# 6.111 Roadmap

- Previously on 6.111…
  - The digital abstraction, digital signaling
  - Combinational logic, Verilog
  - Sequential logic, Verilog, synchronization issues
  - FSMs, major/minor organization & signaling
  - Clocking strategies, DCM resource
- In this episode…
  - Internal memories

# Memories: a practical primer

- The good news: huge selection of technologies
  - Small & faster vs. large & slower
  - Every year capacities go up and prices go down
  - New kid on the block: high density, fast flash memories
    - Non-volatile, read/write, no moving parts! (robust, efficient)
- The bad news: perennial system bottleneck
  - Latencies (access time) haven't kept pace with cycle times
  - Separate technology from logic, so must communicate between silicon, so physical limitations (# of pins, R's and C's and L's) limit bandwidths
    - New hopes: capacitive interconnect, 3D IC's
  - Likely the limiting factor in cost & performance of many digital systems: designers spend a lot of time figuring out how to keep memories running at peak bandwidth
  - "It's the memory, stupid"

# Memories in Verilog

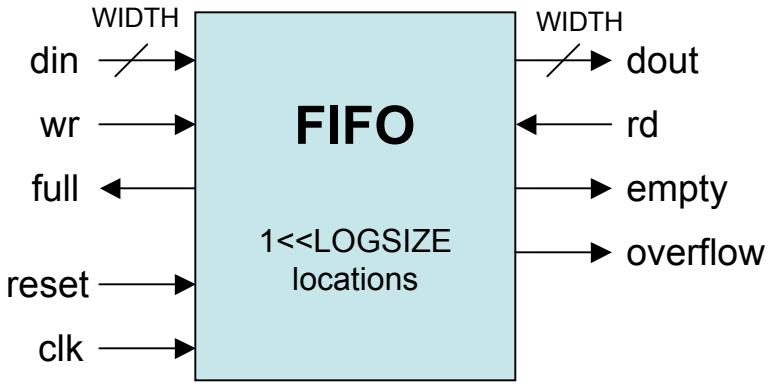
- `reg bit;  // a single register`
- `reg [31:0] word;  // a 32-bit register`
- `reg [31:0] array[15:0];  // 16 32-bit regs`

- 
```
wire [31:0] read_data,write_data;
wire [3:0] index;

// combinational (asynch) read
assign read_data = array[index];

// clocked (synchronous) write
always @ (posedge clock)
    array[index] <= write_data;
```

# Multi-port Memories (aka regfiles)

```
reg [31:0] regfile[30:0]; // 31 32-bit words

// Beta register file: 2 read ports, 1 write
wire [4:0] ra1,ra2,wa;
wire [31:0] rd1,rd2,wd;

assign ra1 = inst[20:16];
assign ra2 = ra2sel ? inst[25:21] : inst[15:11];
assign wa = wasel ? 5'd30 : inst[25:21];

// read ports
assign rd1 = (ra1 == 31) ? 0 : regfile[ra1];
assign rd2 = (ra2 == 31) ? 0 : regfile[ra2];
// write port
always @ (posedge clk)
  if (werf) regfile[wa] <= wd;

assign z = ~| rd1;   // used in BEQ/BNE instructions
```

# FIFOs



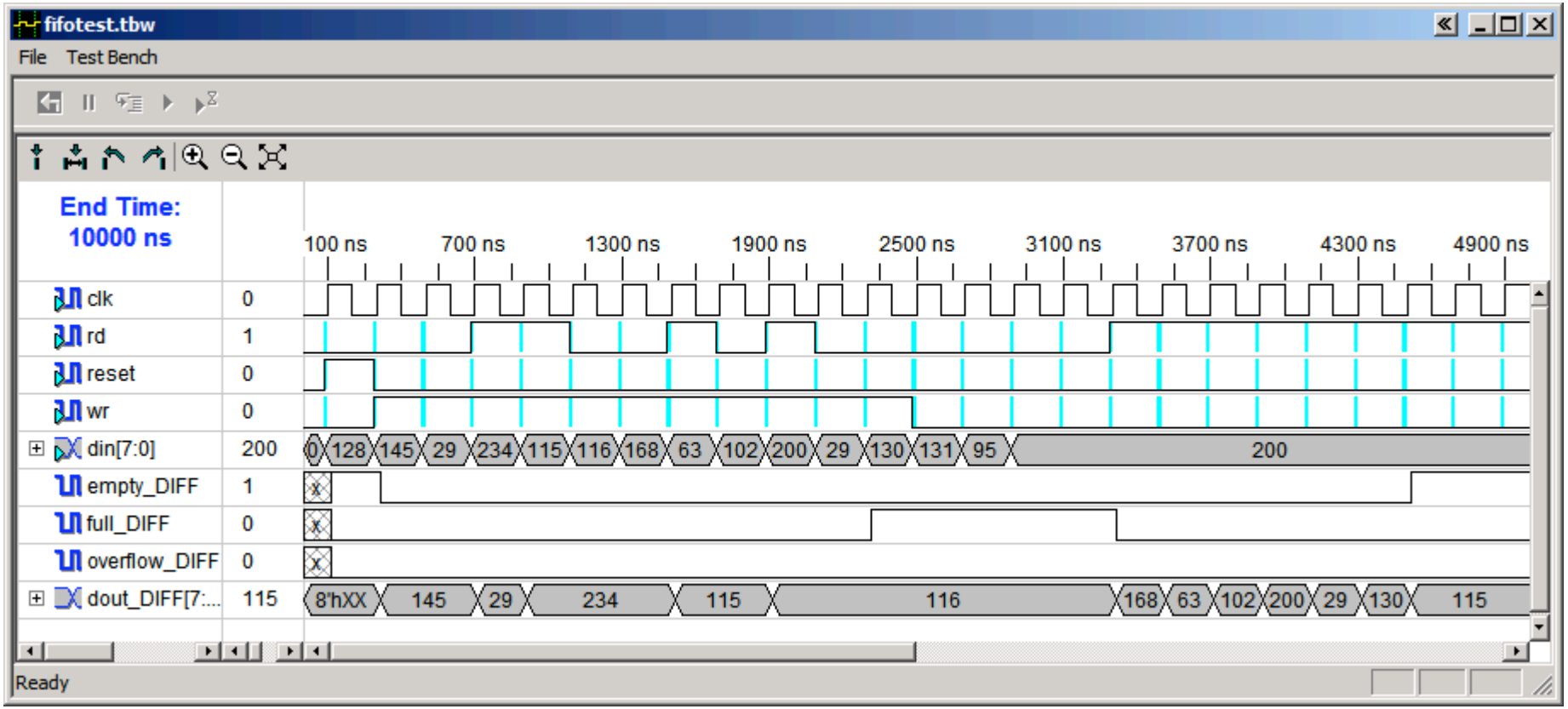

```
// a simple synchronous FIFO (first-in first-out) buffer
// Parameters:
//    LOGSIZE   (parameter) FIFO has 1<<LOGSIZE elements
//    WIDTH     (parameter) each element has WIDTH bits
// Ports:
//    clk       (input) all actions triggered on rising edge
//    reset     (input) synchronously empties fifo
//    din       (input, WIDTH bits) data to be stored
//    wr        (input) when asserted, store new data
//    full      (output) asserted when FIFO is full
//    dout      (output, WIDTH bits) data read from FIFO
//    rd        (input) when asserted, removes first element
//    empty     (output) asserted when fifo is empty
//    overflow (output) asserted when WR but no room, cleared on next RD
module fifo(clk,reset,din,wr,full,dout,rd,empty,overflow);
  parameter LOGSIZE = 2;    // default size is 4 elements
  parameter WIDTH = 4;      // default width is 4 bits
  ...
endmodule
```

