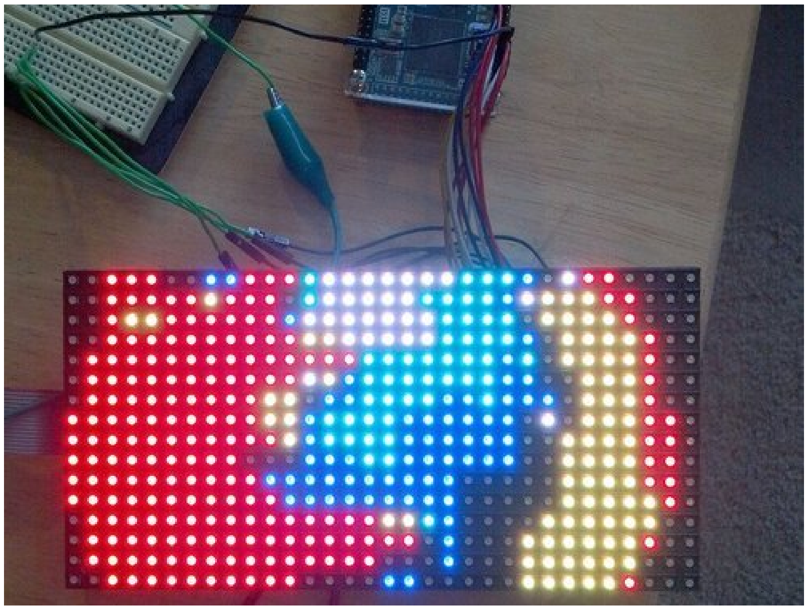
adafruit learning system

FPGA RGB Matrix

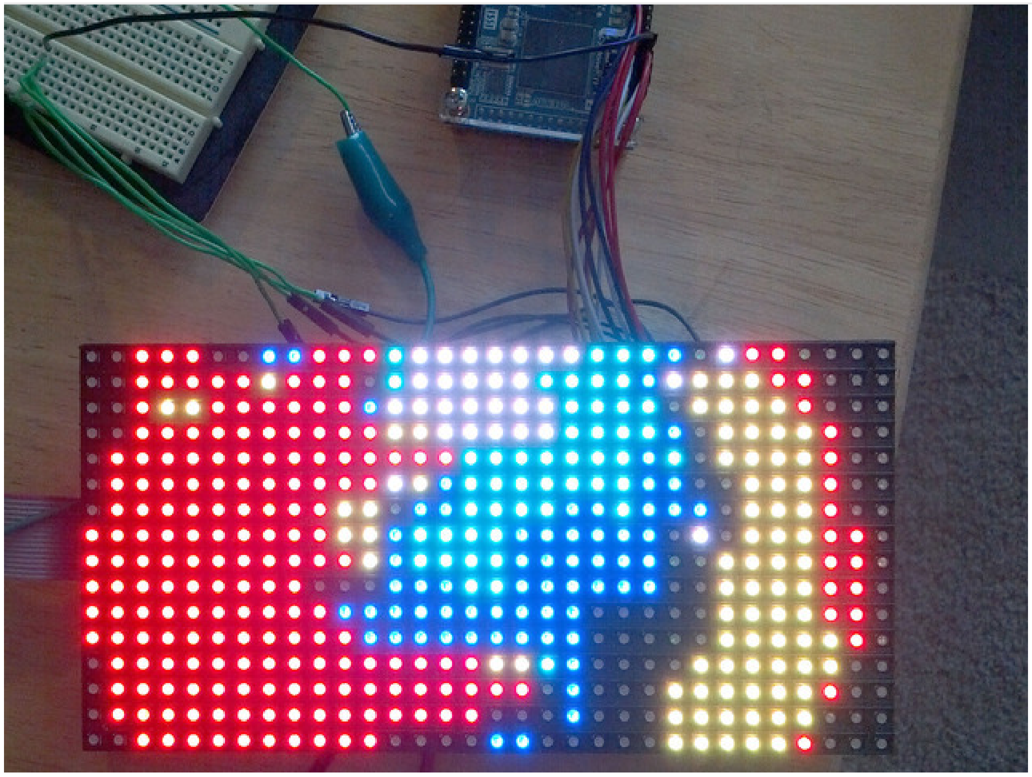
Created by lady ada



Last updated on 2018-08-22 03:30:51 PM UTC

Guide 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/alK), 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/alL) 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/alM)).

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

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).

