Programmable Digital LED controller – WS2811/12

This project is a digital LED controller but it can be a template for other projects like motor control, robotics, drones or any generic programmable controller.

The controller is based on an FPGA board, CMOD-A7 from Digilent, the FPGA part number is – Artix7-35T but also it will fit and work with an Artix7-15T.

The controller can handle up to 16 independent LED strips up to 300 LEDs; it is a mix of high speed logic and a soft core 16 bits custom processor (Opus16) which receives commands from a serial interface. The serial interface is an UART connected to either the on board USB port through an FTD232 chip or a Bluetooth device, BT-HC04, simple ASCII command characters are sent to the processor to select different tasks/routines inside the FPGA.

The custom processor is programmed in Assembly language, it runs at 100MHz and it is close to a RISC processor. Also anyone with FPGA experience can replace the processor with any other, either internally soft core or hard core, or an external microcontroller. The advantage of using the custom Opus16 processor is that it is very simple to program in assembly code, there is no need for a separate software environment, and the code has a Boot loader to allow you to download a new program without the need for re-programming the FPGA.

The FPGA fabric handles all the timing for the 16 different strips and logic control for high speed switching. It contains the 16 buffers needed for each channel, PWMs and timing control for WS2811/12.

The Opus16 allows connecting the FPGA board to either a Bluetooth or USB local Cmod-A7 port to receive and send single ASCII characters.

The USB port can be connected to a Terminal emulator, the Bluetooth can be driven by an Android application. In both cases you can select channels, colors, effects, themes, etc.

This is an open design so anything but the custom processor can be changed and/or modified, the processor is an ip-core targeted to a Xilinx Artix7 family, and I can create different cores for different families or vendors. The Opus16 was tested to work with Altera and Xilinx FPGAs, from Cyclones and Stratix on Altera to Spartans, Artix, Virtex, and Kintex on Xilinx. I can create an ip-core for any of these families; to test it again I'll need a board. So far it was tested on Intel/Altera Cyclon1/2/3/4, Stratix/II/III, AMD/ Xilinx Spartan3/6, Artix7, Virtex4/5/6/7, and Kintex7.

The project contains 6 directories:

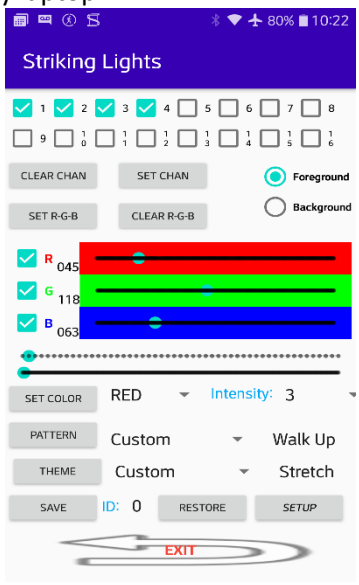
1. **Android_App**
2. **Aprj_Docs**
3. **Aprj_Lib**
4. **ASM_compiler**
5. **Project Directory structure**
6. **Fpga_top and MasterFpga_top**
7. **Hardware Board layout**
8. **Notes**

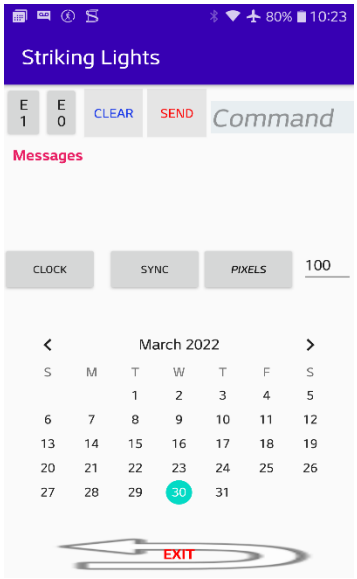
1.) Android application (Android_App)

I learned Android Studio in about 3 to 4 months, enough to try to control my board using an old phone. I continued improving it but really it is not my goal to be an expert neither on JAVA nor Android. I'm an MSEE expert in FPGAs/ASICs and custom processors. I worked just enough to make the whole system to work. I would like Android/IOS experts to take the app and make it professional, right now it is a decent app but far to be delivered. ***I thanks to many developers that allow people like me to learn Android Studio in a short period of time, I used a lot of examples from many of you, with copy and paste and some changes I was able to drive my hardware from a phone/tablet. Thanks again***.

