

Introduction and the architecture of FPGA

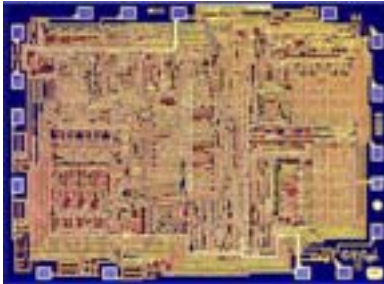
Evolution of Integrated Circuits

- < 1960 individual transistors
- 1960s – 1970s:
 - SSI, MSI, LSI (10,000 transistors)
- 1980s:
 - Programmable Logic Devices (PLAs, PALs)
 - 16-bit, 32-bit processors (> 1,000,000 transistors)
- 1990s:
 - Full custom chips
 - Gate arrays (semi-custom chips)
 - **Field-Programmable Gate Arrays**
 - > 100,000,000 transistors
- **2010: > 1,000,000,000 transistors**
 - **Intel Quad-core Itanium (2 billion)**



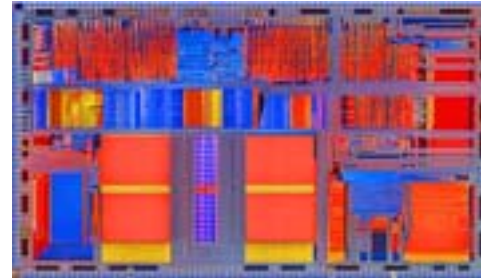
First transistor,
Patented 1964

Intel i4004 (1971)



2,300 transistors

Intel i486 (1988)

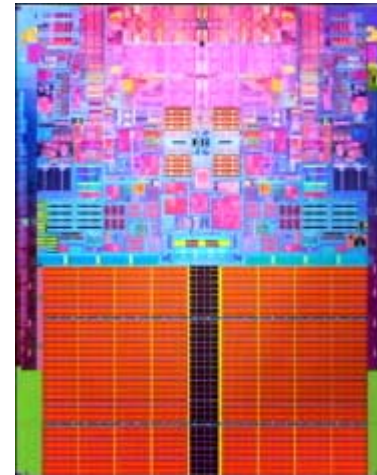


1.2 million transistors



P4 (2000)

55 million transistors

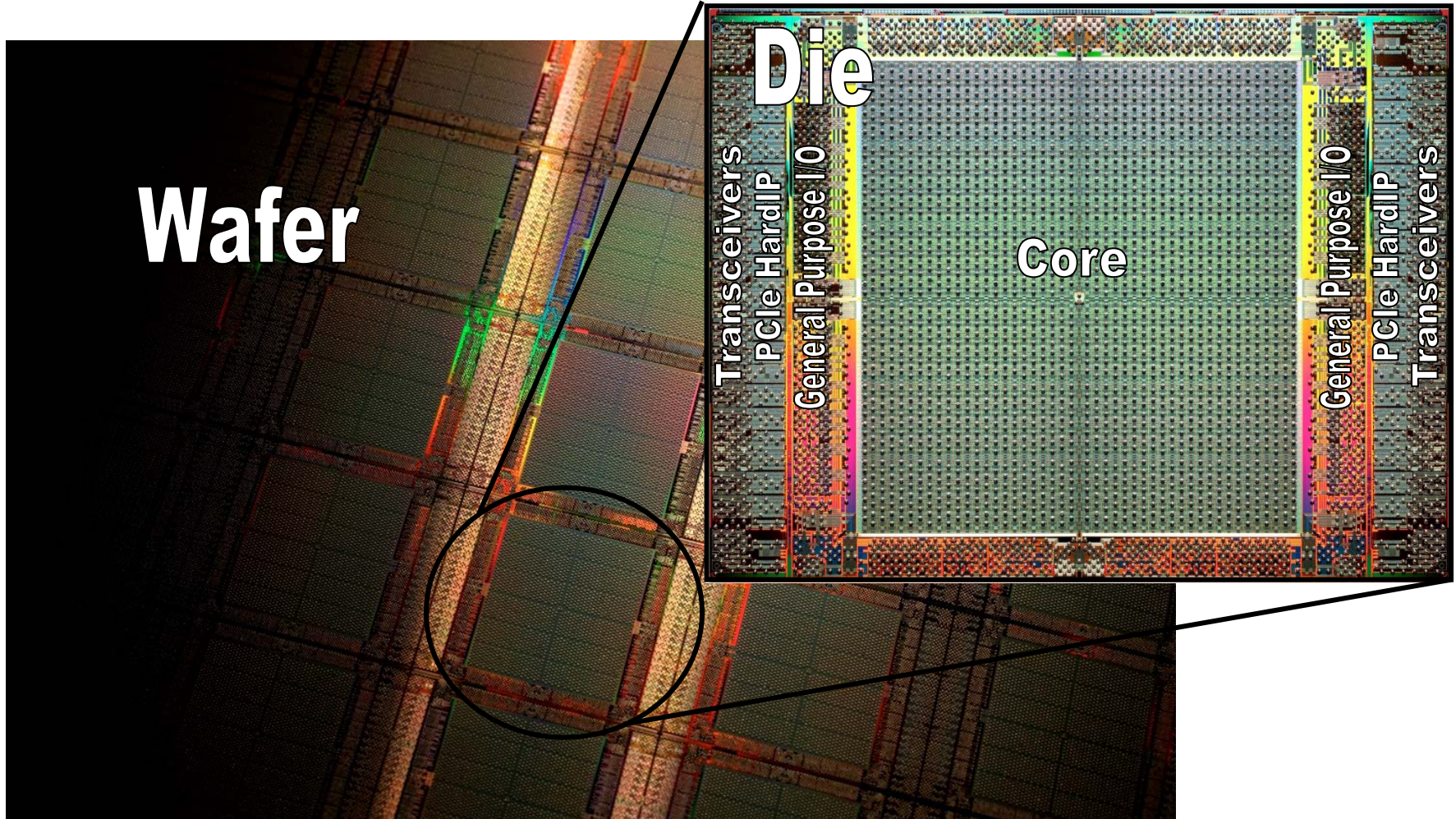


**Core2 Duo
(2006)**

291 million transistors

Stratix IV GX FPGA

2.5 billion transistors



Why Teach Students about FPGAs?

- Field-Programmable Gate Arrays are programmable hardware chips
 - Can be used to implement any digital hardware circuit
- Digital hardware is found in almost all modern products
 - Consumer products, like audio and video players
 - Telephone and television equipment
 - Cars, airplanes, trains
 - Medical equipment, and industrial control
- FPGAs are ... everywhere

Industries that use FPGAs

*Consumer
Automotive*



*Test,
Measurement
& Medical*



*Communications
Broadcast*



Military & Industrial



*Computer &
Storage*



Traditional FPGA Applications



Altera FPGA Applications



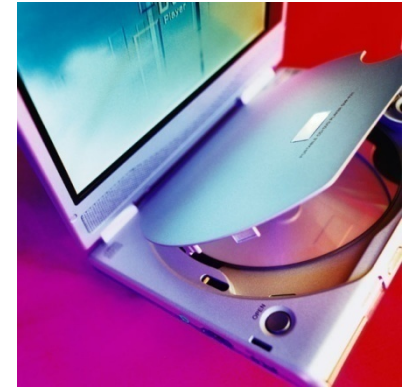
Consumer Applications



Set-Top Boxes



Touch Panels



DVD Players



**LCD, Plasma
Displays/TVs**



Consumer Music



Printers



**Handheld
Media Players**



Camcorders

What are FPGAs and why should we use them?

FPGA

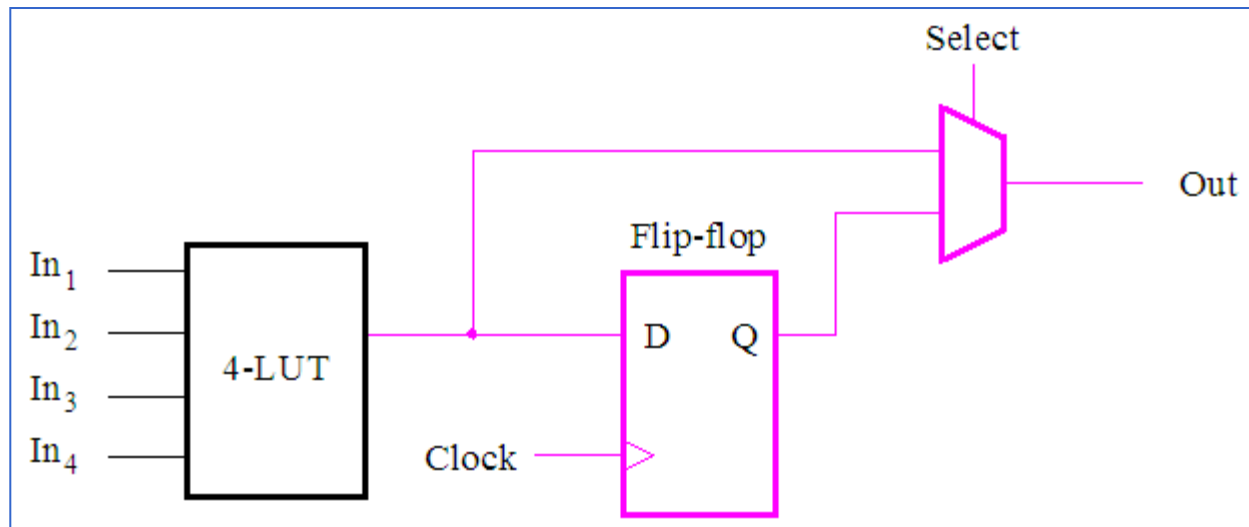
- **Field Programmable** Gate Array

- A large set of *programmable* logic elements
- Connected with *programmable switches*

