Attempt To Make Persistence Of Vision Wand

What

What I am attempting to do is to make a wand (not a light saber; those are already all over Bellingham); but a wand, about 3 feet long, that will flicker and create images by taking advantage of persistence of vision.

Imaging looking at me at night as I am waving the wand over my head. If the wand is lit with a constant color and light; through persistence of vision, it would appear briefly in your eye as an arc. It would be a single color arc.

Now imagine that same wand but it has a strip of reg/green/blue (RGB) LED; with each LED independently controlled by a computer or controller. These LED's would flash in coordination so that they would form patterns that would briefly appear in your eye as you stare at me waving the wand.

This is similar to how early television and movies worked. A quick succession of images (in the case of a movie) or a flying spot varying in brightness (in the case of early television) took advantage of persistence of vision.

There are requirements in timing. The faster one can wave the wand, the longer the entire image can remain in a person's persistence of vision. Also, the faster the updating of the wand, the quicker the image can be put out on the wand, and the longer the entire image can remain in a person's persistence of vision.

Why

I feel that this can give me a challenging FPGA design project. I had already discovered that I could not accomplish this with a Raspberry PI as the Raspberry PI running Linux would be too slow.

I feel also that this can have an artistic application. I have had a fantasy for a long time of standing on a prominent street corner (such as Bay Street and Holly Street in Downtown Bellingham, Washington, which is a very artistic university town in Northwest Washington State), waving a magical wand that creates colorful artistic images for all to see, especially at twilight and the darkness of night.

I have no professional experience in FPGA work. I am a retired security software engineer from Intel Corporation who has done some device driver work. All of my programming was in C and Perl.

However, I was never allowed into the FPGA groups at Intel as I was considered software only. I have had a hardware background and have worked with digital logic, so this is why I decided to try to take this on for my own enjoyment and education.

Here in Bellingham, Washington, there are very limited resources in FPGA's. It turns out that I end up teaching others my limited knowledge; I do not have anyone older and wiser in this game than I. Therefore, I am seeking guidance from you folks here on Reddit FPGA and other online groups as I find them for help.

I am developing this in Verilog.

Perhaps, if this goes well over time (and being retired, I am in no hurry), I could be that older and wiser one for FPGA's here in Bellingham, Washington.

What I have To Work With

The FPGA board is the Arty A7 development board by Digelent (information at https://store.digilentinc.com/arty-a7-artix-7-fpga-development-board-for-makers-and-hobbyists/)

This board has the XC7A35TICSG324-1L Xilinx FPGA of the Artix 7 family of FPGAs from Xilinx.

The board does have RAM, but I did not have the knowledge or experience on how to use it. Therefore I ended up using registers for the storage of the row of RGB led values.

The LED string I use is what is called Dotstar, which is a product of Adafruit. Web page for information is https://learn.adafruit.com/adafruit-dotstar-leds/overview

Dotstar LED strips are not the same as the more famous Neopixel LED strips.

The Dotstar LED strips have both a clock and a data line while Neopixel LED strips have only a data line, therefore utilizing a very strict regime of timing for data, which is too slow for many persistence of vision applications.

The Dotstar LED strips use a smat RGB LED element which is described in this document: https://core-electronics.com.au/tutorials/what-are-dotstar-leds.html

Basically, this is the protocol:

1. The entire string begins with four bytes of 0.

3. The LED's are sent in four byte groups composing for each LED's. The first byte is a global byte controlling the overall brightness for all three colors. It starts with three 1 bits followed by a 5 bit brightness value. I set this to all ones so that each color would be it's full setting as set by its own 8 bit word. After this first byte, the second byte is the green brightness. The third byte is the blue brightness. And finally, the fourth byte is the red brightness.

4. After the last LED's four byte string is sent, then a string of foury FF (all ones) is sent. There is some controversy regarding this; if the string has more than 32 LED's then we need to send another four byte group at the end. I have not had this issue, however.

5. The transmission is based on SPI. Data is set (either high or log). Then rising edge of clock is when the data is clocked in. Clock goes low while the data changes and so on.

6. The four zeroes (first set of data) resets the string. That means that the first led will grab the first set of four bytes. Then it passes all others on. Which means that as each LED down the strip gets it's four bytes, it will then pass the other down and so on. This is overall the same behavior as the more famous Neopixel LED strings.