# FIFO.V

```
// a simple synchronous FIFO (first-in first-out) buffer
// Parameters:
//    LOGSIZE  (parameter) FIFO has 1<<LOGSIZE elements
//    WIDTH    (parameter) each element has WIDTH bits
// Ports:
//    clk      (input) all actions triggered on rising edge
//    reset    (input) synchronously empties fifo
//    din      (input, WIDTH bits) data to be stored
//    wr       (input) when asserted, store new data
//    full     (output) asserted when FIFO is full
//    dout     (output, WIDTH bits) data read from FIFO
//    rd       (input) when asserted, removes first element
//    empty    (output) asserted when fifo is empty
//    overflow (output) asserted when WR but no room, cleared on next RD
module fifo(clk,reset,din,wr,full,dout,rd,empty,overflow);
  parameter LOGSIZE = 2;   // default size is 4 elements
  parameter WIDTH = 4;     // default width is 4 bits

  parameter SIZE = 1 << LOGSIZE;  // compute size

  input clk,reset,rd,wr;
  input [WIDTH-1:0] din;
  output [WIDTH-1:0] dout;
  output full,empty,overflow;

  reg [WIDTH-1:0] fifo[SIZE-1:0];   // fifo data stored here
  reg overflow;                     // true if WR but no room, cleared on RD
  reg [LOGSIZE-1:0] wptr,rptr;  // fifo write and read pointers

  wire [LOGSIZE-1:0] wptr_inc = wptr + 1;

  assign empty = (wptr == rptr);
  assign full = (wptr_inc == rptr);
  assign dout = fifo[rptr];

  always @ (posedge clk) begin
    if (reset) begin
      wptr <= 0;
      rptr <= 0;
      overflow <= 0;
    end
    else if (wr) begin
      // store new data into the fifo
      fifo[wptr] <= din;
      wptr <= wptr_inc;
      overflow <= overflow | (wptr_inc == rptr);
    end

    // bump read pointer if we're done with current value.
    // RD also resets the overflow indicator
    if (rd && (!empty || overflow)) begin
      rptr <= rptr + 1;
      overflow <= 0;
    end
  end
endmodule
```

# FIFOs in action

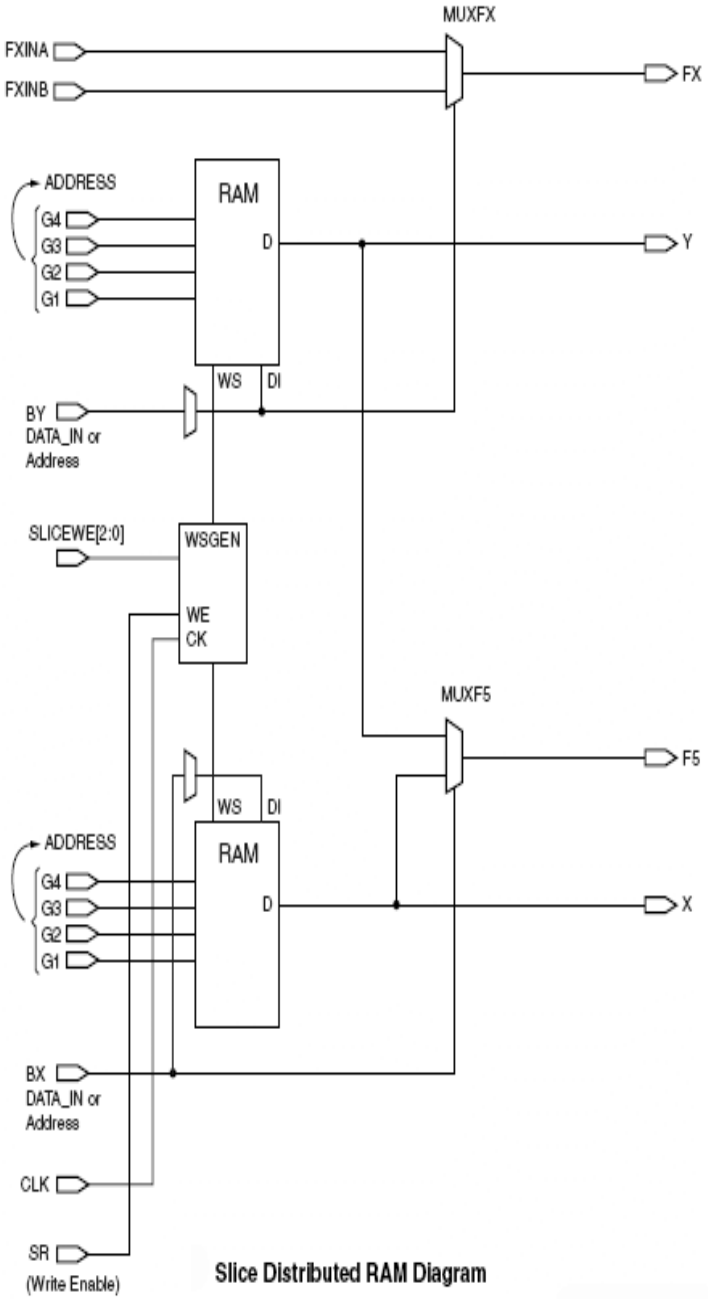
```
// make a fifo with 8 8-bit locations
fifo f8x8(clk,reset,din,wr,full,dout,rd,empty,overflow);
defparam f8x8.LOGSIZE = 3;
defparam f8x8.WIDTH = 8;
```



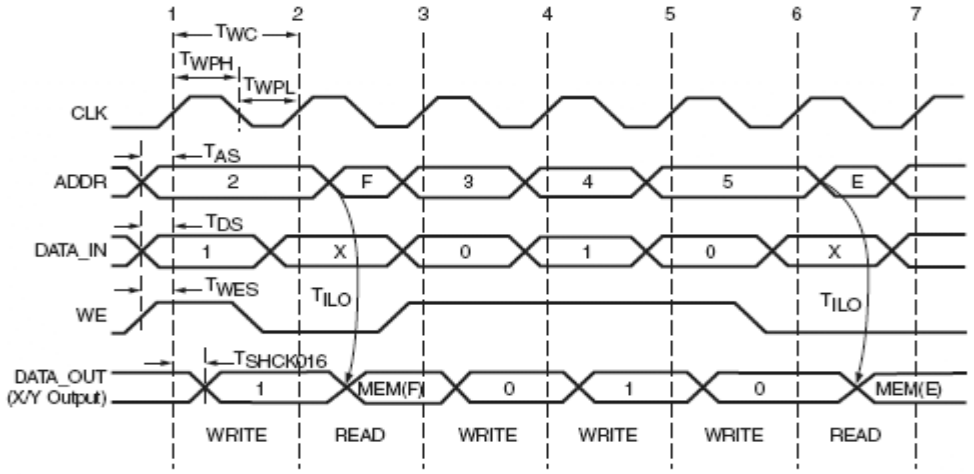
# FPGA memory implementation

- Regular registers in logic blocks
  - Piggy use of resources, but convenient & fast if small
- [Xilinx Vertex II] use the LUTs:
  - Single port: 16x(1,2,4,8), 32x(1,2,4,8), 64x(1,2), 128x1
  - Dual port (1 R/W, 1R): 16x1, 32x1, 64x1
  - Can fake extra read ports by cloning memory: all clones are written with the same addr/data, but each clone can have a different read address
- [Xilinx Vertex II] use block ram:
  - 18K bits: 16Kx1, 8Kx2, 4Kx4
    with parity: 2Kx(8+1), 1Kx(16+2), 512x(32+4)
  - Single or dual port
  - Pipelined (clocked) operations
  - Labkit XCV2V6000: 144 BRAMs, 2952K bits total

