
FIFO

This lab has you implement a simple FIFO memory.

BACKGROUND

Review the notes on memory.

TASK DESCRIPTION

You are to implement a FIFO with the following features:

- Can hold up to 7 data elements (data is 8 bits wide)
- Empty, Full flags

The module interface is given below:

```
module fifo(clk, reset, sclr, wren, rden, full, empty, din, dout );
input clk, reset, sclr, wren, rden;
input [7:0] din;
output full, empty;
output [7:0] dout;
```

The input/output definition is:

- *clk* – clock input
- *reset* – high true reset
- *wren* – asserted for a write; write input data present on *din* bus
- *rden* – asserted for a read; read input data present on *dout* bus
- *din* – data input bus for writes
- *dout* – data output bus for reads.
- *empty* – asserted when FIFO is empty
- *full* – asserted when FIFO is full
- *sclr* – when asserted, synchronously reset the internal read/write counters to 0.

Implementation: Memory block

To implement the FIFO, use a true dual port memory generated by the IP Coregen tool using block memory. See the appendix for the options needed to generate this memory component. You will put logic around this core memory to implement the FIFO. The block memory will contain 8 locations, of which only 7 can contain useful data at any given time. Use the *clk* signal for the two clock inputs on the true dual port memory (dual port memory will have a common clock).

The logic you create will have two counters: a *write* counter and a *read* counter. The write counter generates the memory address for write operations, and the read counter generates the memory address for a read operation. These counters are 3-bits wide since the FIFO logic is using a memory that contains 8 locations.

The FIFO is EMPTY when the write counter is equal to the read counter.

The FIFO is FULL when the next write address (write counter + 1) is equal to the read counter. This means that an N-element memory used as a FIFO can only hold N-1 active elements.

A write operation is performed when *wren* is asserted. This writes data to the current memory location addressed by the write counter and increments the write counter. If the FIFO is full, then a *wren* assertion is ignored (a write to a full FIFO changes no internal state).

A read operation is performed when *rden* is asserted. The reads data from the current memory location addressed by the read counter and increments the read counter. If the FIFO is empty, then a *rden* assertion is ignored (a read from an empty FIFO changes no internal state).

You do not need a FSM to implement this logic. Everything can be done as combinational logic attached to the write/read counters.

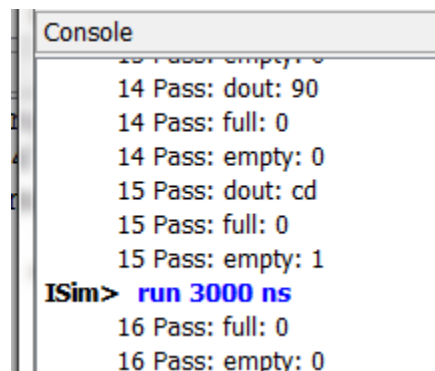
Associated files

The ZIP archive associated with this lab contains the following files:

- *fifo.v* -- complete this module
- *tb_fifo.v* – test bench for the FIFO
- *fifo_vectors.txt* – test vectors for FIFO
- *report.doc* – report file that needs to be filled out

fifo.v

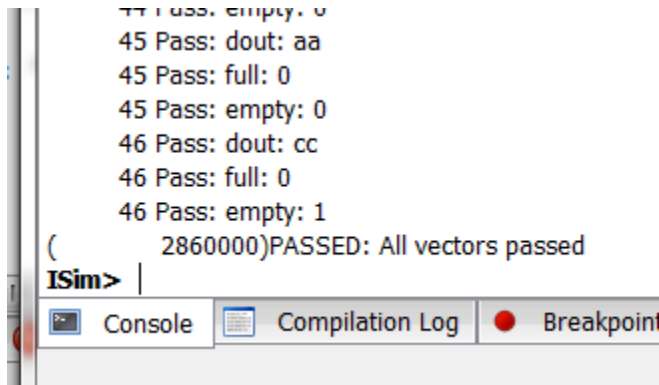
Create a project named *lab7_fifo* and finish implementing the *fifo.v* module (use the Spartan Family Xilinx device you have used for previous labs). Verify both behavioral simulation and Post-route simulation using the *tb_fifo.v* testbench. When running the testbench, you will need to execute the command 'run 3000 ns' to run all of the test vectors.



```

Console
13 Pass: empty: 0
14 Pass: dout: 90
14 Pass: full: 0
14 Pass: empty: 0
15 Pass: dout: cd
15 Pass: full: 0
15 Pass: empty: 1
ISim> run 3000 ns
16 Pass: full: 0
16 Pass: empty: 0

```



Fill out the requested information in the *report.doc* file.

