

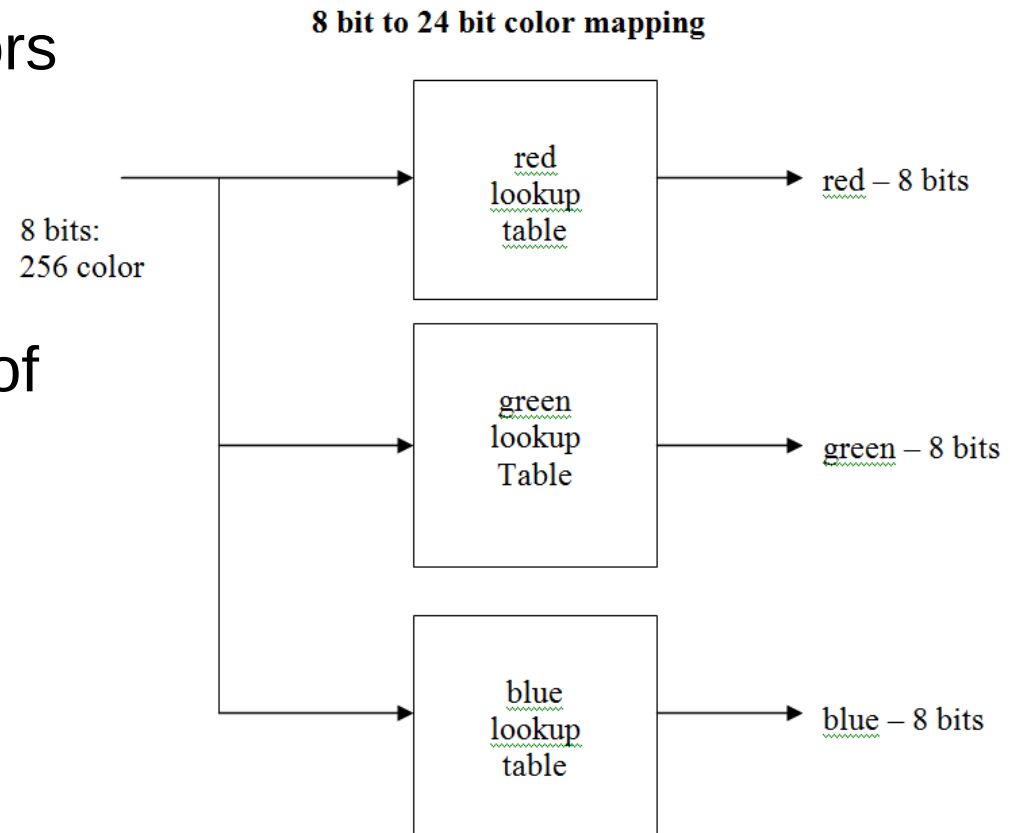
Displaying Images

Memory Requirements

- $1024*768*24 = 18,874,368$
- $800*600*24 = 11,520,000$
- $640*480*24 = 7,372,800$
- $640*480*8 = 2,457,600$
- $640*480*4 = 1,228,800$
- Labkit XCV2V6000: 144 BRAMs, 2952K bits total
- Generate/synthesis time dependent upon memory size. For large memory 2-3 hours!

Color Maps

- Using 24bit RGB the number of unique colors is $2^8 * 2^8 * 2^8 = 2^{24}$ or 16 million colors.
- Using 8 bits for each pixel, select a palette of 256 colors and use a color map. Save on memory by using 16 colors.
- Use photoshop and Matlab to create color maps



```

////////////////////////////////////
//
// blob: generate rectangle on screen
//
////////////////////////////////////
module blob
    #(parameter WIDTH = 64,    // default width
        HEIGHT = 64,    // default height
        COLOR = 3'b111)    // default
    color: white
    (input [10:0] x,hcount,
     input [9:0] y,vcount,
     output reg [2:0] pixel);

    always @ * begin
        if ((hcount >= x && hcount < (x+WIDTH)) &&
            (vcount >= y && vcount < (y+HEIGHT)))
            pixel = COLOR;
        else pixel = 0;
    end
endmodule

```

```

////////////////////////////////////
//
// picture_blob: display a picture
//
////////////////////////////////////
module picture_blob
    #(parameter WIDTH = 128,    // default picture width
        HEIGHT = 256,    // default picture height
        (input pixel_clk,
         input [10:0] x,hcount,
         input [9:0] y,vcount,
         output reg [23:0] pixel);

    wire [11:0] image_addr;    // num of bits for 128*256 ROM
    wire [7:0] image_bits, red_mapped,
              green_mapped, blue_mapped;

    // note the one clock cycle delay in pixel!
    always @ (posedge pixel_clock) begin
        if ((hcount >= x && hcount < (x+WIDTH)) &&
            (vcount >= y && vcount < (y+HEIGHT)))
            pixel <= {red_mapped, green_mapped, blue_mapped};
        else pixel = 0;
    end

    // calculate rom address and read the location
    assign image_addr = (hcount-x) + (vcount-y) * WIDTH;
    moscow_image_rom rom1(image_addr, clock_65mhz,
                          image_bits);

    // use color map to create 8bits R, 8bits G, 8 bits B;
    red_color_map rcm (image_bits, pixel_clk, red_mapped);
    green_color_map gcm (image_bits, pixel_clk.green_mapped);
    blue_color_map bcm (image_bits, pixel_clk, blue_mapped);

endmodule

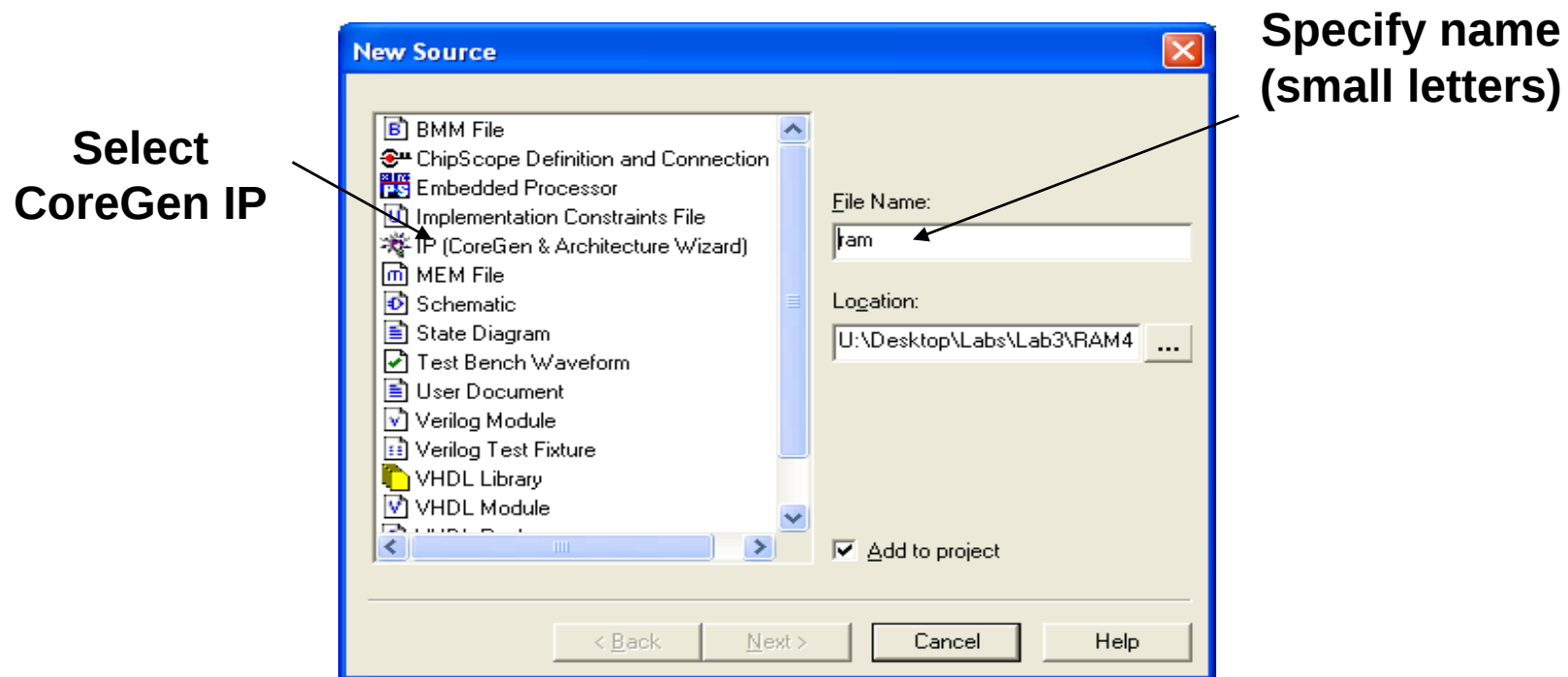
```

Block RAM/ROM



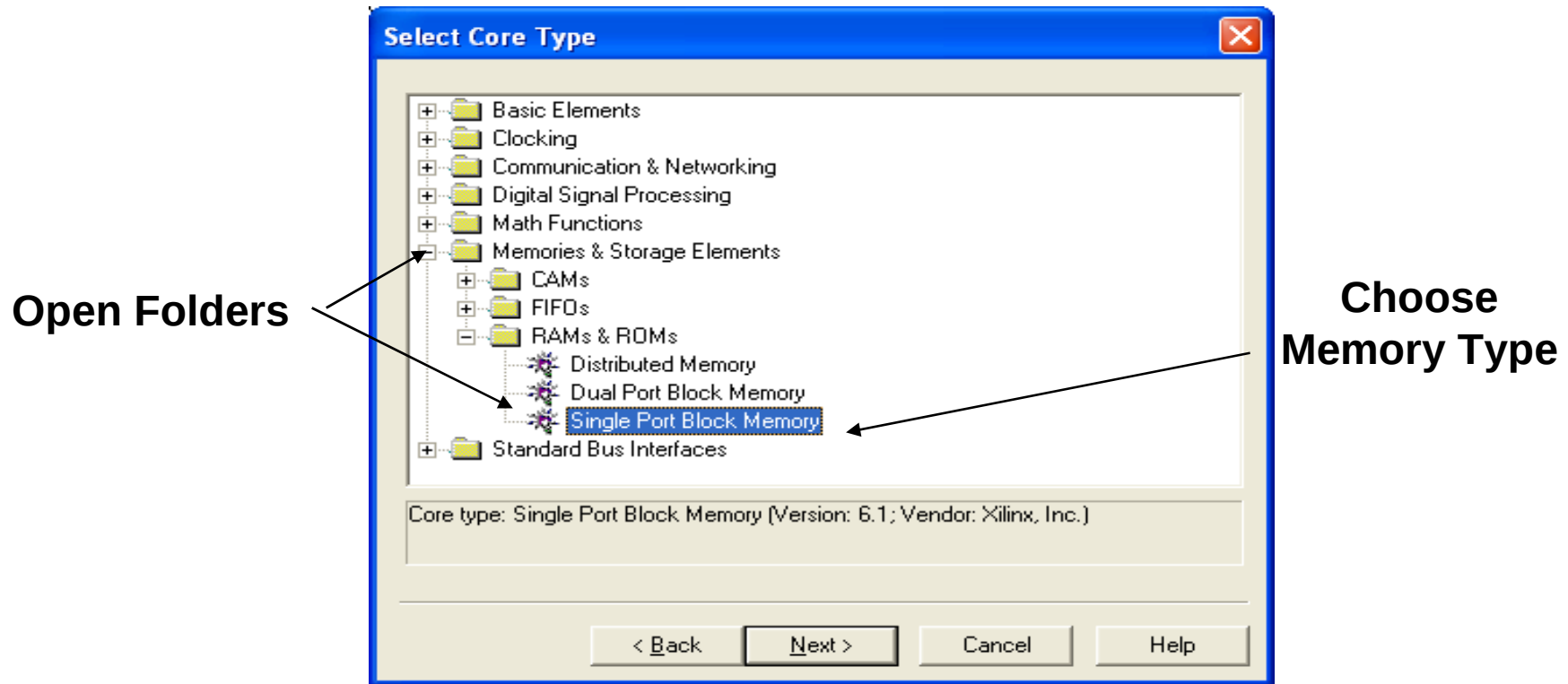
Acknowledgements: Theodoros Konstantakopoulos

- Adding a Block RAM in your Project
 - Project → New Source



Click “Next”

Block RAMs and ROMs using Coregen



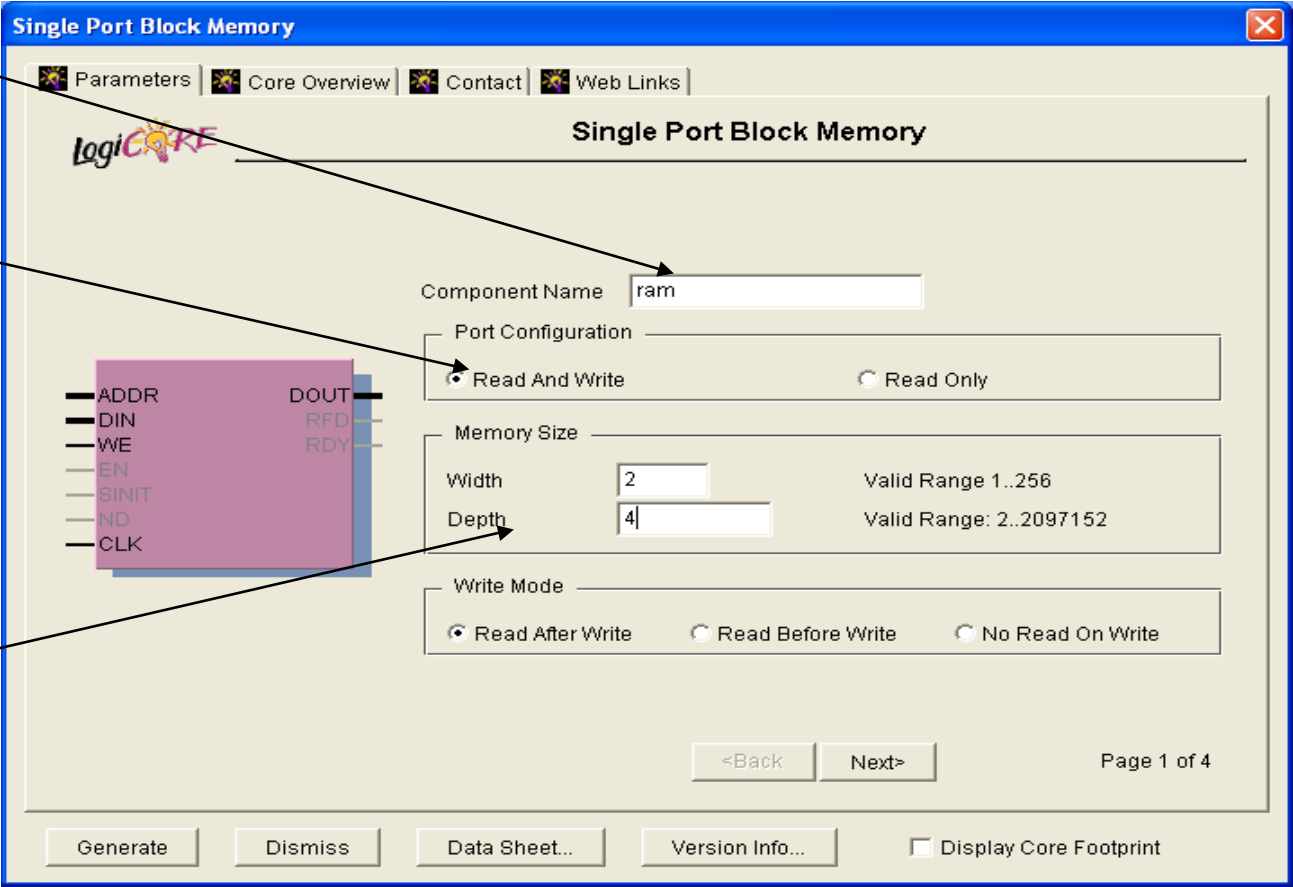
Click “Next” and then “Finish” on the Next Window

Block Memory Properties

Specify name

Select RAM or ROM

Specify Width/Depth



The screenshot shows the 'Single Port Block Memory' configuration window. It has a blue title bar and a tabbed interface with 'Parameters', 'Core Overview', 'Contact', and 'Web Links'. The 'Parameters' tab is active. On the left, there is a block diagram of a memory component with input/output pins: ADDR, DIN, WE, EN, SINIT, ND, CLK on the left; and DOUT, RFD, RDY on the right. The component is highlighted in purple. On the right, the configuration fields are: 'Component Name' (text box with 'ram'), 'Port Configuration' (radio buttons for 'Read And Write' (selected) and 'Read Only'), 'Memory Size' (Width: 2, Depth: 4, with valid ranges 1..256 and 2..2097152 respectively), and 'Write Mode' (radio buttons for 'Read After Write' (selected), 'Read Before Write', and 'No Read On Write'). At the bottom, there are buttons for '<Back', 'Next>', 'Generate', 'Dismiss', 'Data Sheet...', 'Version Info...', and a checkbox for 'Display Core Footprint'. The page number 'Page 1 of 4' is in the bottom right corner.

Single Port Block Memory

Parameters | Core Overview | Contact | Web Links

logiCORE

Single Port Block Memory

Component Name: ram

Port Configuration:

- ☒ Read And Write
- ☐ Read Only

Memory Size:

Width: 2 (Valid Range: 1..256)

Depth: 4 (Valid Range: 2..2097152)

Write Mode:

- ☒ Read After Write
- ☐ Read Before Write
- ☐ No Read On Write

<Back Next> Page 1 of 4

Generate Dismiss Data Sheet... Version Info... ☐ Display Core Footprint

Click "Next"

Block Memory Properties

Add Optional
Control Pins
(if desired)

Single Port Block Memory

Parameters | Core Overview | Contact | Web Links

logiCORE

Single Port Block Memory

Primitive Selection

☒ Optimize For Area ☐ Select Primitive 16kx1

Design Options

Optional Pins: ☐ Enable Pin ☐ Handshaking Pins

Register Options: ☐ Register Inputs

Output Register Options

Additional Output Pipe Stages 0

☐ SINIT pin (sync. reset of output registers)

Init Value (Hex) 0

<Back Next>

Page 2 of 4

Generate Dismiss Data Sheet... Version Info... ☐ Display Core Footprint

Click "Next"

Block Memory Properties

Select Polarity
of Control Pins
Default is
Active High

Single Port Block Memory

Parameters | Core Overview | Contact | Web Links

logiCORE

Single Port Block Memory

Implementation Options

☐ Limit Data Pitch 18

Pin Polarity

Active Clock Edge ☒ Rising Edge Triggered ☐ Falling Edge Triggered

Enable Pin ☒ Active High ☐ Active Low

Write Enable ☒ Active High ☐ Active Low

Initialization Pin ☒ Active High ☐ Active Low

<Back Next>

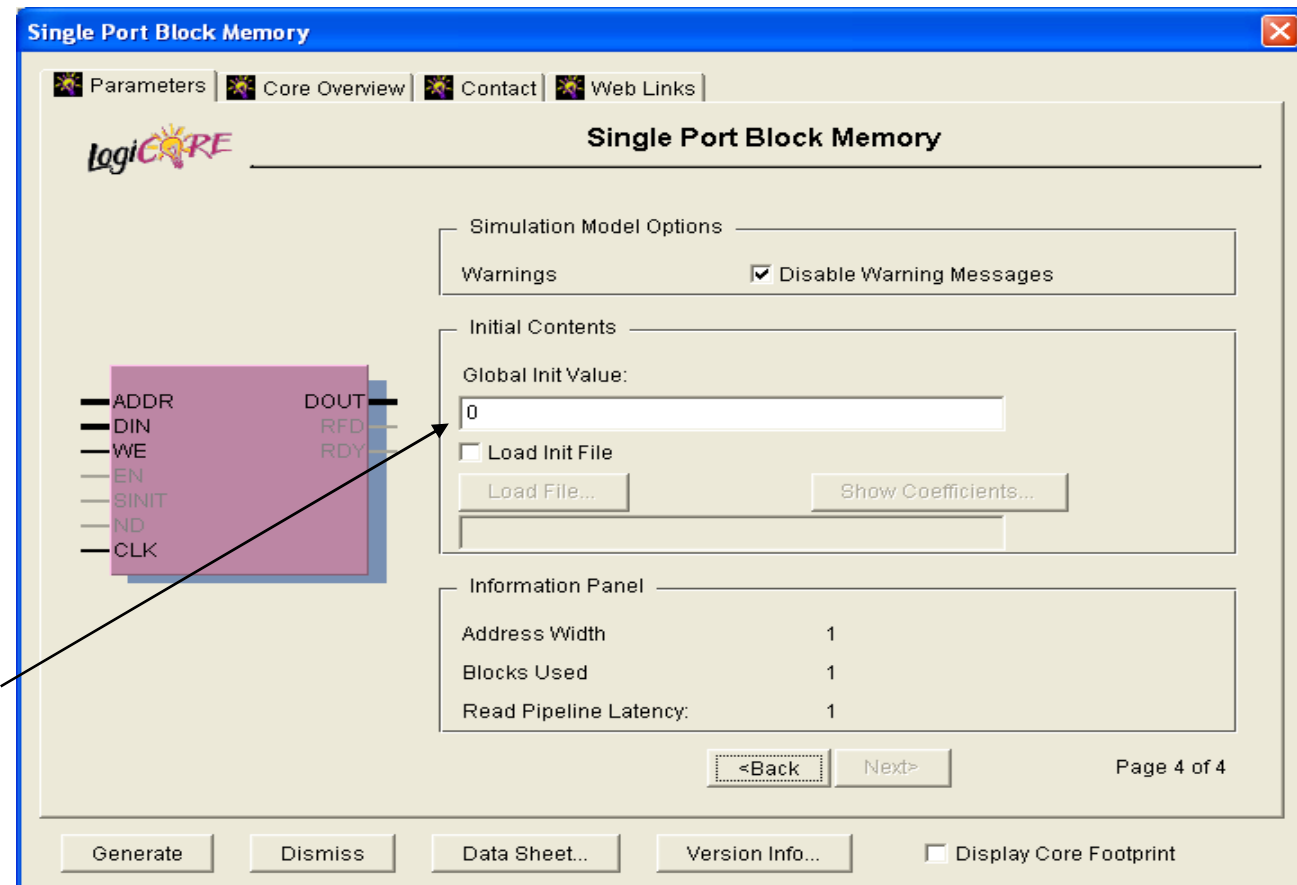
Page 3 of 4

Generate Dismiss Data Sheet... Version Info... ☐ Display Core Footprint

Click "Next"

Block Memory Properties

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



The image shows the 'Single Port Block Memory' configuration window in a software tool. The window has a blue title bar and a tabbed interface with 'Parameters', 'Core Overview', 'Contact', and 'Web Links'. The 'Parameters' tab is active. On the left, there is a block diagram of a memory core with inputs ADDR, DIN, WE, EN, SINIT, ND, and CLK, and outputs DOUT, RFD, and RDY. An arrow points from the text 'Click to name a .coe file that contains initial contents (eg. for a ROM)' to the 'Load Init File' checkbox. The 'Initial Contents' section includes a 'Global Init Value' field set to '0', a 'Load Init File' checkbox, a 'Load File...' button, and a 'Show Coefficients...' button. The 'Simulation Model Options' section has a 'Warnings' checkbox with 'Disable Warning Messages' checked. The 'Information Panel' at the bottom right shows 'Address Width: 1', 'Blocks Used: 1', and 'Read Pipeline Latency: 1'. Navigation buttons '<Back' and 'Next>' are present, along with 'Page 4 of 4'. At the bottom of the window are buttons for 'Generate', 'Dismiss', 'Data Sheet...', 'Version Info...', and a 'Display Core Footprint' checkbox.

Single Port Block Memory

Parameters | Core Overview | Contact | Web Links

logiCORE

Single Port Block Memory

Simulation Model Options

Warnings ☒ Disable Warning Messages

Initial Contents

Global Init Value:

0

☐ Load Init File

Load File... Show Coefficients...

Information Panel

Address Width	1
Blocks Used	1
Read Pipeline Latency:	1

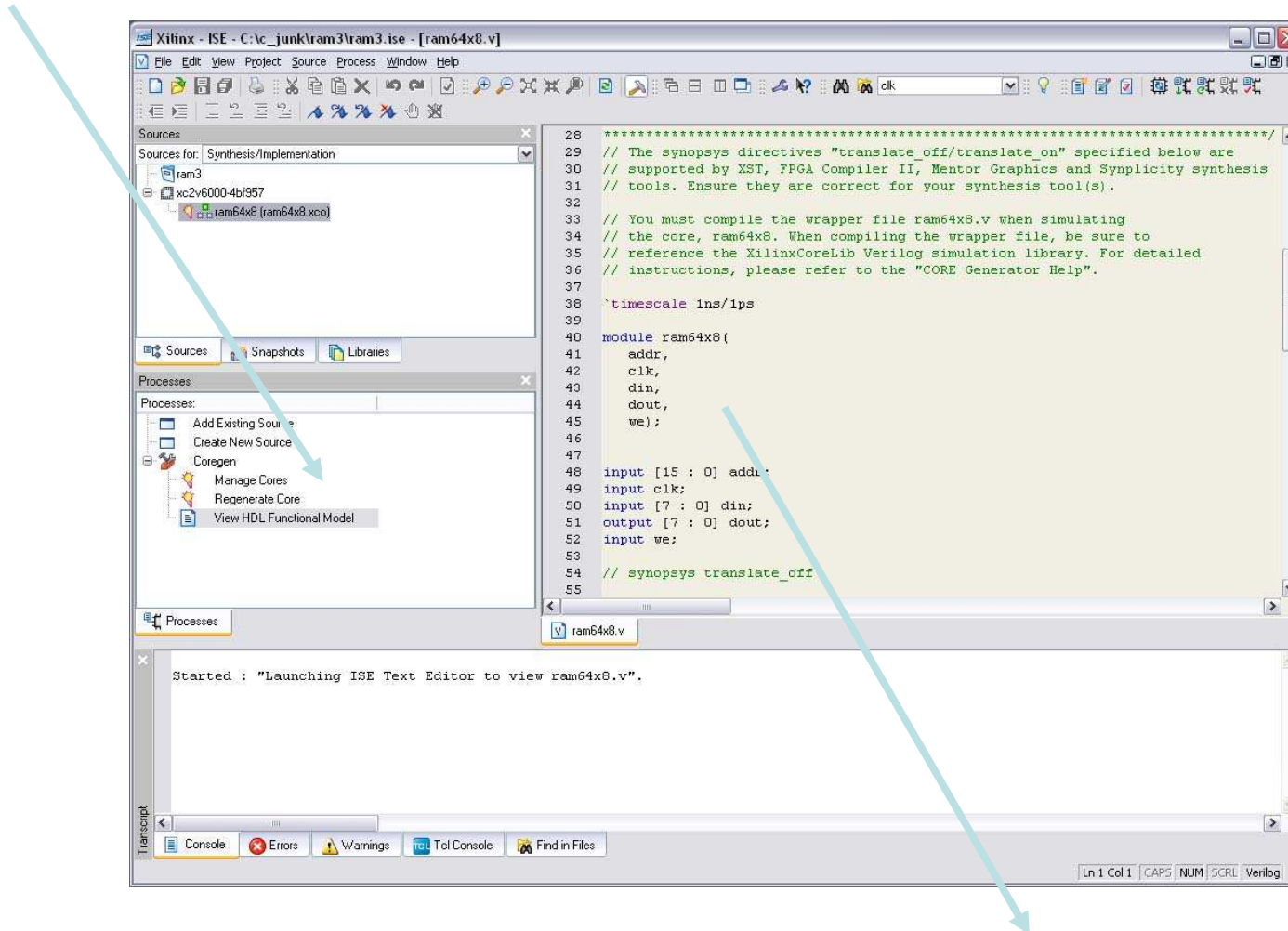
<Back Next> Page 4 of 4

Generate Dismiss Data Sheet... Version Info... ☐ Display Core Footprint

Click "Generate" to Complete

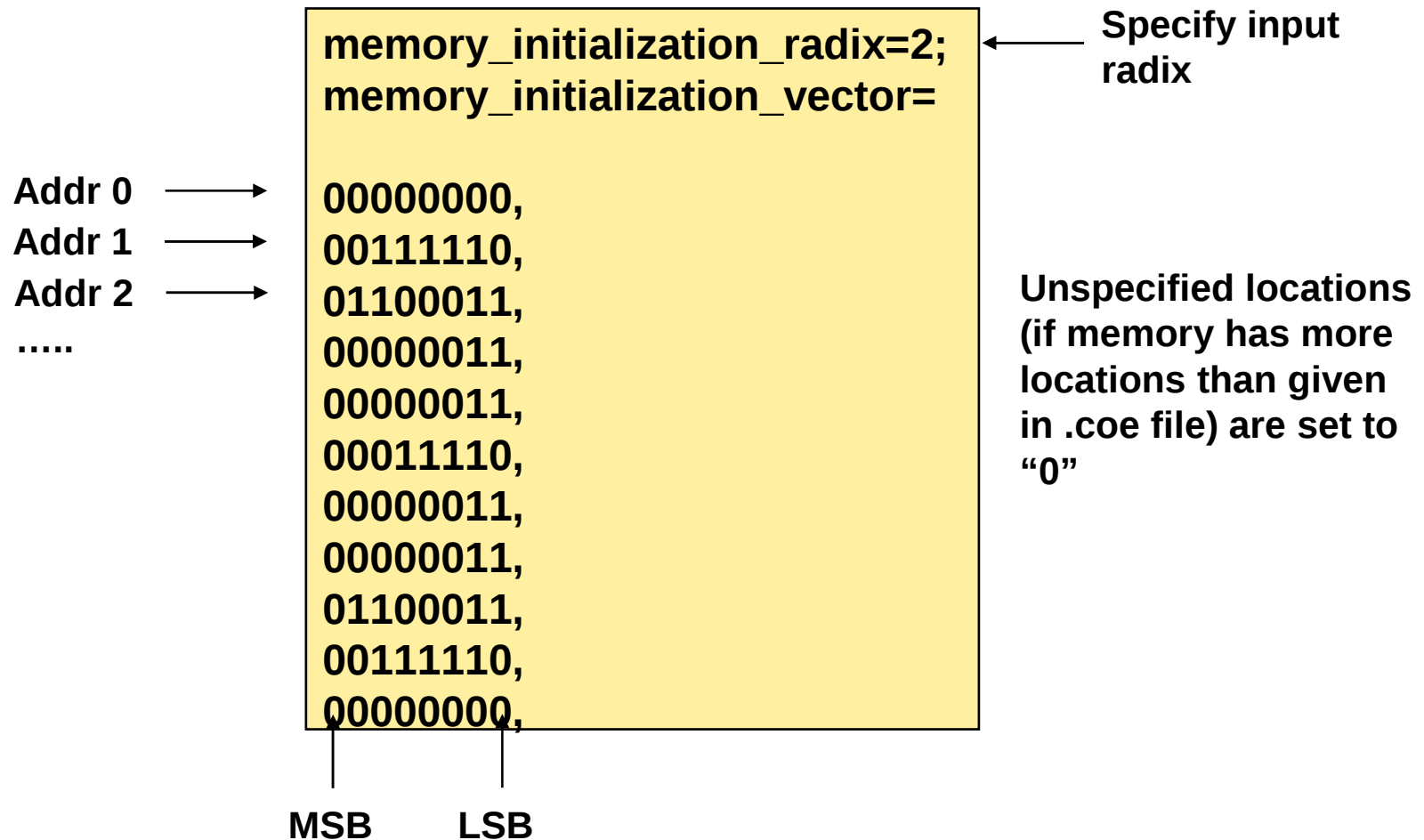
Using result in your Verilog

- Look at generated Verilog for module definition (click on “View HDL Functional Model” under Coregen):



- Use to instantiate instances in your code:
ram64x8 foo(.addr(addr), .clk(clk), .we(we), .din(din), .dout(dout));

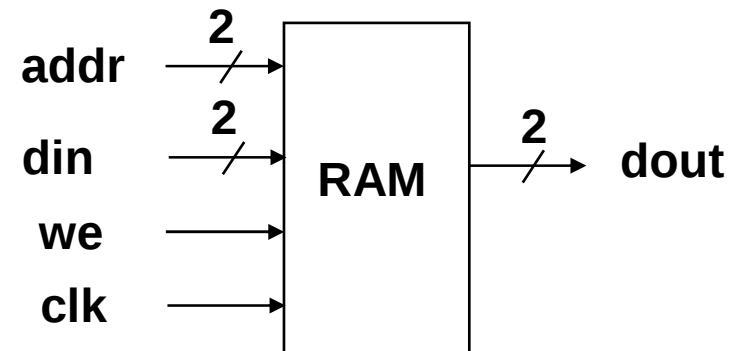
Block RAM/ROM Contents – COE File



Block RAM Module

- Generated Module looks like:

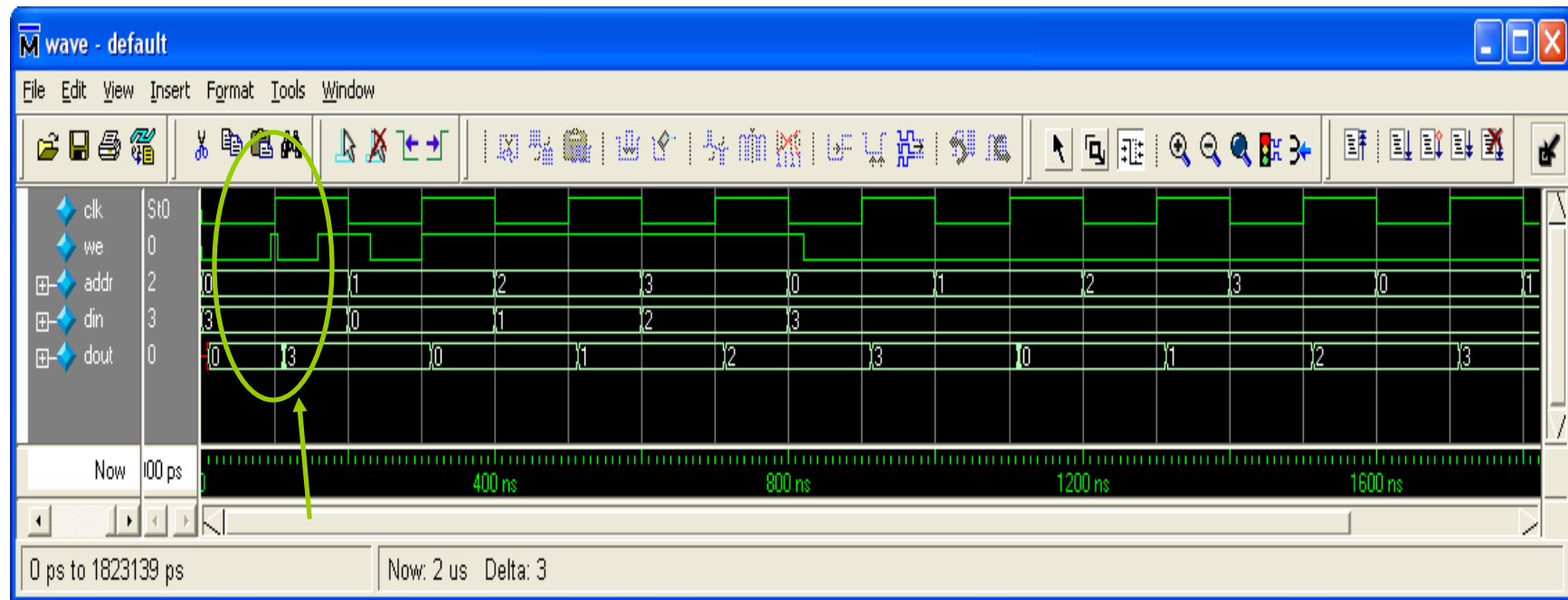
```
module ram (  
  input  [1 : 0] addr,  
  input  clk,  
  input  [1 : 0] din,  
  output [1 : 0] dout,  
  input  we);  
  
  BLKMEMSP_V6_1 #(  
    2,  // c_addr_width  
    .....  
  endmodule
```



Instantiate instances in labkit.v using:

```
ram my_bram (  
  .addr(my_addr),  
  .clk(my_clk),  
  .din(my_din),  
  .dout(my_dout),  
  .we(my_we)  
);
```

Block RAM Simulation



Register interface:

Address, data and we should be setup and held on the rising edge of clock

If we=1 on the rising edge, a write operation takes place

If we=0 on the rising edge, a read operation takes place

Block RAM using Verilog Code

- Block RAM

```
module ram (  
    input [1 : 0] addr, din,  
    input clk, we,  
    output [1 : 0] dout);  
  
    reg [1:0] memory[3:0];  
    reg [1:0] dout_r;  
  
    always @(posedge clk)  
    begin  
        if (we) memory[addr] <= din;  
        dout_r <= memory[addr];  
    end  
endmodule
```

RAM contents are initialized to “0”, by default.