***Notes***

This project is for a controller to drive up to 16 LED digital strips but the design structure is valid for any digital electronic project, could be motor control, lighting control, and robotics in general, etc.

The mix of a high speed FPGA plus a soft core processor is very powerful, this project uses a custom 16 bits processor, the Opus16, a 16 bits integer custom soft processor, there is no need for a software environment or high level language, is based on a PERL compiler and assembly language, very simple.

Any question or request you may have please email me at [fpgahelp@gmail.com](mailto:fpgahelp@gmail.com), I’ll try to answer as quickly as I can, at some point, depends on how things 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 type of applications. Notes are in ascending order, new notes are on top.

**Modification History** (Date, Initials, Comments)

03-28-22 ghr note #2

03-20-22 ghr note #1

**Notes #2**

1. Bootloader
   1. This is a protected code and can only be overwritten when the FPGA is programmed, the download of a new application program only write up to the start of the boot loader.
   2. The boot loader is always at the top of the program memory size, the last 256 words.
   3. The .mem file extension is a proprietary format, simple ASCII text where the first 16 bits is the length of the file, no checksum.
   4. In this application and template the ’L’ is the command to download a .mem file, also the hardware has the start and end of the app code and the write protection signal to overwrite the code, look in fpga\_top.vhd to find the logic and how it works.
   5. In this application or master template code you need to turn off the write protect, download the .mem file and automatically the write protection turns on after download process. Look into fpga\_top.vhd to see the Opus16 input ports to control the write code protection.
   6. ECODE is the end of the protected program code, it doesn’t include the writable RAM variables, the BCODE is the start of the boot code and PCODE is the signal to overwrite the download.
2. SESMA ( **SE**quential **S**tate **MA**chine)
   1. This a typical (enhanced) software sequential state machine, it processes the ASCII command coming from the CPU-UART to execute tasks/routines.

// Field definitions:

// PST : Present State address (kpst##)

// VALUE : input key or single value.

// JCODE : "J" -> jump to next state.

// "S" -> save present state and jump to next state

// "R" -> restore last state to present state and jump into it.

// ARADD : Action routine address

// NXST : Next State address

PST VALUE PCODE ARADD NXST

------------|------|-----|---------|-------------------------------

kpst00: dw "d" ,"J", d\_debug, kpst10; //memory/register access

dw "s" ,"J", d\_sysset, kpst30; //system setup

dw NULL ,"J", d\_menu, kpst00; //d\_menu

kpst10: dw "A" ,"J", d\_mema, kpst10; //memory access mode

dw "B" ,"J", d\_memb, kpst10; //memory byte mode

dw "P" ,"J", d\_pagenum,kpst10; // set page SRAM

dw "m" ,"S", s\_bitop, wpst01; //memory

dw "x" ,"J", d\_menu, kpst00; // return to main menu

dw " " ,"J", d\_debug, kpst10;

dw NULL ,"J", d\_nothing,kpst10;

wpst01: dw "," ,"J", a\_setadd, wpst02; //set addr and wait for data

dw CR ,"R", a\_getval, kpst00; //get current value

dw NULL ,"J", a\_getadr, wpst01; //get 4 characters

When the ASCII character received from the UART is equal to VALUE and the JCODE is a J (Jump) or S (Substate) the next state is NXST. In the next state you can have an J to NXST or a R(Return) to its previous state.

**Note #1**

1. Vivado setup of the memory configuration for this version of the Digilent Cmod-A7 is part number Micron serial OR flash N25Q032A13EF440F 32Mb x1/2/4 3.3v; new Cmod-A7 can have a different part numbers.
2. Select following serial interface connections:
3. FTD232 to BT-HC04 for programming baud rate, device name, etc.
   1. Hold btn0 (reset) and btn1, LED will light up
   2. release btn0 reset
   3. Connected to BT-HC04, program Bluetooth
   4. Press btn0 (reset) again to normal operation
4. CPU-UART to BT-HC08 for normal operation with Android or any Bluetooth device
   1. jumper J1 ON, see xdc constraint file for pin numbers and names
5. FTD232 to CPU-UART for debugging and testing with a Terminal emulator
   1. jumper J1 OFF, see xdc constraint file for pin numbers and names
6. Look in fpga\_top.vhd for information about the jumper
7. Timing for ws2811/12 led strip can be found in the document folder or in the vendor website, the design implemented in fpga\_top.vhd has the following timing with test points to check with a logic analyzer or Vivado analyzer:

