An FPGA Learning Experience: SPI Interface to Max10 FPGA

Gregory Raven, KF5N
greg.electricity@gmail.com

July 30, 2019

Abstract

A story of learning FPGA technology with the evolution of a simple project.

1 Introduction

This is the "Golden Age" of SBCs (Single Board Computers) and microcontroller boards. Walk into your favorite brick-and-mortar bookstore (not many remain), and check out the magazine shelf. You will find several magazines, or "book-a-zines" dedicated to "Arduino" or "Raspberry Pi". On the bookshelves you find a few books on Arduino and RPi, maybe even one for BeagleBone.

Or look in the magazine "Make", or use their "Makers' Guide to Boards":

https://makezine.com/comparison/boards/

Using the above webpage, you can select a "Type" of board from three categories:

- Microcontroller
- Single Board Computer
- FPGA

While there are dozens of boards listed, there are only three shown for "FPGA". This list is not entirely accurate, as more FPGA boards o exist, however, it gives an idea of the relative scarcity of FPGA boards.

There are a ton of resources online, and dozens of books and magazines on SBCs which you can find at the local bookstore. The ARRL store sells several SBC themed books as well. Do a search for FPGA. Nothing!

I think it is fair to say that FPGA technology has not made it very far into the ranks of hobbyists. Amateur radio operators have certainly been pioneers in this, with numerous "Software Defined Radio" projects going back to the era when the devices were really expensive. FPGA is an amazing technology, a sort of "3D printer of digital electronics". Perhaps there are other applications of FPGA within the amateur radio community in addition to SDR which can be explored?

I wanted to experiment with this FPGA stuff! The current crop of devices and boards has lowered the cost of entry. This is my story of a first project in FPGA. Hopefully it can be shown that FPGA projects are within reach, and other hams can be encouraged to try working with this fascinating technology.

2 Why?

Why work with FPGAs? Aren't SBCs good enough?

I like to think of FPGAs as a sort of "3D printer for electronics". A loose analogy yes, but the point is that FPGAs allow you to create new digital circuits at will. It is HARDWARE not software! This gives you the power to create multiple specifically targeted digital machines which can work independently of one another. This is the capability that SDR uses to crunch DSP math very efficiently while handling data flow simultaneously. A list of FPGA applications from Wikipedia:

https://en.wikipedia.org/wiki/Field-programmable_gate_array#Common_applications

FPGAs will almost certainly be used in combination with conventional "hard" computing devices (like your laptop or Raspberry Pi). Think of an FPGA as a capability you can add to your SBC to expand its capabilities into high-efficiency computing and the real-time domain.

3 Where to Start

When I began my FPGA quest, I was already familiar with the most common SBCs and microcontroller boards, the Arduino, the BeagleBone, and the Raspberry Pi. It a good thing to have some experience developing with an SBC before attempting to tackle FPGA. In my case, I spent quite a bit of time developing several projects on the BeagleBone Black SBC. Many of the skills learned working with SBCs will apply to FPGAs.

One of the ways I went up the learning curve on SBCs was this lecture series by Bruce Land of Cornell University:

https://www.youtube.com/watch?v=FYy6JN0vpg0&list=PLKcjQ_UFkrd4z2qoFuJ1jtVhCSuxxCTpk

This course uses a PIC32 microcontroller board. However, the material in the lectures is generic enough to apply to any board which can be programmed in C. Even better, a second course covers FPGA!

https://www.youtube.com/playlist?list=PLKcjQ_UFkrd7UcOVMm39A6VdMbWWq-e_c

and the matching website:

http://people.ece.cornell.edu/land/courses/ece5760/

Watching a few of the youtube videos gave me a pretty good idea of what was involved in FPGA work. I didn't use the same development board as used in this course, but I did use another Intel FPGA based board. So the development tools and general flow of working with the FPGA are similar.

You will need to use a variety of resources to answer questions and solve problems as you go up the FPGA learning curve. It seemed to me that most of the effort required to learn FPGA is in handling a large

quantity of details. You will need to spend a lot of hours absorbing this stuff. I think that FPGAs are at least a little bit harder than working with SBCs.

