

More Lab 3b

CSE141L

5/23

dmem.v

- 'reset' Signal
 - For generating 'refused' signal
- Testbench Setup
 - Choose 'Falling Edge'
 - Set Properties → Simulation Run time

Simulation Setup

Initial Timing and Clock Wizard - Initialize Timing

The diagram shows a clock signal with a high period (purple arrow) and a low period (blue arrow). A green arrow indicates the maximum output delay from the clock high period. A grey arrow indicates the minimum input setup time before the clock low period.

Clock Timing Information
Inputs are assigned at "Input Setup Time" and outputs are checked at "Output Valid Delay".

Rising Edge Falling Edge

Dual Edge (DDR or DE1)

Clock High Time: 10 ns
Clock Low Time: 10 ns
Input Setup Time: ns
Output Valid Delay: ns
Offset: 100 ns

Clock Information
 Single Clock: clk
 Multiple Clocks
 Combinatorial (or internal clock)

Combinatorial Timing Information
Inputs are assigned, outputs are decoded then checked. A delay between inputs and outputs avoids assignment/checking conflicts.

Check Outputs: 50 ns After Inputs are Assigned
Assign Inputs: 50 ns After Outputs are Checked

Global Signals
 PRLD (CPLD) GSR (FPGA)
High for Initial: 100 ns

Initial Length of Test Bench: 3000 ns
Time Scale: ns
 Add Asynchronous Signal Support

More Info | < Back | Finish | Cancel

Process Properties

Category: ISE Simulator Properties

ISE Simulator Properties

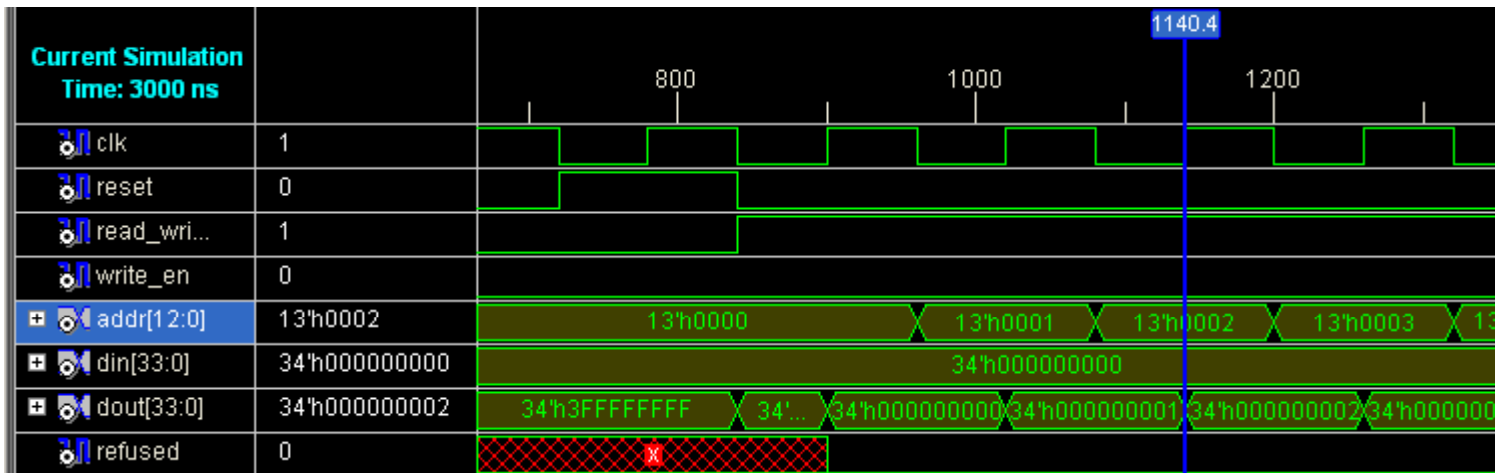
Property Name	Value
Testbench Model Target Language	Verilog
Use Custom Simulation Command File	<input type="checkbox"/>
Custom Simulation Command File	...
Incremental Compilation	<input checked="" type="checkbox"/>
Compile for HDL Debugging	<input checked="" type="checkbox"/>
Use Custom Compile File List	<input type="checkbox"/>
Custom Compile File List	...
Run for Specified Time	<input checked="" type="checkbox"/>
Simulation Run Time	3000 ns
Store All Signal Transitions During Simulation	<input type="checkbox"/>

VHDL Properties

Property display level: Advanced | Default

OK | Cancel | Apply | Help

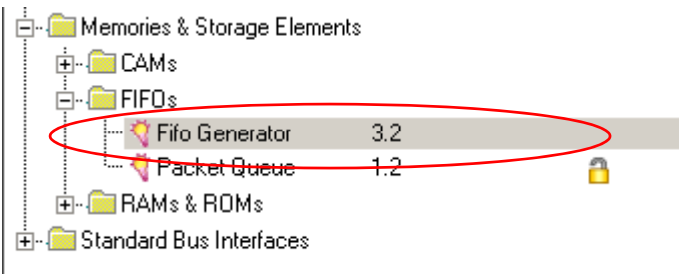
Behavioral Simulation



Correctly work at any cycle time constraint

*.mif file?

Generate a FIFO



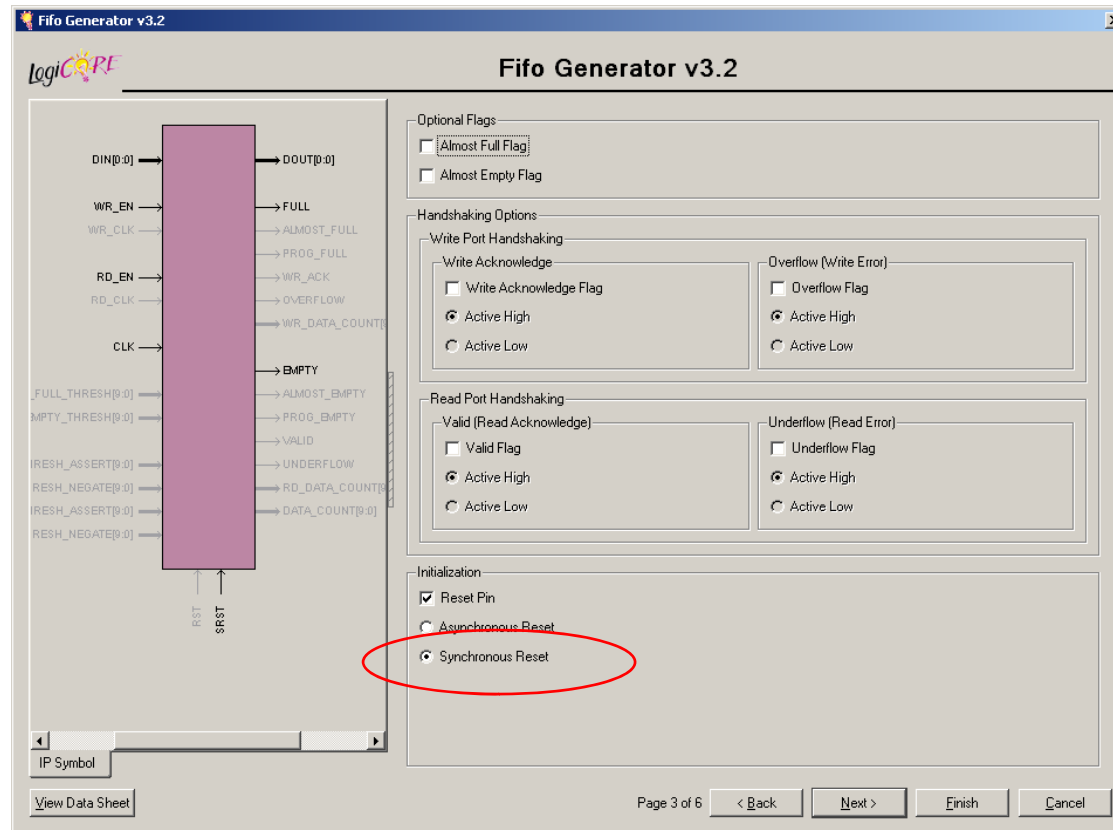
The 'Fifo Generator v3.2' configuration window is shown. It features several sections:

- Data Port Parameters:** Write Width (30), Write Depth (16), Read Width (30), and Read Depth (16). The 'Write Width' and 'Write Depth' fields are circled in red.
- Built-in FIFO Options:** Read Clock Frequency (MHz) and Write Clock Frequency (MHz), both set to 100.
- Optional Flags:** Includes checkboxes for 'Almost Full Flag' and 'Almost Empty Flag', both of which are unchecked.
- Handshaking Options:** Includes 'Write Port Handshaking' and 'Read Port Handshaking'. Under 'Write Port Handshaking', 'Write Acknowledge' is set to 'Active High'. Under 'Read Port Handshaking', 'Valid (Read Acknowledge)' is checked and set to 'Active High'. The 'Valid (Read Acknowledge)' checkbox and its 'Active High' radio button are circled in red.
- Initialization:** Includes a checked 'Reset Pin' option and 'Asynchronous Reset' and 'Synchronous Reset' radio buttons. The 'Synchronous Reset' radio button is circled in red.

The central part of the window shows a block diagram of the FIFO with various input and output signals. At the bottom, there are buttons for 'View Data Sheet', 'Page 3 of 6', '< Back', 'Next >', 'Finish', and 'Cancel'.