# LUT-based RAMs




Slice Distributed RAM Diagram

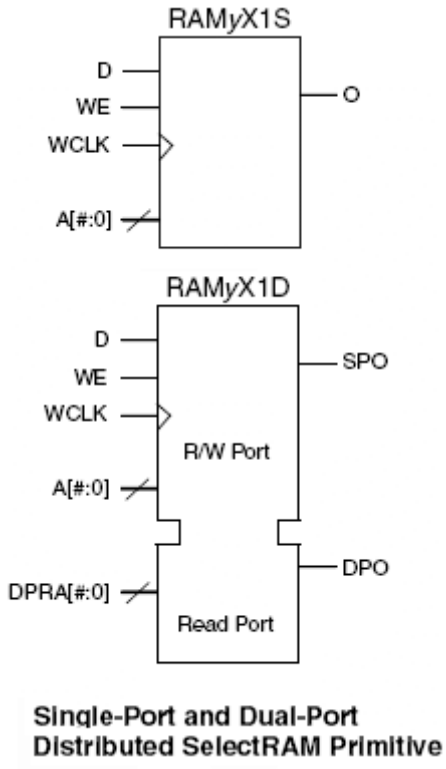



Slice Distributed RAM Timing Diagram

**CLB Distributed RAM Switching Characteristics**

| Description | Symbol | Speed Grade | | | Units |
|---|---|---|---|---|---|
| | | -6 | -5 | -4 | |
| **Sequential Delays** | | | | | |
| Clock CLK to X/Y outputs (WE active) in 16 x 1 mode | $T_{SHCKO16}$ | 1.63 | 1.79 | 2.05 | ns, Max |
| Clock CLK to X/Y outputs (WE active) in 32 x 1 mode | $T_{SHCKO32}$ | 1.97 | 2.17 | 2.49 | ns, Max |
| Clock CLK to F5 output | $T_{SHCKOF5}$ | 1.77 | 1.94 | 2.23 | ns, Max |
| **Setup and Hold Times Before/After Clock CLK** | | | | | |
| BX/BY data inputs (DIN) | $T_{DS}/T_{DH}$ | 0.53/–0.09 | 0.58/–0.10 | 0.67/–0.11 | ns, Min |
| F/G address inputs | $T_{AS}/T_{AH}$ | 0.40/ 0.00 | 0.44/ 0.00 | 0.50/ 0.00 | ns, Min |
| SR input (WS) | $T_{WES}/T_{WEH}$ | 0.42/–0.01 | 0.46/–0.01 | 0.53/–0.01 | ns, Min |
| **Clock CLK** | | | | | |
| Minimum Pulse Width, High | $T_{WPH}$ | 0.57 | 0.63 | 0.72 | ns, Min |
| Minimum Pulse Width, Low | $T_{WPL}$ | 0.57 | 0.63 | 0.72 | ns, Min |
| Minimum clock period to meet address write cycle time | $T_{WC}$ | 1.14 | 1.25 | 1.44 | ns, Min |
| **Combinatorial Delays** | | | | | |
| 4-input function: F/G inputs to X/Y outputs | $T_{ILO}$ | 0.35 | 0.39 | 0.44 | ns, Max |

# LUT-based RAM Modules




Single-Port and Dual-Port Distributed SelectRAM Primitive

**Single-Port and Dual-Port Distributed SelectRAM**

| Primitive | RAM Size | Type | Address Inputs |
|---|---|---|---|
| RAM16X1S | 16 bits | single-port | A3, A2, A1, A0 |
| RAM32X1S | 32 bits | single-port | A4, A3, A2, A1, A0 |
| RAM64X1S | 64 bits | single-port | A5, A4, A3, A2, A1, A0 |
| RAM128X1S | 128 bits | single-port | A6, A5, A4, A3, A2, A1, A0 |
| RAM16X1D | 16 bits | dual-port | A3, A2, A1, A0 |
| RAM32X1D | 32 bits | dual-port | A4, A3, A2, A1, A0 |
| RAM64X1D | 64 bits | dual-port | A5, A4, A3, A2, A1, A0 |

**Wider Library Primitives**

| Primitive | RAM Size | Data Inputs | Address Inputs | Data Outputs |
|---|---|---|---|---|
| RAM16x2S | 16 x 2-bit | D1, D0 | A3, A2, A1, A0 | O1, O0 |
| RAM32X2S | 32 x 2-bit | D1, D0 | A4, A3, A2, A1, A0 | O1, O0 |
| RAM64X2S | 64 x 2-bit | D1, D0 | A5, A4, A3, A2, A1, A0 | O1, O0 |
| RAM16X4S | 16 x 4-bit | D3, D2, D1, D0 | A3, A2, A1, A0 | O3, O2, O1, O0 |
| RAM32X4S | 32 x 4-bit | D3, D2, D1, D0 | A4,A3, A2, A1, A0 | O3, O2, O1, O0 |
| RAM16X8S | 16 x 8-bit | D <7:0> | A3, A2, A1, A0 | O <7:0> |
| RAM32X8S | 32 x 8-bit | D <7:0> | A4,A3, A2, A1, A0 | O <7:0> |

```
// instantiate a LUT-based RAM module
RAM16X1S mymem(.D(din),.O(dout),.WE(we),.WCLK(clock_27mhz),
               .A0(a[0]),.A1(a[1]),.A2(a[2]),.A3(a[3]));
defparam mymem.INIT = 16'b0110111100110101100;  // msb first
```