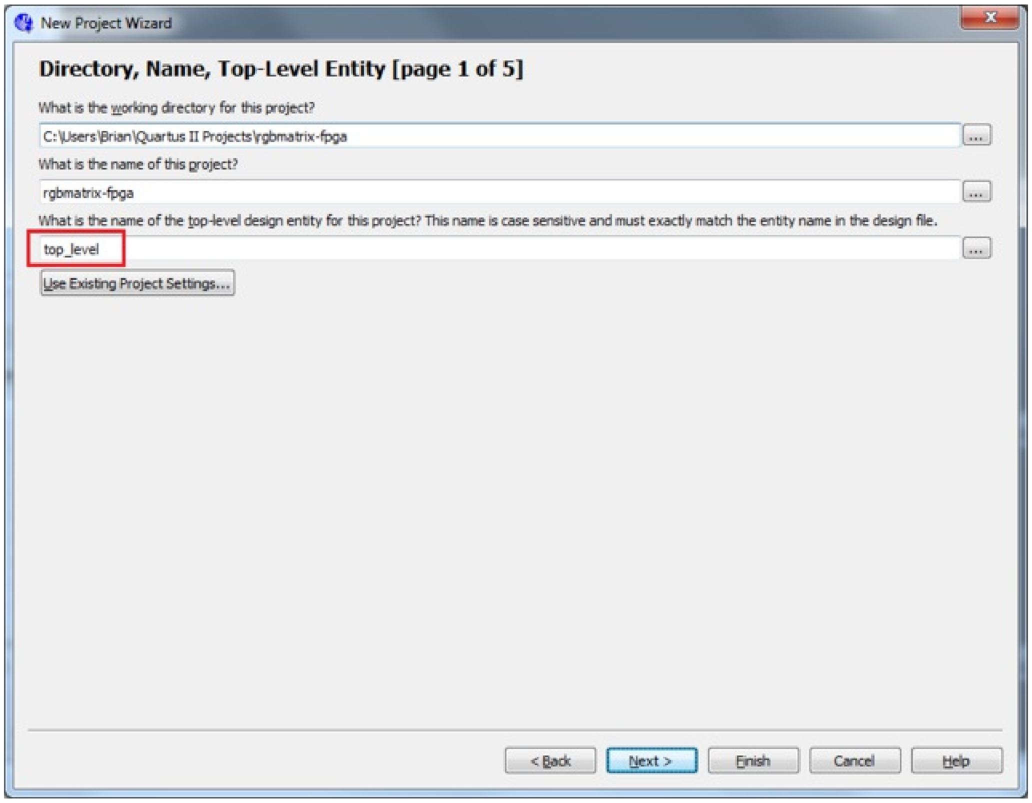
Open the file vhdl/config.vhd in a text editor and change line 32 **(constant NUM_PANELS...)** to indicate the total number of LED panels you have daisy-chained together in your display. For example, if you are using a 1x2 or 2x1 grid, you will want to change the line to:

> **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.


New Project Wizard

Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?

C:\Users\Brian\Quartus II Projects\rgbmatrix-fpga

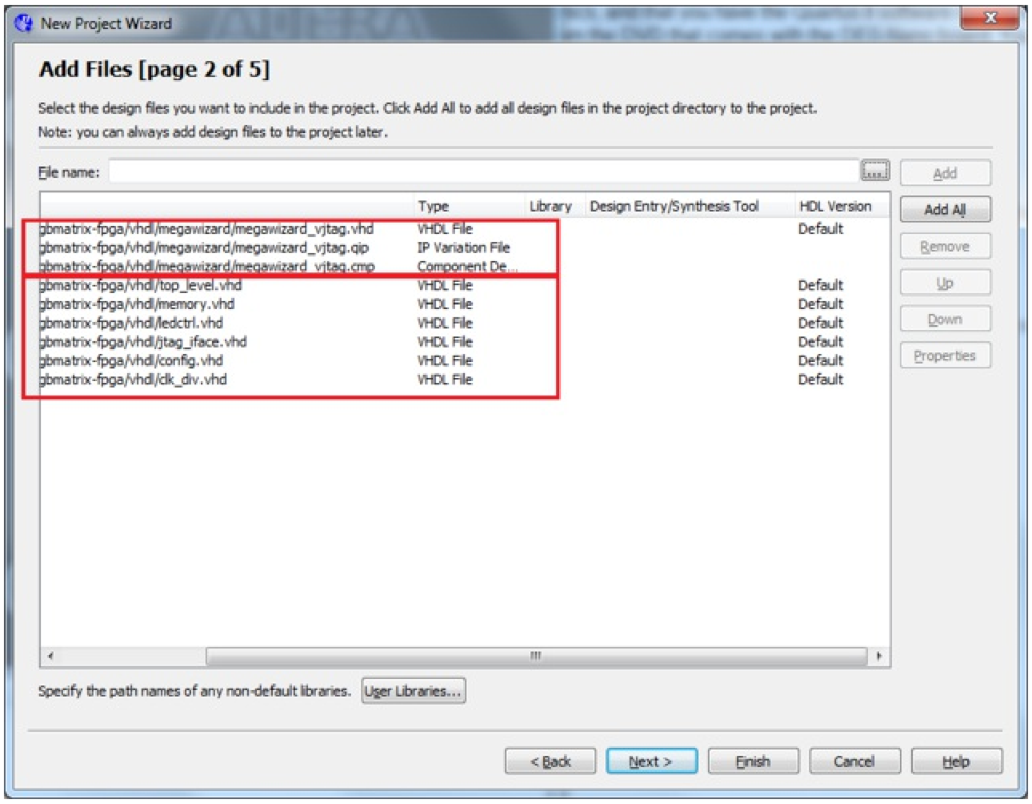
What is the name of this project?

rgbmatrix-fpga

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.

top_level

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.



New Project Wizard

Add Files [page 2 of 5]

