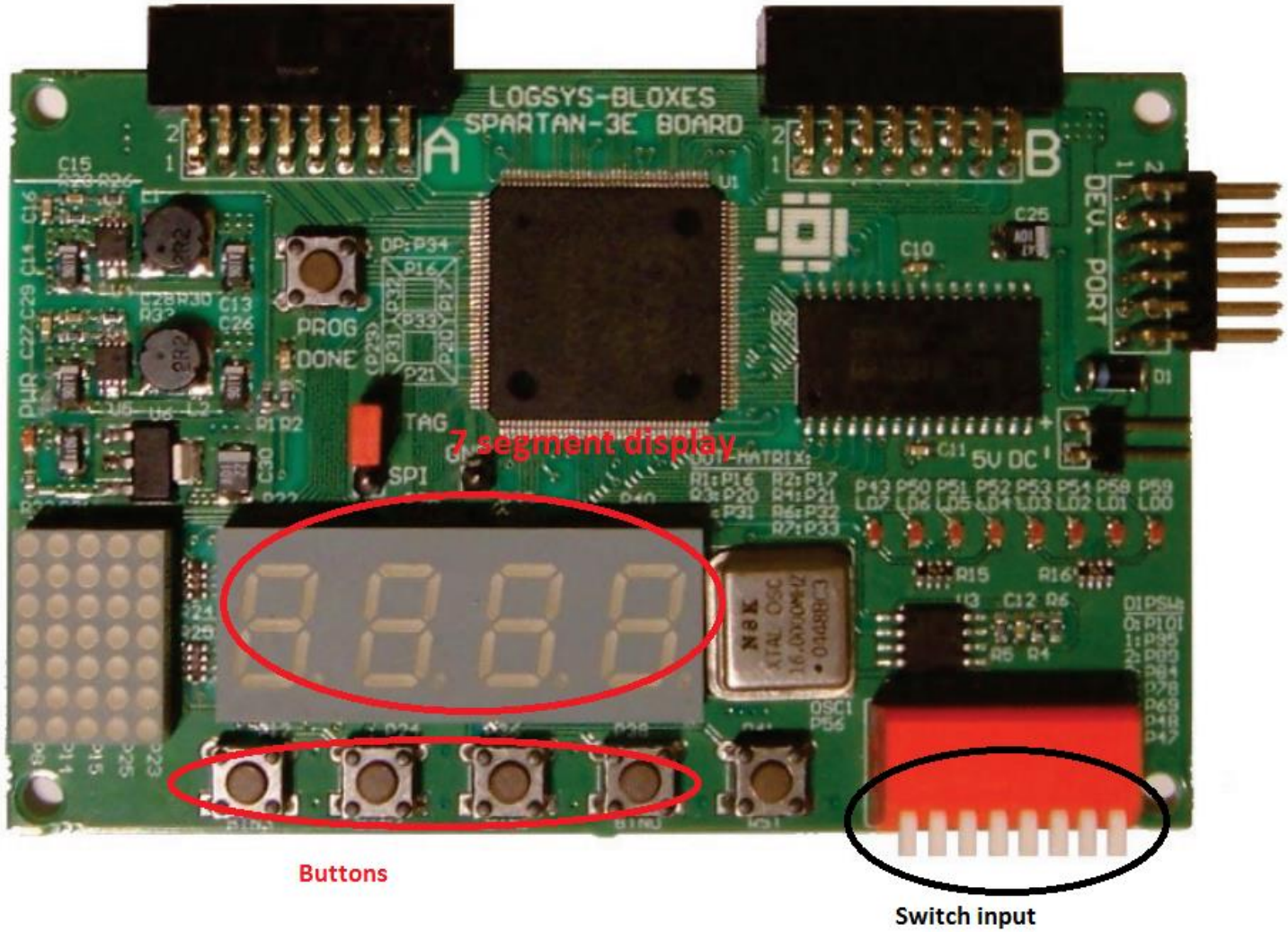
Digital design laboratory 7

Counters

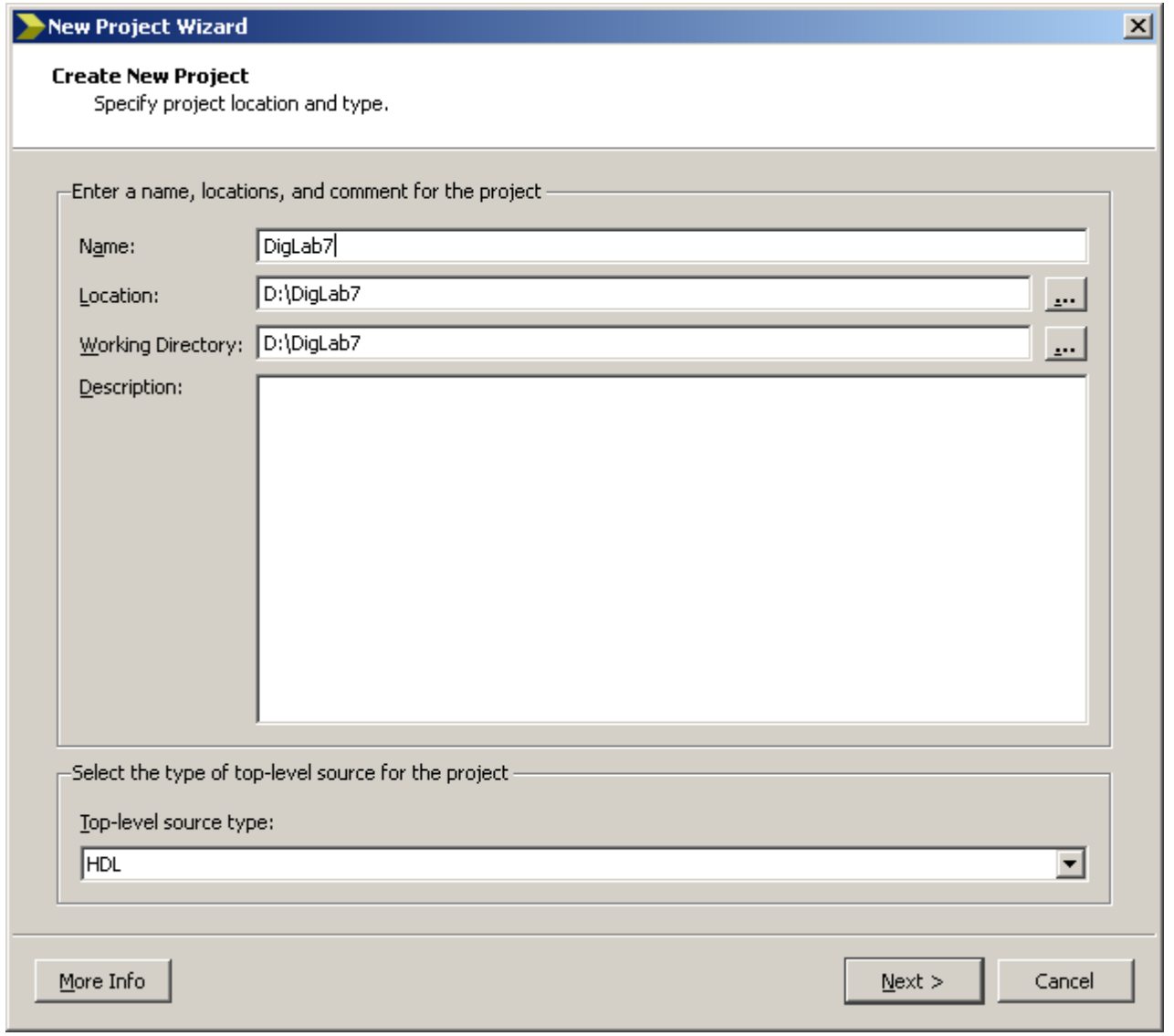
- Today we are going to implement a 16 bit counter using four 4-bit counters
- The output of the counters will be displayed on the 7 segment led display
- The counter can be enabled using the buttons of the FPGA
- The value set on the switches can be loaded into the first two counters (lower 8 bit).

Counters


LOGSYS-BLOXES
SPARTAN-3E BOARD
A
B
DEV. PORT
PROG
DONE
TAG
SPI
5V DC
7 segment display
Buttons
Switch input

