



Lab 7: Matrix Multiplication Circuit for Real

Chun-Jen Tsai and Lan-Da Van
Department of Computer Science
National Yang Ming Chiao Tung University
Taiwan, R.O.C.
Fall, 2021



Lab 7: Matrix Multiplication

Lab 7

- ◆ In this lab, you will design a circuit to do 4×4 matrix multiplications.
 - Your circuit has a Block RAM (BRAM) that stores two 4×4 matrices.
 - The user presses BTN1 to start the circuit.
 - The circuit reads the matrices, performs the multiplication, and prints the output matrix through the UART to a terminal window.

- ◆ The lab file submission deadline is on 11/22 by 6:00pm.

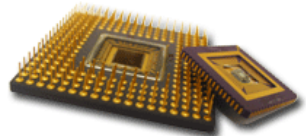




Design Constraint of Lab 7

Lab 7

- ◆ 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.



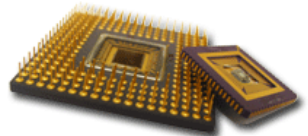


Instantiation of an On-Chip SRAM

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.

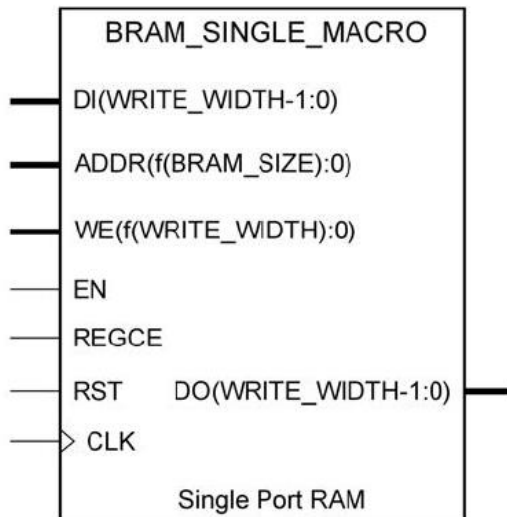




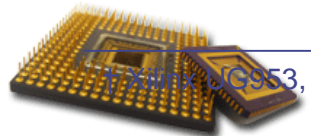
SRAM on FPGAs

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

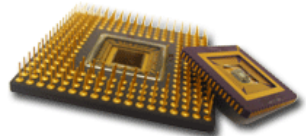
- The read, write and reset functionality of the port is only active when this signal is enabled.

◆ **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)

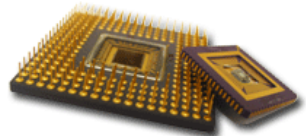
Lab 7

◆ **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.