4 A Most Significant Source of Learning: Vendor Web Site

You will need to get an account at your FPGA vendor's website. I used an Intel based board, and the account was free. On the Intel site, you will have access to large amounts of learning resources for FPGA. Be prepared to spend many hours watching videos and studying documentation, whatever vendor you have chosen. Intel has good material, and you can learn a lot! A recommended starting point is the video series "Become an FPGA Designer in 4 Hours". The 4 hours part is perhaps a bit optimistic, but it will give you a good early acceleration:

https://www.intel.com/content/www/us/en/programmable/support/training/course/odswbecome.html

5 Github Repository for this Project

The documentation and code for this project is located in this git repository:

https://github.com/Greg-R/spiavalonfpga

6 Choosing a Development Board

The first FPGA development board I purchased was the Numato Lab "Elbert V2" ($29.95) This board has the Xilinx XC3S50A Spartan 3A FPGA device containing 1584 logic cells and 54 KB RAM. The board has a nice selection of peripherals which can be driven by the FPGA:

- 16 MB Flash Memory
- USB 2.0 interface for Flash programming
- 8 LEDs
- 6 push buttons
- DIP switch
- VGA connector
- Audio connector
- SD card adapter
- 3x 7 segment LED
- 39 IOs

Note that the USB capability of this board is for Flash programming only. It is not a general purpose interface to the FPGA.

The reason for choosing the Elbert V2 was the book “Programming FPGAs" by Simon Monk. The book features the Elbert, along with the “Mojo” and “Papilio” boards.

If you have a specific application, then it will be straightforward to decide if the development board meets your requirements. For a person motivated to learn FPGA without without a specific application in mind should choose a board with more peripheral devices than less. Also note that the example above has 39 IOs, which allows for additional peripherals to be installed. If the IO pins are in the format used by Arduino, the addition of peripherals will be easy to accomplish.

I was able to install the development software for the Elbert and work through a few of the projects in the book. However, I found the development tools lacking in one area. I was interested in using a more modern version of Verilog, called “SystemVerilog”. The development tools for the Elbert V2 do not support that, so more searching was required.

My motivation for SystemVerilog was to try some of the new features, including those used for “test-benches” (simulation). That is only an expression of my particular interest, as the Verilog used in the Xilinx tool for this device is fine and can be used for maximum benefit with this device. This does, however, indicate that the development environment which mates up with the FPGA device is as important as the device itself. So before you choose a board, be sure to download and install the development tools first. Look at the documentation and decide if you will be comfortable with that particular development tool capabilities.

Looking around a bit more, I found this board after viewing Bruce Land’s great lecture series on youtube:

https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=836

The DE1-SOC is used in Cornell’s “ECE 5760 Advanced Microcontroller Design and system-on-chip” course.

The board requires the "Quartus" development tool, which met my requirement for SystemVerilog. However, the FPGA on this board is well beyond a beginner! This board uses an advanced ”System On Chip” (SOC) which is a combination of microcontroller (dual-core ARM Cortex A9) and FPGA in a single device. Price is $249, which seemed a little steep for a project which might not work out. The added complexity of the ARM processors, having to deal with an unfamiliar distribution of Linux running on the ARM processors, and the all of the extra work involved seemed like too much. Indeed this board is immensely capable, so maybe I will come back to it in the future!

Fortunately the same company, Terasic, makes boards with less complex FPGAs (and less expensive). Perusing their site, I found this board, the DE10-Lite ($85):

https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=234&No=1021

This looked like a good choice, and I purchased it. Later when doing internet searches, I found there are plenty of example projects already available for this board. It seems to be at moderately popular in the academic world, and some courses have been based on this board. The MAX10 FPGA is well supported,

although it is not included in the latest version of the Quartus development tool (latest version supported 2018).

A good feature of this board is the Arduino compatible expansion header. In general, the DE10-Lite has been easy to work with, and seems to be robust. I've been using it for many months and it is still alive!

7 A Project Inspiration and JTAG to FTDI SPI Interface

The inspiration for this project came from a a search at github.com on DE10-Lite:

https://github.com/hildebrandmw/de10lite-hdl

This is a nice collection of work done with the DE10-Lite board for academic purposes. What really got my attention is the project play_gif:

https://github.com/hildebrandmw/de10lite-hdl/tree/master/projects/play_gif

The interesting feature of this project is the usage of the USB to load data (animated GIF image file) to the FPGA. So this is a data pipeline from a Linux desktop to the FPGA which is built into the board! This met my requirement that I be able to control the FPGA remotely from a desktop (or SBC) computer.

