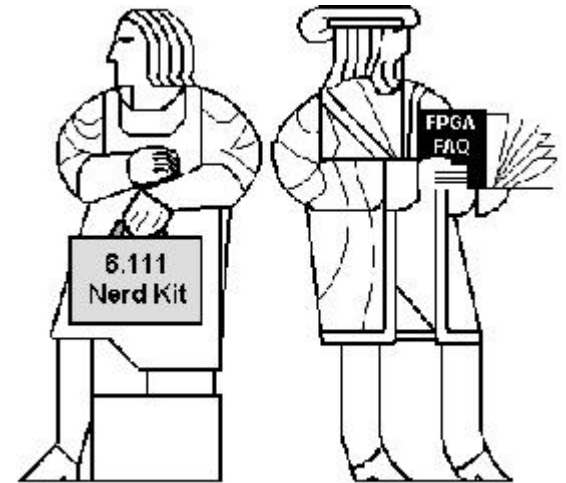


6.111 Lecture 7

Today:

Demo! (or die): An Electronic Lock

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



Demo!

GOAL:

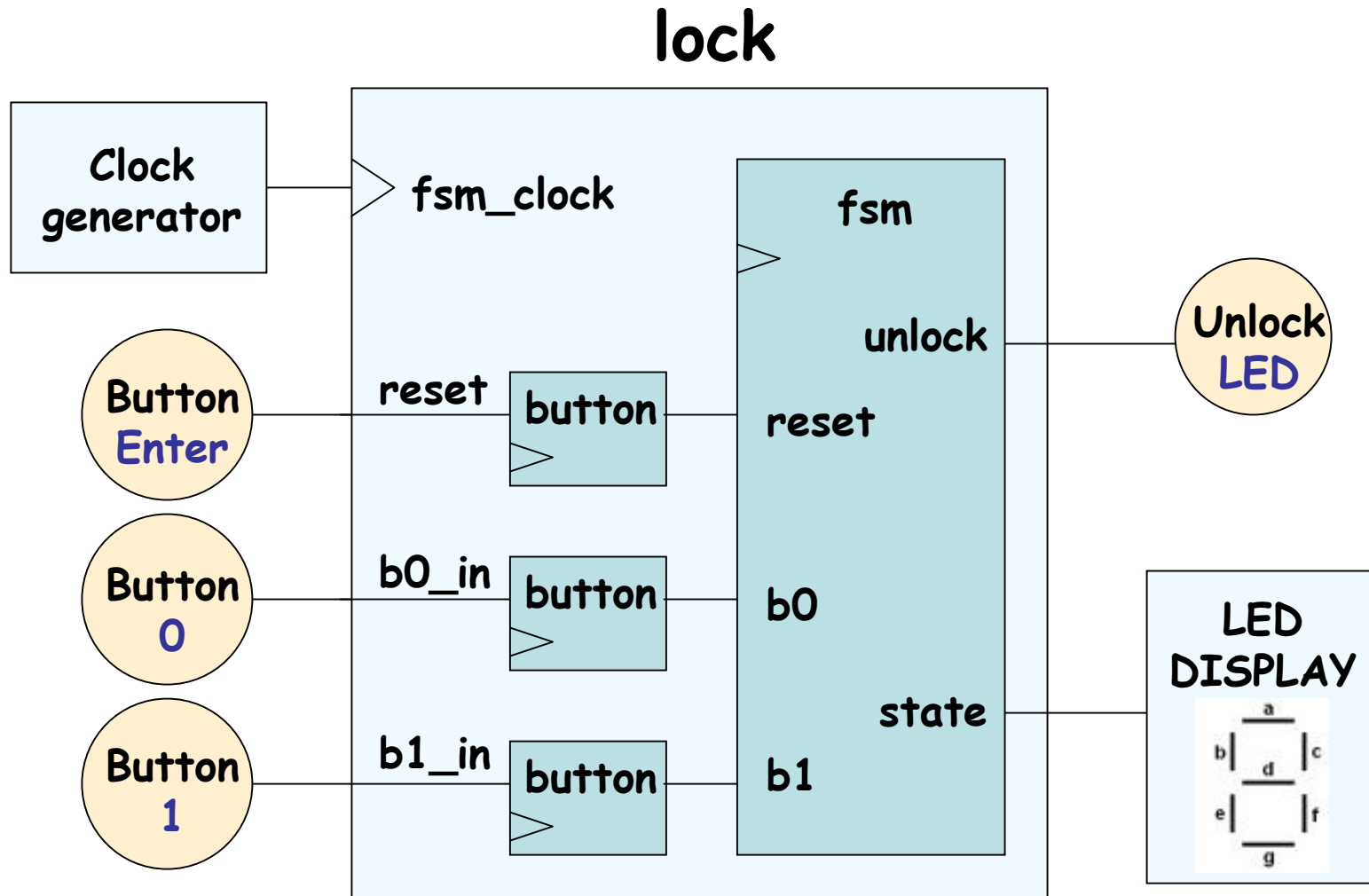
Build an electronic combination lock with a reset button, two number buttons (0 and 1), and an unlock output. The combination should be **01011**.



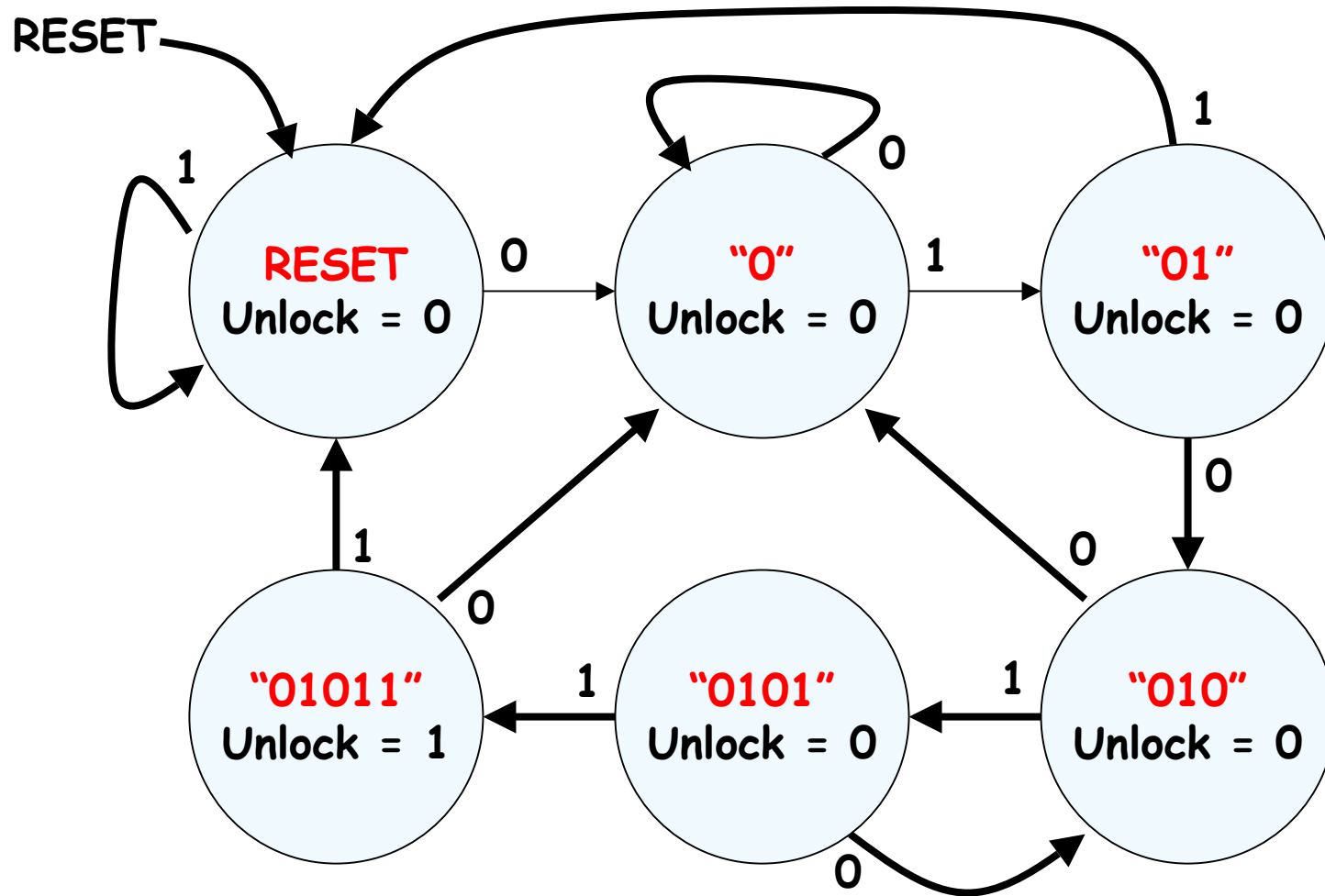
STEPS:

1. Design lock FSM (block diagram, state transitions)
2. Write Verilog module(s) for FSM
3. Use Xilinx ISE6.3 (synthesis, simulation)
4. Program FPGA, 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
```

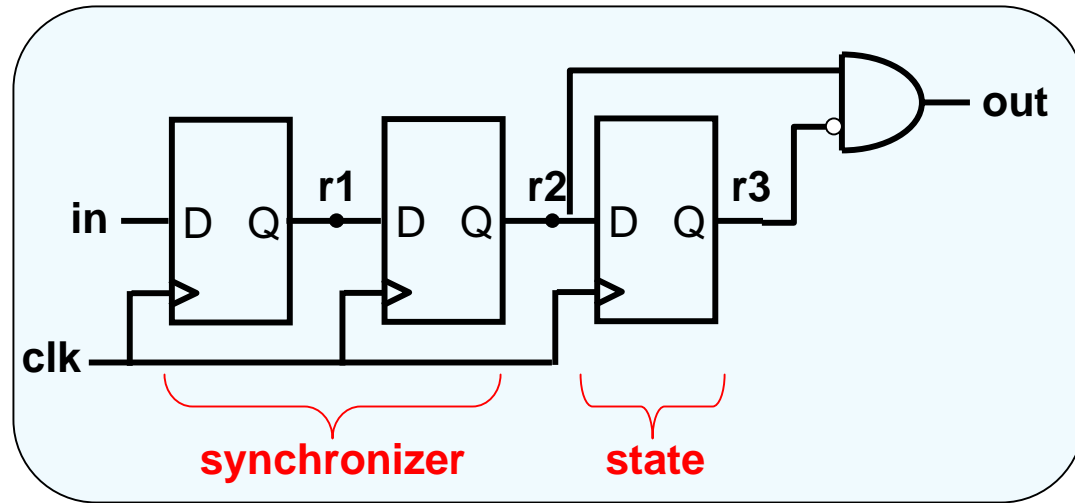
```
    r1 <= in;    // first reg in synchronizer  
    r2 <= r1;    // second reg in synchronizer, output is in sync!  
    r3 <= r2;    // remembers previous state of button
```

```
  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

```

```

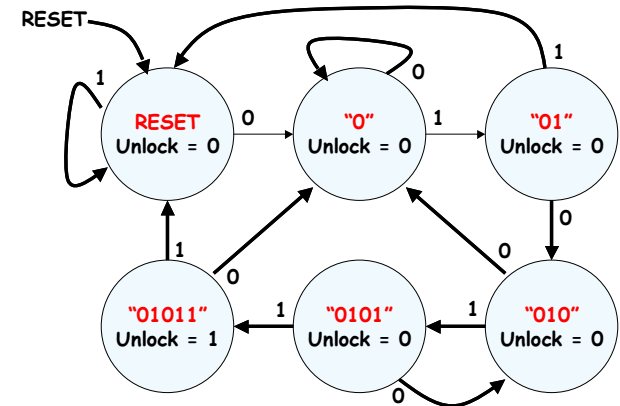
    endcase

```

```

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);

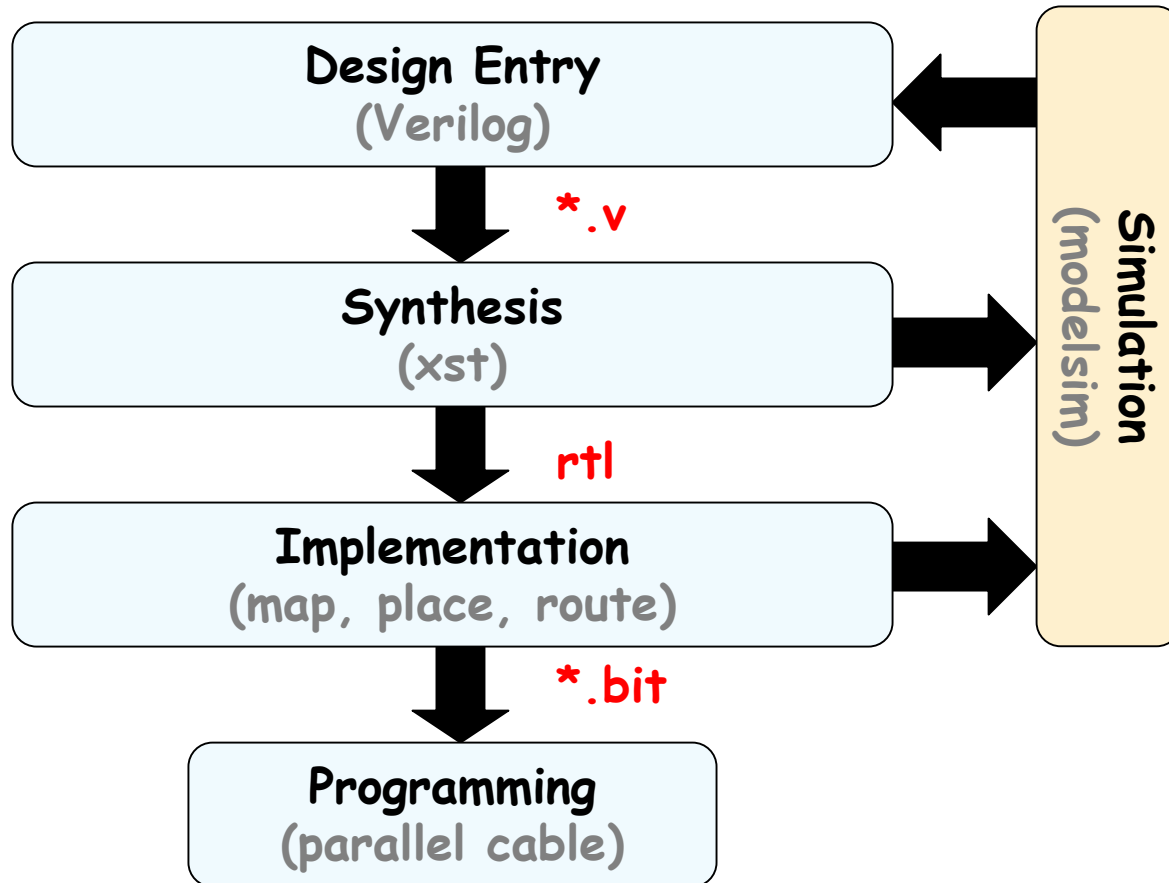
    parameter S_RESET = 0; parameter S_0 = 1; // state assignments
    parameter S_01 = 2; parameter S_010 = 3;
    parameter S_0101 = 4; parameter S_01011 = 5;

    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

- Fall '05 6.111: This year, we will use the Xilinx toolset
- Software: ISE 6.3i (windows / linux)



Step 3A: Load source file lock.v

The screenshot shows the Xilinx Project Navigator interface. The top window displays the project structure with 'demo1.isc' and 'xc2v6000-4bf957' as sources. A context menu is open over the 'lock (lock.v)' file, showing options like 'New Source...', 'Add Source...', 'Add Copy of Source...', 'Remove', 'Move to Library...', 'Open', 'Toggle Paths', and 'Properties...'. The 'Add Source...' option is selected. Below this, the 'Processes for Source: "lock"' window is visible, showing a list of processes including 'Add Existing Source', 'Create New Source', 'View Design Summary', 'Design Utilities', 'User Constraints', 'Synthesize - XST', 'Implement Design', 'Generate Programming File', 'Programming File Generation Report', 'Generate PROM, ACE, or JTAG File', and 'Configure Device (IMPACT)'. The main editor window shows the Verilog code for the 'lock' module, which includes input/output declarations, synchronization comments, wire declarations, button declarations, and a state transition diagram implementation.

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

Step 3B: Compile/Synthesize

The screenshot displays the Xilinx Project Navigator interface. The top window shows the 'Sources in Project' tree with 'demo1.isc', 'xc2v6000-4bf957', 'lock (lock.v)', and 'button (lock.v)'. The 'Processes for Source: "lock"' window shows the 'Synthesize -XST' step selected. The main editor window displays the Verilog code for 'lock.v', which defines a module with inputs 'clk', 'reset_in', 'b0_in', 'b1_in', and outputs 'out', 'hex_display'. The code includes comments for synchronizing push buttons and implementing a state transition diagram. The bottom console window shows the synthesis results, including the minimum period, minimum input arrival time, maximum output required time, and maximum combinational path delay.

Sources in Project:

- demo1.isc
- xc2v6000-4bf957
- lock (lock.v)
- button (lock.v)

Processes for Source: "lock"

- Add Existing Source
- Create New Source
- View Design Summary
- Design Utilities
- User Constraints
- Synthesize -XST
- Implement Design
- Generate Programming File

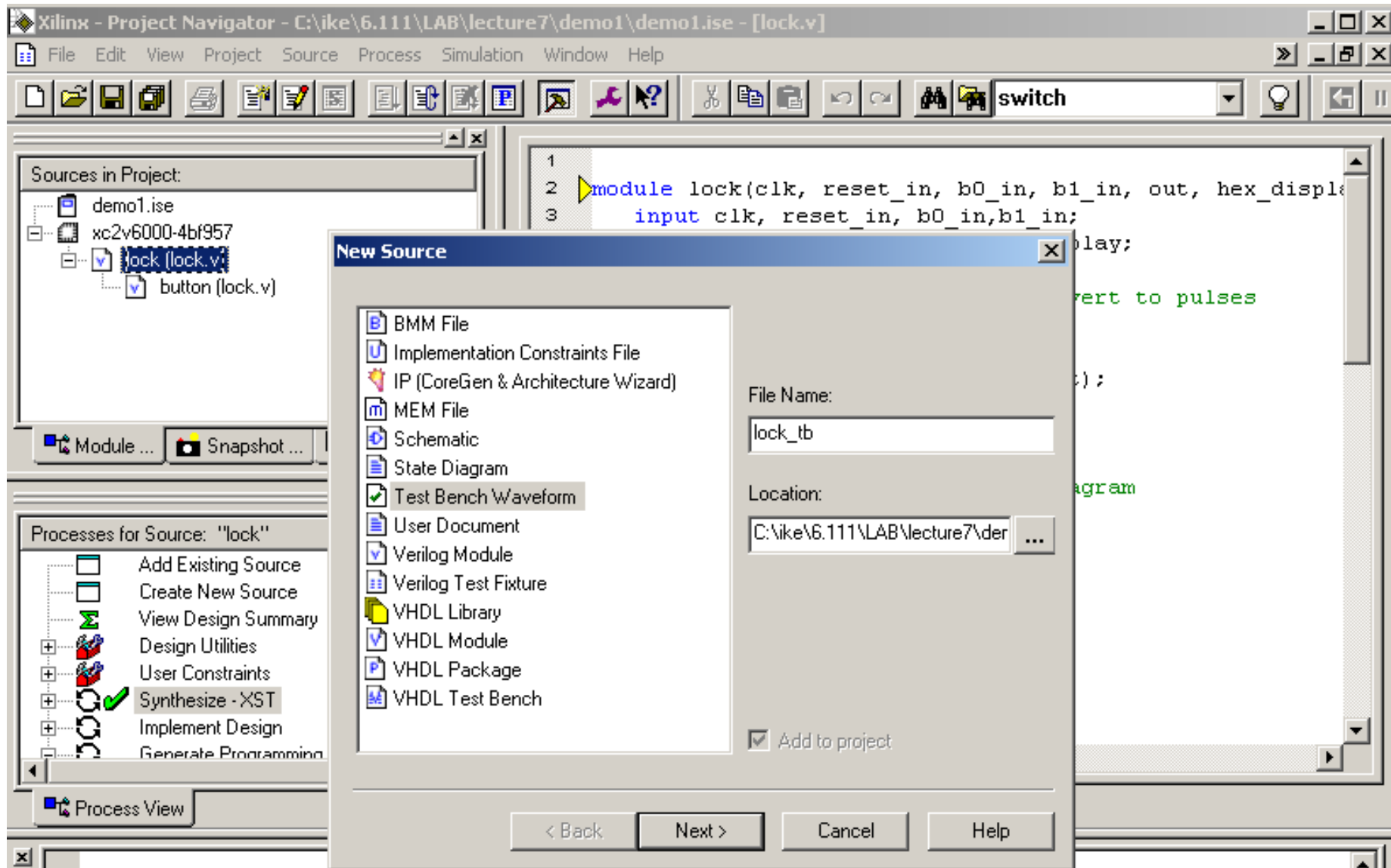
lock.v

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

Console:

```
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

The screenshot shows the Xilinx Project Navigator on the left with a project named 'demo1' containing files 'demo1.isc', 'xc2v6000-4bf95', 'lock (lock.v)', and 'button (button.v)'. The 'Processes for Source' list includes 'Add Existing', 'Create New', 'View Design', 'Design Utilities', 'User Constraints', 'Synthesize', 'Implement', and 'Generate Report'. The 'Initialize Timing' dialog box is open in the center, featuring a timing diagram at the top and several configuration sections.

Initialize Timing

The timing diagram illustrates the clock signal and its relationship to input and output delays. It shows a clock signal with a period of 100 ns, high for 100 ns and low for 100 ns. The diagram also indicates the 'Maximum output delay' (green arrows) and 'Minimum input setup' (grey arrows) relative to the clock edges.

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

Buttons: OK, Cancel, Next >, Help

Step 3B: Generate Simulation Results

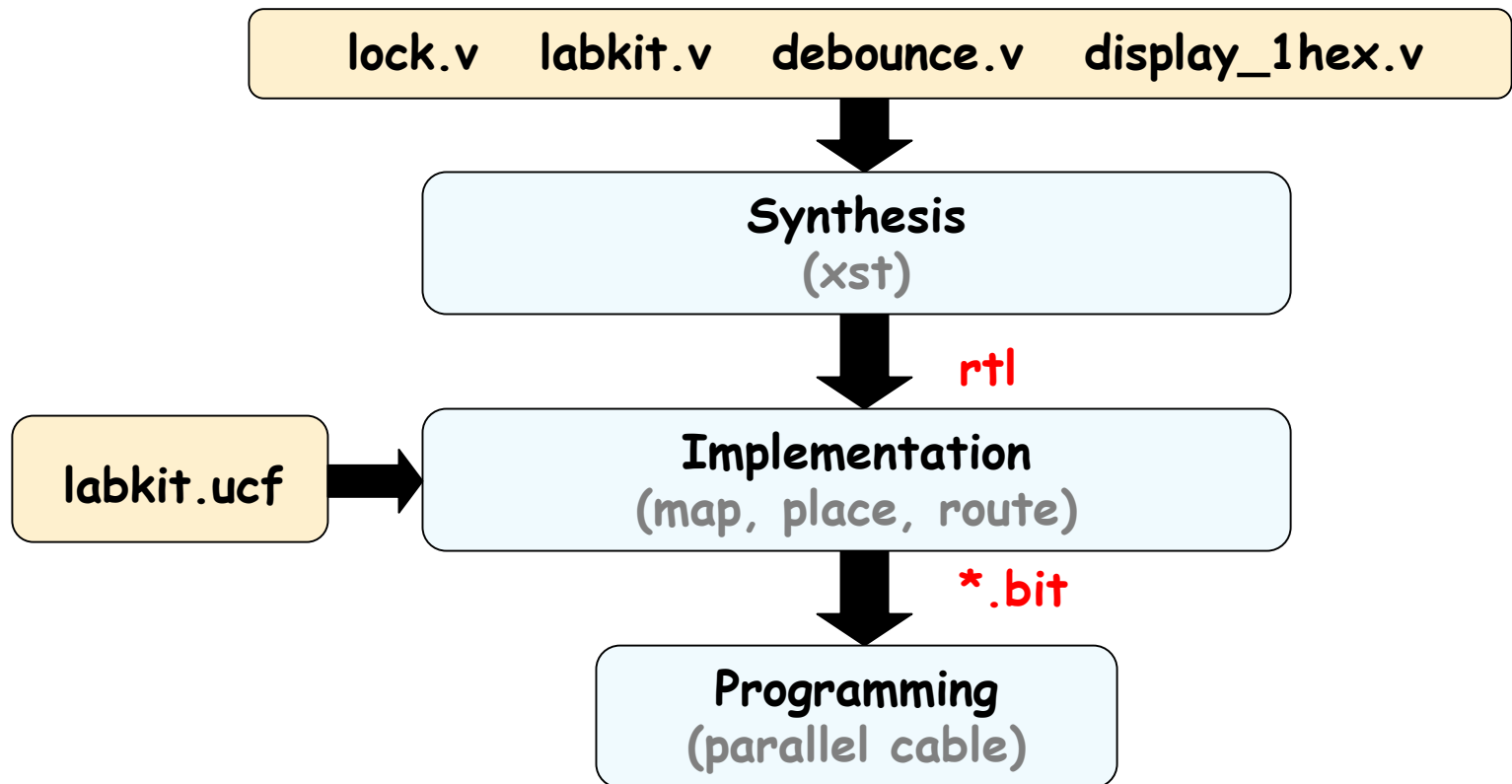
The screenshot displays the Xilinx ISE Project Navigator interface. The top menu bar includes File, Edit, View, Project, Source, Process, Simulation, Test Bench, Window, and Help. The toolbar contains various icons for file operations and simulation. The 'Sources in Project' pane on the left shows the project structure: demo1.isc, xc2v6000-4bf957, lock (lock.v), lock_tb (lock_tb.tbw), and button (lock.v). The 'Processes for Source: "lock_tb"' pane shows the simulation process: Add Test Bench To Project, Xilinx ISE Simulator, Generate Expected Simulation Results (highlighted), Simulate Behavior, and Simulate Post-Place. The main window displays the simulation results for the 'lock_tb.tbw' test bench. The 'End Time' is 4000 ns. The simulation time scale is 100 ns. The signals shown are clk, b0_in, b1_in, reset_in, out, and hex_display[...]. The 'out' signal is a square wave. The 'hex_display[...]' signal shows the hexadecimal values 0, 1, 2, 3, 4, and 5. The bottom console window shows the following text:

```
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"
```

The console window also has buttons for Console, Find in Files, Errors, and Warnings.

Step 4: Implementation - Program FPGA

- Pin assignments: User constraints file
- Labkit: peripherals definitions
- Optimization: Placing and Routing



Step 4: Implementation - Program FPGA

- Pin assignments: User constraints file

labkit.ucf

Design Object List - I/O Pins

	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

#	Group	Loc	I/O Std.
8	vga_out_red		LVTTL
8	vga_out_green		LVTTL
8	vga_out_blue		LVTTL
10	tv_out_ycrb		LVDCl_33
20	tv_in_ycrb		
36	ram0_data		LVDCl_33
18	ram0_address		LVDCl_33

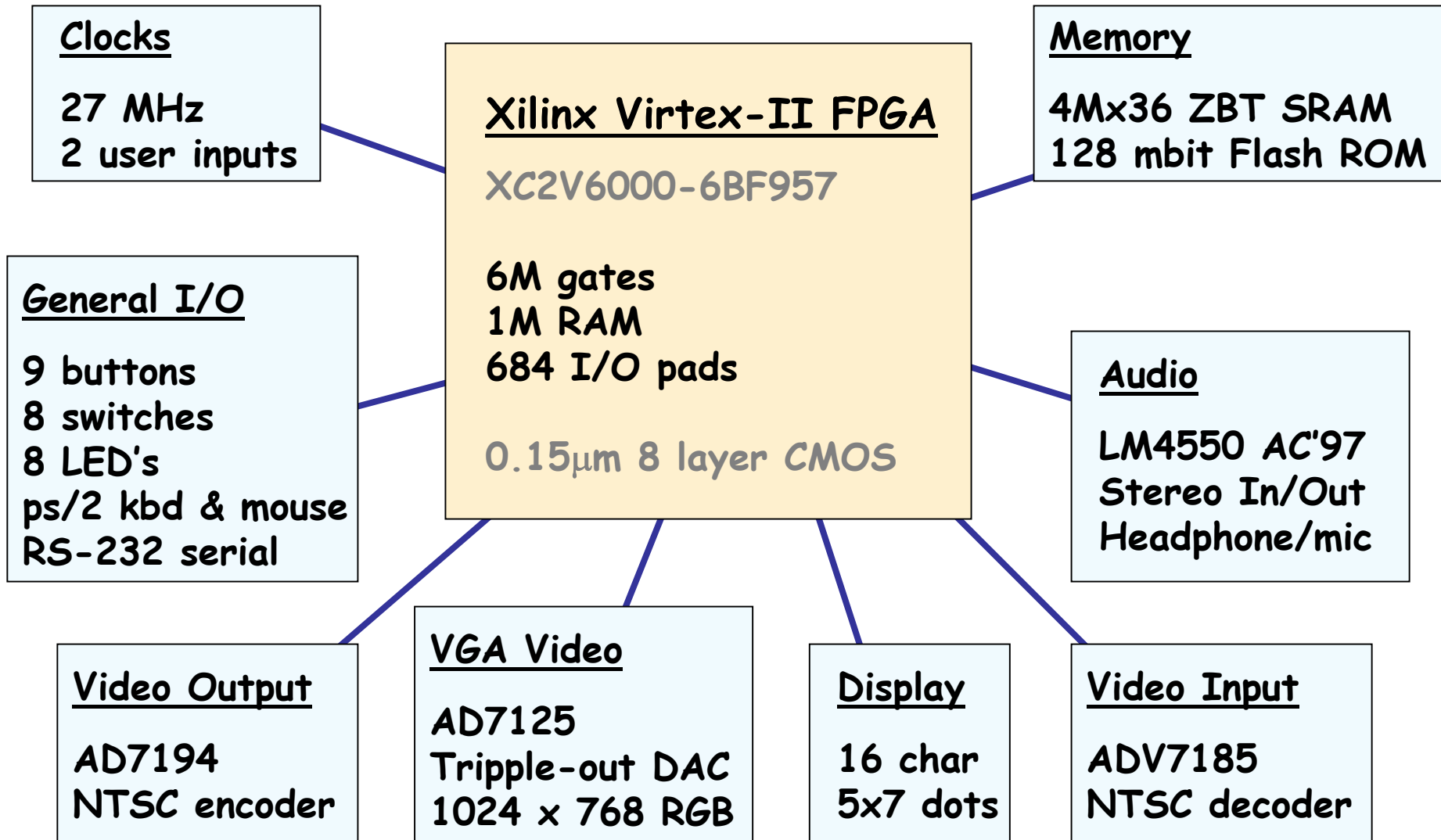
Package Pins for xc2v6000-4-bf957 (Flight Time)

Top View

Package View | Architecture View

The 6.111 Labkit: Subsystems

Nathan Ickes



Step 4A: FPGA Device Assignment

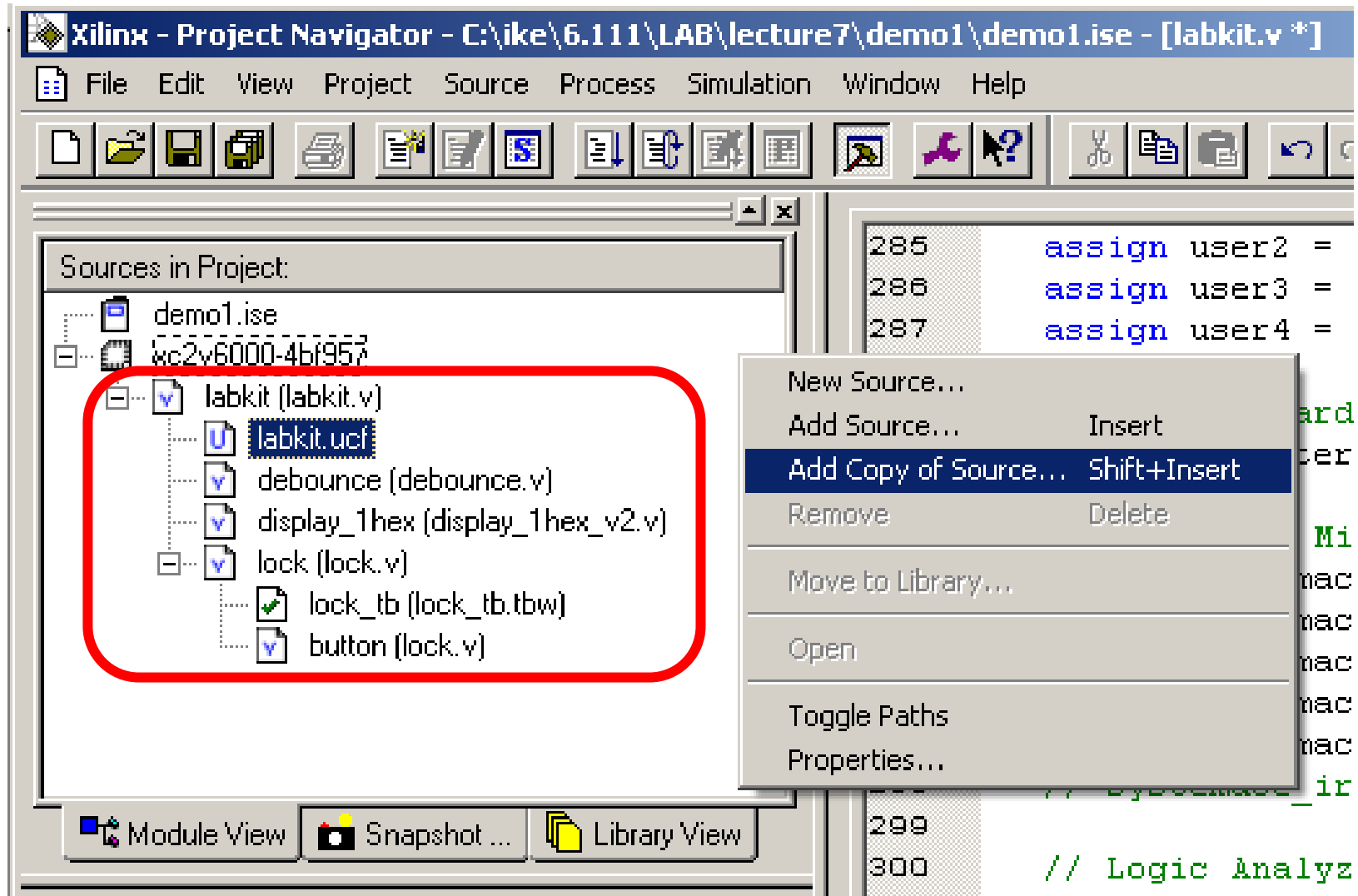
The screenshot shows the Xilinx Project Navigator interface. The 'Project Properties' dialog box is open, displaying the following settings:

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

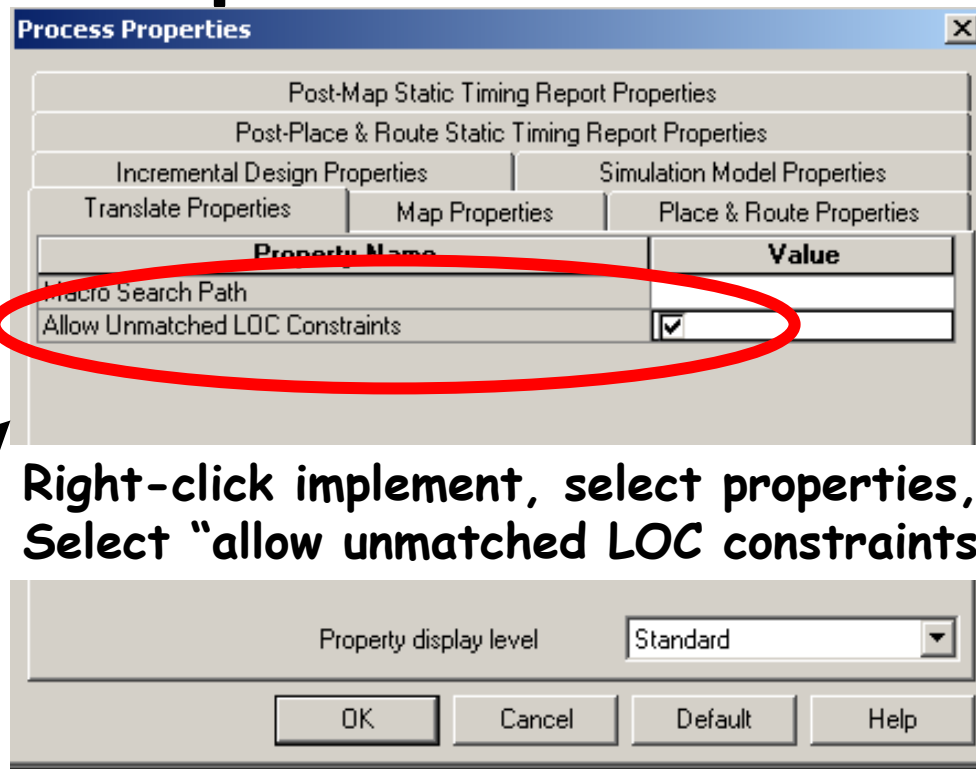
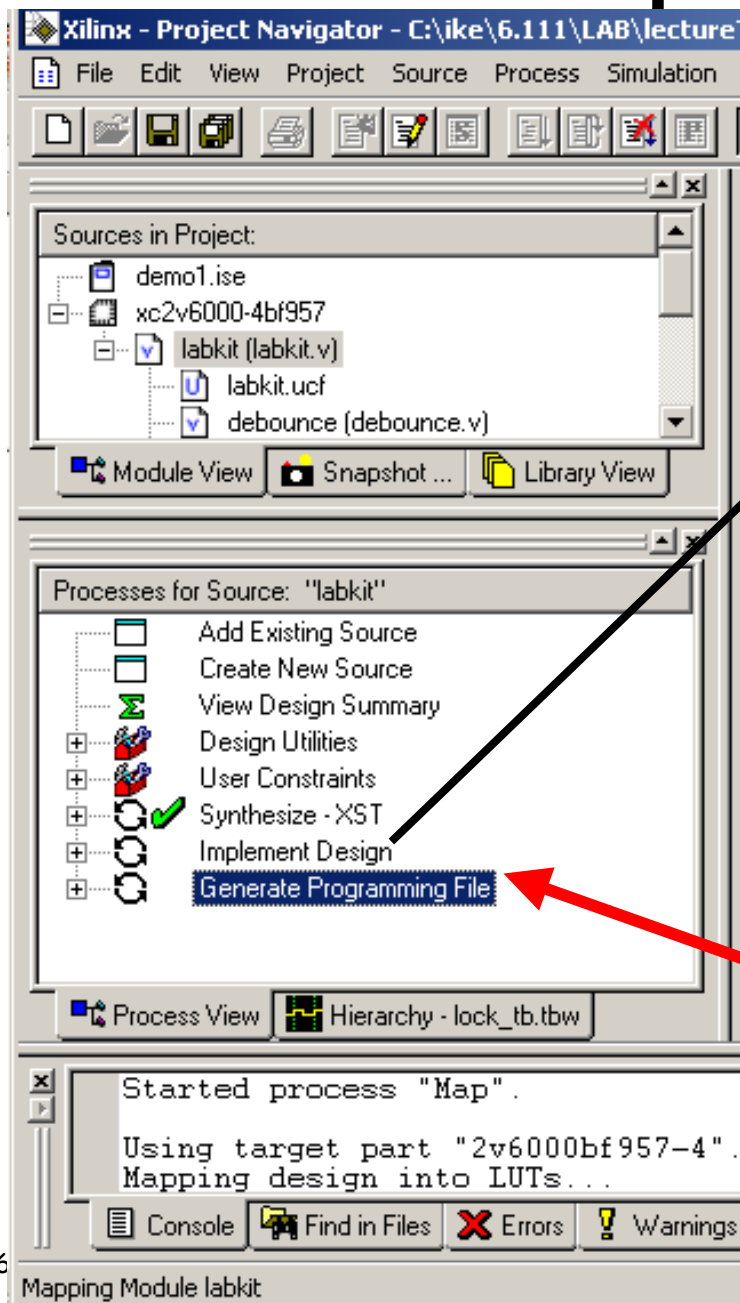
Annotations in the image:

- Red text at the bottom: **Virtex 2: xc2v6000** (with an arrow pointing to the 'Device' field)
- Red text at the bottom: **Package: bf957** (with an arrow pointing to the 'Package' field)
- Red text on the right: **Speed: -4** (with an arrow pointing to the 'Speed Grade' field)
- Green text on the right: **dy are inputs**

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 title bar reads "Xilinx - Project Navigator - C:\ike\6.111\LAB\lecture7\demo1\demo1.isc - [Design Summary]". The menu bar includes File, Edit, View, Project, Source, Process, Simulation, Window, and Help. The toolbar contains various icons for file operations and design actions. The left pane shows the "Sources in Project" tree with files like demo1.isc, xc2v6000-4bf957, labkit (labkit.v), and labkit.ucf. Below it, the "Processes for Source: 'labkit'" list shows steps like Add Existing Source, Create New Source, View Design Summary (highlighted), Design Utilities, User Constraints, Synthesize - XST, Implement Design, Translate, Map, and Place & Route. The right pane displays the "Design Overview for labkit" with a table of properties 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

The bottom of the window shows a tabbed interface with tabs for lock.v, lock..., labk..., and Design... The status bar at the bottom indicates "Process View" and "Hierarchy - lock_tb.tbw".

Step 4C: Implement

- Useful reports: Resource Utilization, Timing, RTL diagram

The screenshot shows the Xilinx Project Navigator interface. The top window displays the project files, including 'demo1.isc', 'xc2v6000-4bf957', 'labkit (labkit.v)', and 'labkit.ucf'. The bottom window shows the 'Processes for Source: "labkit"' list, with 'Place & Route' and 'Asynchronous Delay Report' highlighted. The right window displays the 'Asynchronous Delay Report' for 'labkit.dly', showing the 20 worst nets by delay.

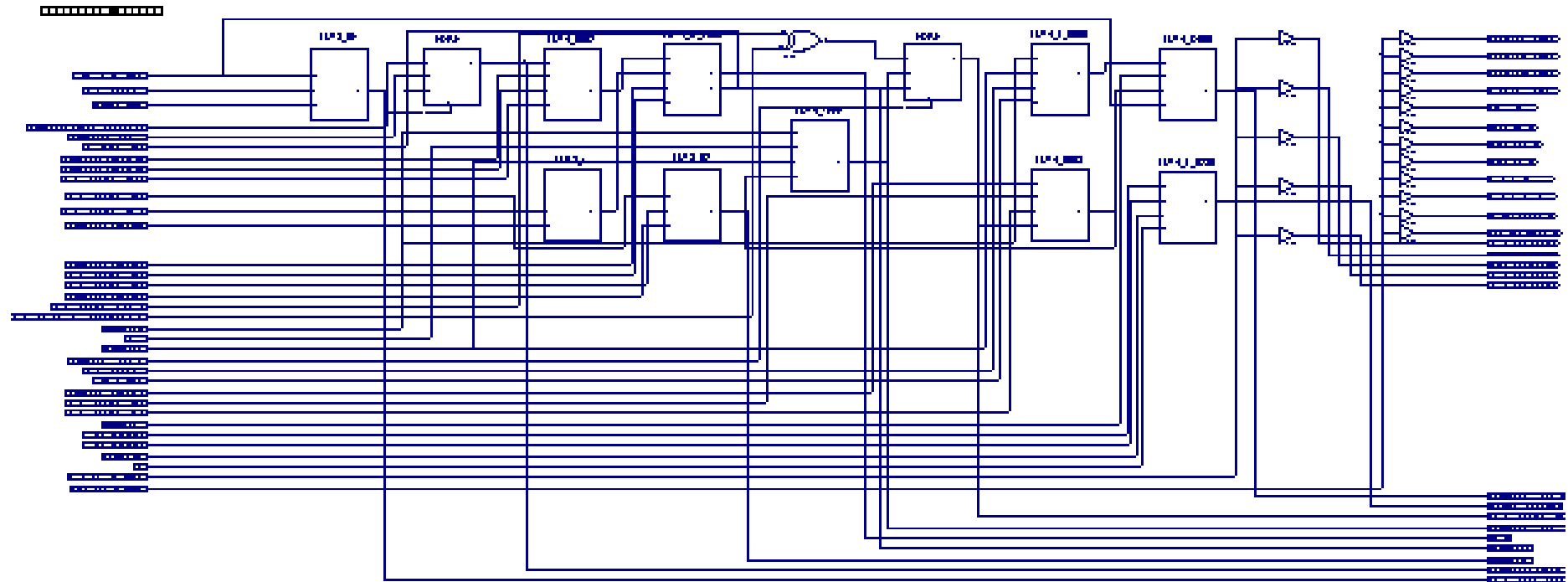
Release 7.1.03i - reportgen H.41
Copyright (c) 1995-2005 Xilinx, Inc. All rights reserved.
Sun Sep 25 16:39:00 2005
File: labkit.dly

The 20 worst nets by delay are:

Max Delay	Netname
5.778	button_enter_IBUF
5.467	debounce0/clean
5.184	button1_IBUF
4.807	debounce1/clean
4.462	hexdisp1/clock
4.300	button0_IBUF
3.827	hexdisp1/dispc_b
3.760	hexdisp1/state_FFd5
3.583	hexdisp1/dreset
3.340	hexdisp1/disprs

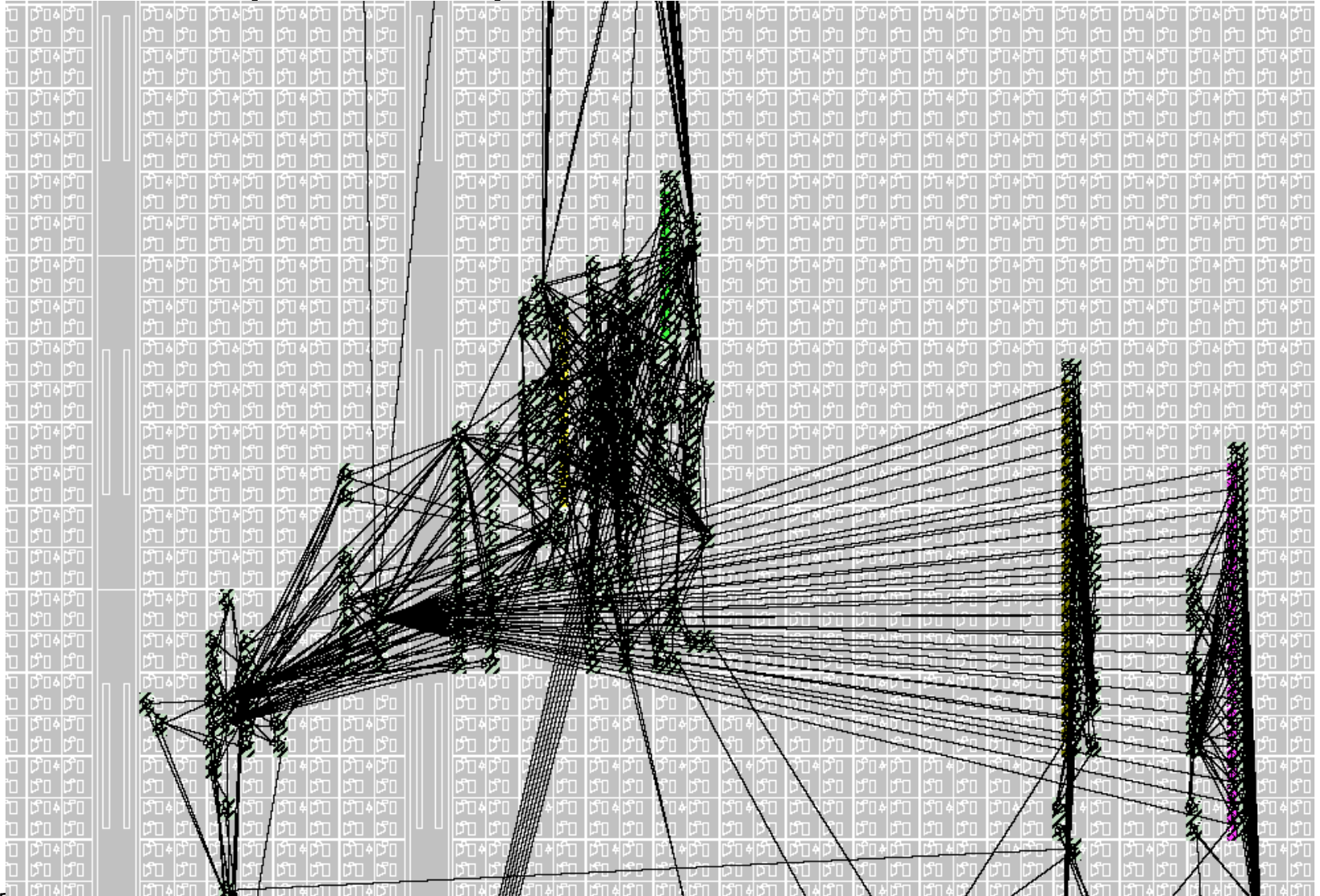
Step 4C: Implement

- Useful reports: Resource Utilization, Timing, RTL diagram



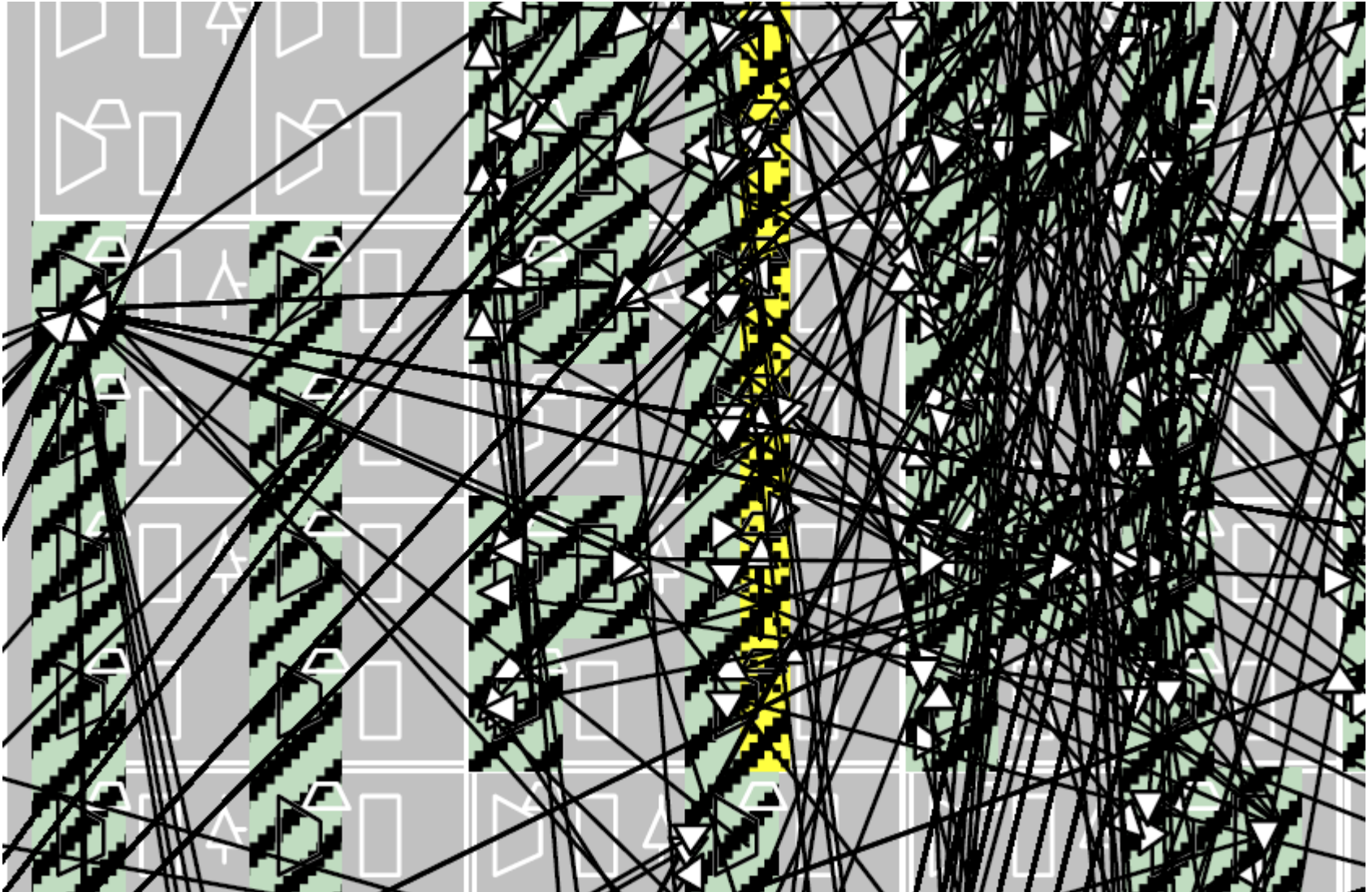
Step 4C: Implement

- Useful reports: Floorplan



Step 4C: Implement

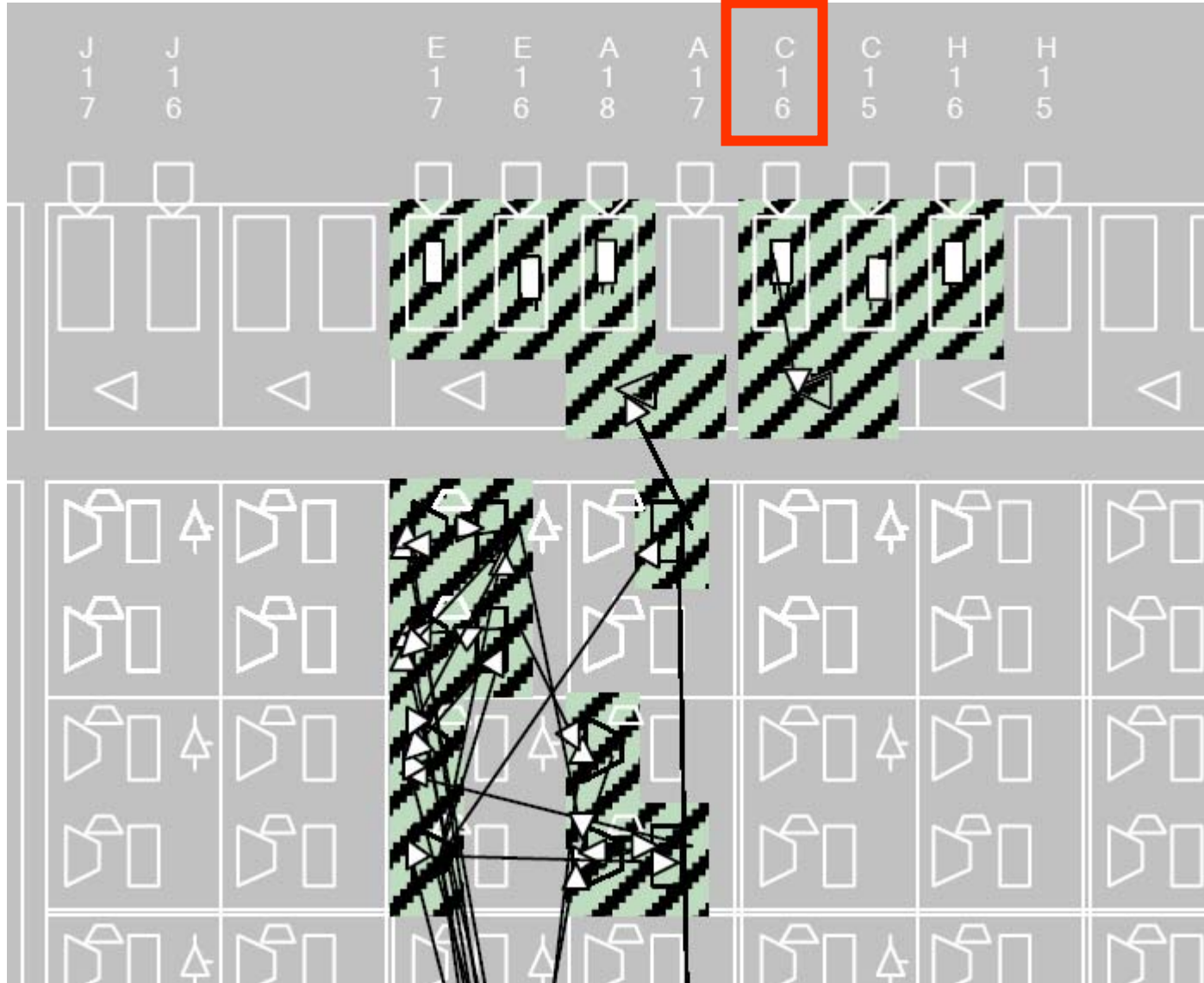
- Useful reports: Floorplan



Step 4C: Implement

- Useful reports: Floorplan

clock_27mhz



Step 4D: Program FPGA

- Fall '2005 6.111: We'll use the parallel port IV cable
- Transfer program: "IMPACT" - uses JTAG serial chain