- FPGA: re-programmable hardware

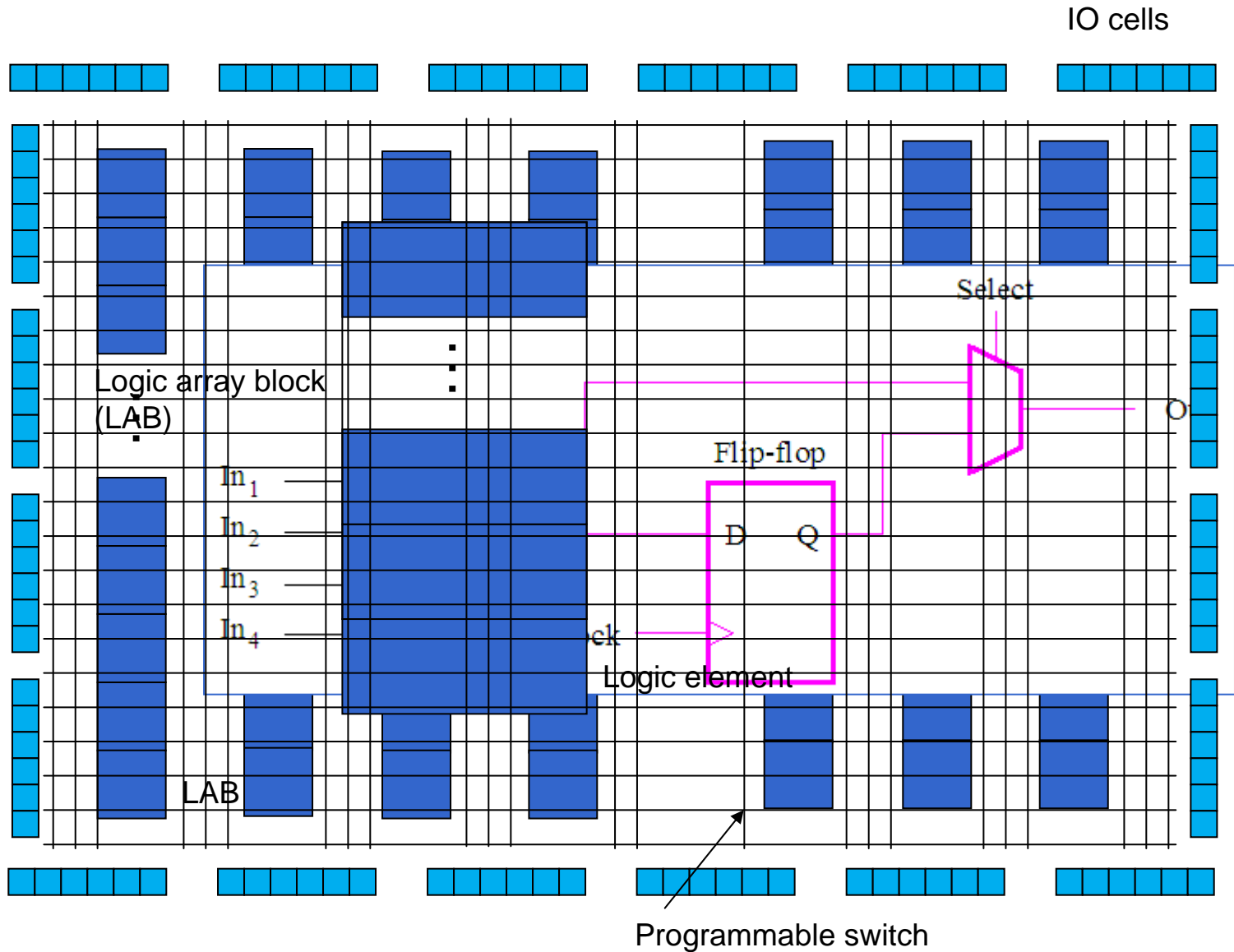
FPGA Architecture: Logic Element

- Lookup table (LUT) implements any 4-input logic function

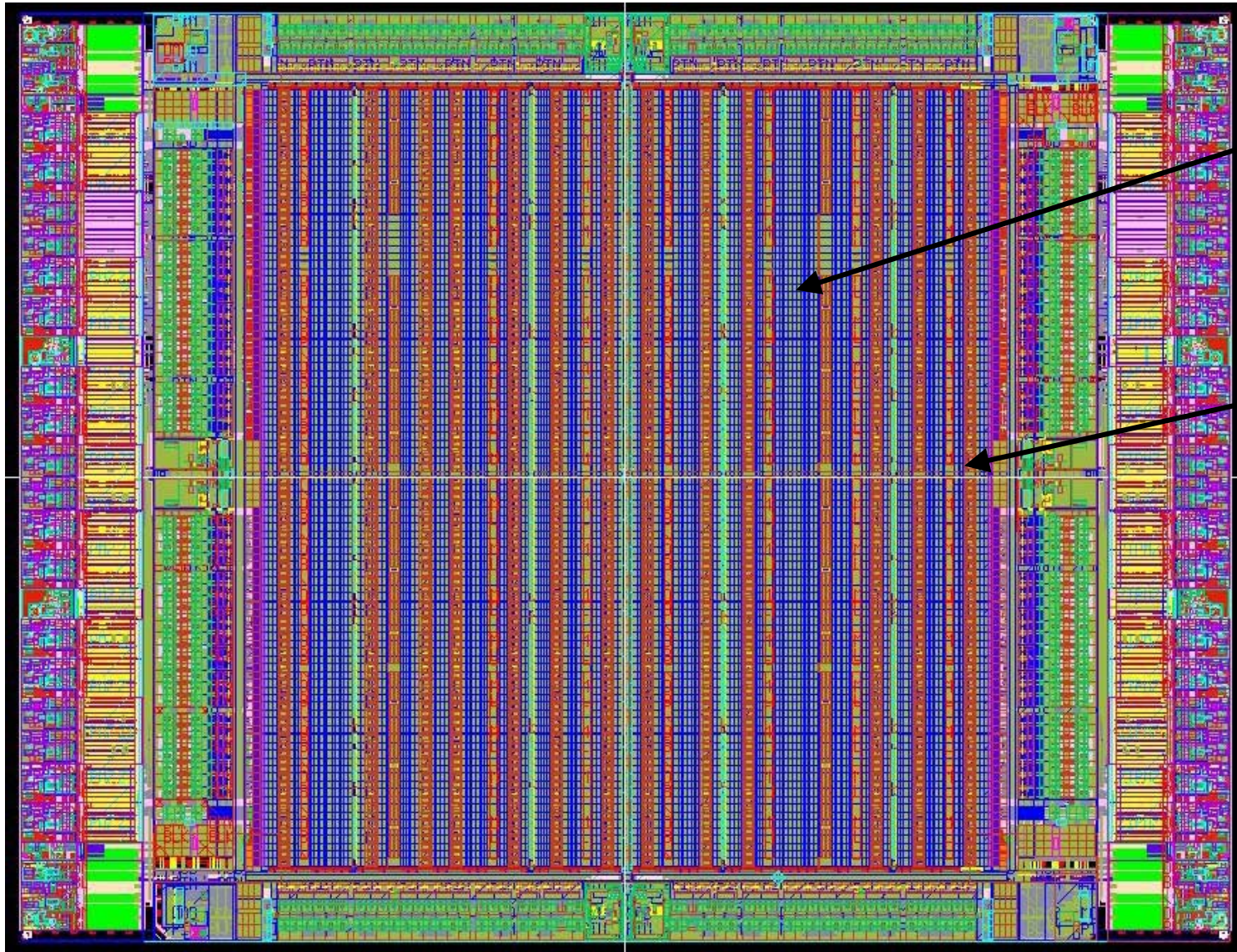


- Actual LE is significantly more complex

FPGA Architecture



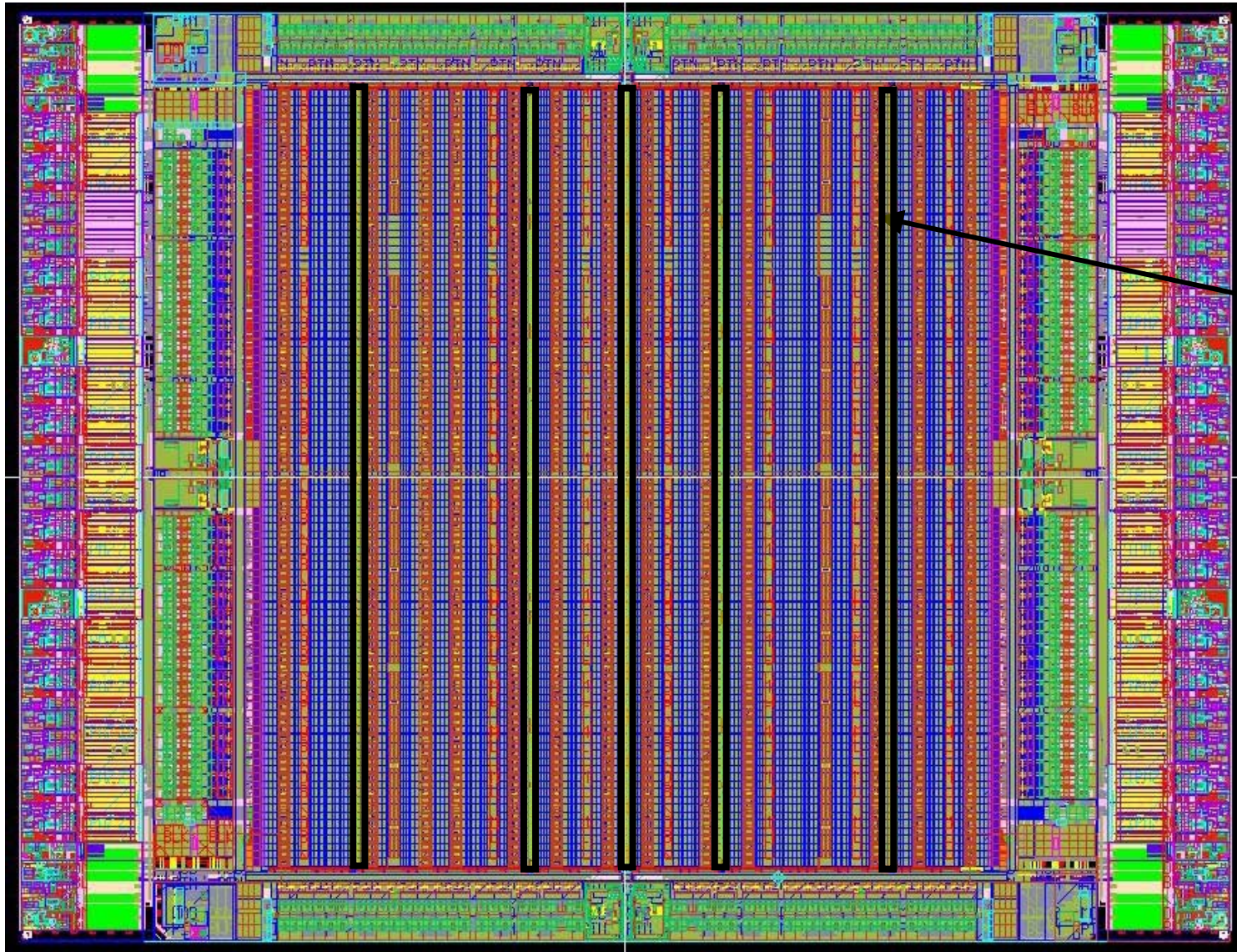
Stratix IV GX (more than 10 million gates)



Adaptive
Logic
Modules

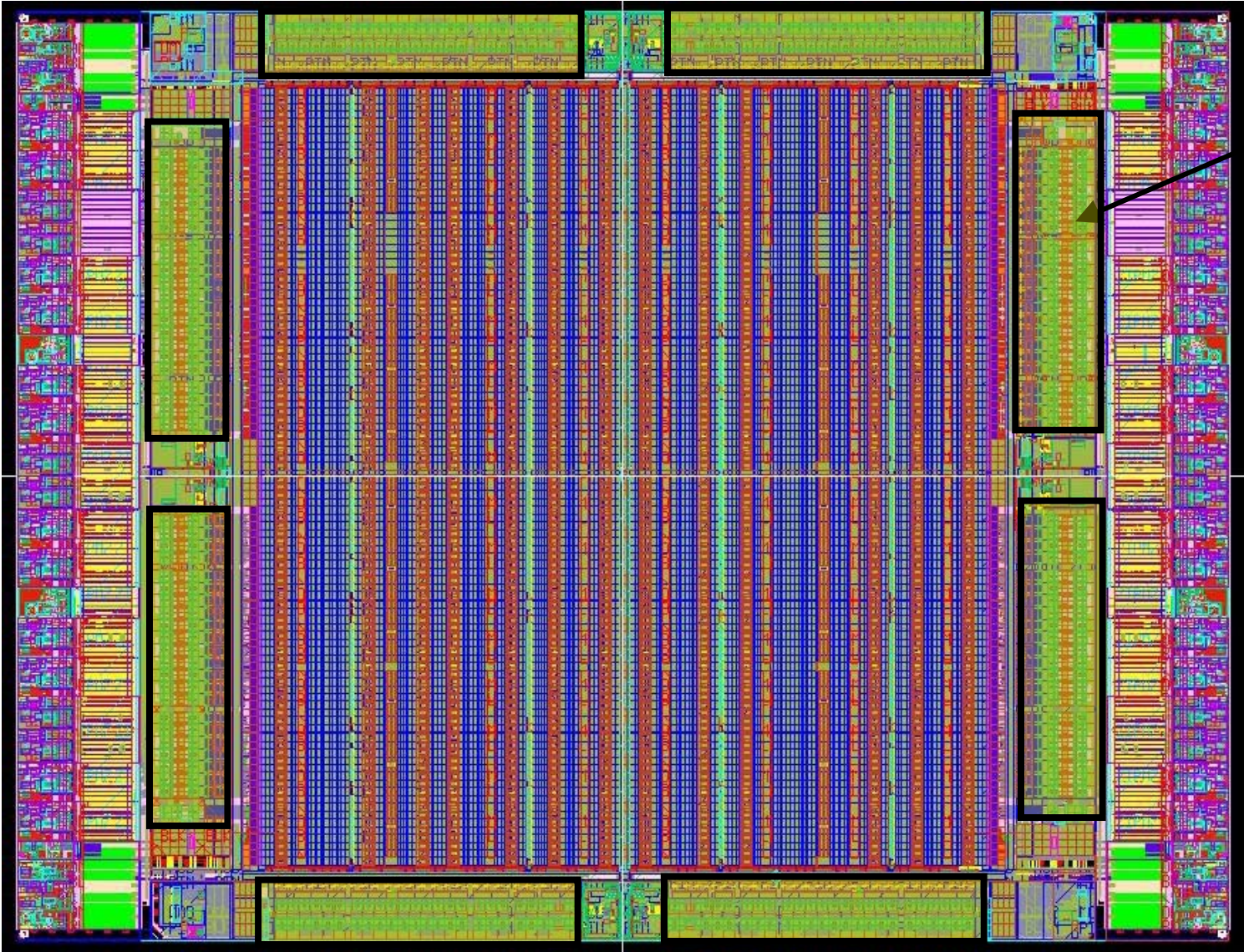
RAM
Blocks
(M9K &
M144K)

Digital Signal Processing



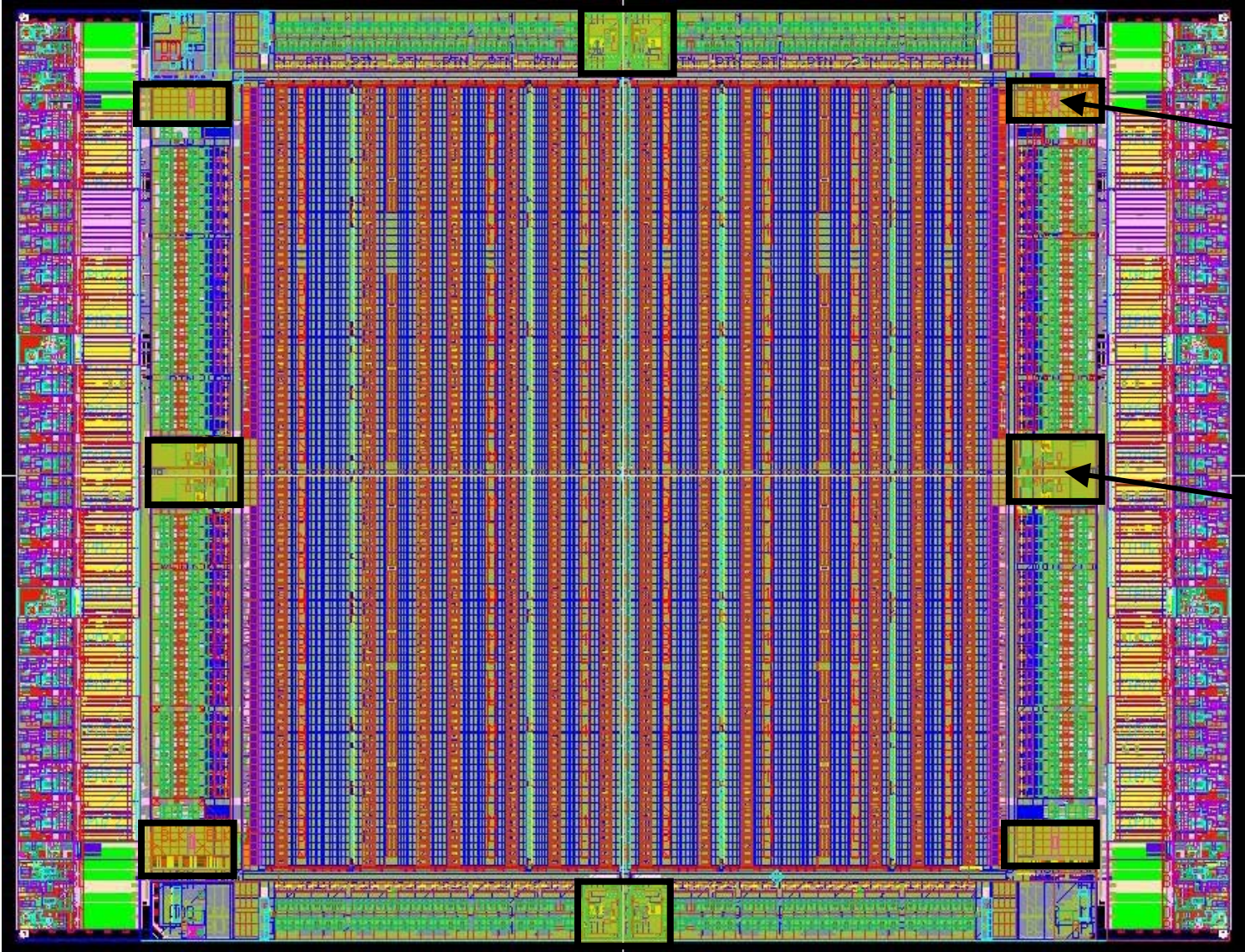
DSP
Blocks

General I/O



Programmable I/O

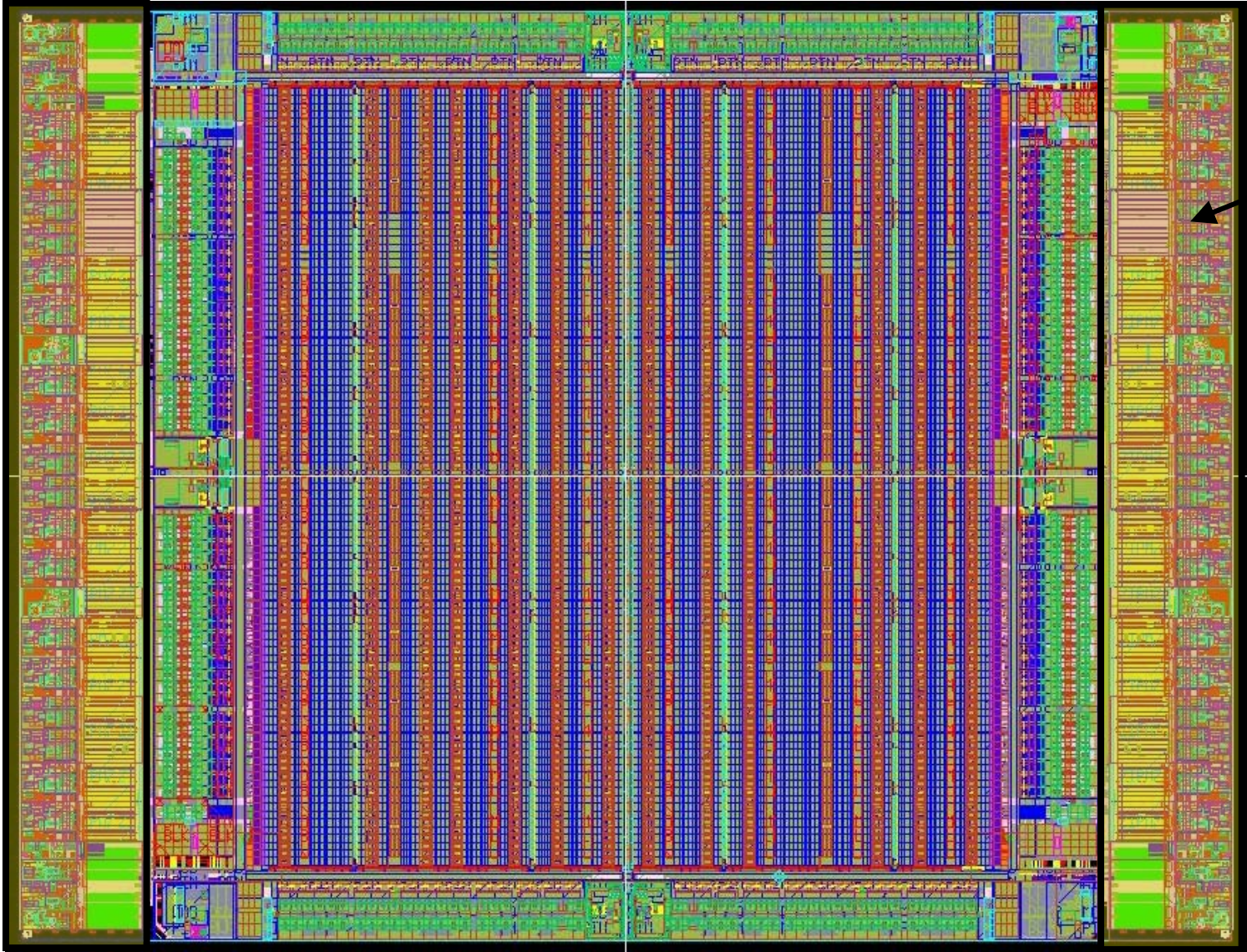
Clock Management



Delay Locked Loops (DLLs)

Phase Locked Loops (PLLs)

Serial Interfaces



High Speed Serial Interfaces



Introducing Arria II GX FPGAs

- High functionality
 - Optimized logic, memory, and digital signal processing (DSP) ratios for 3-Gbps applications
 - Up to 16 transceivers @ 3.75 Gbps
- Lowest power 3-Gbps FPGA
 - 40-nm process with 0.9V core voltage
 - <100 mW per transceiver channel
 - Integrated power optimization tool
- Design with ease
 - Built-in PCI Express hard IP
 - Single design environment
 - Jump-start design with protocol IP packs, reference designs and development kits



Lowest power, cost optimized for 3 Gbps

Arria II GX Family

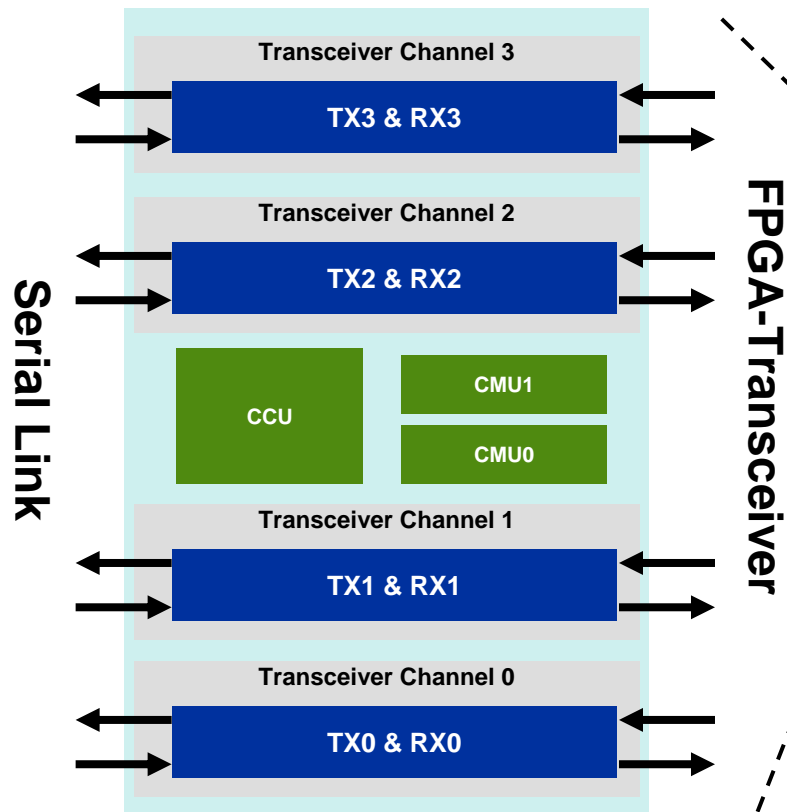
Device	Equiv LEs	Maximum Transceiver Channels ⁽¹⁾	PCI Express Hard IP Blocks	Total Memory MBits	18 X 18 Multipliers	Max. LVDS I/O ⁽¹⁾⁽²⁾	Phase-locked loops (PLLs) ⁽¹⁾
EP2AGX20	16K	4	1	0.98	56	48	4
EP2AGX30	27K	4	1	1.6	144	48	4
EP2AGX45	45K	8	1	3.4	228	56	4
EP2AGX65	63K	8	1	5.2	312	56	4
EP2AGX95	94K	12	1	6.7	448	64	6
EP2AGX125	124K	12	1	8.1	576	64	6
EP2AGX190	190K	16	1	9.9	656	96	6
EP2AGX260	256K	16	1	11.8	736	96	6