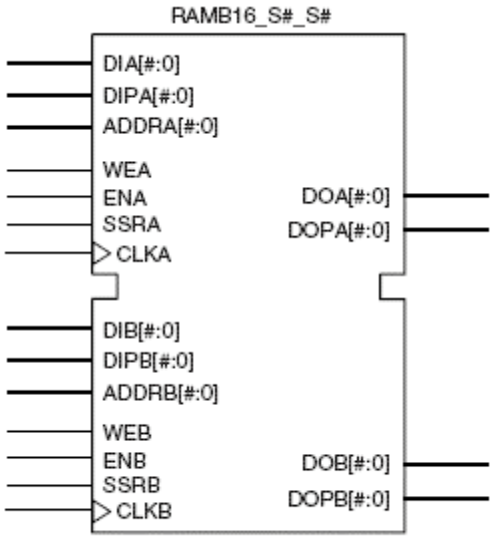
# Tools will often build these for you…

## From Lab 2:

```
reg [7:0] segments;
always @ (switch[3:0]) begin
  case (switch[3:0])
  4'h0: segments[6:0] = 7'b0111111;
  4'h1: segments[6:0] = 7'b0000110;
  4'h2: segments[6:0] = 7'b1011011;
  4'h3: segments[6:0] = 7'b1001111;
  4'h4: segments[6:0] = 7'b1100110;
  4'h5: segments[6:0] = 7'b1101101;
  4'h6: segments[6:0] = 7'b1111101;
  4'h7: segments[6:0] = 7'b0000111;
  4'h8: segments[6:0] = 7'b1111111;
  4'h9: segments[6:0] = 7'b1100111;
  4'hA: segments[6:0] = 7'b1110111;
  4'hB: segments[6:0] = 7'b1111100;
  4'hC: segments[6:0] = 7'b1011000;
  4'hD: segments[6:0] = 7'b1011110;
  4'hE: segments[6:0] = 7'b1111001;
  4'hF: segments[6:0] = 7'b1110001;
  default: segments[6:0] = 7'b00000000;
  endcase
  segments[7] = 1'b0;  // decimal point
end
```

```
=========================================================
*                  HDL Synthesis                        *
=========================================================

Synthesizing Unit <lab2_2>.
    Related source file is "../lab2_2.v".
    ...
    Found 16x7-bit ROM for signal <$n0000>.
    ...
    Summary:
            inferred   1 ROM(s).
        ...
Unit <lab2_2> synthesized.

=========================================================
Timing constraint: Default path analysis
Total number of paths / destination ports: 28 / 7
---------------------------------------------------------
Delay:               7.244ns (Levels of Logic = 3)
Source:            switch<3> (PAD)
Destination:       user1<0> (PAD)

Data Path: switch<3> to user1<0>
                     Gate     Net
Cell:in->out fanout Delay   Delay  Logical Name
-------------------------------  ------------
IBUF:I->O       7   0.825   1.102  switch_3_IBUF
LUT4:I0->O      1   0.439   0.517  Mrom__n0000_inst_lut4_01
OBUF:I->O           4.361          user1_0_OBUF
-------------------------------------
Total               7.244ns (5.625ns logic, 1.619ns route)
                            (77.7% logic, 22.3% route)
```

