Displaying Images

Memory Requirements

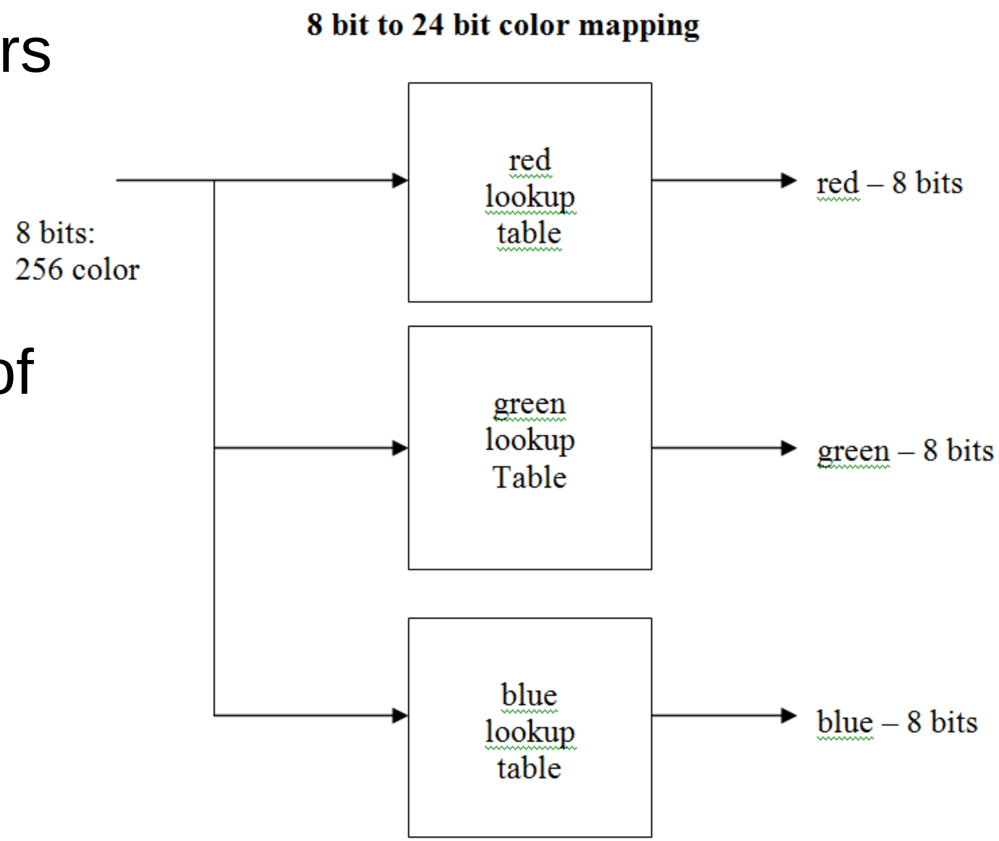
- 1024*768*24 = 18,874,368
- 800*600*24 = 11,520,000
- 640*480*24 = 7,372,800
- 640*480*8 = 2,457,600
- 640*480*4 = 1,228,800
- Labkit XCV2V6000: 144 BRAMs, 2952K bits total
- Generate/synthesis time dependent upon memory size. For large memory 2-3 hours!