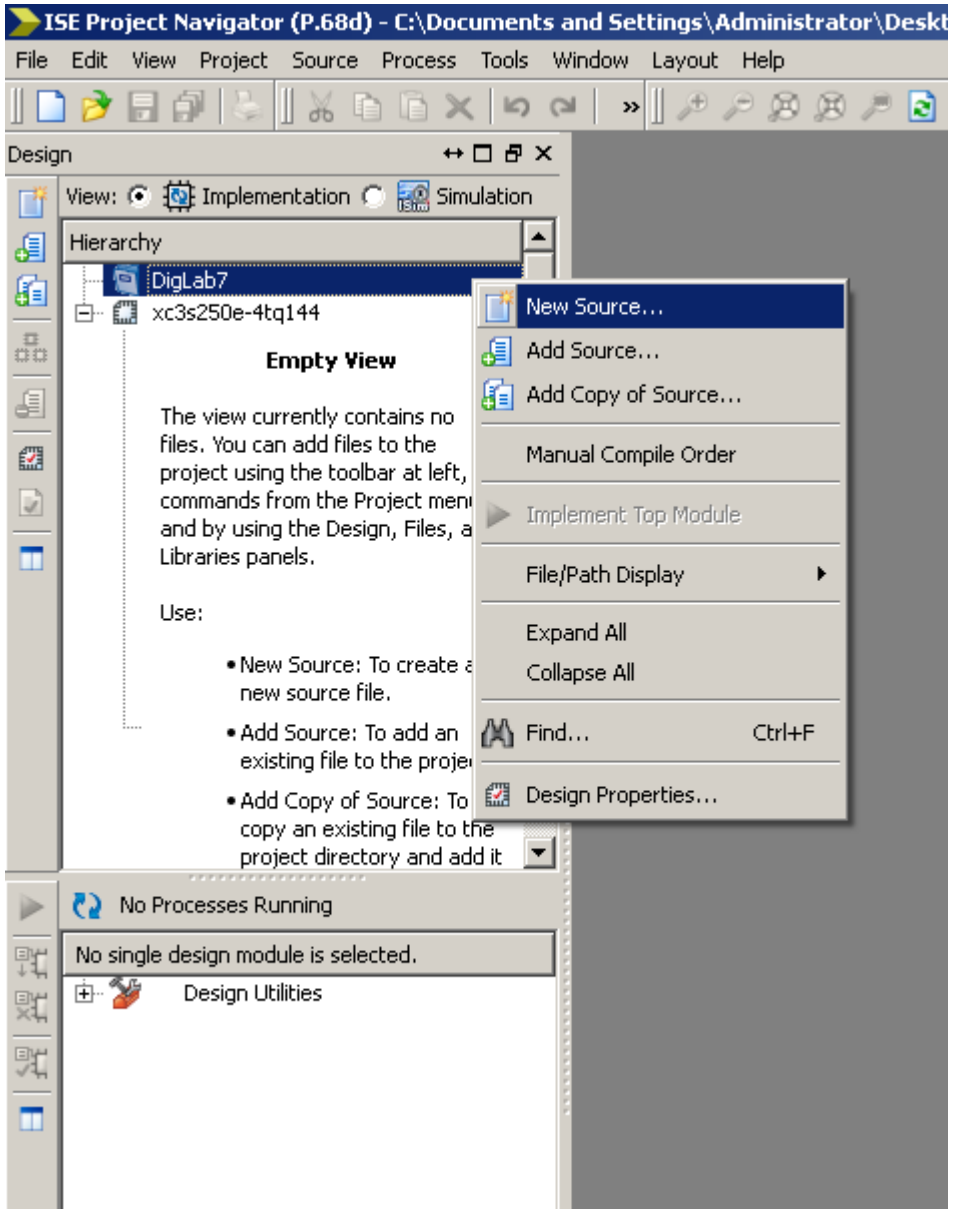
Counters

- Launch Xilinx ISE Design Suite
- File->New Porject…
- Name: DigLab7
- Location: D:\DigLab7
- Press Next twice
- Press Finish