# Block Memories (BRAMs)



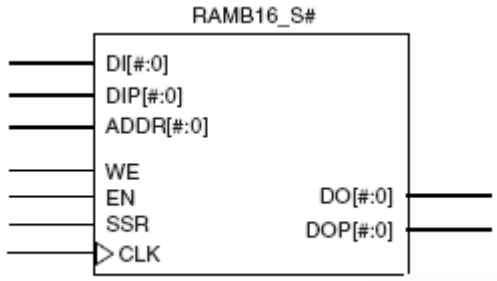

Dual-Port Block RAM Primitive

$(W_{DATA} + W_{PARITY})*(LOCATIONS) = 18K$ bits

1,2,4

16K,8K,4K,2K,1K,512

1
2
4
8
16
32




Single-Port Block RAM Primitive

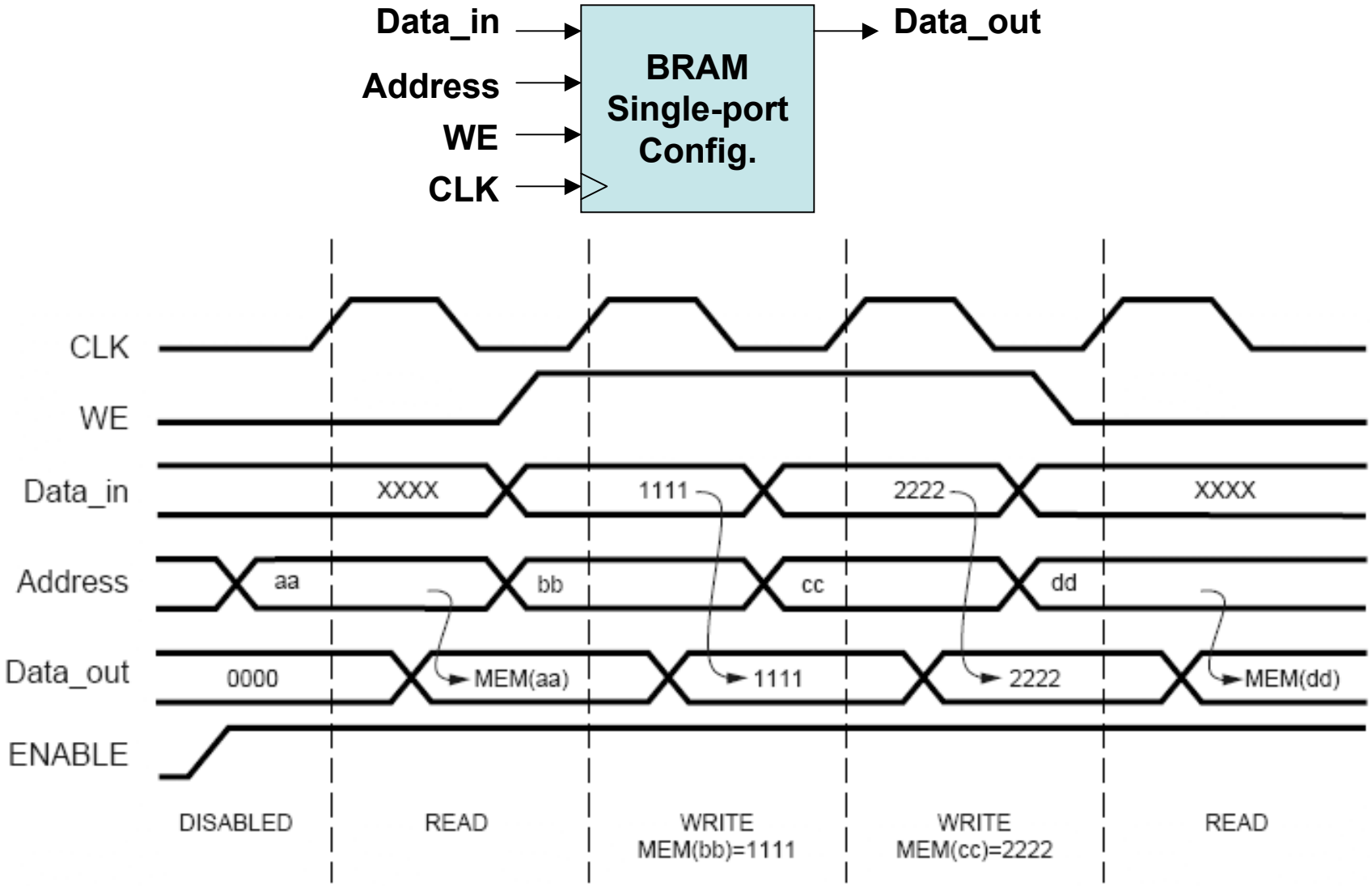
**Dual-Port Block RAM Primitives**

| Primitive | Port A Width | Port B Width |
|---|---|---|
| RAMB16_S1_S1 | 1 | 1 |
| RAMB16_S1_S2 | | 2 |
| RAMB16_S1_S4 | | 4 |
| RAMB16_S1_S9 | | (8+1) |
| RAMB16_S1_S18 | | (16+2) |
| RAMB16_S1_S36 | | (32+4) |
| RAMB16_S2_S2 | 2 | 2 |
| RAMB16_S2_S4 | | 4 |
| RAMB16_S2_S9 | | (8+1) |
| RAMB16_S2_S18 | | (16+2) |
| RAMB16_S2_S36 | | (32+4) |
| RAMB16_S4_S4 | 4 | 4 |
| RAMB16_S4_S9 | | (8+1) |
| RAMB16_S4_S18 | | (16+2) |
| RAMB16_S4_S36 | | (32+4) |
| RAMB16_S9_S9 | (8+1) | (8+1) |
| RAMB16_S9_S18 | | (16+2) |
| RAMB16_S9_S36 | | (32+4) |
| RAMB16_S18_S18 | (16+2) | (16+2) |
| RAMB16_S18_S36 | | (32+4) |
| RAMB16_S36_S36 | (32+4) | (32+4) |

**Single-Port Block RAM Primitives**

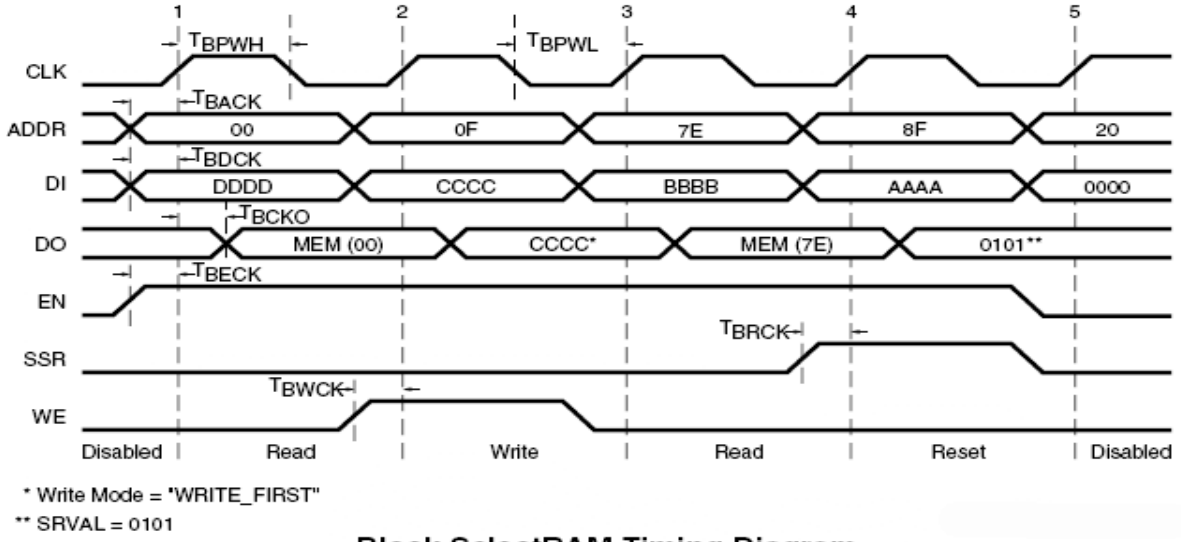
| Primitive | Port Width |
|---|---|
| RAMB16_S1 | 1 |
| RAMB16_S2 | 2 |
| RAMB16_S4 | 4 |
| RAMB16_S9 | (8+1) |
| RAMB16_S18 | (16+2) |
| RAMB16_S36 | (32+4) |