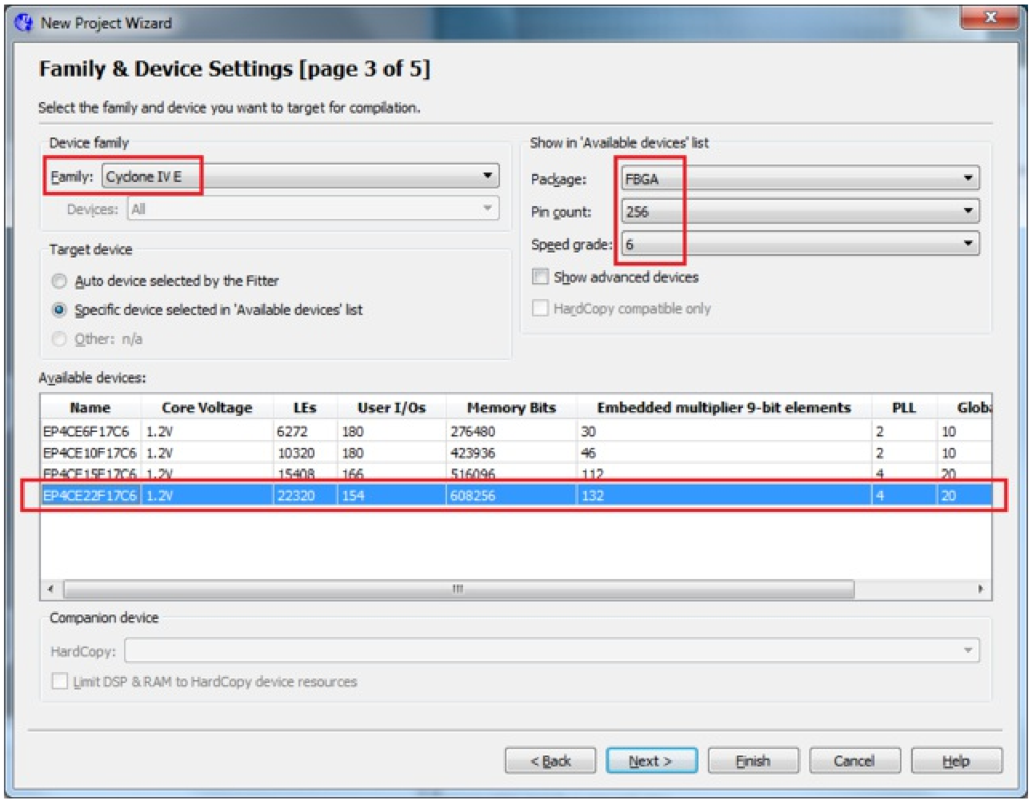
Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.
Note: you can always add design files to the project later.

File name:

	Type	Library	Design Entry/Synthesis Tool	HDL Version
gbmatrix-fpga/vhdl/megawizard/megawizard_vjtag.vhd	VHDL File			Default
gbmatrix-fpga/vhdl/megawizard/megawizard_vjtag.qip	IP Variation File			
gbmatrix-fpga/vhdl/megawizard/megawizard_vjtag.cmp	Component De...			
gbmatrix-fpga/vhdl/top_level.vhd	VHDL File			Default
gbmatrix-fpga/vhdl/memory.vhd	VHDL File			Default
gbmatrix-fpga/vhdl/ledctrl.vhd	VHDL File			Default
gbmatrix-fpga/vhdl/jtag_iface.vhd	VHDL File			Default
gbmatrix-fpga/vhdl/config.vhd	VHDL File			Default
gbmatrix-fpga/vhdl/clk_div.vhd	VHDL File			Default

Specify the path names of any non-default libraries.

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.



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 IV E
Devices: All

Target device
Auto device selected by the Fitter
Specific device selected in 'Available devices' list
Other: n/a

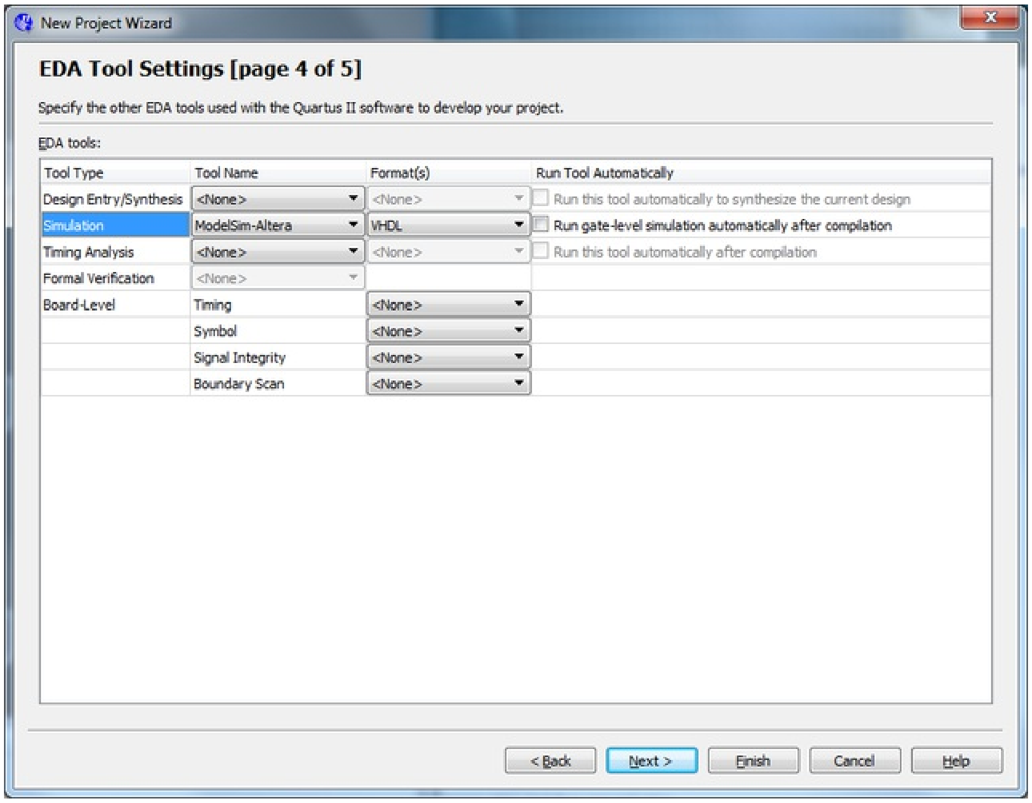
Show in 'Available devices' list
Package: FBGA
Pin count: 256
Speed grade: 6
Show advanced devices
HardCopy compatible only

Available devices:

Name	Core Voltage	LEs	User I/Os	Memory Bits	Embedded multiplier 9-bit elements	PLL	Glob
EP4CE6F17C6	1.2V	6272	180	276480	30	2	10
EP4CE10F17C6	1.2V	10320	180	423936	46	2	10
EP4CE15F17C6	1.2V	15408	166	516096	112	4	20
EP4CE22F17C6	1.2V	22320	154	608256	132	4	20

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

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



EDA Tool Settings [page 4 of 5]

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

EDA tools:

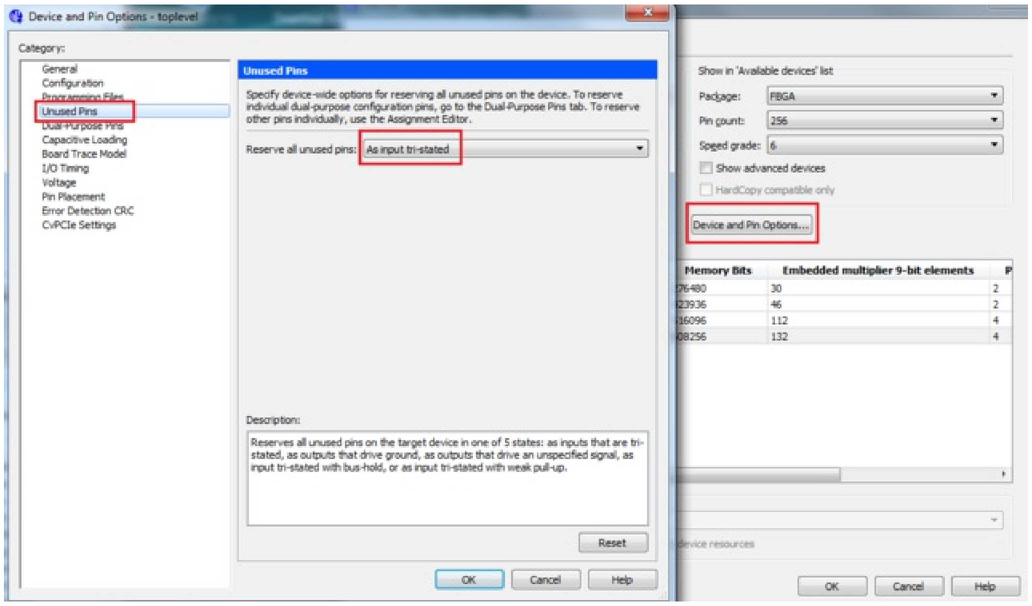
Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/Synthesis	<None>	<None>	Run this tool automatically to synthesize the current design
Simulation	ModelSim-Altera	VHDL	Run gate-level simulation automatically after compilation
Timing Analysis	<None>	<None>	Run this tool automatically after compilation
Formal Verification	<None>		
Board-Level	Timing	<None>	
	Symbol	<None>	
	Signal Integrity	<None>	
	Boundary Scan	<None>	

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.



Device and Pin Options - toplevel

Category:

- General
- Configuration
- Programming Files
- Unused Pins
- Dual-Purpose Pins
- Capacitive Loading
- Board Trace Model
- I/O Timing
- Voltage
- Pin Placement
- Error Detection CRC
- CvPCIe Settings

Unused Pins

Specify device-wide options for reserving all unused pins on the device. To reserve individual dual-purpose configuration pins, go to the Dual-Purpose Pins tab. To reserve other pins individually, use the Assignment Editor.

Reserve all unused pins: As input tri-stated

Description:

Reserves all unused pins on the target device in one of 5 states: as inputs that are tri-stated, as outputs that drive ground, as outputs that drive an unspecified signal, as input tri-stated with bus-hold, or as input tri-stated with weak pull-up.

Show in 'Available devices' list

Package: FBGA

Pin count: 256

Speed grade: 6

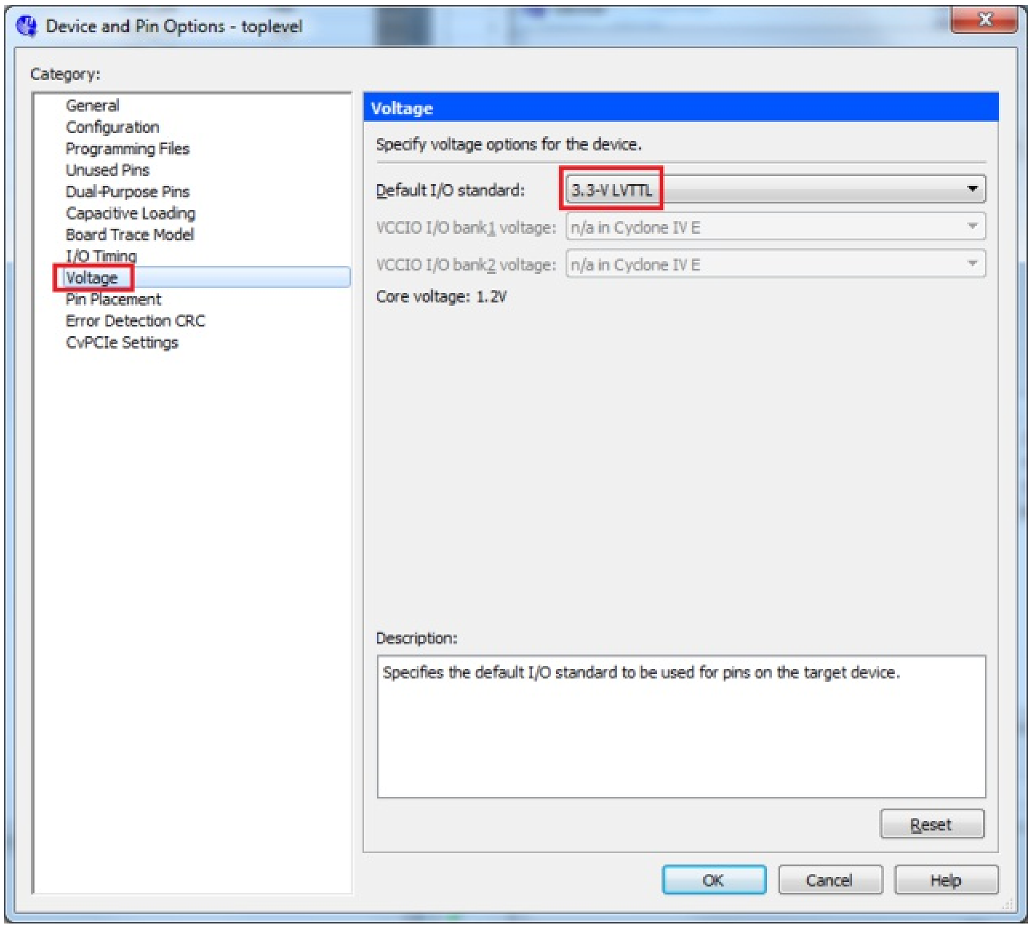
Show advanced devices

HardCopy compatible only

Memory Bits	Embedded multiplier 9-bit elements	P
76480	30	2
23936	46	2
16096	112	4
08256	132	4

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.



Specify voltage options for the device.

Default I/O standard: 3.3-V LVTTL

VCCIO I/O bank1 voltage: n/a in Cyclone IV E

VCCIO I/O bank2 voltage: n/a in Cyclone IV E

Core voltage: 1.2V

Description:

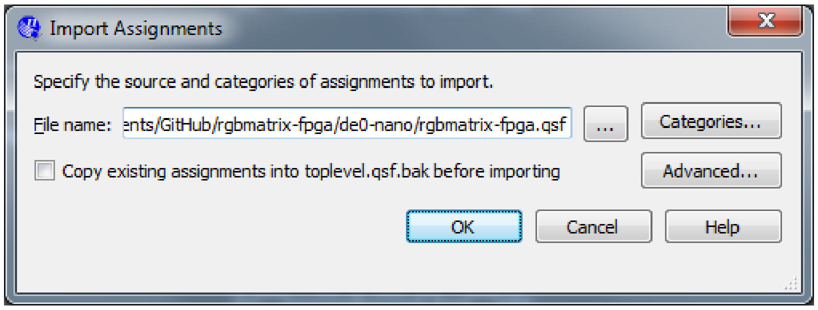
Specifies the default I/O standard to be used for pins on the target device.

Click OK, then click OK again.

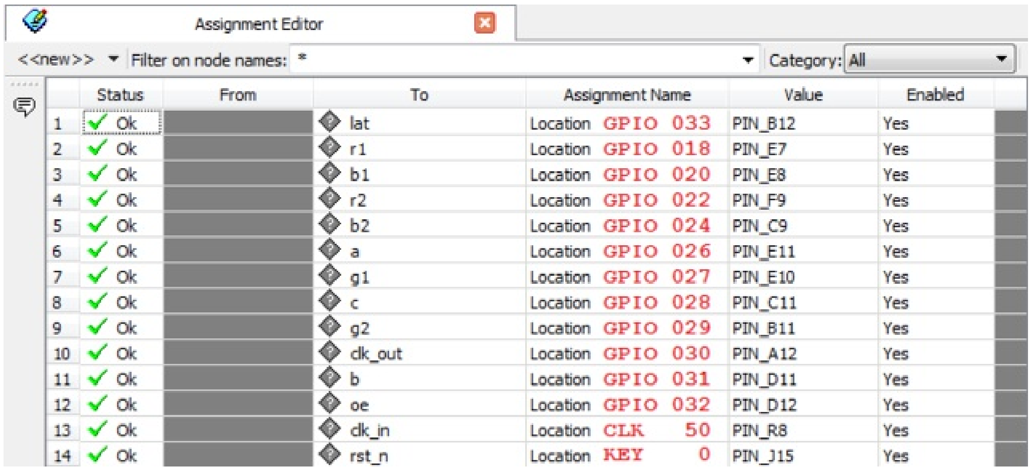
Pin Assignments

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)."

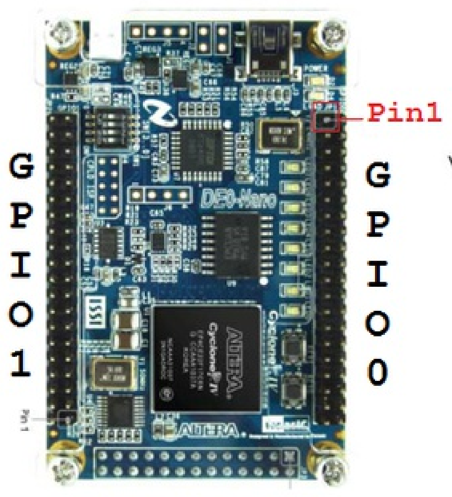

Import Assignments
Specify the source and categories of assignments to import.
File name: ents/GitHub/rgbmatrix-fpga/de0-nano/rgbmatrix-fpga.qsf
...
Categories...
Copy existing assignments into toplevel.qsf.bak before importing
Advanced...
OK
Cancel
Help

