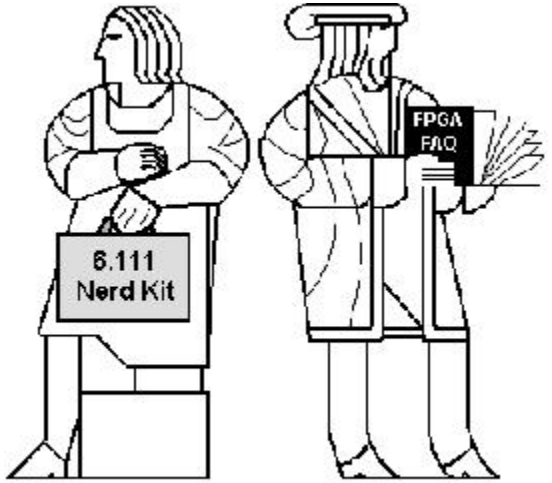
6.111 Lecture 7

Today:

Demo! (or die): An Electronic Lock

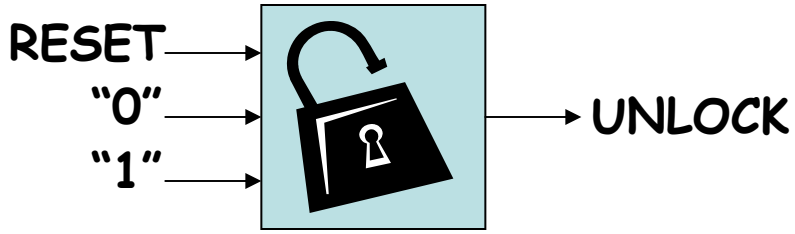
1. Design FSM
2. Implement in Verilog
3. Compile: Xilinx tool-chain
4. Program labkit



Demo!

GOAL:

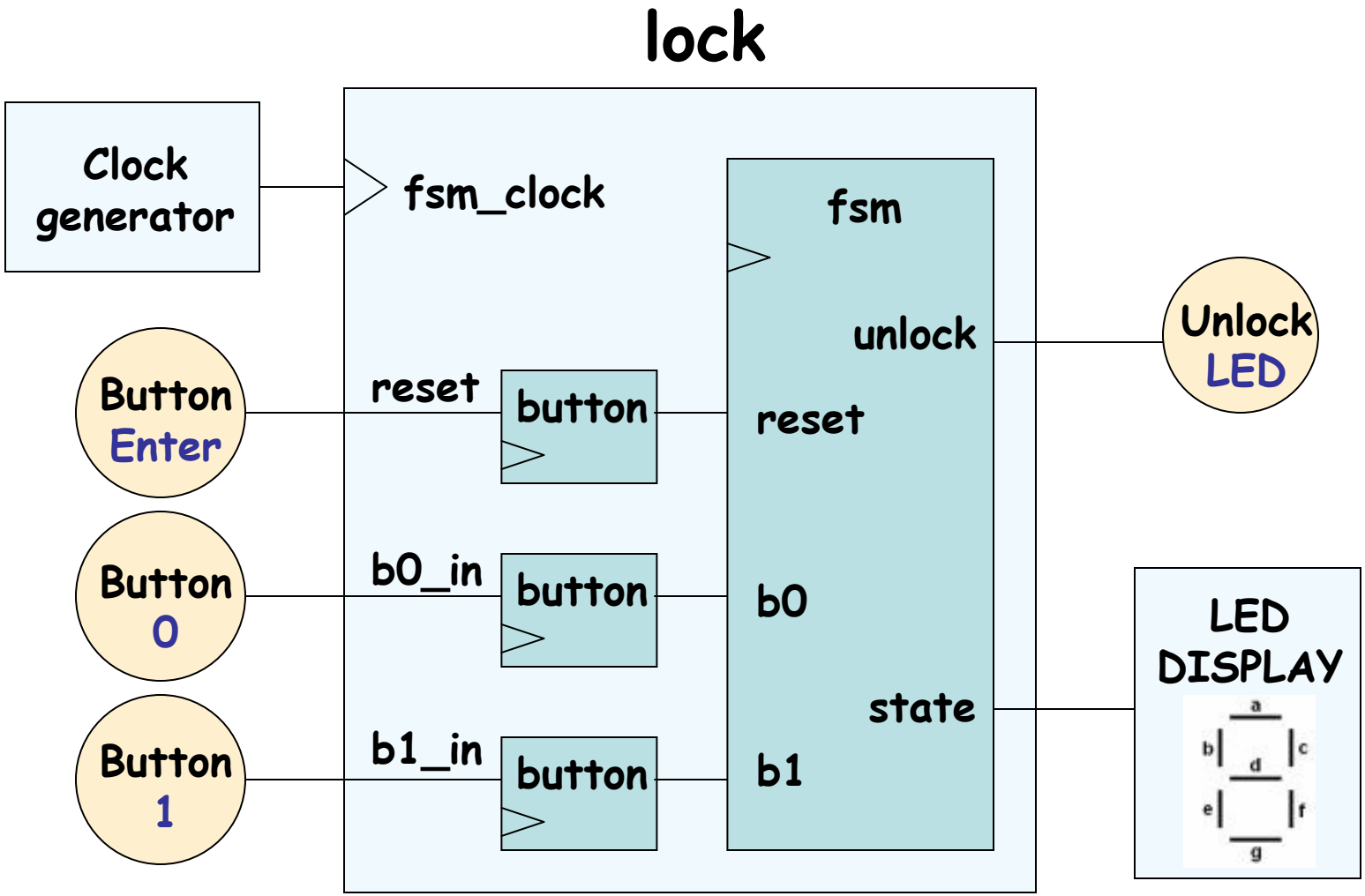
Build an electronic combination lock with a reset button, two number buttons (0 and 1), and an unlock output. The combination should be 01011.



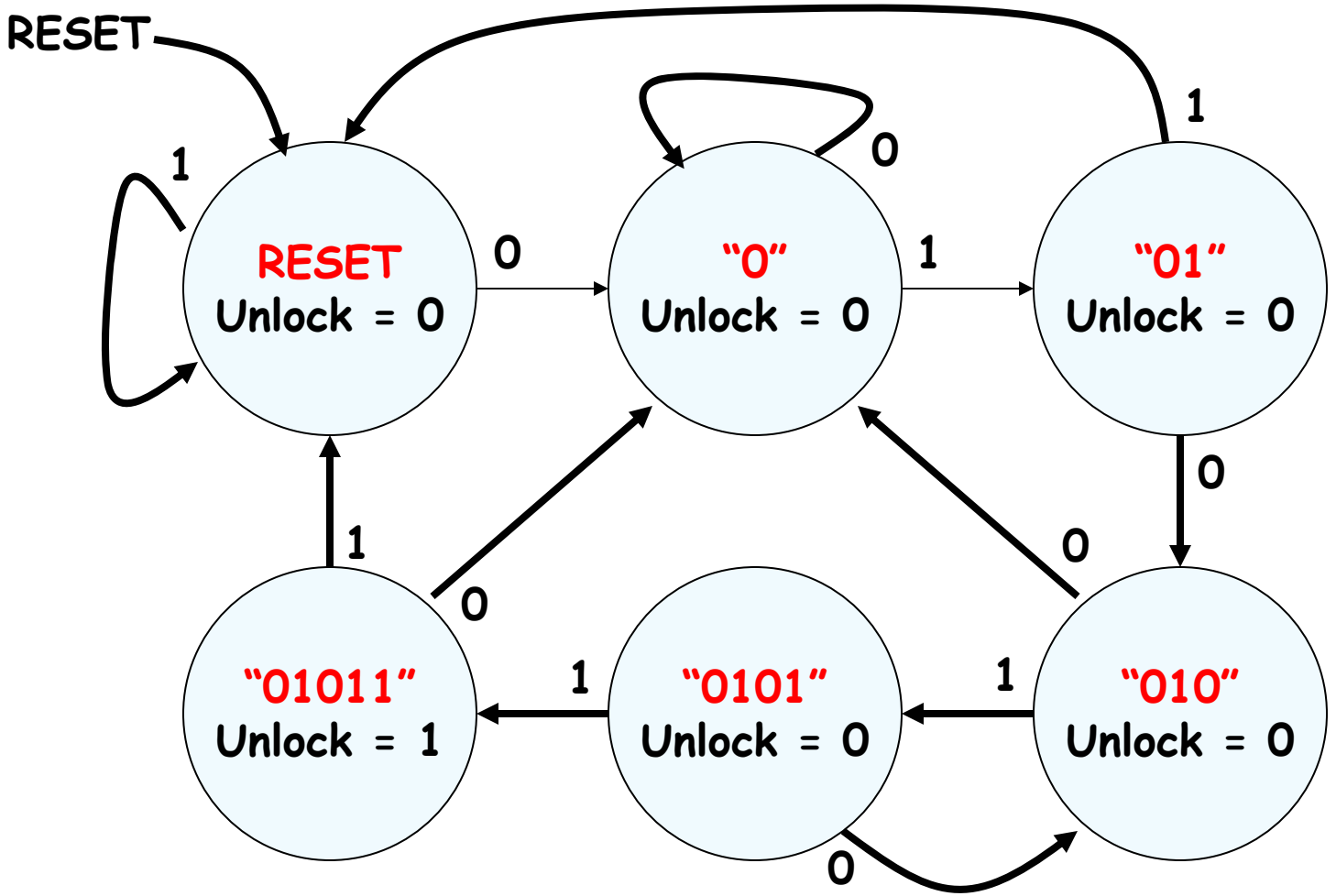
STEPS:

1. Design lock FSM (block diagram, state transitions)
2. Write Verilog module(s) for FSM
3. Use Xilinx ISE7.1 (synthesis, simulation)
4. Program FGPA, give it a whirl!