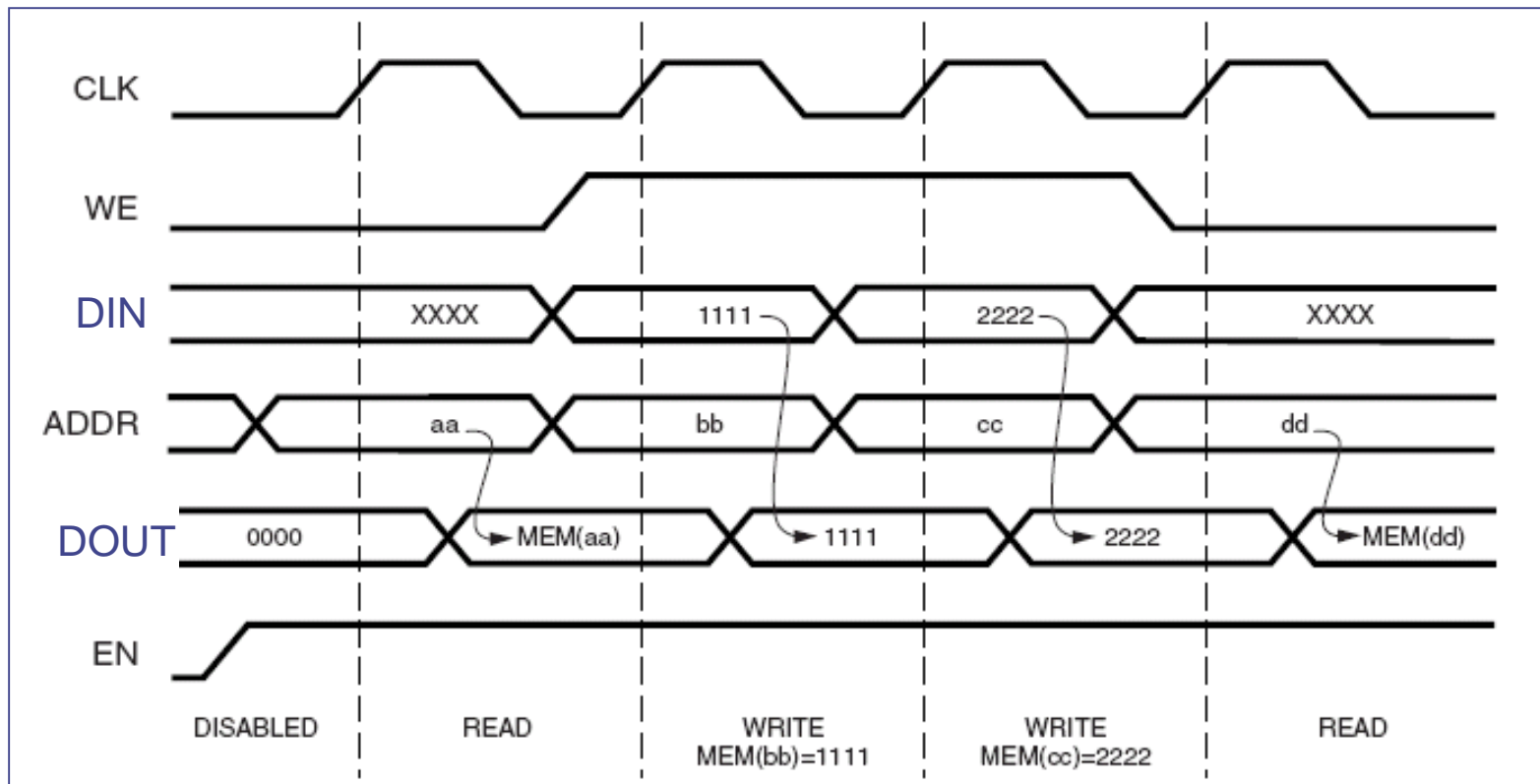




Timing Diagram

Lab 7

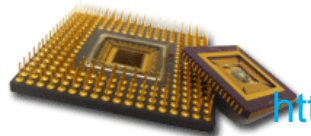
◆ For single-port SRAM:



↑
read request

↑
read data fetch
and write request

↑
write data fetched





Instantiate an SRAM by Inference

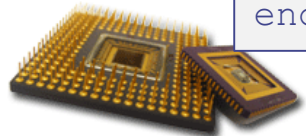
Lab 7

- ◆ 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
```





The Sample Code of Lab 7 (1/2)

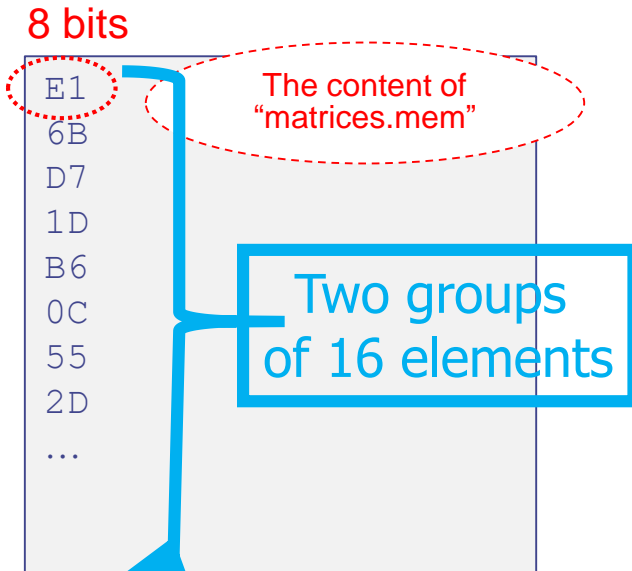
Lab 7

- ◆ 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:

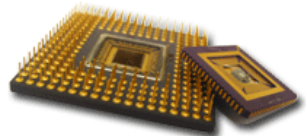
```
// This is a code segment from the sram module.