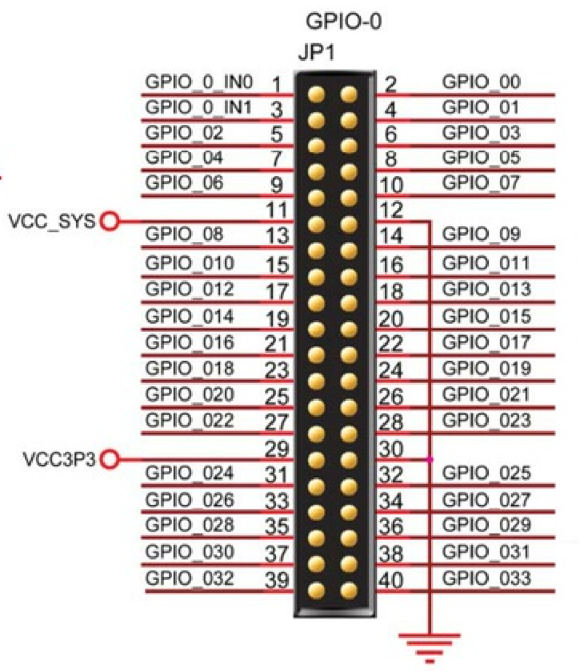
You can (optionally) customize the pin assignments that were imported by going to the "Assignments" menu and selecting "Assignment Editor". Additional information on the GPIO headers can be found in the DE0-Nano PDF manual (https://adafru.it/aIP) (pages 18-20). A mapping of FPGA pins to GPIO headers can also be found in the de0-nano/DE0_Nano.qsf file (open it with a text editor).



	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

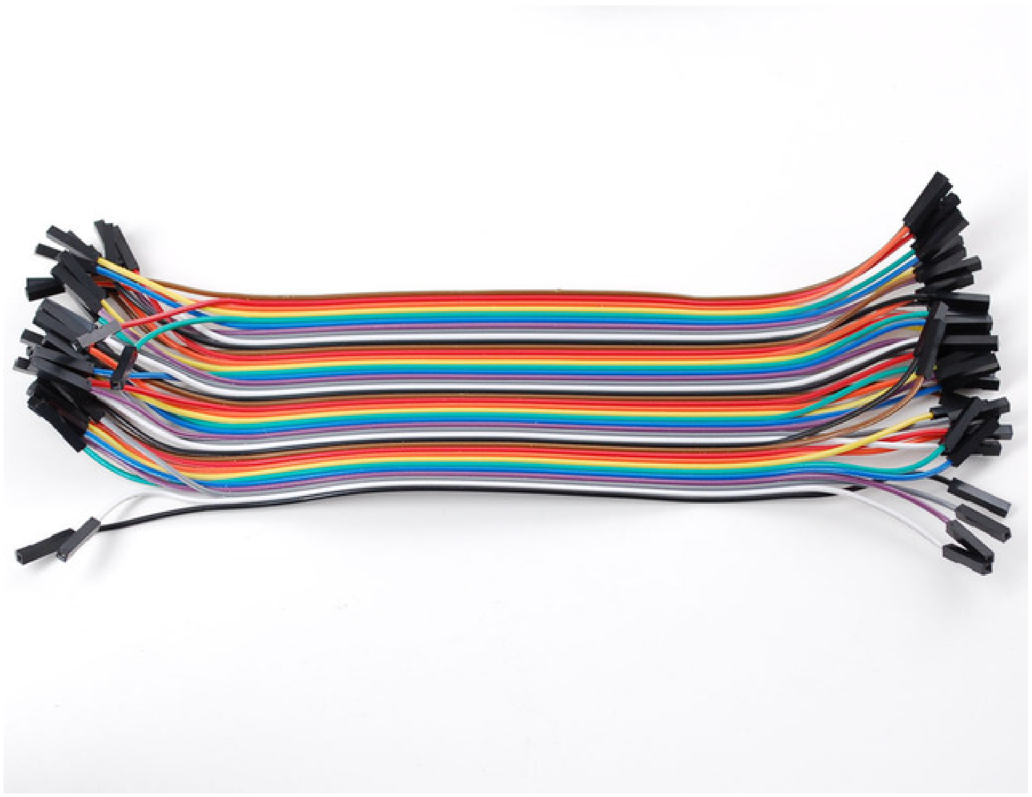
Save any changes. Now we are ready to connect the pins on the FPGA to the pins on the RGB LED matrix panel!





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!

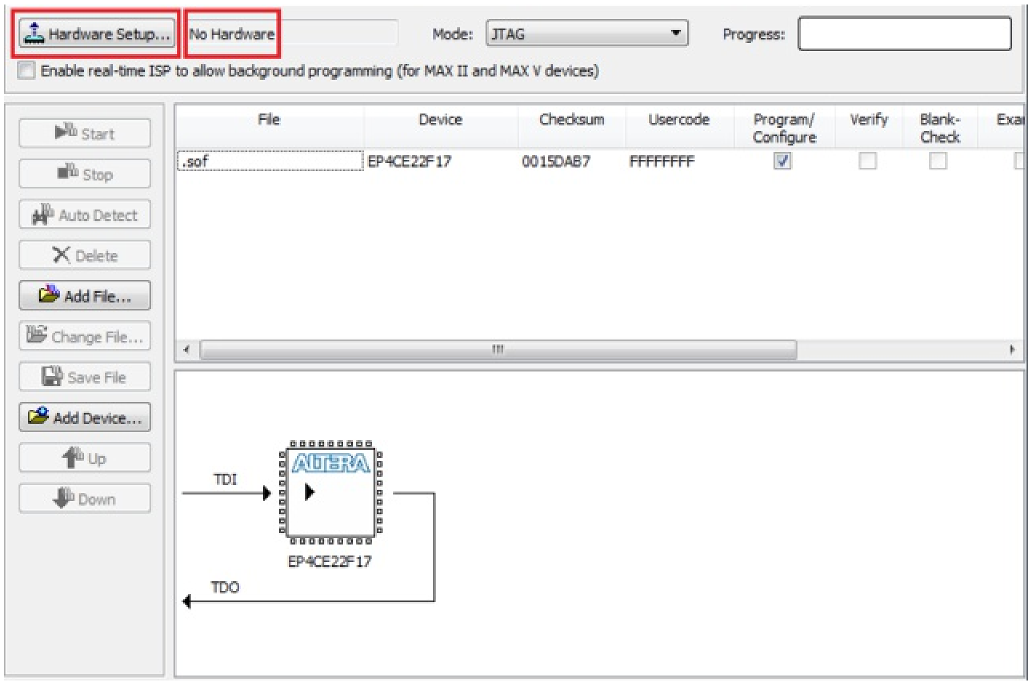
Synthesize and Upload

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.

