

# How to Program Your First FPGA Device

PDF created: December 13, 2017  
Validated using tools release: 17.0.0

## Contents

<b>Description</b> . . . . .	<b>1</b>
<b>Materials</b> . . . . .	<b>1</b>
Hardware . . . . .	1
Software . . . . .	1
<b>Create the FPGA Design</b> . . . . .	<b>3</b>
<b>Experiment on Your Own</b> . . . . .	<b>21</b>
<b>Sidebar Topics</b> . . . . .	<b>23</b>

## Description

This tutorial shows you how to create the hardware equivalent of “Hello World”: a blinking LED. This is a simple exercise to get you started using the Intel® Quartus® software for FPGA development.

You’ll learn to compile Verilog\* code, make pin assignments, create timing constraints, and then program the FPGA to blink one of the eight green user LEDs on the board. You’ll use a 50 MHz clock input (from the on-board oscillator) to drive a counter, and assign an LED to one of the counter output bits.

**Level:** beginner

## Materials

### Hardware

- Terasic DE10-Nano kit

The Terasic DE10-Nano development board, based on a Cyclone® V SoC FPGA, provides a reconfigurable hardware design platform for makers, IoT developers and educators. You can buy the kit [here](#).

### Software

- Intel Quartus Prime Software Suite Lite Edition

The FPGA design software used here is ideal for beginners as it’s free to download and no license file is required. You can download the software [here](#).

**Note:** The installation files are large (several gigabytes) and can take a long time to download and install. To minimize download time and disk space required, we recommend you download only those items necessary for this exercise. When prompted which files to download, uncheck **Select All** and select only **Quartus Prime** and **Cyclone V device support** only.

---

Cyclone, Intel, and Quartus are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Other names and brands may be claimed as the property of others.

**Select All**

**Quartus Prime Lite Edition (Free)**

- ☒ **Quartus Prime (includes Nios II EDS)**  
Size: 1.7 GB MD5: EEBA0E3703C46C806FE7A61E5868DE9E
- ☐ **ModelSim-Intel FPGA Edition (includes Starter Edition)**  
Size: 1.1 GB MD5: 81EAE2F2D55ABECD4F1265FCDFE74760

**Devices**  
You must install device support for at least one device family to use

- ☐ **Arria II device support**  
Size: 499.6 MB MD5: 77E151BBF3876F6110AB94F7C6A68047
- ☐ **Cyclone IV device support**  
Size: 466.7 MB MD5: 70A27B31D439D6271650C832A9785F2C
- ☒ **Cyclone V device support**  
Size: 1.1 GB MD5: 8386E6891D17DC1FAF29067C46953FC7
- ☐ **MAX II, MAX V device support**  
Size: 11.4 MB MD5: AFC8FF969FBA63E84D5D40AE812F83A2
- ☐ **MAX 10 FPGA device support**  
Size: 331.3 MB MD5: 013AACB391EAD32FF8E094D9D14987C3

Figure 1: Intel Quartus Prime Installation Options

Once you've downloaded and installed the Intel Quartus software, you're ready to get started creating a project!

Why is the Intel Quartus software download so big? [Sidebar Topic](#)

**Note:** User experience may vary when using earlier or later versions of Intel Quartus software. Screenshots in this document are based on the 16.1 release.

# Create the FPGA Design

## Step 1. Create an Intel Quartus Software Project

### Step 1a. Open Intel Quartus Prime Software Suite Lite Edition.

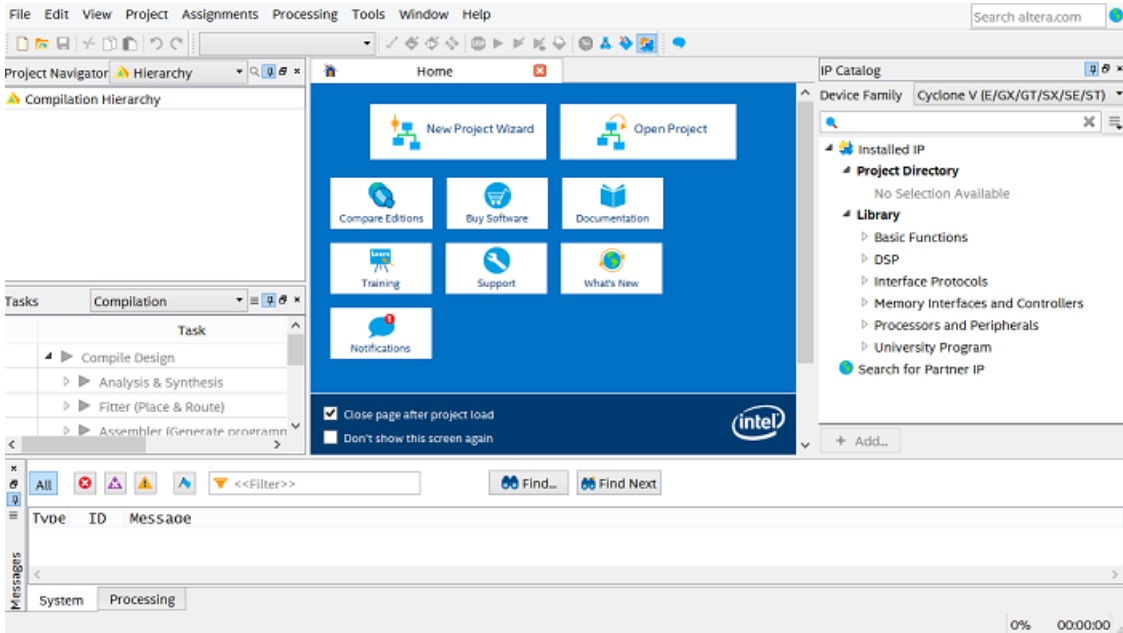


Figure 2: Intel Quartus Prime Software Window

### Step 1b. Open a New Project Wizard

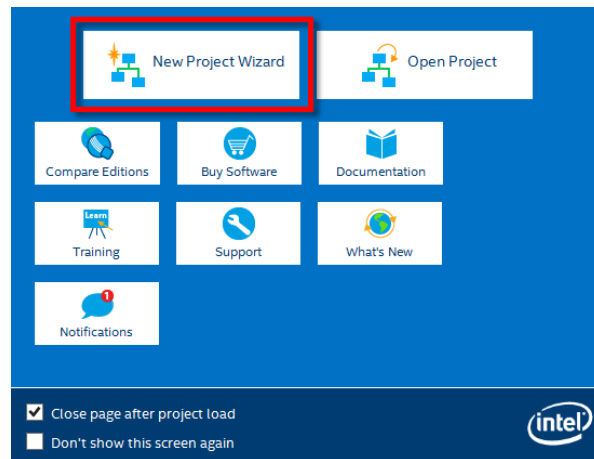


Figure 3: New Project Wizard Button

**Step 1c. Select Next**

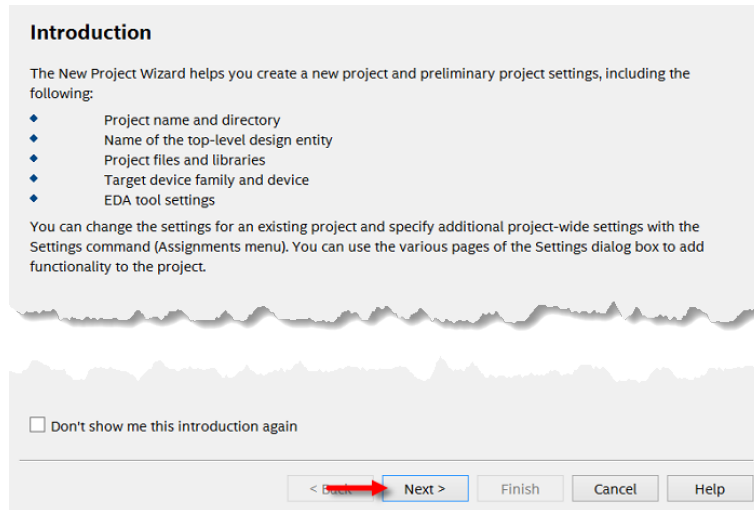


Figure 4: Introduction Dialog

**Step 1d. Directory, Name, Top-Level Entity**

Choose a directory to put your project under. Let's name our project "blink" and place it under the IntelFPGA\_lite folder. Select **Next**.

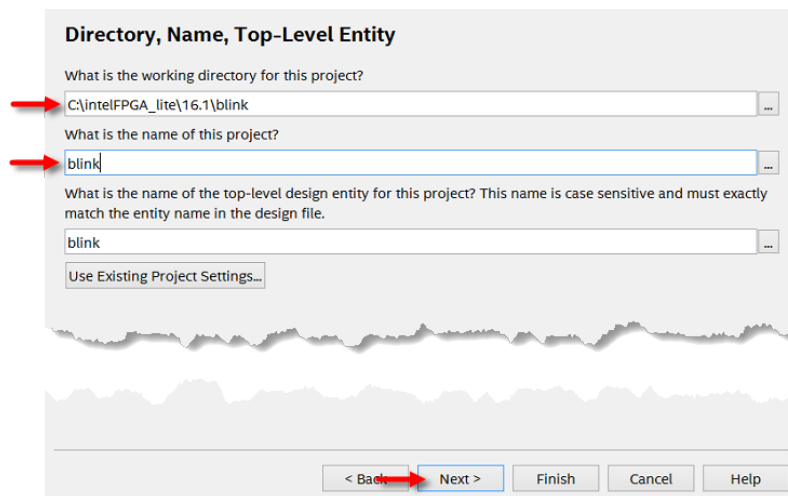


Figure 5: Project Location Details

When prompted to create the directory, choose **Yes**.

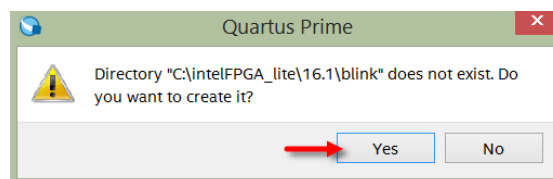


Figure 6: Create Directory Dialog

This project directory is convenient for an example tutorial, but isn't what we would recommend for future projects.

Where should I put my future project files? [Sidebar Topic](#)

**Step 1e.** Project Type

Select **Empty Project**, and then click **Next**.

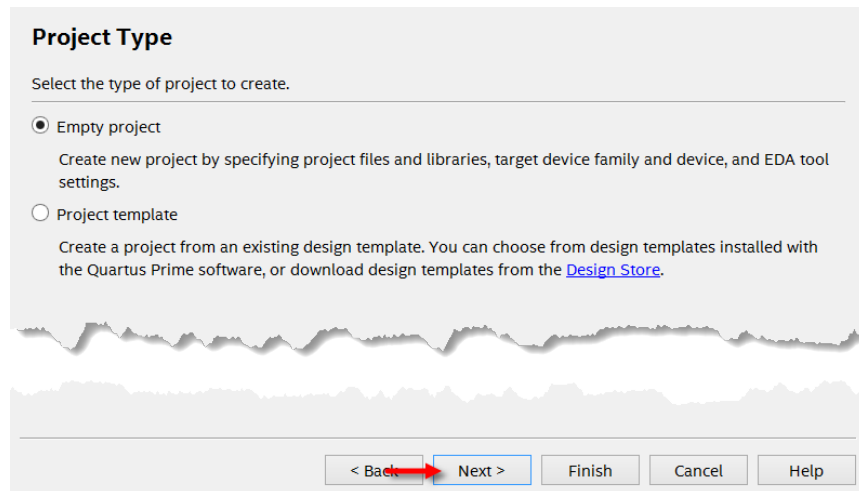


Figure 7: Project Type Dialog

**Step 1f.** Add Files

You won't be adding any files here. Click **Next**.

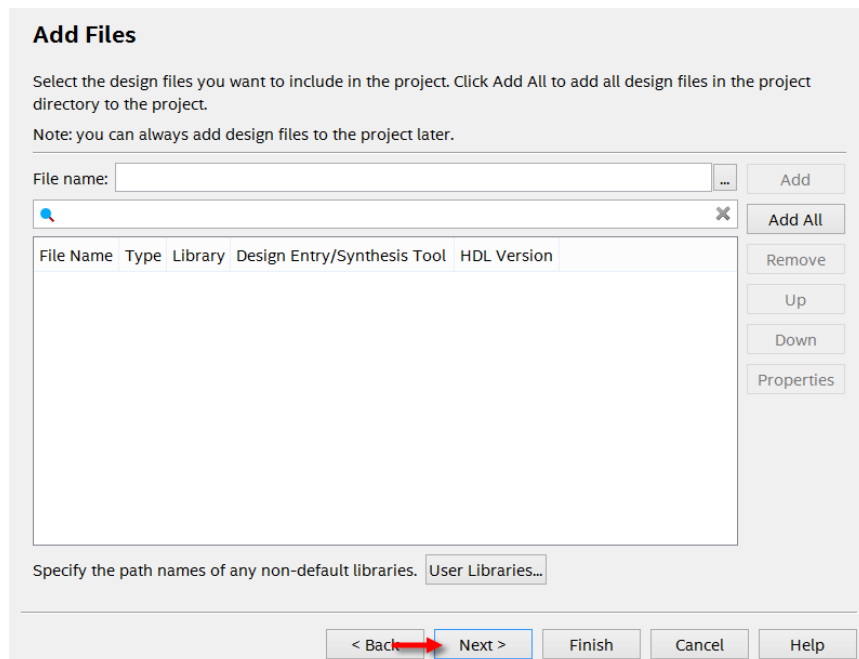


Figure 8: Add Files Dialog

**Step 1g.** Family, Device, and Board Settings

**Note:** You may need to expand the window to view more device names

Select the following:

- *Family:* **Cyclone V**
- *Device:* **Cyclone V SE Base**
- *Device name:* **5CSEBA6U23I7**

**Note:** To select the specific device you will need to click the up/down arrows to scroll through the list of supported devices until you find 5CSEBA6U23I7. Alternatively, you can enter the device name in the **Name filter** box to narrow down the list of devices. You may also need to expand the **Name** field to see the full device name.

**Family, Device & Board Settings**

Device Board

Select the family and device you want to target for compilation.  
You can install additional device support with the Install Devices command on the Tools menu.

To determine the version of the Quartus Prime software in which your target device is supported, refer to the [Device Support List](#) webpage.

Device family

Family: Cyclone V (E/GX/GT/SX/SE/ST)

Device: Cyclone V SE Base

Target device

☐ Auto device selected by the Fitter

☒ Specific device selected in 'Available devices' list

☐ Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: Any

Core speed grade: Any

Name filter:

☒ Show advanced devices

Available devices:

Name	Core Voltage	ALMs	Total I/Os	GPIOs	GXB Channel PMA	GXB Channel PCS
5CSEBA6U23I7	.1V	41910	314	314	0	0

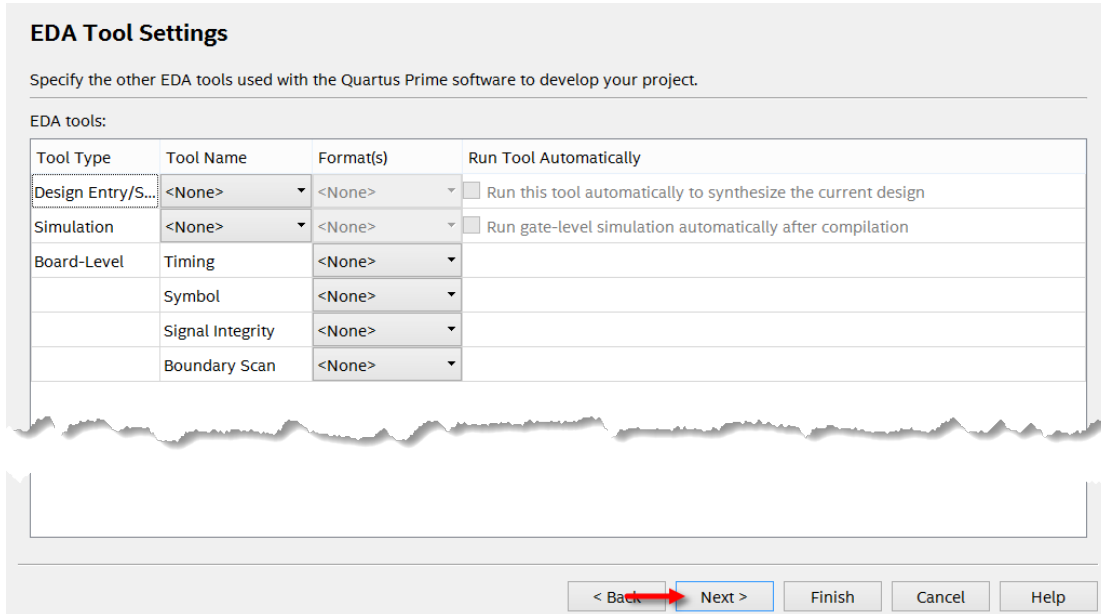
< Back Next > Finish Cancel Help

Figure 9: Family and Device Dialog

Click **Next**.

## Step 1h. EDA Tool Settings

We will be using the default EDA tools and settings so there are no changes to be made. Click **Next**.



The EDA Tool Settings dialog box is titled "EDA Tool Settings" and contains the instruction: "Specify the other EDA tools used with the Quartus Prime software to develop your project." Below this is a table with the following data:

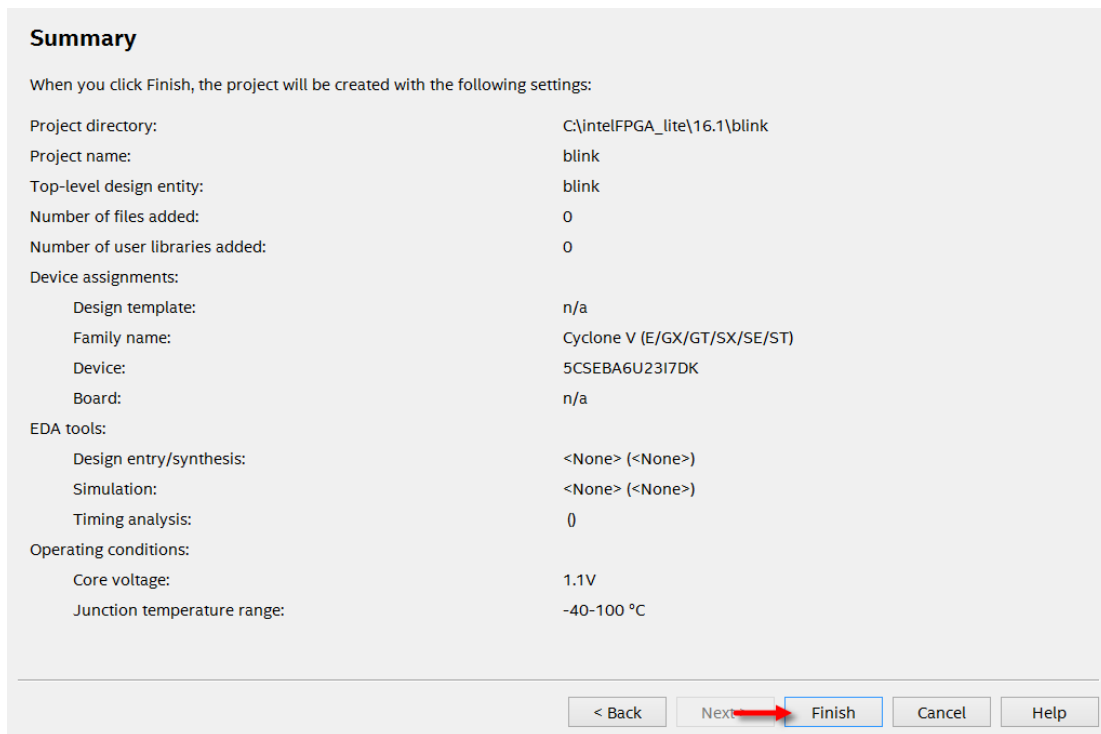
Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/S...	<None>	<None>	<input type="checkbox"/> Run this tool automatically to synthesize the current design
Simulation	<None>	<None>	<input type="checkbox"/> Run gate-level simulation automatically after compilation
Board-Level	Timing	<None>	
	Symbol	<None>	
	Signal Integrity	<None>	
	Boundary Scan	<None>	

