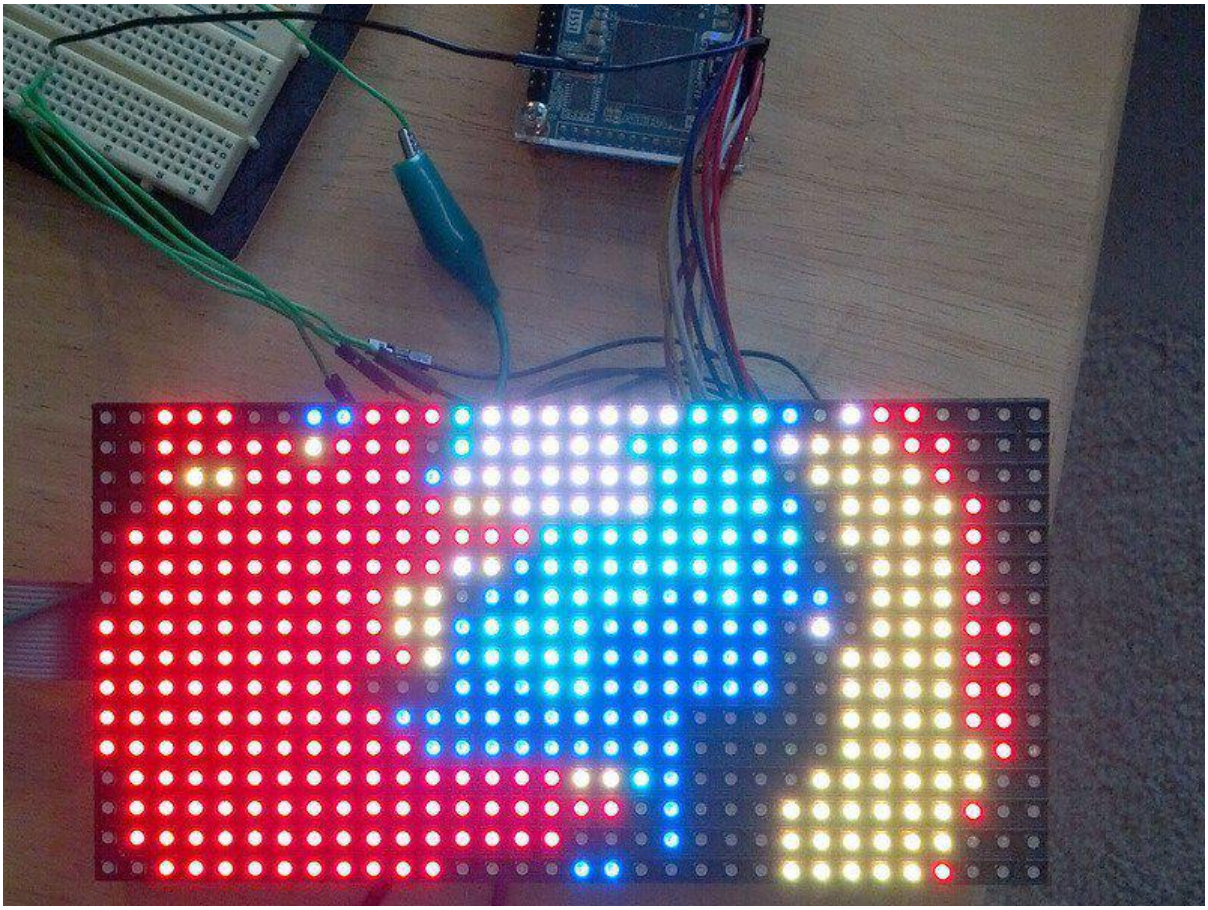
FPGA RGB Matrix

Created by lady ada

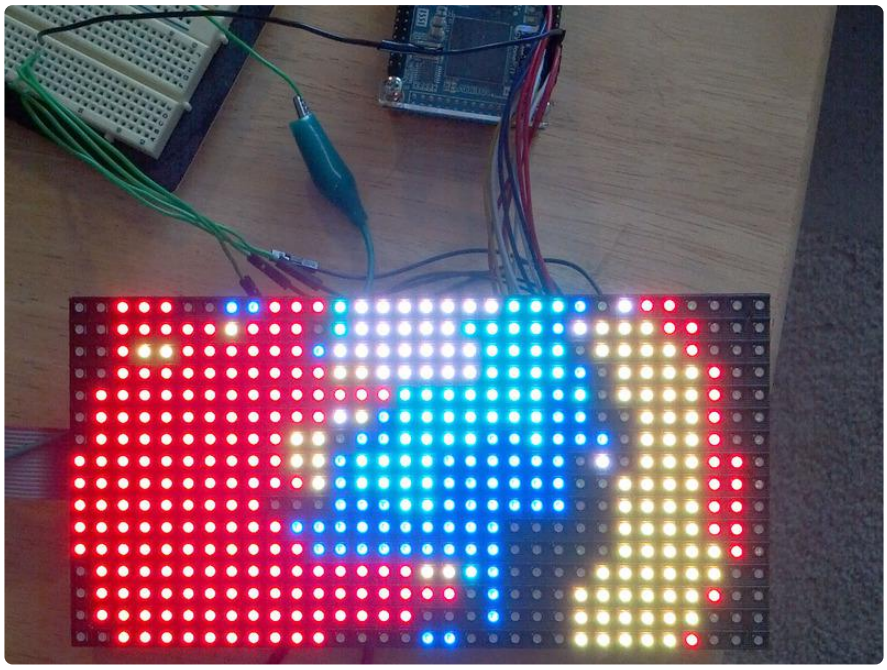


https://learn.adafruit.com/fpga-rgb-matrix

Last updated on 2021-11-15 05:49:11 PM EST

Table of Contents

Overview



Controlling the Adafruit 32x16 RGB LED Matrix with a DE0-Nano FPGA Board

Adafruit currently sells a really cool 16x32 RGB LED matrix panel (http://adafru.it/420) in their store that is "designed to be driven by an FPGA or other high speed processor." The purpose of this tutorial is to help you get started driving a small handful of these displays with the DE0-Nano board (https://adafru.it/aIK), which contains a mid-range Altera FPGA.

Prerequisites

This tutorial is for those who are familiar with electronics, microcontrollers, programming IDEs and noodling around on a windows computer with drivers, command prompts, editing text files, etc. Its a good introductory FPGA project but not a good introductory microcontroller/electronics project.

