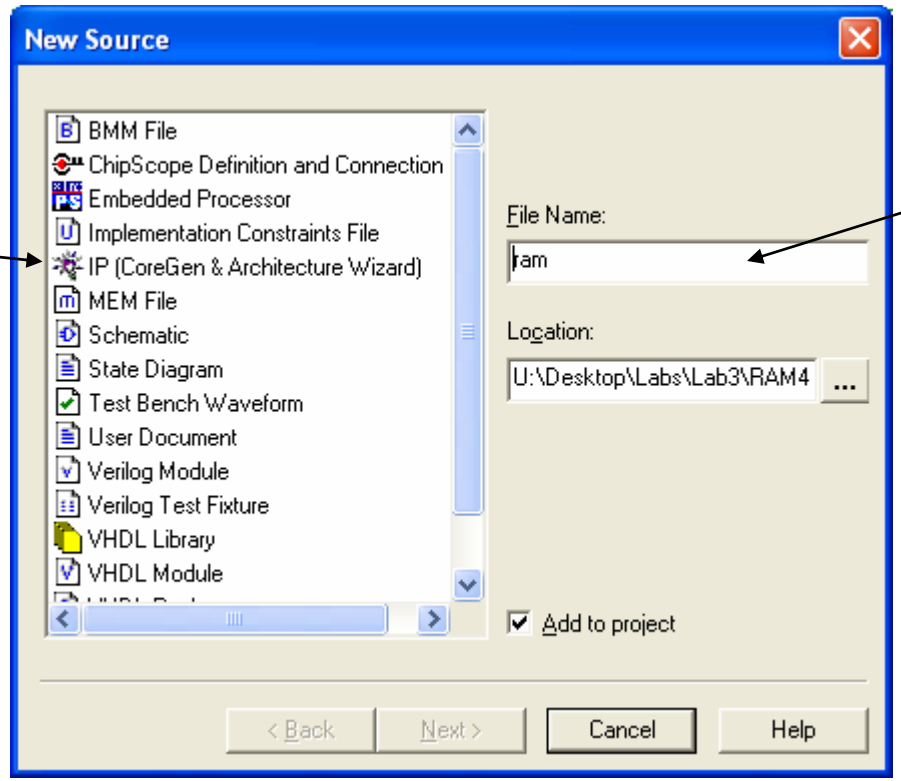
Block RAM/ROM

Acknowledgements: Theodoros Konstantakopoulos

Block RAMs and ROMs using Coregen (to be covered in Recitation)