Step 1A: Block Diagram



Step 1B: State transition diagram



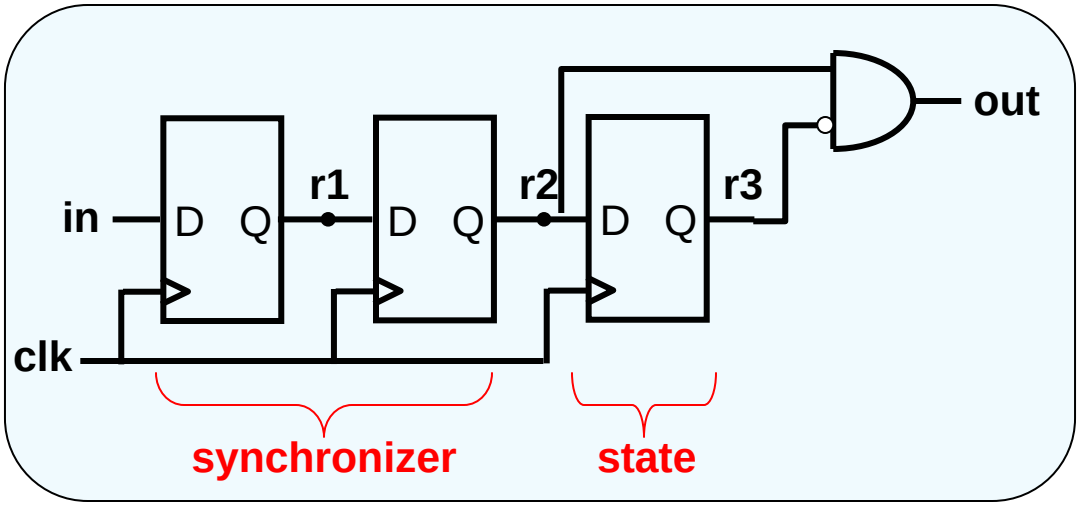
6 states → 3 bits

Step 2: Write Verilog

```
module lock(clk,reset_in,b0_in,b1_in,out);

  input clk,reset,b0_in,b1_in;
  output out;

  // synchronize push buttons, convert to pulses

  // implement state transition diagram
  reg [2:0] state;
  always @ (posedge clk)
  begin
    state <= ???;
  end

  // generate output
  assign out = ???;

  // debugging?
endmodule
```

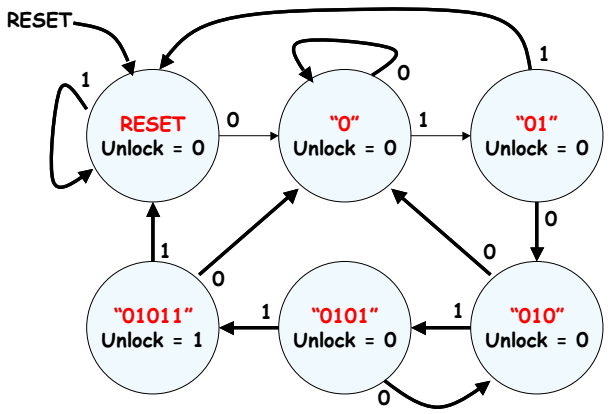
Step 2A: Synchronize buttons

```
// button -- push button synchronizer and level-to-pulse converter
// OUT goes high for one cycle of CLK whenever IN makes a
// low-to-high transition.

module button(clk,in,out);
  input clk;
  input in;
  output out;

  reg r1,r2,r3;
  always @ (posedge clk)
  begin
    r1 <= in;   // first reg in synchronizer
    r2 <= r1;   // second reg in synchronizer, output is in sync!
    r3 <= r2;   // remembers previous state of button
  end

  // rising edge = old value is 0, new value is 1
  assign out = ~r3 & r2;
endmodule
```



Step 2B: state transition diagram

```
parameter S_RESET = 0; // state assignments
parameter S_0 = 1;
parameter S_01 = 2;
parameter S_010 = 3;
parameter S_0101 = 4;
parameter S_01011 = 5;

reg [2:0] state;
always @ (posedge clk)
begin      // implement state transition diagram
  if (reset) state <= S_RESET;
  else case (state)
    S_RESET: state <= b0 ? S_0   : b1 ? S_RESET : state;
    S_0:     state <= b0 ? S_0   : b1 ? S_01    : state;
    S_01:    state <= b0 ? S_010 : b1 ? S_RESET : state;
    S_010:   state <= b0 ? S_0   : b1 ? S_0101  : state;
    S_0101:  state <= b0 ? S_010 : b1 ? S_01011 : state;
    S_01011: state <= b0 ? S_0   : b1 ? S_RESET : state;
    default: state <= S_RESET;  // handle unused states
  endcase
end
```



Step 2C: generate output

```
// it's a Moore machine!  Output only depends on current state

assign out = (state == S_01011);
```

Step 2D: debugging?

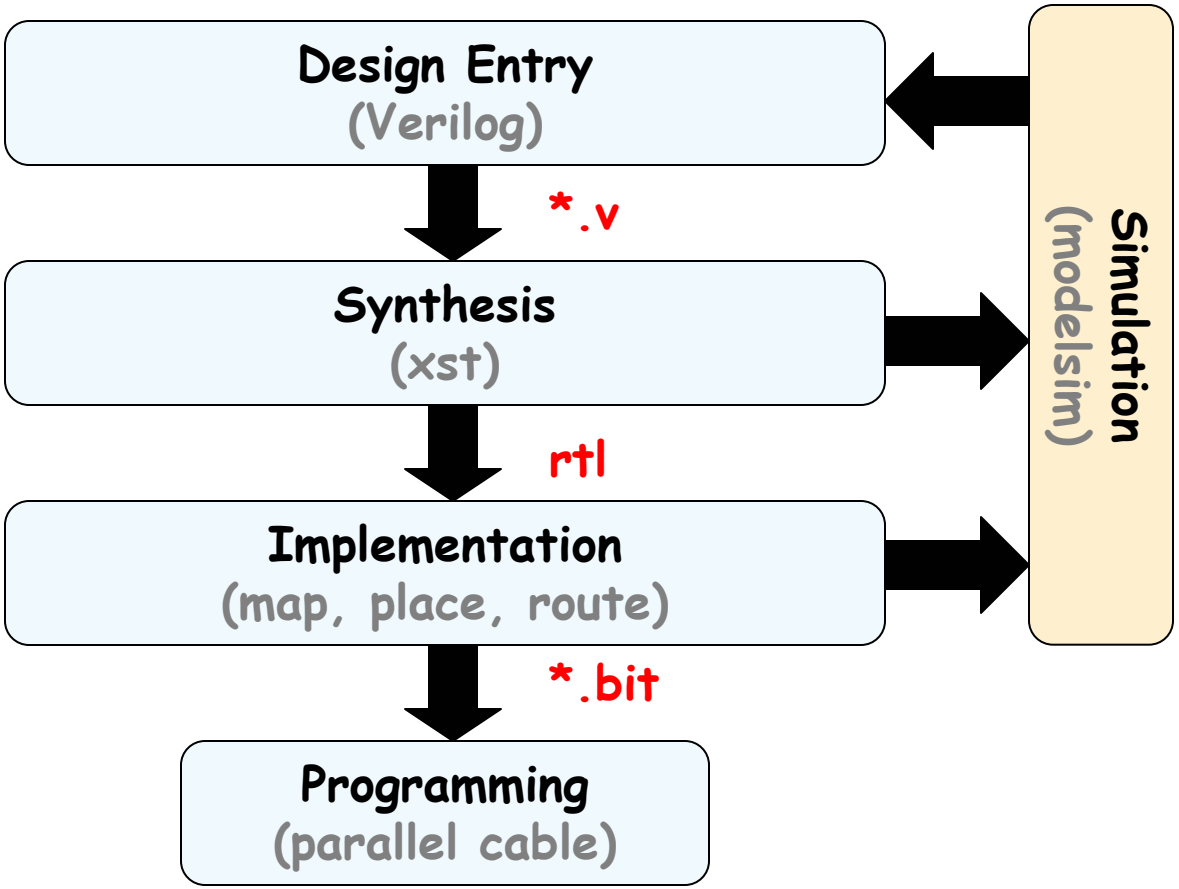
```
// hmmm.  What would be useful to know?  Current state?

assign hex_display = {1'b0,state[2:0]};
```

Step 2: final Verilog implementation

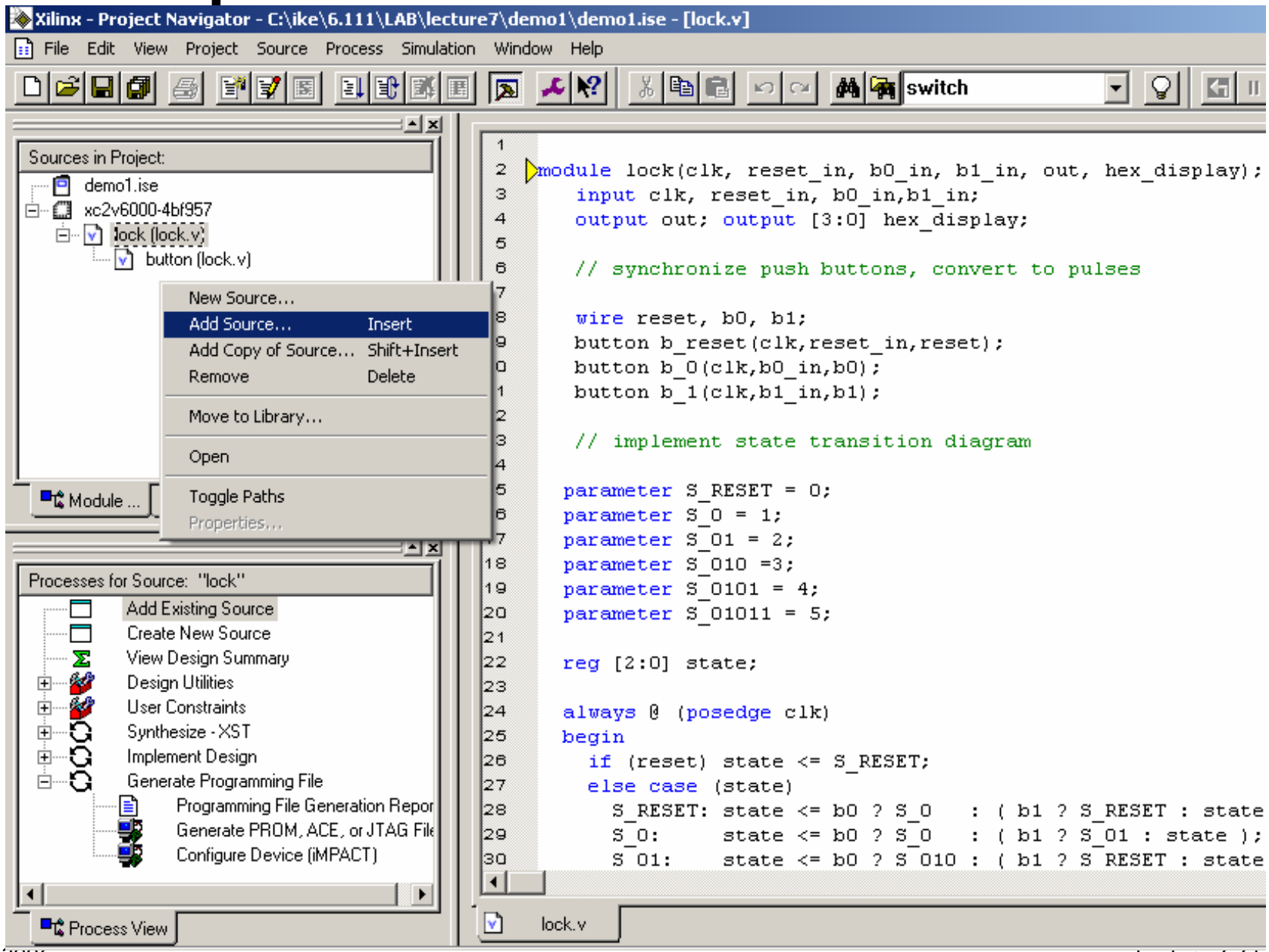
```
module lock(clk,reset_in,b0_in,b1_in,out, hex_display);

  input clk,reset,b0_in,b1_in;
  output out; output[3:0] hex_display;

  wire reset, b0, b1; // synchronize push buttons, convert to pulses
  button b_reset(clk,reset_in,reset);
  button b_0(clk,b0_in,b0);
  button b_1(clk,b1_in,b1);

  parameter S_RESET = 0; parameter S_0 = 1; // state assignments
  parameter S_01 = 2;  parameter S_010 = 3;
  parameter S_0101 = 4;  parameter S_01011 = 5;

  reg [2:0] state;
  always @ (posedge clk)
  begin                              // implement state transition diagram
    if (reset) state <= S_RESET;
    else case (state)
      S_RESET: state <= b0 ? S_0   : b1 ? S_RESET : state;
      S_0:     state <= b0 ? S_0   : b1 ? S_01    : state;
      S_01:    state <= b0 ? S_010 : b1 ? S_RESET : state;
      S_010:   state <= b0 ? S_0   : b1 ? S_0101  : state;
      S_0101:  state <= b0 ? S_010 : b1 ? S_01011 : state;
      S_01011: state <= b0 ? S_0   : b1 ? S_RESET : state;
      default: state <= S_RESET;          // handle unused states
    endcase

 assign out = (state == S_01011);        // assign output: Moore machine
 assign hex_display = {1'b0,state};      // debugging
endmodule
```

Step 3: Synthesis & Simulation

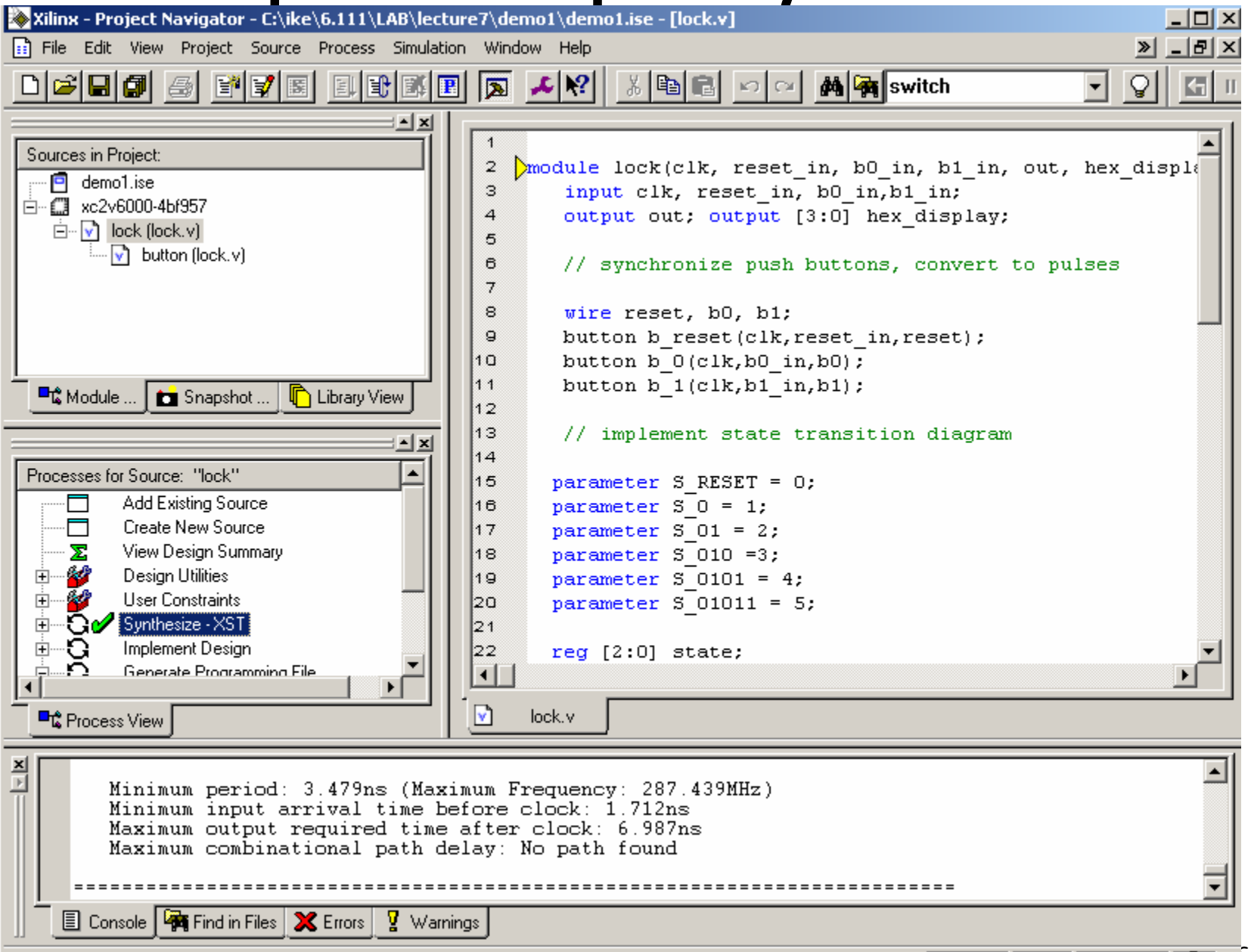
- We will be using the Xilinx toolchain
- Software: ISE 7.1i (windows / linux)



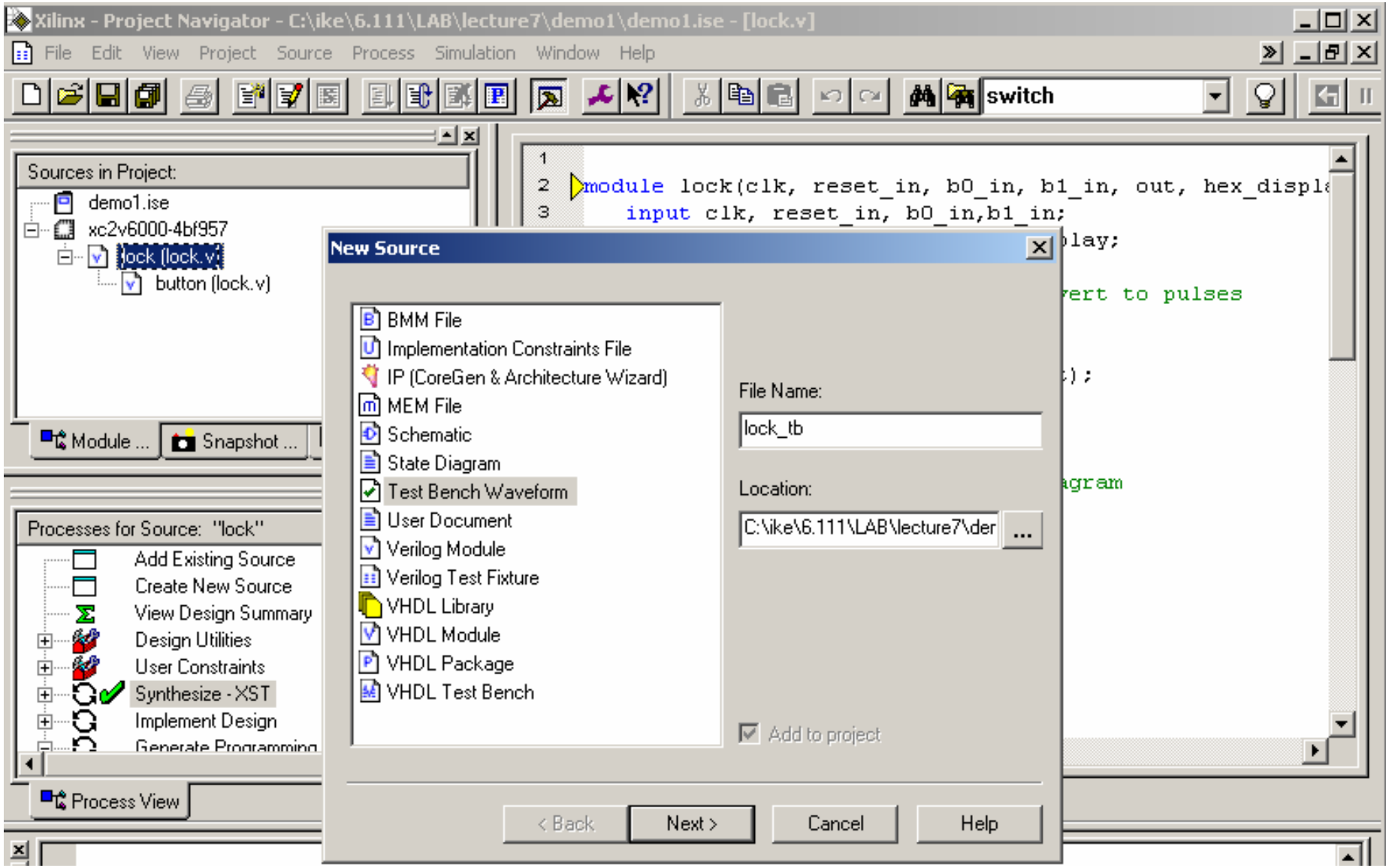
Step 3A: Load source file lock.v



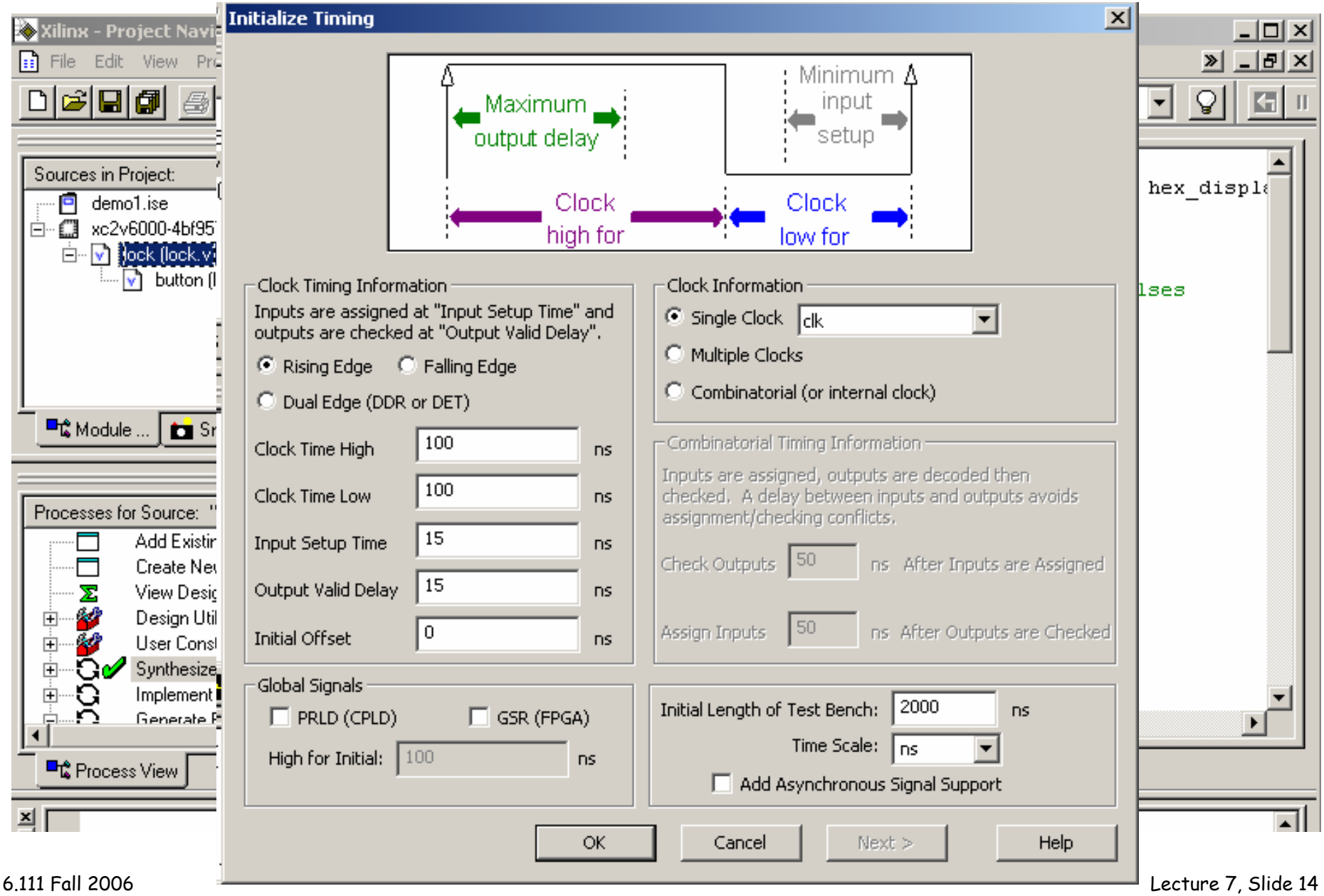
Step 3B: Compile/Synthesize



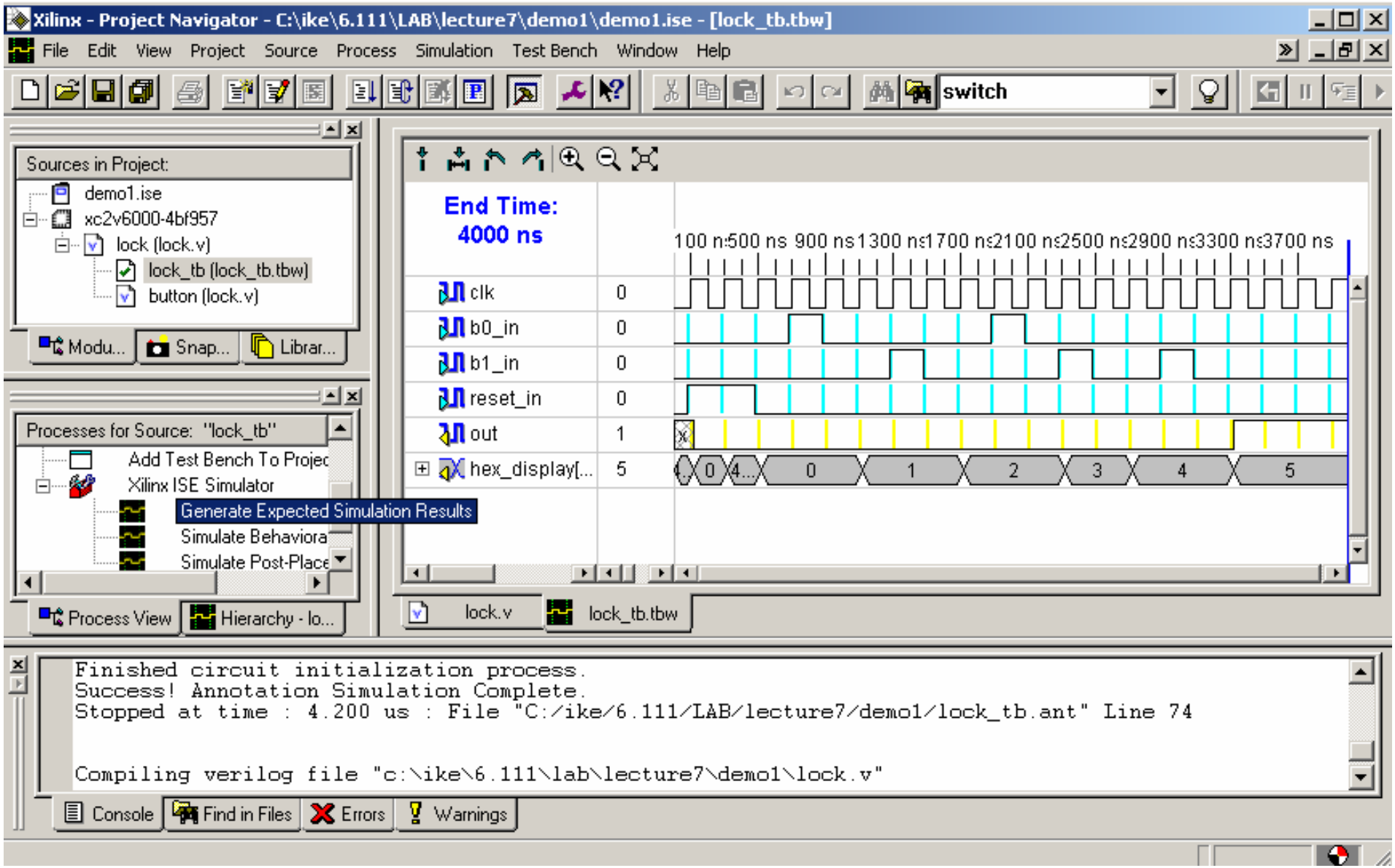
Step 3B: Create testbench



Step 3B: Create testbench

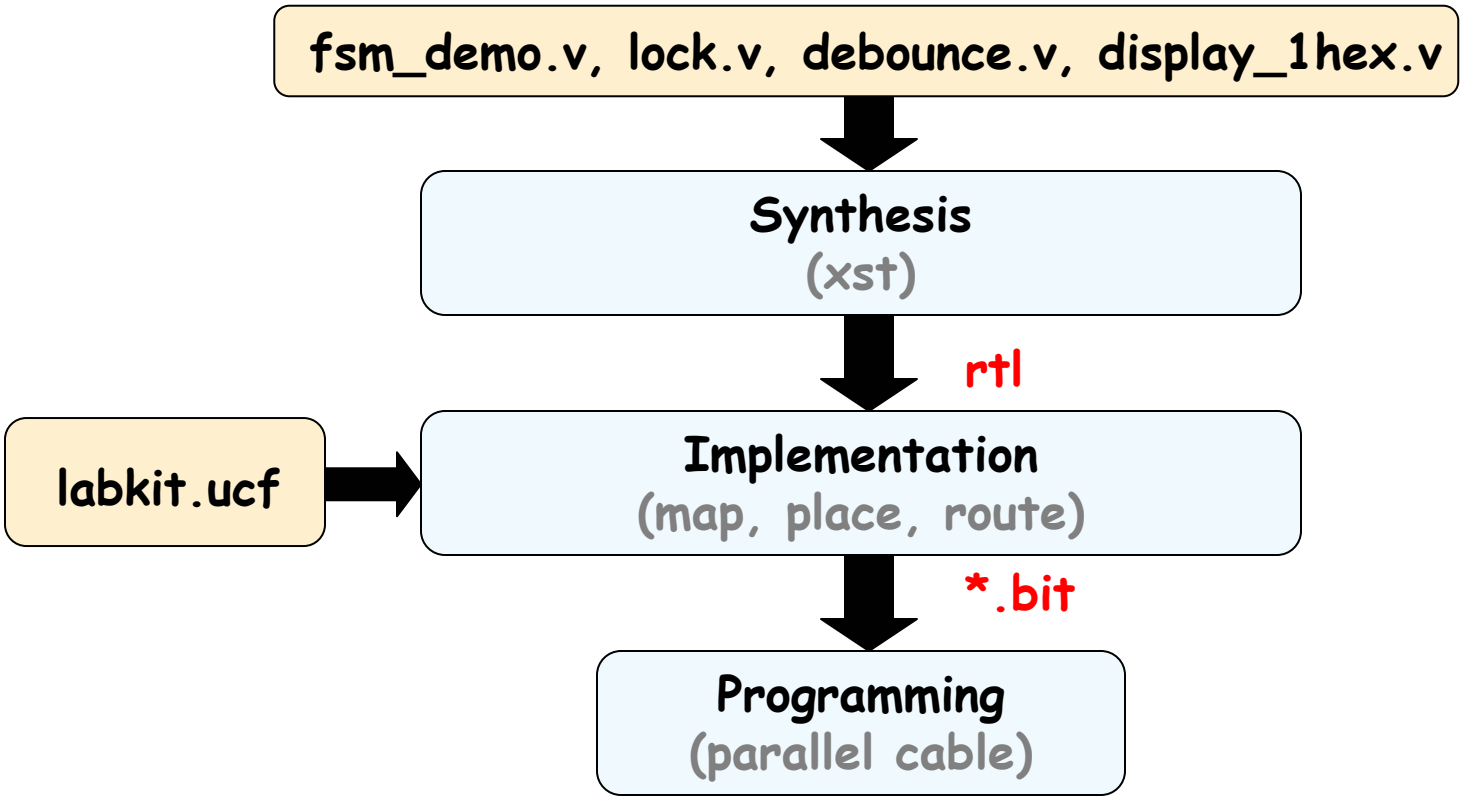


Step 3B: Generate Simulation Results



Step 4: Implementation – Program FPGA

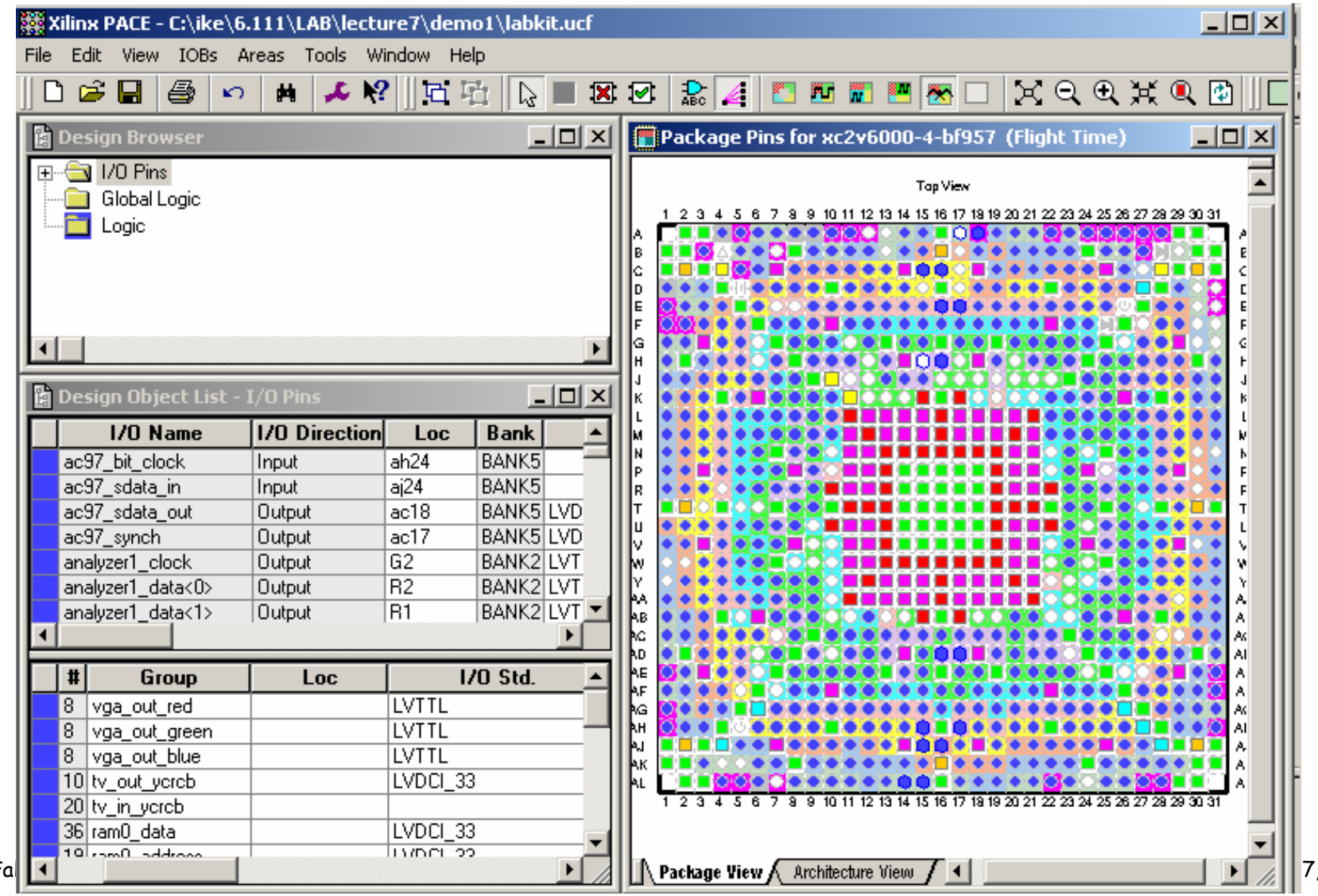
- Pin assignments: User constraints file
- fsm_demo.v (modified copy of labkit.v): peripherals definitions
- Optimization: Placing and Routing



Step 4: Implementation – Program FPGA

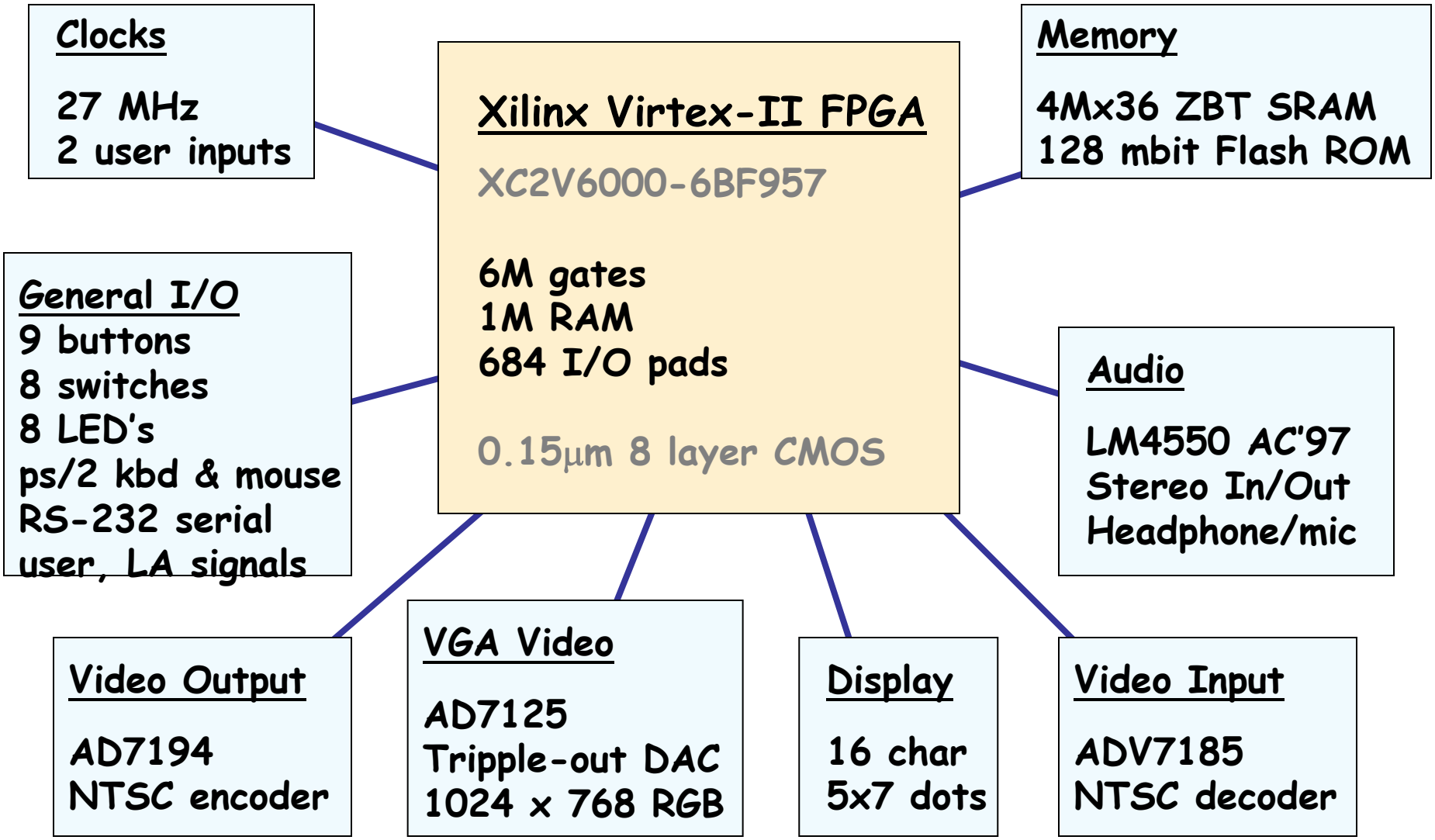
- Pin assignments: User constraints file