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
```

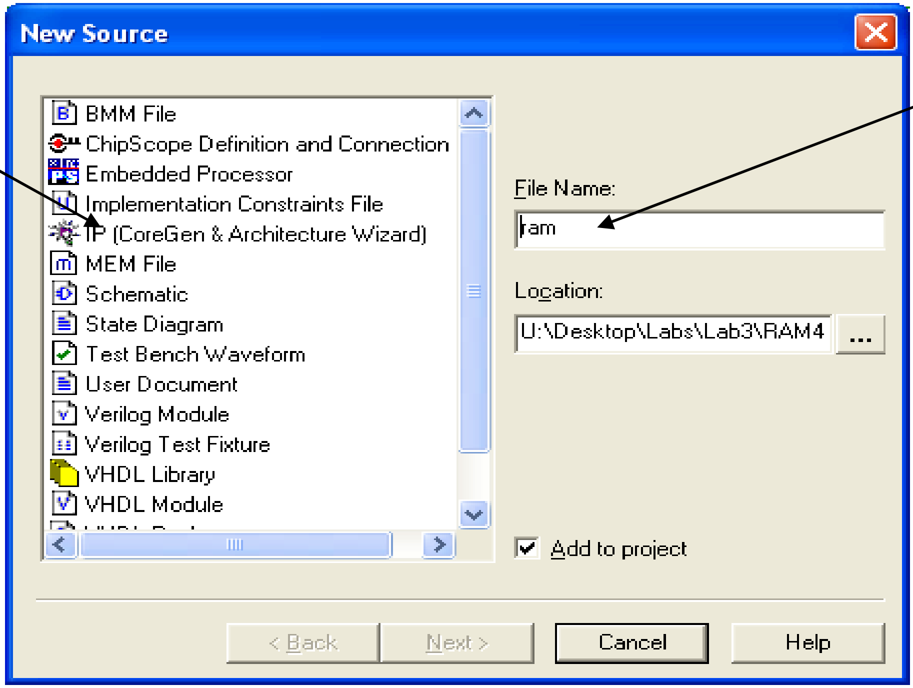
```
////////////////////////////////////////////////////
//
// picture_blob: display a picture
//
//////////////////////////////////////////////////
module picture_blob
   #(parameter WIDTH = 128,     // default picture width
               HEIGHT =256,     // default picture height
   (input pixel_clk,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

   wire [11:0] image_adr;   // 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

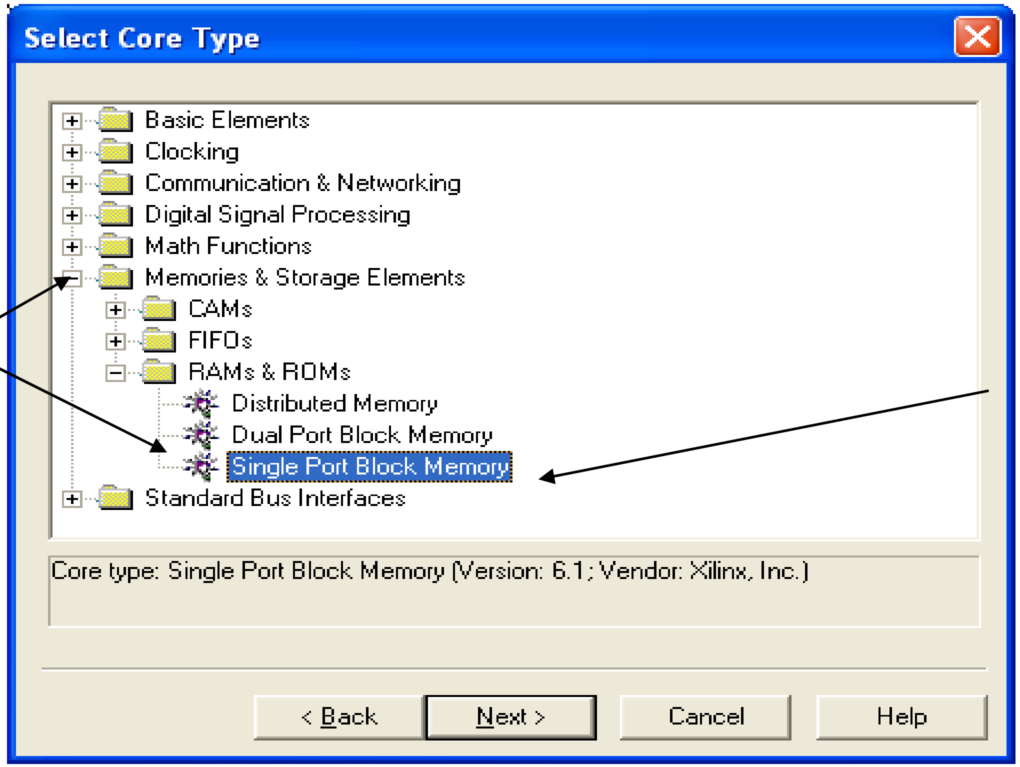
Select CoreGen IP



Specify name (small letters)

