

[Learn More](#)

How to Program Your First FPGA Device

Published: 03/24/2017

Last Updated: 03/24/2017

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

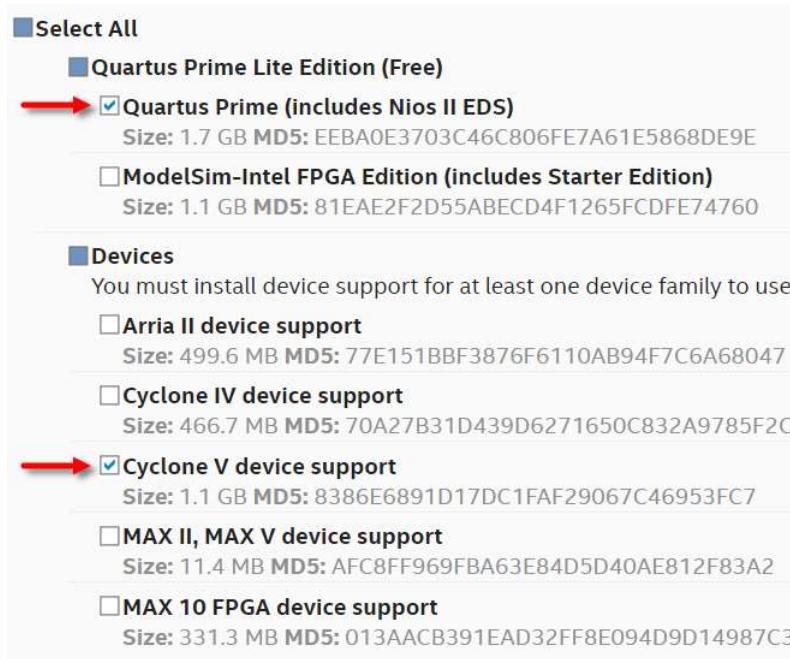
The Terasic DE10-Nano development board, based on an Intel® Cyclone V SoC FPGA, provides a reconfigurable hardware design platform for makers, IoT developers and educators. You can buy the kit [here](#).

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.



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

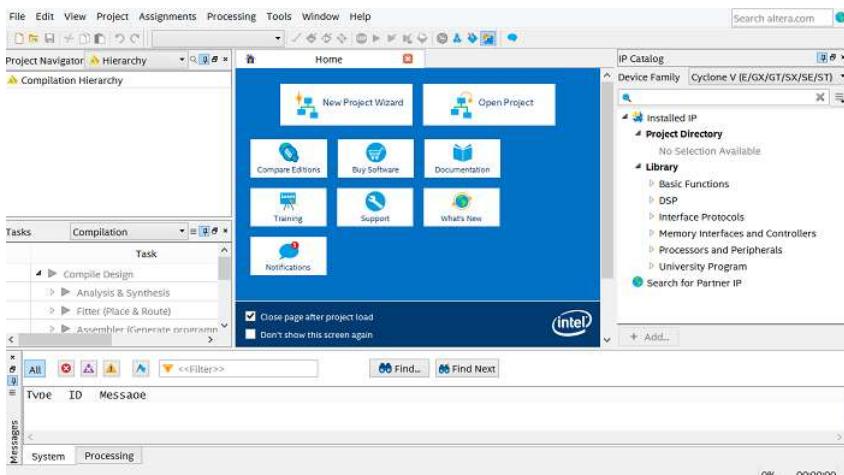
Why is the Quartus download so big?

The 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 Quartus Prime 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.

Note: Screenshots are based on the latest release v16.1. User experience may vary when using earlier or later versions of Intel® Quartus® software.

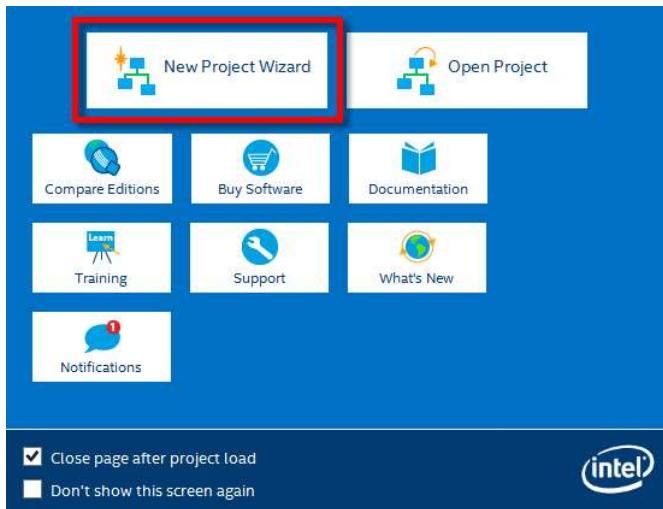
Step 1: Create an Intel® Quartus® Software Project

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



Step 1.b: Open a New Project Wizard

Give Feedback



Step 1.c: Select **Next**

Introduction

The New Project Wizard helps you create a new project and preliminary project settings, including the following:

- Project name and directory
- Name of the top-level design entity
- Project files and libraries
- Target device family and device
- EDA tool settings

You can change the settings for an existing project and specify additional project-wide settings with the Settings command (Assignments menu). You can use the various pages of the Settings dialog box to add functionality to the project.

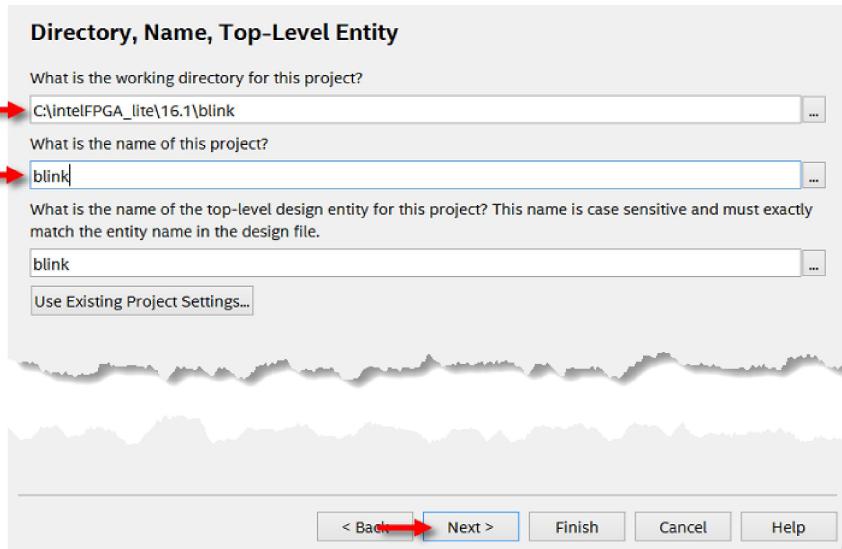
Don't show me this introduction again

[**< Back**](#) [Next >](#) [Finish](#) [Cancel](#) [Help](#)

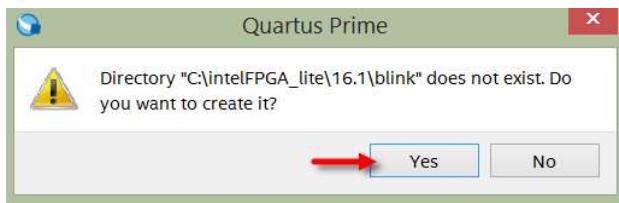
Step 1.d: Directory, Name, Top-Level Entity

Choose a directory to put your project under. Here, we name our project "Blink" and place it under the intelFPGA_lite folder but you can place it wherever you want. Select **Next**.

[Give Feedback](#)



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



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?

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

- Don't put projects within the Quartus tool directory. New Quartus 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.
- 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.

Step 1.e: Project Type

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

[Give Feedback](#)

Project Type

Select the type of project to create.

Empty project

Create new project by specifying project files and libraries, target device family and device, and EDA tool settings.

Project template

Create a project from an existing design template. You can choose from design templates installed with the Quartus Prime software, or download design templates from the [Design Store](#).

< Back  Next > Finish Cancel Help

Step 1.f: Add Files

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

Add Files

Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.

Note: you can always add design files to the project later.

File name: ... Add

File Name	Type	Library	Design Entry/Synthesis Tool	HDL Version

Add All Remove Up Down Properties

Specify the path names of any non-default libraries. [User Libraries...](#)

< Back  Next > Finish Cancel Help

Step 1.g: Family, Device, and Board Settings (Note: You may need to expand window to view more device names)

Select the following:

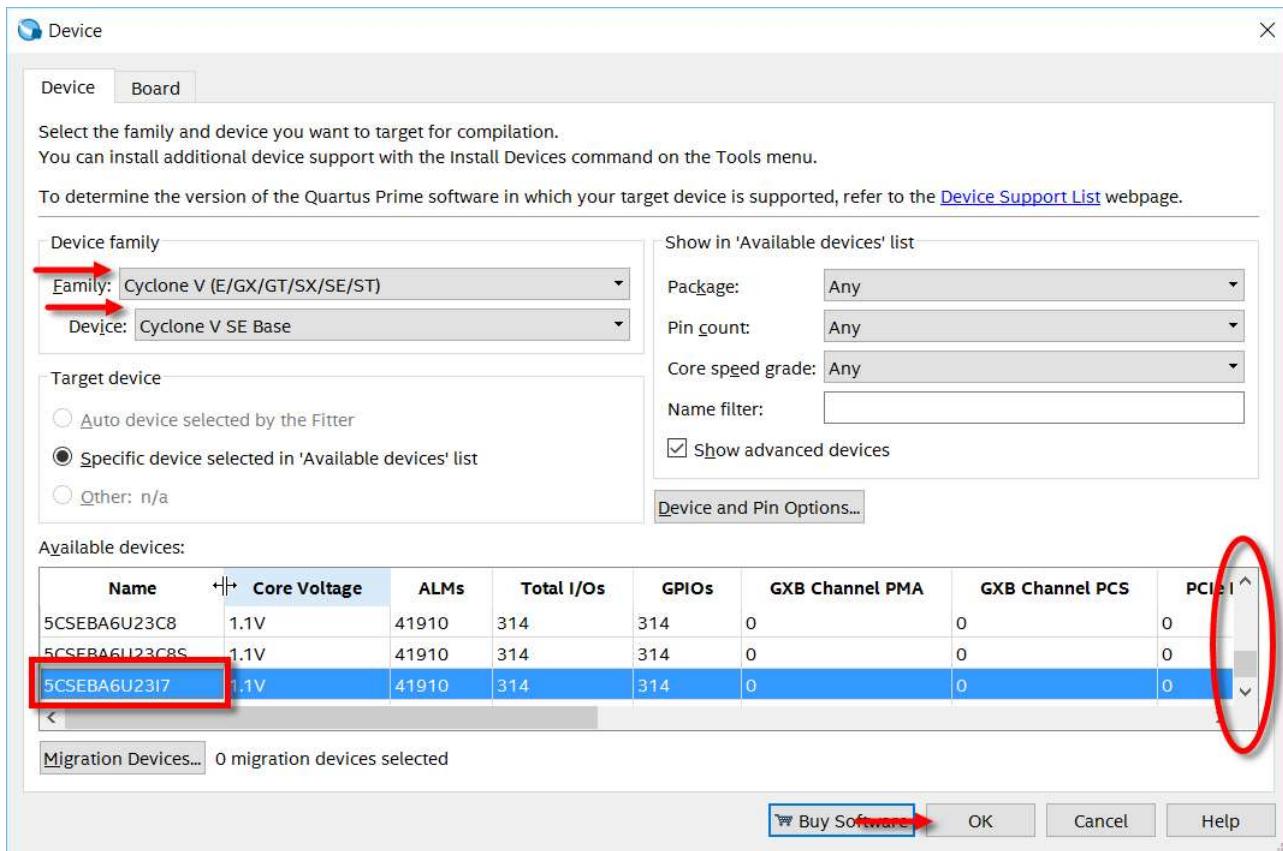
Family: Cyclone V

Device: Cyclone V SE Base

Device name: 5CSEBA6U23I7

Note: The specific device you will need to click the up/down arrows to scroll through the list of supported devices until you find 5CSEBA6U23I7. You

may also need to expand the Name field to see the full device name.



Click **Next**.

Step 1.h: EDA Tool Settings

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

[Give Feedback](#)

EDA Tool Settings

Specify the other EDA tools used with the Quartus Prime software to develop your project.

EDA tools:

Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/Synthesis	<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>	

< Back  Next > Finish Cancel Help

Step 1.i: Summary

Click **Finish**.