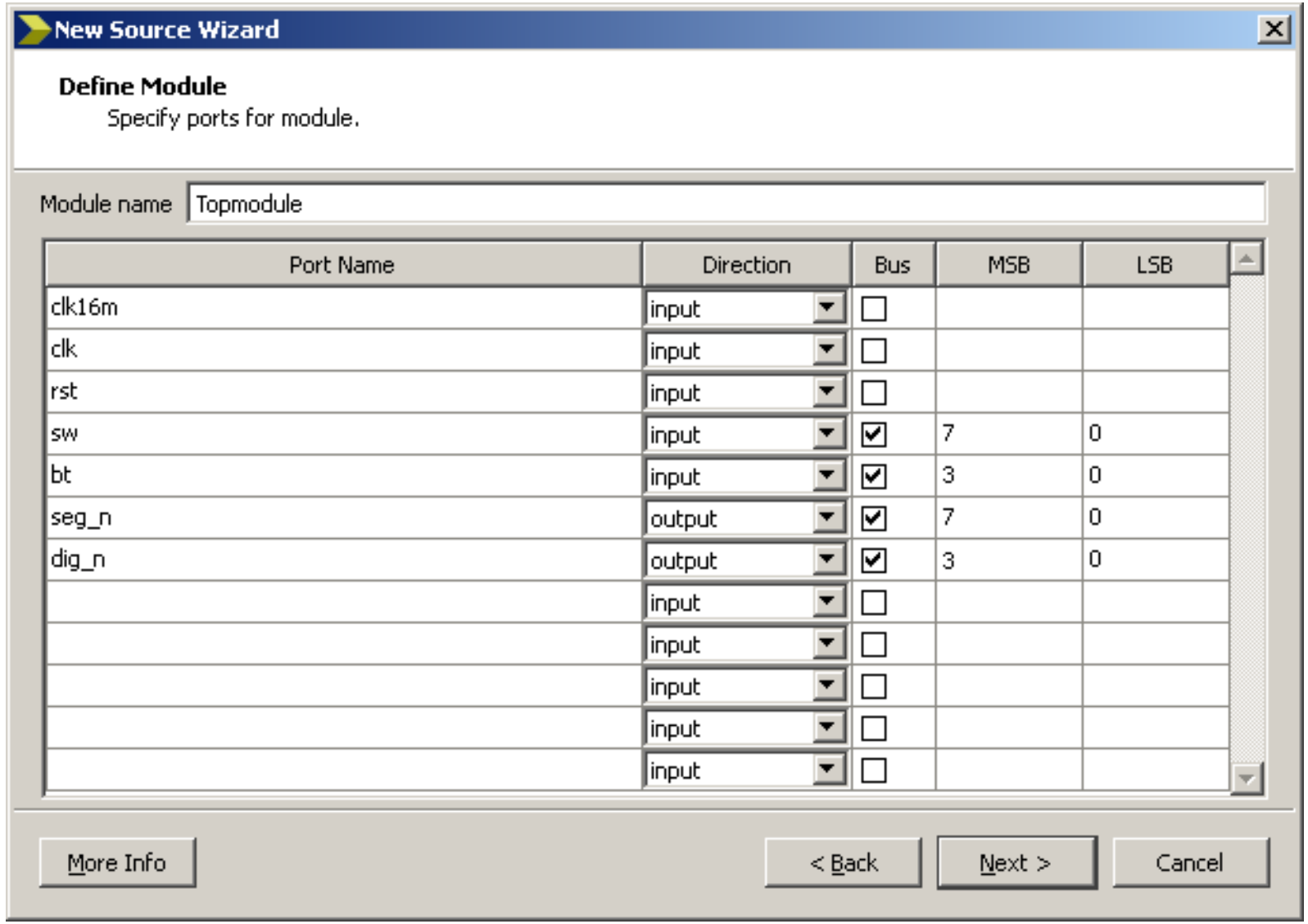
Counters

- Add the Topmodule
- Right click on the project
- Select New Source…



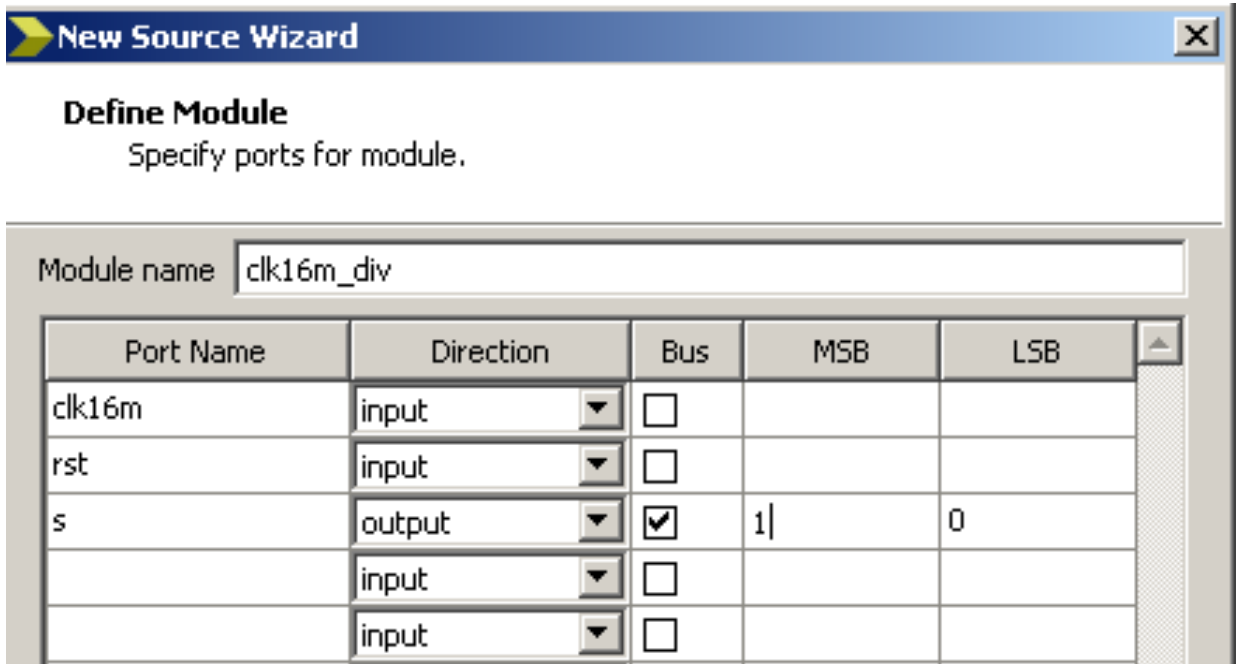
Counters

- Select Verilog Module
- Name: Topmodule
- Press next
- Inputs:
  clk16m, clk, rst,
  sw [7:0], bt[3:0]
- Outputs:
- seg_n [7:0],
  dig_n [3:0]
- Press Next, then Finish



Counters

- To use the 7 segment display, we need a 4x1 multiplexer and a decoder. First we need a select signal to control these circuits.
- The refresh rate of the LEDs will be 1 kHz
- The 16 MHz clk16m signal will be used
- Add a new module to the project
- Name: clk16m_div
- Input: clk16m, rst
- Output: s [1:0]