// 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!





The Sample Code of Lab 7 (2/2)

Lab 7

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

The screenshot shows the Vivado 2018.2 IDE. The **Project Manager** window displays the project **lab6** with the following design sources:

- lcd0: LCD_module (LCD_module.v)
- btn_db0: debounce (debounce.v)
- btn_db1: debounce (debounce.v)
- ram0: sram (sram.v)
- Memory File (1): matrices.mem

The **Source File Properties** window for **sram.v** shows the **General** tab with the **Enabled** checkbox checked.

The **Project Summary** window shows the source file **lab6.v** with the following code:

```

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

The **Design Runs** window shows the following table:

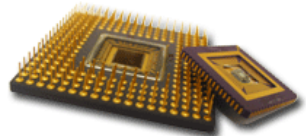
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BF
synth_1	constrs_1	synth_design Complete!								108	1...	
impl_1	constrs_1	write_bitstream Complete!	4.967	0.0	0.140	0.0	0.000	0.071	0	108	1	



Input Matrix Format (MEM File)

Lab 7

- ◆ Each input matrix has 16 unsigned 8-bit elements of values between 0 ~ 255 in the column-major format.
- ◆ The starting 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 7 Sample System

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





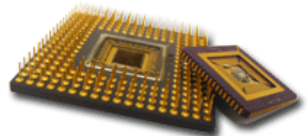
Things to do in Lab 7

Lab 7

- ◆ For Lab 7, after the multiplication, your circuit must print 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 ]
```

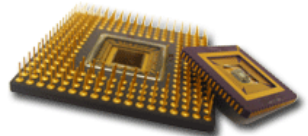




Connecting SRAM to Datapath

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 Long Combinational Path

Lab 7

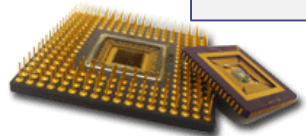
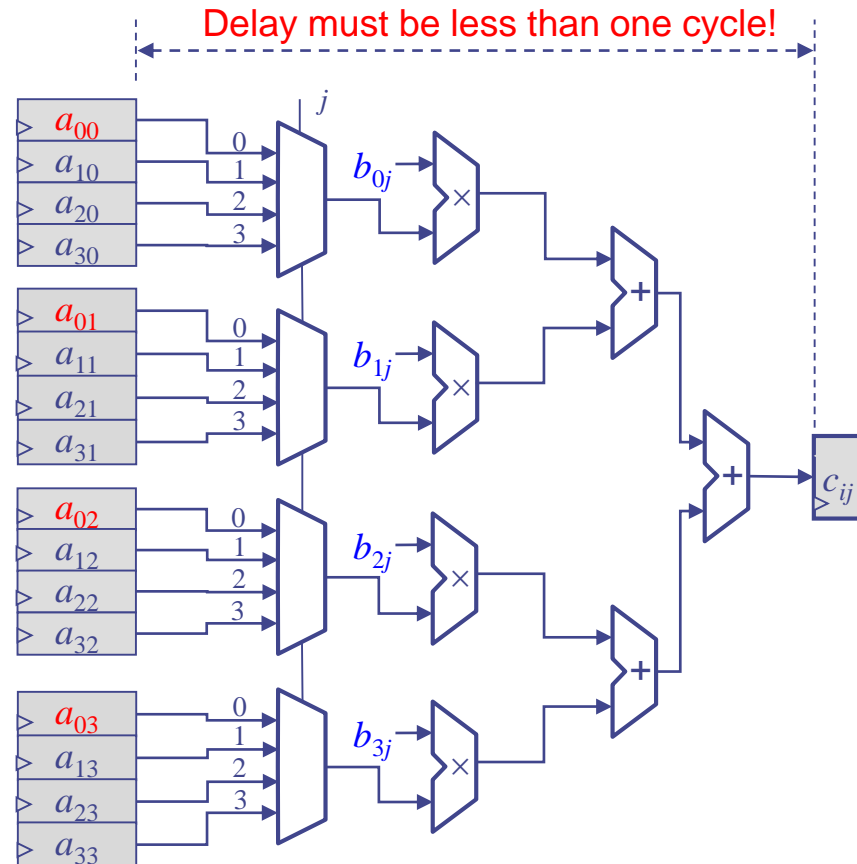
- ◆ A long arithmetic equation will be synthesized into a multi-level combinational circuit path:

```

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