Notes:

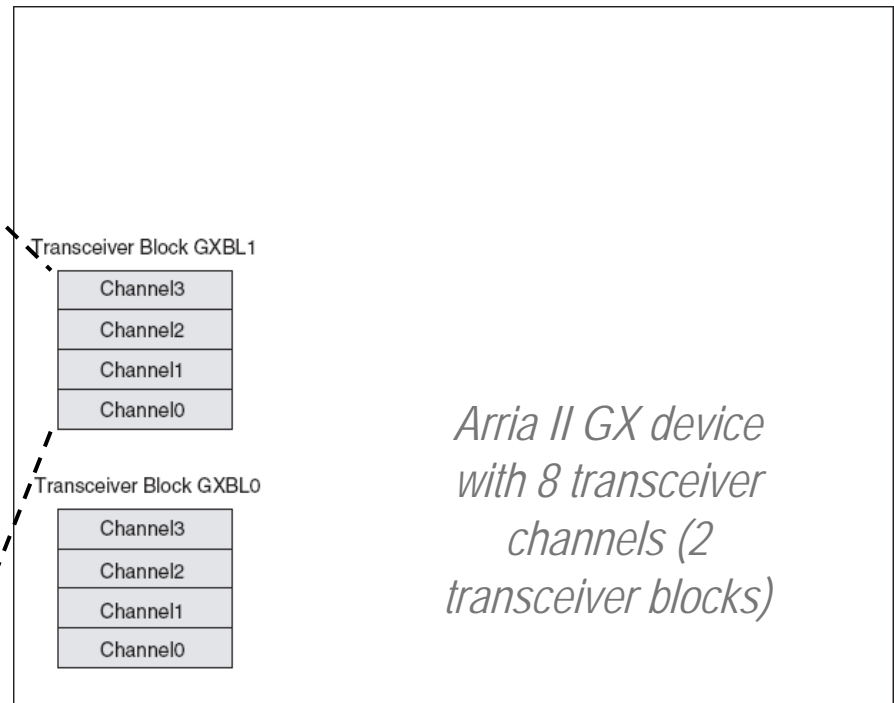
- 1) Number of transceiver channels and LVDS I/O determined by packaging.
- 2) Maximum LVDS I/O denotes channels with built-in SERDES (up to 1 Gbps). See Handbook for details.

Arria II GX Transceiver Block Architecture

- Up to 4 transceiver blocks
- Left side of device only
- Dynamic reconfiguration support



CCU = Clock Control Unit; CMU = Clock Management Unit



*Arria II GX device
with 8 transceiver
channels (2
transceiver blocks)*

Why use FPGAs?

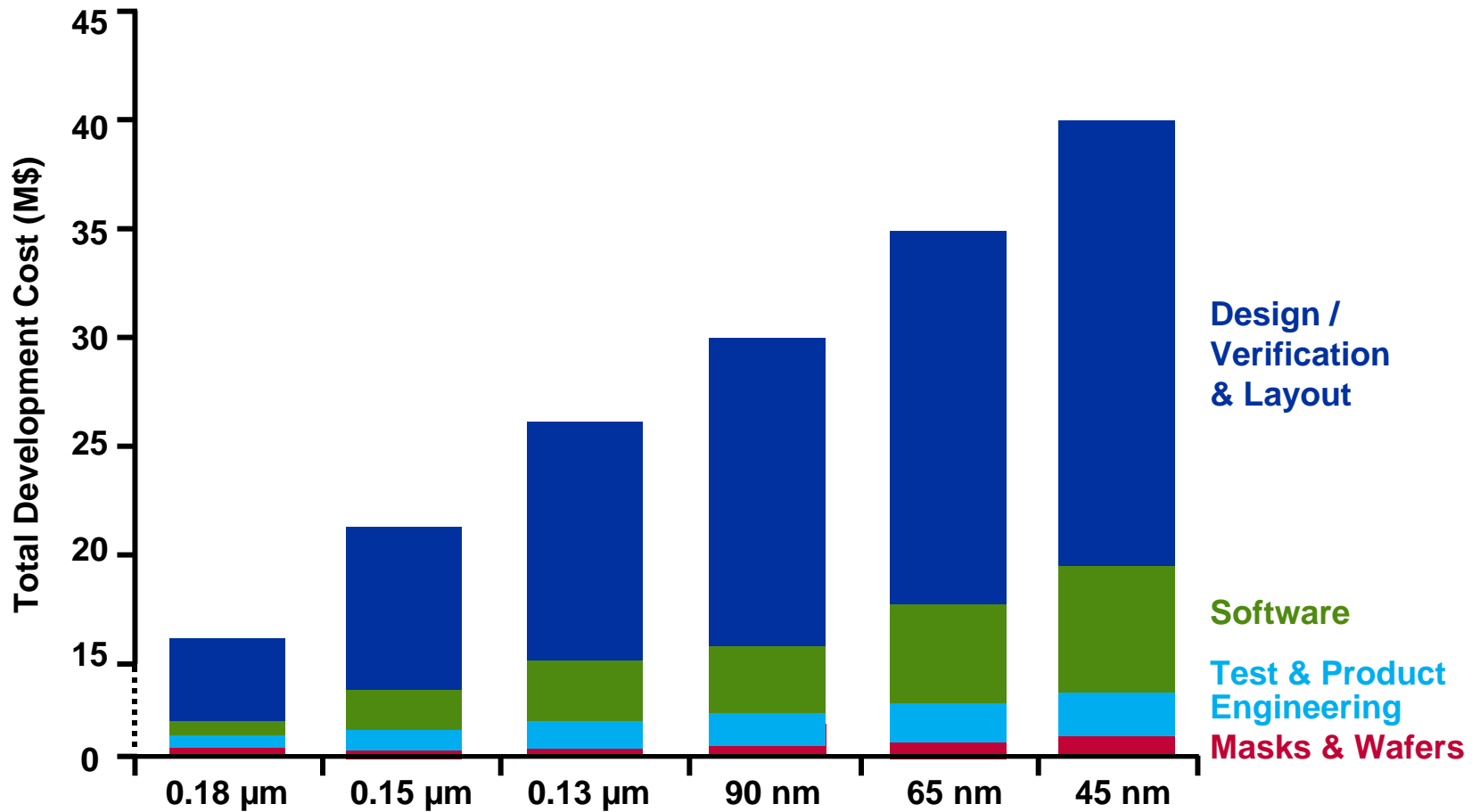
■ Problem:

- Create an application to process a lot of data quickly
- How?

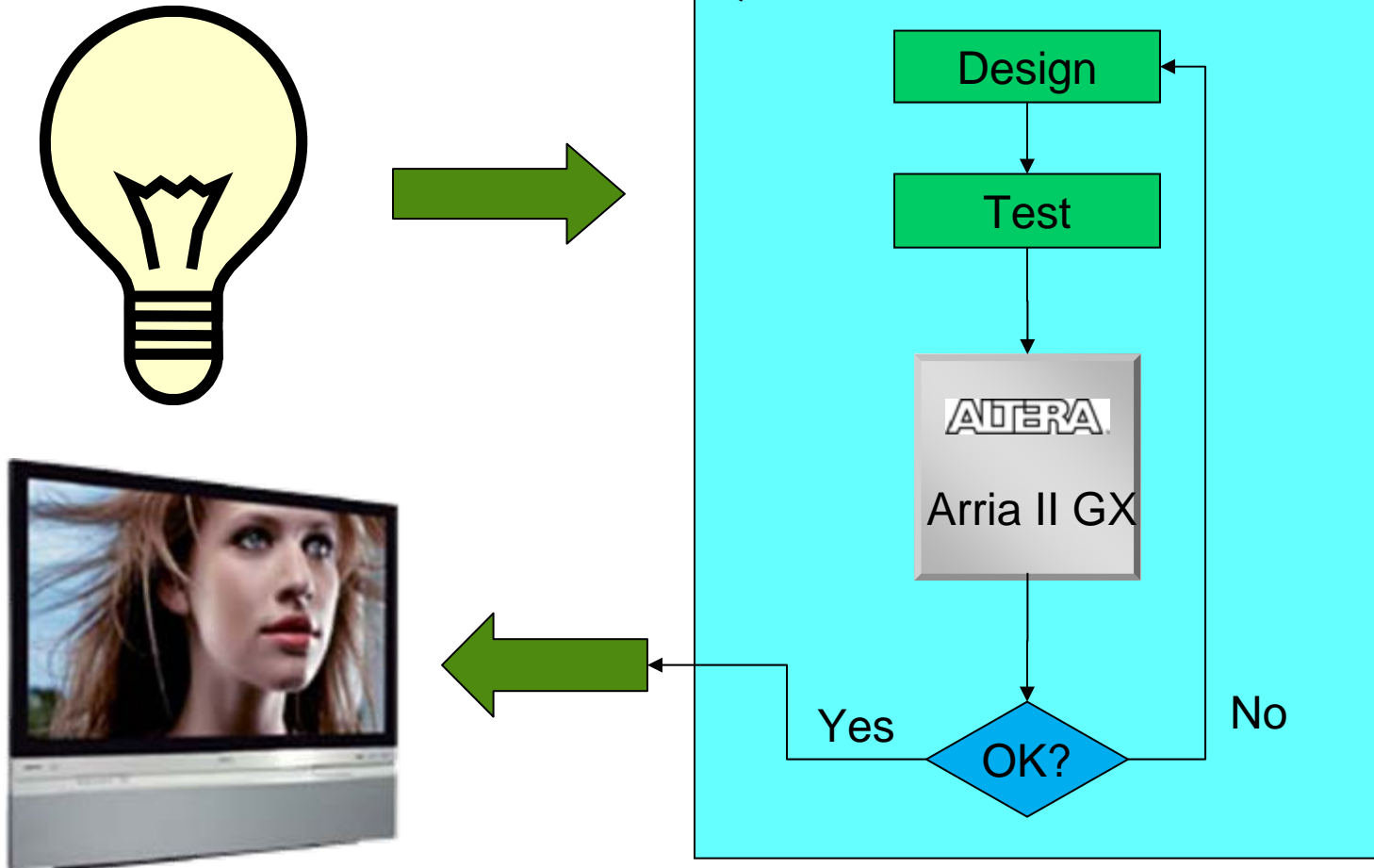
■ Alternatives:

- Processor?
 - **Easy** to write code, **low performance**, **power** hungry
- Gate Array (ASIC)?
 - Very high **performance**, **low power**, very **hard** to design, **expensive** to manufacture
- Field-Programmable Gate Array?
 - **no manufacturing** needed (just program), **easier** to design for than ASIC, high **performance**, **lower power** than a processor