The app can be tested without the hardware, I'm using another Android phone/tablet running a terminal emulator, and in this way you can connect and see the app layout, three frames working. This app works only with BT classic, it does not work with BT-LE, there are many examples how to do it, I have one but I'm not expending time in that.

Any question about the Android app just let me know through email only fpgahelp@gmail.com , in this folder are two Android folders, A4 for phones that support Android 4.0 and up and A10 for tablets and support also Android 4 and up. The difference between A4 and A10 is the size of the text for phones and tablets; I don't know how deal with that so I created two different folders. If you download the app you must be knowledgeable with Android Studio and makes the necessary changes, I had some issues just moving from my desktop to my laptop.





There are three frames; Frame1 is obvious, it is connected to SLi001 Bluetooth, my hardware, also you can connect to TS7_EEA tablet running a terminal BT emulator. Frame2 is the application to control the LED strip, the commands and controls are explained in a doc in the Android folder. Frame3 send any command to the hardware and also is an attempt to implement a calendar where you can setup a special pattern/theme lights and colors for a time and day of the month like any calendar. When the time/day/month has an event it will send a sequence of commands similar to doing by hand. This feature is not implemented. (I don't know how to do it!).

To be able to run the Android app you need the hardware to connect, but there is a way to do it without a hardware using an old Android phone/tablet running a Bluetooth terminal emulator. When the StrikingLights app connects to the emulator you will be able to see the three frames and the command sent by the app. In the terminal emulator you can send back asci data but there is no need because the Android app does not verify any data back, though you can display in messages frame 3

2.) Documents (Aprj_Docs)

Some documents for the CMOD-A7 Digilent board, the HC04 Bluetooth device, the WS2811 timing and specifications. Some pictures of my hardware and block diagrams.

More in detail documentation can be found in their respective websites.

3.) Libraries (Aprj_Lib)

Clk_xil_a7 contains the ip-core from Xilinx for the clock generator (DCM) it must be updated for your current Vivado version

Prog_ram_16k is also a Xilinx ip-core and it must be re-generated for Vivado. Also it must be a new ip if you want a different RAM size. Control lines must match the Opus16 core, a new program RAM size involves to change the RAM size generic/parameter in Fpga_top.vhd located in fpga_top\design\rtl\top.

UART is my serial RS232 interface, 8bits, no parity, one stop bit, simple written in VHDL, you can replaced it with your own design or another ip-core from Xilinx.

4.) Assembler compiler and Assembly application code (ASM_compiler)

This is by far the more complex but at the same time simple to use. It has three directories, Docs, Opus16_asm and Opus16_cores_xil.

Docs contain the detailed description of the Opus16 custom processor and the ASCII commands list used by the LED controller application. The test assembly file sim_isa_order.asm is an example, the PERL compiler generates the mem, coe and lst files. The code and the listing file shows all the instructions and opcodes.

Opus16_asm contains the Examples directory with many asm files for study and training on the custom processor. The PROG directory contains the real application program for the digital LED controller WS2811.asm and the Exec directory contains the PERL program that converts asm files in executable and the operation code table. The PERL compiler converts asm file into a coe, mem and lst files to be used by Vivado and the bootloader, .coe is for Vivado to initialize the RAM, .mif is for Altera, .mem is an ASCII file to use by the boot loader to download the program on the fly, without the need to reprogram the FPGA and of course the .lst that shows the opcodes for the asm file after conversion. The process to compile an assembly program is as follow;

Open a cmd window in PROG directory

Run the following commands:

asm ws2811, it creates all the files in the same directory

save ws2811, it copies the coe/mem/mif files to opus16_cores_xil, also to temp directory.