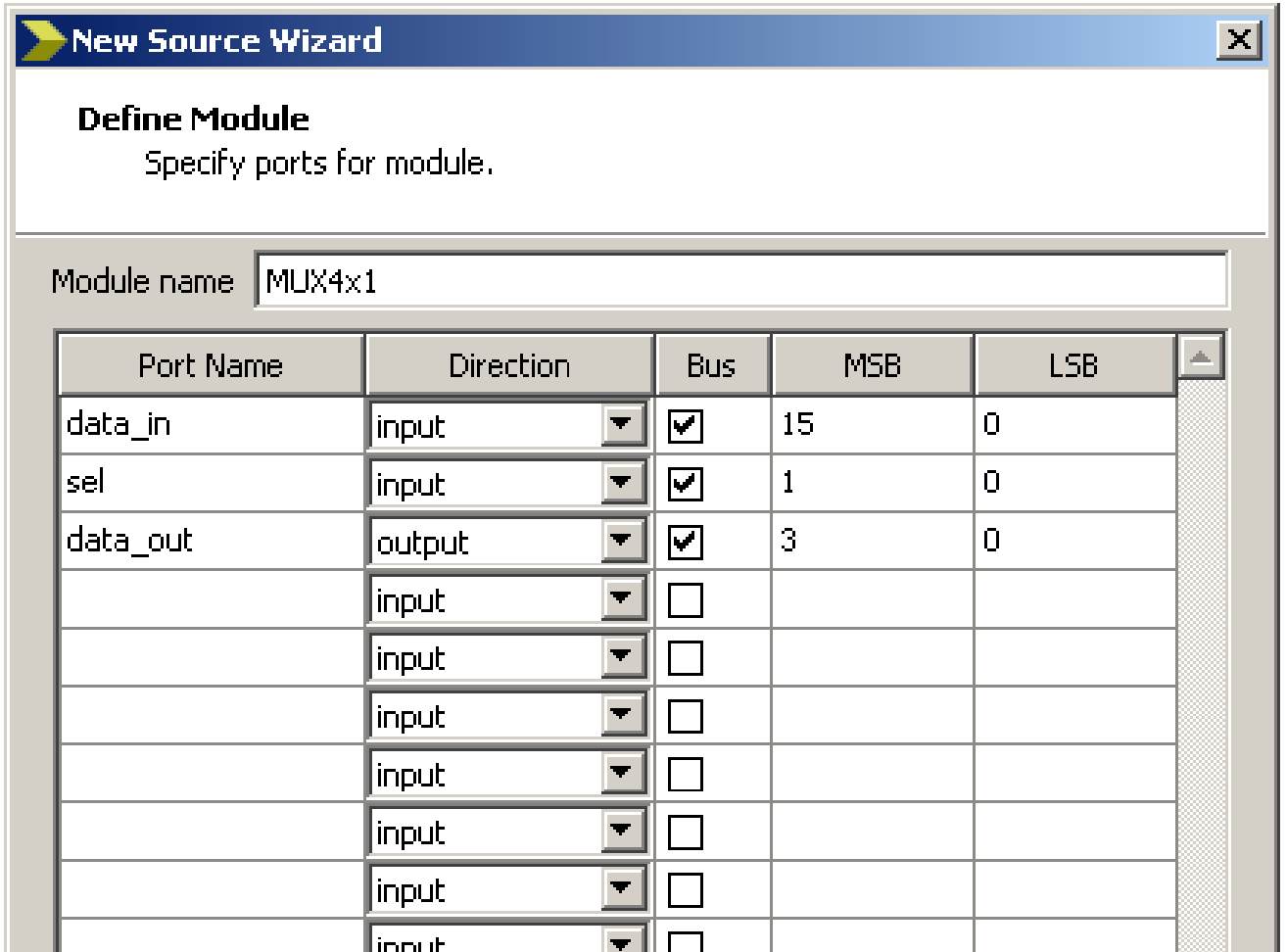
Counters

- Define the s output as a register
- Add the following source code to clk16_div

```
21 module clk16m_div(
22     input clk16m,
23     input rst,
24     output reg [1:0] s
25     );
26 
27    reg [15:0] count_reg;
28    wire count_16000;
29 
30    assign count_16000 = (count_reg == 16'd15999);
31 
32    always @ (posedge clk16m)
33    if (rst)
34       begin
35          count_reg <= 16'd0;
36          s <= 2'd0;
37       end
38    else if (count_16000)
39       begin
40          count_reg <= 16'd0;
41          s <= s + 2'd1;
42       end
43    else count_reg <= count_reg + 1;
```

Counters

- Now implement the 4-bit 4x1 MUX circuit
- Add module: MUX4x1
- Inputs:
  data_in [15:0], sel [1:0]
- Output:
  data_out [3:0]