Click "Next"

Block RAMs and ROMs using Coregen



Open Folders

Choose Memory Type

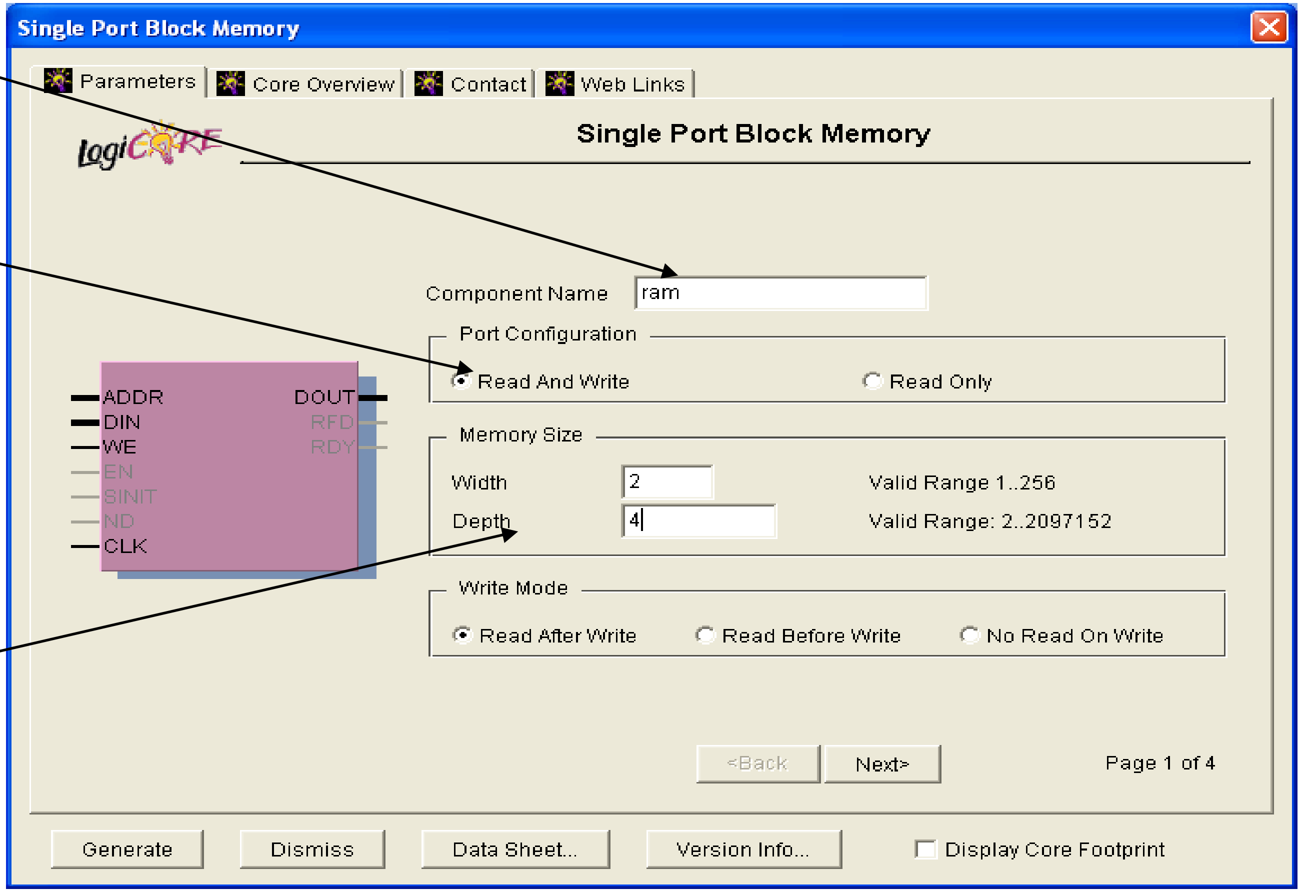
Click "Next" and then "Finish" on the Next Window