You need to have the Quartus II software installed on your computer. If not, you can download it from Altera's website (https://adafru.it/aIL) or install it from the DVD that comes with the DE0-Nano board. You will also need to install the USB-Blaster drivers that enable your computer to communicate with the FPGA (see this short YouTube video (https://adafru.it/aIM)).

This tutorial was written by Brian Nez! (https://adafru.it/aIN)

New Project

User configuration

Once you are all set up and ready to begin, download the necessary files for this project from its Github repository (https://adafru.it/diq) (click on the "ZIP" icon).

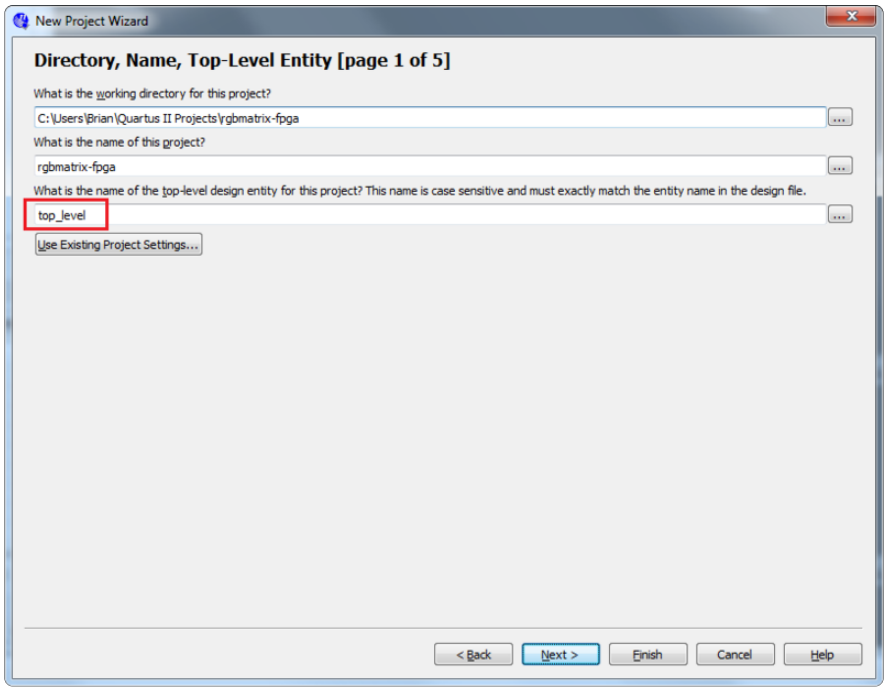
Open the file vhdl/config.vhd in a text editor and change line 32 (constant NUM_PANELS...) to indicate the total number of LED panels you have daisy-chained together in your display. For example, if you are using a 1x2 or 2x1 grid, you will want to change the line to:

```
constant NUM_PANELS : integer := 2
```