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

Select CoreGen IP

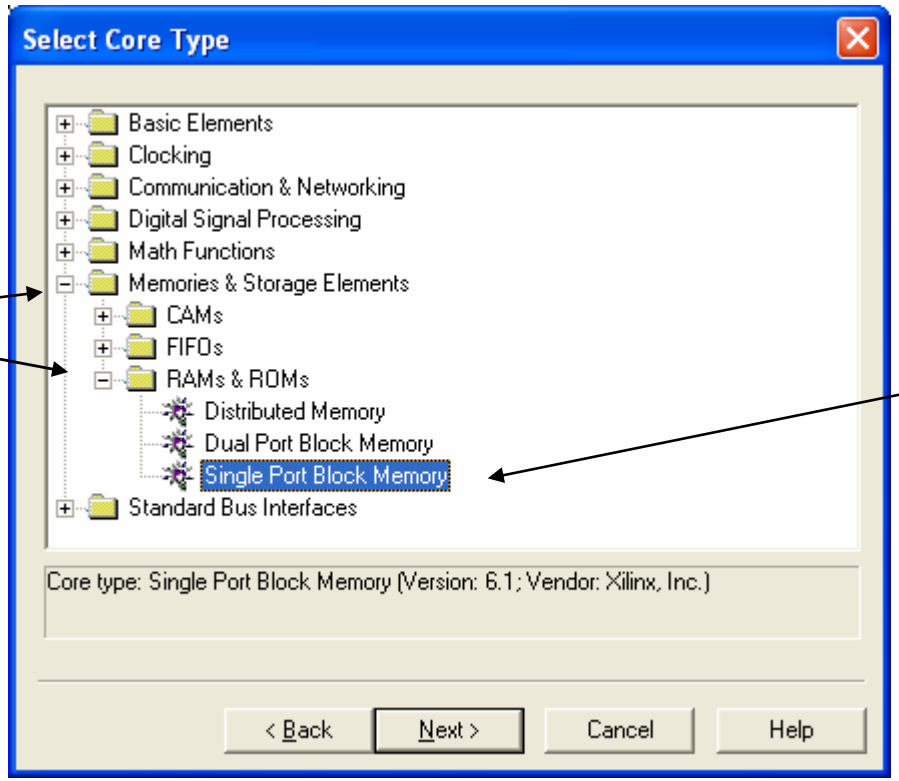


Specify name (small letters – no numbers)

Click "Next"