# BRAM Operation




**Source: Xilinx App Note 463**

# BRAM timing



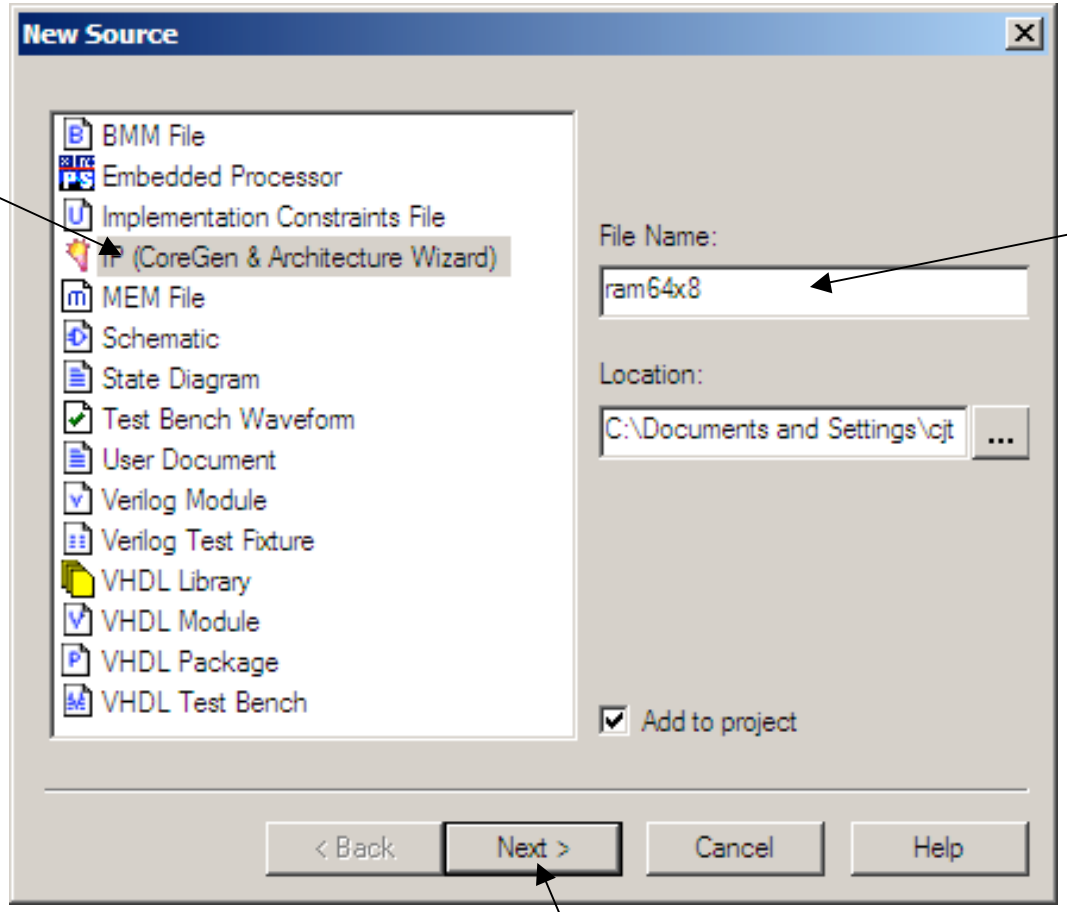

**Block SelectRAM Timing Diagram**

**Block SelectRAM Switching Characteristics**

| Description | Symbol | Speed Grade | | | Units |
|---|---|---|---|---|---|
| | | -6 | -5 | -4 | |
| **Sequential Delays** | | | | | |
| Clock CLK to DOUT output | $T_{BCKO}$ | 2.10 | 2.31 | 2.65 | ns, Max |
| **Setup and Hold Times Before Clock CLK** | | | | | |
| ADDR inputs | $T_{BACK}/T_{BCKA}$ | 0.29/ 0.00 | 0.32/ 0.00 | 0.36/ 0.00 | ns, Min |
| DIN inputs | $T_{BDCK}/T_{BCKD}$ | 0.29/ 0.00 | 0.32/ 0.00 | 0.36/ 0.00 | ns, Min |
| EN input | $T_{BECK}/T_{BCKE}$ | 0.95/–0.46 | 1.04/–0.50 | 1.20/–0.58 | ns, Min |
| RST input | $T_{BRCK}/T_{BCKR}$ | 1.31/–0.71 | 1.44/–0.78 | 1.65/–0.90 | ns, Min |
| WEN input | $T_{BWCK}/T_{BCKW}$ | 0.57/–0.19 | 0.63/–0.21 | 0.72/–0.25 | ns, Min |
| **Clock CLK** | | | | | |
| CLKA to CLKB setup time for different ports | $T_{BCCS}$ | 1.0 | 1.0 | 1.0 | ns, min |
| Minimum Pulse Width, High | $T_{BPWH}$ | 1.17 | 1.29 | 1.48 | ns, Min |
| Minimum Pulse Width, Low | $T_{BPWL}$ | 1.17 | 1.29 | 1.48 | ns, Min |

# Using BRAMs (eg, a 64Kx8 ram)

- From menus: Project → New Source…




Select "IP"

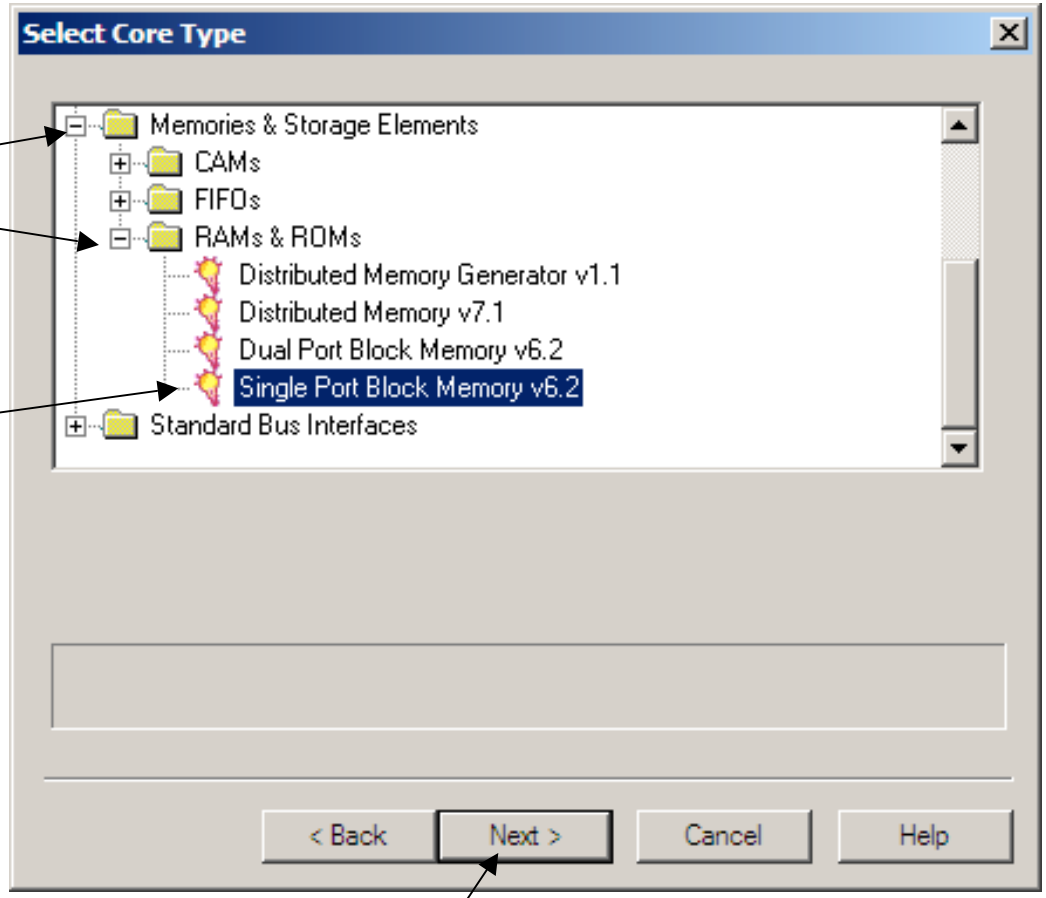
Fill in name

Click "Next" when done…

# BRAM Example

Click open folders

Select "Single Port Block Memory"