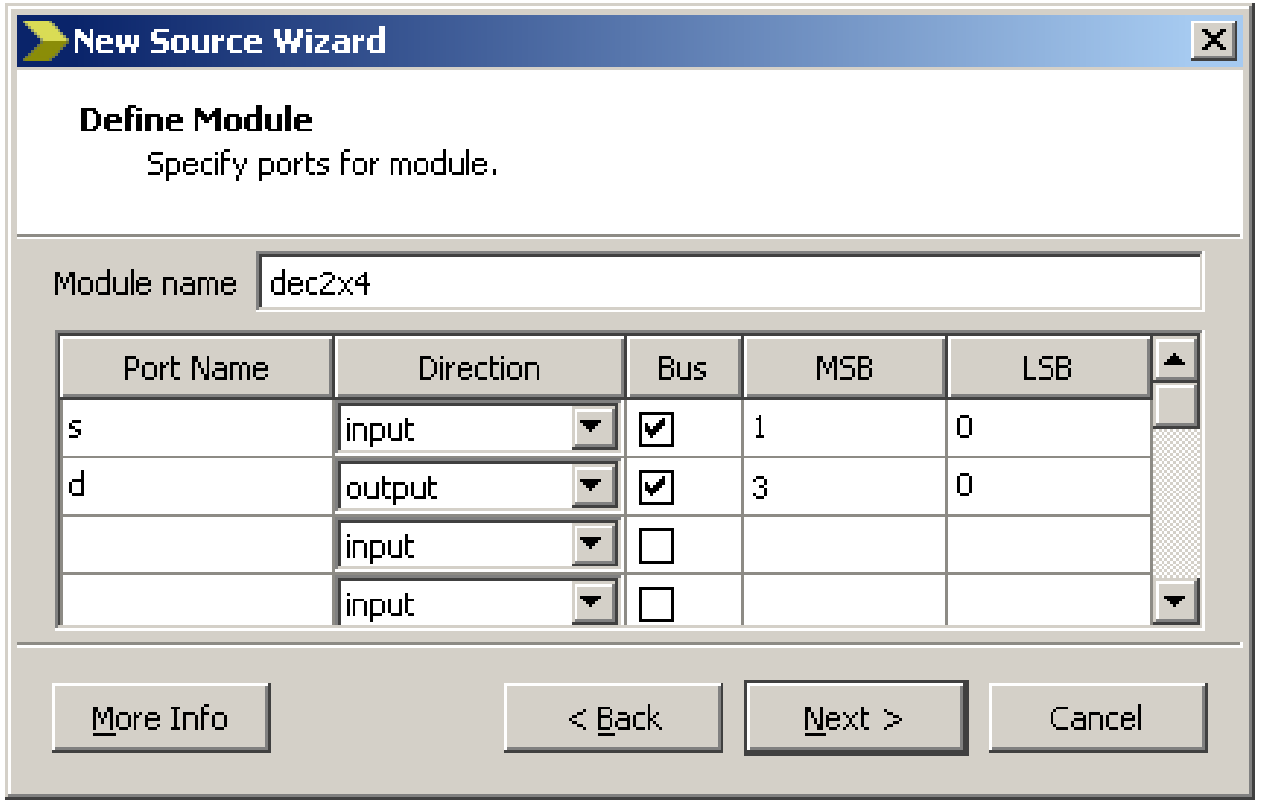
Counters

- Multiplexer implementation:

```
21 module MUX4x1(
22     input [15:0] data_in,
23     input [1:0] sel,
24     output reg [3:0] data_out
25     );
26 
27     always @ (*)
28     case (sel)
29        2'b00: data_out <= data_in[3:0];
30        2'b01: data_out <= data_in[7:4];
31        2'b10: data_out <= data_in[11:8];
32        2'b11: data_out <= data_in[15:12];
33     endcase
34 
35 
36 endmodule
```

Counters

- Now add the decoder module
- New source -> Verilog module
- Name: dec2x4



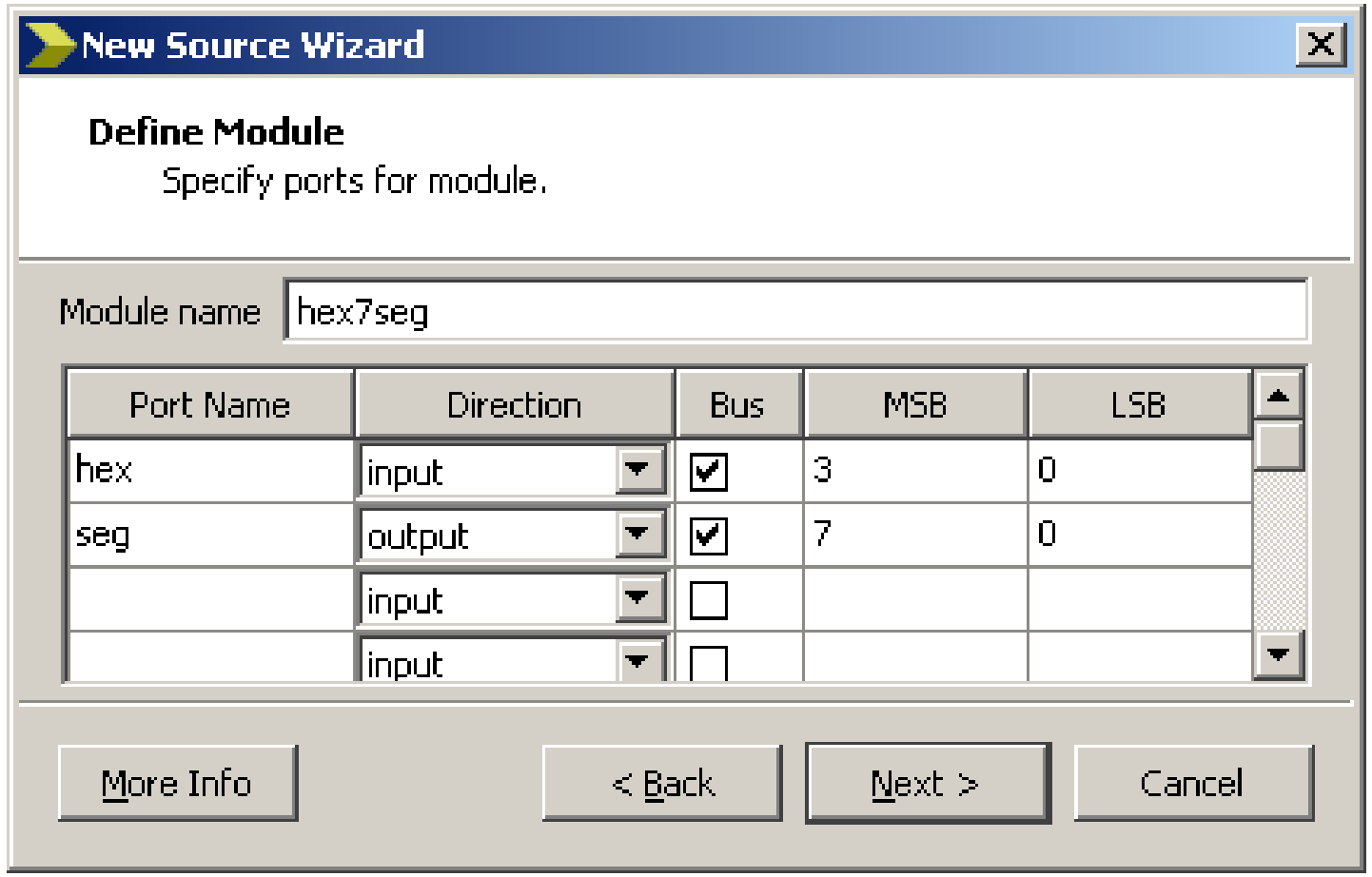
Counters

- Decoder implementation:

```
21 module dec2x4(
22     input [1:0] s,
23     output reg [3:0] d
24     );
25
26    always @ (*)
27    case (s)
28       2'b00: d <= 4'b0001;
29       2'b01: d <= 4'b0010;
30       2'b10: d <= 4'b0100;
31       2'b11: d <= 4'b1000;
32    endcase
33
34 endmodule
35
```

Counters

- We need a converter for the 7-segment display
- Add new source
- Name: Hex7Seg
- Input: hex [3:0]
- Output seg [7:0]



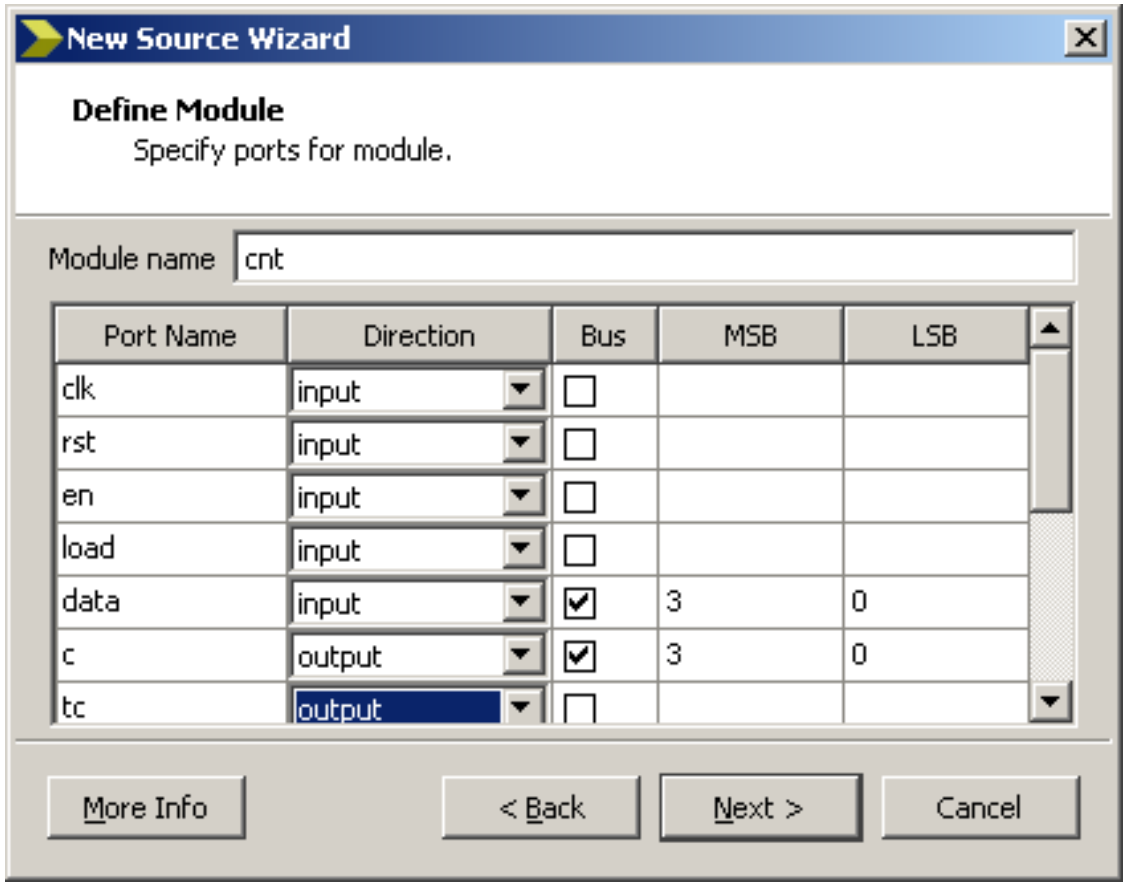
Counters

- Add the reg keyword to the seg output
- Copy&Paste the code from the next slide

```
always @ (*)
        case (hex)
                4'b0000 : seg <= 8'b00111111;   // 0
                4'b0001 : seg <= 8'b00000110;   // 1
                4'b0010 : seg <= 8'b01011011;   // 2
                4'b0011 : seg <= 8'b01001111;   // 3
                4'b0100 : seg <= 8'b01100110;   // 4
                4'b0101 : seg <= 8'b01101101;   // 5
                4'b0110 : seg <= 8'b01111101;   // 6
                4'b0111 : seg <= 8'b00000111;   // 7
                4'b1000 : seg <= 8'b01111111;   // 8
                4'b1001 : seg <= 8'b01101111;   // 9
                4'b1010 : seg <= 8'b01110111;   // A
                4'b1011 : seg <= 8'b01111100;   // b
                4'b1100 : seg <= 8'b00111001;   // C
                4'b1101 : seg <= 8'b01011110;   // d
                4'b1110 : seg <= 8'b01111001;   // E
                4'b1111 : seg <= 8'b01110001;   // F
                default : seg <= 8'b00000000;   // 0
        endcase
```

Counters

- Now lets implement a four-bit counter circuit
- New source -> Verilog Module
- Name cnt
- Inputs:
  clk, rst, load, en, data [3:0]
- Outputs:
  c [3:0], tc



Counters

- Add the reg keyword to the c output
- Source:

```
21 module cnt(
22     input clk,
23     input rst,
24     input en,
25     input load,
26     input [3:0] data,
27     output reg [3:0] c,
28     output tc
29     );
30 
31 
32    assign tc = (c == 4'b1111);
33 
34    always @ (posedge clk)
35    if (rst) c <= 4'd0;
36    else if (load)
37       c <= data;
38    else if (en)
39       c <= c + 4'd1;
40 
41 endmodule
```

Counters

- Now go back to the Topmodule file
- The next task is the instantiation of the previoud modules
- We start with the clk16m_div module
- Add the following lines to the Topmodule:

```
27      output [7:0] seg_n,
28      output [3:0] dig_n
29      );
30
31      wire [1:0] sel;
32
33      clk16m_div d(.clk16m(clk16m), .rst(rst), .s(sel));
34
35
```

Counters

- Now add the multiplexer circuit:

```
wire [1:0] sel;
wire [15:0] mux_in;
wire [3:0] mux_out;

clk16m_div d(.clk16m(clk16m), .rst(rst), .s(sel));
MUX4x1 m(.data_in(mux_in), .sel(sel), .data_out(mux_out));
```

Counters

- Next we instantiate the decoder and the hex7seg converter:

```
wire [1:0] sel;
wire [15:0] mux_in;
wire [3:0] mux_out;
wire [7:0] seg;
wire [3:0] dig;

assign dig_n = ~dig;
assign seg_n = ~seg;

clk16m_div d(.clk16m(clk16m), .rst(rst), .s(sel));
MUX4x1 m(.data_in(mux_in), .sel(sel), .data_out(mux_out));
dec2x4 dec(.s(sel), .d(dig));
hex7seg h(.hex(mux_out), .seg(seg));
```

Counters

- Finally, implement the 4 counters:

```
wire [3:0] dig;
wire en, load;
wire [2:0] tc;
assign en = bt[0]|bt[1];
assign load = bt[2]|bt[3];

assign dig_n = ~dig;
assign seg_n = ~seg;

clk16m_div d(.clk16m(clk16m), .rst(rst), .s(sel));
MUX4x1 m(.data_in(mux_in), .sel(sel), .data_out(mux_out));
dec2x4 dec(.s(sel), .d(dig));
hex7seg h(.hex(mux_out), .seg(seg));

cnt c0(.clk(clk), .rst(rst), .en(en), .load(load), .data(sw[3:0]), .c(mux_in[3:0]), .tc(tc[0]));
cnt c1(.clk(clk), .rst(rst), .en(tc[0]), .load(load), .data(sw[7:4]), .c(mux_in[7:4]), .tc(tc[1]));
cnt c2(.clk(clk), .rst(rst), .en(tc[1]&tc[0]), .load(load), .data(4'd0), .c(mux_in[11:8]), .tc(tc[2]));
cnt c3(.clk(clk), .rst(rst), .en(tc[2]&tc[1]&tc[0]), .load(load), .data(4'd0), .c(mux_in[15:12]), .tc());
```

Counters

- Now you have to add the .ucf file
- Download it: link
- Unzip the file to the project directory
- Add it to the project with add source or add copy of source
- Uncomment the following lines:
  clk16m, clk, rst, bt (all), sw (all), seg_n (all), dig_n (all)
- Now generate the programming file
- After downloading, add CLK and test the counter

Counters

- Try to add some functionality on your own:
  1. Modify the counter and the Topmodule:
     bt[0]: enable and count up, bt[1]: enable and count down,
     bt[2]: load value from switches into c0 and c1 counters
     bt[3] load value from switches into c2 and c3 counters