Block RAMs and ROMs using Coregen

Open Folders



Choose Memory Type

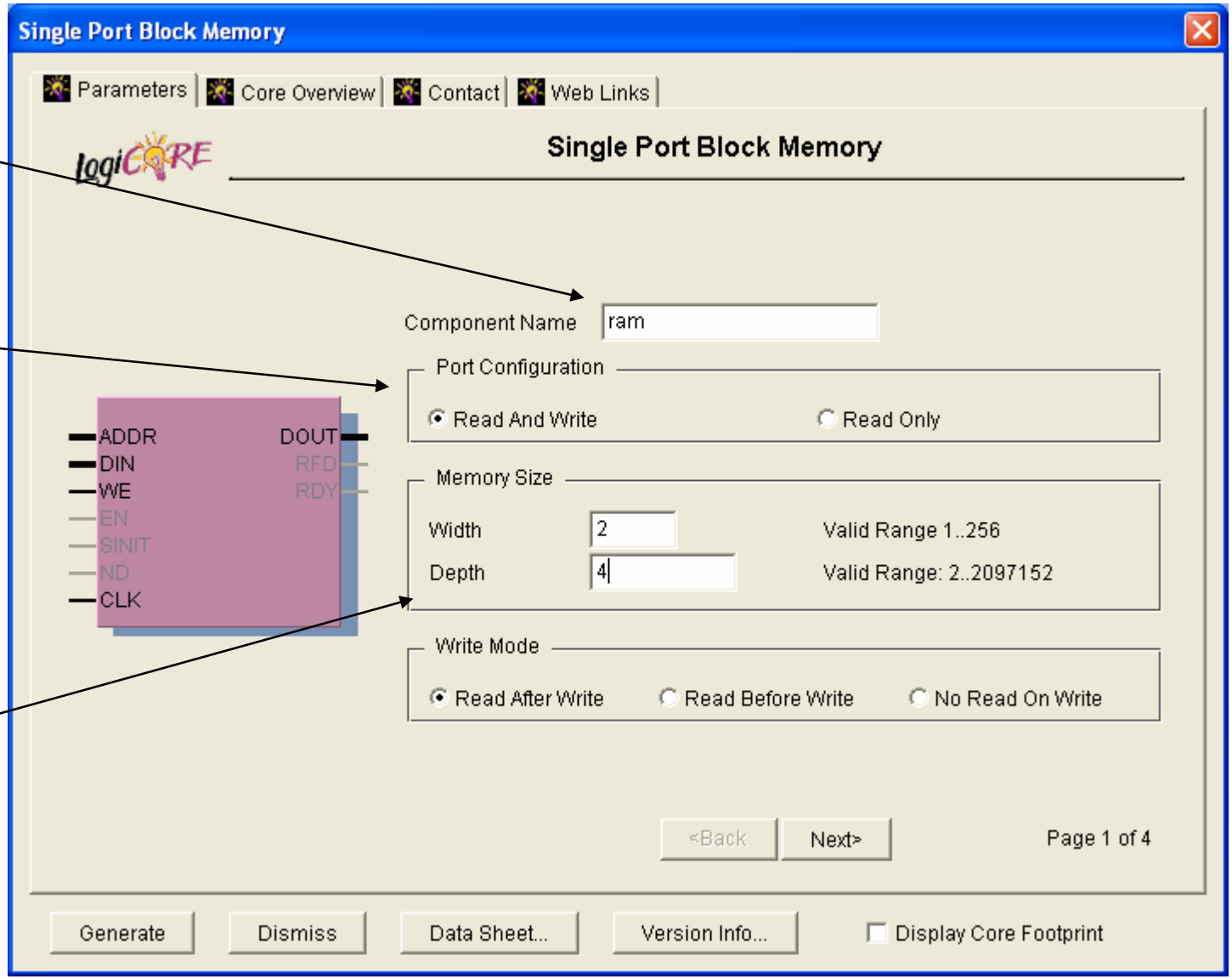
Click "Next" and then "Finish" on the Next Window