Block Memory Properties

Specify name

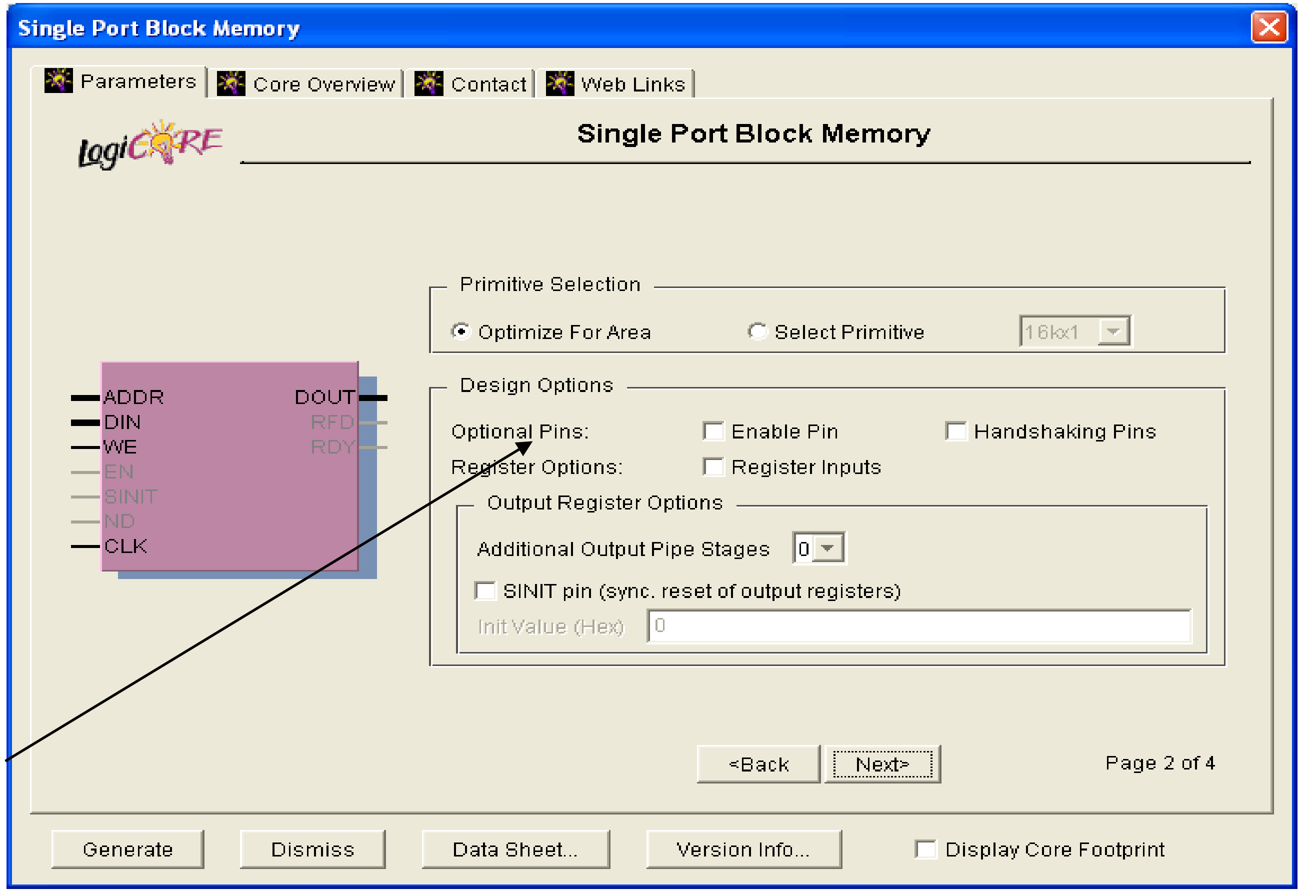
Select
RAM or ROM

Specify
Width/Depth



Click "Next"