Submission

For submission, create a directory named 'lab7_netid', i.e., (lab7_rbr5).

Copy your *lab7_fifo* project directory to this directory.

Copy your completed *report.doc* to this directory.

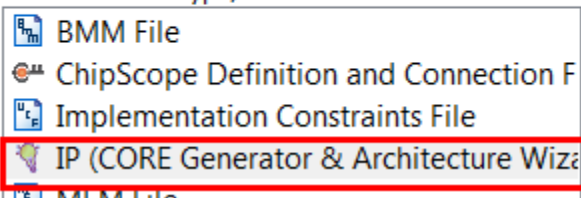
Create a ZIP archive of this directory and submit it.

Appendix: Parameters for generating the block memory

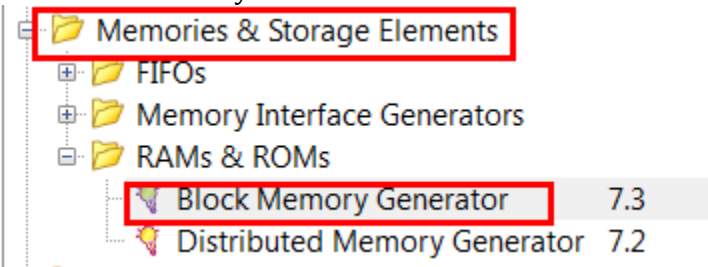
Add new source using the IP Coregen tool to make a block memory that will be used within your *fifo.v* module.

Select Source Type

Select source type, file name and its location.

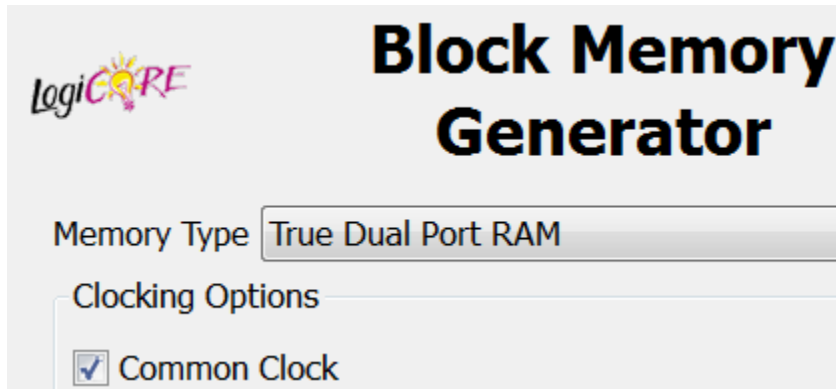


Select Block Memory Generator



Parameters: Page 1, use all defaults.

Parameters: Page 2, select 'True Dual Port RAM' with common clock. Use defaults for other parameters.



Parameters: Page 3, Write Width=8, Write Depth = 8 , Write First for both ports, and select the enable pins (ENA, ENB). Port A will be used for writing, and Port B for reading. When you write to PortA, then the ENA input must be asserted. When you read from Port B, then the ENB input must be asserted.

Port A Options

Memory Size

Write Width Range: 1..4608

Read Width:

Write Depth Range: 2..9011200

Read Depth: 8

Operating Mode

☒ Write First

☐ Read First

☐ No Change

Enable

☐ Always Enabled

☒ Use ENA Pin

Port B Options

Memory Size

Write Width

Read Width:

Write Depth: 8

Read Depth: 8

Operating Mode

☒ Write First

☐ Read First

☐ No Change

Enable

☐ Always Enabled

☒ Use ENB Pin

Parameters: Page 4 --- use all defaults.

Parameters Page 5 – use all defaults.

Parameters Page 6 – use all defaults.

Final interface looks like:

