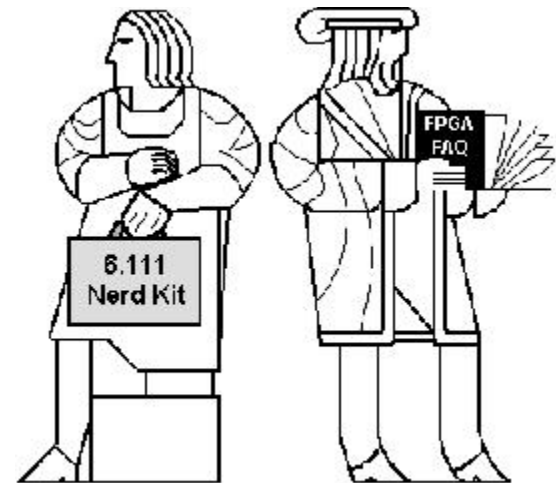


6.111 Lecture 7

Today:

Demo! (or die): An Electronic Lock

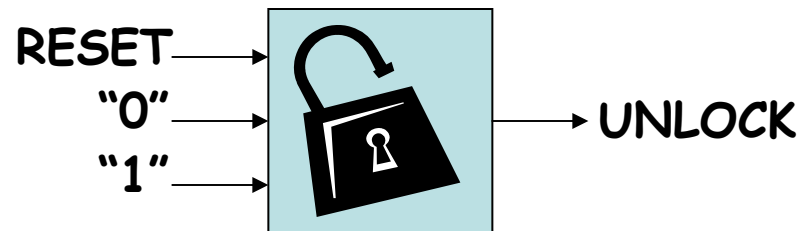
1. Design FSM
2. Implement in Verilog
3. Compile: Xilinx tool-chain
4. Program labkit



Demo!

GOAL:

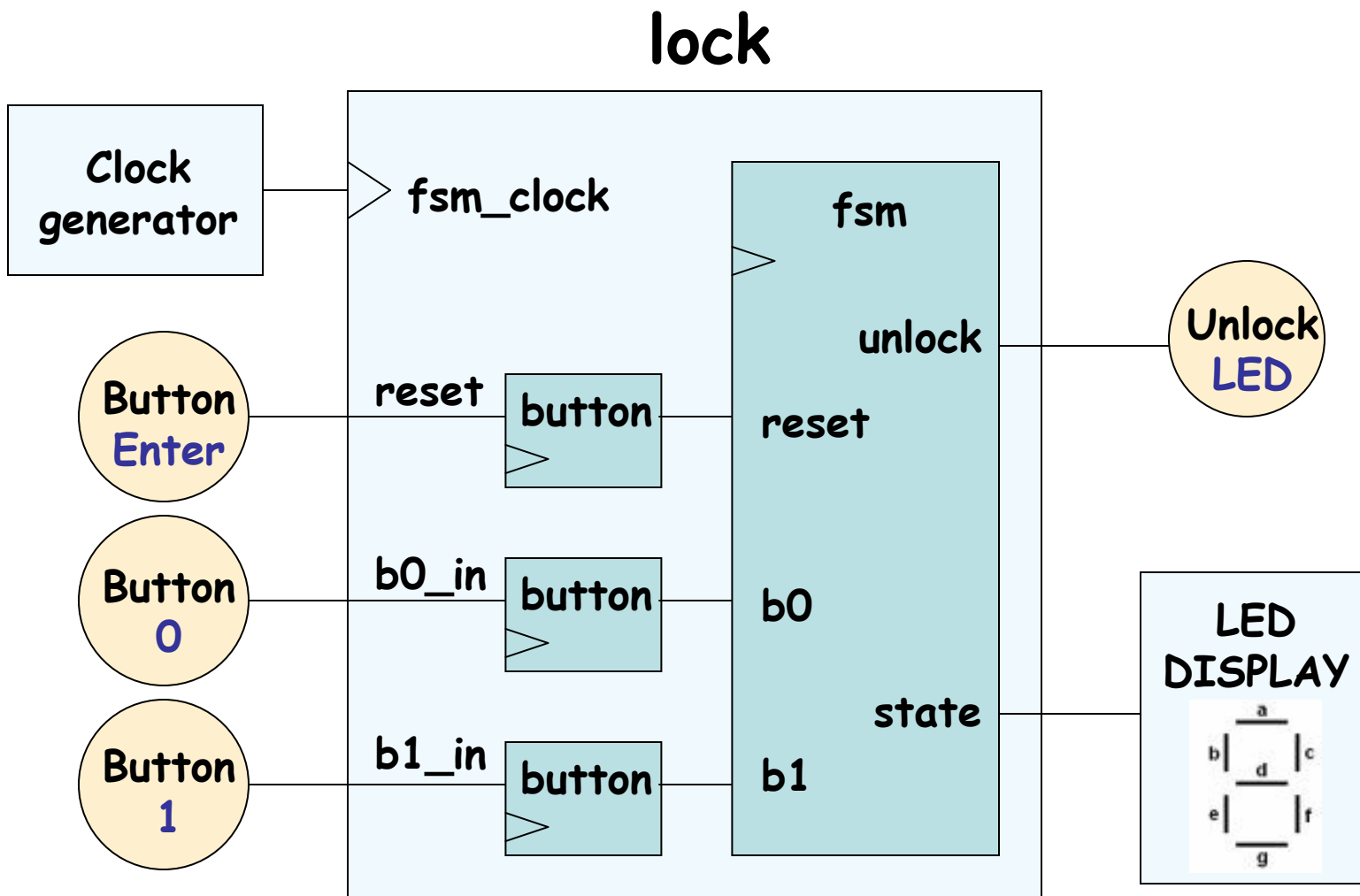
Build an electronic combination lock with a reset button, two number buttons (0 and 1), and an unlock output. The combination should be **01011**.



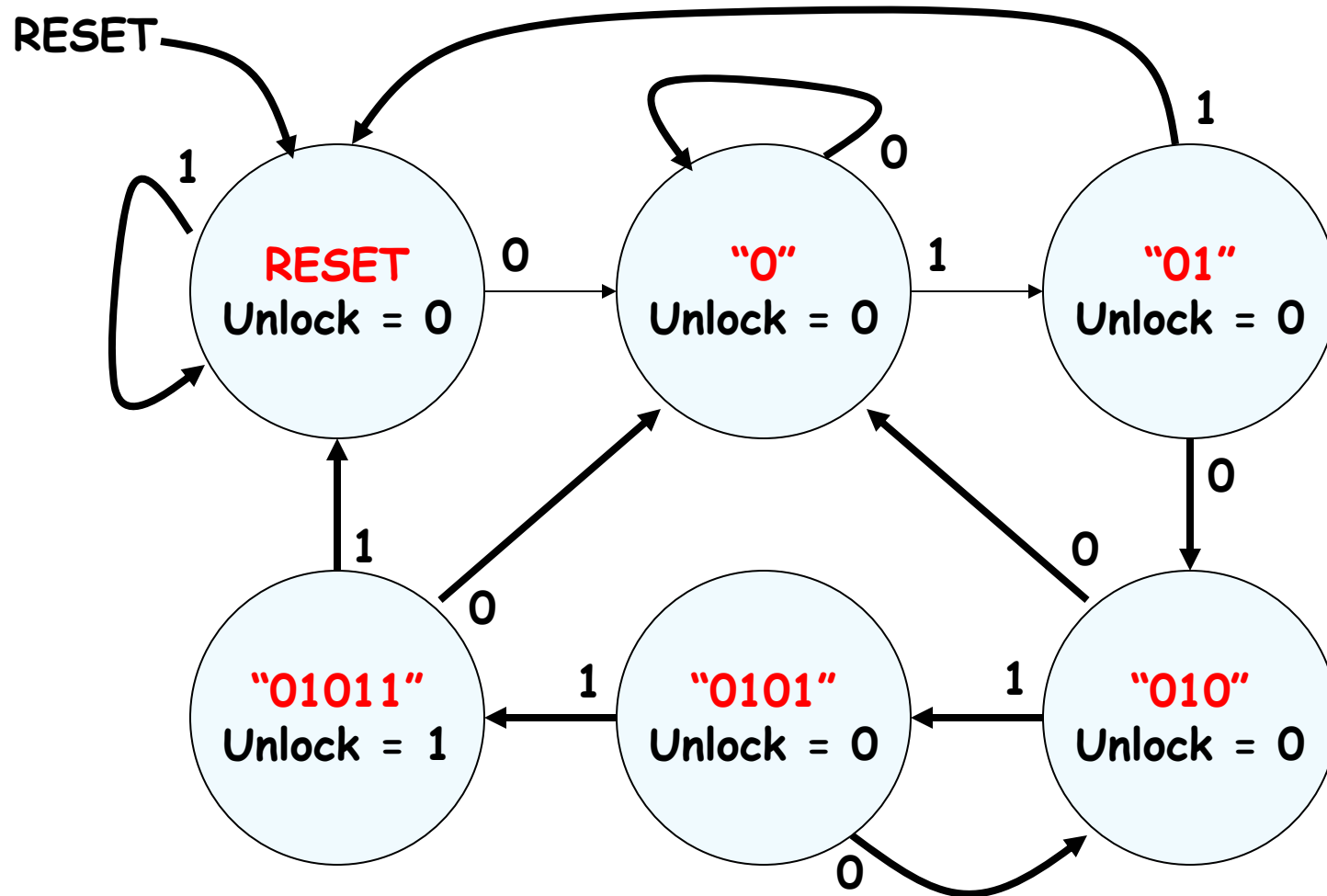
STEPS:

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 = ???;
```

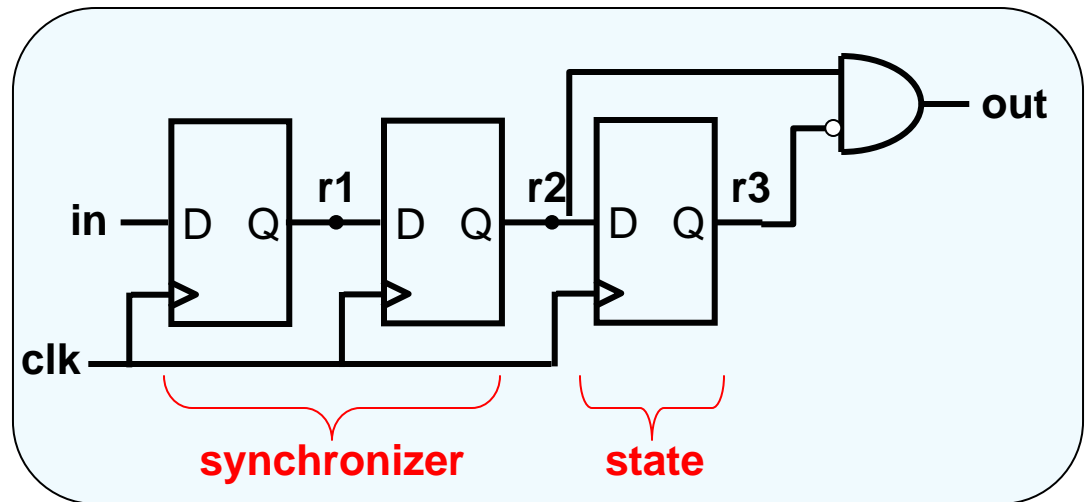
```
    // debugging?
```

```
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
```

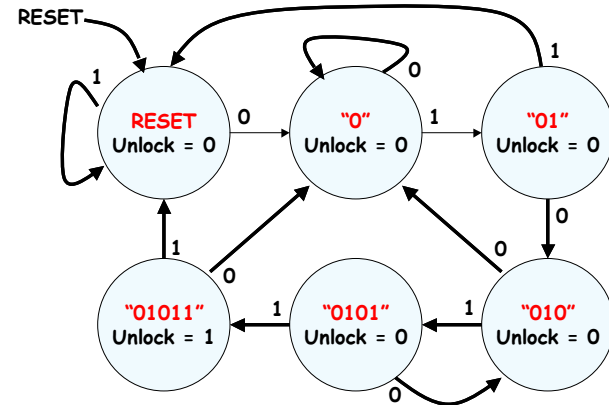


Step 2B: state transition diagram

```

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);

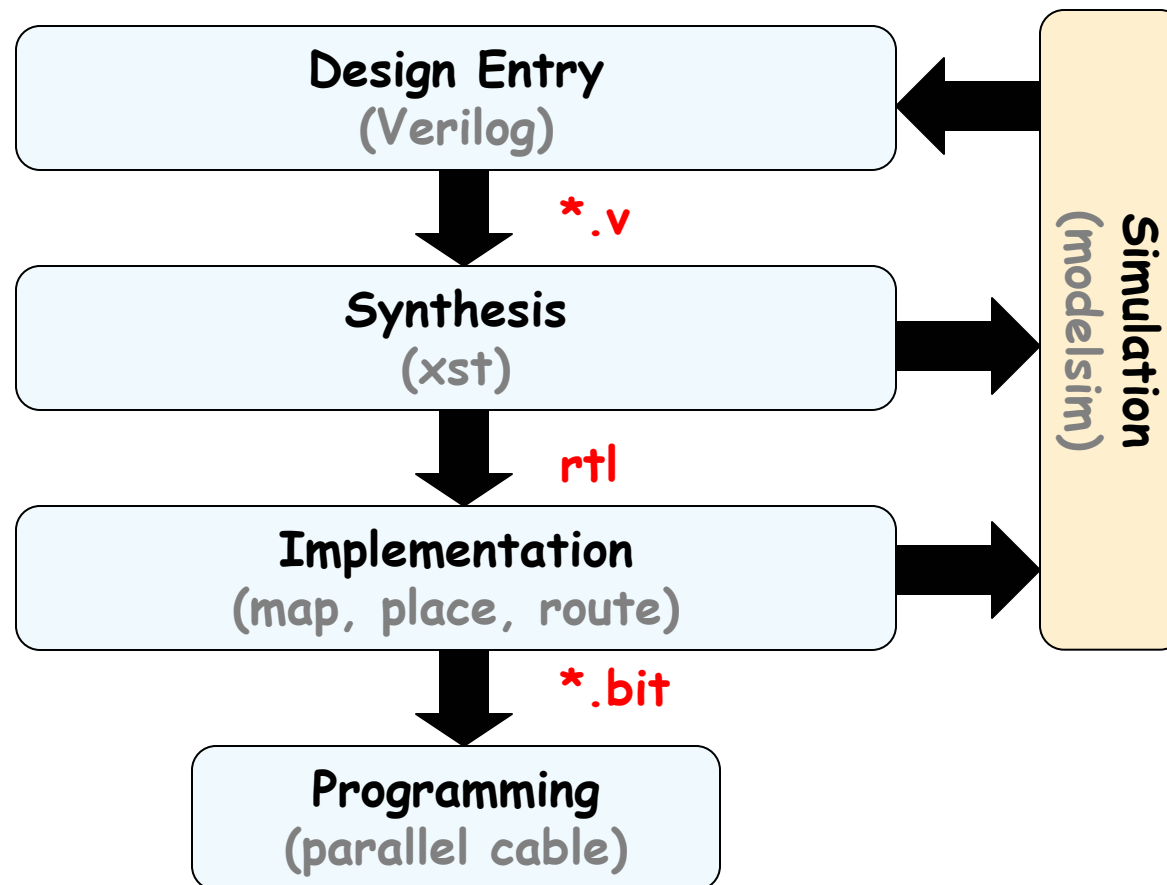
    parameter S_RESET = 0; parameter S_0 = 1; // state assignments
    parameter S_01 = 2; parameter S_010 = 3;
    parameter S_0101 = 4; parameter S_01011 = 5;

    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

The screenshot shows the Xilinx Project Navigator interface. The main window displays the source code for a Verilog module named 'lock.v'. The code defines a module with inputs 'clk', 'reset_in', 'b0_in', and 'b1_in', and outputs 'out' and 'hex_display'. It includes comments for synchronizing push buttons and implementing a state transition diagram. The code defines parameters for state values and a register for the state, followed by a state machine implementation using an 'always' block and a 'begin' block.

```
1
2 module lock(clk, reset_in, b0_in, b1_in, out, hex_display);
3   input clk, reset_in, b0_in,b1_in;
4   output out; output [3:0] hex_display;
5
6   // synchronize push buttons, convert to pulses
7
8   wire reset, b0, b1;
9   button b_reset(clk,reset_in,reset);
10  button b_0(clk,b0_in,b0);
11  button b_1(clk,b1_in,b1);
12
13  // implement state transition diagram
14
15  parameter S_RESET = 0;
16  parameter S_0 = 1;
17  parameter S_01 = 2;
18  parameter S_010 =3;
19  parameter S_0101 = 4;
20  parameter S_01011 = 5;
21
22  reg [2:0] state;
23
24  always @ (posedge clk)
25  begin
26    if (reset) state <= S_RESET;
27    else case (state)
28      S_RESET: state <= b0 ? S_0 : ( b1 ? S_RESET : state
29      S_0:     state <= b0 ? S_0 : ( b1 ? S_01 : state );
30      S_01:   state <= b0 ? S_010 : ( b1 ? S_RESET : state
```

The interface includes a 'Sources in Project' pane on the left showing the project structure, a 'Processes for Source' pane at the bottom left, and a toolbar at the top with various icons and a 'switch' dropdown menu. A context menu is open over the 'lock (lock.v)' source file, showing options like 'Add Source...', 'Remove', and 'Open'.

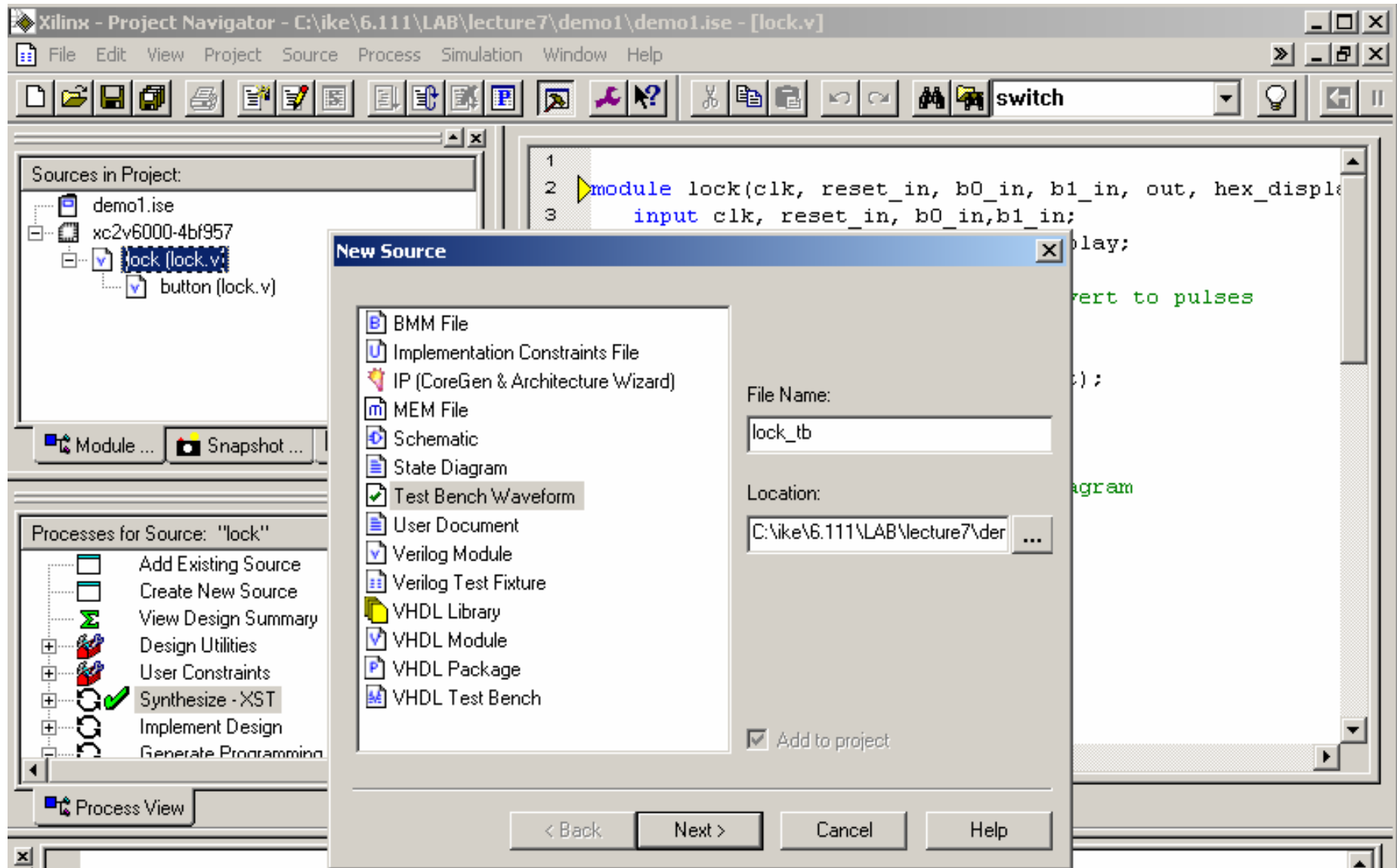
Step 3B: Compile/Synthesize

The screenshot displays the Xilinx Project Navigator interface during the synthesis phase. The top window shows the project structure with 'demo1.isc' and 'xc2v6000-4bf957' as the target device. The 'lock (lock.v)' module is selected, and its source files are listed in the 'Sources in Project' pane. The 'Processes for Source: "lock"' pane shows the 'Synthesize -XST' step as the current active process. The main editor displays the Verilog code for the 'lock' module, which includes input/output declarations, synchronization comments, button components, state transition logic, and parameter definitions. The bottom console window provides timing analysis results.

```
1
2 module lock(clk, reset_in, b0_in, b1_in, out, hex_display)
3     input clk, reset_in, b0_in, b1_in;
4     output out; output [3:0] hex_display;
5
6     // synchronize push buttons, convert to pulses
7
8     wire reset, b0, b1;
9     button b_reset(clk, reset_in, reset);
10    button b_0(clk, b0_in, b0);
11    button b_1(clk, b1_in, b1);
12
13    // implement state transition diagram
14
15    parameter S_RESET = 0;
16    parameter S_0 = 1;
17    parameter S_01 = 2;
18    parameter S_010 = 3;
19    parameter S_0101 = 4;
20    parameter S_01011 = 5;
21
22    reg [2:0] state;
```