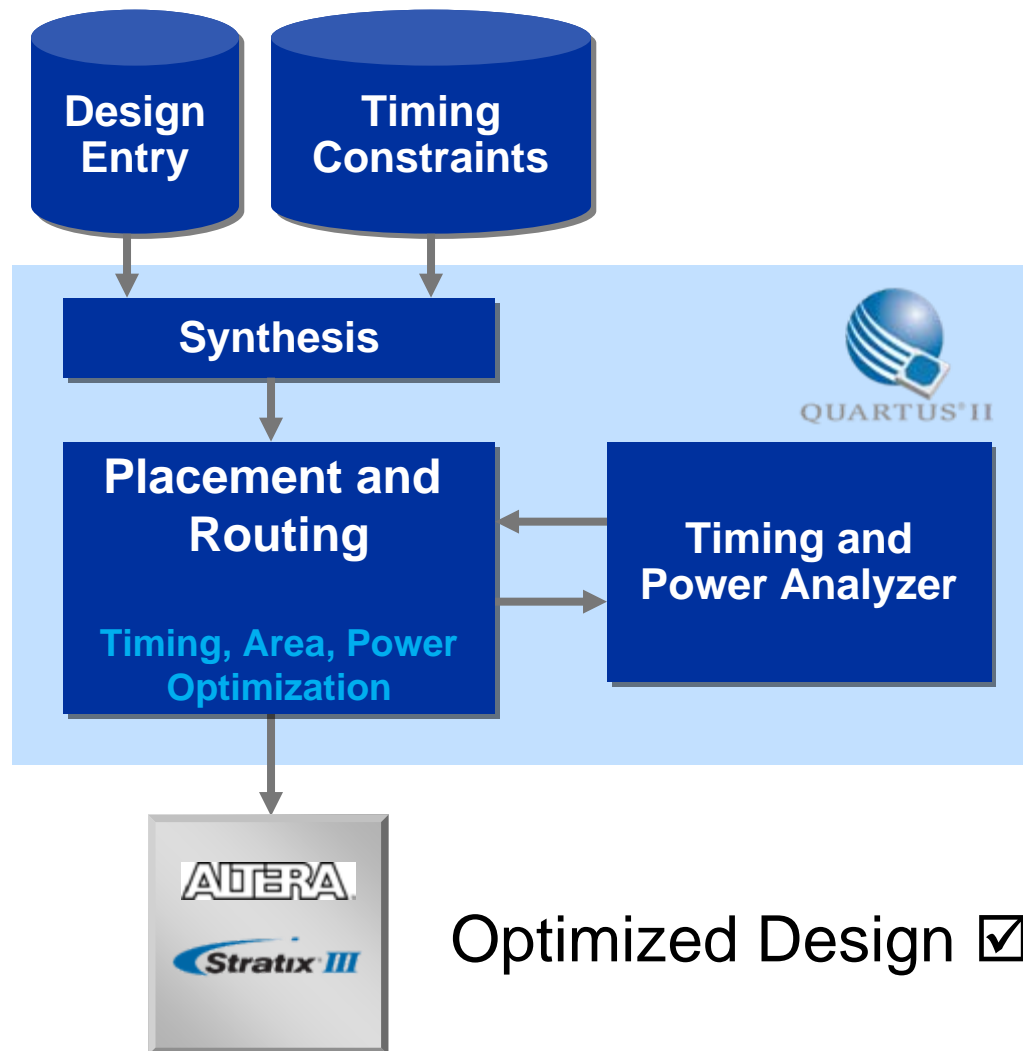
Increasing Chip Development Cost



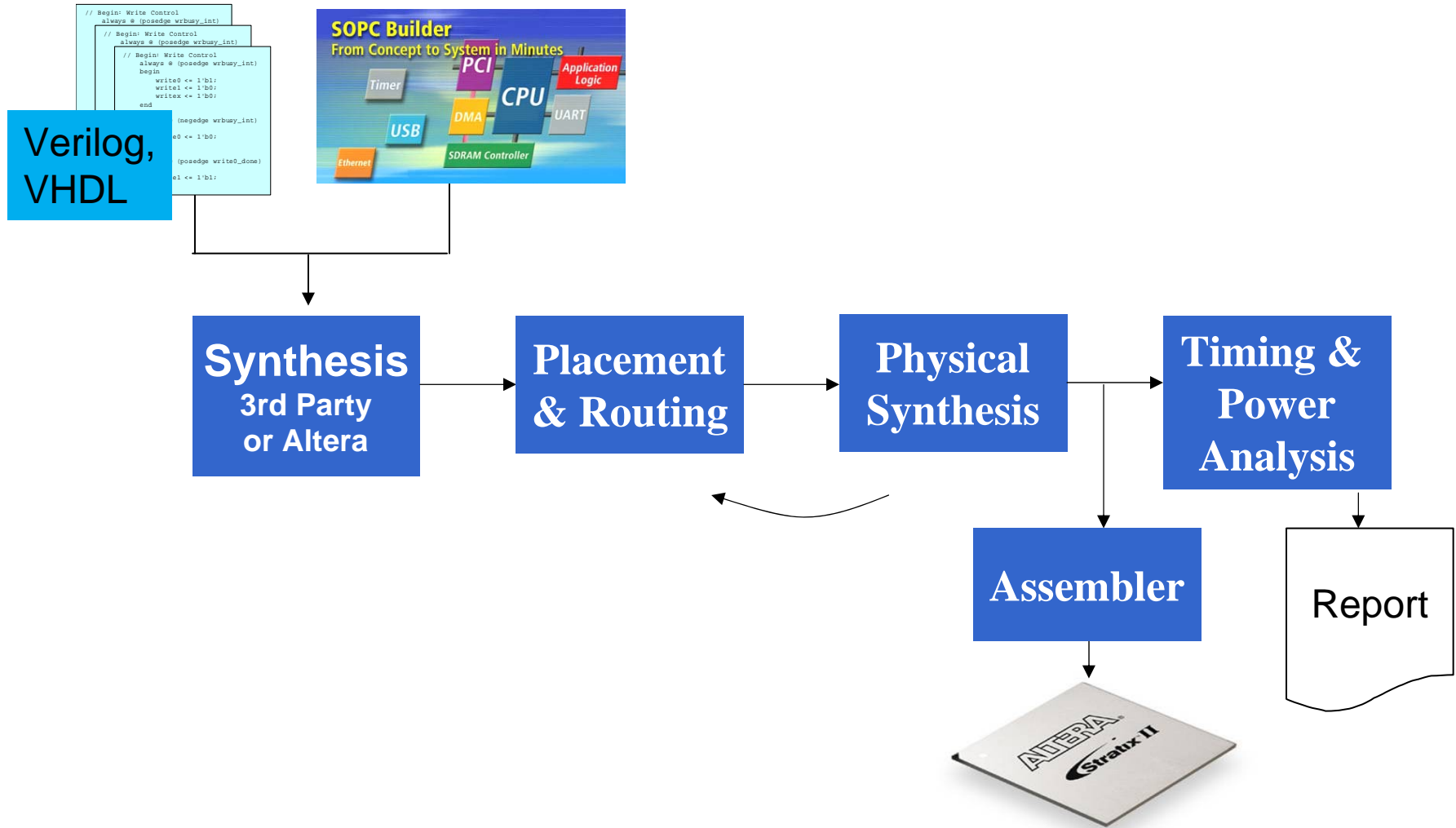
Getting a Product Out



Quartus II Design Flow

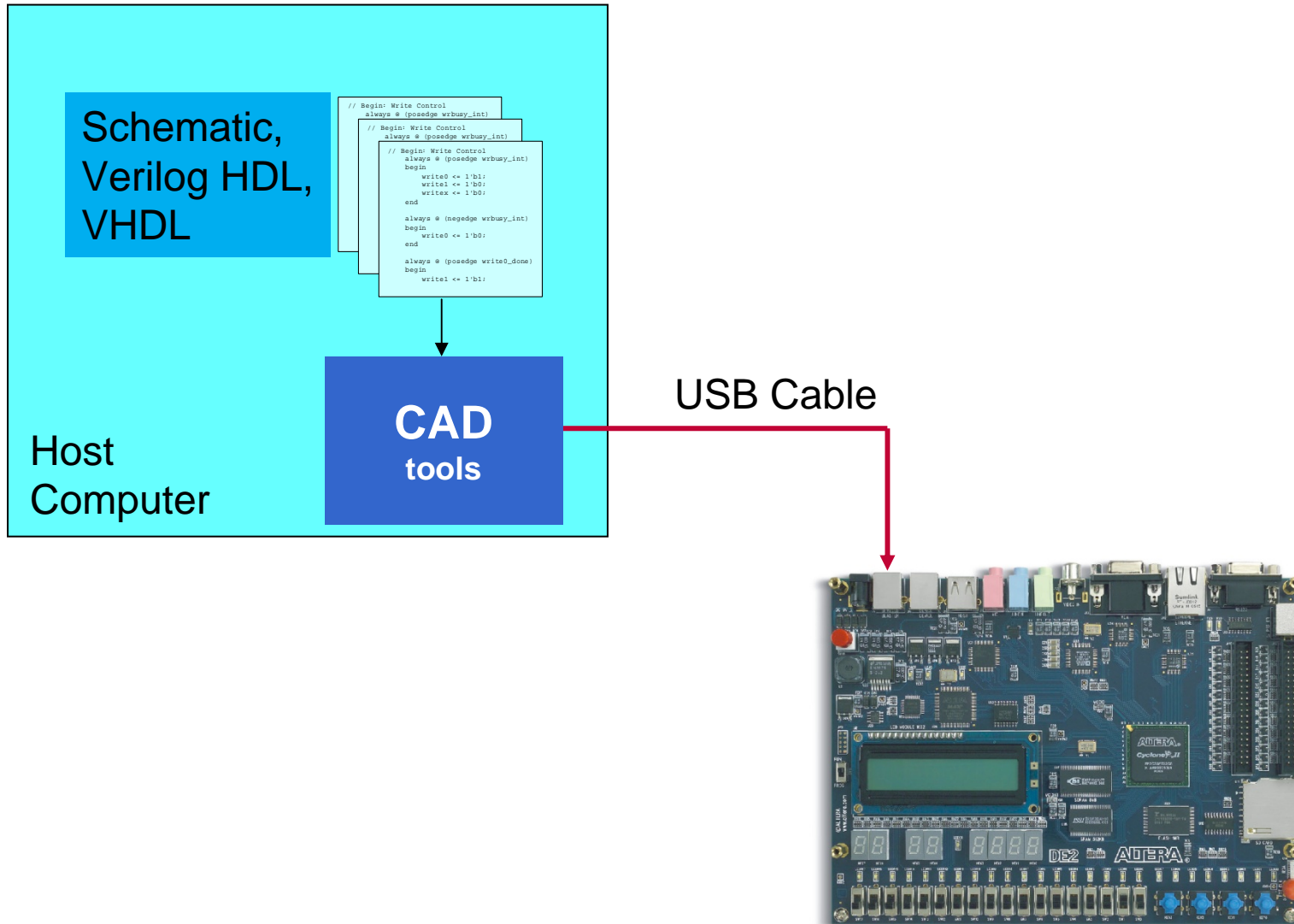


Design Flow: Quartus II



Teaching with Altera CAD Tools and Educational Platform

Altera Quartus II CAD Tools



How Students can use an FPGA

- Create design
 - Schematic, or Hardware Description Language (Verilog HDL, VHDL)
- Compile with FPGA CAD tools
- Simulate, debug
- Program into an FPGA lab board
- If a processor is included, write application code
- Test, debug

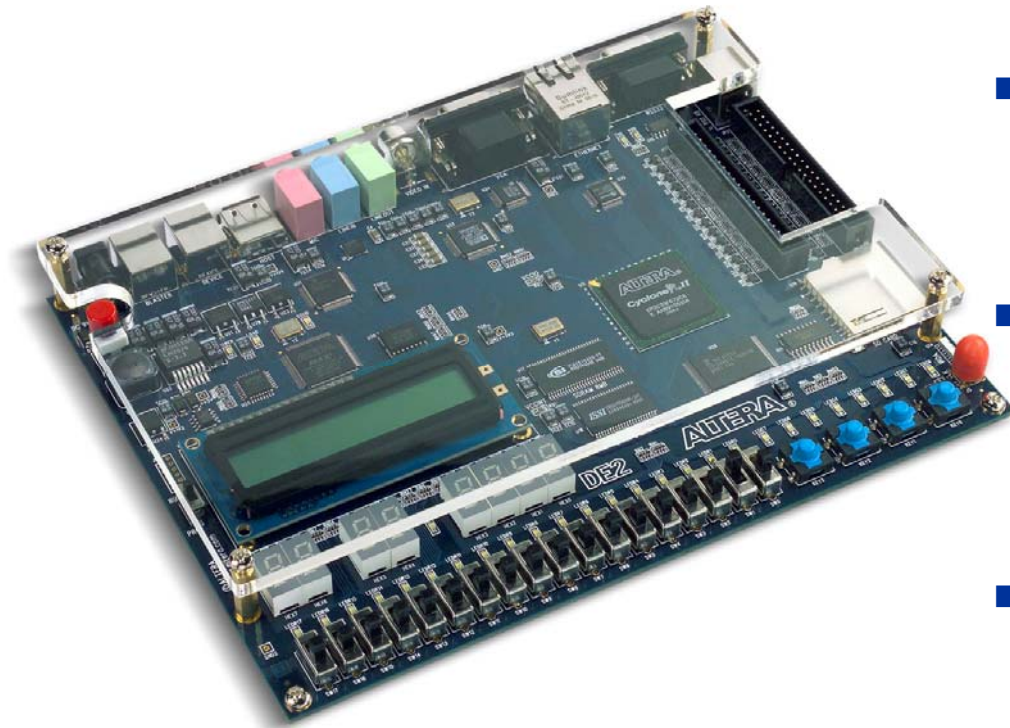
Teaching Materials Needed

- FPGA lab board
 - with all the right pedagogical features
- CAD tools – Quartus II
 - with tutorials to learn how to use the software
- Lab experiments
 - that fit into a modern curriculum
 - that cover the fundamentals and are fun/challenging/interesting

Developing Teaching Materials

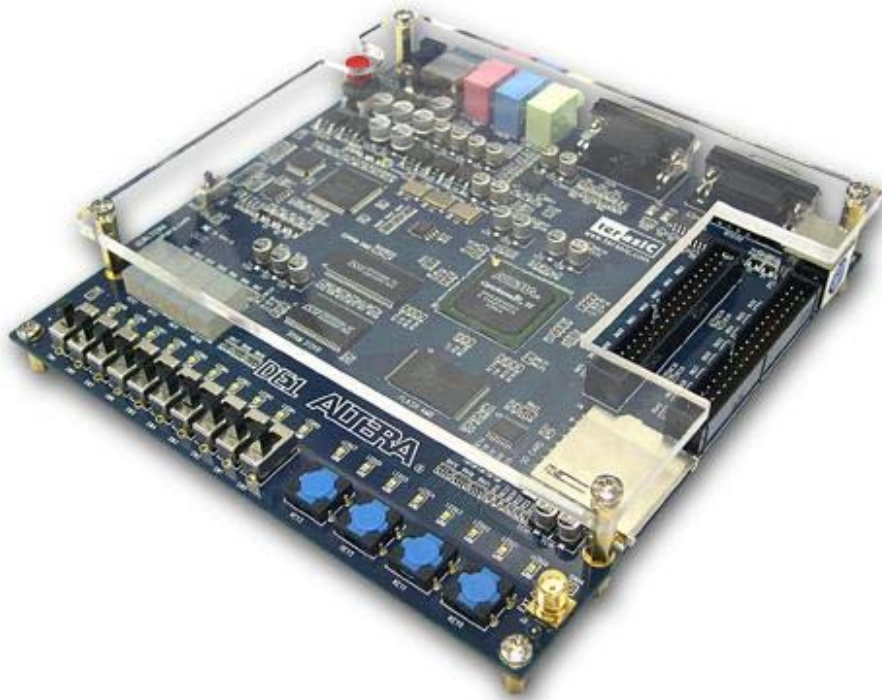
- Materials that help with teaching Digital Logic and Computer Organization
- What is our approach?
 - Examine the material covered in a course
 - Create exercises that enhance student learning
 - Creative and Interesting examples
 - Progressive learning
 - Build knowledge base with easy examples first
 - Extend examples to allow students to enhance their understanding
- Materials are designed for our Educational Boards (DE2/DE1)

DE2



- Ideal for undergraduate courses
 - Adopted in many Universities around the World
- Cyclone II 35/70 FPGA
 - 35k or 70k LUTs
 - More space than undergraduate students need for their projects
- User interface
 - Switches
 - 7-segment displays
 - LEDs
 - 16 x 2 Character LCD Display
- Advanced peripherals
 - Memory (SRAM, SDRAM, Flash)
 - USB
 - Video/Audio in/out
 - Ethernet
 - SD Card slot

DE1



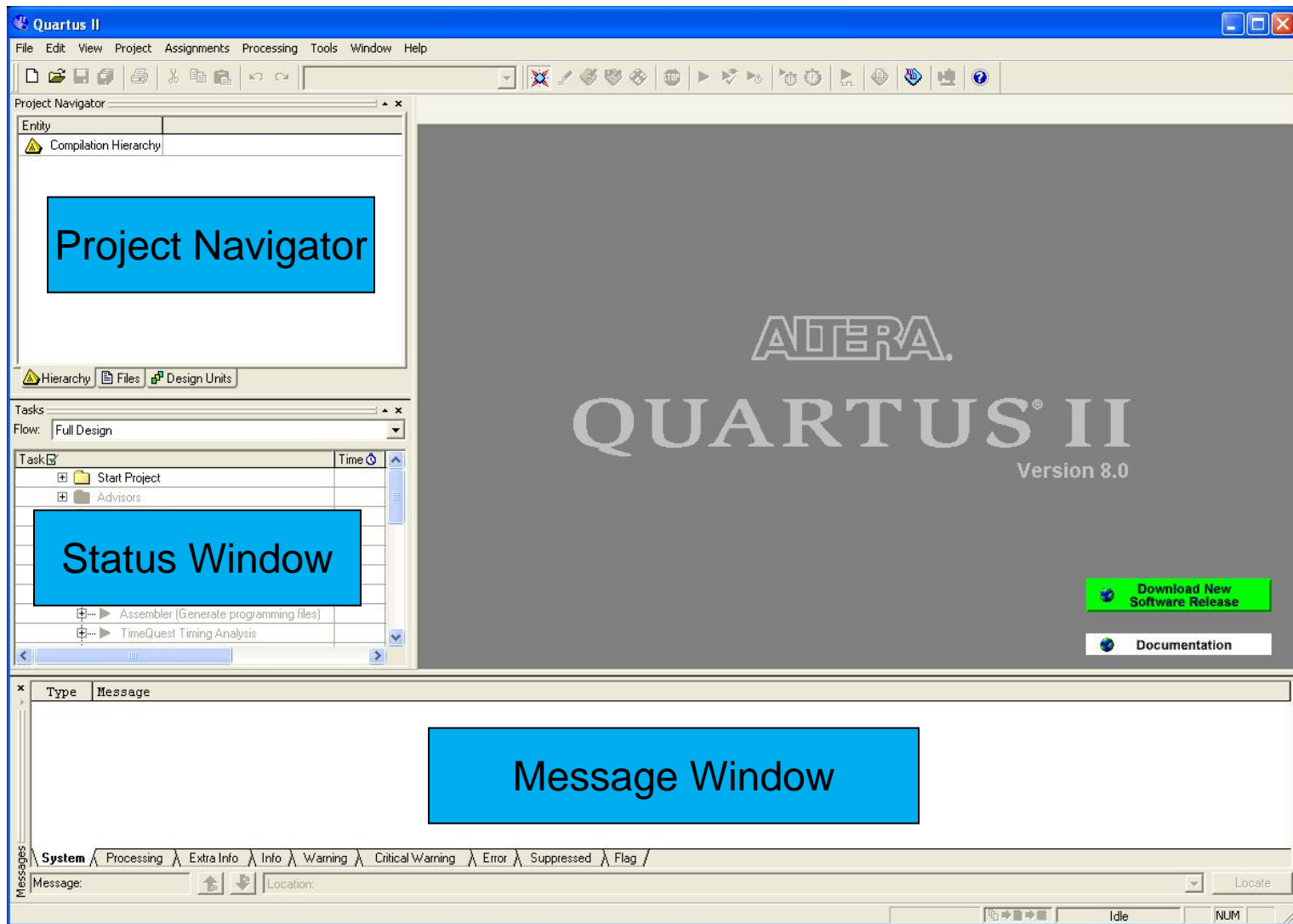
- Great for undergraduate courses
 - For student budget, so they can buy one for personal use
- Cyclone II 20 FPGA
 - 20k LUTs
- User interface
 - Switches
 - 7-segment displays
 - LEDs
- Advanced peripherals
 - memory
 - USB
 - Video/Audio in/out
 - Ethernet
 - SD Card slot

Digital Logic Exercises

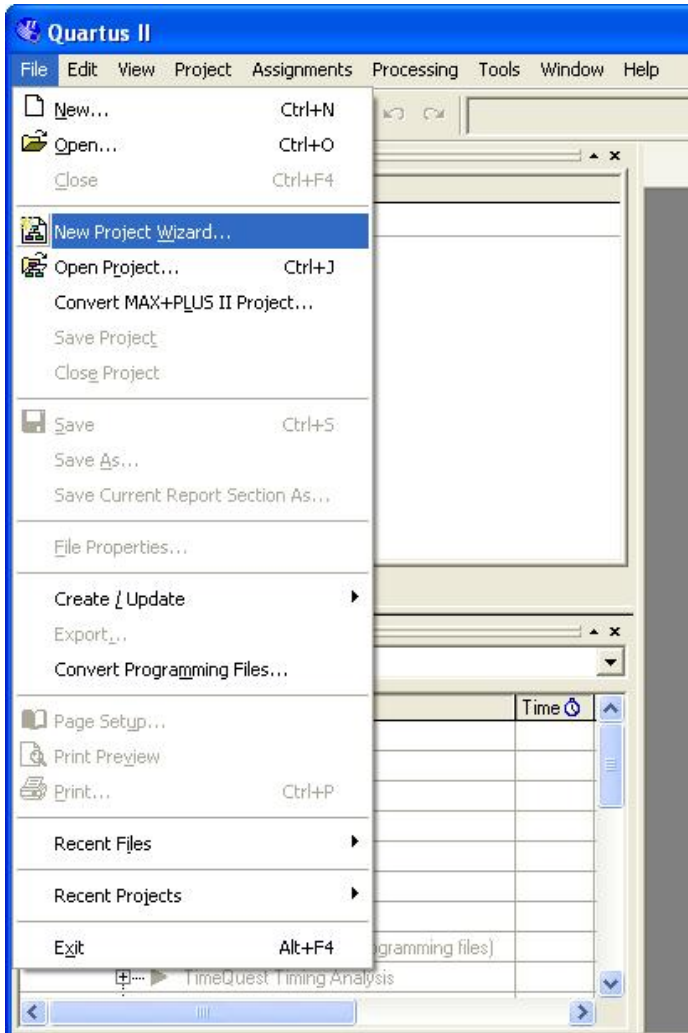
Outline

- Creating projects in Quartus II
- Targeting a project for a DE2 Board
- Downloading a circuit onto a DE2 board
- Compiling and debugging

Step 1: Start Quartus II



Step 2: Create a New Project



- Click **File** Menu
- Select **New Project Wizard**
- This will open a new window where project information can be specified

Project Name and Directory

New Project Wizard: Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?