```





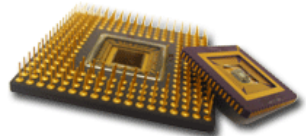
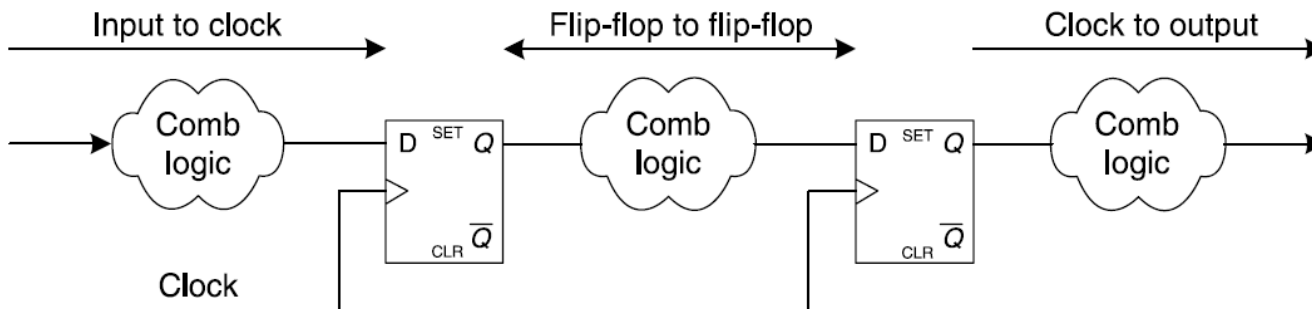
Setup Time and Hold Time

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).