Generate a FIFO

- For other options, use default values



Verilog for SMIPSV1 control logic

```
// Set the control signals based on the decoder output

wire br_type = cs[7];
assign pc_mux_sel = ( br_type == br_pc4 ) ? 1'b0
                  : ( br_type == br_neq ) ? ~branch_cond_eq
                  : 1'bx;

assign op0_mux_sel      = cs[6];
assign op1_mux_sel      = cs[5];
assign wb_mux_sel       = cs[4];
assign rf_wen           = ( reset ? 1'b0 : cs[3] );
assign dmemreq_bits_rw = cs[2];
assign dmemreq_val      = ( reset ? 1'b0 : cs[1] );
wire  tohost_en         = ( reset ? 1'b0 : cs[0] );

// These control signals we can set directly from the instruction bits

assign rf_raddr0 = inst[25:21];
assign rf_raddr1 = inst[20:16];
assign rf_waddr  = inst[20:16];
assign inst_imm  = inst[15:0];

// We are always making an imemreq

assign imemreq_val = 1'b1;
```

Source: Krste's Slides from 6.375, MIT

Verilog for SMIPSV1 control logic

```
`define LW      32'b100011_?????_?????_?????_?????_?????
`define SW      32'b101011_?????_?????_?????_?????_?????
`define ADDIU   32'b001001_?????_?????_?????_?????_?????
`define BNE     32'b000101_?????_?????_?????_?????_?????

localparam cs_sz = 8;
reg [cs_sz-1:0] cs;

always @(*)
begin
  cs = {cs_sz{1'b0}};
  casez ( imemresp_bits_data )
    //                op0 mux  op1 mux  wb mux  rfile mreq  mreq  tohost
    //                br type sel    sel    sel    wen  r/w    val  en
    `ADDIU : cs = { br_pc4, op0_sx,  op1_rd0, wmx_alu, 1'b1, mreq_x, 1'b0, 1'b0 };
    `BNE   : cs = { br_neq, op0_sx2, op1_pc4, wmx_x,   1'b0, mreq_x, 1'b0, 1'b0 };
    `LW    : cs = { br_pc4, op0_sx,  op1_rd0, wmx_mem, 1'b1, mreq_r, 1'b1, 1'b0 };
    `SW    : cs = { br_pc4, op0_sx,  op1_rd0, wmx_x,   1'b0, mreq_w, 1'b1, 1'b0 };
    `MTC0  : cs = { br_pc4, op0_x,   op1_x,   wmx_x,   1'b0, mreq_x, 1'b0, 1'b1 };
  endcase
end
```

**casez or casex performs simple
pattern matching and can be very
useful when implementing decoders**

Source: Krste's Slides from 6.375, MIT

casez and casex

```
1 module case_compare;
2
3 reg sel;
4
5 initial begin
6   #1 $display ("\n Driving 0");
7   sel = 0;
8   #1 $display ("\n Driving 1");
9   sel = 1;
10  #1 $display ("\n Driving x");
11  sel = 1'bx;
12  #1 $display ("\n Driving z");
13  sel = 1'bz;
14  #1 $finish;
15 end
16
17 always @ (sel)
18 case (sel)
19   1'b0 : $display("Normal : Logic 0 on sel");
20   1'b1 : $display("Normal : Logic 1 on sel");
21   1'bx : $display("Normal : Logic x on sel");
22   1'bz : $display("Normal : Logic z on sel");
23 endcase
24
25 always @ (sel)
26 casex (sel)
27   1'b0 : $display("CASEX : Logic 0 on sel");
28   1'b1 : $display("CASEX : Logic 1 on sel");
29   1'bx : $display("CASEX : Logic x on sel");
30   1'bz : $display("CASEX : Logic z on sel");
31 endcase
32
33 always @ (sel)
34 casez (sel)
35   1'b0 : $display("CASEZ : Logic 0 on sel");
36   1'b1 : $display("CASEZ : Logic 1 on sel");
37   1'bx : $display("CASEZ : Logic x on sel");
38   1'bz : $display("CASEZ : Logic z on sel");
39 endcase
40
41 endmodule
```

Driving 0
Normal : Logic 0 on sel
CASEX : Logic 0 on sel
CASEZ : Logic 0 on sel

Driving 1
Normal : Logic 1 on sel
CASEX : Logic 1 on sel
CASEZ : Logic 1 on sel

Driving x
Normal : Logic x on sel
CASEX : Logic 0 on sel
CASEZ : Logic x on sel

Driving z
Normal : Logic z on sel
CASEX : Logic 0 on sel
CASEZ : Logic 0 on sel

casez and casex are synthesizable in Xilinx

Only the first matched item in case statement is executed

Any Questions?