Click "Next" and then "Finish" on next window

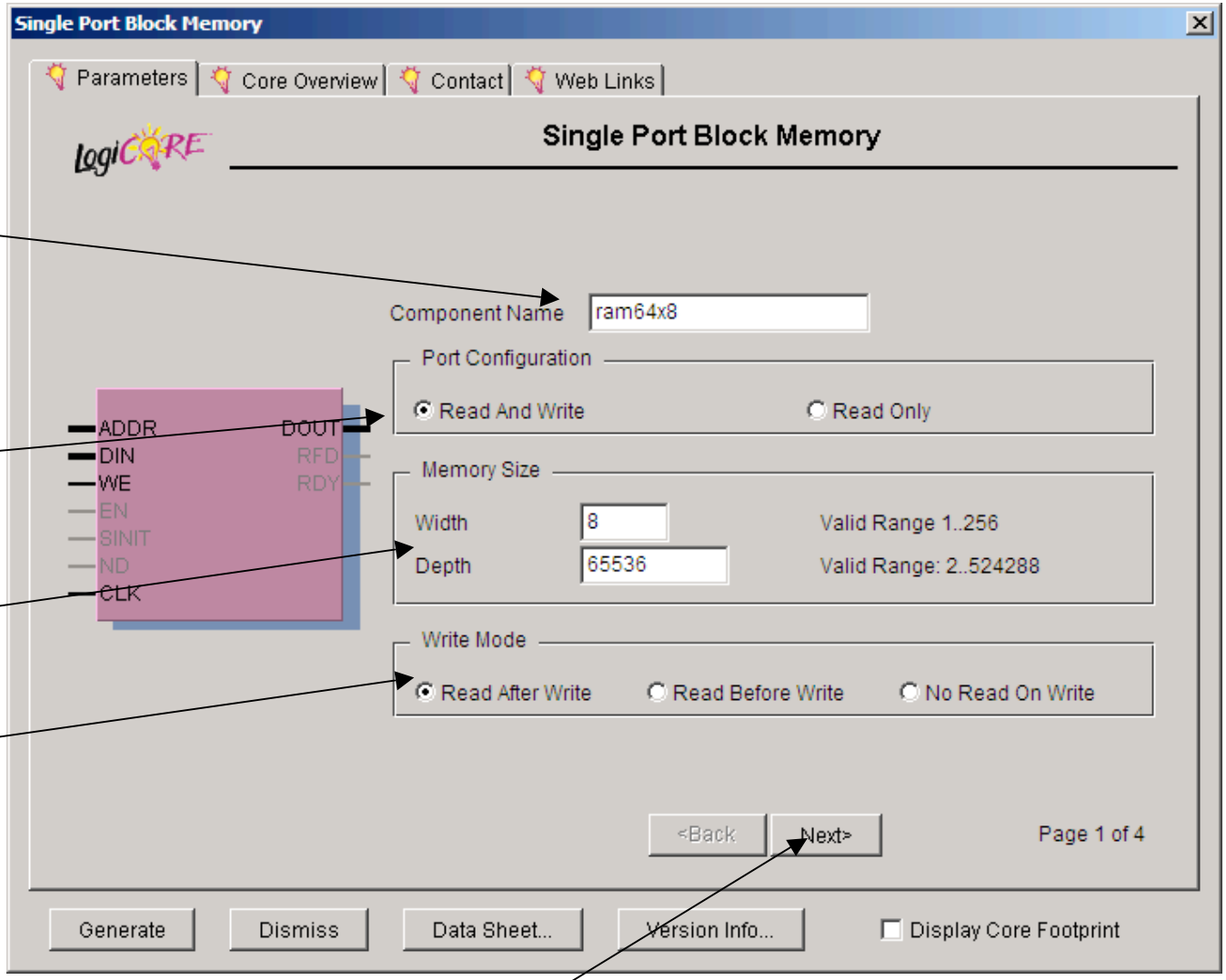
# BRAM Example

Fill in name (again?!)

Select RAM vs ROM

Fill in width & depth

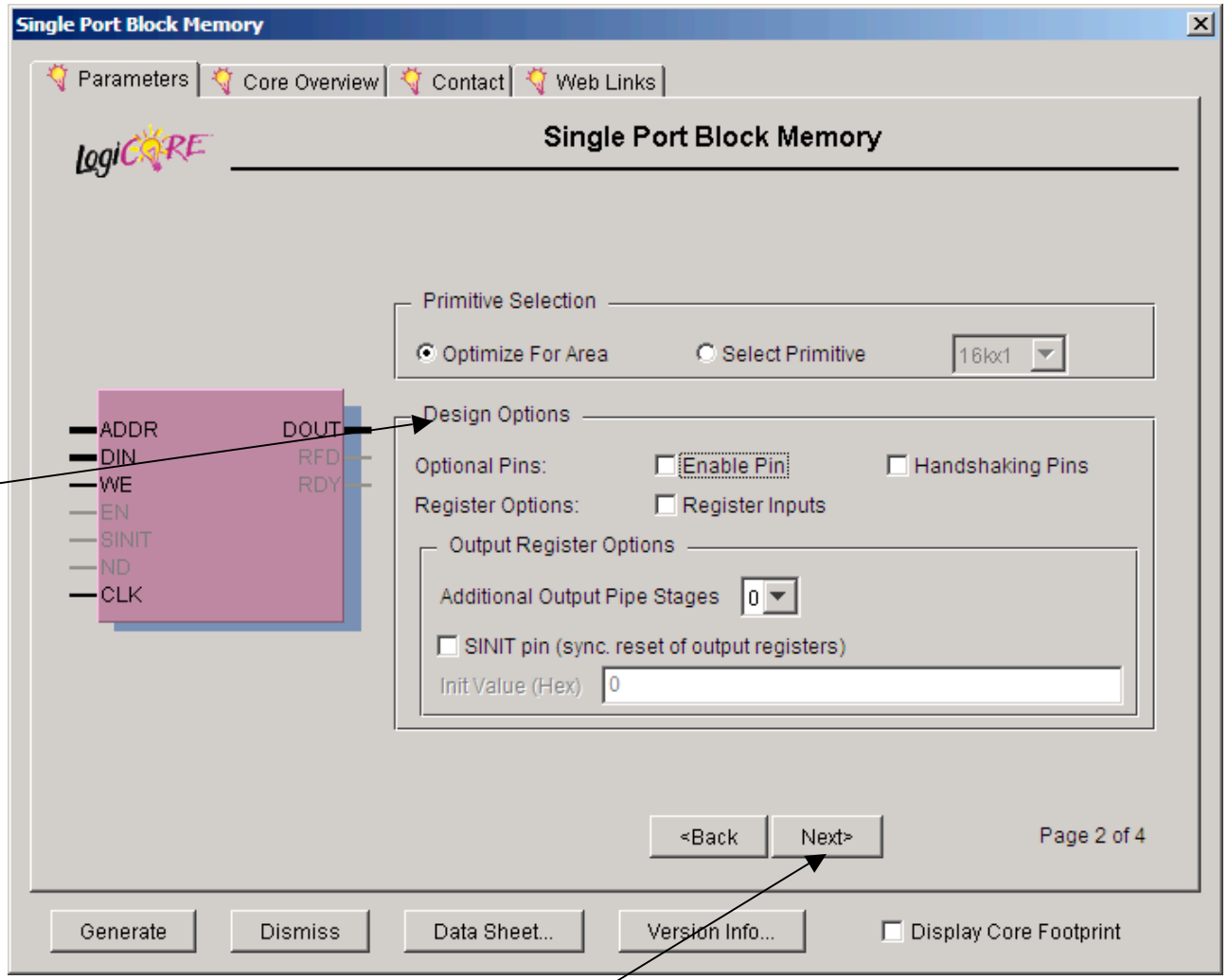
Usually "Read After Write" is what you want