joint test access group

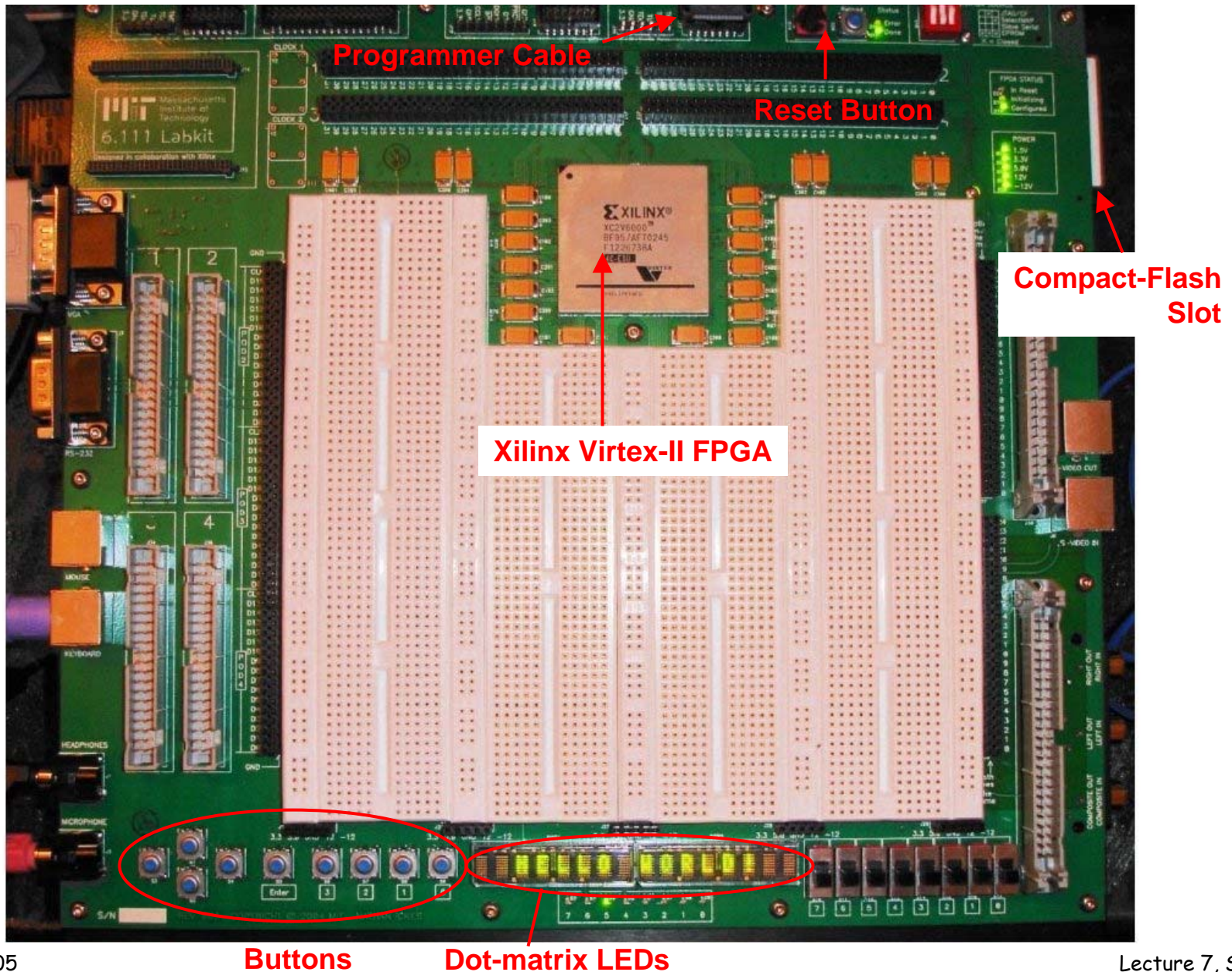
The screenshot shows the Xilinx ISE software interface. On the left, the 'Processes for Source: "labkit"' window is open, displaying a tree of tasks. The 'Configure Device (iMPACT)' task is highlighted. Below this, a context menu is open with options: Run, Rerun, Rerun All, Stop, Open Without L, and Properties... The 'Run' option is selected. On the right, the 'Boundary-Scan' window is open, showing a JTAG chain diagram. The diagram consists of two Xilinx devices connected in series. The first device is labeled 'xc2v6000 labkit.bit' and the second is labeled 'xc2v6000 labkit.bit'. A red circle highlights the 'Program...' button in the context menu, which is also highlighted by a red arrow. The 'Boundary-Scan' window also shows a menu with options: Program..., Verify, Get Device ID, Get Device Signature/Usercode, IDCODE Looping..., and Assign New Configuration File...

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?