The interface is via "JTAG", which is typically used as a debugging interface. It is not specifically intended for mass data transfer, but in this case it was pressed into service.

The interface is a bit clunky to use. It requires a "TCL Server", and a running instance of the Quartus development tool! Not exactly what I was looking for, but I got the demo to work easily! It is very nicely done work demonstrating several features of FPGA technology.

The way it works is conceptually simple. The FPGA part of the project implements a VGA[1] interface to the connector on the DE10-Lite board. The image loaded from the desktop computer is sliced into its constituent "frames". Another interesting aspect of the project is the interface to the SDRAM of the DE10-Lite which is a 64MB external part on the board. The sliced-up image is loaded into the SDRAM, and then another control module pages the VGA output through the memory. Thus you see the animated GIF displayed on the monitor. Really you are seeing in a very direct manner the data loaded into the SDRAM. Cool!

7.1 IP and Platform Designer

First, a little bit of FPGA jargon. "Intellectual Property" (IP) in the context of semiconductor devices is a block of circuitry which has been heavily engineered and refined to perform some particular function. It could be patented or otherwise protected from duplication by competitors. Due to the way integrated circuits are manufactured, blocks of "IP" can be added to the silicon and be expected to perform to the IP owner's specifications. Typically IP can be included as part of a design kit, or it can be paid for with a license fee.

[1] VGA is a relatively simple video standard which seems to be common on many FPGA development boards.

IP is good because it can reduce engineering design effort, improve performance, and enhance quality. The trade-off is license fee cost, and you don't necessarily get exactly what you want.

In our case, we are given a whole bunch of IP for free that we can experiment with! This is bundled into "Platform Designer" which is a tool-within-a-tool in the Quartus design suite.

To do justice to this there should be an entire section on "Platform Designer". I will summarize here. There are excellent video Platform Designer tutorials which you can access if you register for a free account at the Intel web site.

"Platform Designer" is a building-block system. You get a library of IP, along with a mechanism to hook them together. The design is bundled into a "Qsys" file. Let's have a look at the Qsys part of the play_gif project:

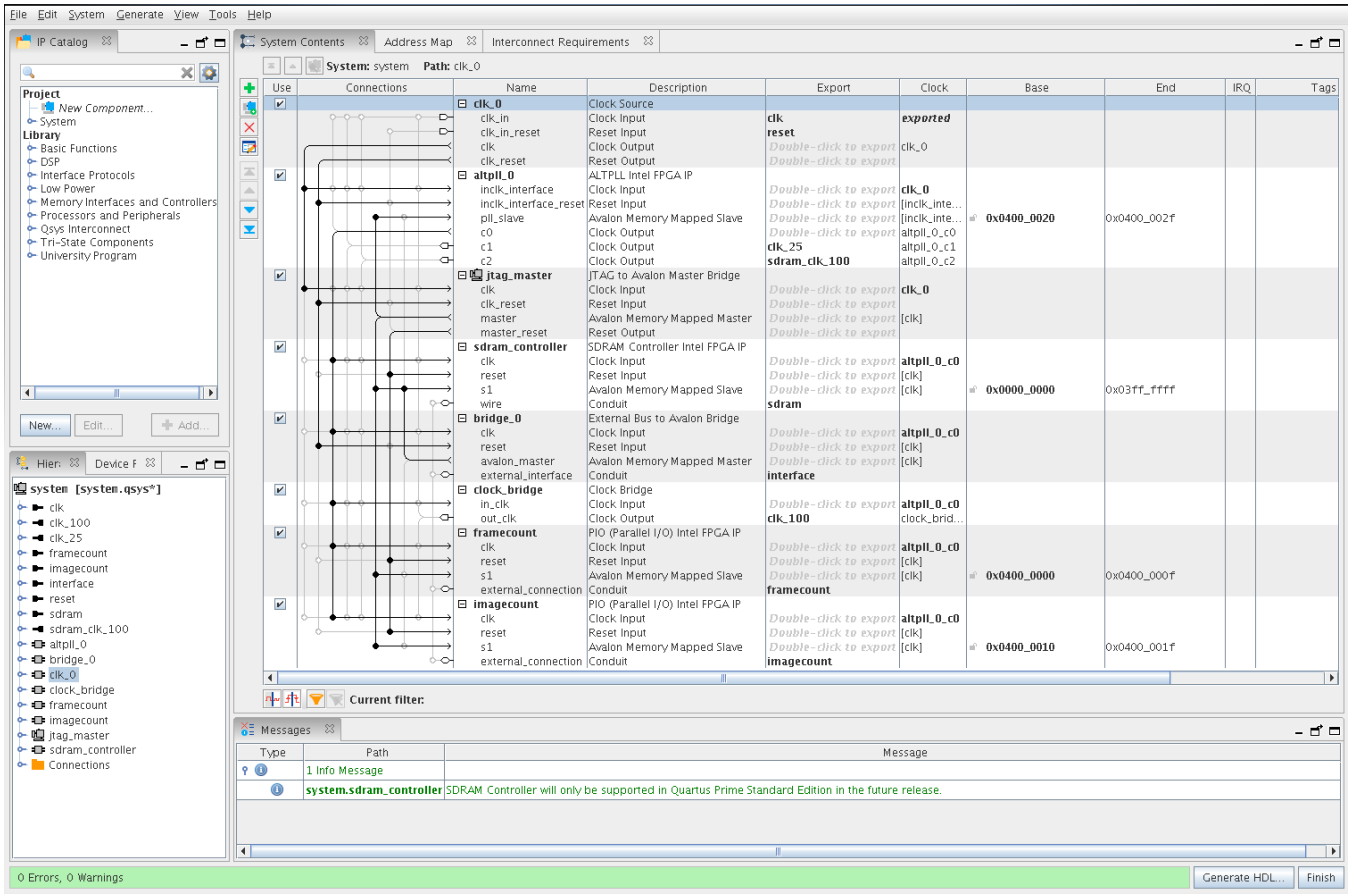

Figure 1: Play_gif project Qsys file

The upper left corner of the GUI is the library of IP. Some of the categories:

- Basic Functions
- ESP
- Memory Interfaces
- Processors and Peripherals (including a "Nios" processor)

- Memory Interfaces and Controllers

There is a sort of “sub-library” called “University Program” which includes:

- Audio and Video
- Bridges
- Clocks
- Communication
- Generic IO
- Memory

The project used this IP:

1. ALTPLL Intel FPGA IP
2. JTAG to Avalon Master Bridge
3. DRAM Controller Intel FPGA IP
4. External Bus to Avalon Bridge
5. Clock Bridge
6. (Parallel IO) Intel FPGA IP

The above IP can be seen in the column “Name”. There are two instantiations of the Parallel IO. In the column “Connections” can be seen the graphical interconnections between the IP blocks. Connections are made by simply clicking on the circles at the intersections of the “wires” between the IP. Thus an entire system can be assembled using this GUI. No writing of Verilog required! The project does include some hand-written Verilog. This “Qsys” design is “dropped in” to the project as a Verilog module.

Here is what the system looks like:

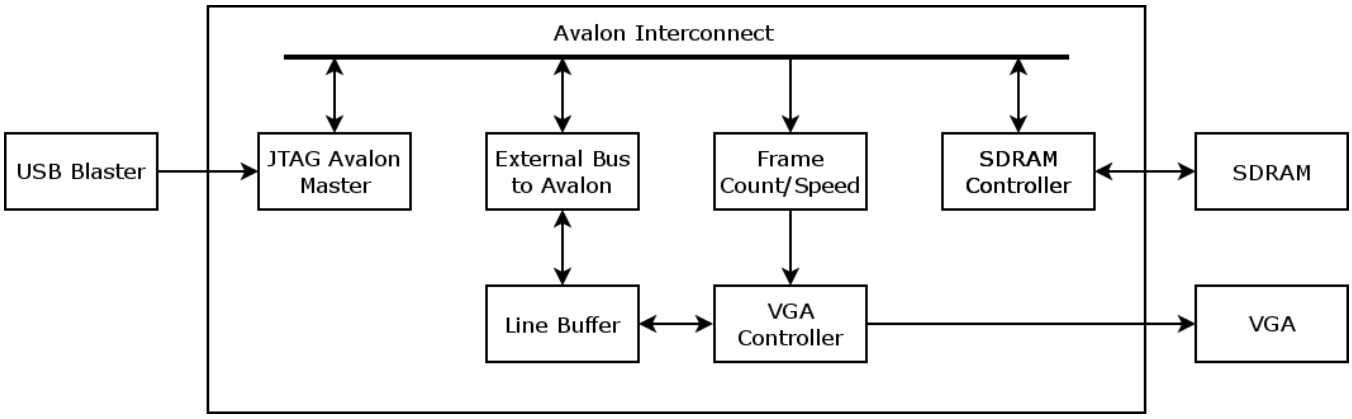

Figure 2: Play_gif JTAG Interface System Diagram

What this diagram does not show is the requirement for a running Quartus and a TCL/JTAG server program.

The “Avalon Interconnect” deserves some explanation. This is a system bus used in the MAX10 FPGA. From the Avalon Interface specification:

> Avalon® interfaces simplify system design by allowing you to easily connect components in Intel® FPGA. The Avalon interface family defines interfaces appropriate for streaming high-speed data, reading and writing registers and memory, and controlling off-chip devices. Components available in Platform Designer incorporate these standard interfaces. Additionally, you can incorporate Avalon interfaces in custom components, enhancing the interoperability of designs.

It is an internal bus standard used to connect Avalon bus masters and slaves. So it is a single-click process to connect Avalon components in Platform Designer. It is really amazing what you get for such little effort!

This project is interesting, and shows a path to communication between a desktop computer and the FPGA. However, the JTAG + Quartus + TCL/JTAG Server is cumbersome. A SPI to Avalon bus IP component is listed in the catalog. What if the JTAG and stuff could be replaced by something simpler like a SPI bus?

So that is what evolved into my “introductory FPGA project”. The revised system diagram:

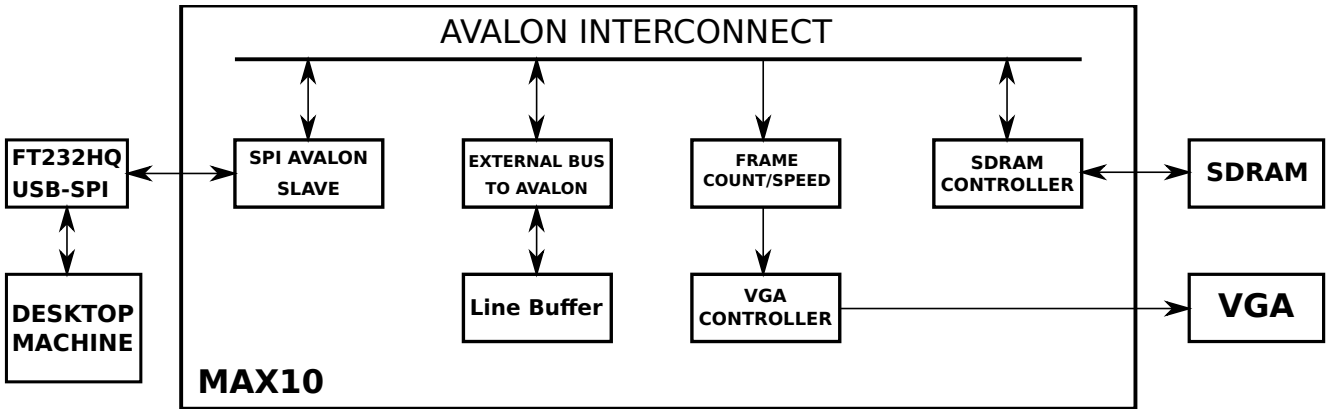

Figure 3: Play_gif with SPI System Diagram

The significant change on the FPGA is the swapping of the JTAG Avalon bus master with the SPI-Avalon slave. This was not entirely a drop-in replacement, as there were changes to reset and clock connections in addition to swapping JTAG to SPI components. But it is easy, and the swapping can be done in a couple of minutes.

External to the DE10-Lite board, there is a new USB to SPI adapter board. This board is based on the FT232H chip by FTDI. This can be bought from eBay for about $10. Search for “ft232 spi” and you will find several options. I recommend one with headers to allow it to be plugged into a common breadboard.

Other changes required for SPI:

- The ports on the QSYS module changed (JTAG -> SPI), thus the Verilog module in which it is instantiated required minor changes. This was done with the text editor feature of Quartus.
- Another change is required to the FPGA pins. The new SPI bus must be routed to some easily accessible header on the DE10-Lite board. Since the board has an Arduino compatible header, and

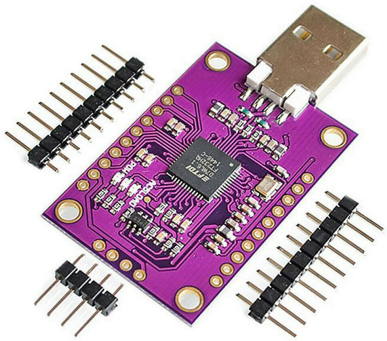

Figure 4: FTDI FT232H USB to SPI Break-out Board

this header has a standard set of four pins for SPI, those pins were used. The details can be seen in the DE10-Lite manual provided by Terasic. The "Pin Planner" tool was used to make the changes.

- The "Synopsys Design Constraints" (.sdc) file was updated to incorporate the SPI bus.

Here is the crazy breadboard hook-up:

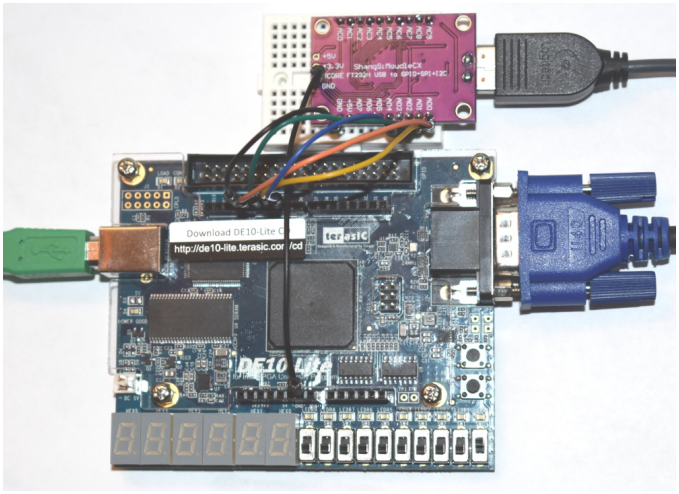

Figure 5: DE10-Lite Connected to FTDI SPI Breakout

In spite of the length of the breadboard jumper wires, the SPI interface performed remarkably well right up to the FTDI clock limit of 30 MHz.

8 SPI Driver in Julia Programming Language

The github project from which mine was derived uses a programming language called "Julia":

https://julialang.org/

This was a language I had heard about, but I've never tried it. To duplicate the original project, I had to install the language and run the program. The program does the processing of the GIF image, and then sends the data to the FPGA via the JTAG server and USB-to-JTAG interface.

It was simple to install the server and run the Julia program. It all worked first time! Later, I installed the Atom IDE which has a Julia plug-in. It's great!

Rather than reinventing the wheel, I decided to use the image processing portion of the Julia program. However, how to drive the FTDI SPI device? I needed something to replace the JTAG server.

The FTDI USB-to-SPI device is supported with a C shared library. This library has the initialization, read, write, and shutdown command necessary to work with the device.

Fortuitously, the Julia language includes the capability to call C library functions in a very direct way! I was skeptical at first, but I quickly had the SPI device's initialization function running and returning with no error. The other required functions were quickly added. I now had full control of the SPI bus from the command line!

When I say "command line", in the case of Julia I am referring to the "Read Eval Print Loop", called the REPL. This functionality is similar to Python, and is my favorite way to develop code. I also used the "Atom" IDE, which has a plug-in for the Julia language. It is a new language, and has a few quirks like all of them do, but so far I am very impressed!

At first, Julia was also used to access a shared library which was taken from an Altera demo project of the SPI-Avalon bus master. This library was responsible for reading and writing "Avalon Packets". This is the protocol used by the Avalon bus. I was able to successfully translate the Altera library to Julia. This is working well and the system is able to read and write to the SPI and thus the FPGA Avalon bus, the parallel ports, and the SDRAM.

