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.

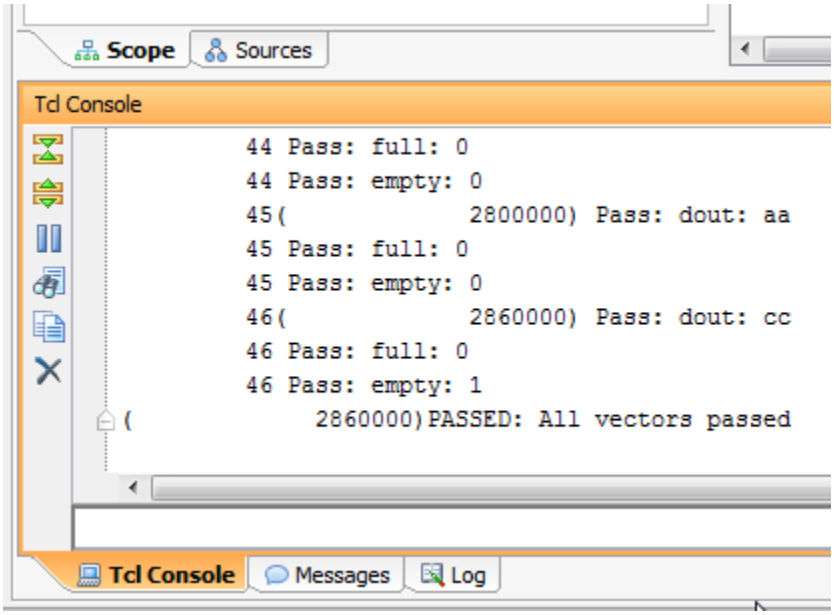
Associated files

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 *lab8_fifo* and finish implementing the *fifo.v* module. Verify both behavioral simulation and Post-implementation timing simulation using the *tb_fifo.v* testbench. When running the testbench, you will need to execute the command 'run 3 us' to run all of the test vectors.



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

Submission

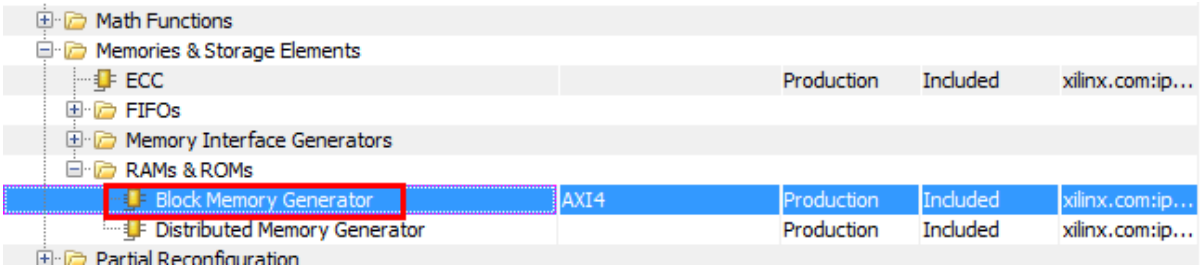
For submission, create a directory named 'lab8_*netid*', i.e., (lab8_rbr5).

Copy your *lab8_fifo* project directory to this directory.
Copy your completed *report.doc* to this directory.

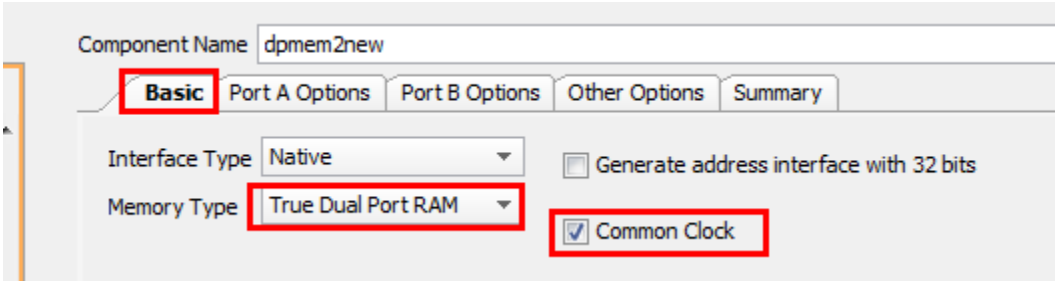
Create a ZIP archive of this directory and submit it.