D:\Course\Digital Logic\simple

What is the name of this project?

simple

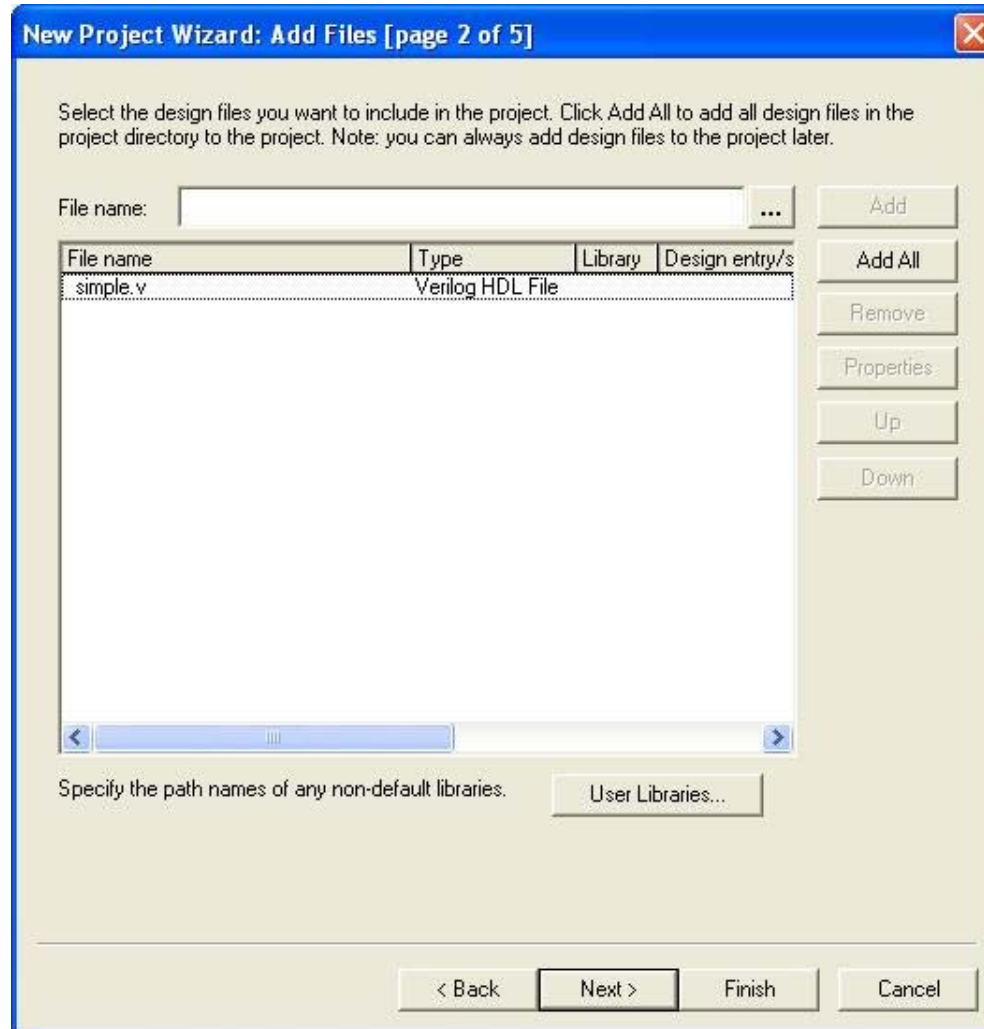
What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

simple

Use Existing Project Settings ...

< Back Next > Finish Cancel

Add Source Files to Project



Specify FPGA Device

- Select the FPGA device on the board
 - Cyclone II Family
 - For DE2 – EP2C35F672C6
 - For DE1 – EP2C20F484C7

New Project Wizard: Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.

Device family:
Family: Cyclone II
Devices: All

Target device:
 Auto device selected by the Fitter
 Specific device selected in 'Available devices' list

Show in 'Available device' list:
Package: Any
Pin count: Any
Speed grade: Any
 Show advanced devices
 HardCopy compatible only

Available devices:

Name	Core v...	LEs	User I/...	Memor...	Embed...	PLL
EP2C20F256C7	1.2V	18752	152	239616	52	4
EP2C20F256C8	1.2V	18752	152	239616	52	4
EP2C20F256I8	1.2V	18752	152	239616	52	4
EP2C20F484C6	1.2V	18752	315	239616	52	4
EP2C20F484C7	1.2V	18752	315	239616	52	4
EP2C20F484C8	1.2V	18752	315	239616	52	4
EP2C20F484I8	1.2V	18752	315	239616	52	4
EP2C20Q240C8	1.2V	18752	142	239616	52	4
EP2C20F484C6	1.2V	18752	315	239616	52	4

Companion device:
HardCopy:
 Limit DSP & RAM to HardCopy device resources

< Back Next > Finish Cancel

Additional EDA Tools

- Specify Tools, in addition to Quartus II, that you will use
- These are unnecessary for small student designs
 - Leave all entries as <None>
 - Press Next

New Project Wizard: EDA Tool Settings [page 4 of 5]

Specify the other EDA tools -- in addition to the Quartus II software -- used with the project.

Design Entry/Synthesis

Tool name: <None>

Format:

Run this tool automatically to synthesize the current design

Simulation

Tool name: <None>

Format:

Run gate-level simulation automatically after compilation

Timing Analysis

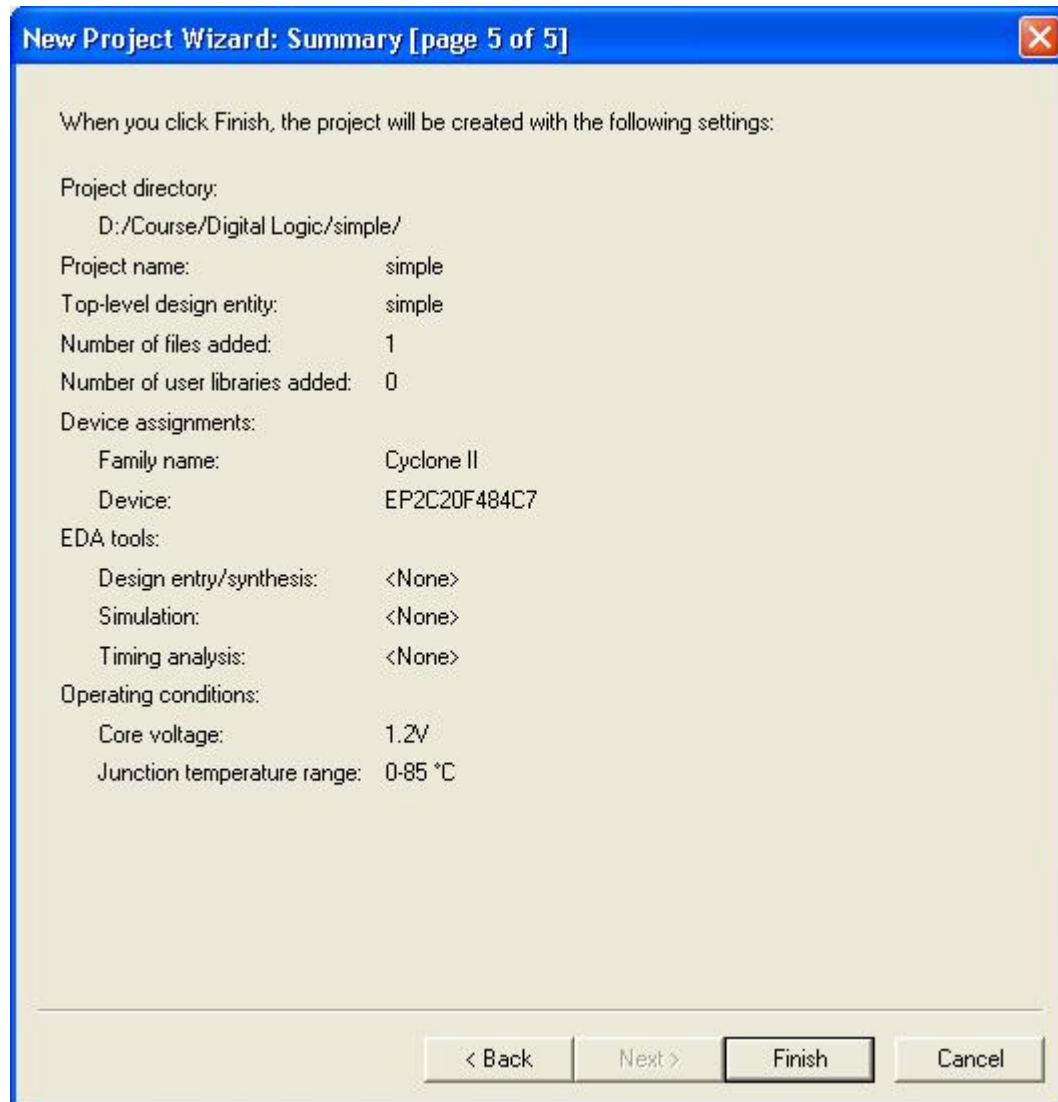
Tool name: <None>

Format:

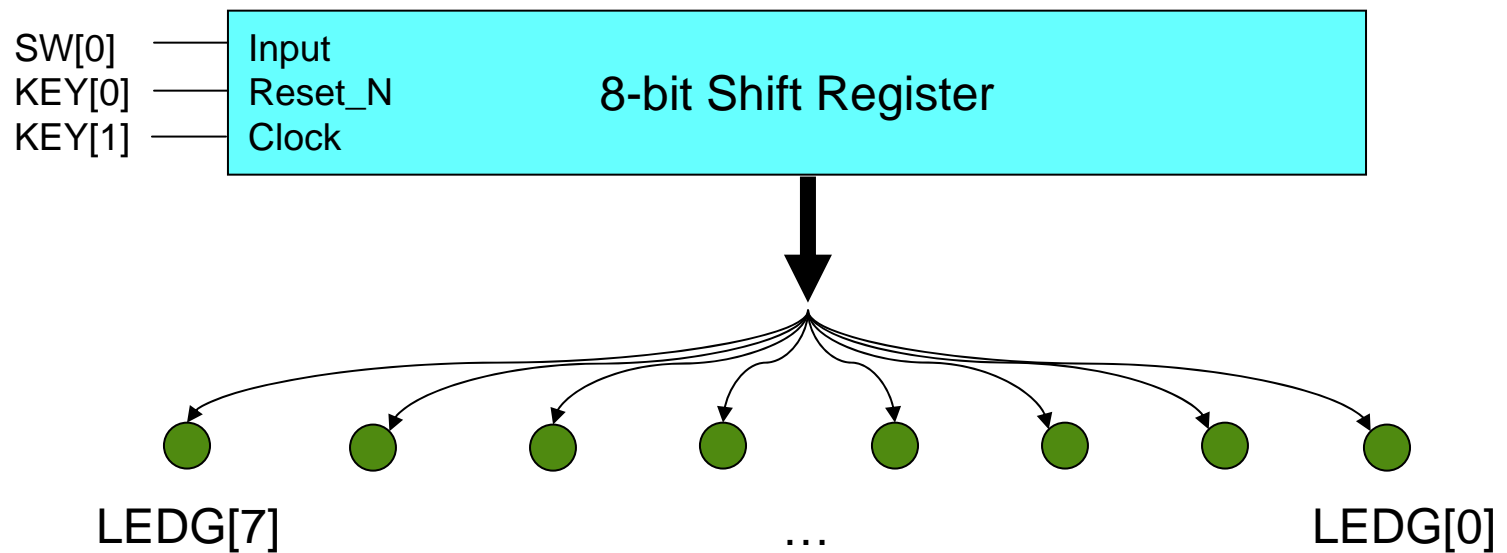
Run this tool automatically after compilation

< Back Next > Finish Cancel

New Project Summary



Simple Project



Step 3: Open Source File

The screenshot displays the Quartus II IDE interface. The main window shows a Verilog source file named 'simple.v' with the following code:

```
1 module simple(SW, KEY, LEDG);
2   /* Declare Inputs/Outputs of the module */
3   input  [1:0] KEY;
4   input  [0:0] SW;
5   output [7:0] LEDG;
6
7   /* Declare shift register bits and local wires */
8   reg    [7:0] shift_reg_bits;
9   wire   clock, reset_n;
10
11  /* Connect inputs to wires. This makes the code easier to read. */
12  assign clock = KEY[0];
13  assign reset_n = KEY[1];
14
15  /* Code describing the behaviour of the shift register. */
16  always@(posedge clock or negedge reset_n)
17  begin
18      if (~reset_n)
19          shift_reg_bits <= 8'd0;
20      else
21      begin
22          shift_reg_bits[7:1] <= shift_reg_bits[6:0];
23          shift_reg_bits[0] <= SW[0];
24      end
25  end
26
27  /* Assign Circuit Outputs */
28  assign LEDG = shift_reg_bits;
29 endmodule
```

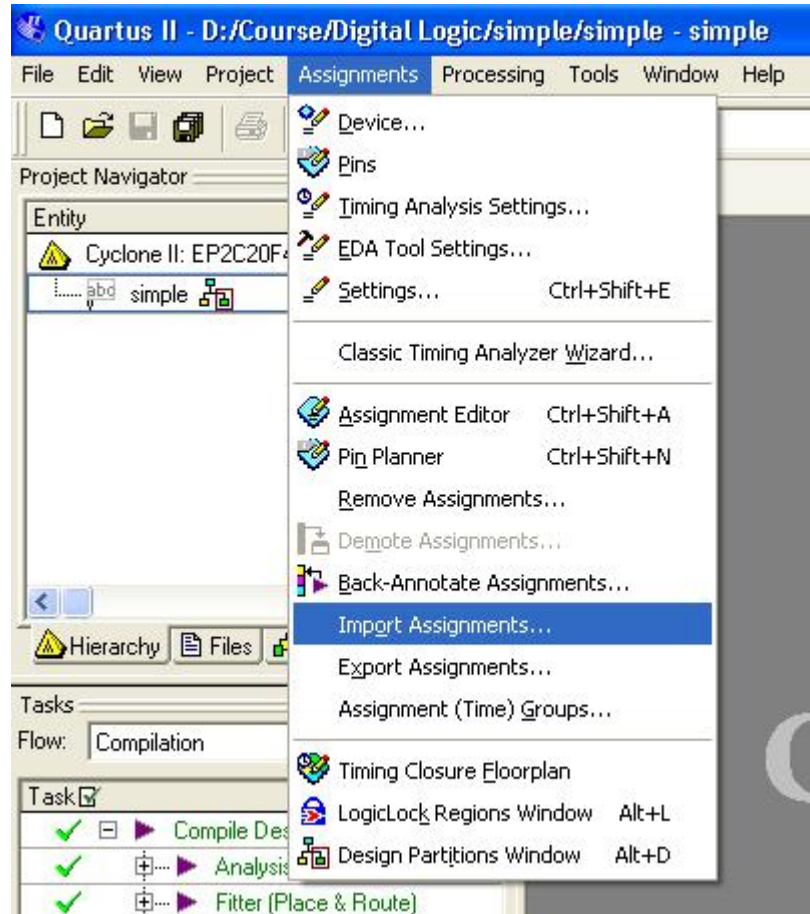
The Project Navigator on the left shows the project hierarchy for 'simple.v' on a Cyclone II: EP2C20F484C7 device. The Tasks window shows the compilation process is complete. The Messages window at the bottom displays the following message:

Info: Quartus II Full Compilation was successful. 0 errors, 429 warnings

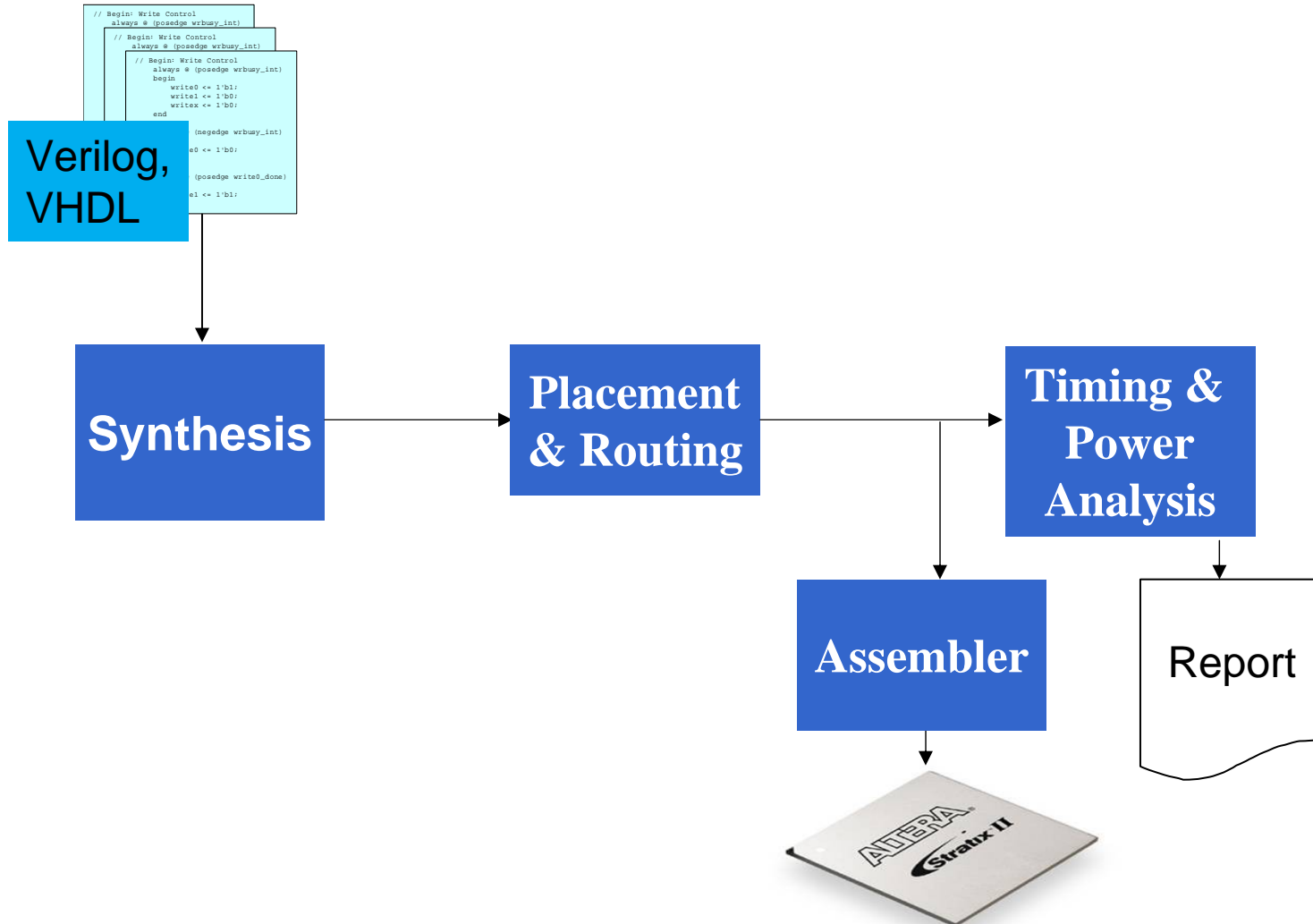
The status bar at the bottom indicates 'Ln 7, Col 30' and 'Idle'.