Block Memory Properties

Specify name

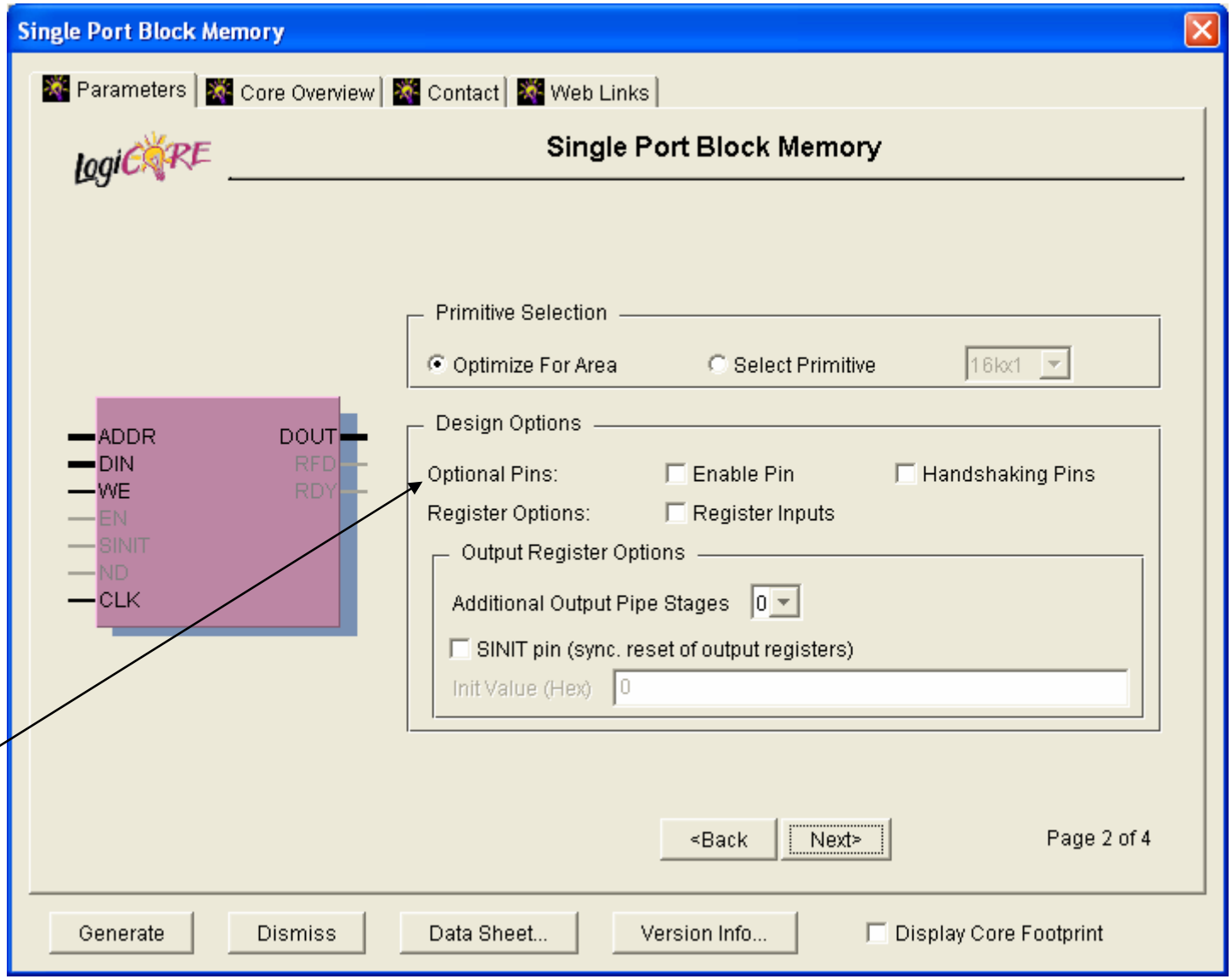
Select RAM or ROM

Specify Width/Depth



Click "Next"