At the bottom of the dialog are five buttons: "< Back", "Next >", "Finish", "Cancel", and "Help". A red arrow points to the "Next >" button.

Figure 10: EDA Tools Dialog

## Step 1i. Summary

Click **Finish**.



The Summary dialog box is titled "Summary" and contains the instruction: "When you click Finish, the project will be created with the following settings:"

Project directory:	C:\intelFPGA_lite\16.1\blink
Project name:	blink
Top-level design entity:	blink
Number of files added:	0
Number of user libraries added:	0
Device assignments:	
Design template:	n/a
Family name:	Cyclone V (E/GX/GT/SX/SE/ST)
Device:	5CSEBA6U23I7DK
Board:	n/a
EDA tools:	
Design entry/synthesis:	<None> (<None>)
Simulation:	<None> (<None>)
Timing analysis:	0
Operating conditions:	
Core voltage:	1.1V
Junction temperature range:	-40-100 °C

At the bottom of the dialog are five buttons: "< Back", "Next >", "Finish", "Cancel", and "Help". A red arrow points to the "Finish" button.

Figure 11: Summary Dialog

The following screen displays.

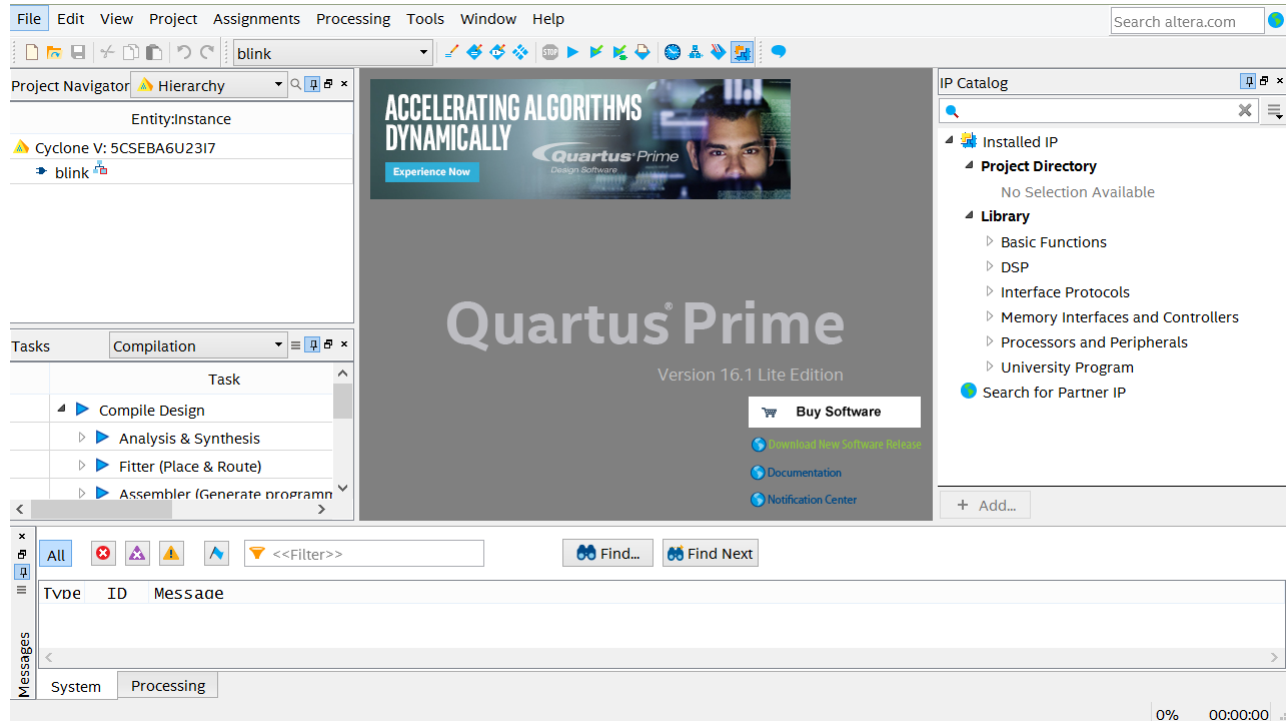


Figure 12: Your First Project Window

## Step 2. Create an HDL File

### Hardware Description Language (HDL)

We use Verilog as the HDL. If you are familiar with the C programming language but new to programming in an HDL, Verilog is similar to C.

**Step 2a.** Navigate to the File tab (main window), and then select **New**.

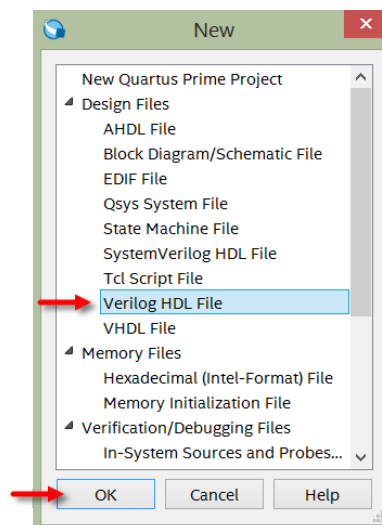


Figure 13: Select New Verilog HDL File

Select **Verilog HDL File**, and then click **OK**.



**Step 2b.** Choose **File > Save As**. Choose “blink” for the file name. This is your top-level file name (blink.v). Click **Save**.

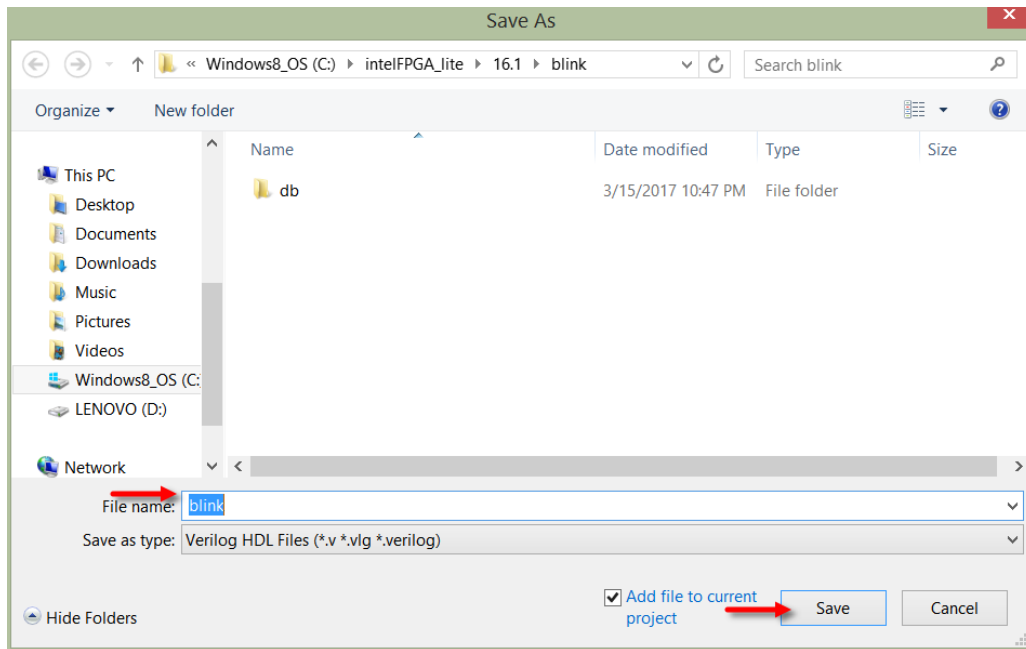


Figure 14: Saving Verilog HDL File

### Step 3. Create a Verilog Module

**Step 3a.** Download this Verilog code from the GitHub\* repository and paste it into the blink.v window, and then save the file.

Download the blink.v file [here](#).