Step 4: Assign Pins to connect switches/lights to inputs and outputs of your circuit

- Click **Assignments**, then **Import Assignments...**
- Import file
 - DE1_pin_assignments.csv
- Imports locations for predefined port names, such as SW, LEDG, KEY, and others
 - Can be done manually for custom port names



Step 5: Compile Design



Step 6: Examine Compilation Report

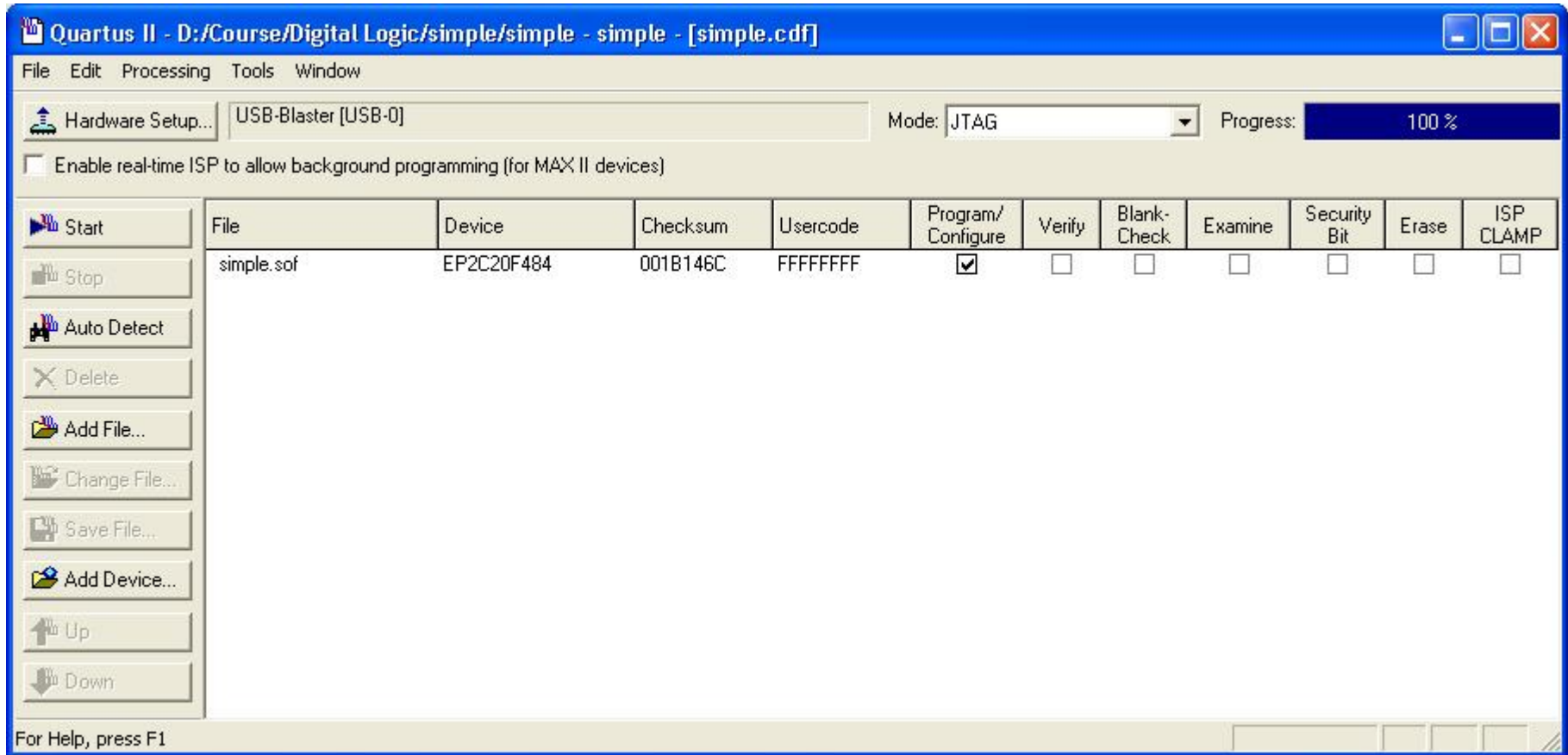
The screenshot shows the Quartus II interface with the 'Compilation Report - Flow Summary' window open. The window is divided into three main sections:

- Project Navigator:** Shows the project hierarchy with 'simple' selected under 'Cyclone II: EP2C20F484C7'.
- Compilation Report - Flow Summary:**
 - Flow Summary:**

Flow Status	Successful - Mon Jul 20 15:03:23 2009
Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Web Edition
Revision Name	simple
Top-level Entity Name	simple
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Met timing requirements	Yes
Total logic elements	8 / 18,752 (< 1 %)
Total combinational functions	0 / 18,752 (0 %)
Dedicated logic registers	8 / 18,752 (< 1 %)
Total registers	8
Total pins	11 / 315 (3 %)
Total virtual pins	0
Total memory bits	0 / 239,616 (0 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
Total PLLs	0 / 4 (0 %)
- Messages:**

Type	Message
Info	Info: Configuring device index 1
Info	Info: Device 1 contains JTAG ID code 0x020B30DD
Info	Info: Configuration succeeded -- 1 device(s) configured
Info	Info: Successfully performed operation(s)
Info	Info: Ended Programmer operation at Mon Jul 20 15:03:33 2009

Step 7: Program the DE1 Board



Quartus II - D:/Course/Digital Logic/simple/simple - simple - [simple.cdf]

File Edit Processing Tools Window

Hardware Setup... USB-Blaster [USB-0] Mode: JTAG Progress: 100 %

Enable real-time ISP to allow background programming (for MAX II devices)

	File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Examine	Security Bit	Erase	ISP CLAMP
Start	simple.sof	EP2C20F484	001B146C	FFFFFFF	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Stop

Auto Detect

Delete

Add File...

Change File...

Save File...

Add Device...

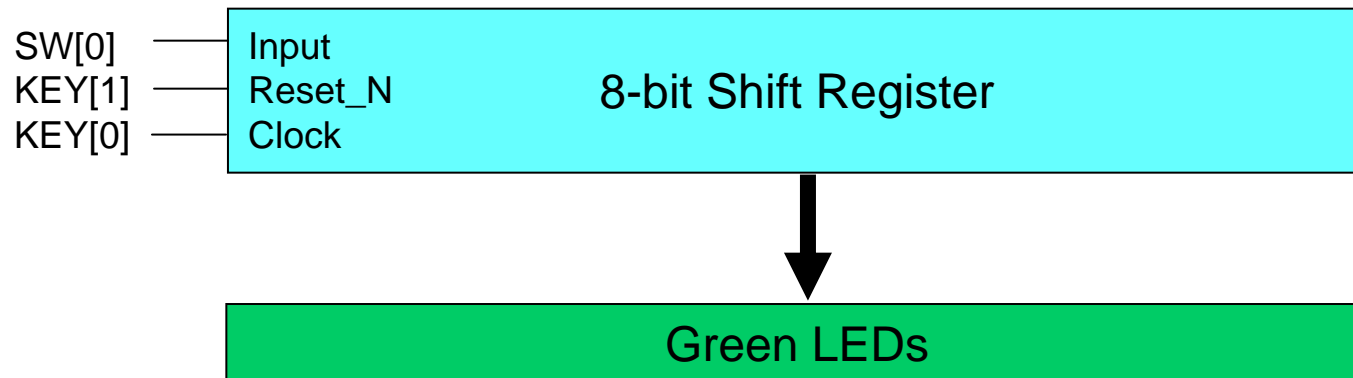
Up

Down

For Help, press F1

Step 8: See your design work on the board

- Reset the shift register using KEY[1]
- SW[0] is the input to the shift register
- Press the KEY[0] to clock the circuit



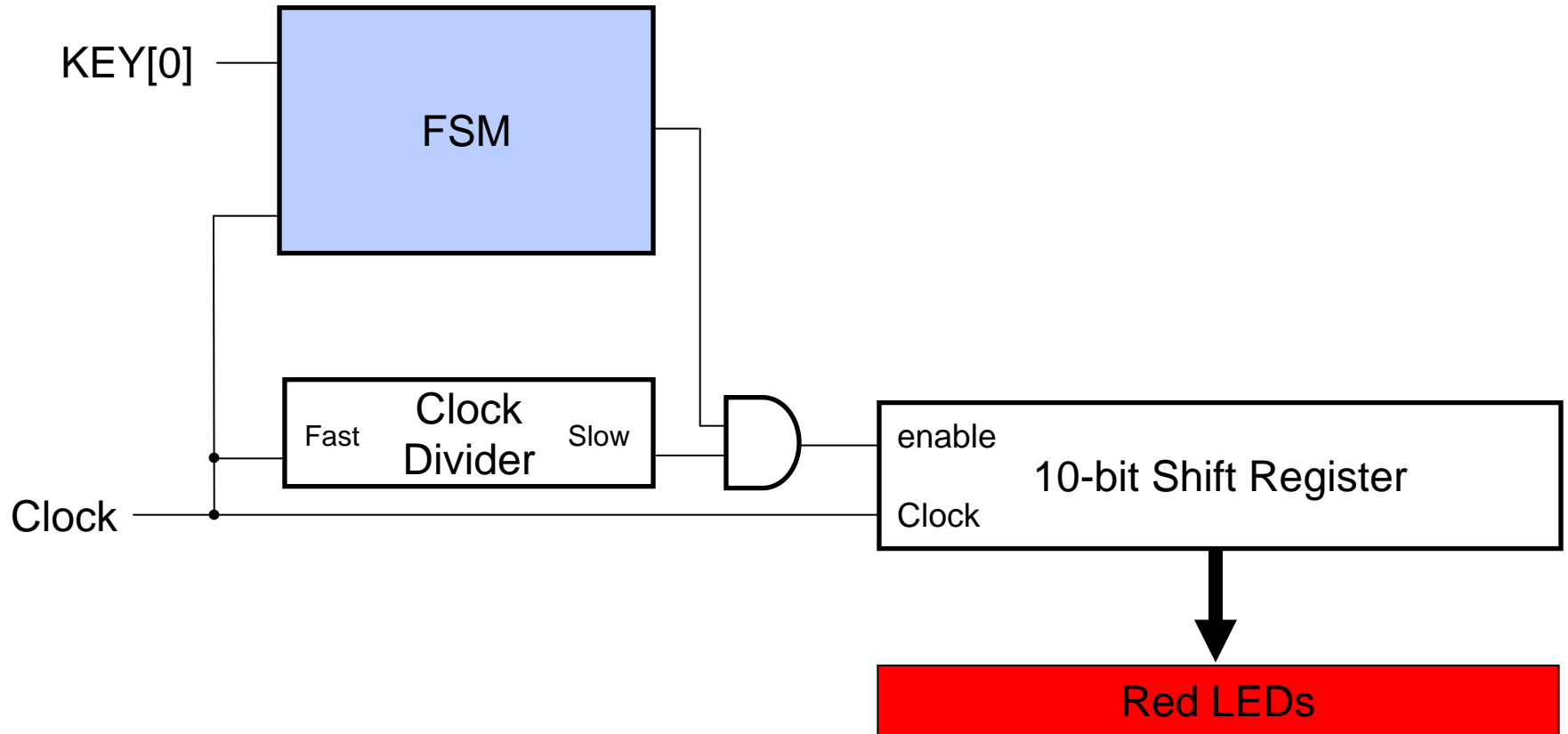
Next Example

- Open the **Digital Logic** Directory
- Go into **stopper** subdirectory
- Double-click on stopper.qpf to open an existing project

Example 2: Stopper

- Shift the contents of a register once every second
 - The circuit is clocked using a 50MHz clock
- Press KEY[0] to start or stop the shift register
 - FSM examines if the key was pressed
- Purpose:
 - Look at FSM implementation in Quartus II
 - Finite State Machine Viewer

Circuit Diagram



Step 1: Open Stopper Project

The screenshot displays the Quartus II IDE with the 'stopper.v' file open. The Project Navigator on the left shows the project hierarchy for 'Cyclone II: EP2C20F484C7'. The Tasks window shows the 'Compilation' flow selected. The main editor window contains the following Verilog code:

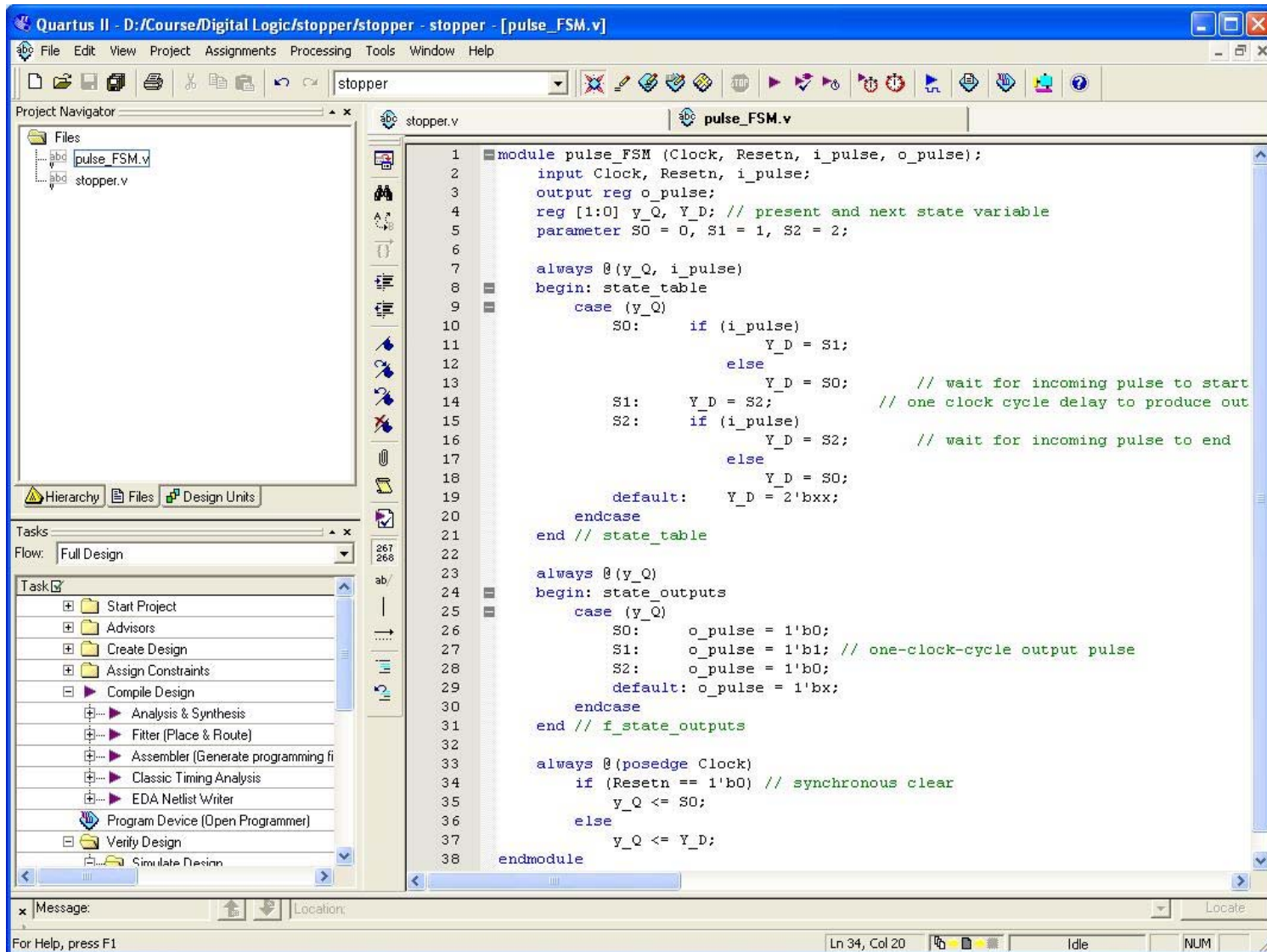
```
1 module stopper(CLOCK_50, KEY, LEDR);
2     input        CLOCK_50;
3     input  [1:0]  KEY;
4     output [9:0]  LEDR;
5
6     reg [9:0]    shift_reg;
7     reg [20:0]   counter;
8     reg         go;
9     wire        reset_n, enable, toggle_go;
10
11     /* Glue logic */
12     assign reset_n = KEY[1];
13     assign enable = &counter;
14
15     /* Control FSM */
16     pulse_FSM FSM(.Clock(CLOCK_50), .Resetn(reset_n), .i_pulse(~KEY[0]),
17                  .o_pulse(toggle_go));
18
19     always @(posedge CLOCK_50 or negedge reset_n)
20     begin
21         if (~reset_n)
22         begin
23             go <= 1'b0;
24             counter <= 21'd0;
25             shift_reg <= 10'd1;
26         end
27         else
28         begin
29             counter <= counter + 1'b1;
```