Appendix: Parameters for generating the block memory

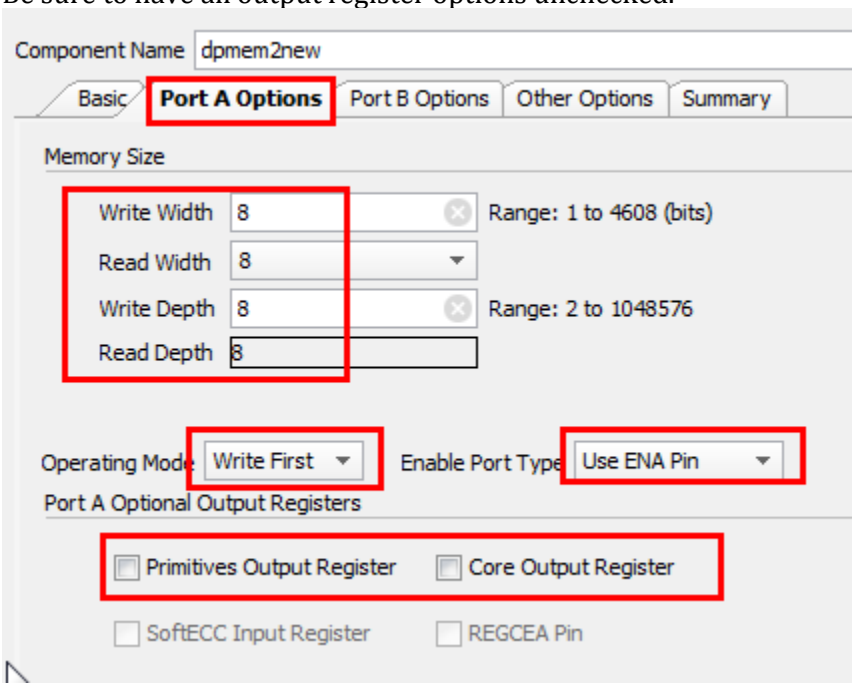
Use the IP Catalog to generate the block memory that is used within your *fifo.v* module. The screenshots below show the applicable options.

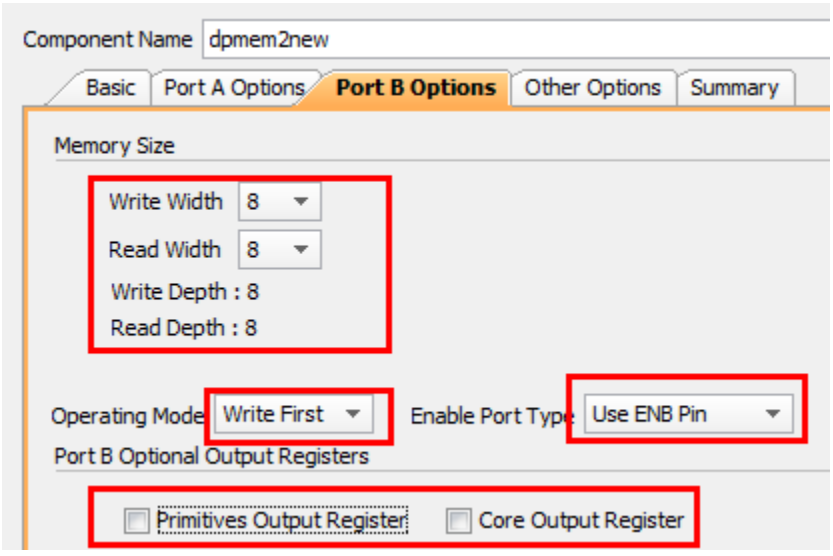


Select Block Memory Generator

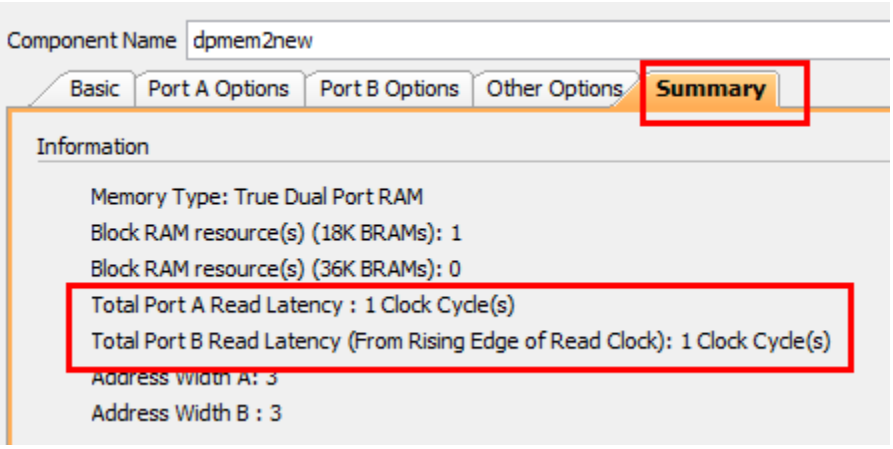


Be sure to have all output register options unchecked.

