Parameterized FIFO in Verilog

Philip Tracton

July 4, 2018

Contents

1 FIFO Operations

This is a parameterized FIFO implemented in Verilog. The width of the data and the depth of the storage are the 2 parameters that can be modified. This FIFO will wrap around and over write data when full. The number_samples is the current number of data samples stored.

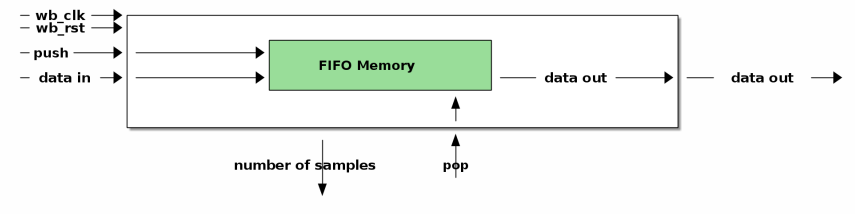
1.1 Parameters

- **dw** is the data width. This is used to size the internal memory of the FIFO and the data_in and data_out parameters. It should specify the exact size of the data since internally 1 is subtracted to zero the size. For example set this to 32 for 32 bits and data_in and data_out will be [31:0].
- **depth** is the number of samples stored in the FIFO. The number_samples output is based on this as it is just [$clog2(depth):0] bits wide.

1.2 Signals

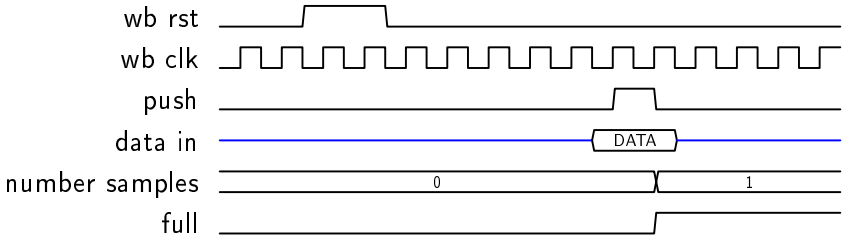
Signal Name	Direction	Size	Behavior
wb_clk	Input	1	Clock for synchronous behavior
wb_rst	Input	1	Synchronous reset
push	Input	1	Signal to write data into FIFO
pop	Input	1	Signal to read next data from FIFO
data_in	Input	dw	Data that push writes into the FIFO
data_out	Output	dw	The current output value of the FIFO
empty	Output	1	Goes high if there is no data in the FIFO
full	Output	1	Goes high if the FIFO is full
number_samples	Output	based on depth	The current number of samples in the FIFO

1.3 Block Diagram



1.4 Pushing Data

Pushing data is the act of writing information into the FIFO. This is a synchronous process. The data_in can be set up at any time. It does not get written to the FIFO until the PUSH signal is high and samples on a rising edge of the clock. At this time the data is then written into the FIFO and the number of samples is incremented. If the PUSH signal remains high, then data will be written to the FIFO on each clock edge. If this process fills in the FIFO, the FULL signal is asserted until data is popped out.

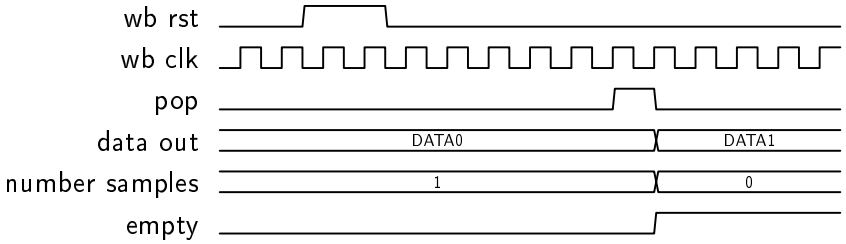



```
//
// Write data into FIFO memory when there is a push.
// Increment the write pointer and wrap around as needed.
//
always @(posedge wb_clk)
  if (wb_rst) begin
     for (i=0; i< depth; i=i+1)
       memory[i] <= 0;
     wr_ptr <= 0;
  end else begin
     if (push) begin
        memory[wr_ptr] <= data_in;
        wr_ptr <= wr_ptr + 1;
        if (wr_ptr >= depth-1) begin
           wr_ptr <= 0;
        end
     end
  end // else: !if(wb_rst)
```

1.5 Popping Data

Popping data is the process of moving the output to the next value. The data_out holds the current top of the FIFO and is always available for reading. Once this data is no longer needed, the POP process will get the

next value from the FIFO and make it the data output. The POP signal operates just like the PUSH signal. It is synchronous and will execute a POP on each clock edge while high. The empty signal is asserted once there is no more data in the FIFO.




```
//
// Always present the next sample to the output.
//
// If we pop (read) a sample increment and wrap
// around the read pointer if needed.
//
assign data_out = memory[rd_ptr];
always @(posedge wb_clk)
  if (wb_rst) begin
     rd_ptr <= 0;
  end else begin
     if (pop) begin
        rd_ptr <= rd_ptr + 1;
        if (rd_ptr >= depth-1) begin
           rd_ptr <= 0;
        end
     end
  end
```

2 FIFO Testing

2.1 Run Simulations

Several different simulators were used to verify the RTL. The test bench is self checking and indicates pass/fail for both individual tests and the overal test run.

2.1.1 Icarus Verilog

- This is the primary tool since it is free and runs everywhere. To run the simulation with this tool use ***make sim***. This will produce a dump.vcd that can be viewed with GtkWave and a fifo_iverilog.log

2.1.2 Modelsim

- Use ***make modelsim*** to execute the simulation via the modelsim command line options
- If the modelsim GUI is started, the modelsim.do file will run and produce the waveforms

2.1.3 Xcelium

- Use ***make xrun*** run the simulation with the Cadence Xcelium tools if you have access to them. This will produce a dump.vcd that van be viewed with Simvision.

2.2 Cleaning Up

- Use ***make clean*** to remove all produced output from any of the simulations or documentation tools.

2.3 Linting

- Use ***make lint*** to use verilator in it's lint-only mode on fifo.v. It will pass silently. There is only feedback if there is a problem.

3 FIFO Synthesis

3.1 Yosys

- Use the command ***make synthesis*** to synthesize the fifo.v file into a fifo_synth.v for Xilinx technology. This is a new tool that is being learned as this is developed. It will silently run and produce both a fifo_xilinx_synthesis.v and a fifo_yosys.log.