Minimum period: 3.479ns (Maximum Frequency: 287.439MHz)
Minimum input arrival time before clock: 1.712ns
Maximum output required time after clock: 6.987ns
Maximum combinational path delay: No path found

Step 3B: Create testbench



Step 3B: Create testbench

Initialize Timing

Maximum output delay

Minimum input setup

Clock high for

Clock low for

Clock Timing Information
Inputs are assigned at "Input Setup Time" and outputs are checked at "Output Valid Delay".

Rising Edge Falling Edge
 Dual Edge (DDR or DET)

Clock Time High: 100 ns
Clock Time Low: 100 ns
Input Setup Time: 15 ns
Output Valid Delay: 15 ns
Initial Offset: 0 ns

Clock Information

Single Clock: clk
 Multiple Clocks
 Combinatorial (or internal clock)

Combinatorial Timing Information
Inputs are assigned, outputs are decoded then checked. A delay between inputs and outputs avoids assignment/checking conflicts.

Check Outputs: 50 ns After Inputs are Assigned
Assign Inputs: 50 ns After Outputs are Checked

Global Signals

PRLD (CPLD) GSR (FPGA)
High for Initial: 100 ns

Initial Length of Test Bench: 2000 ns
Time Scale: ns
 Add Asynchronous Signal Support

OK Cancel Next > Help

Step 3B: Generate Simulation Results

The screenshot displays the Xilinx Project Navigator interface. The top window shows the project structure with sources like `demo1.isc`, `xc2v6000-4bf957`, and `lock (lock.v)`. The testbench `lock_tb (lock_tb.tbw)` is selected. The main window shows a timing diagram for the simulation. The end time is 4000 ns. The diagram shows signals `clk`, `b0_in`, `b1_in`, `reset_in`, `out`, and `hex_display[...]`. The `hex_display` signal shows a sequence of hexadecimal values: 0, 4, 0, 1, 2, 3, 4, 5. The console window at the bottom shows the simulation completion message: "Finished circuit initialization process. Success! Annotation Simulation Complete. Stopped at time : 4.200 us : File 'C:/ike/6.111/LAB/lecture7/demo1/lock_tb.ant' Line 74".

End Time: 4000 ns

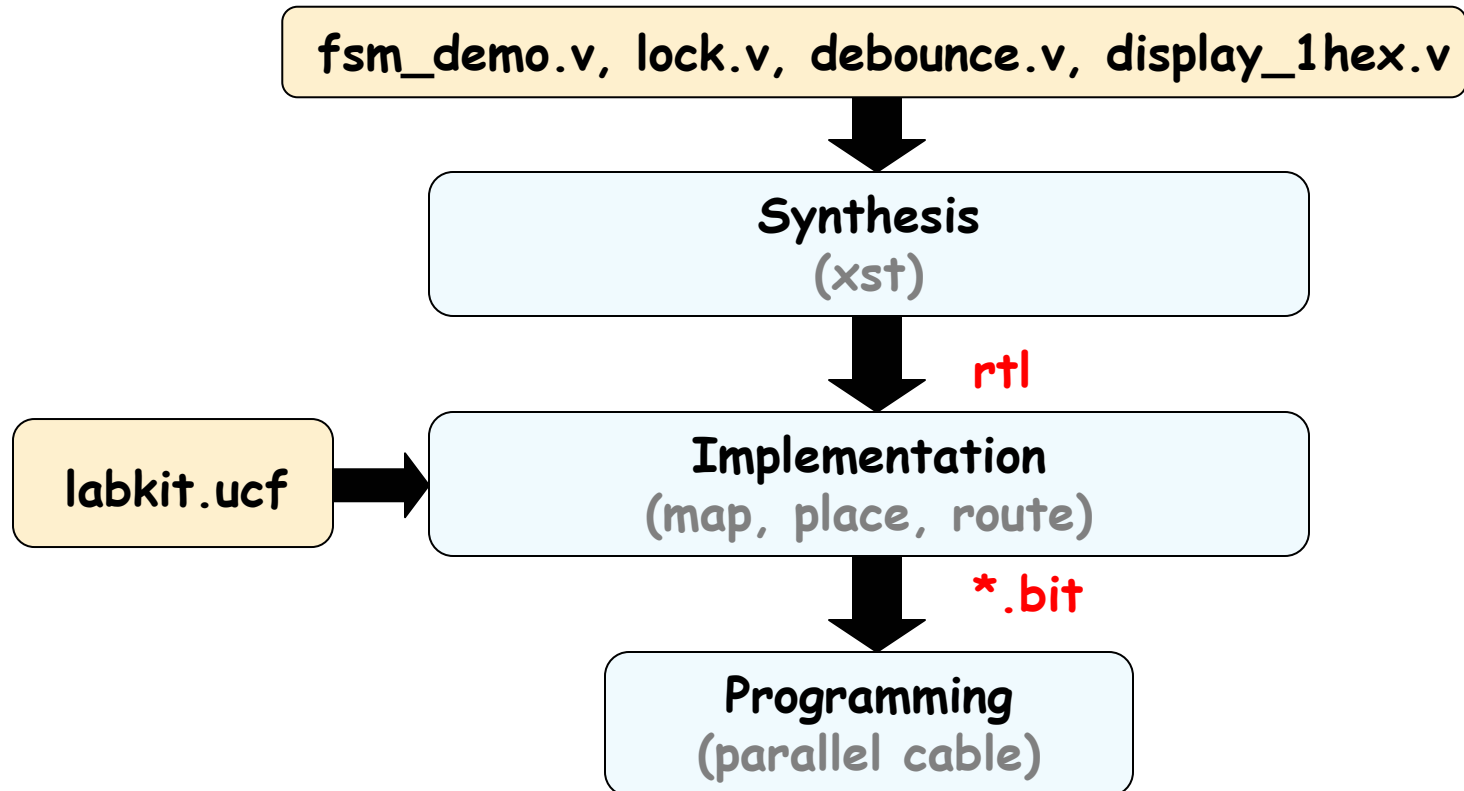
Signal	Value	Hex Display
clk	0	
b0_in	0	
b1_in	0	
reset_in	0	
out	1	
hex_display[...]	5	0 4 0 1 2 3 4 5

```
Finished circuit initialization process.
Success! Annotation Simulation Complete.
Stopped at time : 4.200 us : File "C:/ike/6.111/LAB/lecture7/demo1/lock_tb.ant" Line 74

Compiling verilog file "c:/ike/6.111/lab/lecture7/demo1/lock.v"
```

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 screenshot shows the Xilinx PACE software interface for pin assignments. The main window displays the 'Package Pins for xc2v6000-4-bf957 (Flight Time)' in a 'Top View' grid. The grid is labeled with columns 1-31 and rows A-AL. Various pins are assigned to specific locations on the package, indicated by colored squares and circles.

On the left side, the 'Design Object List - I/O Pins' table lists the following assignments:

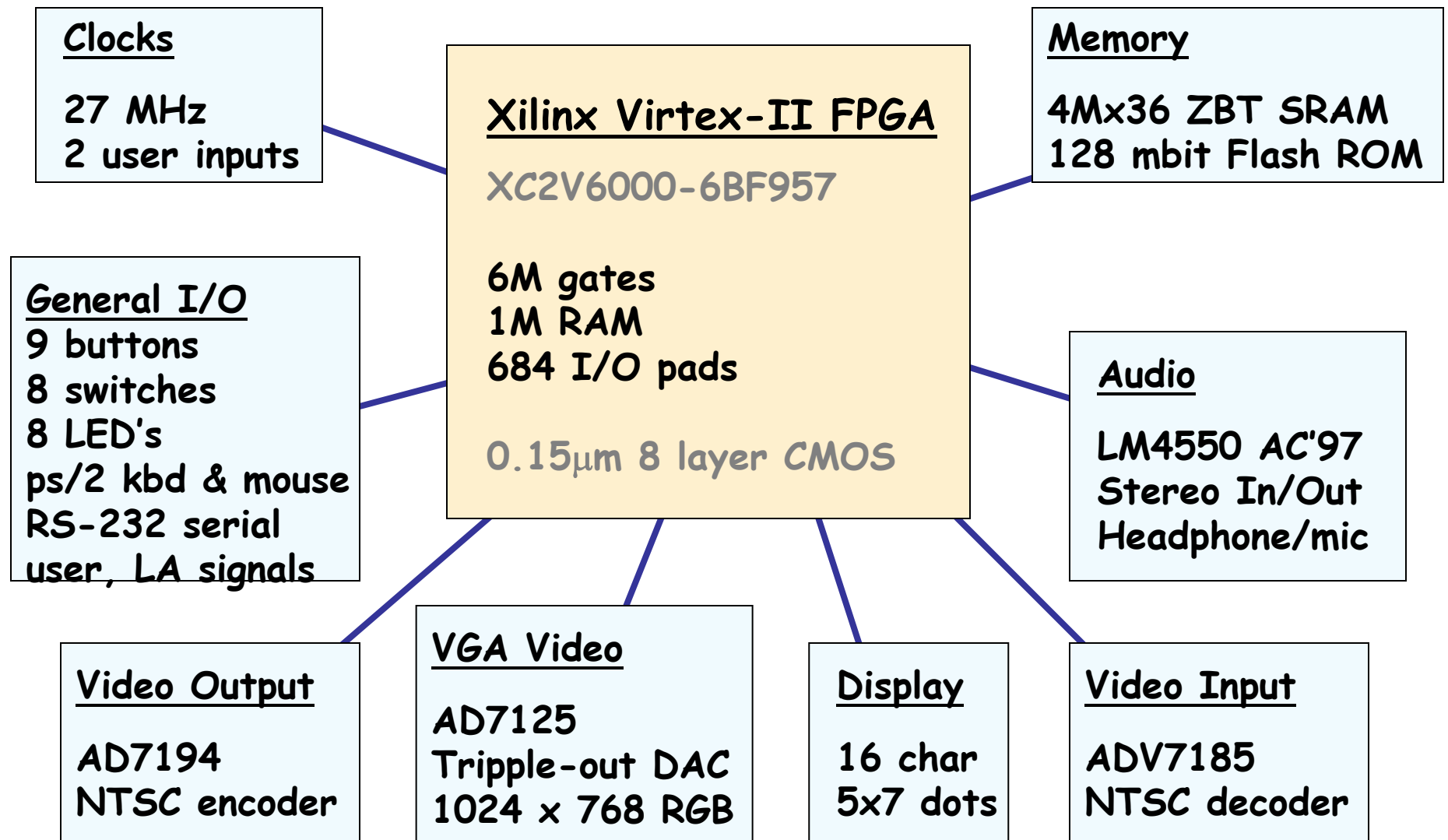
I/O Name	I/O Direction	Loc	Bank
ac97_bit_clock	Input	ah24	BANK5
ac97_sdata_in	Input	aj24	BANK5
ac97_sdata_out	Output	ac18	BANK5 LVD
ac97_synch	Output	ac17	BANK5 LVD
analyzer1_clock	Output	G2	BANK2 LVT
analyzer1_data<0>	Output	R2	BANK2 LVT
analyzer1_data<1>	Output	R1	BANK2 LVT

Below this table, another table shows additional I/O standards:

#	Group	Loc	I/O Std.
8	vga_out_red		LVTTL
8	vga_out_green		LVTTL
8	vga_out_blue		LVTTL
10	tv_out_ycrCb		LVDCI_33
20	tv_in_ycrCb		
36	ram0_data		LVDCI_33
19	ram0_address		LVDCI_33

The 6.111 Labkit: Subsystems

Nathan Ickes



Step 4A: FPGA Device Assignment

The screenshot shows the Xilinx Project Navigator interface. The main window displays a Verilog code snippet with three assignment statements:

```
285 assign user2 = 32'hZ;  
286 assign user3 = 32'hZ;  
287 assign user4 = 32'hZ;
```

The Project Properties dialog box is open, showing the following table:

Property Name	Value
Device Family	Virtex2
Device	xc2v6000
Package	bf957
Speed Grade	-4
Top-Level Module Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISE Simulator
Generated Simulation Language	Verilog

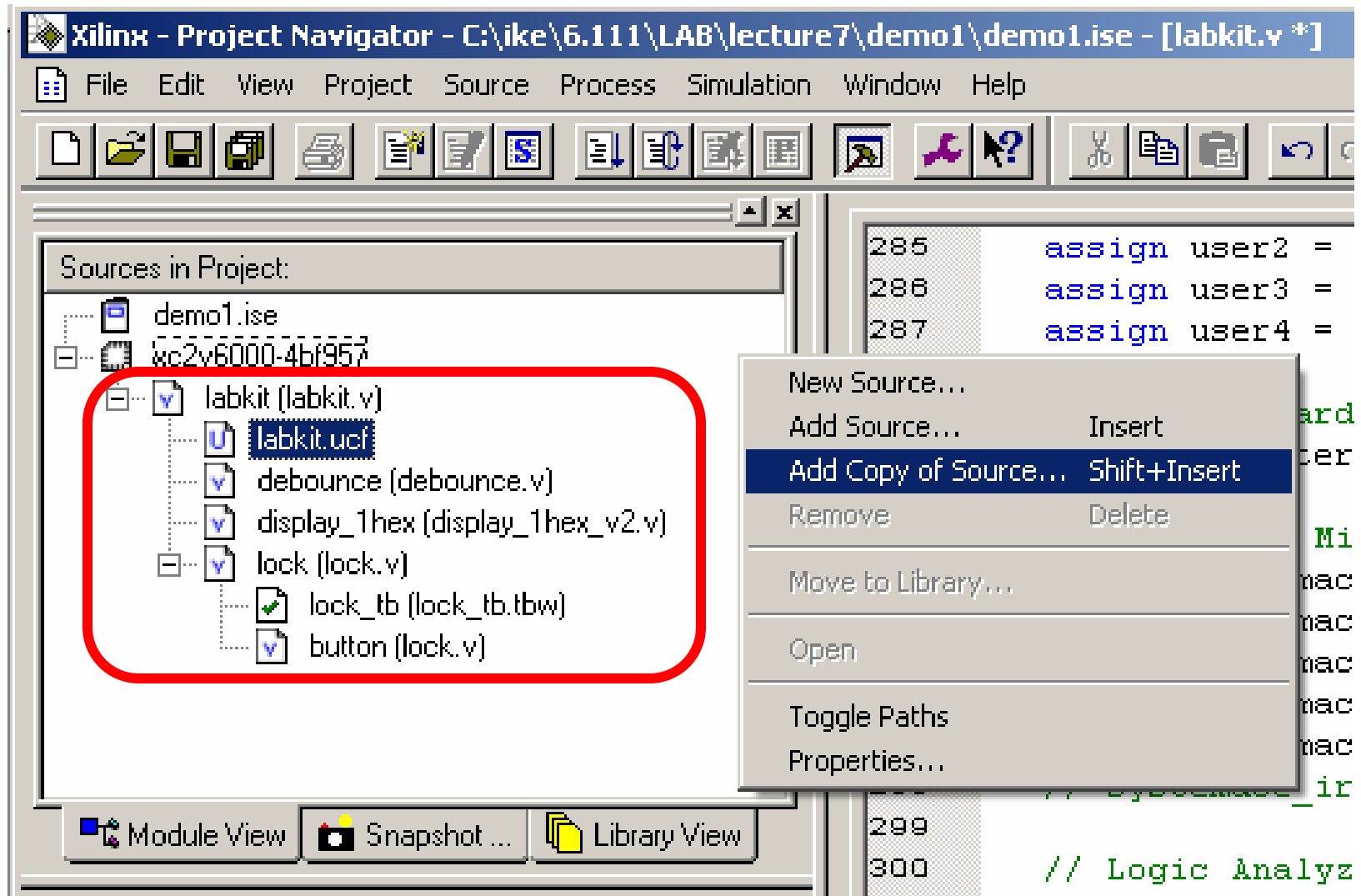
Annotations in the image include:

- A red arrow pointing from the text "Virtex 2: xc2v6000" at the bottom to the "Device" field in the Project Properties dialog.
- A red arrow pointing from the text "Package: bf957" at the bottom to the "Package" field in the Project Properties dialog.
- A red arrow pointing from the text "Speed: -4" to the "Speed Grade" field in the Project Properties dialog.
- Green text "dy are inputs" is visible in the background of the Project Properties dialog.

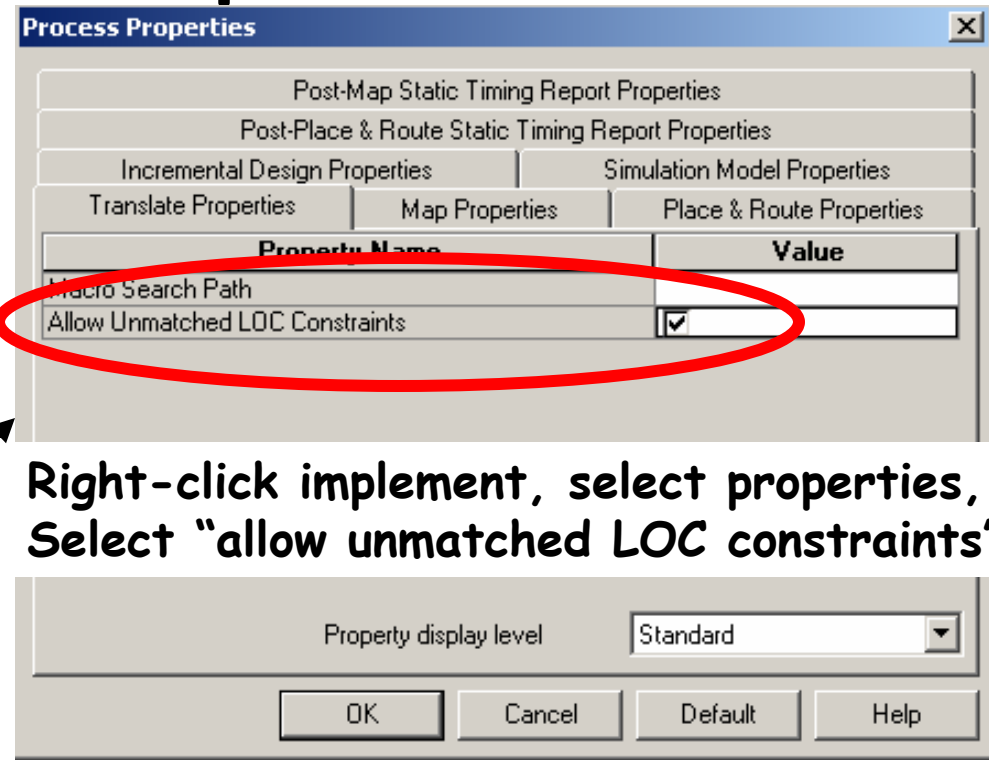
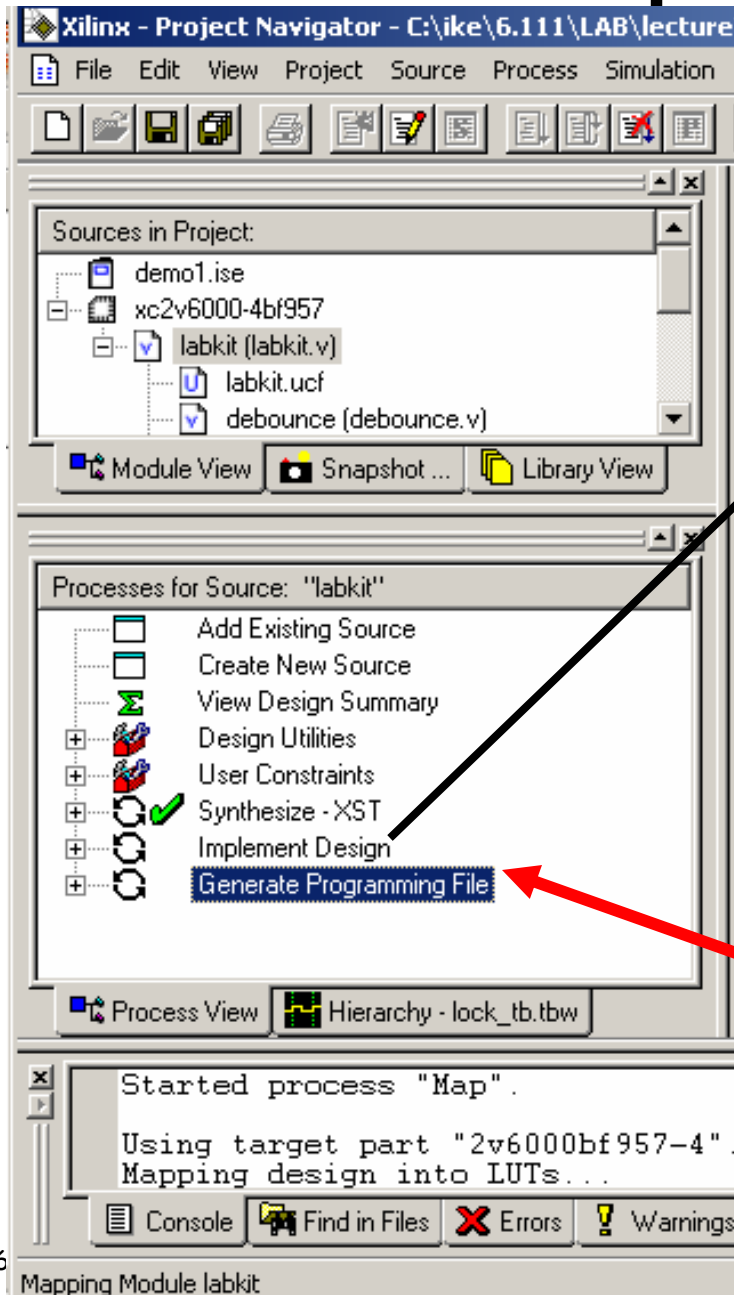
Virtex 2: xc2v6000

Package: bf957

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

The screenshot shows the Xilinx Project Navigator interface. The main window displays the 'Design Summary' for the 'labkit' design. The interface includes a menu bar, a toolbar, and several panels. The 'Sources in Project' panel shows the project structure, including 'demo1.isc', 'xc2v6000-4bf957', and 'labkit (labkit.v)'. The 'Processes for Source: "labkit"' panel shows the implementation steps, with 'View Design Summary' highlighted. The 'Design Overview for labkit' panel contains a table with project details and a 'Device Utilization Summary' table.

Design Overview for labkit

Property	Value
Project Name:	c:\ike\6.111\lab\lecture7\demo1
Target Device:	xc2v6000
Report Generated:	Sunday 09/25/05 at 16:22
Printable Summary (View as HTML)	labkit_summary.html

Device Utilization Summary

Logic Utilization	Used	Available	Utilization	Note
Number of Slice Flip Flops:	127	67,584	1%	
Number of 4 input LUTs:	161	67,584	1%	
Logic Distribution:				
Number of occupied Slices:	142	33,792	1%	
Number of Slices containing only related logic:	142	142	100%	
Number of Slices containing unrelated logic:	0	142	0%	
Total Number 4 input LUTs:	198	67,584	1%	
Number used as logic:	161			
Number used as a route-thru:	37			
Number of bonded I/Os:	576	684	84%	

Performance Summary

Step 4C: Implement

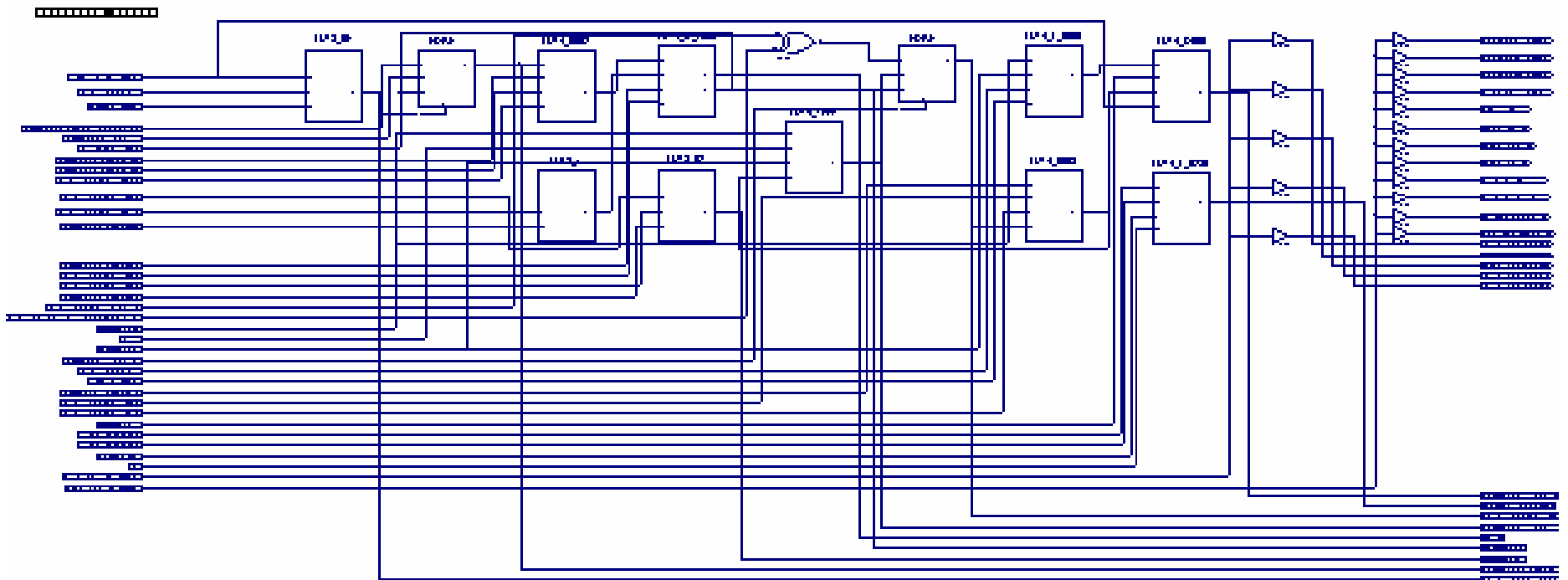
- Useful reports: Resource Utilization, Timing, RTL diagram

The screenshot shows the Xilinx Project Navigator interface. The top window displays the project structure with sources like demo1.ise, xc2v6000-4bf957, labkit (labkit.v), and labkit.ucf. The bottom-left pane shows the 'Processes for Source: "labkit"' list, with 'Asynchronous Delay Report' highlighted. The main window displays the output of the report, showing the 20 worst nets by delay.

```
1 Release 7.1.03i - reportgen H.41
2 Copyright (c) 1995-2005 Xilinx, Inc. All rights reserved.
3
4 Sun Sep 25 16:39:00 2005
5
6 File: labkit.dly
7
8 The 20 worst nets by delay are:
9 +-----+
10 | Max Delay | Netname |
11 +-----+
12 5.778      button_enter_IBUF
13 5.467      debounce0/clean
14 5.184      button1_IBUF
15 4.807      debounce1/clean
16 4.462      hexdisp1/clock
17 4.300      button0_IBUF
18 3.827      hexdisp1/disp_ce_b
19 3.760      hexdisp1/state_FFd5
20 3.583      hexdisp1/dreset
21 3.340      hexdisp1/disp_rs
```