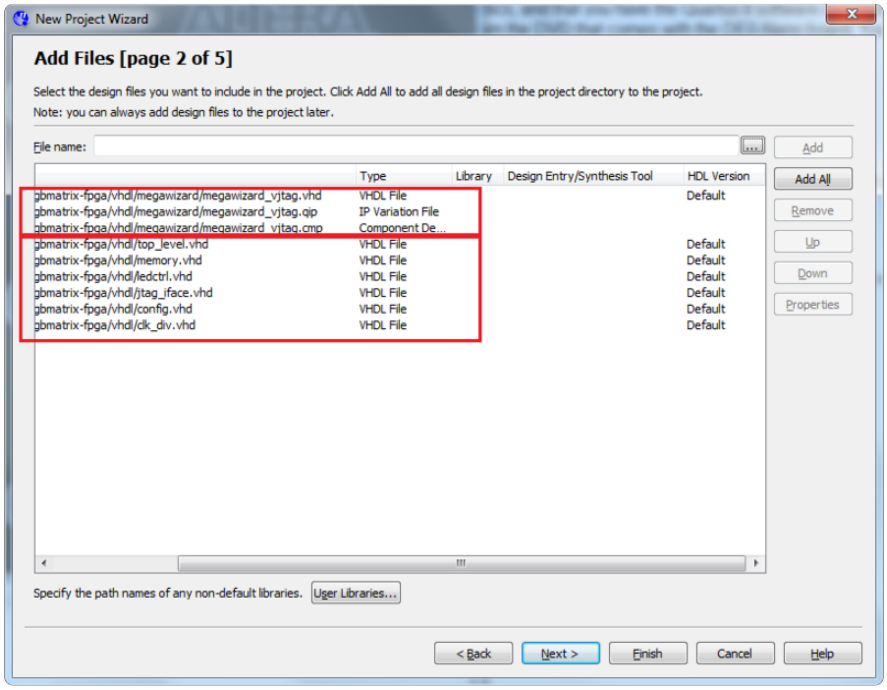
You may optionally edit line 33 (constant PIXEL_DEPTH...) in a similar manner to indicate how many bits-per-pixel you want to use. This will affect the level of brightness control available to you later. Finally, save the file!

Creating the Quartus II project

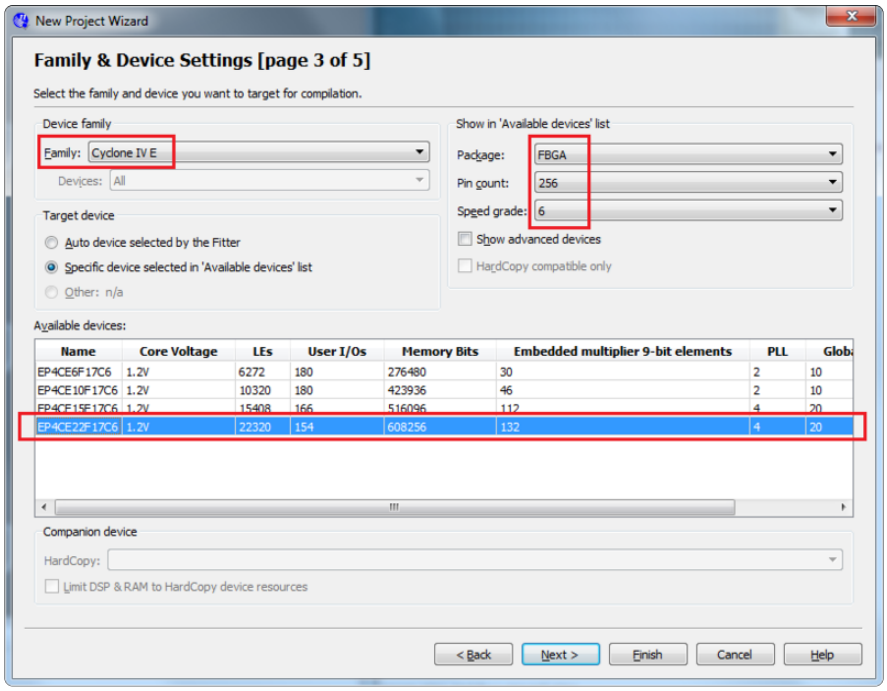
Start Quartus II and open the "New Project Wizard" from the "File" menu. On the first page, name the project rgbmatrix-fpga (or something similar) and name the top-level entity top_level. Click Next.



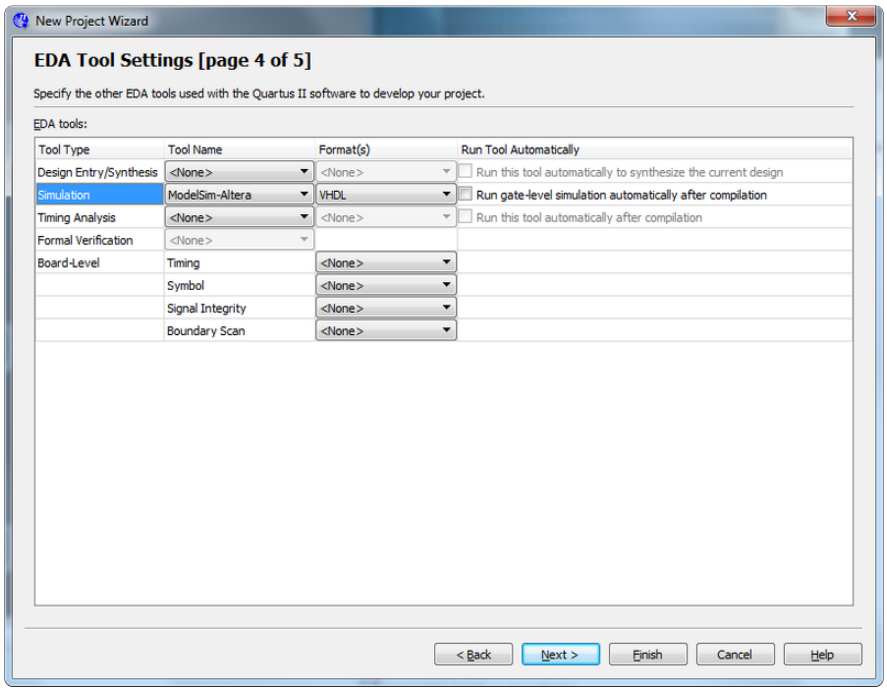
Now we will add the source code files to the project. Click the "..." button to open the file browser and select the .vhd files in the vhdl folder you downloaded earlier (do not include the testbenches directory). Click "..." again and open the megawizard folder. Set the type drop-down menu to "All Files (*.*)" so you can select the .qip, .cmp, and megawizard_vjtag.vhd files (do not include megawizard_vjtag_inst.vhd). Add them to the project and click Next.



The FPGA chip in use on the DE0-Nano is the Cyclone IV EP4CE22F17C6N. You can find it by setting the device family to "Cyclone IV E", package to "FBGA", pin count to 256, and speed grade to 6. Select the chip and click Next.



Set the "Simulation" tool name to "ModelSim-Altera" and the format to "VHDL". Leave everything else as "<None>" and click Next.

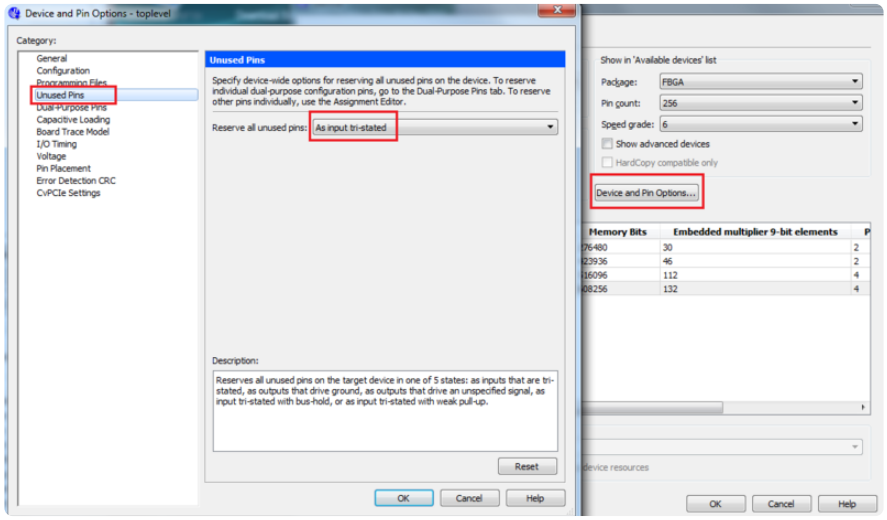


Click Finish to create the project!

Pin settings

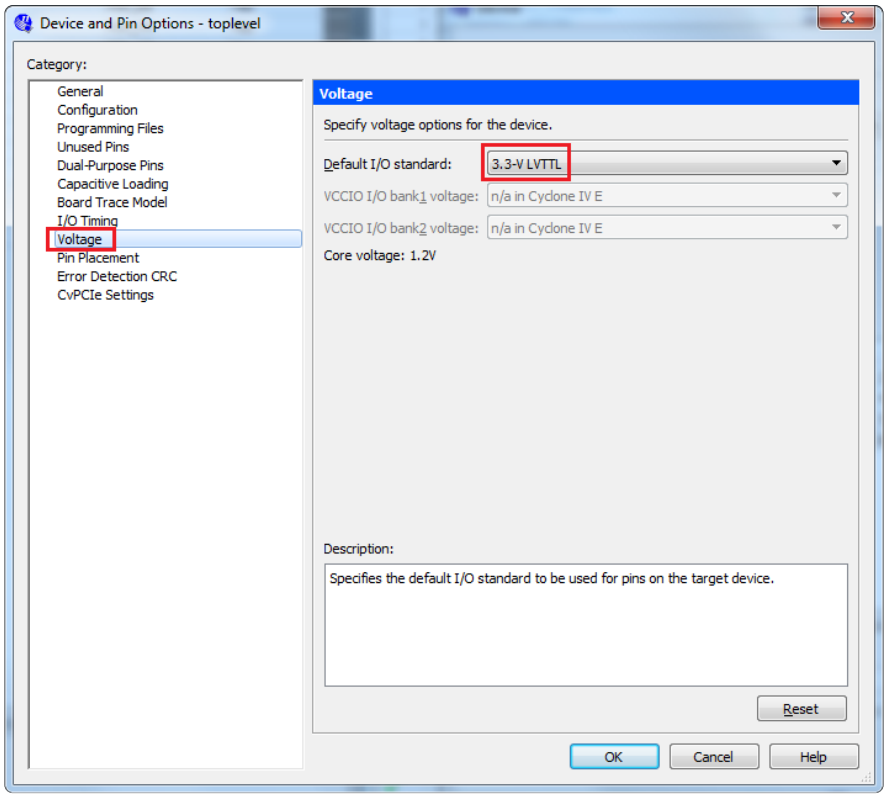
Pin settings

Now that the project has been created, you need to change two more settings before we can move on. Go to the Project Navigator panel in the top left area of Quartus and right click on the device ("Cyclone IV E: ..."). Select "Device" from the menu.



A window will open. Click "Device and Pin Options...". In the left hand side of the new window that comes up, open the "Unused Pins" category. Change the "Reserve all unused pins" settings to "As input tri-stated". This will essentially prevent the unused pins on the FPGA from doing anything unwanted on the DE0-Nano when we program the design.

Now select the "Voltage" category. Change the "Default I/O standard" to "3.3-V LVTTL". This is essential to do because the panels will not recognize a signal below this voltage.

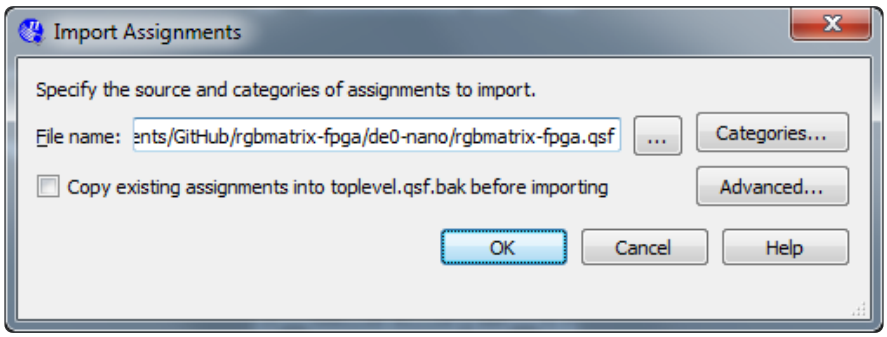


Click OK, then click OK again.

Pin Assignments

Making pin assignments

Go to the "Assignments" menu and select "Import Assignments...". Import the de0-nano/rgbmatrix-fpga.qsf file. After you do this, a message should appear in the "System" console tab at the bottom of Quartus: "Import completed. 14 assignments were written (of of 14 read)."



You can (optionally) customize the pin assignments that were imported by going to the "Assignments" menu and selecting "Assignment Editor". Additional information on the GPIO headers can be found in the DE0-Nano PDF manual (https://adafru.it/aIP)

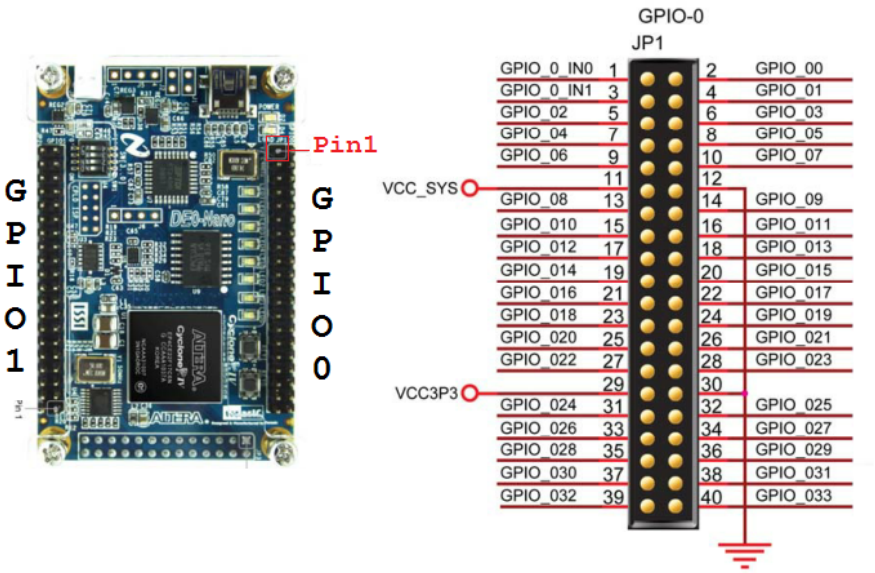
(pages 18-20). A mapping of FPGA pins to GPIO headers can also be found in the de0-nano/DE0_Nano.qsf file (open it with a text editor).

Assignment Editor

<<new>> Filter on node names: * Category: All

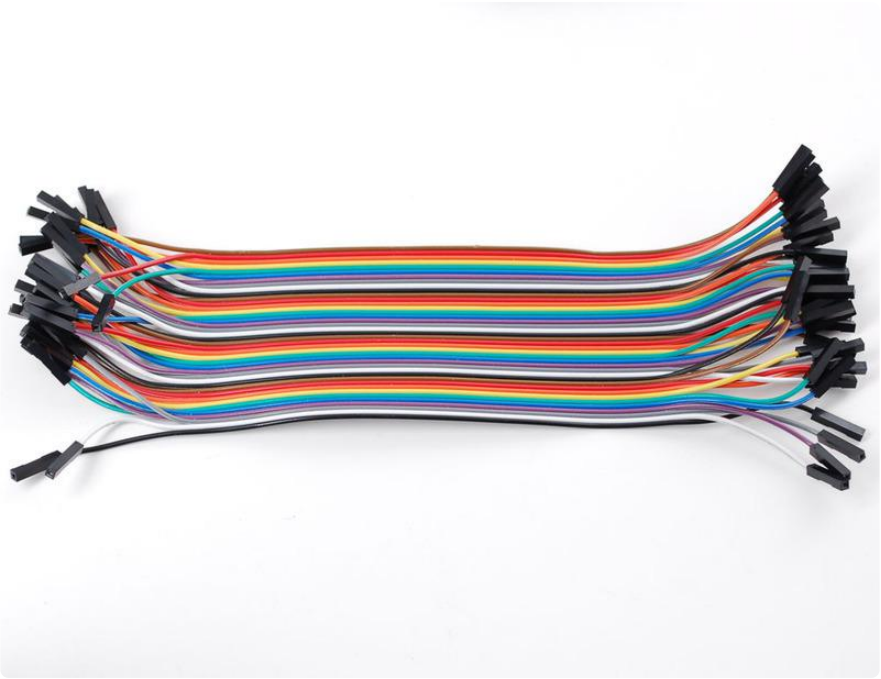
	Status	From	To	Assignment Name	Value	Enabled
1	Ok		lat	Location GPIO 033	PIN_B12	Yes
2	Ok		r1	Location GPIO 018	PIN_E7	Yes
3	Ok		b1	Location GPIO 020	PIN_E8	Yes
4	Ok		r2	Location GPIO 022	PIN_F9	Yes
5	Ok		b2	Location GPIO 024	PIN_C9	Yes
6	Ok		a	Location GPIO 026	PIN_E11	Yes
7	Ok		g1	Location GPIO 027	PIN_E10	Yes
8	Ok		c	Location GPIO 028	PIN_C11	Yes
9	Ok		g2	Location GPIO 029	PIN_B11	Yes
10	Ok		clk_out	Location GPIO 030	PIN_A12	Yes
11	Ok		b	Location GPIO 031	PIN_D11	Yes
12	Ok		oe	Location GPIO 032	PIN_D12	Yes
13	Ok		clk_in	Location CLK 50	PIN_R8	Yes
14	Ok		rst_n	Location KEY 0	PIN_J15	Yes

Save any changes. Now we are ready to connect the pins on the FPGA to the pins on the RGB LED matrix panel!



Please refer to the Adafruit guide for wiring details (https://adafru.it/cl4) on the panel side.

You may want to use female-female jumper wires to make the connections between the IDC pins! (http://adafru.it/266)



Important: DOUBLE-CHECK ALL YOUR CONNECTIONS BEFORE POWERING ON! Be sure the board's orientation matches the diagrams when you connect the wires!

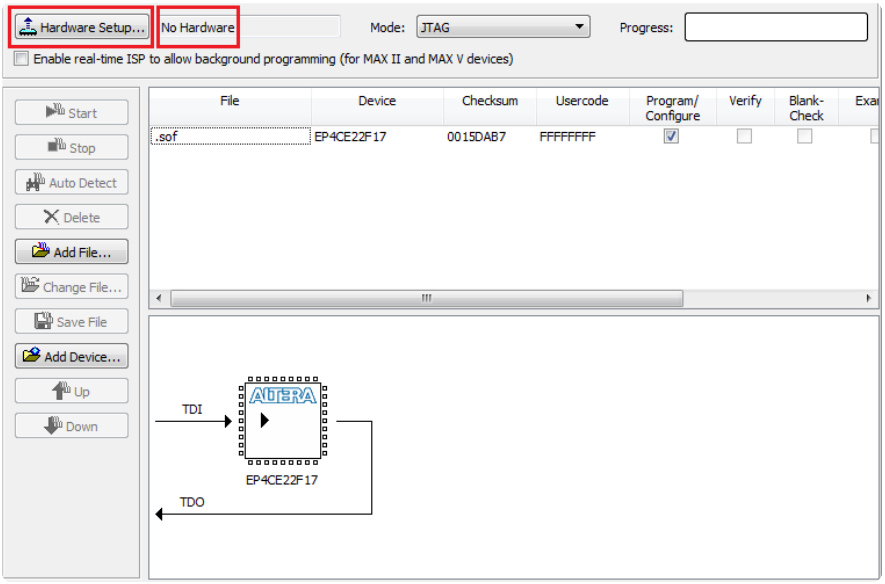
Synthesize and Upload

Synthesizing the design

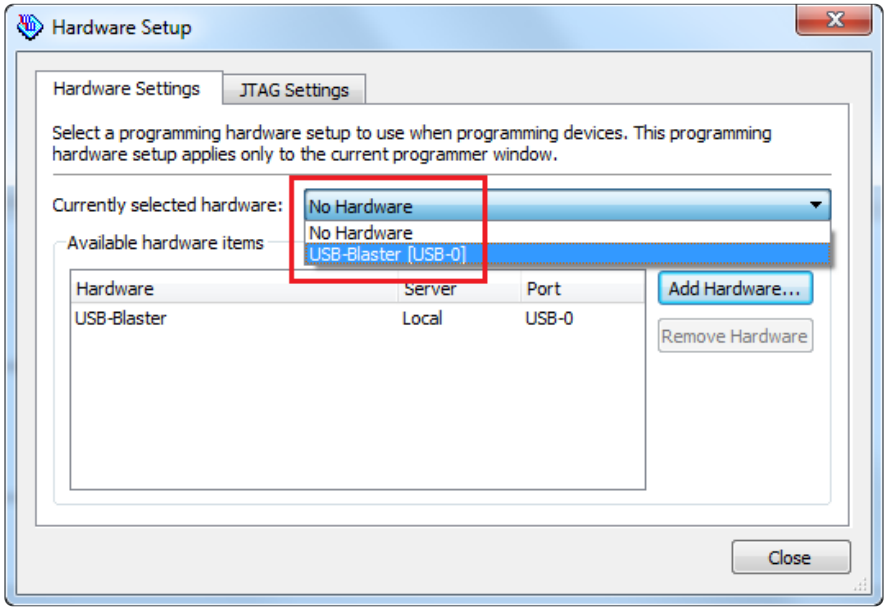
To synthesize the design, go to the "Processing" menu and select "Start Compilation", or click on the purple arrow icon in the toolbar. Synthesis should be quite fast since the design is small. After compilation is successful, you should have a new .sof file in your Quartus project directory. It should be 703,642 bytes long.

Uploading the bitfile

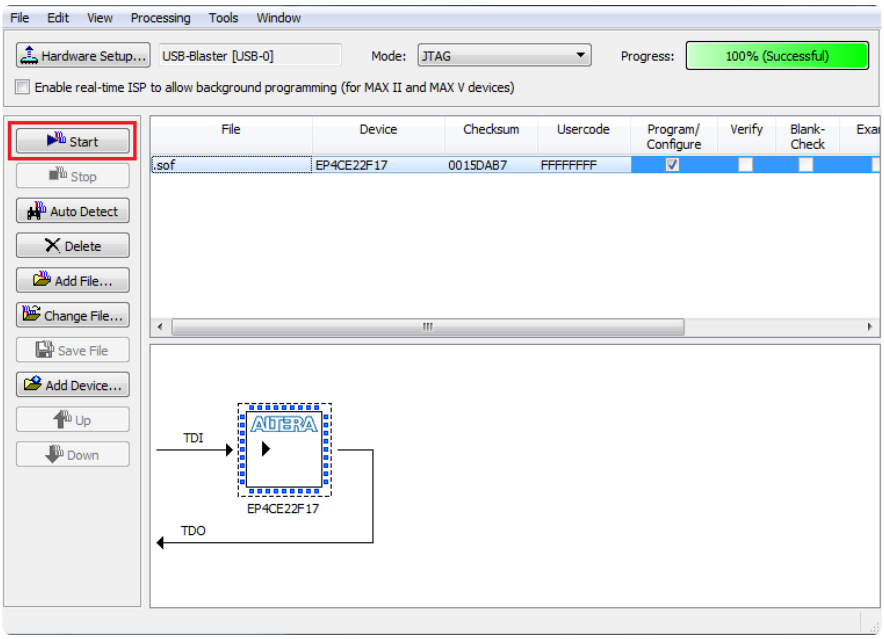
Plug in your DE0-Nano board via the USB connector. Now, go to the "Tools" menu and select "Programmer".



In the top left of the window that appears, you should see "USB-Blaster [USB-0]". If instead you see "No Hardware", click on "Hardware Setup..." and (re-)select your device.



Now, select the .sof file in the list, ensure "Program/Verify" is checked, and click "Start"! This should take about a second.



The FPGA is now programmed with your design! (This only programmed the SRAM though, not the onboard EEPROM — so the design is only stored until power is turned off.)

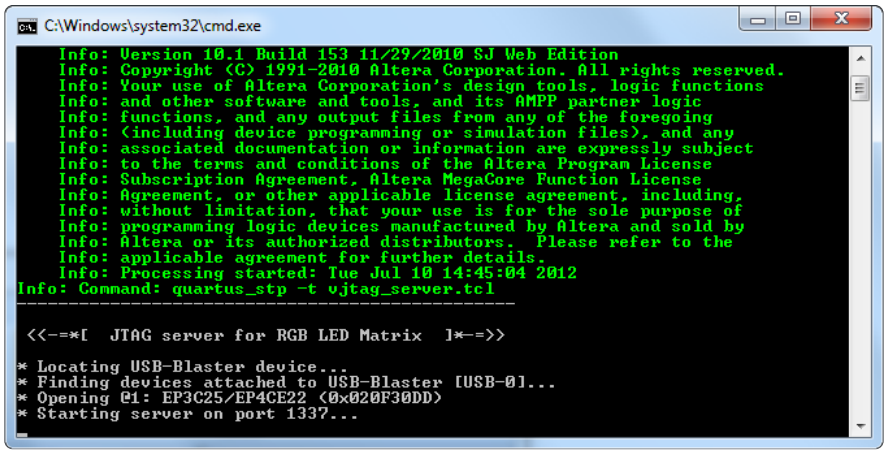
Note: In the future, you can use the command script de0-nano/program.cmd to quickly program the FPGA's SRAM with your .sof file (it uses the Quartus command line programming utility).

Demos

Running the Virtual JTAG interface server

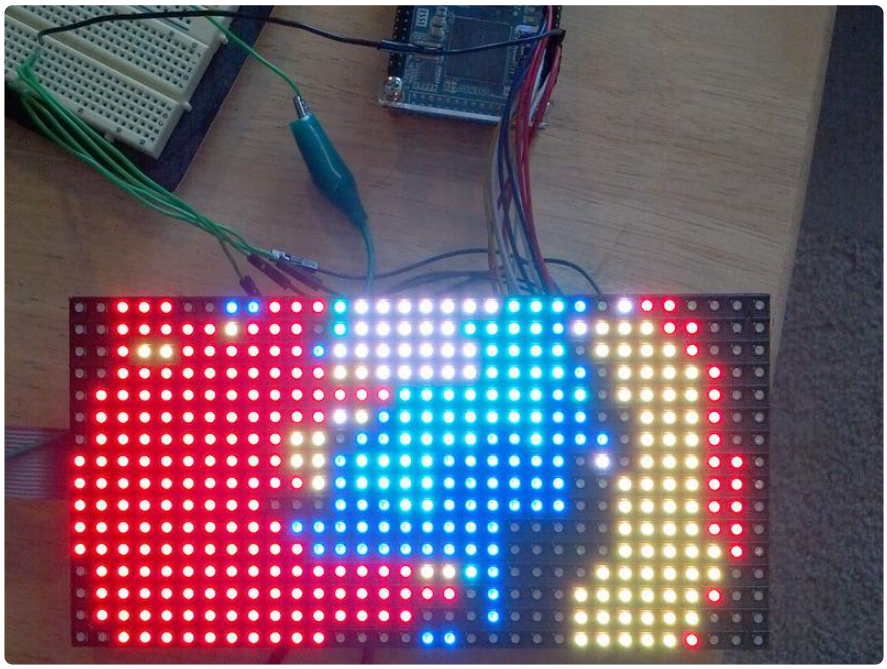
Open the command script tcl/run.cmd in a text editor and ensure that the path to the quartus_stp executable is correct. Then, double click the script to launch the Virtual JTAG interface server (tcl/vjtag_server.tcl). This binds to a TCP port to allow programs and scripts to write data to the FPGA through Altera's Tcl API.

This allows you to send video to the FPGA from any device that can communicate over the network! For example, a remote Arduino with a Wi-Fi shield, or an Android cellphone.



The included Processing demos and code

Two demos written in the Processing programming language (https://adafru.it/aIR) (a dialect of Java) are available in the processing folder. You can run either demo by copying its directory to your local sketchbook folder (usually ~/Processing). The first demo, Chaser is a basic test animation. The second demo, Magnify sends a real-time screen capture to the panels through the FPGA.



More!

Stephen Goadhouse wrote in with an update to this project that eliminates the ghosting effect in the original writeup. Check it out by clicking below

LED_Ctrl_Updates.zip

https://adafru.it/djE