Summary

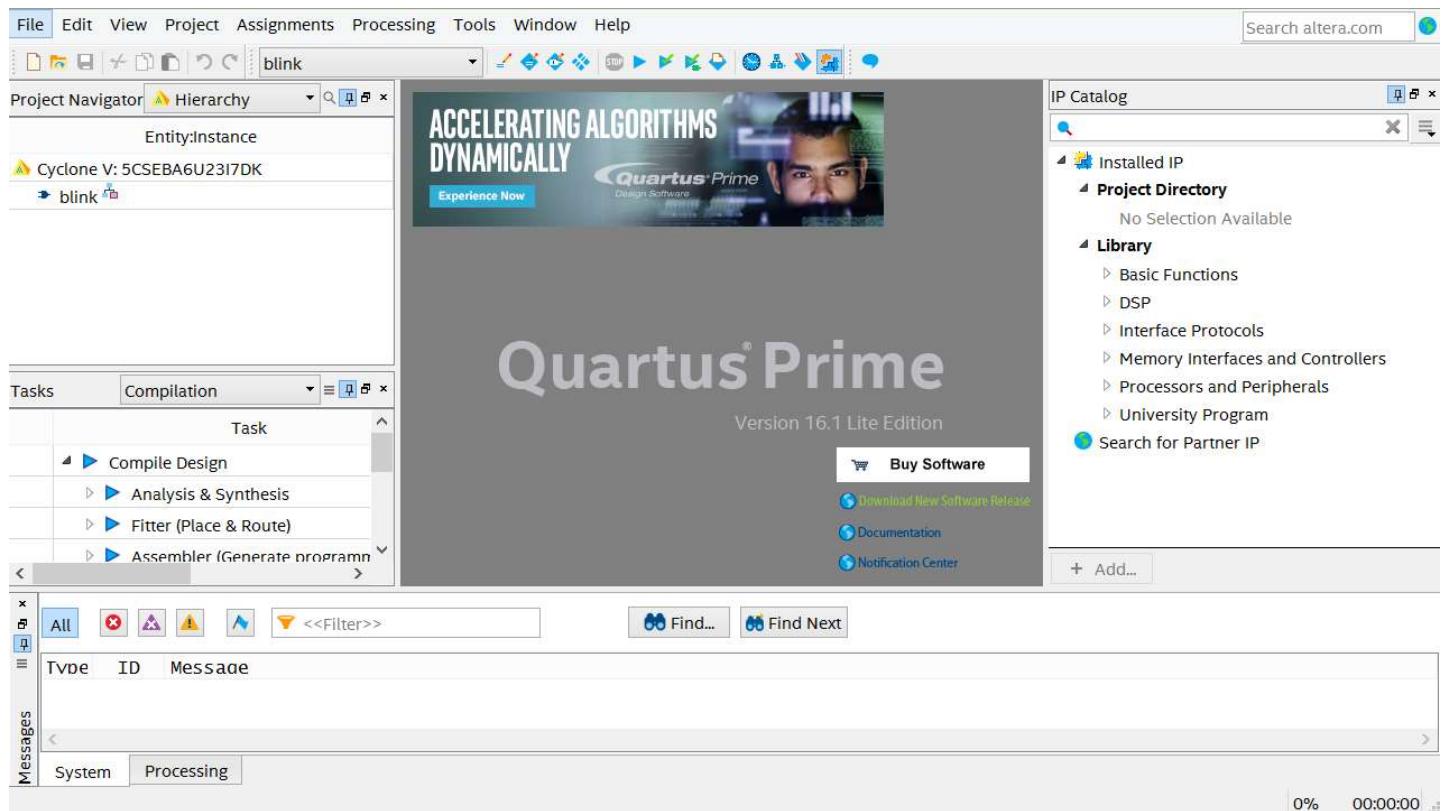
When you click Finish, the project will be created with the following settings:

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

< Back  Next  Finish Cancel Help

The following screen displays.

[Give Feedback](#)

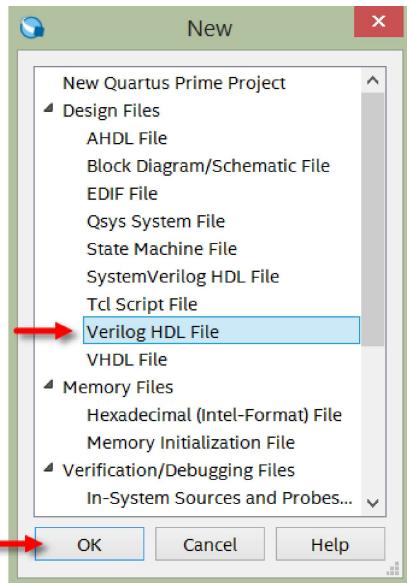


Step 2: Create an HDL File

Hardware Description Language (HDL)

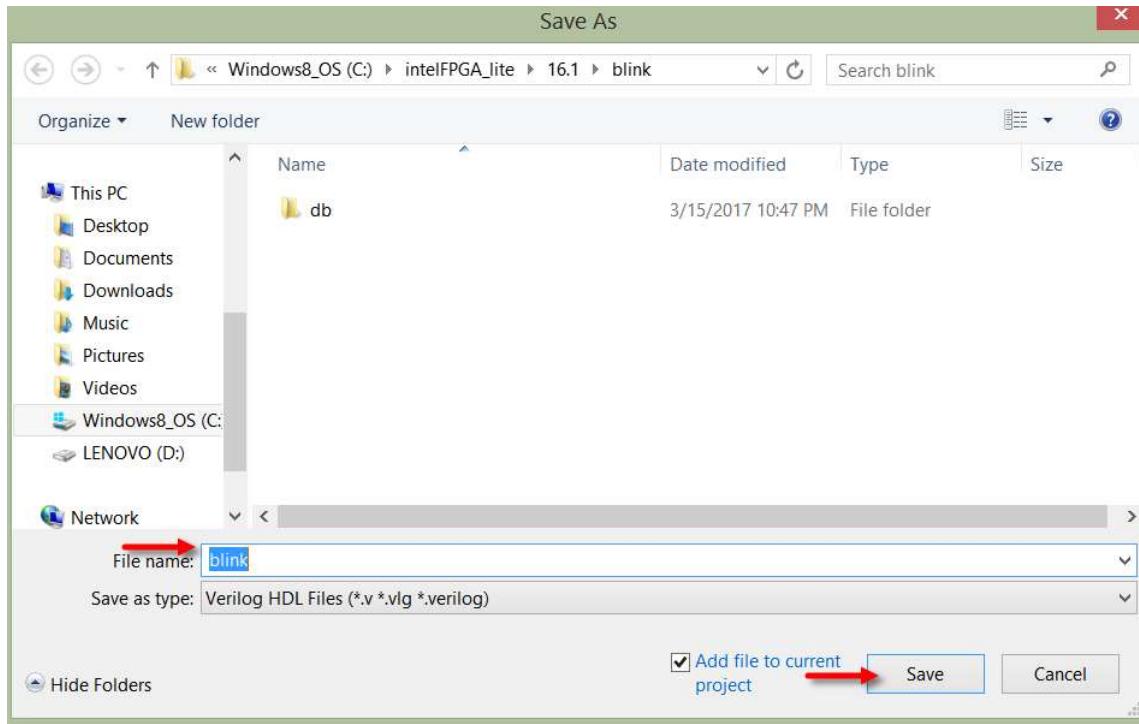
We use Verilog as the HDL. If you are familiar with the C programming language but new to programming in an HDL, Verilog is like C in that you place a semicolon ';' at the end of each statement.

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



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

Step 2.b: Choose **File > Save As**. Choose "blink" for the file name. This is your top-level file name and it must match the name of the project name (blink). Click **Save**.