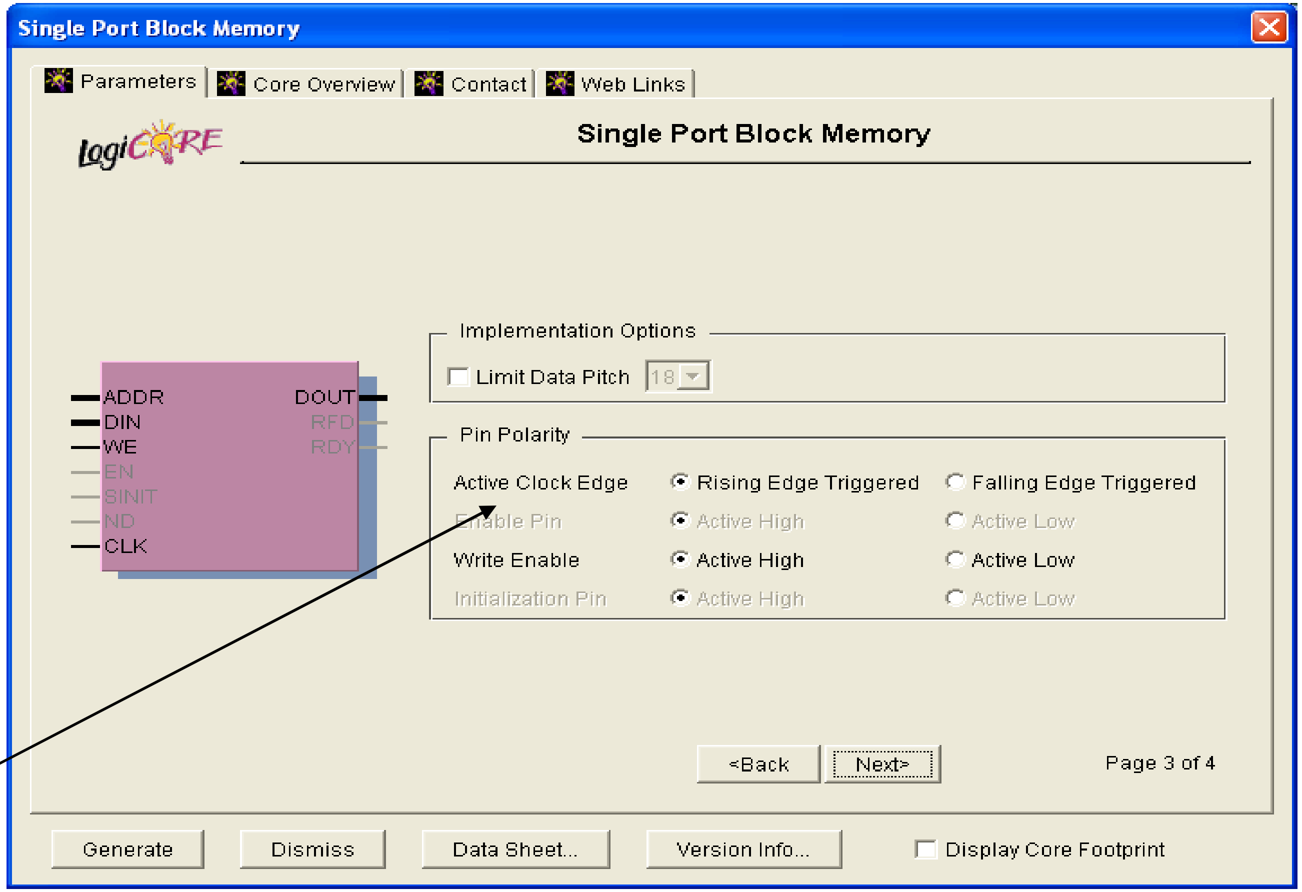
Block Memory Properties



Add Optional Control Pins (if desired)

Click "Next"

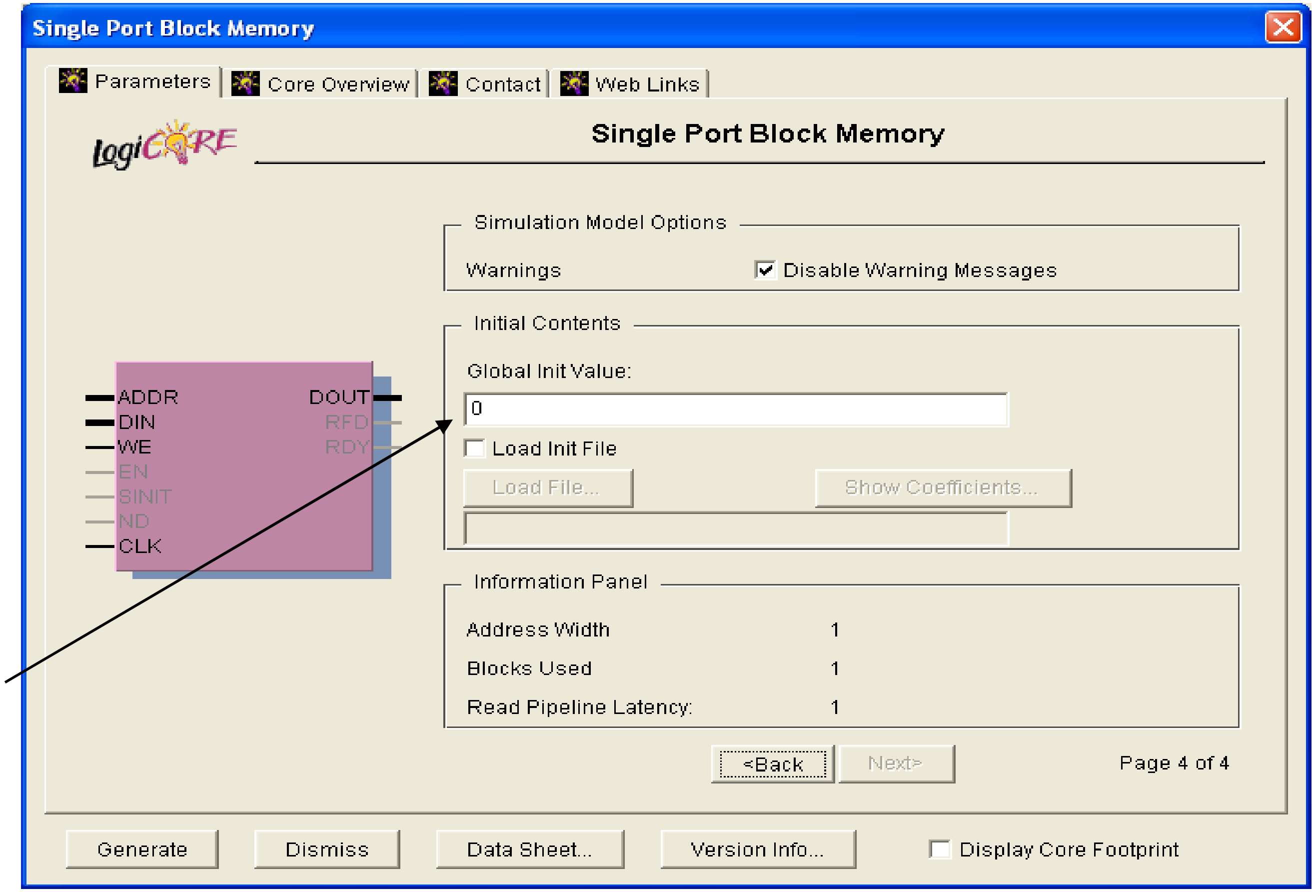
Block Memory Properties



Select Polarity of Control Pins Default is Active High

Click "Next"