Or if you want to run examples to learn the processor and instruction set:

Open a cmd window in Examples directory

Run the following commands:

asm MasterTemplateFull , it creates all the files in the same directory

save MasterTemplateFull, it copies the coe/mem/mif files to opus16_cores_xil, also to temp directory.

Files compiled in the PROG directory will always be the same name; you can change it, prog_ram.coe, prog_ram.mif and prog_ram.memif. The prog_ram.hex is an Intel hex format standard, all of these files are created by the PERL program called oasm.pl, look into the asm.bat to see the PERL command.
The asm files compiled in Examples will be prog_ram_example.coe, prog_ram_examples.mif and prog_ram_examples.mem.

Coe files are used by Vivado to initialize the RAM core, Mif files are used by Quartus to initialize the Ram core, Mem file is Opus16 format and used by the boot loader to download a new program into the fpga without the need to recompile the Vivado project.

There is a big difference between MasterTemplateFull.asm and MasterTemplateBody.asm, the first one contains all the support routines to create an application program. It includes the Bootloader, UART for Bluetooth, FTD232 and SESMA plus debug routines and other utilities, just add the application module to test. The Body.asm is a barefoot template where you add everything either for simulation or standalone running application.

5.) ***Project Directory structure***

Aprj_cmodA7_SL

```
|_____Android_app
|                |____StrikingLights_A4   (Phones)
|                |____StrikingLights_A10 (Tablets)
|
|_____Aprj_Docs
|
|_____Aprj_Lib
|                |____clk_xil_a7 (AMD/Xilinx)
|                |                |____clk_wiz_1
|                |____prog_ram16k (AMD/Xilinx)
|                |____UART (RTL)
|
| ____ASM_Compiler
|                |____Opus16_asm
|                |                |____exec       (PERL)
|                |                |____PROG      (ASM)
|                |                |____Examples (ASM)
|                |____Opus16_cores_xil (AMD/Xilinx)
|                                 |____coe/mif/mem
| ____Fpga_top
|                |_____Design
|                |         |____inc (pinout, xdc constraint file)
|                |         |____pkg
|                |         |____rtl
|                |                   |____opus16_ipcore (AMD/Xilinx)
|                |                   |____core (RTL)
|                |                   |____top   (RTL)       (AMD/Xilinx)
|                |_____Gates
|                          |____project_1 (AMD/Xilinx)
|
| ____MasterFpga_top
                 |_____Design
                 |_____Gates
```

Android_App contains the programmable LED application, there are two, one for phone another for a tablet. The app works on HTC Sensation, HTC One and LG5, also for Winnovo-T7 tablet.

Aprj_Docs contains documents related to Bluetooth HC-04, FPGA board CMOD7 from Digilent, schematics and more.

Aprj_Lib contains the Xilinx clock manager, the 16K RAM, they need to be re-generated for a new version of Vivado. The UART is a basic serial interface, fully RTL, works with Altera and Xilinx.

ASM_Compiler contains everything needed to compile an assembly code into a coe Xilinx file. Also contains the PERL code as an assembler, the ISA (Instruction Set Architecture) of the custom processor, examples of Opus16 assembly code. Also generates an Altera files to be used on Altera boards.

Design is a typical project for an fpga

Gates contains the fpga vendor synthesis, implementation and bit generation

The Fpga_top is the WS2811/12 project, with full support for 16 LED strips; the xdc constraint file is in ….design\inc directory. The full pinout for the board is defined.
The pinout can be modified to adapt to your board layout, the logic is the same, and also the way to manage the switching between the BT device and the UARTs can be changed according to your requirements.