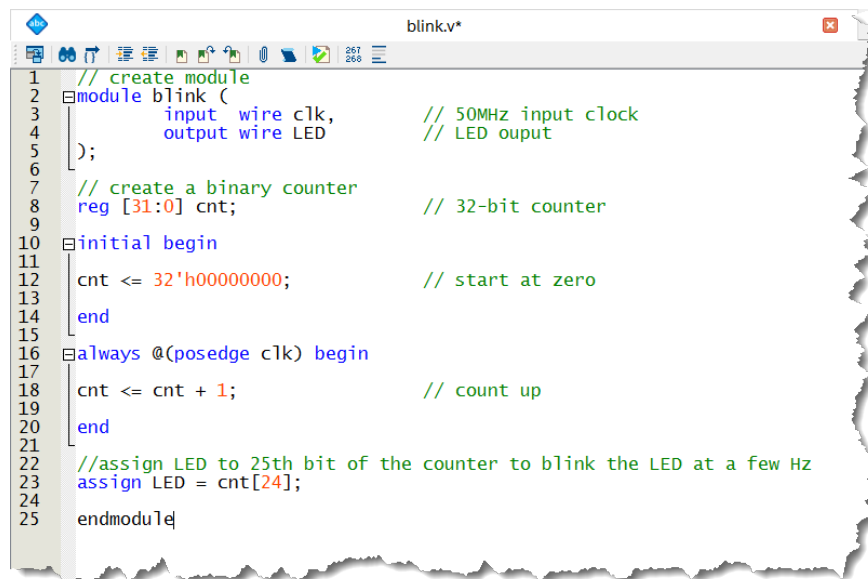
If you wish to view the blink.v file in the GitHub repo you can look [here](#).

```
1  //
2  // Copyright (c) 2017 Intel Corporation
3  //
4  // Permission is hereby granted, free of charge, to any person obtaining a copy
5  // of this software and associated documentation files (the "Software"), to
6  // deal in the Software without restriction, including without limitation the
7  // rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
8  // sell copies of the Software, and to permit persons to whom the Software is
9  // furnished to do so, subject to the following conditions:
10 //
11 // The above copyright notice and this permission notice shall be included in
12 // all copies or substantial portions of the Software.
13 //
14 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15 // IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16 // FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17 // AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
18 // LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
19 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
20 // IN THE SOFTWARE.
21 //
22
23 // create module
24 module blink (
25     input  wire clk,           // 50MHz input clock
26     output wire LED            // LED output
27 );
28
29 // create a binary counter
```

```

30 reg [31:0] cnt;           // 32-bit counter
31
32 initial begin
33
34 cnt <= 32'h00000000;      // start at zero
35
36 end
37
38 always @(posedge clk) begin
39
40 cnt <= cnt + 1;          // count up
41
42 end
43
44 //assign LED to 25th bit of the counter to blink the LED at a few Hz
45 assign LED = cnt[24];
46
47 endmodule

```



```

1 // create module
2 module blink (
3     input wire clk,      // 50MHz input clock
4     output wire LED      // LED output
5 );
6
7 // create a binary counter
8 reg [31:0] cnt;         // 32-bit counter
9
10 initial begin
11
12 cnt <= 32'h00000000;    // start at zero
13
14 end
15
16 always @(posedge clk) begin
17
18 cnt <= cnt + 1;        // count up
19
20 end
21
22 //assign LED to 25th bit of the counter to blink the LED at a few Hz
23 assign LED = cnt[24];
24
25 endmodule

```

Figure 15: Verilog HDL Pasted in Editor

### Step 3b. Analysis & Synthesis

Right click on **Analysis and Synthesis**, and then click **Start** to perform a syntax check and synthesis of the Verilog code. If you do not see the **Tasks** pane in your view, go to the **View** menu and select **Utility Windows** and then select **Tasks**.

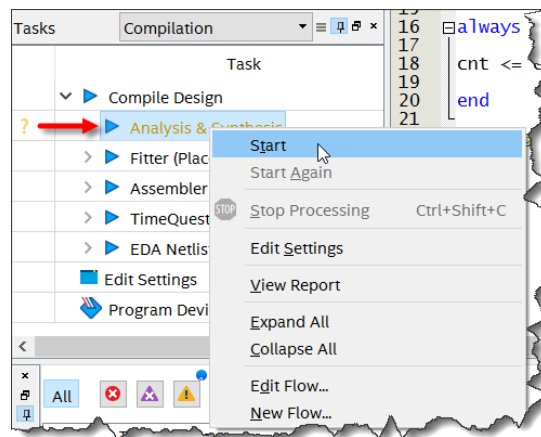


Figure 16: Start Analysis and Synthesis

*What happens during analysis and synthesis?* [Sidebar Topic](#)

If the process completes successfully, a green check mark displays next to Analysis & Synthesis. If you get an error, check your syntax and make sure it matches exactly the code block provided above.

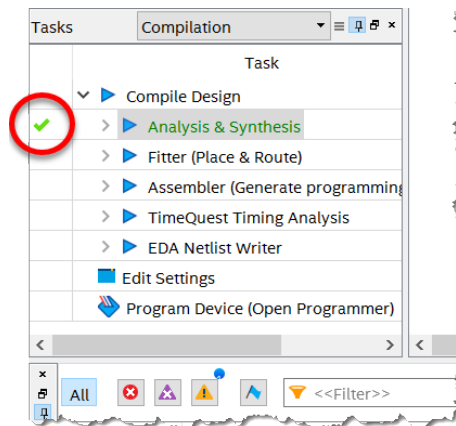


Figure 17: Analysis Complete

#### Step 4. Choose Pin Assignments

**Step 4a.** In the top navigation bar, select **Assignments > Pin Planner**.

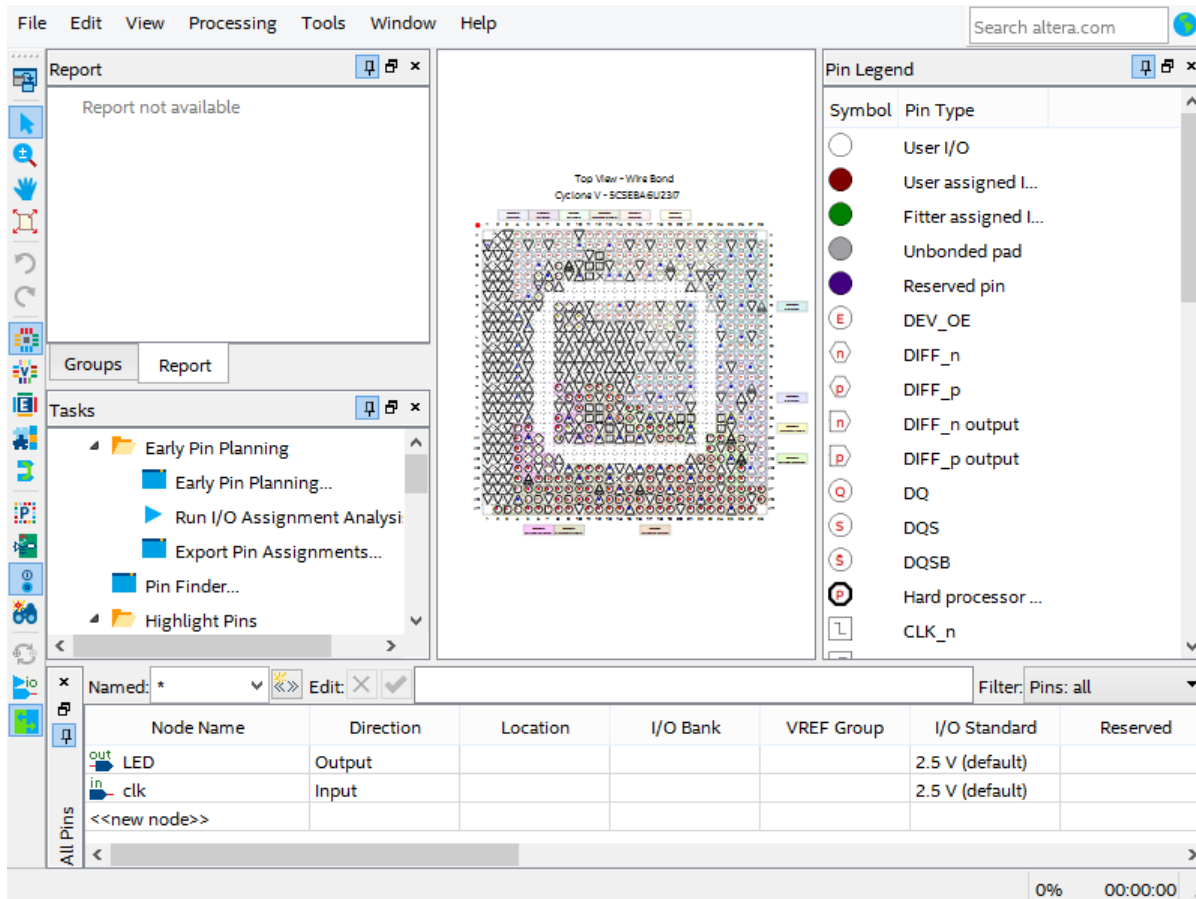


Figure 18: Pin Planner Window

Notice that the input (clk) and output (LED) are listed under **Node Name**. This is because you ran the Analysis & Synthesis.

**Step 4b.** One at a time, click to highlight the **Location** column for each pin, then type the pin location for the LED and clk signals as shown below. The rest of the columns will auto populate with data (some with default values that we'll modify in the next step).

