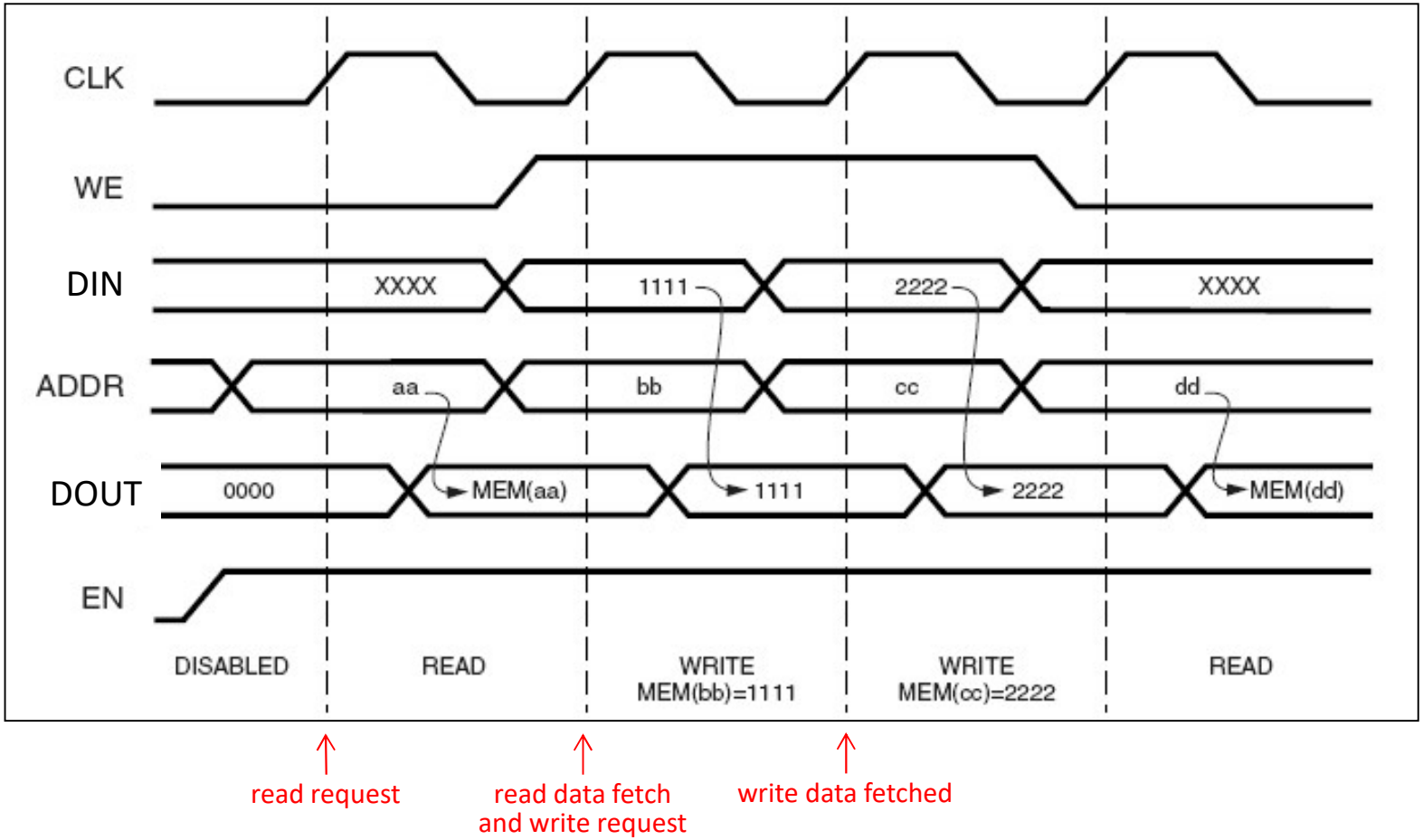
- **CLK** – Clock
  - Independent clock pins for synchronous operations
- **EN** – Enable
  - The read, write and reset functionality of the port is only active when this signal is enabled
- **WE** – Write enable
  - When active, the contents of the data input bus are written to the RAM, and the new 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:



Instantiates 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
```

The Sample Code of Lab 6 (1/2)

- The sample code of lab 6 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
```