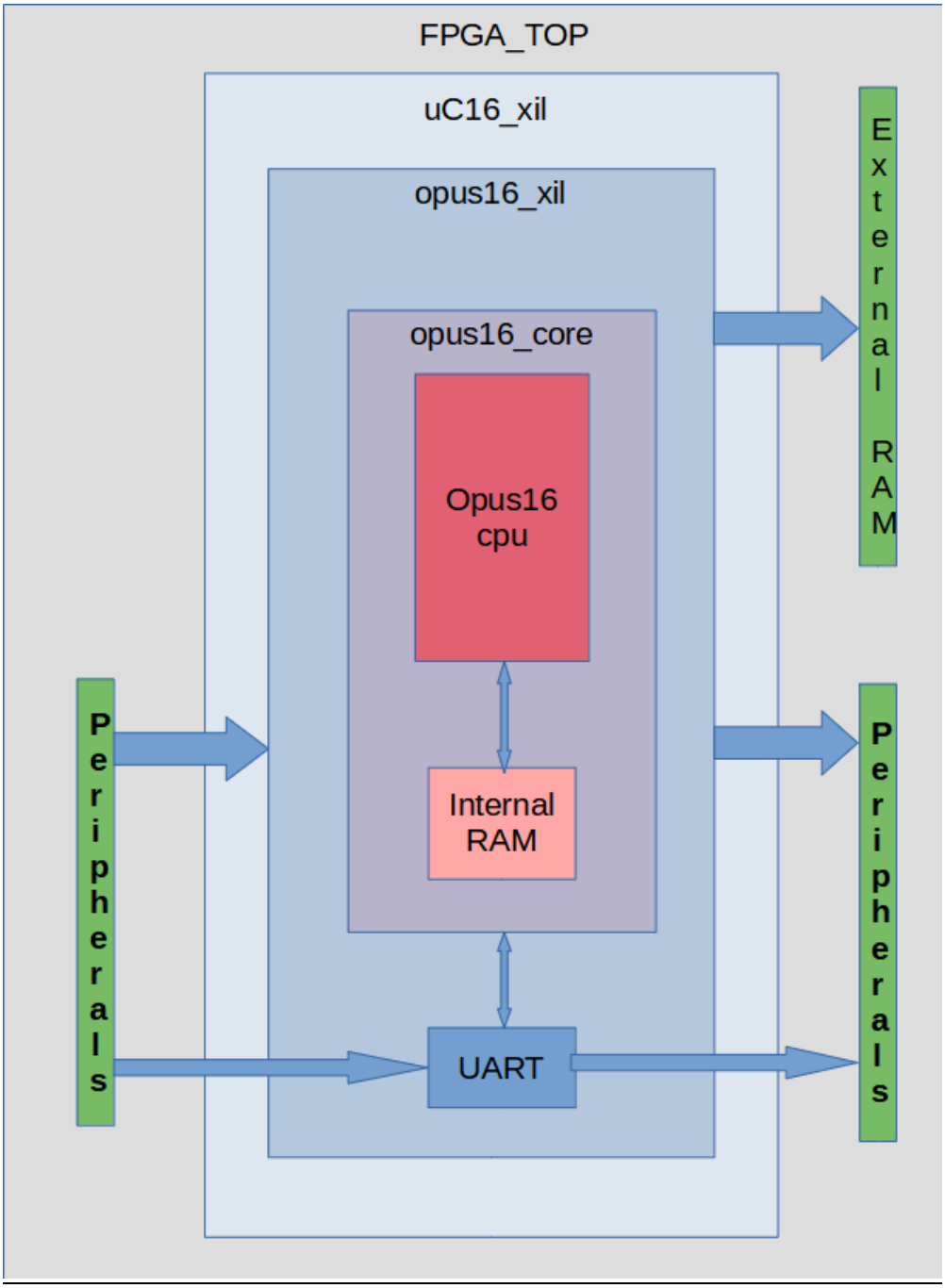
The MasterFpga_top is a template for a generic project, only some pins are define, like clock, reset, UART, Bluetooth and some jumpers and switches/buttons. The test_basic_functions_full.asm is a good assembly code to exercise all the assembly code in Examples directory. There is a simple program to look into for learning and is based on the MasterTemplate.asm. The PERL program compiles an assembly code and generates the following files:

From filename.asm ASCII assembly file 6 files are generated

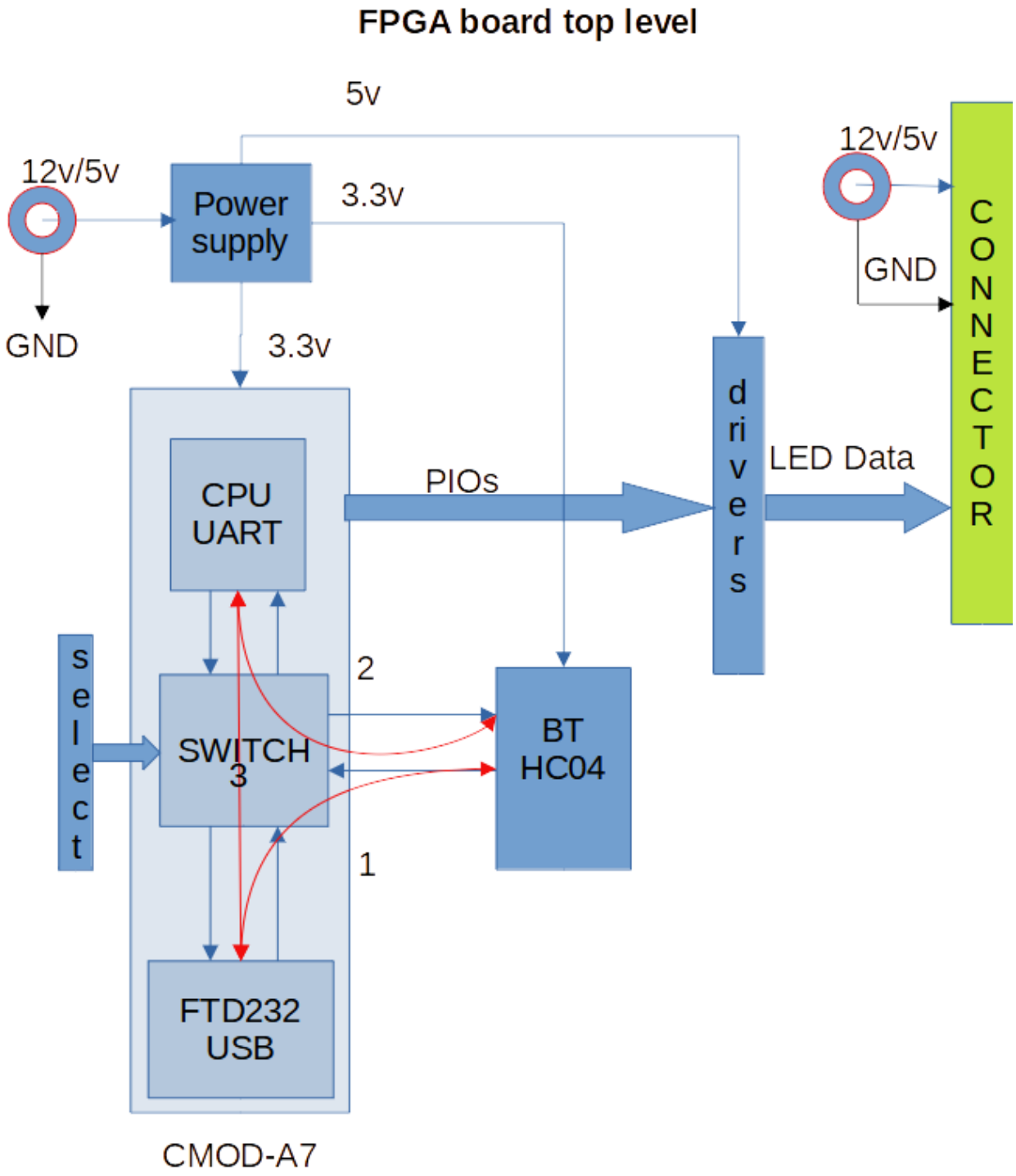
- filename.coe : Xilinx COE file to be used by Vivado to initialize RAM blocks
- filename.hex : Intel hex format
- filename.lst : a listing of the assembly code
- filename.mem: a proprietary format used by the bootloader
- filename_alt.mif: Altera mif format
- filename_xil.mif: Xilinx mif format

I wrote PERL with version of PERL 2001, whatever was, nothing fancy, someday it will be nice to move to C or another readable program language☺, I used PERL because I was good writing code when I was designing ASICs.