Node Name	Location
LED	PIN_W15
clk	PIN_V11

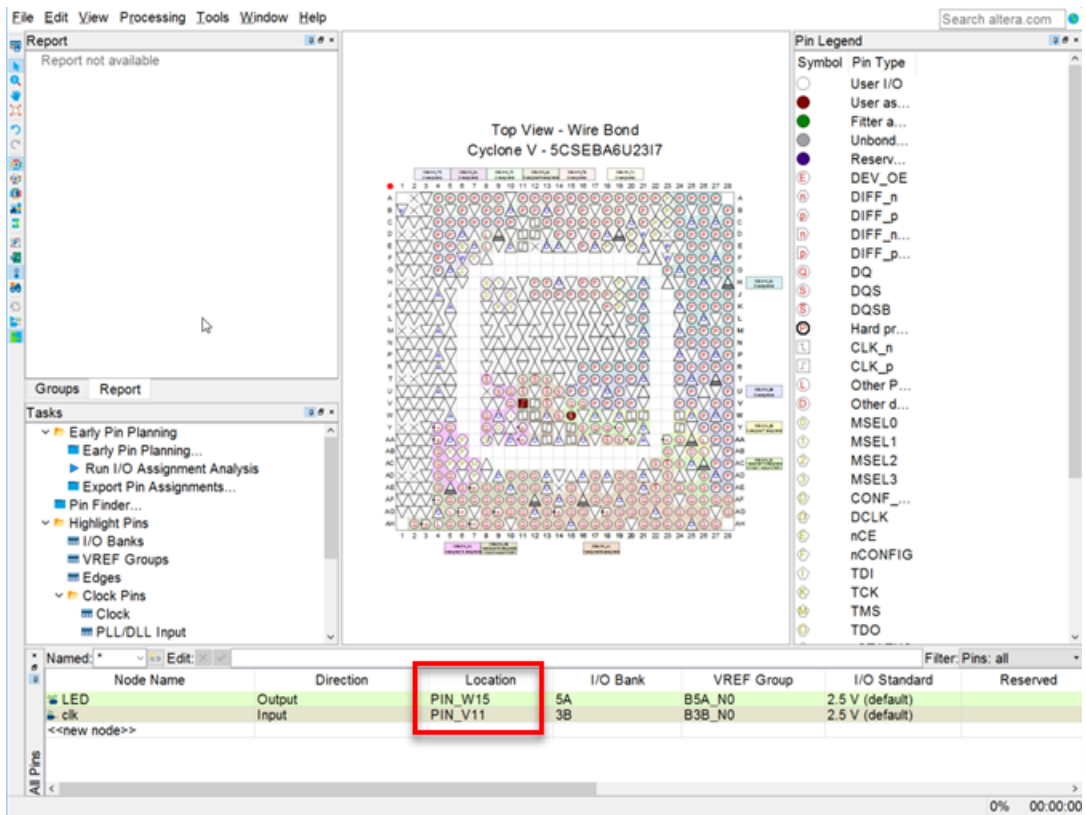


Figure 19: Pin Planner Locations Set

You also need to change the **I/O standard**, **Current Strength** and **Slew Rate** columns from their default settings to the values shown below:

Node Name	Location	I/O Standard	Current Strength	Slew Rate
LED	PIN_W15	3.3-V LVTTL	16ma	1
clk	PIN_V11	3.3-V LVTTL	16ma	

- Step 4c.** To change the **I/O standard**, double click each cell and a pull-down menu will appear. Click the down arrow icon and scroll to the desired value. Change the I/O standard from the default 2.5V to **3.3-V LVTTL**.
- Step 4d.** The Current Strength will by default read 16ma(default) and we need to select 16ma. If we don't specifically set them, then we get warning messages in our compilation. Set **Current Strength** to **16ma** for both input (clk) and output (LED).
- Step 4e.** Next change **Slew Rate** for the LED output from 1(default) to **1(fastest)**. You don't need to set a Slew Rate for the clk. Leave that blank.

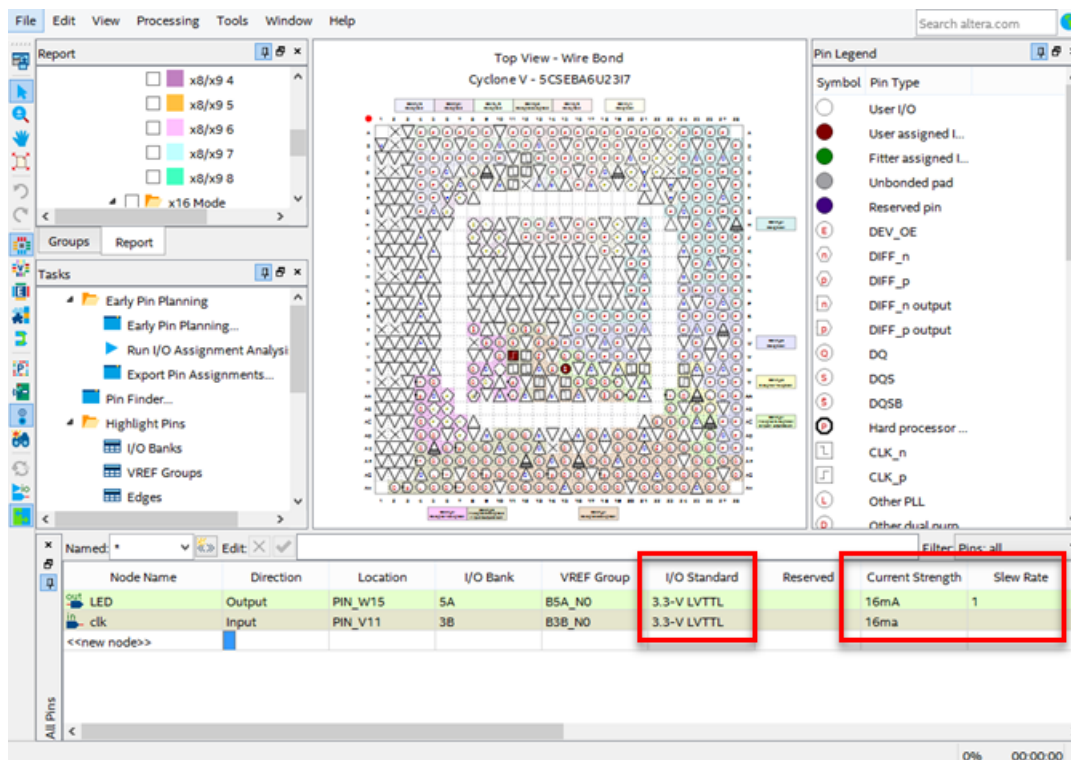


Figure 20: Pin Planner All Set

**Step 4f.** Close the Pin Planner. The changes that you make in the Pin Planner are saved automatically.

**Step 5.** Create an SDC File

Before you compile the Verilog code, you need to provide timing constraints for the design. You'll create an SDC (Synopsis Design Constraints) file that contains commands to let the Intel Quartus software know how to close timing on the design. Without it, you will get warning messages in the compile flow because the Intel Quartus software has no idea how to close timing on the design.

**Step 5a.** To create a blink.sdc and add it to the blink project directory, do the following.

Download the blink.sdc file [here](#).

If you wish to view the blink.sdc file in the GitHub repo you can look [here](#).

```

1  #
2  # Copyright (c) 2017 Intel Corporation
3  #
4  # Permission is hereby granted, free of charge, to any person obtaining a copy
5  # of this software and associated documentation files (the "Software"), to
6  # deal in the Software without restriction, including without limitation the
7  # rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
8  # sell copies of the Software, and to permit persons to whom the Software is
9  # furnished to do so, subject to the following conditions:
10 #
11 # The above copyright notice and this permission notice shall be included in
12 # all copies or substantial portions of the Software.
13 #
14 # THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15 # IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16 # FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17 # AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
18 # LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
19 # FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
20 # IN THE SOFTWARE.
21 #

```

```

22
23 # inform quartus that the clk port brings a 50MHz clock into our design so
24 # that timing closure on our design can be analyzed
25
26 create_clock -name clk -period "50MHz" [get_ports clk]
27
28 # inform quartus that the LED output port has no critical timing requirements
29 # its a single output port driving an LED, there are no timing relationships
30 # that are critical for this
31
32 set_false_path -from * -to [get_ports LED]

```

**Step 5b.** Now save the file as “blink.sdc”. Place it in the blink project directory - the same directory where our Verilog file (.v file extension) lives.