Step 2: Compile and Program

- Compile the design
- Program the design onto the DE2 board

- How does it work?
 - Press KEY[0] to start/stop the circuit
 - Press KEY[1] to reset the circuit

Step 3: Examine the FSM Source Code



The screenshot displays the Quartus II IDE interface. The main window shows the source code for a Verilog module named `pulse_FSM.v`. The code defines a finite state machine with three states (S0, S1, S2) and an output pulse. The state transitions are as follows:

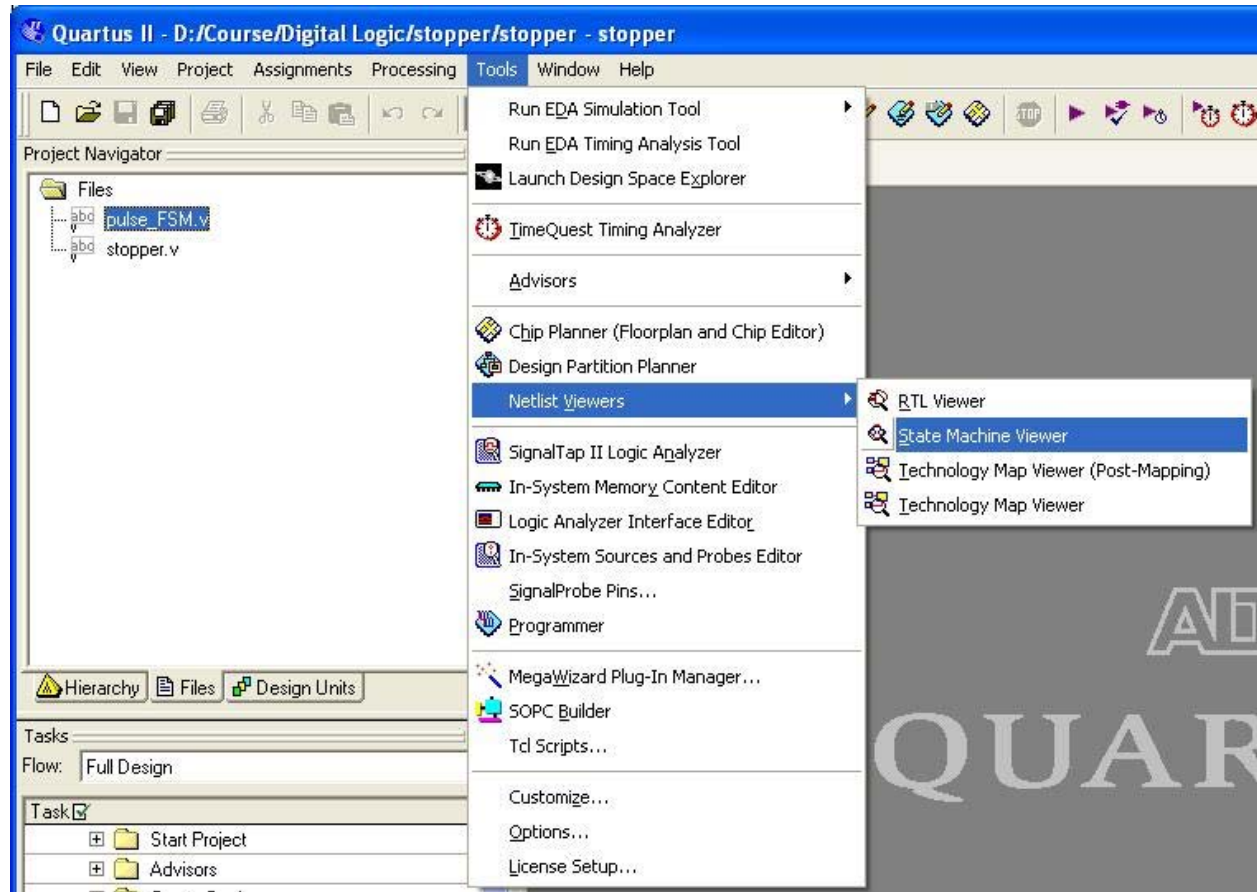
- S0:** If `i_pulse` is high, transition to `Y_D = S1`; otherwise, `Y_D = S0`.
- S1:** `Y_D = S2`.
- S2:** If `i_pulse` is high, transition to `Y_D = S2`; otherwise, `Y_D = S0`.

The output `o_pulse` is set to `1'b0` for S0, `1'b1` for S1, and `1'b0` for S2. A synchronous clear is implemented where `Resetn == 1'b0` sets `y_Q` to `S0`.

```
1 module pulse_FSM (Clock, Resetn, i_pulse, o_pulse);
2   input Clock, Resetn, i_pulse;
3   output reg o_pulse;
4   reg [1:0] y_Q, Y_D; // present and next state variable
5   parameter S0 = 0, S1 = 1, S2 = 2;
6
7   always @(y_Q, i_pulse)
8   begin: state_table
9     case (y_Q)
10      S0:    if (i_pulse)
11              Y_D = S1;
12            else
13              Y_D = S0;    // wait for incoming pulse to start
14      S1:    Y_D = S2;    // one clock cycle delay to produce out
15      S2:    if (i_pulse)
16              Y_D = S2;    // wait for incoming pulse to end
17            else
18              Y_D = S0;
19      default: Y_D = 2'bxx;
20    endcase
21  end // state_table
22
23  always @(y_Q)
24  begin: state_outputs
25    case (y_Q)
26      S0:    o_pulse = 1'b0;
27      S1:    o_pulse = 1'b1; // one-clock-cycle output pulse
28      S2:    o_pulse = 1'b0;
29      default: o_pulse = 1'b0;
30    endcase
31  end // f_state_outputs
32
33  always @(posedge Clock)
34    if (Resetn == 1'b0) // synchronous clear
35      y_Q <= S0;
36    else
37      y_Q <= Y_D;
38  endmodule
```

Step 4: FSM Viewer

- Open the FSM Viewer
 - Click **Tools**
 - Expand **Netlist Viewers**
 - Click **State Machine Viewer**



Examine State Machine

Quartus II - D:/Course/Digital Logic/stopper/stopper - stopper - [State Machine Viewer | pulse_FSM:FSM | y_Q]

File Edit View Project Assignments Processing Tools Window Help

stopper

Project Navigator

- Files
 - abd pulse_FSM.v
 - abd stopper.v

stopper.v | pulse_FSM.v | Compilation Report - Flow Sum... | State Machine Viewer I ...

State Machine: |stopperpulse_FSM:FSM|y_Q

```

    graph LR
      Start(( )) --> S0((S0))
      S0 -- "i_pulse" --> S0
      S0 -- "i_pulse" --> S1((S1))
      S1 -- "!Resetn" --> S0
      S1 -- "Resetn" --> S2((S2))
      S2 -- "!i_pulse" --> S0
      S2 -- "i_pulse" --> S2
    
```

	Source State	Destination State	Condition
1	S0	S0	(!i_pulse) + (!_pulse)!(Resetn)
2	S0	S1	(i_pulse)(Resetn)
3	S1	S0	!(Resetn)
4	S1	S2	(Resetn)
5	S2	S0	!(i_pulse) + (!_pulse)!(Resetn)
6	S2	S2	(i_pulse)(Resetn)

Transitions / Encoding /

Message: 0 of 614 Location: Locate

For Help, press F1

Idle NUM

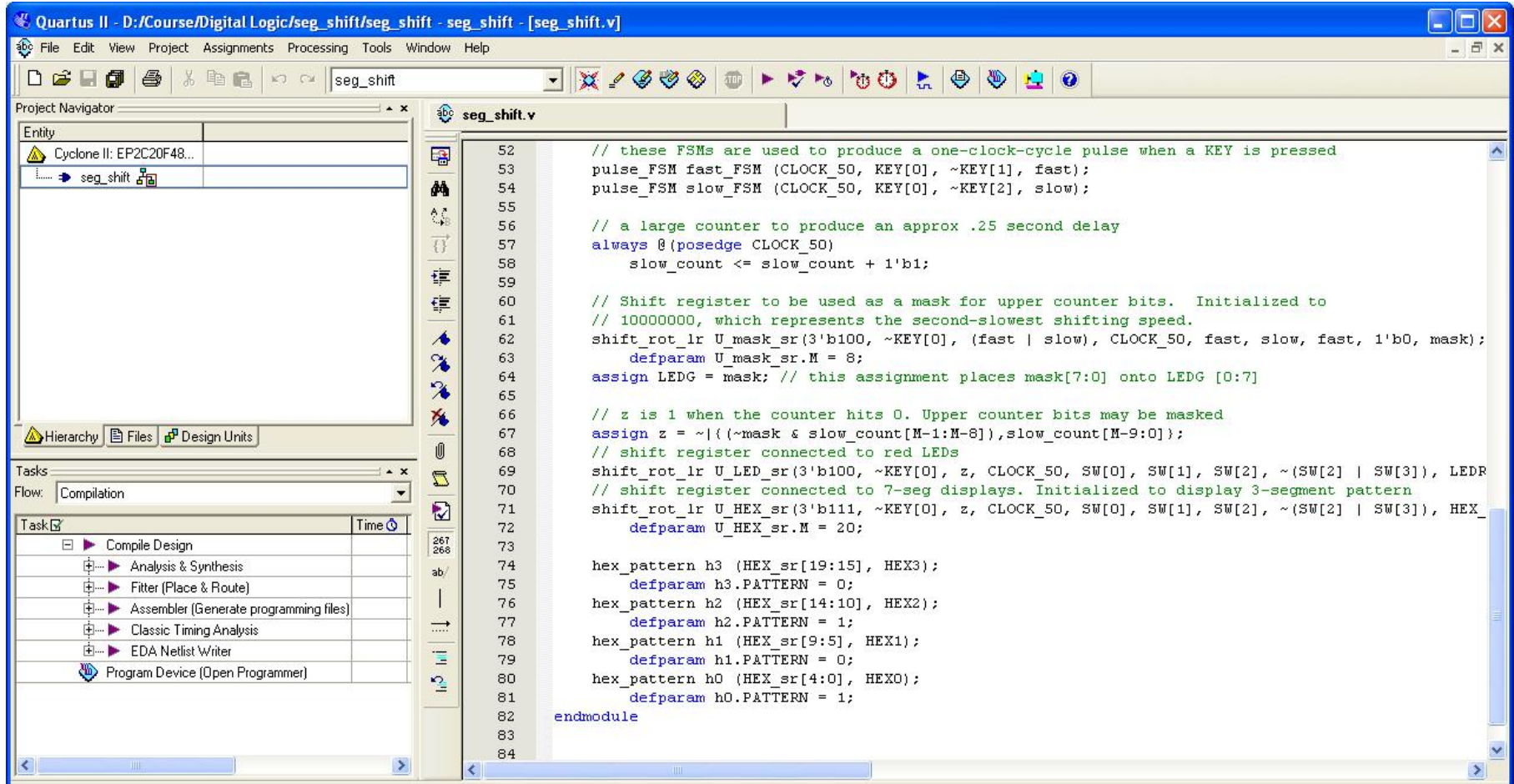
Next Example

- Open the **Digital Logic** Directory
- Go into **seg_shift** subdirectory
- Double-click on **seg_shift.qpf** to open an existing project

Example 3: Segment Shifter

- Combine a few shift registers and a few instances of the FSM from Example 2
- Circuit will scroll lights left/right, both on lights and on 7-Segment displays
- Purpose:
 - Look at building larger circuits
 - RTL Viewer

Step 1: Open seg_shift Project



The screenshot shows the Quartus II software interface. The title bar indicates the project is "seg_shift" located at "D:/Course/Digital Logic/seg_shift/seg_shift - seg_shift - [seg_shift.v]". The Project Navigator on the left shows the project hierarchy with "seg_shift" selected. The Tasks pane shows the compilation process, with "Compile Design" selected. The main editor displays the Verilog code for the seg_shift module, which includes comments and code for pulse FSMs, a counter, a shift register, and LED patterns.

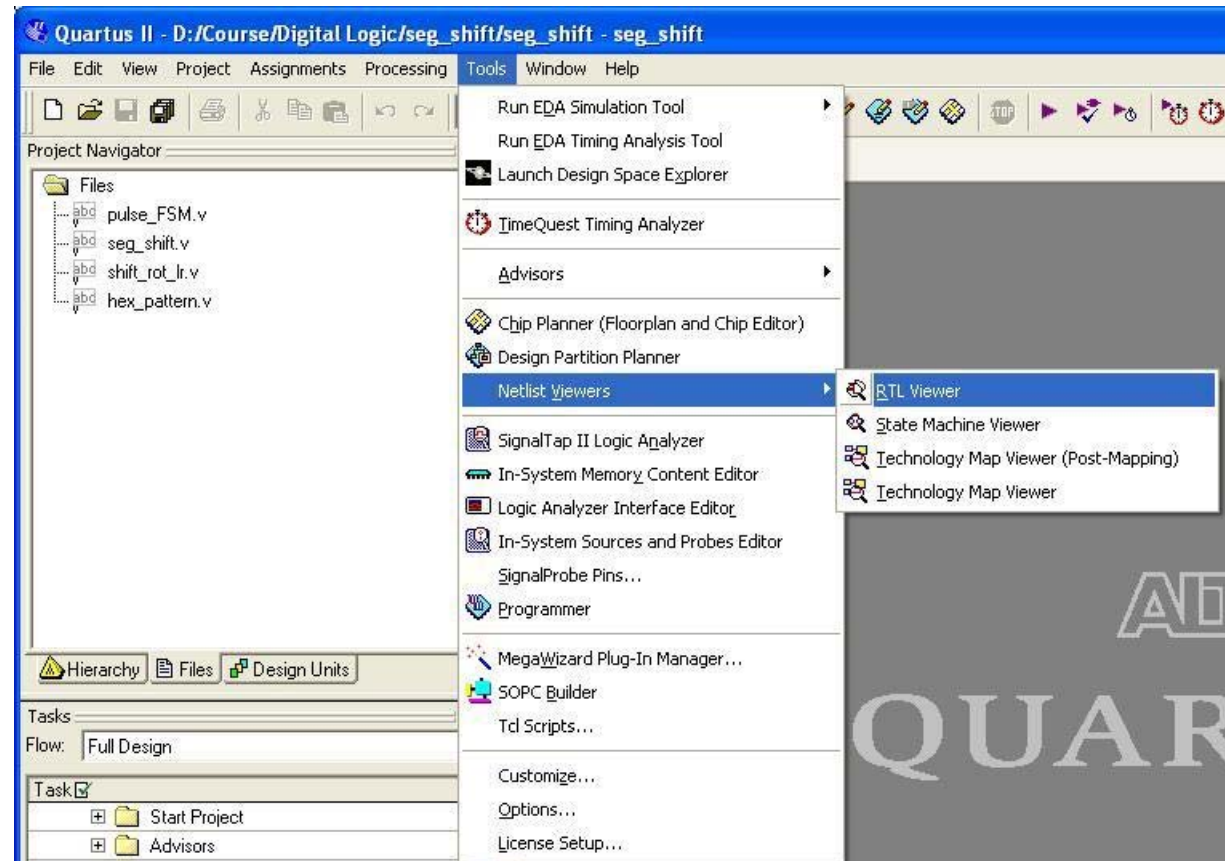
```
52 // these FSMs are used to produce a one-clock-cycle pulse when a KEY is pressed
53 pulse_FSM fast_FSM (CLOCK_50, KEY[0], ~KEY[1], fast);
54 pulse_FSM slow_FSM (CLOCK_50, KEY[0], ~KEY[2], slow);
55
56 // a large counter to produce an approx .25 second delay
57 always @(posedge CLOCK_50)
58     slow_count <= slow_count + 1'b1;
59
60 // Shift register to be used as a mask for upper counter bits. Initialized to
61 // 10000000, which represents the second-slowest shifting speed.
62 shift_rot_lr U_mask_sr(3'b100, ~KEY[0], (fast | slow), CLOCK_50, fast, slow, fast, 1'b0, mask);
63 defparam U_mask_sr.M = 8;
64 assign LEDG = mask; // this assignment places mask[7:0] onto LEDG [0:7]
65
66 // z is 1 when the counter hits 0. Upper counter bits may be masked
67 assign z = ~|{(~mask & slow_count[M-1:M-8]), slow_count[M-9:0]};
68 // shift register connected to red LEDs
69 shift_rot_lr U_LED_sr(3'b100, ~KEY[0], z, CLOCK_50, SW[0], SW[1], SW[2], ~(SW[2] | SW[3]), LEDR
70 // shift register connected to 7-seg displays. Initialized to display 3-segment pattern
71 shift_rot_lr U_HEX_sr(3'b111, ~KEY[0], z, CLOCK_50, SW[0], SW[1], SW[2], ~(SW[2] | SW[3]), HEX_
72     defparam U_HEX_sr.M = 20;
73
74 hex_pattern h3 (HEX_sr[19:15], HEX3);
75     defparam h3.PATTERN = 0;
76 hex_pattern h2 (HEX_sr[14:10], HEX2);
77     defparam h2.PATTERN = 1;
78 hex_pattern h1 (HEX_sr[9:5], HEX1);
79     defparam h1.PATTERN = 0;
80 hex_pattern h0 (HEX_sr[4:0], HEX0);
81     defparam h0.PATTERN = 1;
82
83 endmodule
84
```