Click "Next" …

# BRAM Example

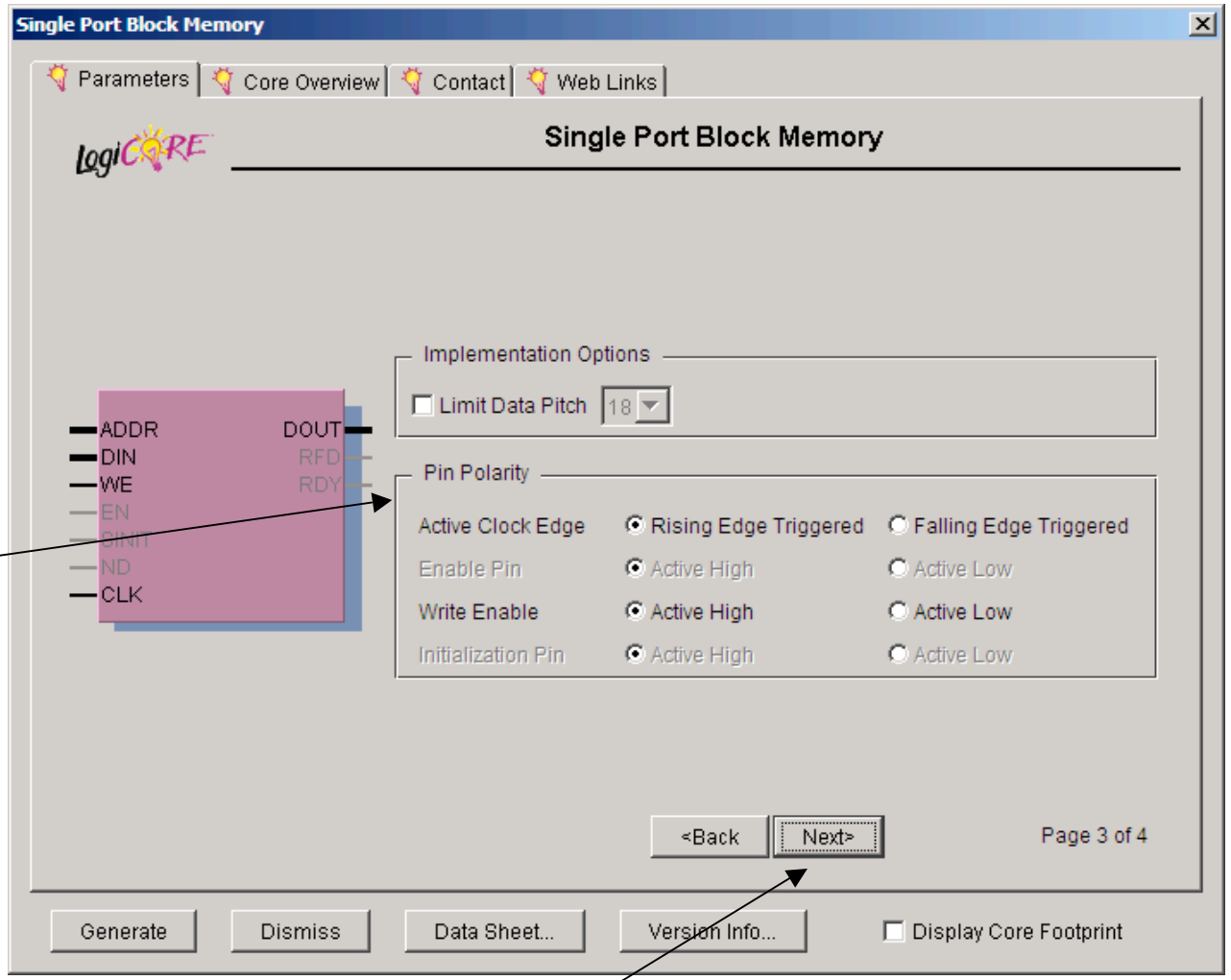
Can add extra control pins, but usually not




Click "Next" …

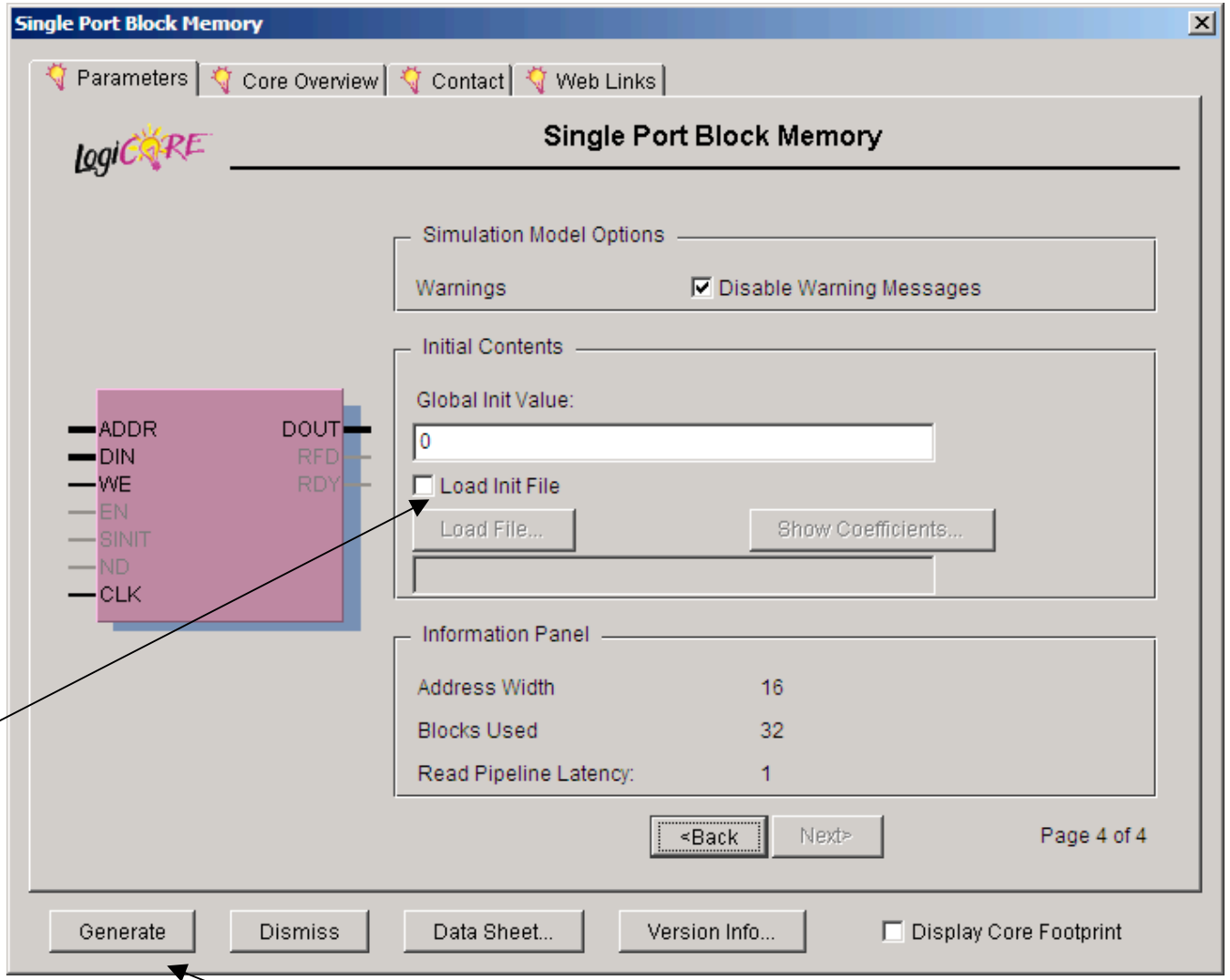
# BRAM Example

Select polarity of control pins; active high default is usually just fine




Click "Next" …

# BRAM Example




Click to name a .coe file that specifies initial contents (eg, for a ROM)

Click "Generate" to complete

# .coe file format

```
memory_initialization_radix=2;
memory_initialization_vector=

00000000,
00111110,
01100011,
00000011,
00000011,
00011110,
00000011,
00000011,
01100011,
00111110,
00000000,
00000000,
```

Memory contents with location 0 first, then location 1, etc. You can specify input radix, in this example we're using binary. MSB is on the left, LSB on the right. Unspecified locations (if memory has more locations than given in .coe file) are set to 0.

# Using result in your Verilog

- Look at generated Verilog for module def'n:

```
module ram64x8 (addr,clk,din,dout,we);
    input [15 : 0] addr;
    input clk;
    input [7 : 0] din;
    output [7 : 0] dout;
    input we;
    …
endmodule
```

- Use to instantiate instances in your code:

```
ram64x8 foo(addr,clk,din,dout,we);
```