Fpga_top and MasterFpga_top



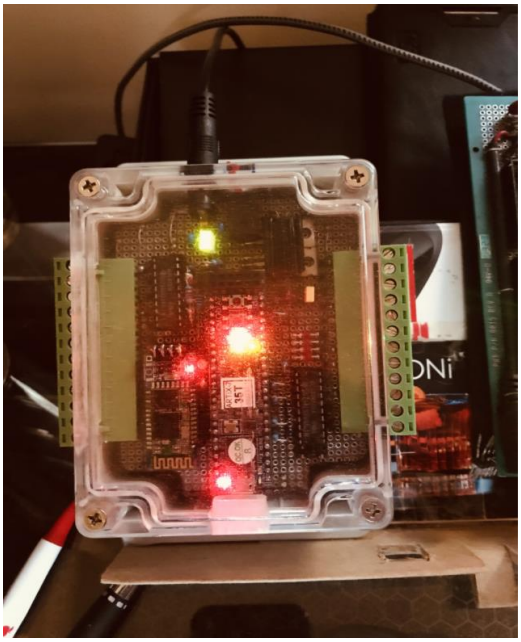
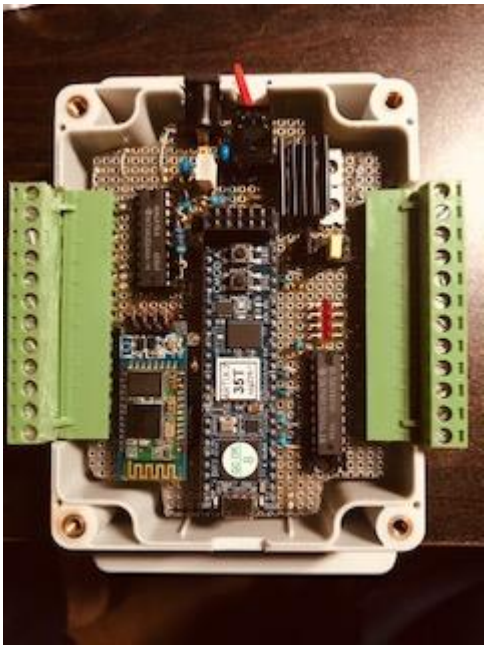
6.) ***Hardware: board layout***



Select following serial interface connections:

1. FTD232 to BT-HC04 for BT-HC04 programming baud rate, device name, etc.
2. CPU-UART to BT-HC08 for normal operation with Android or any Bluetooth device
3. FTD232 to CPU-UART for debugging and testing with a Terminal emulator

Look in fpga_top.vhd for information about the switch

Picture of the prototype, all the components are old devices I use for prototyping except the Artix-35T FPGA board and the BT-HC04 Bluetooth both are visible from top view. The strip connector can be configure in different ways, in my case three points are for PWR/DATA/GND so this can hold 4 channels per connector and I have a second board under knee for the other 8 channels. This is very handy for strip replacements. Also you can use a configuration per connector with 8 DATA and 2 PWR and 2 GND and you must do the wiring.


This is an easy way to connect and remove the strip but is up to the connector you select to change the configuration.

7.) Notes

Any question or request you may have please email me at fpgahelp@gmail.com, I'll try to answer as quickly as I can, at some point and it depends on how thing are going I'll create a simple website to exchange ideas. Also a 15' to 20' YouTube video could be helpful to show procedures, tricks and use of the design for other kind of applications.

There is another document, Notes_qa.docx, this document contains all the latest notes and comments time stamped in order, also Q&A.