Block Memory Properties

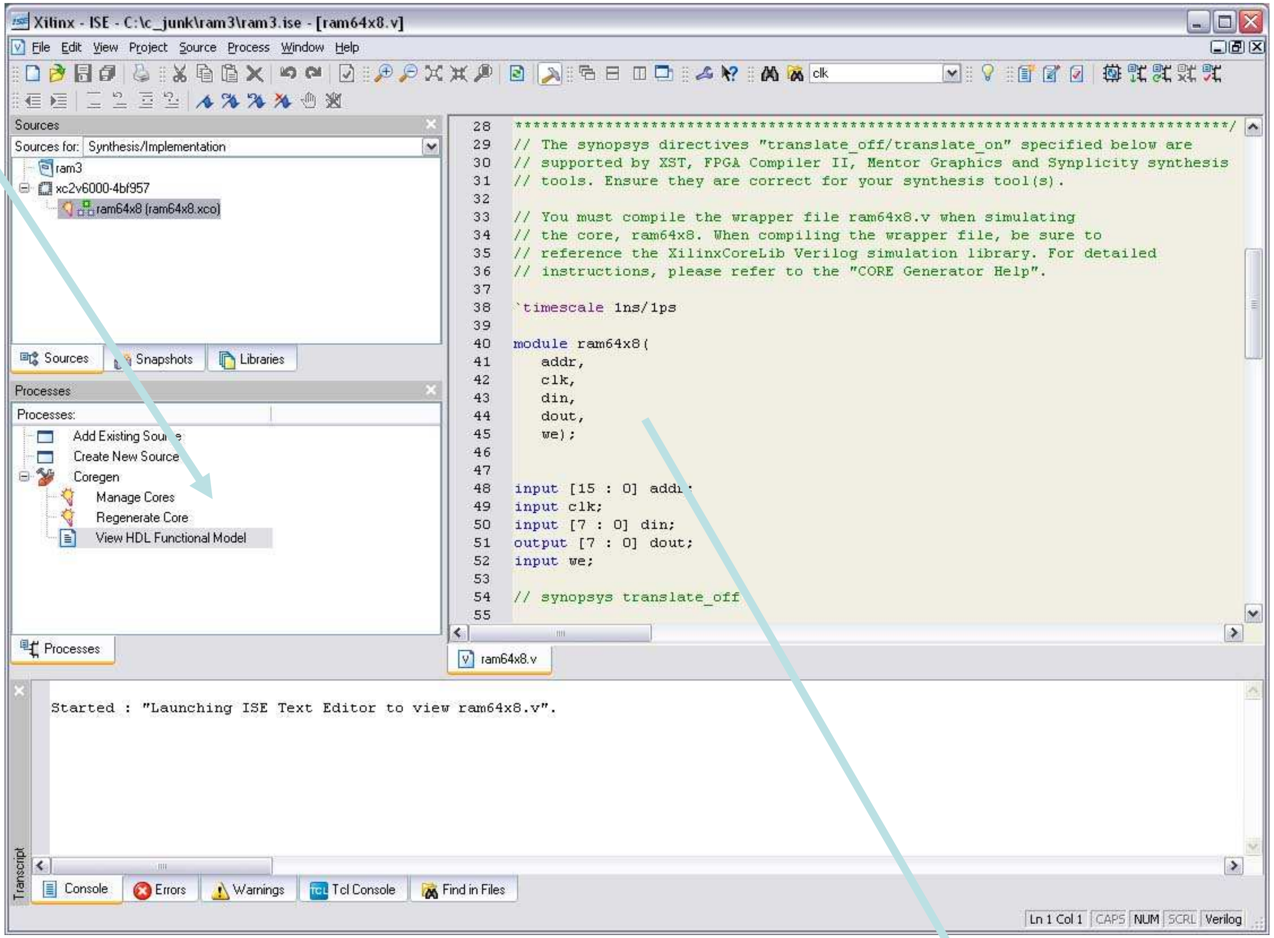


Click to name a .coe file that contains initial contents (eg. for a ROM)