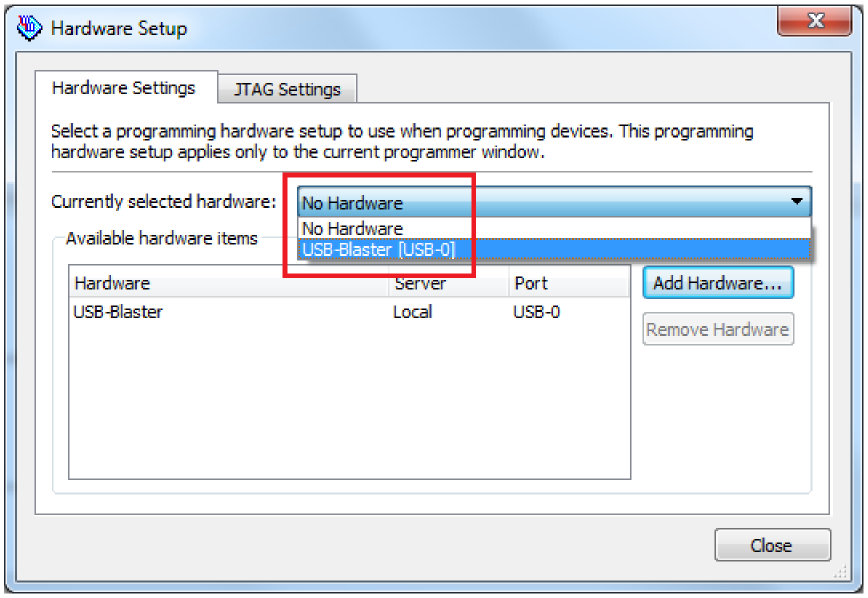
Uploading the bitfile

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



File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Exa
.sof	EP4CE22F17	0015DAB7	FFFFFFFF	✓			

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.



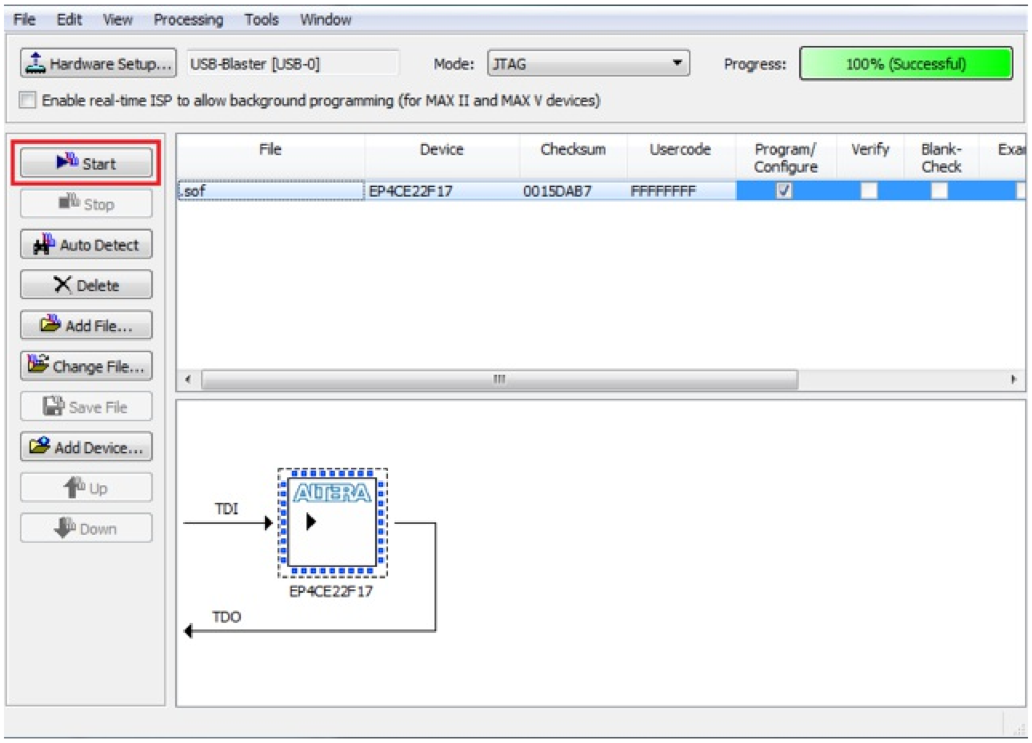
Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: No Hardware
No Hardware
USB-Blaster [USB-0]

Available hardware items

Hardware	Server	Port
USB-Blaster	Local	USB-0

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



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

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

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Exa
.sof	EP4CE22F17	0015DAB7	FFFFFFFF	✓	☐	☐	☐

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.)