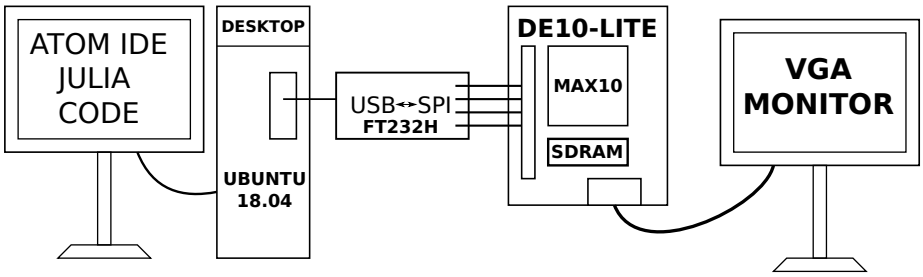

Figure 6: Project System: Desktop -> USB-to-SPI -> FPGA -> VGA Monitor

The Julia code is located in the "julia" directory of the github repository linked in the introduction.

9 FPGA and Verilog Books

9.1 Hobbyist FPGA Books

"Designing Video Game Hardware in Verilog" Steven Hugg, first printing 2018

A very practical introduction to digital hardware using the early history of video games as a means of illustrating the technology. The book includes significant introduction to Verilog and its relationship to the physical circuitry.

However, it is not primarily an FPGA book! The reader is encouraged to use a web-based Verilog simulator:

http://8bitworkshop.com

There is an example FPGA which uses the iCE40HX-1K iCEstick. This is a low-cost device ($40 on Amazon). Development can proceed with the official Lattice tool chain, or with an open-source system known as "IceStorm":

http://www.clifford.at/icestorm

"Programming FPGAs, Getting Started in Verilog" Simon Monk, 2017 McGraw-Hill Education

Good coverage of beginning FPGA using the boards Elbert 2, Mojo, and Papilio.

https://www.amazon.com/Programming-FPGAs-Getting-Started-Verilog/dp/125964376X/ref=sr_1_1?keywords=fpga+monk&qid=1564448368&s=books&sr=1-1

9.2 Verilog

You will need to go up the learning curve on Verilog (or VHDL). Here are a couple of inexpensive books (< $20) which will get you going:

"Designing Digital Systems with SystemVerilog" Brent E. Nelson, Brigham Young University

https://www.amazon.com/gp/product/1075968437/ref=ppx_yo_dt_b_asin_title_o00_s03?ie=UTF8&psc=1

"Exploring Digital Logic" George Self.

http://www.lulu.com/shop/george-self/exploring-digital-logic/paperback/product-22747579.html

Free eBook by Intel:

"FPGAs for Dummies"

https://plan.seek.intel.com/PSG_WW_NC_LPCD_FR_2018_FPGAforDummiesbook

10 Simulation

The FPGA developer should develop skills in FPGA simulation. The original "blinky LED" project I attempted prior to the SPI bus project required me to do that. I had a minor, but persistent bug which brought my work to a halt. Trial and error with the board and re-doing the FPGA got me nowhere fast. After I had a simple simulation running, the problem was quickly resolved. Fortunately the Quartus tool does most of the work to set up the simulation. A small bit of hacking of "do" (TCL) files is required.

I recommend using the resources on the Intel web site to explore the basics of setting up and running simulations via Quartus.

11 Conclusion

I was able to find an FPGA development board and "starter project" that met my requirements for a beginning in FPGA development. A bare minimum of Verilog modifications were required to make the project function correctly.

Most of the FPGA "design" was done using the system-level "Platform Designer" tool.

I was able to add memory-mapped read of the SDRAM and FPGA registers via analysis of the Avalon bus structure. The Julia programming language was used to create and decode Avalon bus transactions. Direct access of the SPI device C shared library from Julia code was used.

The next stage of the project will be a practical ham radio application. I have an antenna rotator which I want to control via wireless. I think it will be possible to create a real-time state machine on the FPGA which will handle the motor drive along with a Hall sensor and counter for positional feedback. Driving the FPGA from the SPI port of an SBC, along with a WIFI connection, should allow the entire control unit to be wireless. Solar powered? Maybe.