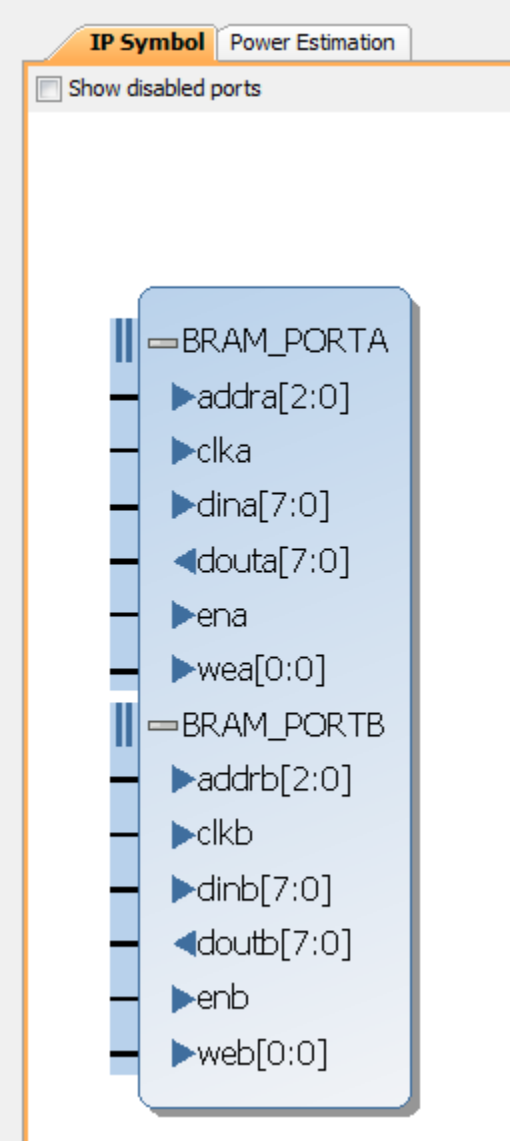


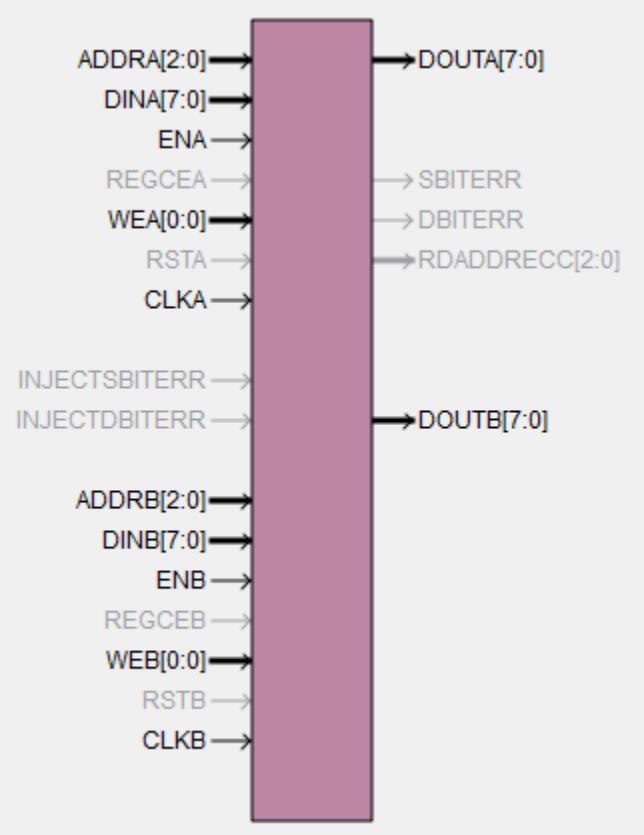


On the summary tab, ensure that the output read latency is 1 clock cycle for both ports. If it is greater than 1 clock cycle, then you did not uncheck the 'primitives output register' for either port A or port B.



Final Interface:


IP Symbol
Power Estimation
Show disabled ports
BRAM_PORTA
addra[2:0]
clka
dina[7:0]
douta[7:0]
ena
wea[0:0]
BRAM_PORTB
addrb[2:0]
clkb
dinb[7:0]
doutb[7:0]
enb
web[0:0]


ADDRA[2:0]
DINA[7:0]
ENA
REGCEA
WEA[0:0]
RSTA
CLKA
INJECTSBITERR
INJECTDBITERR
ADDRB[2:0]
DINB[7:0]
ENB
REGCEB
WEB[0:0]
RSTB
CLKB
DOUTA[7:0]
SBITERR
DBITERR
RDADDRECC[2:0]
DOUTB[7:0]