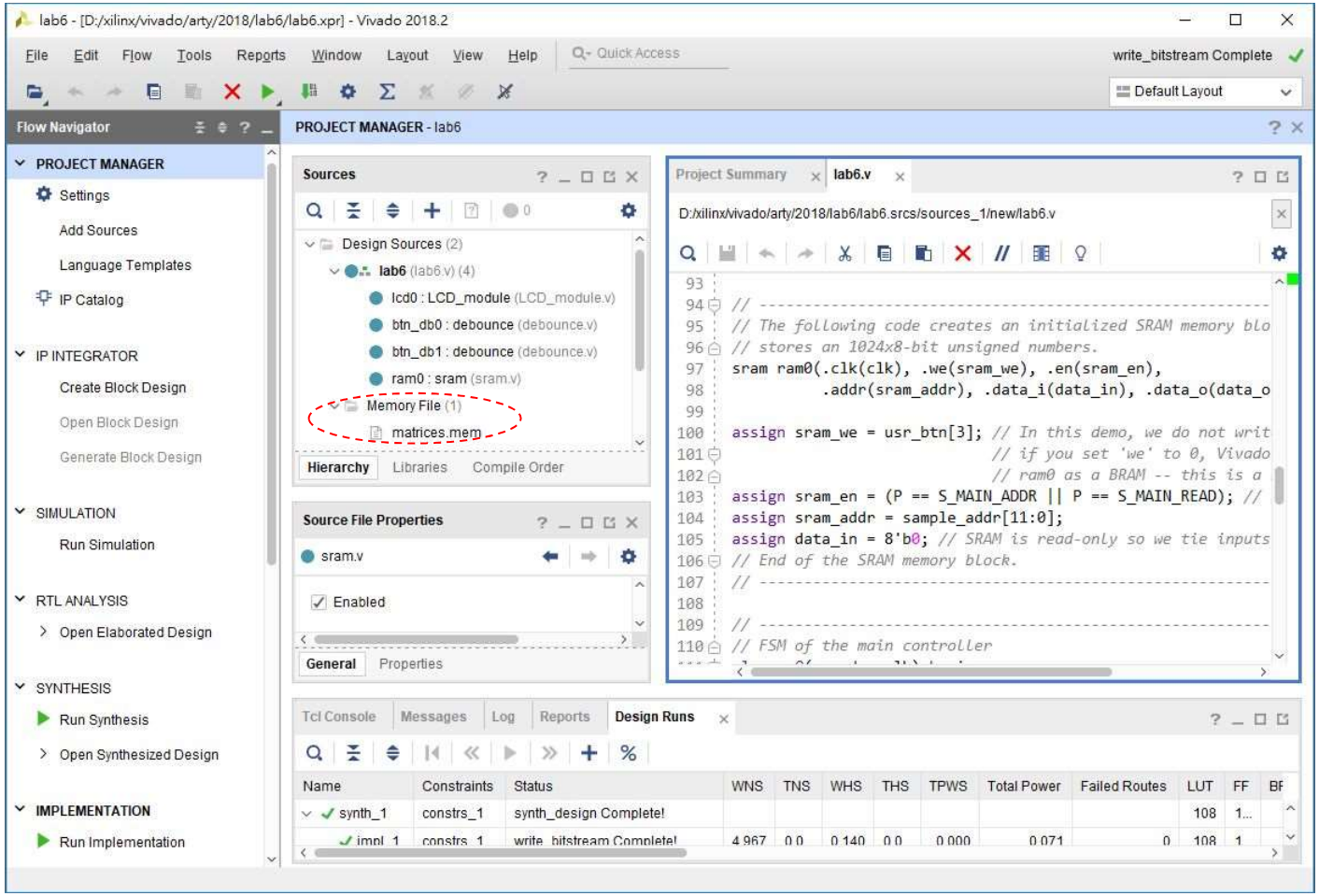
```
E1
6B
D7
1D
B6
0C
55
2D
...
```

The content of "matrices.mem"

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

The Sample Code of Lab 9 (2/2)

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



The Input Matrix Format

- Each input matrix has 16 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 0x0010
- The output matrix has 16 unsigned 18-bit elements

Demo of the Lab 6 Sample System

- 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

Things to do in Lab 6

- For Lab 6, after the multiplication, your circuit must prints the resulting matrix to the UART as follows:

```
The matrix multiplication result is:
[ 11CE9, 18749, 0EE26, 16F64 ]
[ 0ED5B, 1091D, 04768, 06376 ]
[ 167B9, 1BF8A, 0E496, 1504F ]
[ 09901, 0F404, 08F23, 0C4A5 ]
```

Connecting SRAM to Datapath

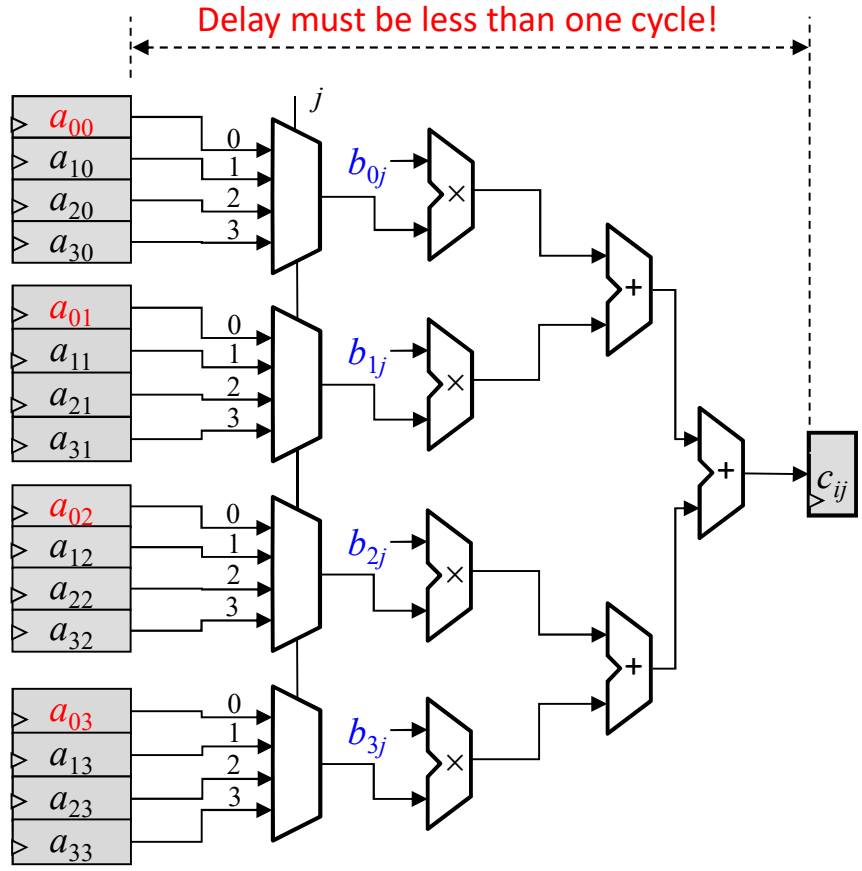
- Since a single-port 8-bit SRAM only output one data per clock cycle, you cannot connect an SRAM directly to a parallel-input matrix multiplication datapath
- Two possible solutions:
  - Use multiple SRAM blocks, each block has one or two address/data ports
  - In the FSM, design a state to sequentially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