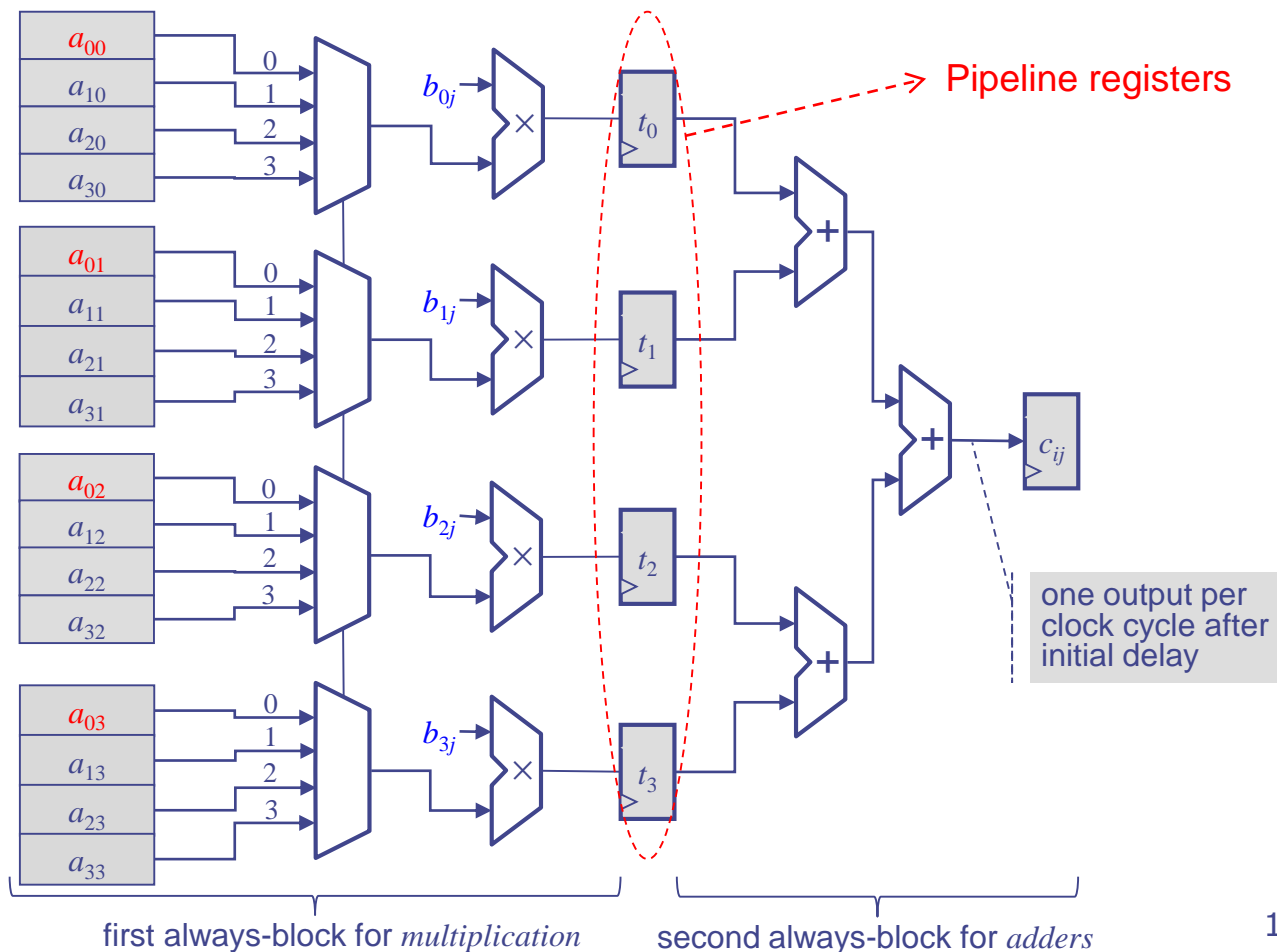




Breaking a Long Combinational Path

Lab 7

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





Check FPGA Resource Utilization

Lab 7

lab6 - [D:/xilinx/vivado/arty/2018/lab6/lab6.xpr] - Vivado 2018.2

File Edit Flow Tools Reports Window Layout View Help Quick Access write_bitstream Complete ✓ Default Layout

Flow Navigator

- Open Elaborated Design
- SYNTHESIS
 - Run Synthesis
 - Open Synthesized Design
- IMPLEMENTATION**
 - Run Implementation
 - Open Implemented Design**
 - Constraints Wizard
 - Edit Timing Constraints
 - Report Timing Summary
 - Report Clock Networks
 - Report Clock Interaction
 - Report Methodology
 - Report DRC
 - Report Noise
 - Report Utilization**
 - Report Power
 - Schematic
- PROGRAM AND DEBUG
 - Generate Bitstream
 - Open Hardware Manager

IMPLEMENTED DESIGN - xc7a35ticsg324-1L (active)

Sources Netlist x

- lab6
 - Nets (140)
 - Leaf Cells (96)
 - btn_db0 (debounce)

Source File Properties

sram.v

General Properties

Project Summary x **Device** x lab6.v x

Utilization x

Summary

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

utilization_1

Used 0.5 block of a 36-kbit BRAM.