Block RAM/ROM

Acknowledgements: Theodoros Konstantakopoulos
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

Component Name: ram

Port Configuration:
- Read And Write
- Read Only

Memory Size:
- Width: 2
- Depth: 4

Valid Range:
- Width: 1..256
- Depth: 2..2037152

Write Mode:
- Read After Write
- Read Before Write
- No Read On Write

Click "Next"
Add Optional Control Pins (if desired)

Click “Next”
Select Polarity of Control Pins
Default is Active High

Click “Next”
Click to name a .coe file that contains initial contents (e.g., for a ROM)

Click “Generate” to Complete
Block RAM/ROM Contents

- .coe file looks like:

```plaintext
memory_initialization_radix=2;
memory_initialization_vector=
00000000,
00111110,
01100011,
00000011,
00000011,
00011110,
00000011,
00000011,
01100011,
00111110,
00000000,

Addr 0
Addr 1
Addr 2
.....
```

Specify input radix

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

### Generated Module looks like:

```verilog
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
```

```plaintext
Instantiate instances in labkit.v using:
```ram my_bram (
    .addr(my_addr),
    .clk(my_clk),
    .din(my_din),
    .dout(my_dout),
    .we(my_we)
);
```
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

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:

```verilog
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

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

ROM delay