What is an SDC file, and why do I need one? [Sidebar Topic](#)

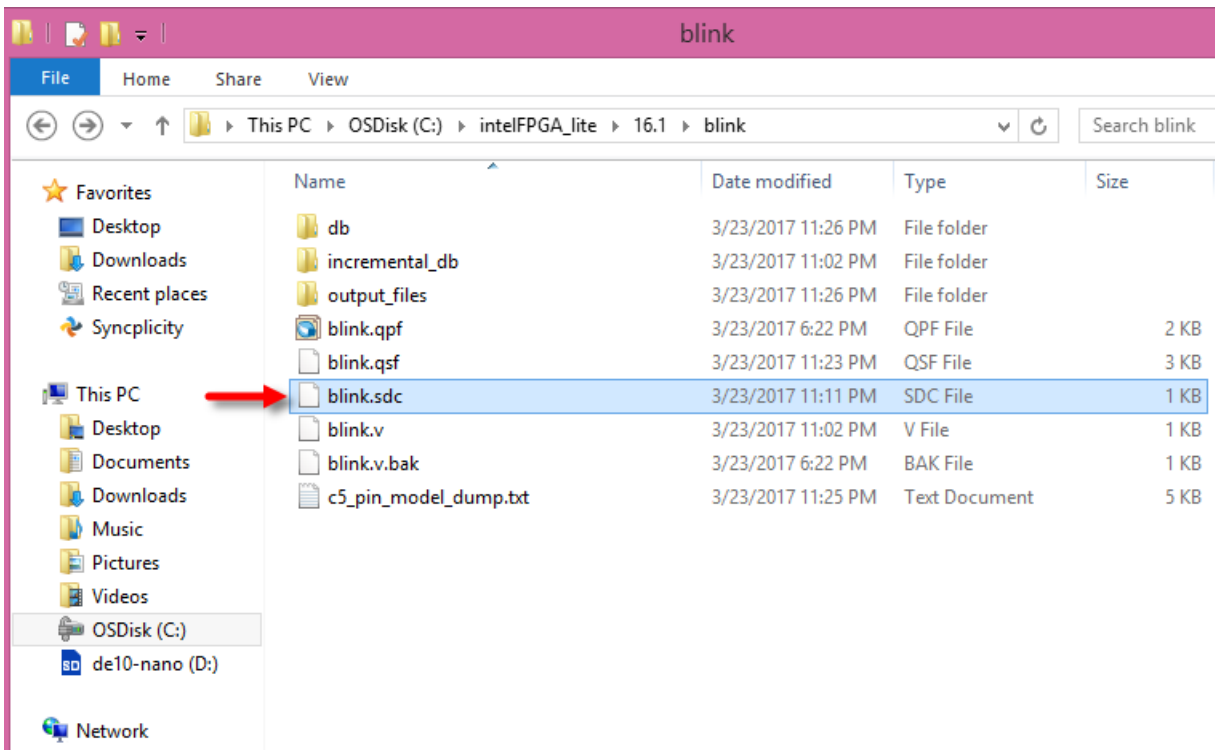


Figure 21: SDC File Location

**Step 5c.** Now you need to add this SDC this file to the Intel Quartus software project. Select the **Project** menu and then **Add/Remove Files in Project...** In the **Settings** dialog that appears, click the browse ... button to the right of the **File name** text box. This will bring up a file browser dialog. Change the **Files of type** option to **Script Files** or **All Files** so that the “blink.sdc” file is displayed. Select the “blink.sdc” file and click the **Open** button to close the dialog. At this point the “blink.sdc” file should appear in the project files list and you can click the **OK** button to close the **Settings** dialog.

## Step 6. Compile the Verilog Code

**Step 6a.** Right click on **Compile Design**, and then click **Start**. The tools will then synthesize, place and route, assemble and perform timing analysis on the design. Because there are only a handful of code lines, the compilation should only take a couple of minutes to complete.

*What happens during the fitter (place & route)?* [Sidebar Topic](#)

*What happens during the assembler?* [Sidebar Topic](#)

*What is timing analysis?* [Sidebar Topic](#)

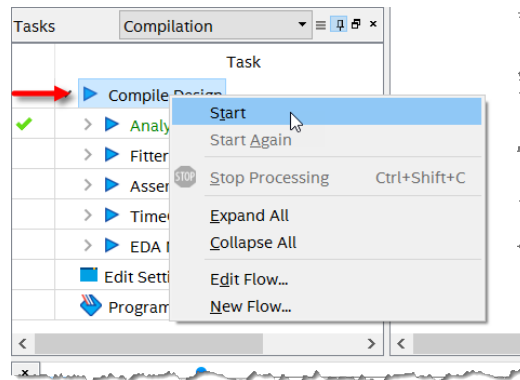


Figure 22: Compile Design

After compilation is complete you will see a summary report like the one below.

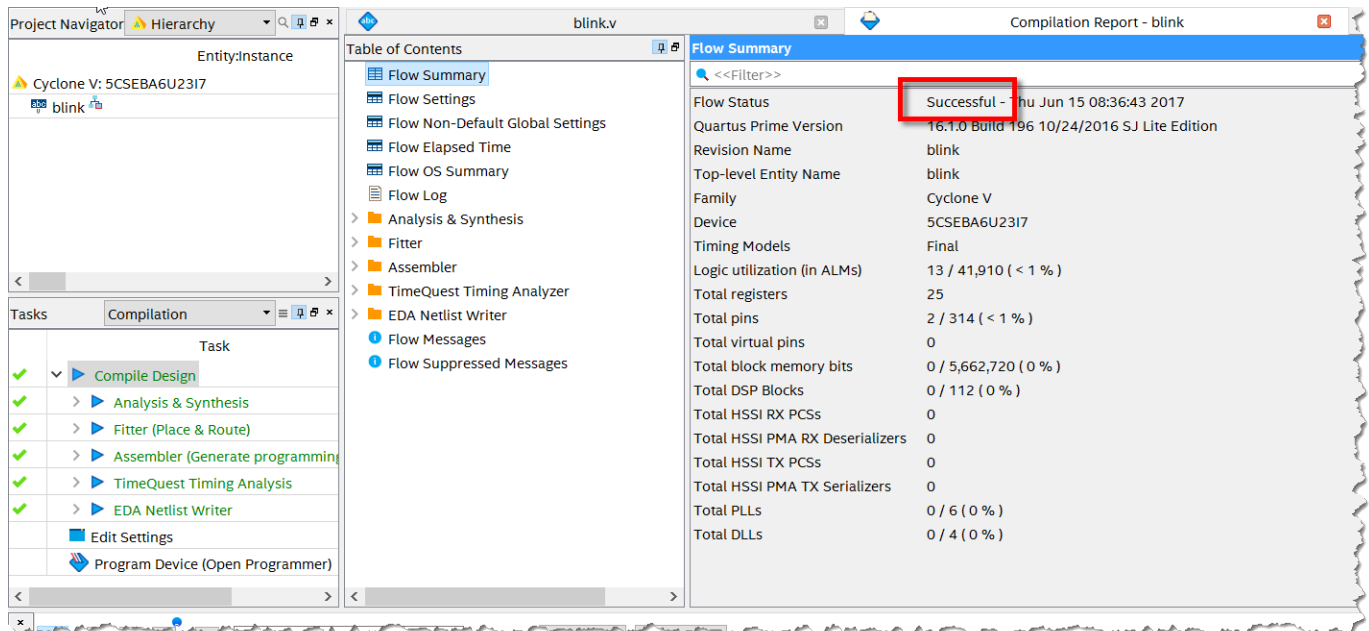


Figure 23: Compile Summary

After you compile the Verilog code, you can program the FPGA.



## Step 7. Program the FPGA

The final step is to program the FPGA. Before we do that, be sure to remove the SD Card from the board.

*Why should I remove the SD Card?* [Sidebar Topic](#)

To program the FPGA, you need to connect the board to your computer via the USB Blaster II port.

**Step 7a.** Connect the board to your computer via the USB Blaster II port. Use the USB cable (mini-b connector) that came with the Terasic DE10-Nano kit. Insert the mini-b connector into the USB Blaster II port (J13) on the Terasic DE10-Nano board and the Type-A end into a standard USB port on your host computer.

**Step 7b.** Connect the board to power and verify there is a blue LED lit near the J13 USB Blaster II port.

**Step 7c.** Right click to open **Program Device**.

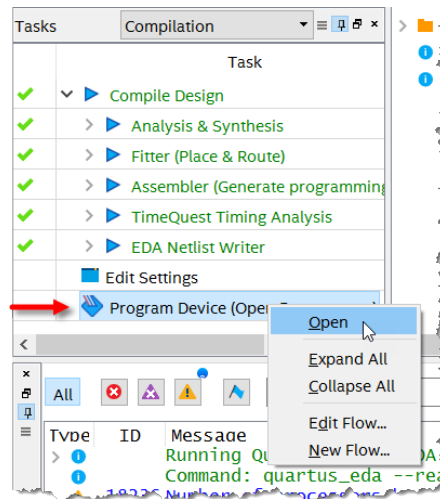


Figure 24: Open Intel Quartus Software Programmer

**Step 7d.** Hardware Setup

Select **Hardware Setup**.

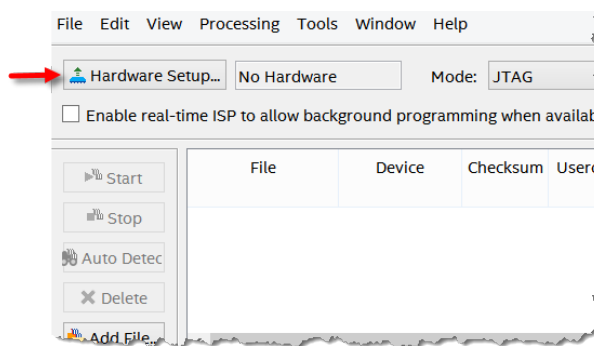


Figure 25: Hardware Setup

Under the drop down for **Currently selected hardware**, choose **DE-SoC**, then click **Close**.