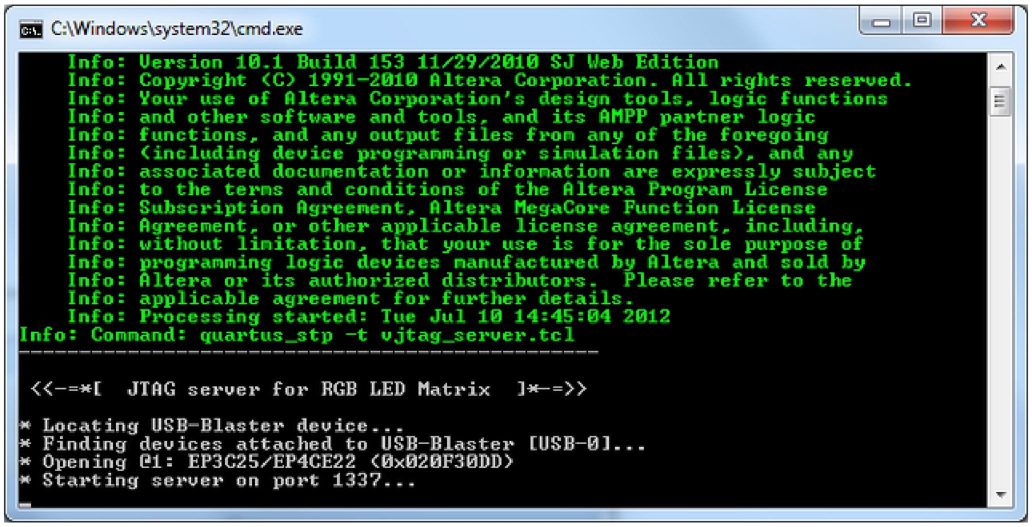
Note: In the future, you can use the command script **de0-nano/program.cmd** to quickly program the FPGA's SRAM with your **.sof** file (it uses the Quartus command line programming utility).

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.



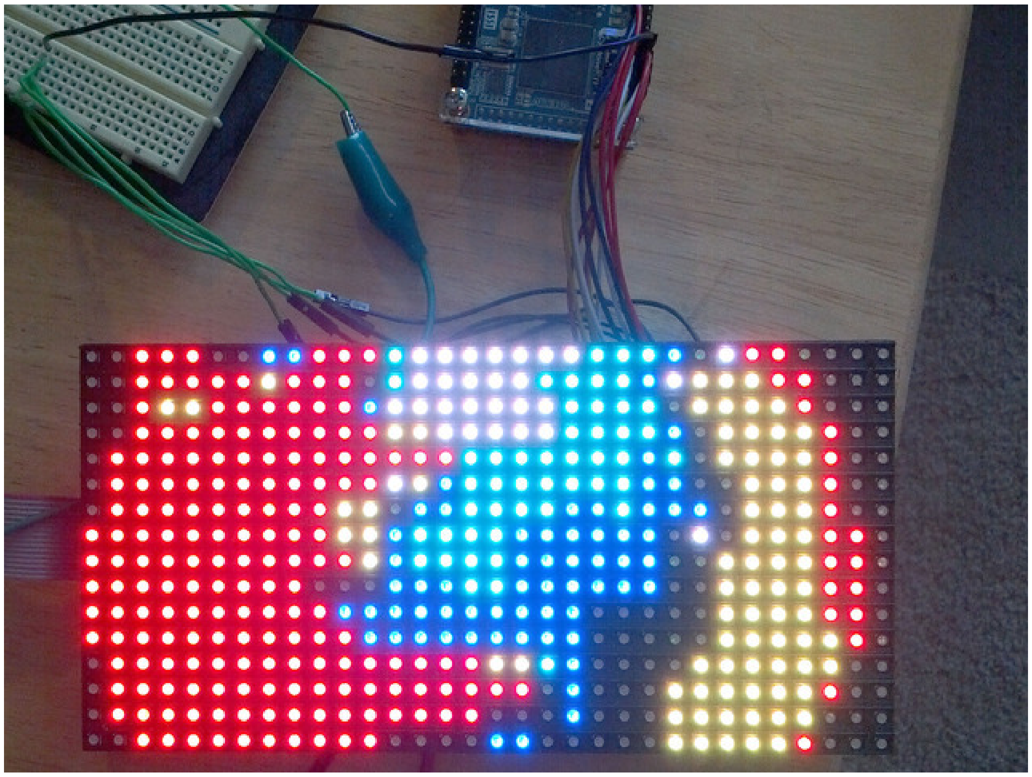
```
    Info: Version 10.1 Build 153 11/29/2010 SJ Web Edition
    Info: Copyright (C) 1991-2010 Altera Corporation. All rights reserved.
    Info: Your use of Altera Corporation's design tools, logic functions
    Info: and other software and tools, and its AMPP partner logic
    Info: functions, and any output files from any of the foregoing
    Info: (including device programming or simulation files), and any
    Info: associated documentation or information are expressly subject
    Info: to the terms and conditions of the Altera Program License
    Info: Subscription Agreement, Altera MegaCore Function License
    Info: Agreement, or other applicable license agreement, including,
    Info: without limitation, that your use is for the sole purpose of
    Info: programming logic devices manufactured by Altera and sold by
    Info: Altera or its authorized distributors.  Please refer to the
    Info: applicable agreement for further details.
    Info: Processing started: Tue Jul 10 14:45:04 2012
Info: Command: quartus_stp -t vjtag_server.tcl
------------------------------------------------

 <<-=*[  JTAG server for RGB LED Matrix  ]*-=>>

* Locating USB-Blaster device...
* Finding devices attached to USB-Blaster [USB-0]...
* Opening @1: EP3C25/EP4CE22 (0x020F30DD)
* Starting server on port 1337...
_
```

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

https://adafru.it/djE

https://adafru.it/djE