Timing Issues on 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
```

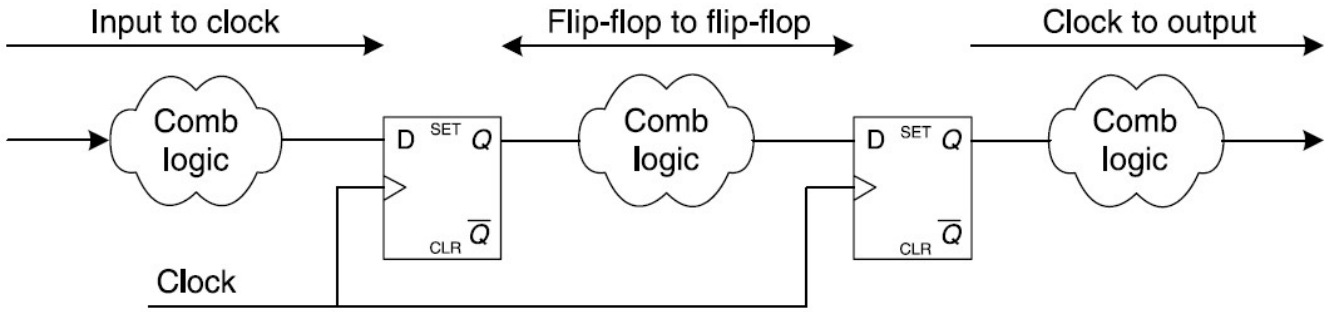


Setup Time and Hold Time

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

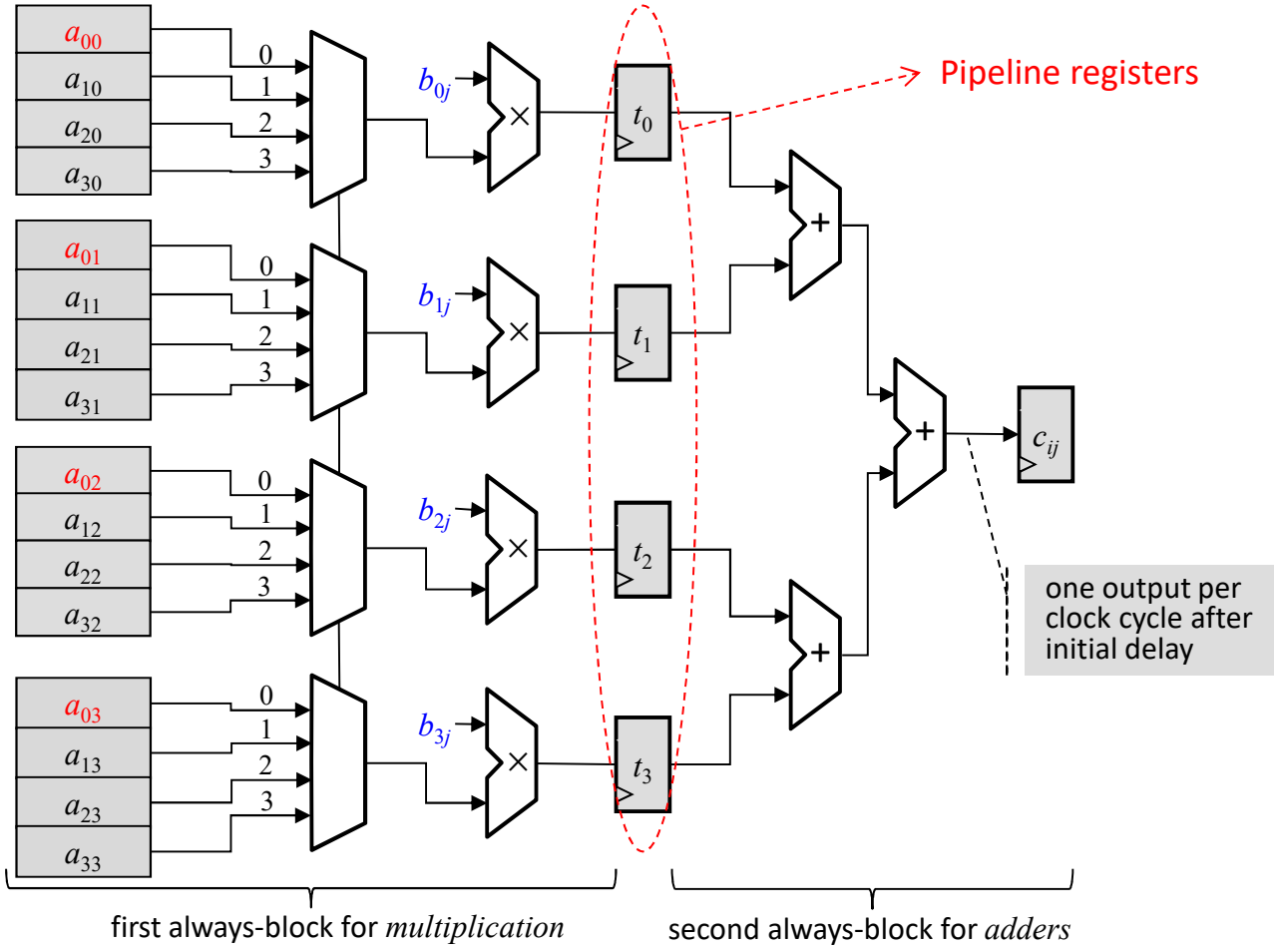
$$T_{min} = T_{path\_delay} + T_{setup}$$

  - 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



Check FPGA Resource Utilization