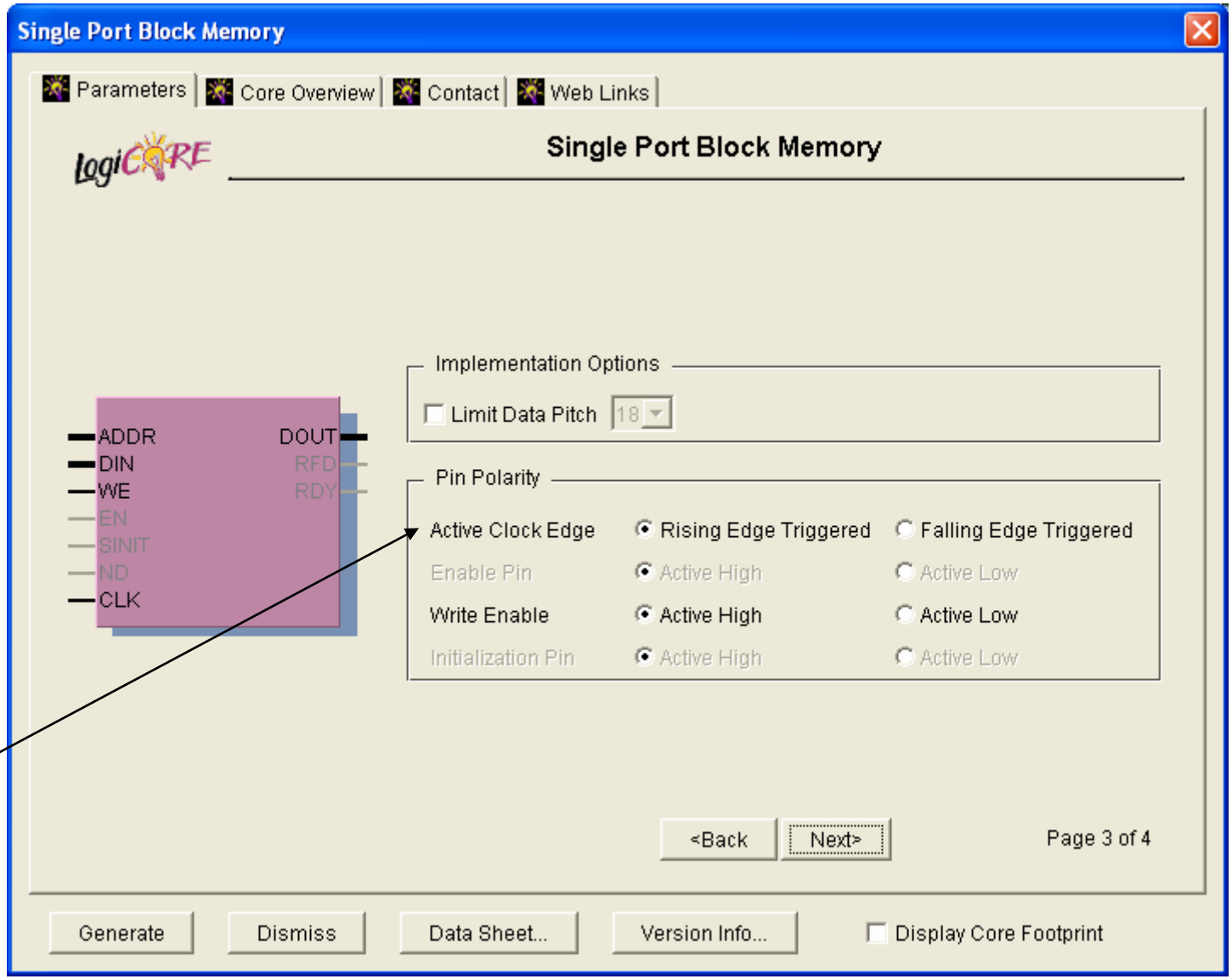
Block Memory Properties



Add Optional Control Pins (if desired)

Click "Next"