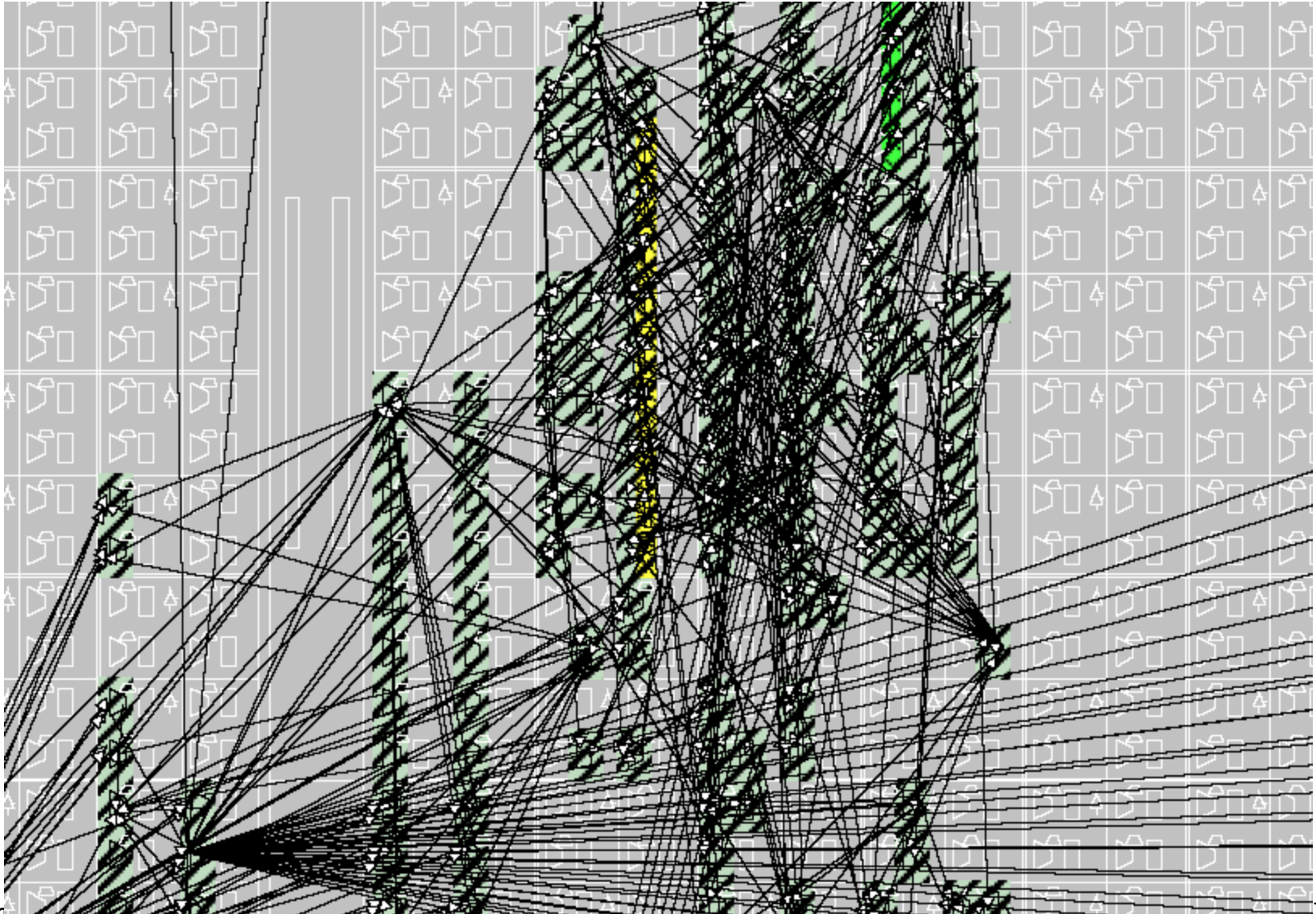
Step 4C: Implement

- 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

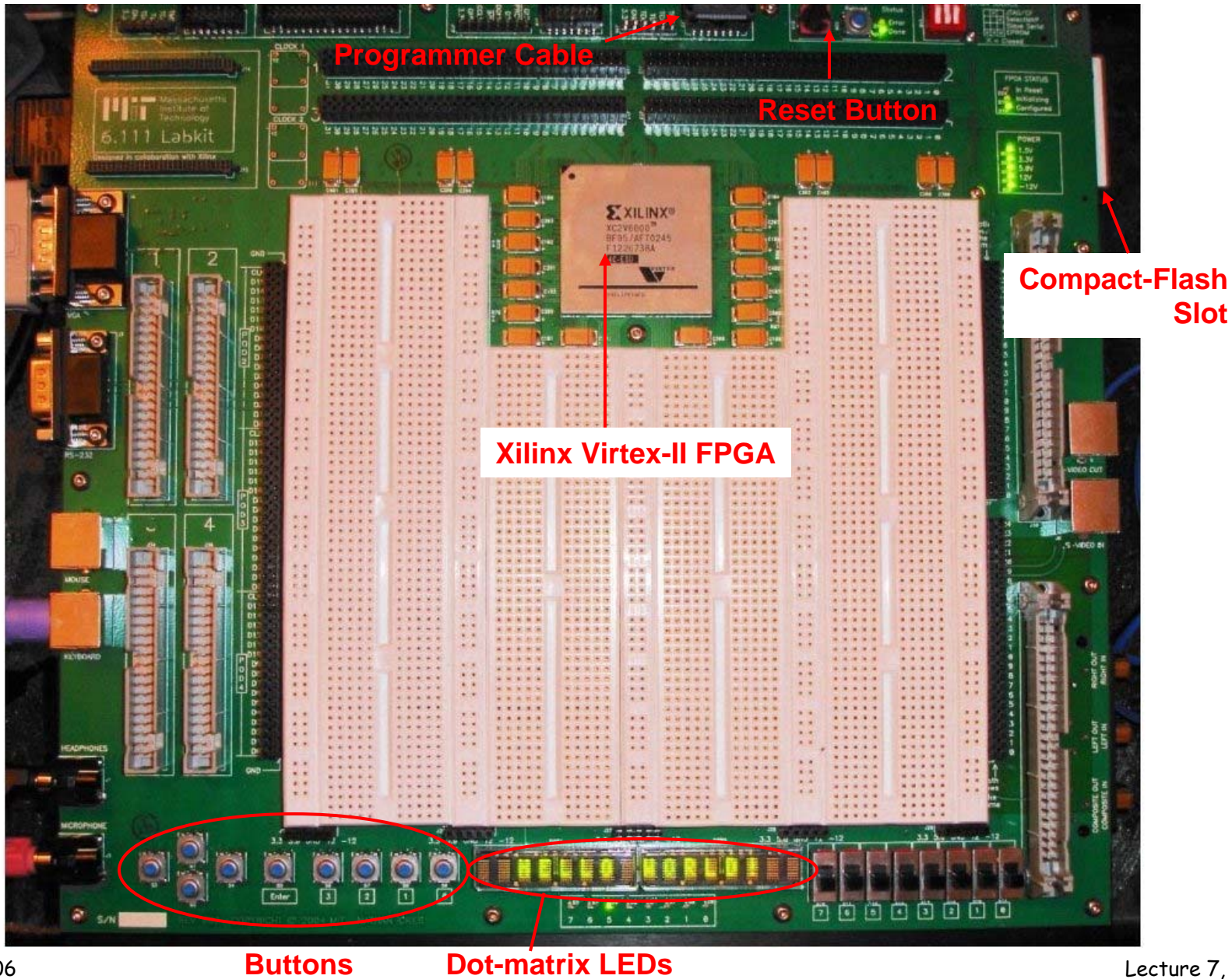
The screenshot shows the Xilinx ISE software interface. On the left, the 'Processes for Source: "labkit"' tree is expanded to 'Configure Device (iMPACT)'. A context menu is open over this option, with 'Run' selected. Below the tree, the console shows two warnings: 'WARNING: PhysDesignRules: 36 The signal does not drive' and 'WARNING: PhysDesignRules: 36 signal does not drive'. On the right, a JTAG chain diagram shows two Xilinx devices connected in series. The first device is labeled 'xccace BYPASS' and the second is 'xc2v6000 labkit.bit'. A red circle highlights the 'Program...' option in the context menu, with a red arrow pointing to it. The diagram also shows 'TDI' and 'TDO' connections.

1. Select "Configure Device"

3. Attach cable;
Program

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

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?