labkit.ucf

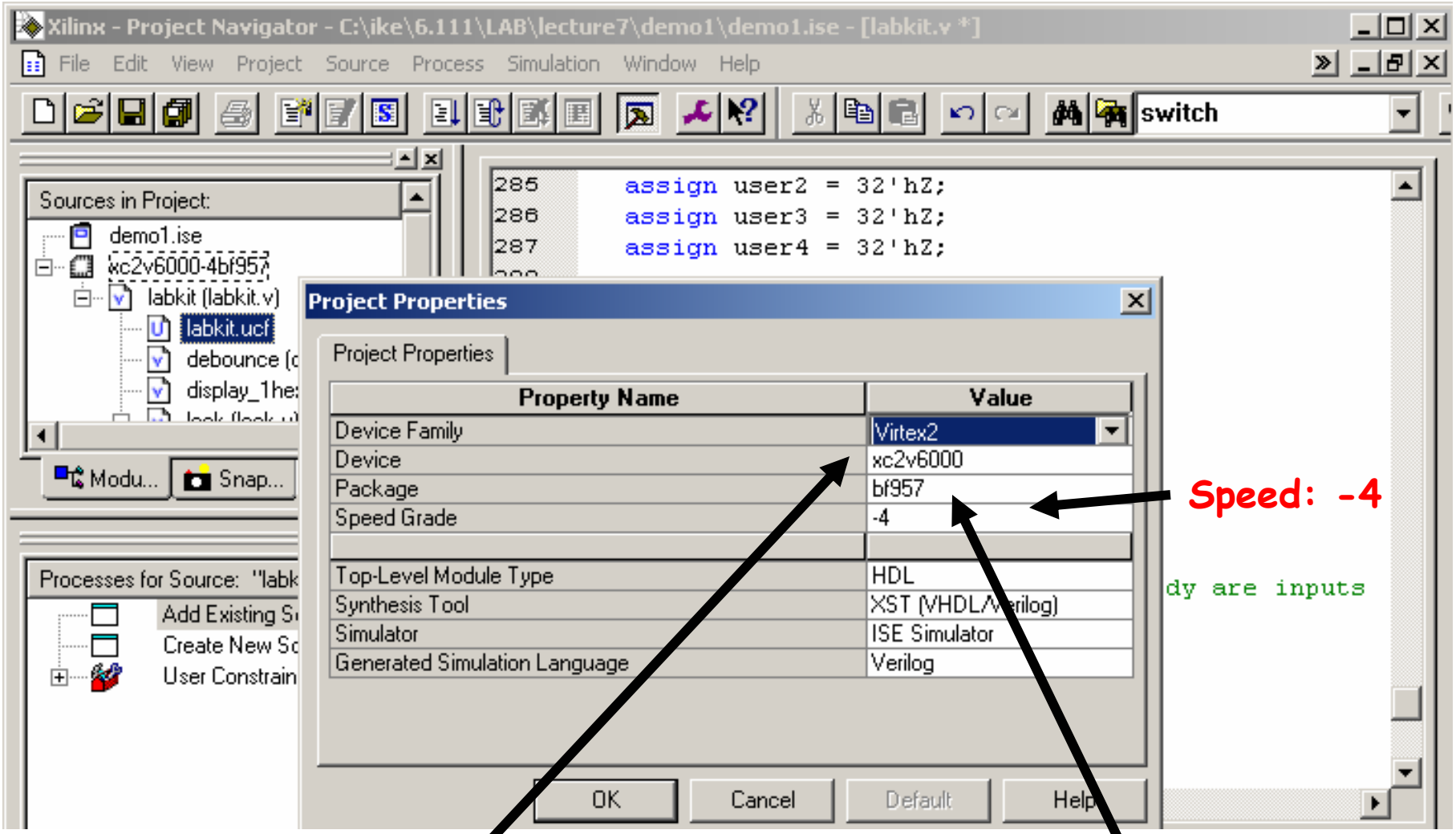


The 6.111 Labkit: Subsystems

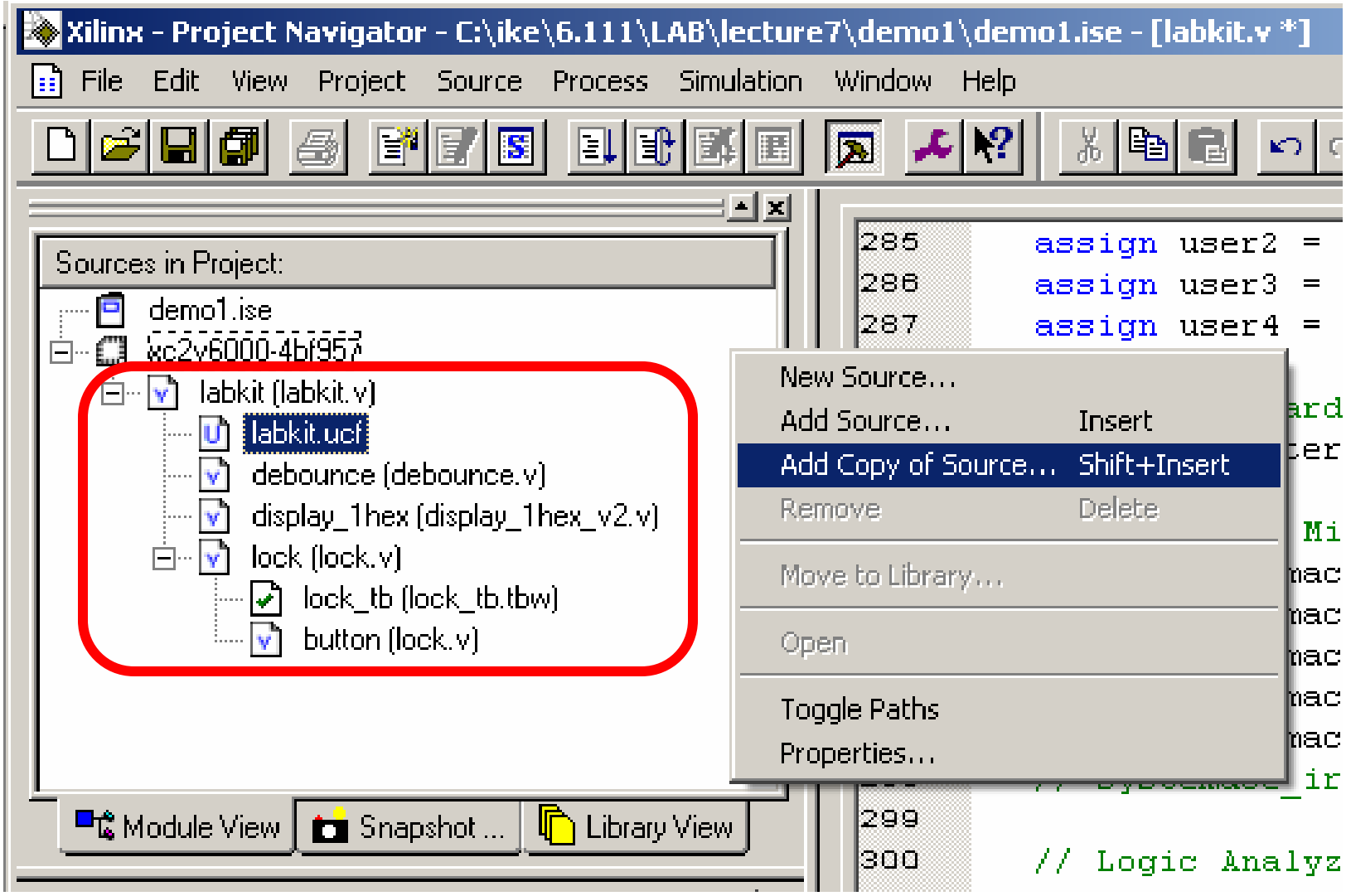
Nathan Ickes



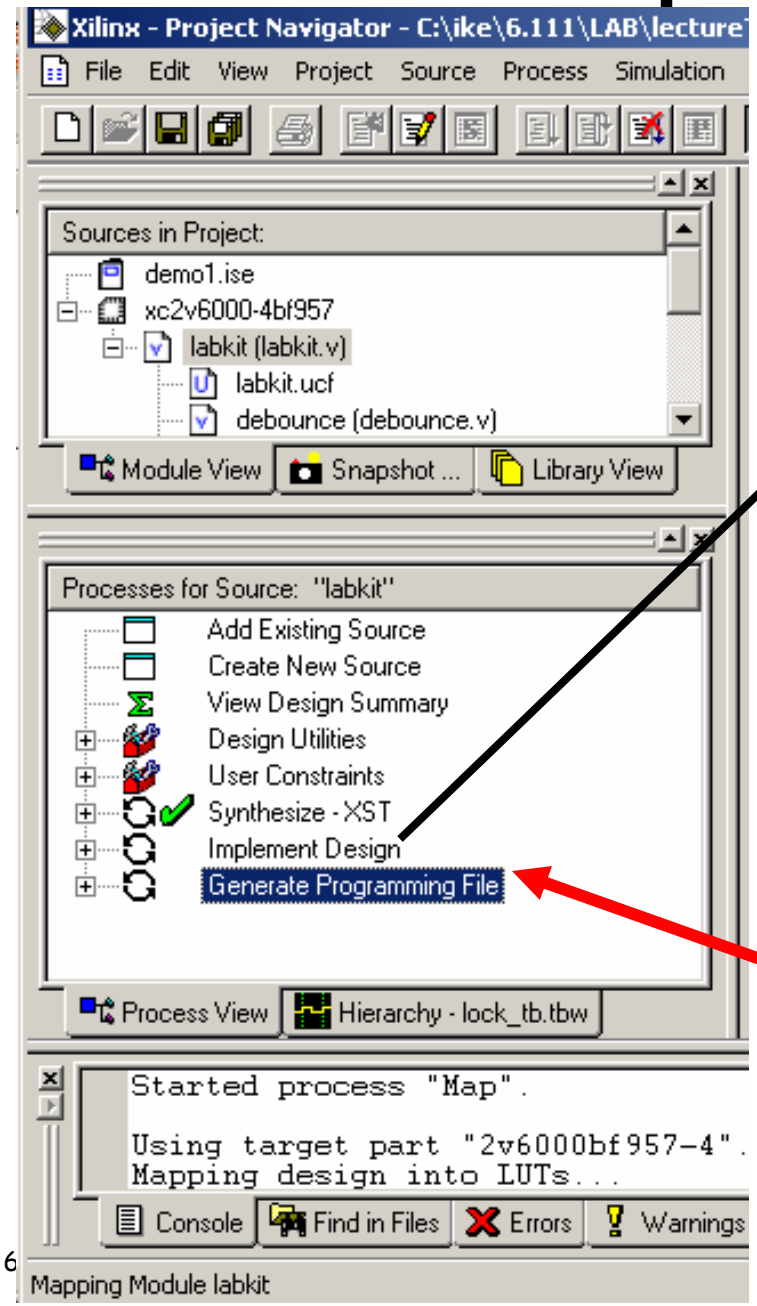
Step 4A: FPGA Device Assignment

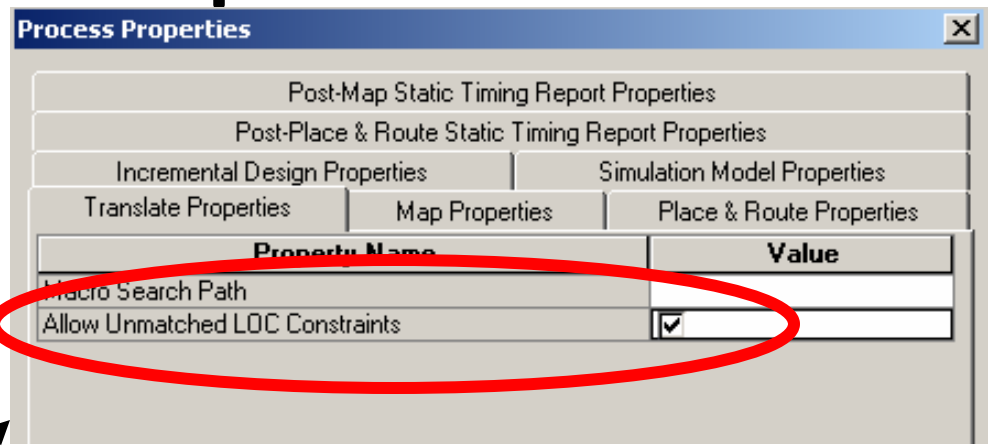


Step 4B: Add labkit files



Step 4C: Implement



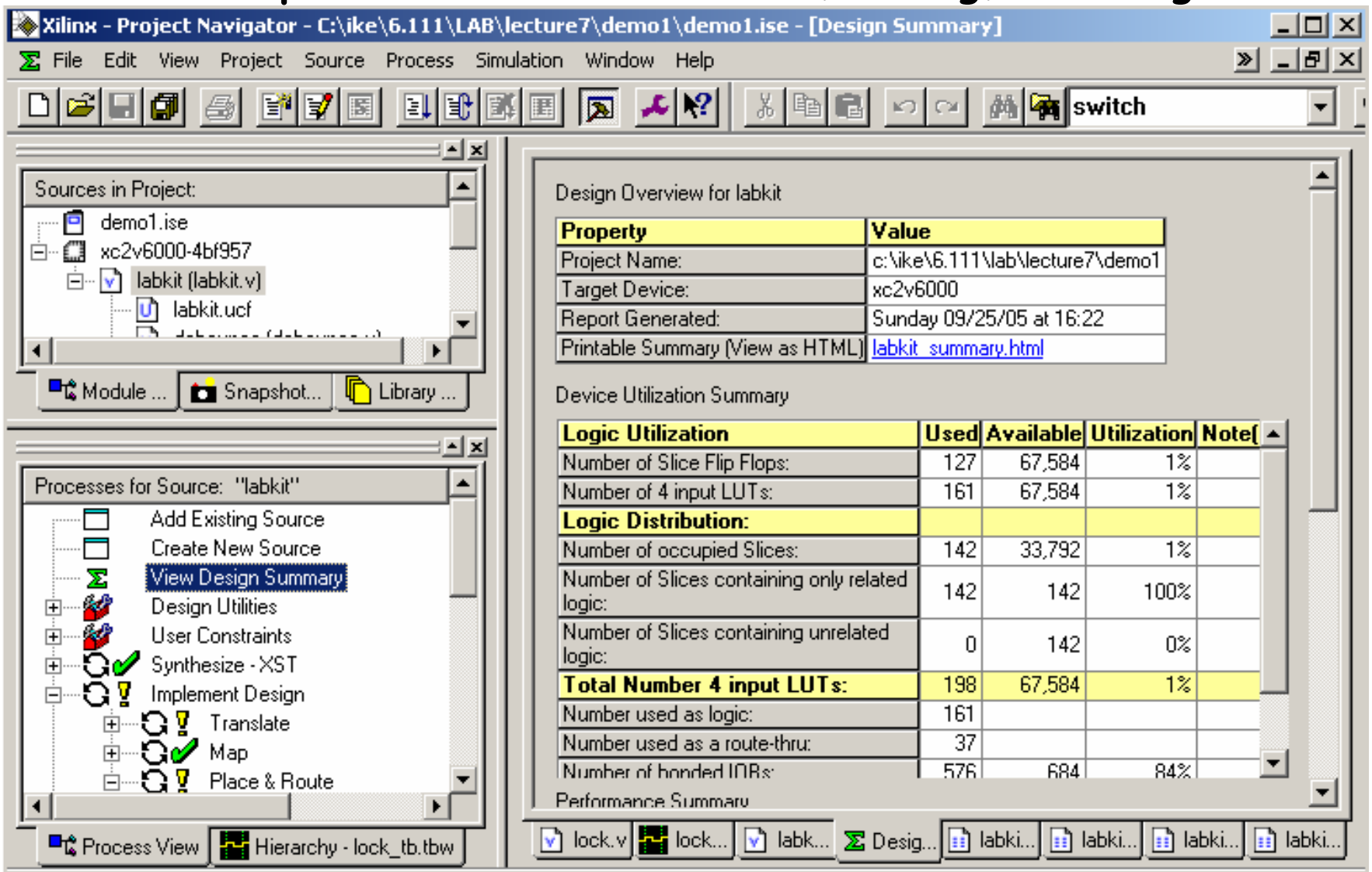


Right-click implement, select properties, Select "allow unmatched LOC constraints"

Double-click here to implement design, and create the labkit.bit file

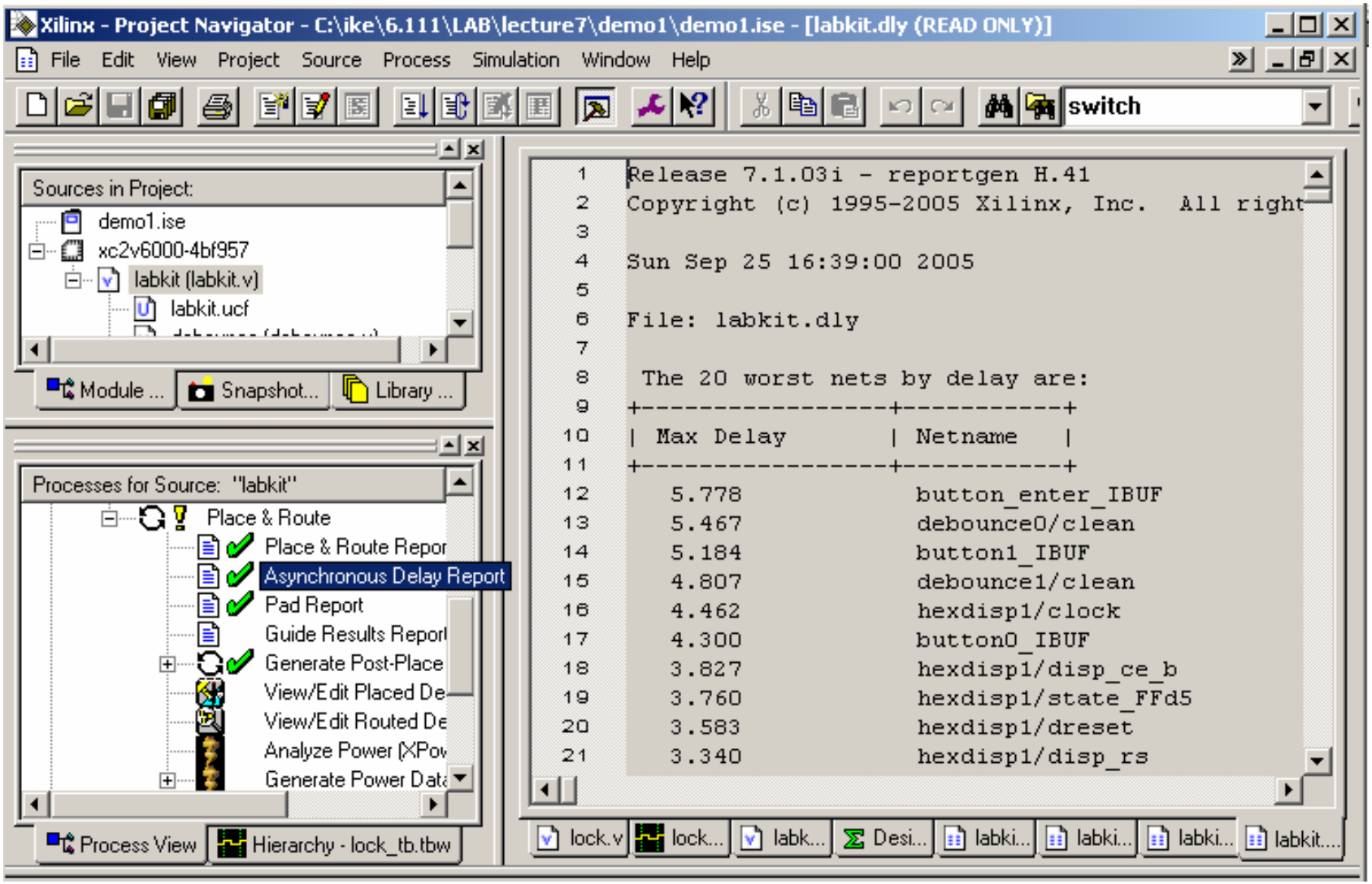
Step 4C: Implement

- Useful reports: Resource Utilization, Timing, RTL diagram



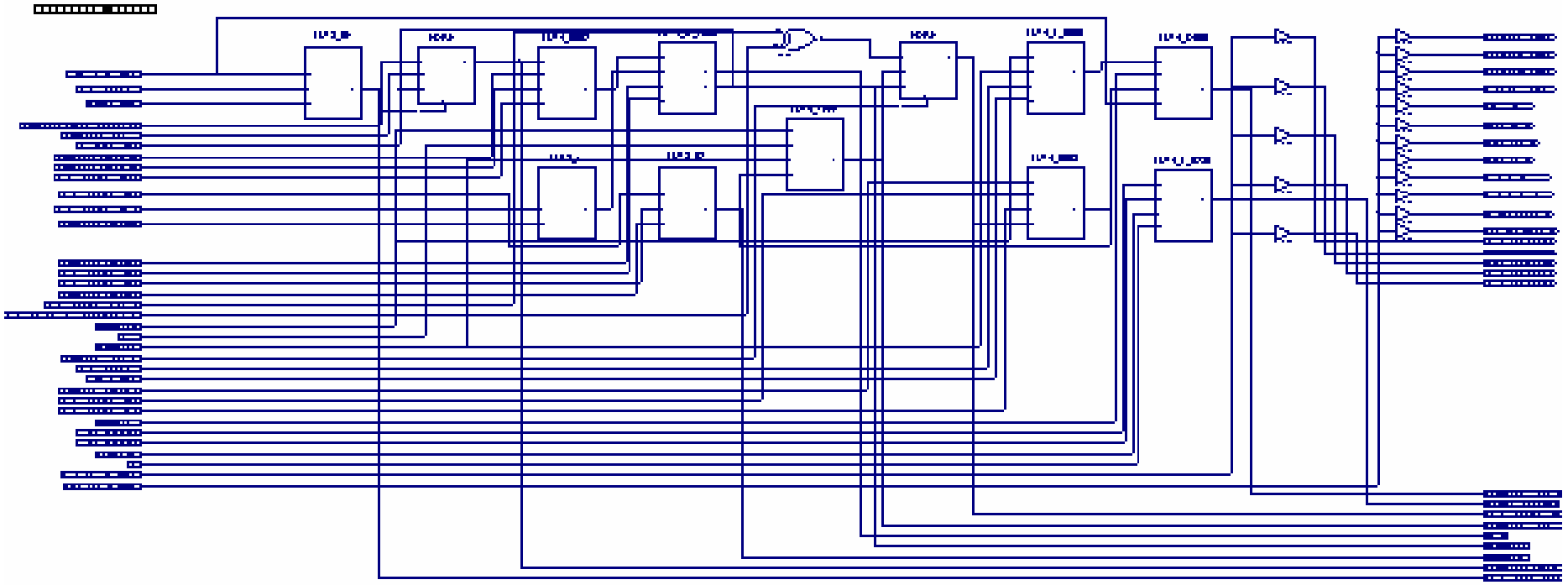
Step 4C: Implement

- Useful reports: Resource Utilization, Timing, RTL diagram



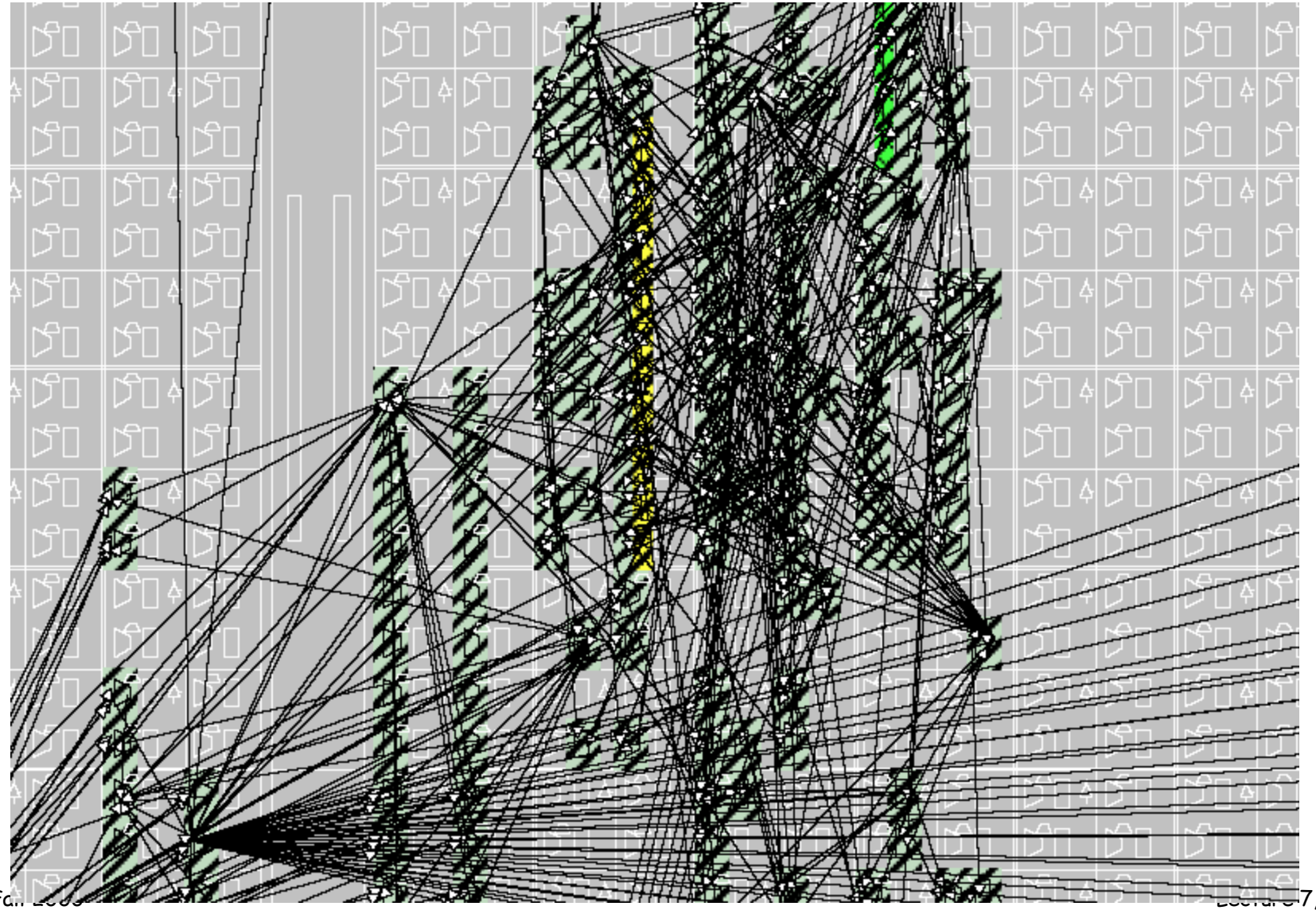
Step 4C: Implement

- Useful reports: Resource Utilization, Timing, RTL diagram



Step 4C: Implement

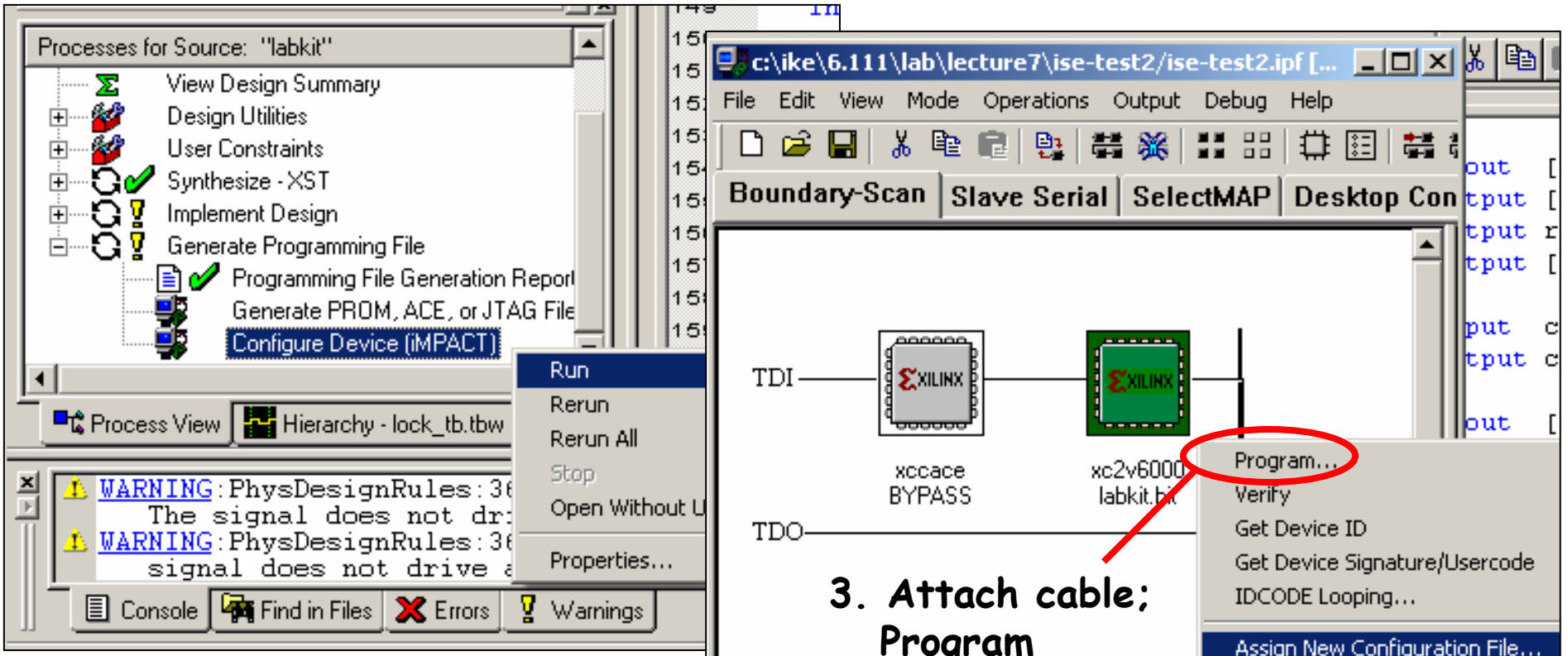
- **Useful reports: Floorplan**



Step 4D: Program FPGA

- We'll use the parallel port IV cable