Step 2: Compile and Program

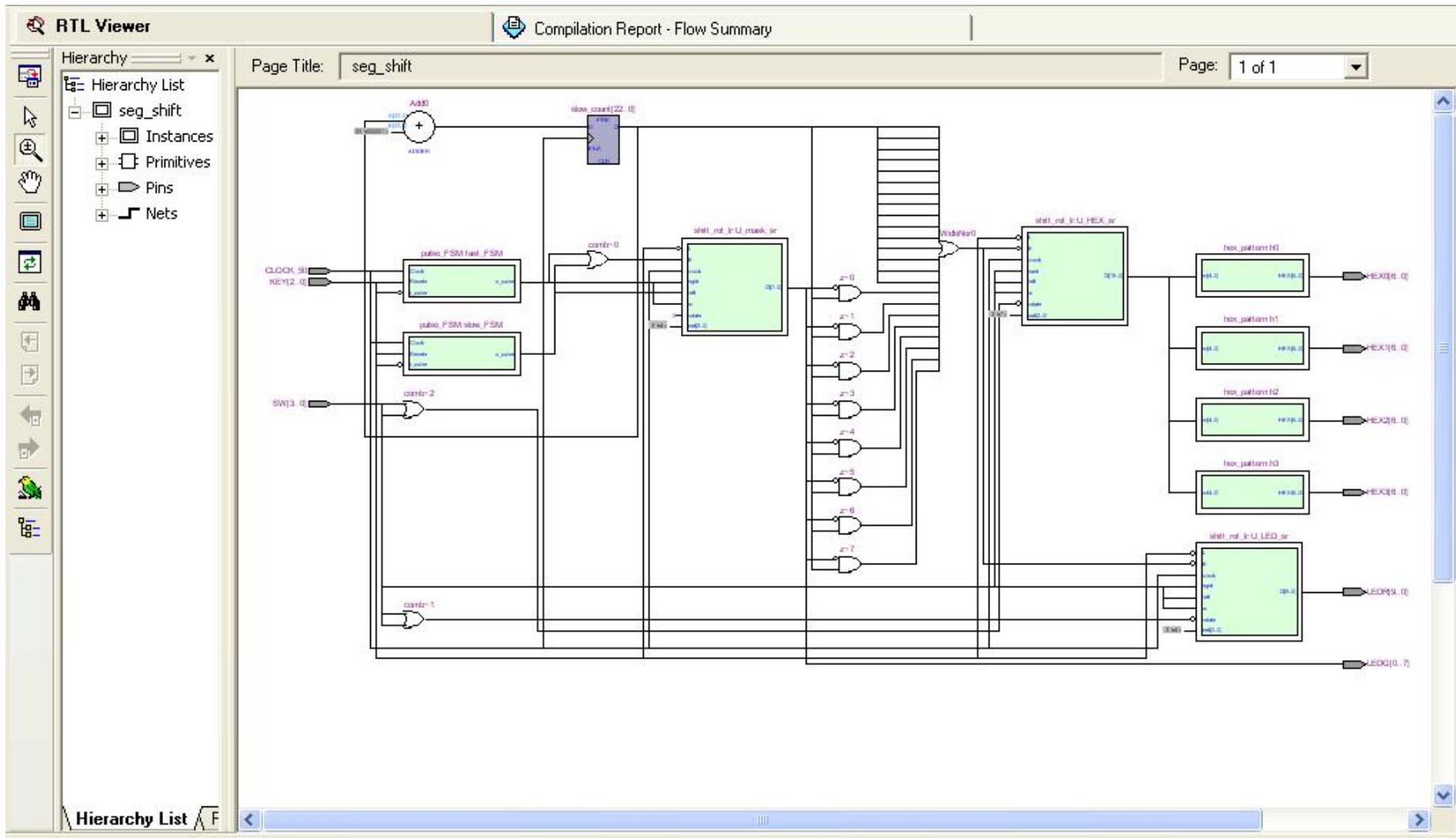
- Compile the design
- Program the design onto the DE2 board
- How does it work?
 - Press KEY[0] to reset the circuit
 - SW[0] to shift right, SW[1] to shift left
 - If SW[2] is high 1's are shifted into the register
 - If SW[3] is high 0's are shifted into the register
 - Press KEY[1] to speed up the circuit and KEY[2] to slow it down

Step 3: See the Circuit in RTL Viewer

- Start the **RTL Viewer**
 - Click **Tools**
 - Expand the **Netlist Viewers** list
 - Click **RTL Viewer**

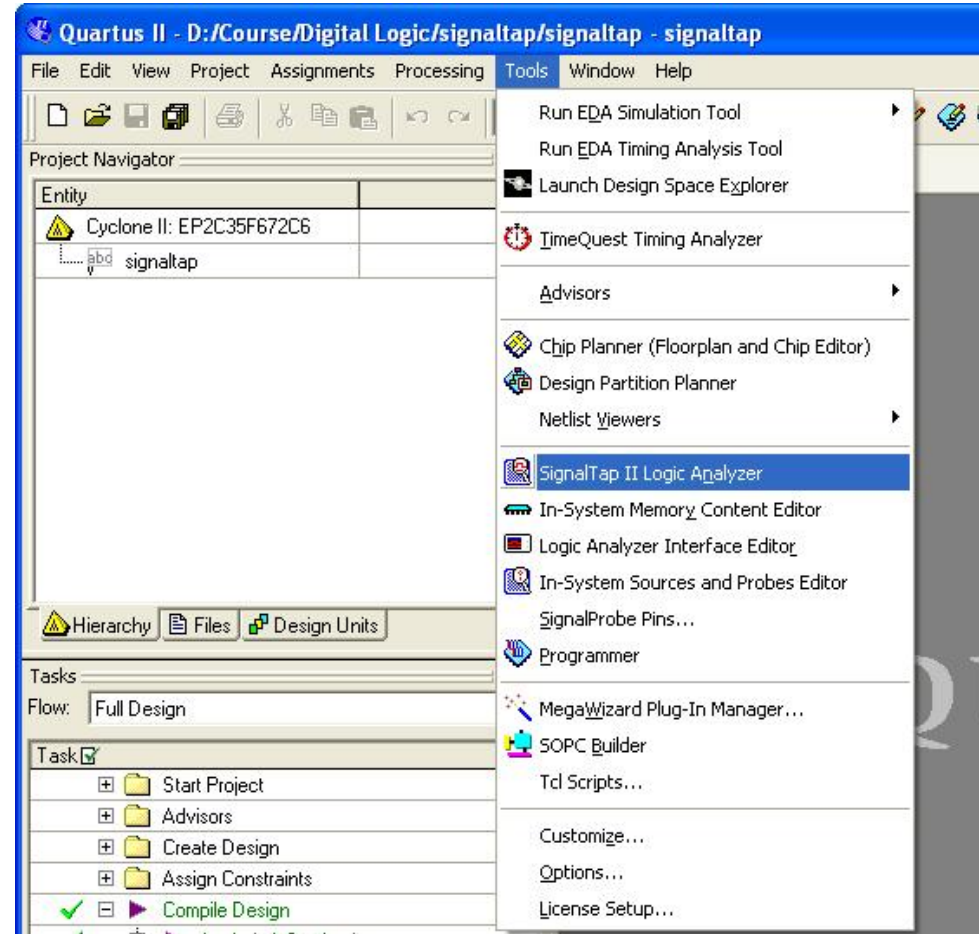


Examine the Circuit

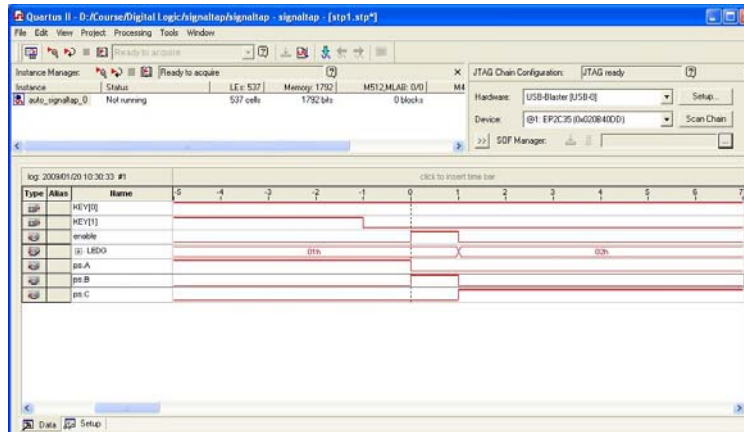


SignalTap II Embedded Logic Analyzer

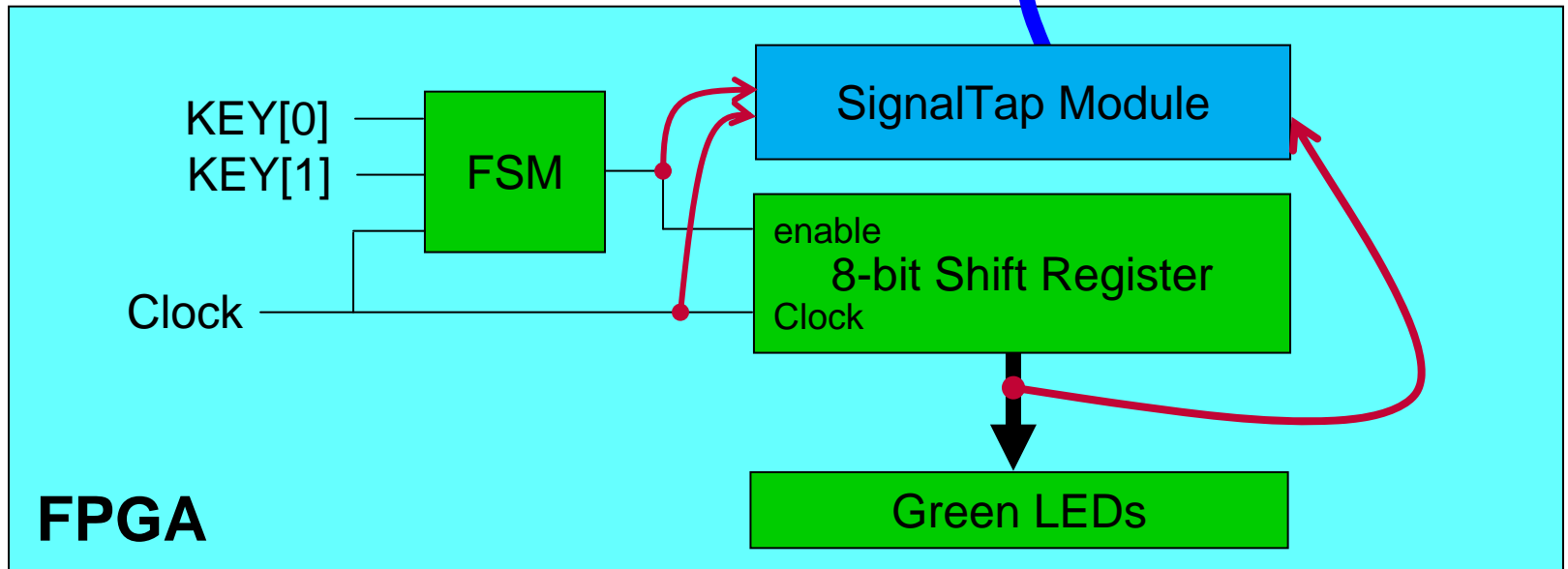
- A soft logic analyzer
 - Instantiate as a module in your design
- Connects to the board on which a design is running
- Collects data when a trigger event occurs
- Displays data on your computer
- How does it work?



SignalTap II Operation



USB-Blaster cable



Setup SignalTap II

The screenshot shows the Quartus II SignalTap II configuration window. Three blue callout boxes with red arrows point to specific settings:

- Specify Hardware Connection:** Points to the "USB-Blaster [USB-0]" dropdown in the "TAG Chain Configuration" section.
- Specify Clock to Specify Signals to Watch:** Points to the "CLOCK_50" dropdown in the "Signal Configuration" section.
- Specify Signals to Watch:** Points to the empty table in the "auto_sigtap_0" configuration area.

Type	Alias	Name	Data Enable	Trigger
Double-click to add nodes				

Other visible settings include:

- Device: @1: EP2C35 (0x020B40DD)
- Sample depth: 128
- RAM type: Auto
- Trigger flow control: Sequential
- Trigger position: Pre trigger position
- Trigger conditions: 1

Adding signals

The screenshot shows the Quartus II SignalTap II configuration window. The main window title is "Quartus II - D:/Course/Digital Logic/signaltap/signaltap - signaltap - [stp1.stp]". The Instance Manager shows one instance named "auto_signaltap_0" with status "Not running". The JTAG Chain Configuration shows "JTAG ready" and hardware "USB-Blaster [USB-0]". The Signal Configuration window is open, showing a table of nodes and a context menu.

Type	Alias	Name	Data Enable	Trigger Enable	Trigger Conditions
		KEY[D]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		KEY[1]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		enable	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		LEDG	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		ps.A	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		ps.B	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		ps.C	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

The context menu is open over the table, showing options such as "Delete", "Select All", "Find...", "Find Next", "Find Bus Value...", "Find Next: Bus Value", "Find Previous Bus Value", "Add Nodes with Plug-In", "Add Nodes...", "Plug-In Options...", "Add State Machine Nodes...", "Recreate State Machine Mnemonics...", "Locate", "Group", "Ungroup", "Rename", "Mnemonic Table Setup...", "Create SignalTap II List File", "Invert Signal", "Align Left", "Align Right", "MSB on Top, LSB on Bottom", "LSB on Top, MSB on Bottom", and "Bus Display Format".

Recompile Project

- For changes to take effect recompile project
- Once recompiled, download it to the board
- Note: The circuit will be larger than before
 - Memory is used to store captured data

Flow Summary

Flow Status	Successful - Tue Jan 20 10:25:56 2009
Quartus II Version	8.0 Build 215 05/29/2008 SJ Full Version
Revision Name	signaltap
Top-level Entity Name	signaltap
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	568 / 33,216 (2 %)
Total combinational functions	368 / 33,216 (1 %)
Dedicated logic registers	454 / 33,216 (1 %)
Total registers	454
Total pins	15 / 475 (3 %)
Total virtual pins	14
Total memory bits	1,792 / 483,840 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Setup Event Trigger

Click here to begin capture

Instance Manager:

Instance	Status	Cells	Memory	M512_MLAB	M4
auto_signaltap_0	Not running	537 cells	1792 bits	0/0	M4

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0] Setup...

Device: @1: EP2C35 (0x020B40DD) Scan Chain

SOF Manager:

trigger: 2009/01/20 10:24:59 #1 Allow all changes

Type	Alias	Name	Data Enable	Trigger Enable	Trigger Condition
		KEY[0]	14	14	Basic
		KEY[1]	14	14	Basic
		enable	14	14	Basic
		LEDG	14	14	XXh
		ps.A	14	14	Basic
		ps.B	14	14	Basic
		ps.C	14	14	Basic

Signal Configuration:

Clock: CLOCK_50

Data

Sample depth: 128 RAM type: Auto

Segmented: 128 1 sample segments

Trigger

Trigger flow control: Sequential

Trigger position: Pre trigger position

Trigger conditions: 1

Trigger in

Source: Don't Care

Pattern: Don't Care

- Don't Care
- Low
- Falling Edge
- Rising Edge
- High
- Either Edge

Insert Value...

Trigger the event and Analyze the results

Quartus II - D:/Course/Digital Logic/signaltap/signaltap - signaltap - [stp1.stp*]

File Edit View Project Processing Tools Window

Ready to acquire

Instance Manager: Ready to acquire

Instance	Status	LEs: 537	Memory: 1792	M512_MLAB: 0/0	M4
auto_signaltap_0	Not running	537 cells	1792 bits	0 blocks	

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0] Setup...

Device: @1: EP2C35 (0x020B40DD) Scan Chain

SOF Manager:

log: 2009/01/20 10:30:33 #1

click to insert time bar

Type	Alias	Name	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
		KEY[0]													
		KEY[1]													
		enable													
		LEDG													
		ps.A													
		ps.B													
		ps.C													

Data Setup

Summary

- Learned how to
 - Use Quartus II CAD Software
 - Compile projects in Quartus II
 - Target design onto DE2 board
 - View results of compilation
 - Use SignalTapII

- Where do we go from here?

Concluding Remarks