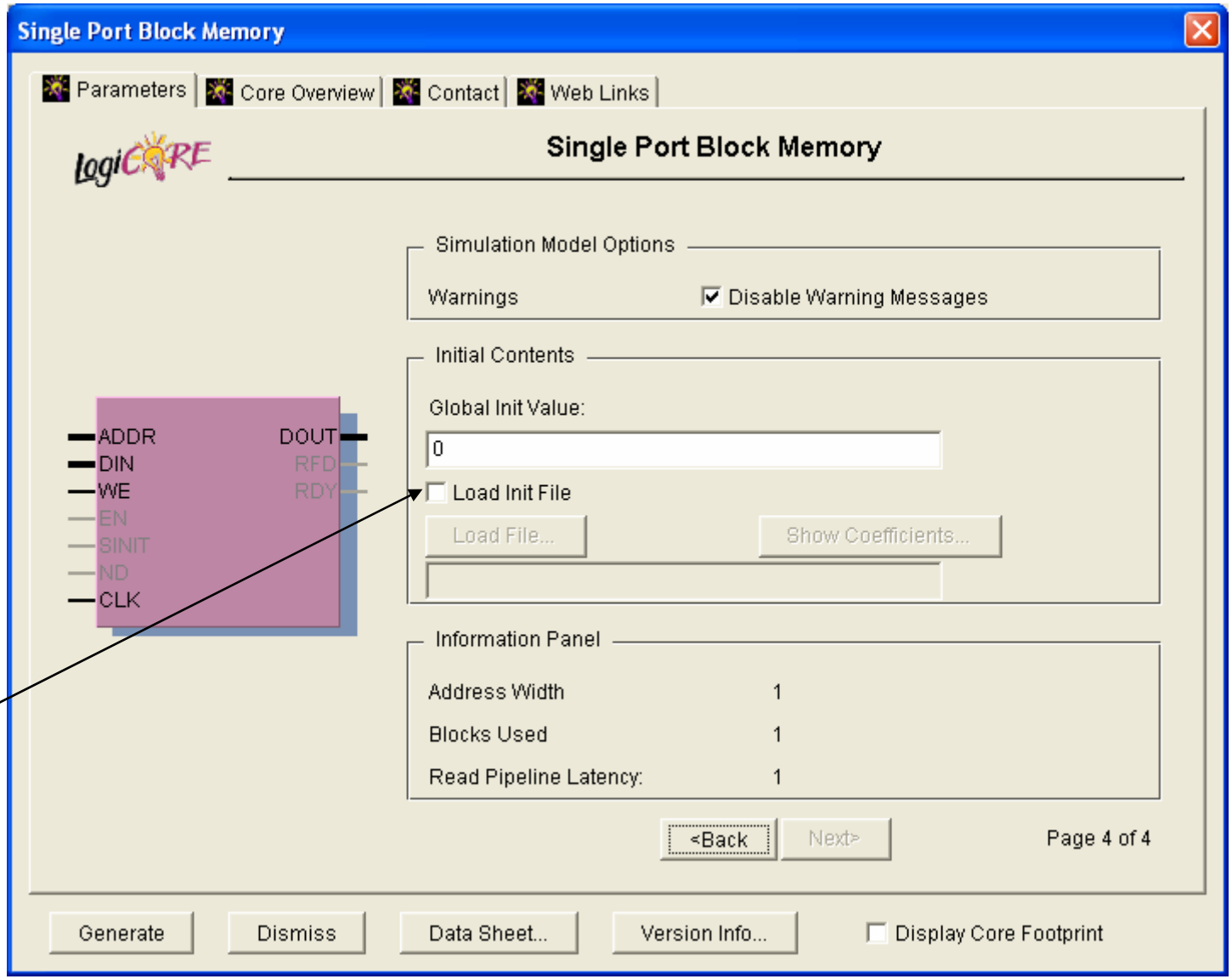
Block Memory Properties



Select Polarity of Control Pins Default is Active High

Click "Next"