Click "Generate" to Complete

Using result in your Verilog

- Look at generated Verilog for module defintion (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

```
memory_initialization_radix=2;
memory_initialization_vector=

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

Specify input radix

Addr 0

Addr 1

Addr 2

…..

Unspecified locations (if memory has more locations than given in .coe file) are set to “0”

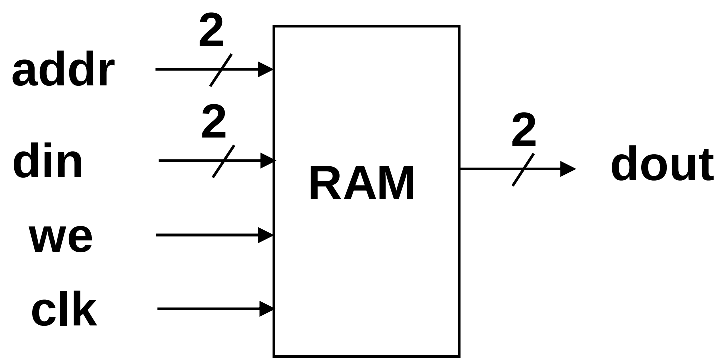
MSB

LSB

Block RAM Module

- Generated Module looks like:

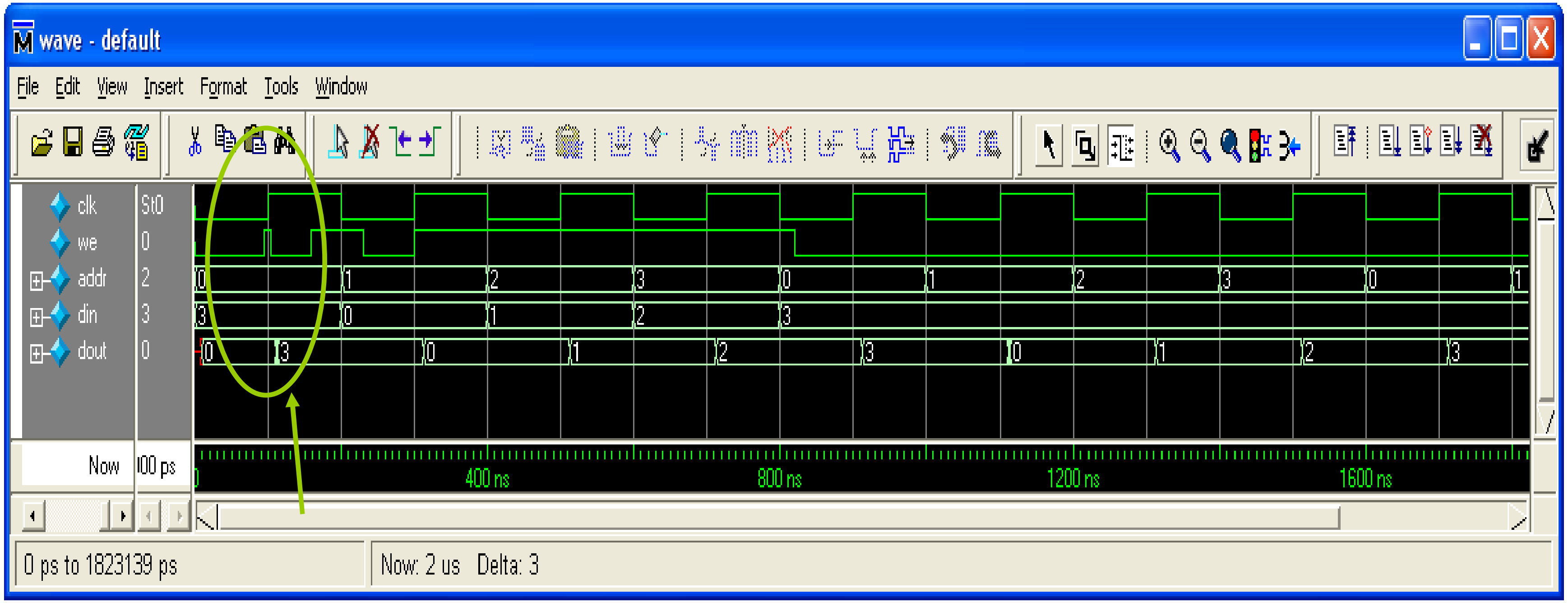
```
module ram (
input    [1 : 0] addr,
input    clk,
input    [1 : 0] din,
output  [1 : 0] dout,
input    we);

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.