- Transfer program: "IMPACT" – uses JTAG serial chain

joint test access group

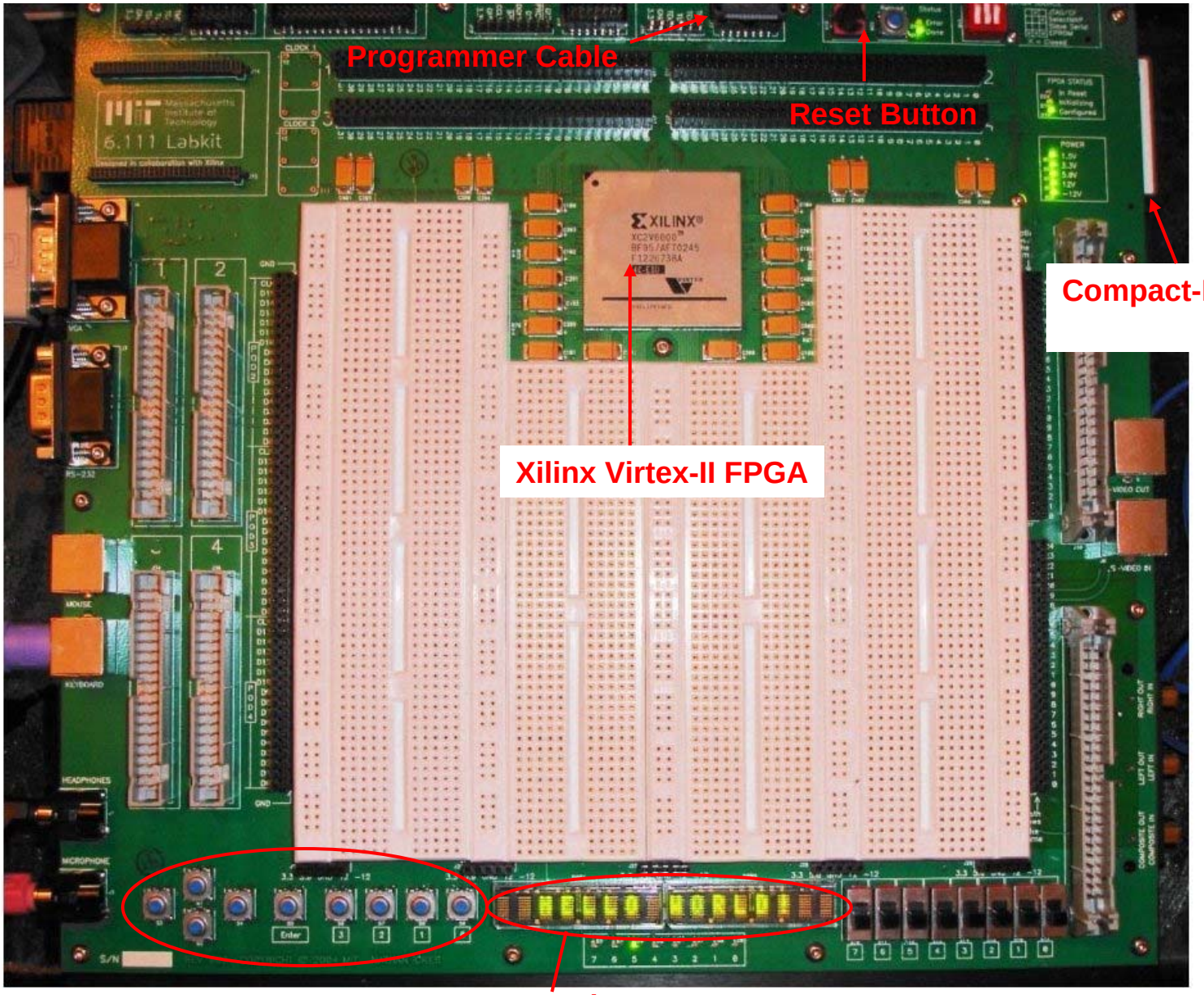


1. Select "Configure Device"

2. Bypass first device
   assign labkit.bit to second device

3. Attach cable;
   Program

Step 4D: Program FPGA



Summary

- Modern digital system design:
  - Hardware description language ➡ FPGA / ASIC
- Toolchain:
  - Design Entry ➡ Synthesis ➡ Implementation
  - Simulate
- New Labkit:
  - Black-box peripherals
  - Almost all functionality is programmed in!
  - How to generate video? Synchronize systems? Create/Digitize Audio? Serial & communications?