Block Memory Properties

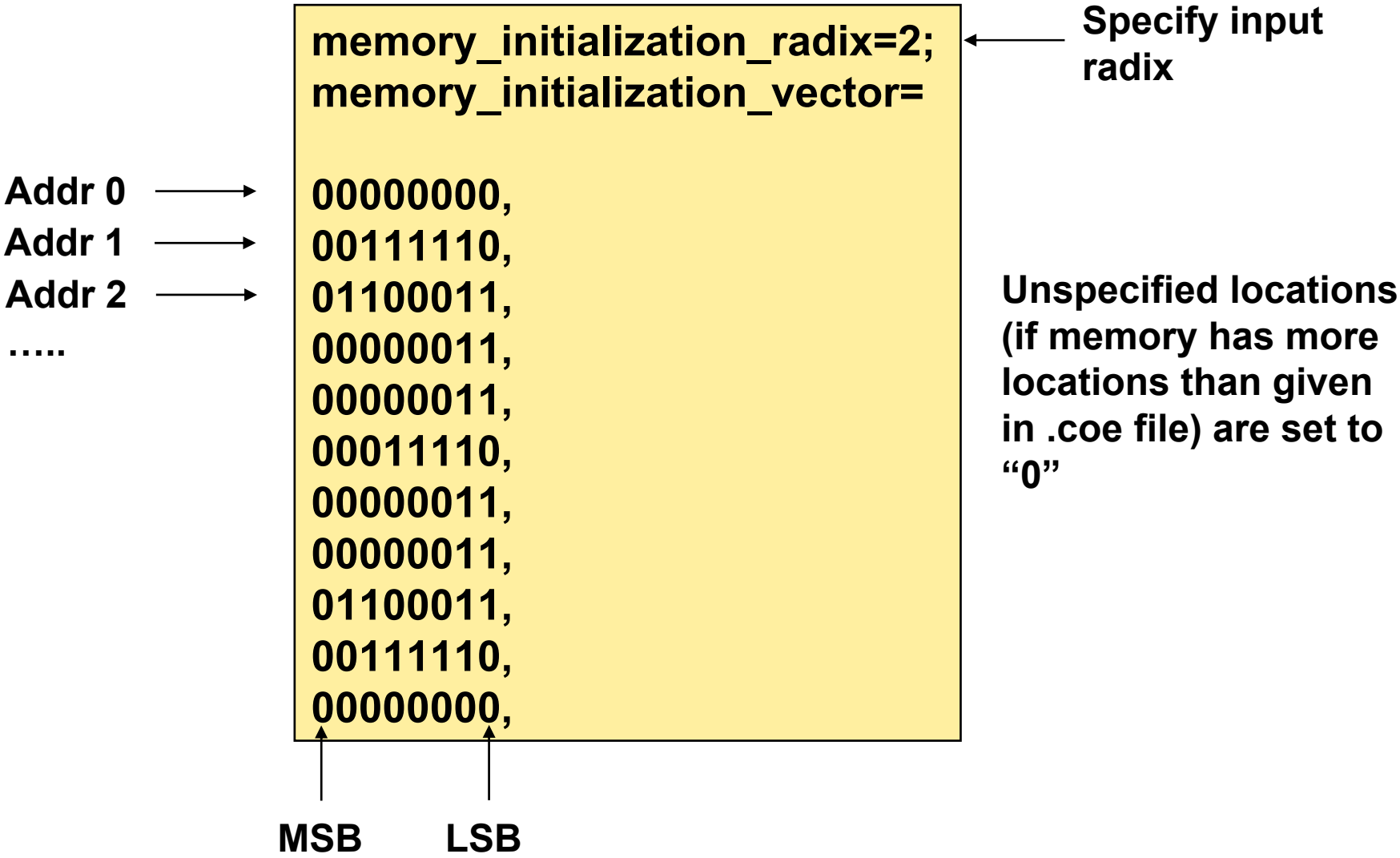


Click to name a .coe file that contains initial contents (eg. for a ROM)

Click "Generate" to Complete

Block RAM/ROM Contents

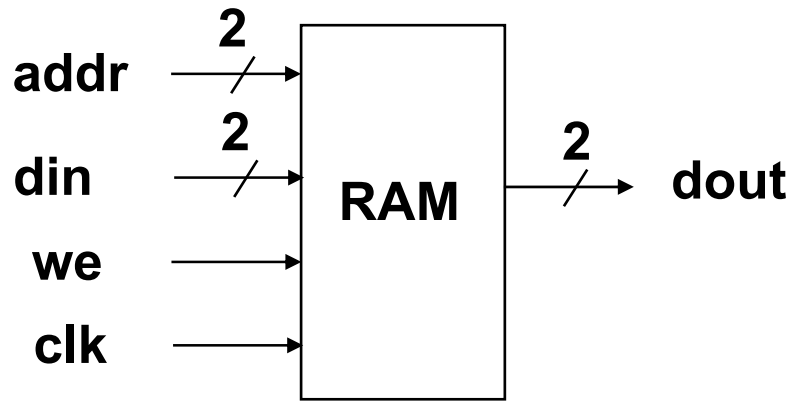
- .coe file looks like:



Block RAM Module

- Generated Module looks like:

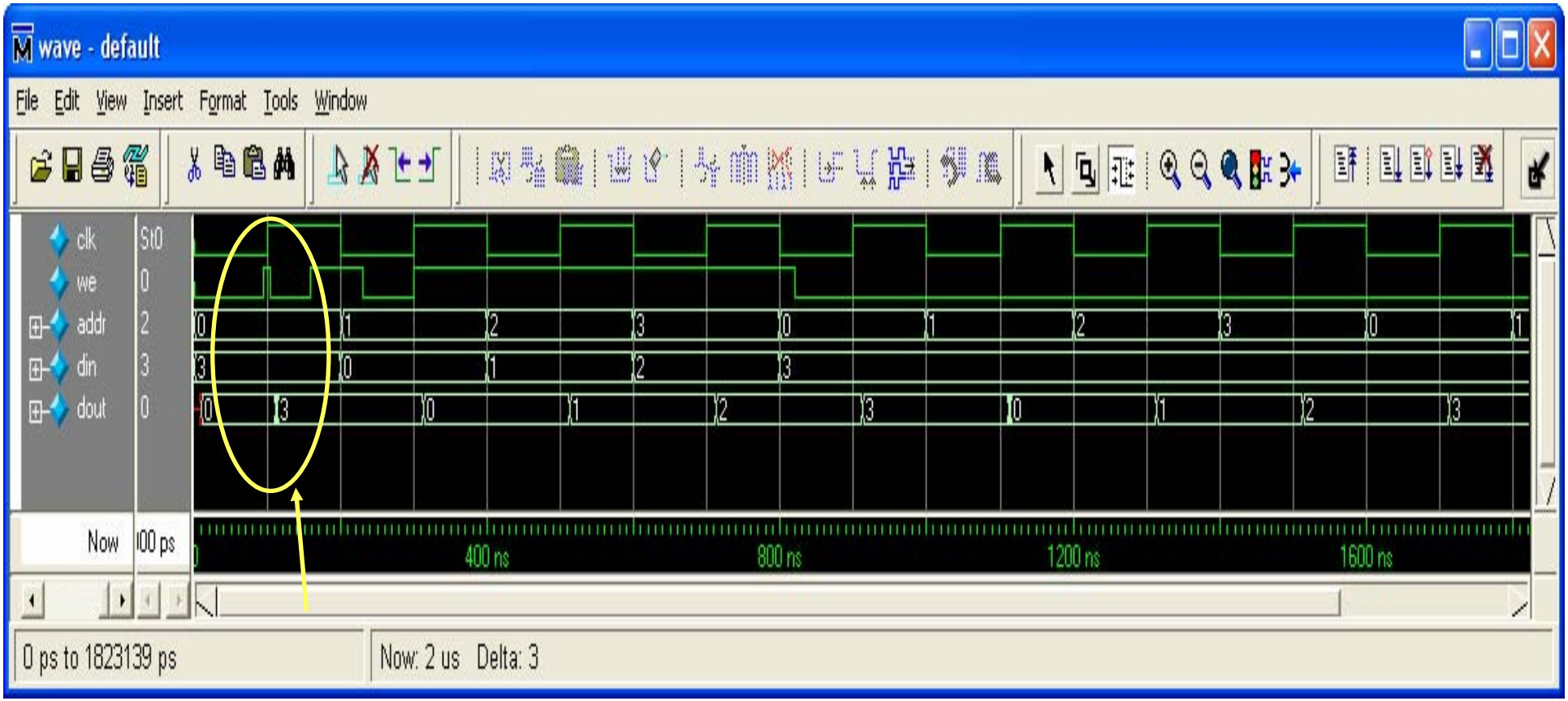
```
module ram (
          addr,
          clk,
          din,
          dout,
          we);

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 (addr, clk, din, dout, we);

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.

If for some reason you need to specify the initial contents of a RAM, then using CoreGen (instead of the Verilog code) is pretty much the only option.

Block ROM using Verilog Code (Synchronous)

- **Block ROM - Synchronous**

ROMs are inferred from case statements:

```
module rom (clk, addr, data);
input clk;
input [1:0] addr;
output [1:0] data;

always @(posedge clk)
        case (addr)
                2'b00: data <= 2'b01;
                2'b01: data <= 2'b10;
                2'b10: data <= 2'b11;
                2'b11: data <= 2'b00;
        endcase
endmodule
```

Unless you have written a specific case for each address you should include in the case statement:

default: data <= 2'bXX;

Block ROM using Verilog Code (Asynchronous)

- **Block ROM**

```
module rom (addr, data);
input [3:0] addr;
output [7:0] data;
reg [7:0] dout_r;
assign dout = dout_r;

always @(addr)
        case (addr)
                8'd0: dout_r <= 8'd7;
                8'd1: dout_r <= 8'd6;
                8'd7: dout_r <= 8'd0;
        endcase
endmodule
```