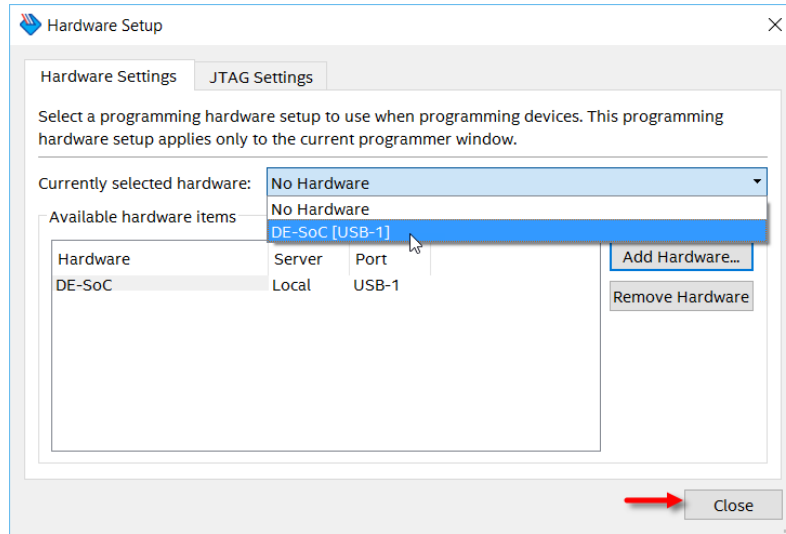


Figure 26: Cable Selection

**Step 7e.** Click **Auto Detect** to identify the JTAG chain on the board.

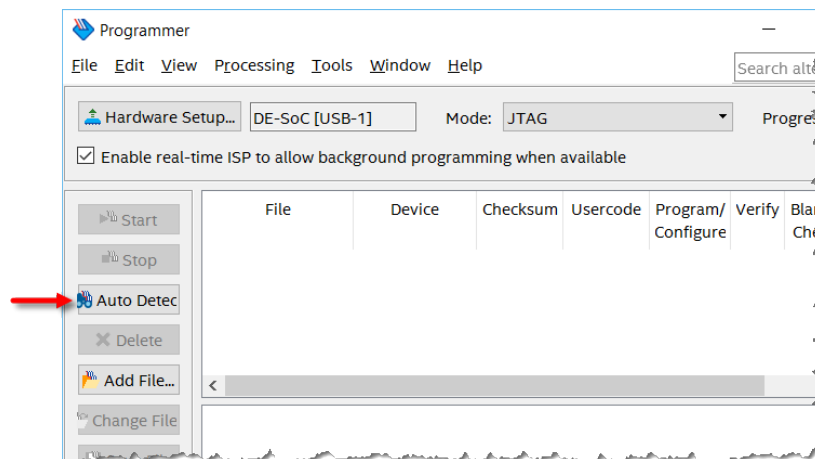


Figure 27: Auto Detect

Select the device **5CSEBA6**. This is the FPGA device.

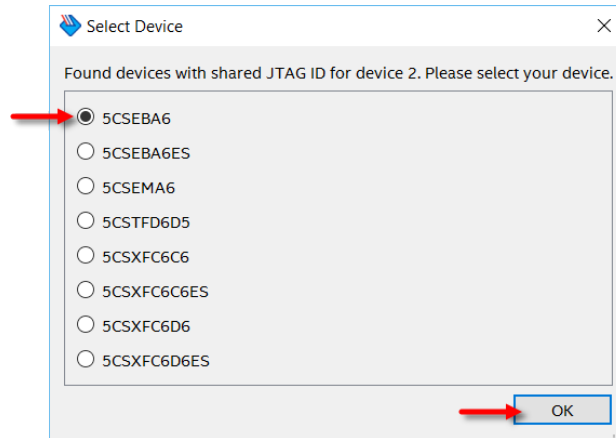


Figure 28: Device Selection

**Step 7f.** Add the .sof file.

Right click on the **File** column for the **5CSEBA6** device and select **Change File**.

*Why are there two devices found in the JTAG chain?* [Sidebar Topic](#)

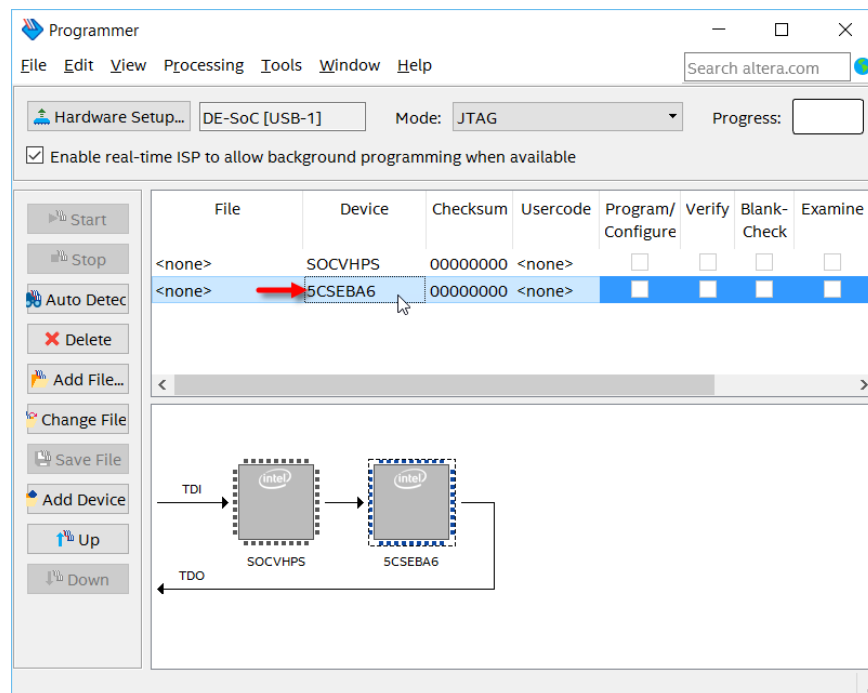


Figure 29: Change File

**Step 7g.** Navigate to the *output\_files* folder, select **blink.sof**, and then click **Open**.

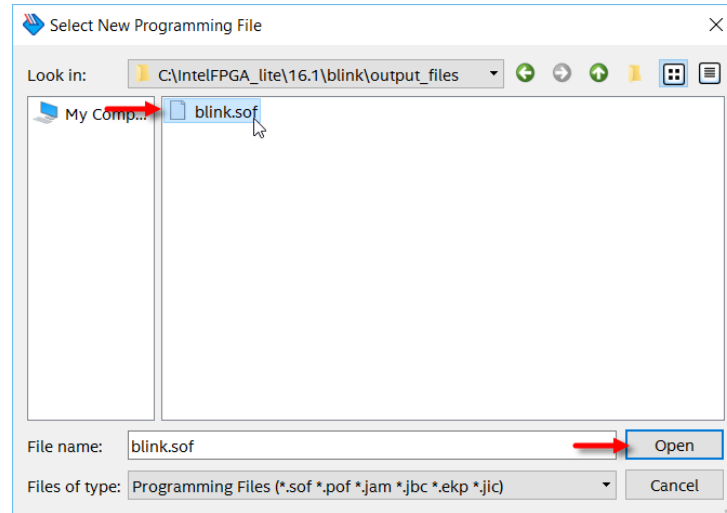


Figure 30: Select SOF File

Here, **blink.sof** is your programming file. SRAM Object Files (.sof files) are binary files containing data for configuring SRAM-based devices, our FPGA is based on SRAM. The Intel Quartus software Program Device (also called the Quartus Programmer) looks into the SOF file and gets the programming bit stream for the device.

**Step 7h.** Check the **Program/Configure** column, and then click **Start**.

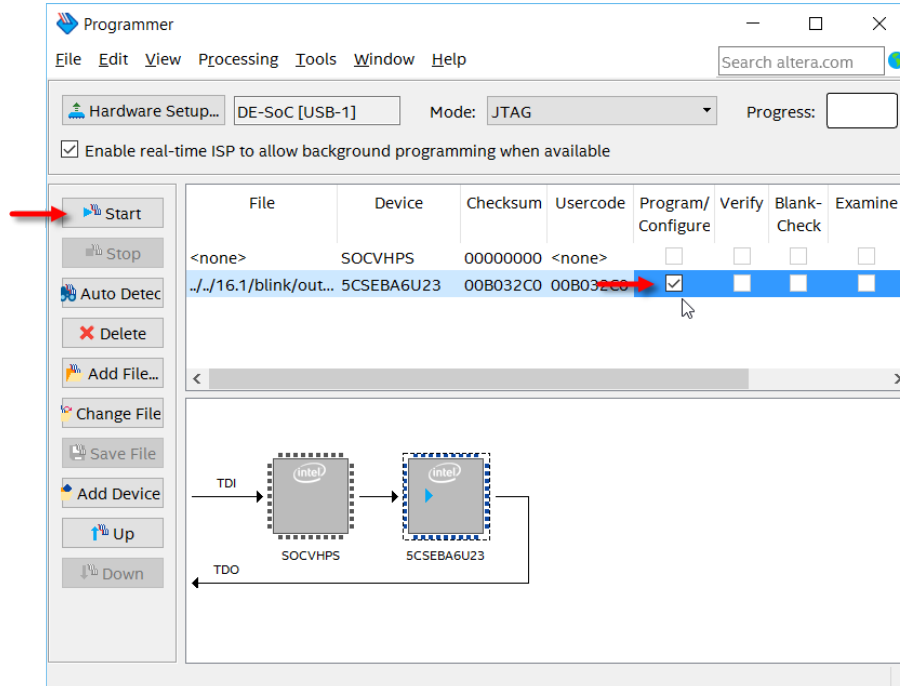


Figure 31: Start Programming

## Step 8. Observe the blinking LED

If your progress bar is 100% (Successful), watch LED[0] blink.