Step 3: Create a Verilog Module

Step 3.a: Copy and paste this Verilog code into the blink.v window, and then save the file.

```
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[31:0] cnt; // 32-bit counter
8
9   initial begin<= 32'h00000000; // start at zero
10
11 end
12
13 always @(posedge clk) begin<= cnt + 1; // count up
14
15 end
16
17 //assign LED to 25th bit of the counter to blink the LED at a few Hz
```

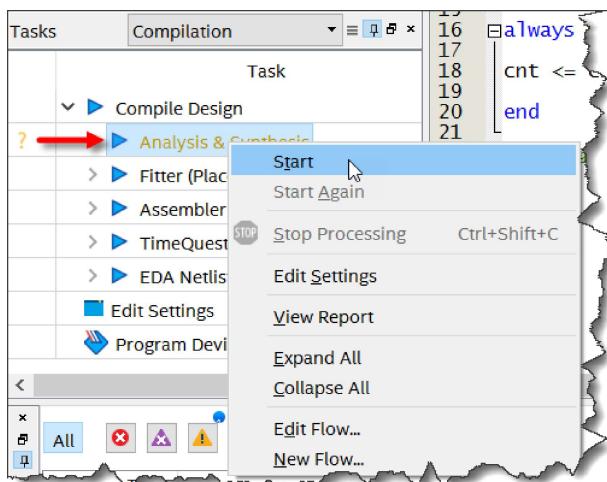
```

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     cnt <= 32'h00000000;    // start at zero
12 end
13
14 always @(posedge clk) begin
15     cnt <= cnt + 1;         // count up
16 end
17
18 //assign LED to 25th bit of the counter to blink the LED at a few Hz
19 assign LED = cnt[24];
20
21 endmodule

```

Step 3.b: Analysis & Synthesis

Right click on Analysis and Synthesis", and then click Start to perform a syntax check and synthesis of the Verilog code.



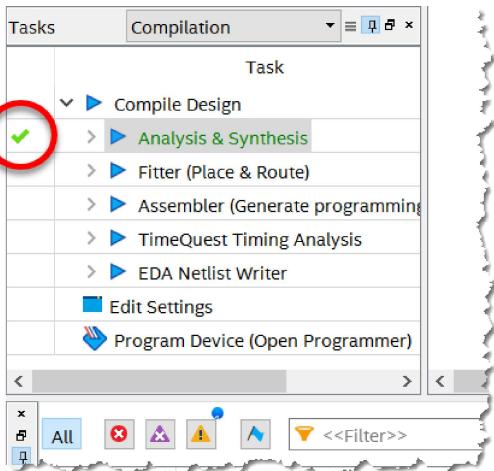
What happens during analysis and synthesis?

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

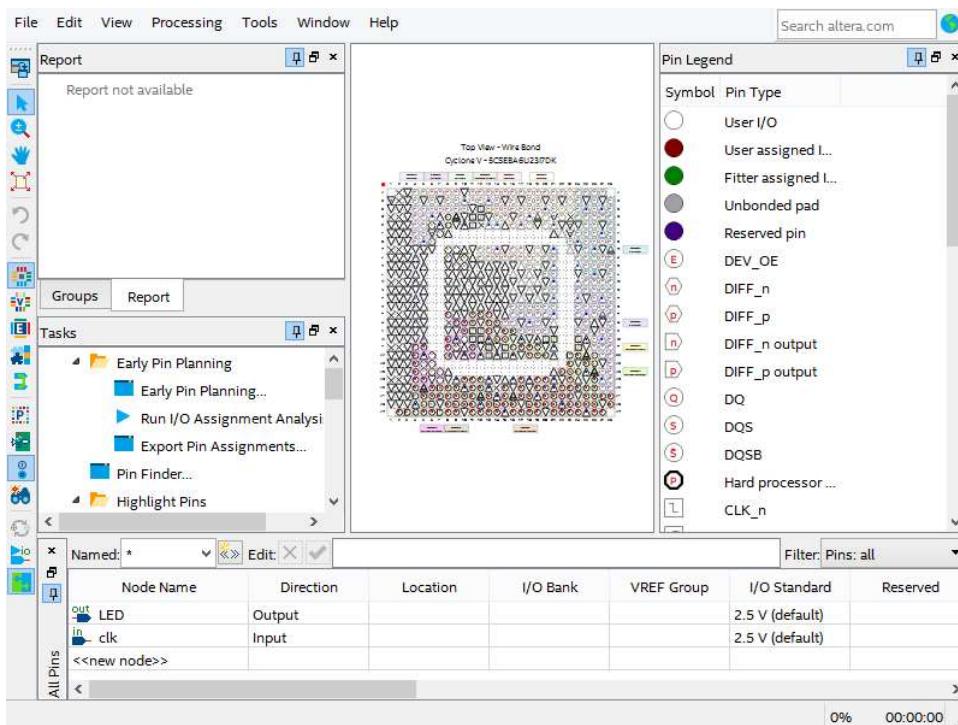
If the process completes successfully, a green check mark displays next to Analysis & Synthesis. If you get an error, check your syntax and make sure it matches exactly the code block provided above. Be sure you used the "View Source" button when you copied the text in step 3.a above.

[Give Feedback](#)



Step 4: Choose Pin Assignments

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

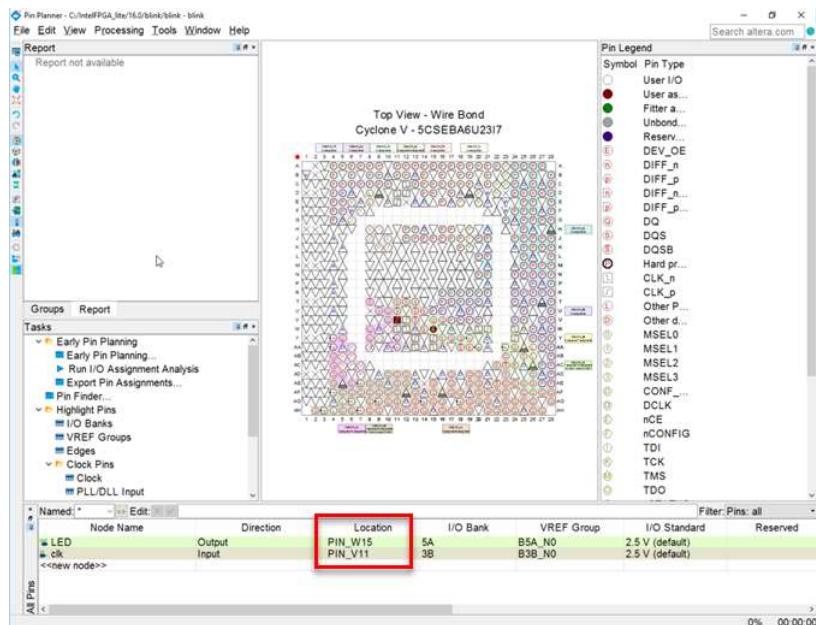


Notice that the input (clk) and output (LED) are listed under **Node Name**. This is because you ran the Analysis & Synthesis.

Step 4.b: 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

[Give Feedback](#)



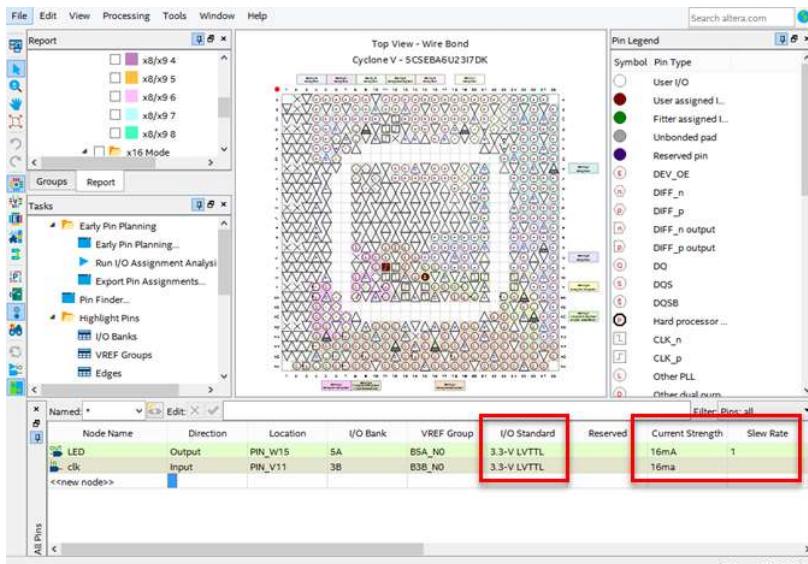
You also need to change the **I/O standard**, **Current Strength** and **Slew Rate** columns from their default settings to the values shown below:

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 4.c: 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 4.d: 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 4.e: Next change **Slew Rate** for the LED output from **1(default)** to **1(fast)**. You don't need to set a Slew Rate for the clk. Leave that blank.



Step 4.f: Close the Pin Planner.

Give Feedback

Step 5: Create an SDC File

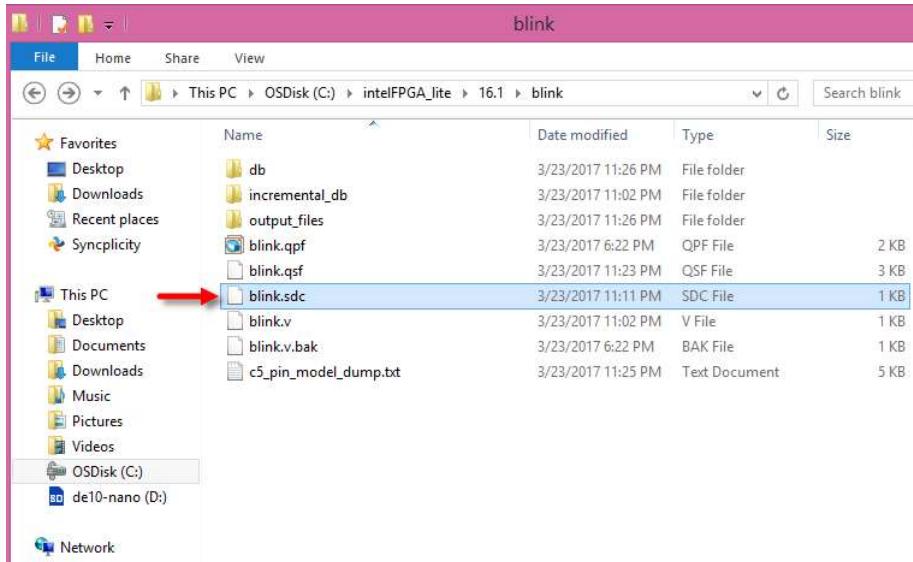
Before you compile the Verilog code, you need to provide timing constraints for the design. You'll create an SDC (Synopsys Design Constraints) file that contains commands to let the Intel® Quartus® software know how to close timing on the design. Without it, you will get warning messages in the compile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

To create a blink.sdc and add it to the blink file directory, do the following.

Copy and paste the following content into a text editor (e.g. notepad):

```
1 # inform quartus that the clk port brings a 50MHz clock into our design so
2     # that timing closure on our design can be analyzed
3
4 create_clock -name clk -period "50MHz" [get_ports clk]
5
6 # inform quartus that the LED output port has no critical timing requirements
7     # its a single output port driving an LED, there are no timing relationships
8     # that are critical for this
9
10 set_false_path -from * -to [get_ports LED]
```

Now save the file as a .sdc file. Place in the blink file directory – same directory where our verilog file (.v file extension) lives.



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 Quartus to verify that the system generated meets its timing requirements.

During synthesis of your FPGA, a design tool called TimeQuest is called by Quartus 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.

Step 6: Compile the Verilog Code

Step 6.a:

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 have a handful of code lines, the compilation should only take a couple of minutes to complete.

What happens during the fitter (place & route)?

Give Feedback

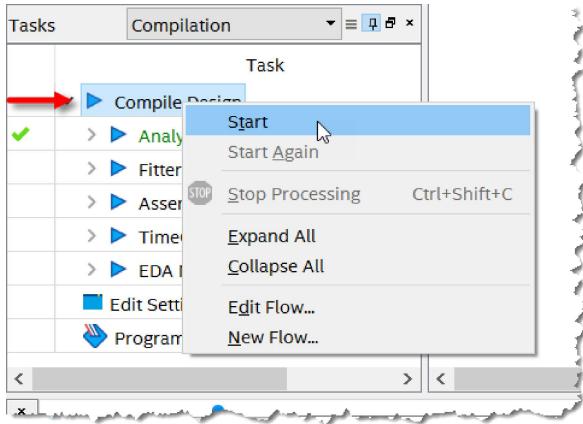
The Fitter places and routes the logic of your synthesized design into target device resources. Think of it like a router that lays out a printed circuit board, connecting the various devices together using copper traces. In this case, however, the devices are logic resources (e.g. lookup tables, registers,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



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

Flow Status	Value
Flow Status	Successful - Thu Jun 15 08:36:43 2017
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	blink
Top-level Entity Name	blink
Family	Cyclone V
Device	5CSEBA6U23I7DK
Timing Models	Final
Logic utilization (in ALMs)	13 / 41,910 (< 1 %)
Total registers	25
Total pins	2 / 314 (< 1 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSS	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSS	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

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 make the SD Card?

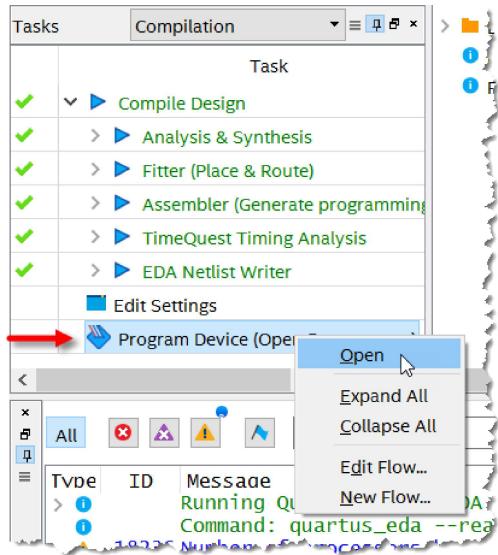
The default behavior of the DE10-Nano kit is to boot from the SD Card. The processor boots, then configures the FPGA under software control. If you leave the SD Card plugged into the board and then reprogram the FPGA, a watchdog timer in the processor would eventually timeout since the FPGA can no longer 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 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.

Step 7.a: Connect the board to your computer via the USB blaster 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 port (J13) on the Terasic DE10-Nano board and the Type-A end into a standard USB port on your host computer.

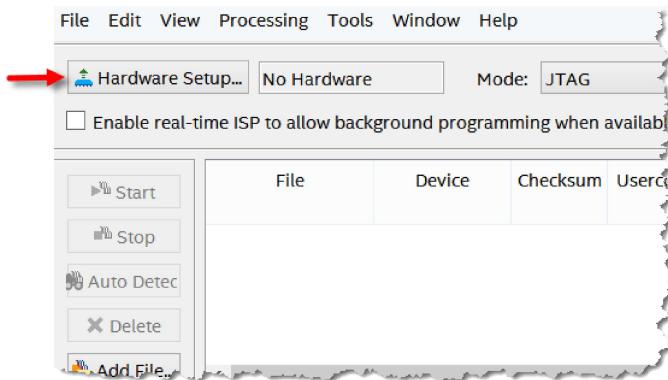
Step 7.b: Connect the board to power and verify there is a blue LED lit near the J13 USB Blaster II port.

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



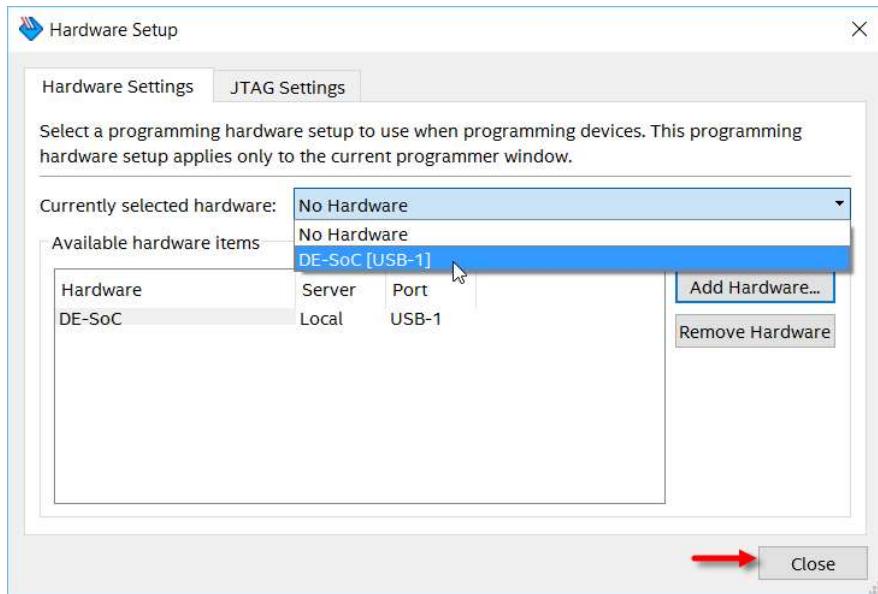
Step 7.d: Hardware Setup

Select **Hardware Setup**.

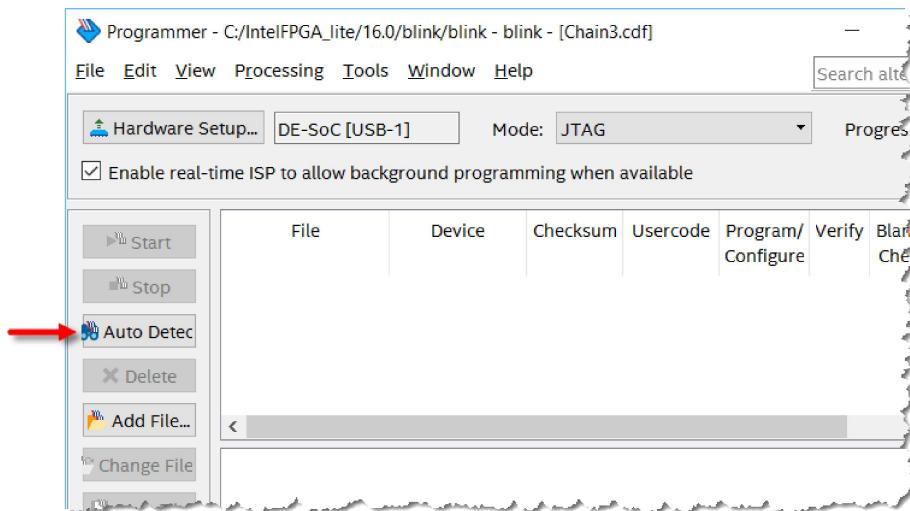


Under the drop down for **Currently selected hardware**, choose **DE-SoC**, then click **Close**.

[Give Feedback](#)

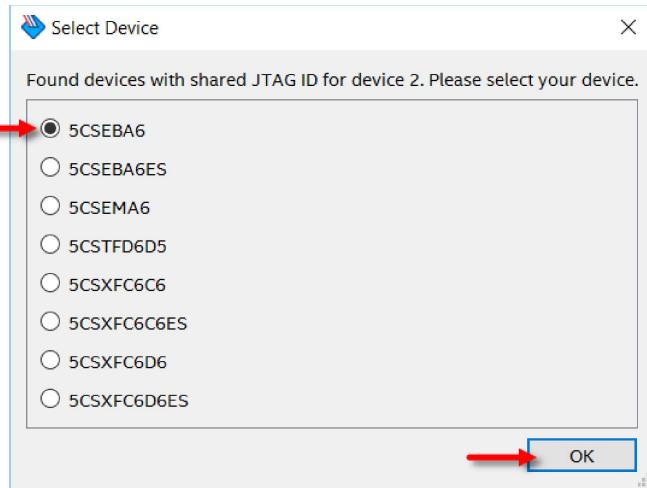


Step 7.e: Click **Auto Detect** to identify the JTAG chain on the board.



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

[Give Feedback](#)

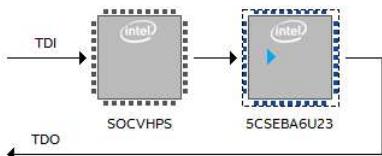


Step 7.f: Add the .sof file.

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

Why are there two devices found in the JTAG chain?

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



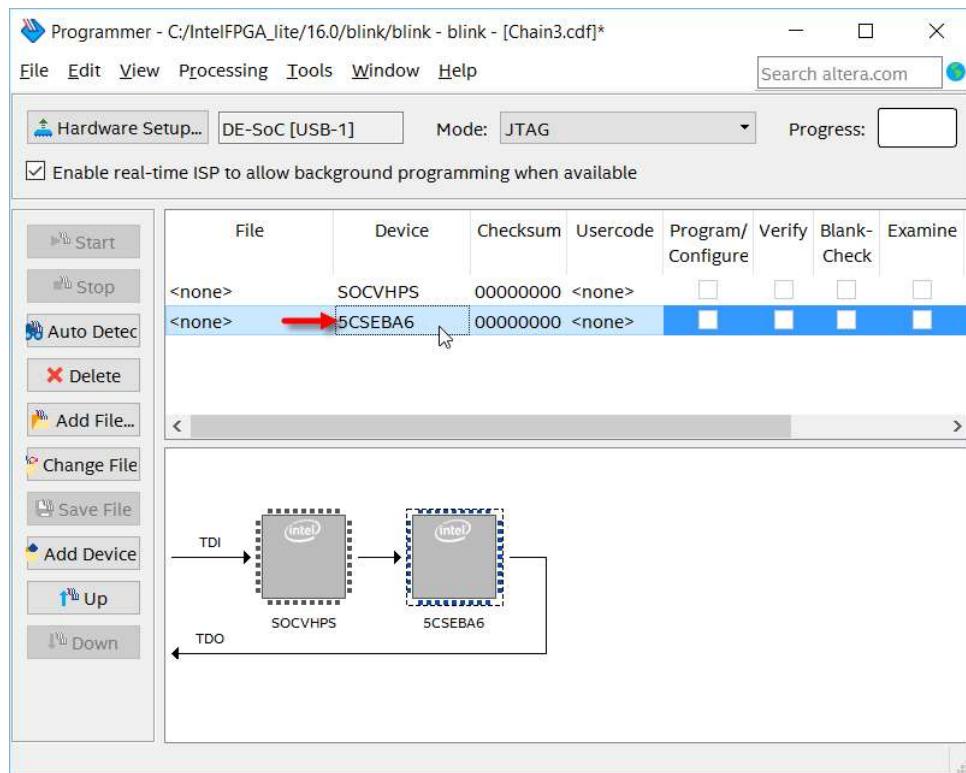
The FPGA JTAG chain is used to configure the FPGA logic, and for hardware debugging using one of several tools such as:

- SignalTap II
 - System Console

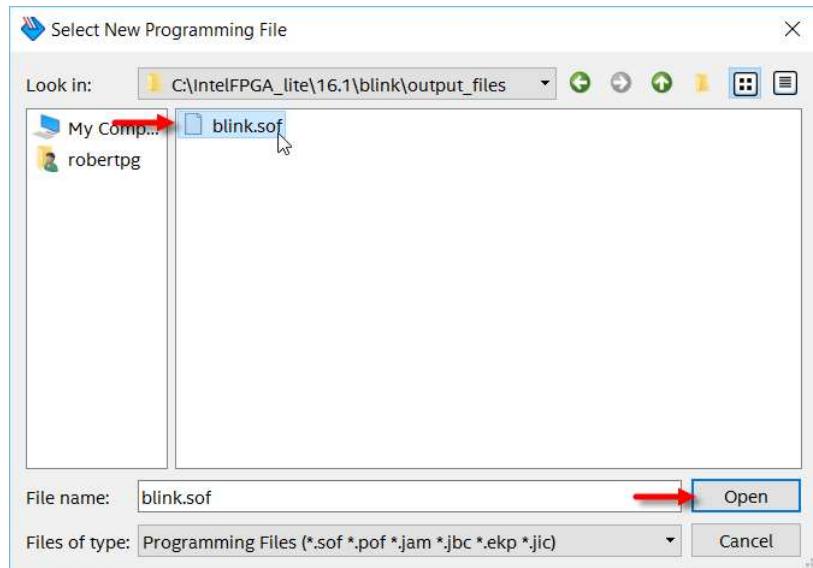
logic analyzer

system-level debugger

The HPS JTAG chain is primarily used for software development using tools like the ARM® Development Studio 5® (DS-5®).



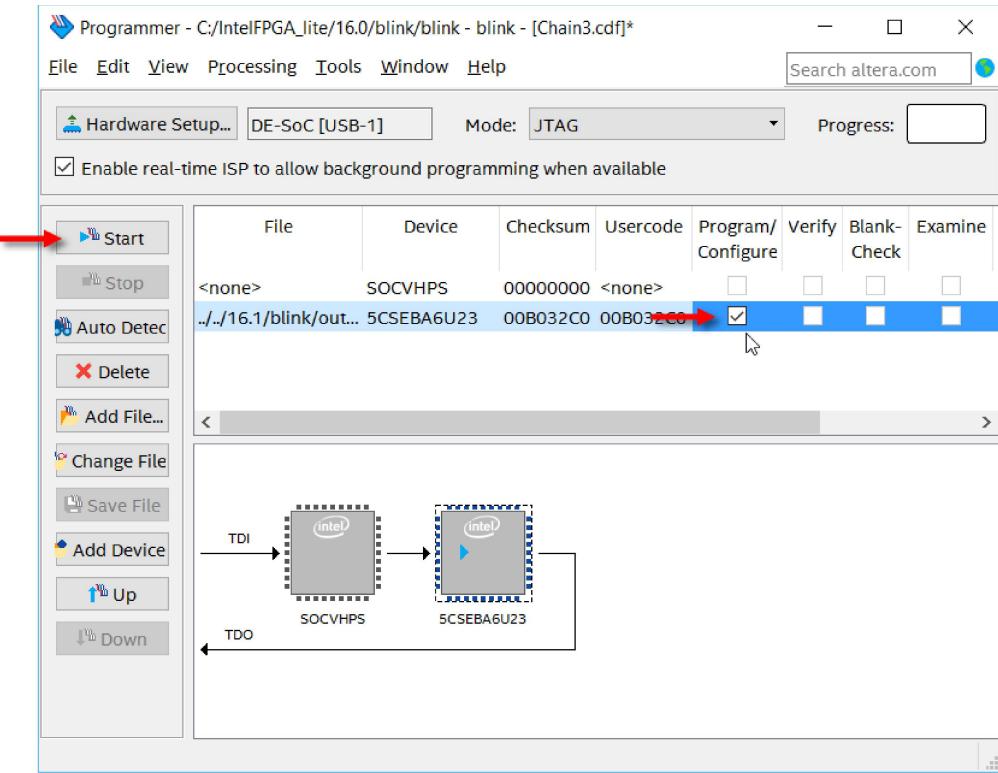
Step 7.g: Navigate to the *output_files* folder, select **blink.sof**, and then click **Open**.



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—using the Intel® Quartus® software Program Device (also called Programmer). The programmer looks into the SOF file and gets the programming bit stream for the device.

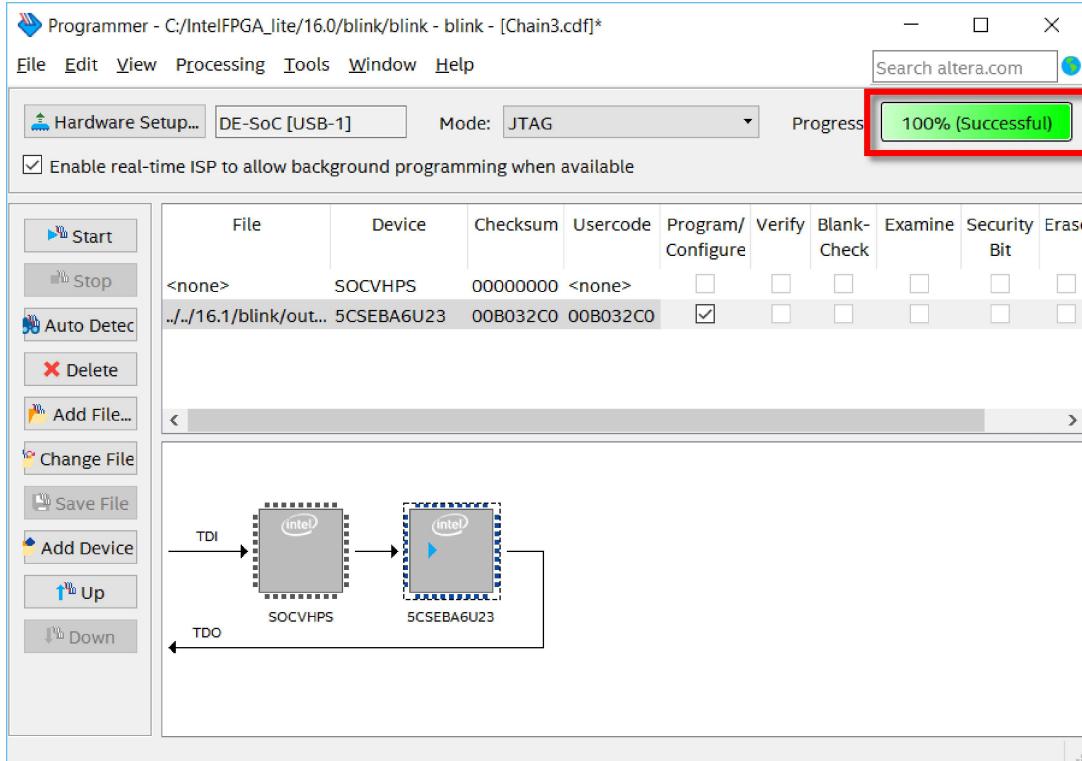
Step 7.h: Check the **Program/Configure** column, and then click **Start**.

[Give Feedback](#)

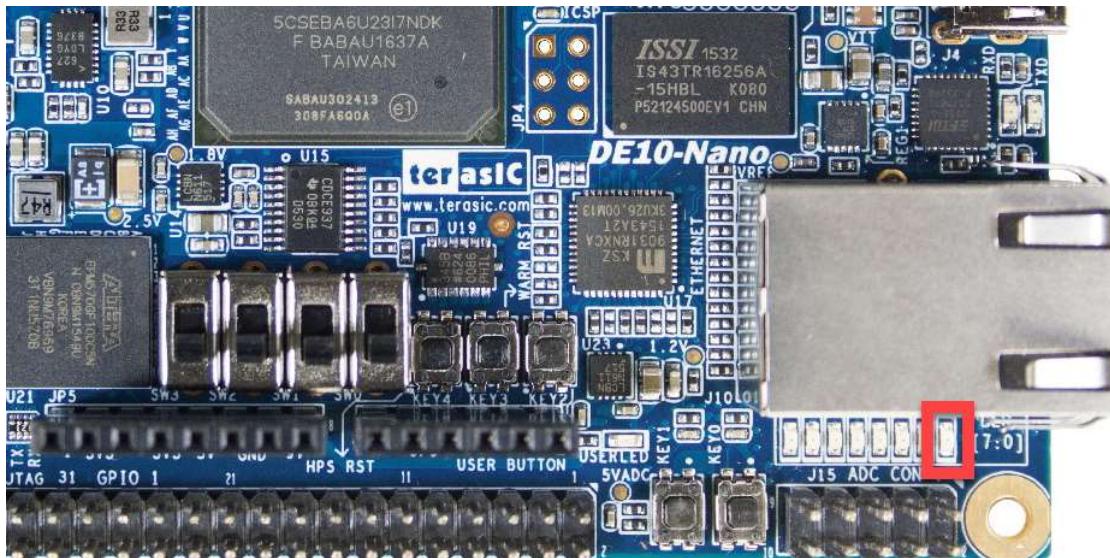


Step 8: Observe the blinking LED

If your progress bar is 100% (Successful), watch LED [7:0] to go through a blinking sequence.



[Give Feedback](#)



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/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 (step 3a):
 - 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 a Verilog file that contains these changes?

Right here:
[Give Feedback](#)

```

// 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

```

2. Run Analysis and Synthesis (step 3b)

3. Assign the LEDs outputs to pins using the Pin Planner (steps 4b - 4e).

Be sure to set the proper I/O standard, drive strength, and slew rate. The I/O pin 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 (step 6a)

5. Reprogram the device (step 7h)

[Get Started](#)

[Related](#)

[Learn More](#)

[Product and Performance Information](#)

¹ Performance varies by use, configuration and other factors. Learn more at www.intel.com/PerformanceIndex

[Company Overview](#)

[Contact Intel](#)

[Newsroom](#)

[Investors](#)

[Careers](#)

[Corporate Responsibility](#)

[Give Feedback](#)

[Diversity & Inclusion](#)

[Public Policy](#)

[Recycling](#)



© Intel Corporation

[Terms of Use](#)

[*Trademarks](#)

[Cookies](#)

[Privacy](#)

[Supply Chain Transparency](#)

[Site Map](#)

[Do Not Share My Personal Information](#)

Intel technologies may require enabled hardware, software or service activation. // No product or component can be absolutely secure. // Your costs and results may vary. // Performance varies by use, configuration and other factors. // See our complete legal [Notices and Disclaimers](#)

// Intel is committed to respecting human rights and avoiding complicity in human rights abuses. See Intel's [Global Human Rights Principles](#). Intel's products and software are intended only to be used in applications that do not cause or contribute to a violation of an internationally recognized human right.



[Give Feedback](#)