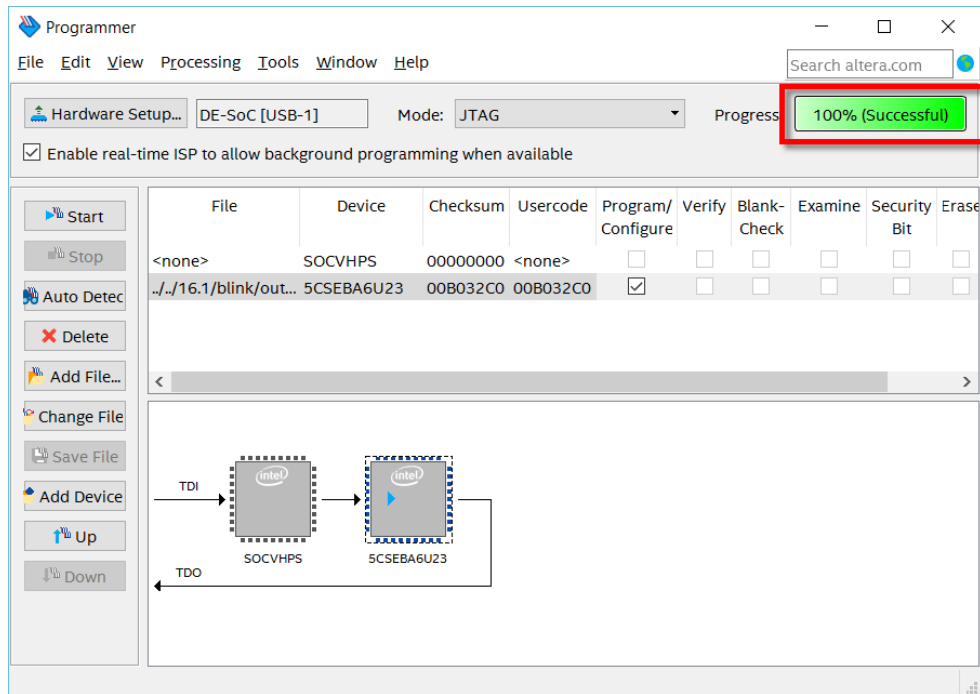


Figure 32: Programming Success

Image used with permission from Terasic Technologies Inc.

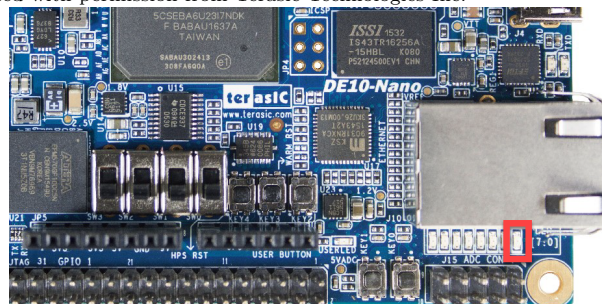


Figure 33: Blinking LED

## Experiment on Your Own

### Change the blink rate

Now that you've got one successful blinking LED under your belt, you can modify the blink rate of the LED by using a different "bit" of the counter. 1 Hertz is 1 per second (cycles per second) and blinking at 1 Hz means the LED blinks once per second. For 2 Hz, the LED blinks twice per second. And 0.25 Hz would blink the LED once every 4 seconds (slow blink). For a slower blink, use a higher bit of the counter and for faster, use a smaller bit (e.g., cnt[22]). Test out different counter bits and see what you get.

### *Clock and Counter Math*

`cnt[n]` where `n` = the counter bit

$2^n$ ; we've chosen `n` = 24 in our Verilog code sample

$2^{24} = 16,777,216$

Our clock is 50 MHz or 50,000,000

$\text{Clock} / 2^n = \text{blinks per second}$

$50,000,000 / 16,777,216 = 2.9802$

That's about 3 blinks per second.

To make this change, you will need to:

1. Modify the Verilog file (`blink.v`) and select a different counter bit in the assignment statement.
2. Recompile the design
3. Reprogram the FPGA

### **Add More LEDs**

Try connecting one or more of the remaining 7 LEDs to other counter bits, with each blinking at a different rate. For example, you could add a new LED connected to counter bit 23 which would blink twice as fast as bit 24. Or you could add all the remaining 7 LEDs and connect each to a unique counter bit.

To make this change, you will need to:

1. Modify the Verilog code:
  - (a) Add the new LEDs to the module definition
  - (b) List the new LEDs as being outputs
  - (c) Assign each of the new LEDs to a unique counter bit (`cnt[n]`)

*I'm new to hardware design. Where can I get the Verilog code that contains these changes?* [Sidebar Topic](#)

2. Run Analysis and Synthesis
3. Assign the LEDs outputs to pins using the Pin Planner.

Be sure to set the proper I/O standard, current strength, and slew rate. The I/O pins connected to the LEDs are as follows:

LED[0]: PIN\_W15  
LED[1]: PIN\_AA24  
LED[2]: PIN\_V16  
LED[3]: PIN\_V15  
LED[4]: PIN\_AF26  
LED[5]: PIN\_AE26  
LED[6]: PIN\_Y16  
LED[7]: PIN\_AA23

4. Recompile the design
5. Reprogram the device

## Sidebar Topics

### Why is the Intel Quartus software download so big?

The Intel Quartus download contains several sophisticated tools to create a custom chip design, such as simulators, synthesis tools, place and route engines, timing analyzers, and device programmers, to name a few. Nearly all those functions are built into the Intel Quartus Prime Software Suite FPGA design software itself. The download also includes the embedded software design suite for the Nios II soft CPU, and one or more FPGA family databases - in our case the Cyclone V FPGA database.

### Where should I put my future project files?

Here are a few guidelines you should adopt when choosing a directory for your project:

- Don't put projects within the Intel Quartus software tool directory. New Intel Quartus software versions come out every six months, so placing them within a specific version directory will make them "orphans" once a new version is installed. Even worse, you might lose them if you delete the older tool version.
- Avoid paths with spaces in the name since some of the tools don't like spaces in directory paths. You might use underscore characters or hyphens instead of space characters in your path names.
- Use directories where you have read/write access. This sounds intuitive, but sometimes IT departments limit administrator rights. Be sure the folder you create doesn't require admin rights.

### What happens during analysis and synthesis?

The Analysis & Synthesis process:

- Checks design files for syntax and semantic errors
- Performs netlist extraction to build a database which integrates all the design files
- Synthesizes the hardware description language (HDL) code into hardware blocks appropriate for the target FPGA family

### What is an SDC file, and why do I need one?

SDC stands for Synopsys Design Constraints and is an industry standard format which defines the timing constraints for a hardware (silicon) design such as the target frequency of the device, and the timing to external peripherals. The SDC file provides a way for the Intel Quartus software to verify that the system generated meets its timing requirements.

During synthesis of your FPGA, a design tool called TimeQuest is called by the Intel Quartus software which reads the timing constraints files, calculates the timing of the internal FPGA signals, and compare these timings to the timing requirements specified by the SDC files. A report is created which verifies timing is met and / or identifies signals which fail to meet timing and require optimization.

### What happens during the fitter (place & route)?

The fitter places and routes the logic of your synthesized design into target device resources. Think of it like a router that lays out a printed circuit board, connecting the various devices together using copper traces. In this case, however, the devices are logic resources (e.g. lookup tables, registers, memories, multipliers, etc.) and the traces are routing "wires" inside the FPGA device.

### What happens during the assembler?

The assembler generates a programming image from a successful fit which can then be downloaded to the FPGA device.

### What is timing analysis?

Timing analysis is the process of evaluating the timing of logic in the device after it has been synthesized, placed and routed to ensure the all timing requirements are met. The goal is to achieve "timing closure" where all the timing constraints are met.

### Why should I remove the SD Card?

The default behavior of the Terasic DE10-Nano kit is to boot from the SD Card. The processor boots, then configures the FPGA under software control. If you leave the SD Card plugged into the board and then reprogram the FPGA, a

watchdog timer in the processor would eventually timeout since the FPGA can no longer respond to the processor because the image has changed. The timeout would force a warm reset which would cause the FPGA to be reprogrammed, overwriting the “blink” design you just downloaded.

This behavior is how the Terasic DE10-Nano kit was designed to operate by default, but is not the way all systems have to behave. You decide how the system responds to warm and cold reset. You may choose to leave the FPGA running “as is” when a processor reset condition occurs.

### Why are there two devices found in the JTAG chain?

The Cyclone V SoC device has two JTAG chains, one dedicated to the FPGA and one dedicated to the hard processor system (HPS). As the programmer illustration shows, they are connected in serial on the Terasic DE10-Nano board so you only need one JTAG connection to communicate with both.

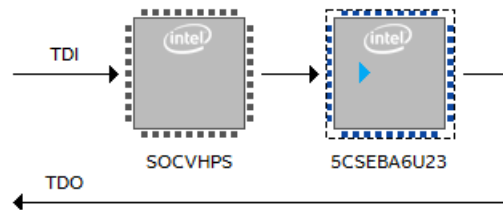


Figure 34: Terasic DE10-Nano JTAG Chain

The FPGA JTAG chain is used to configure the FPGA logic, and for hardware debugging using one of several tools such as:

- [SignalTap II](#) logic analyzer
- [System Console](#) system-level debugger

The HPS JTAG chain is primarily used for software development using tools like the ARM\* Development Studio 5\* ([DS-5\\*](#)).

### I'm new to hardware design. Where can I get the Verilog code that contains these changes?

Right here:

```
// create module
module blink(
    input wire    clk,           // 50MHz input clock
    output wire [7:0] LED        // array of 8 LEDs
);

// create a binary counter
reg [31:0] cnt;                 // 32 bit counter

initial begin
    cnt <= 32'h00000000;         // start count at zero
end

always @(posedge clk) begin
    cnt <= cnt+1;                // count up
end
```



```
//assign LEDs to bits 28 through 21 of the counter  
assign LED = cnt[28:21];  
endmodule
```