The Dotstar LED’s us 5 volt signals. The FPGA board uses 4 volt signals. A signal level converter is needed to convert the 3 volt logic from the FPGA to the LED string’s 5 volts. This conversion is only one way as there is no return data from the LED strip.

Here are some photographs of my setup. I hope they make sense to you.

The first photo shows the Arty A7 board on the upper left. In front of the board, that small red board is the logic level converter. The device with the heat sink fins on the right of the board is the 12 volt to 5 volt bucking power converter. I plan to use a 12 volt Lithium battery while dancing on the streets of Bellingham.

If you look carefully on the lower center of the picture, you can see the top of the LED striip.

The LED strip, if it is bare does not blend enough for a good smooth persistence of vision experience. Therefor, I decided to enclose the LED strip into a frosted translucent tube as shown below.

Below is a overall shot of the setup.

Status of Project

The first iteration of the project is complete. The code that is currently here in GitHub is of the first iteration that does operate.

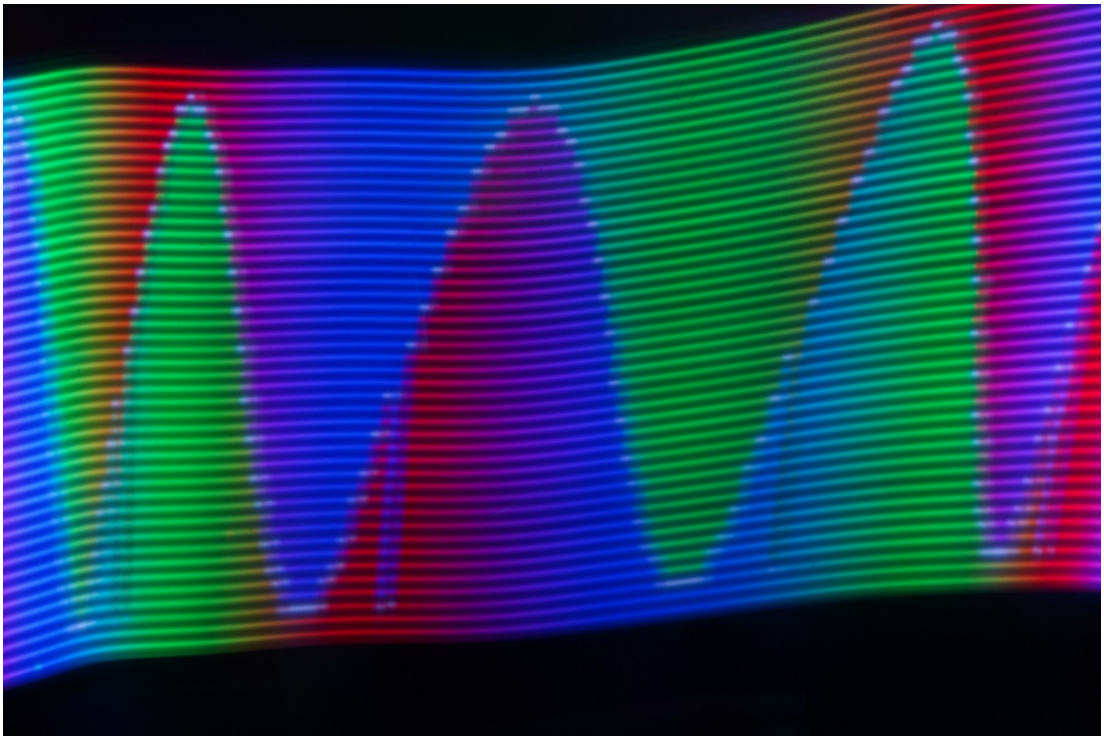
Please be aware that this is very basic image of colorful sine waves. There is no control of where the image is in relation to the top of the swing of the wand, as many people ask. I plan this in the future when I learn to deal with gyro and accelerometer such as this from Adafruit https://www.adafruit.com/product/3387?gclid=CjwKCAjw44jrBRAHEiwAZ9igKF8U2PK4w-exWXc7rT0Hd2KF9r53r6ZNba5OPuzEGhfIlNPL413LKBoCs8kQAvD_BwE

Perhaps in the future I may need your help.

What I have now is a beginning; which proves that I can do FPGA work and have something that is a worthwhile piece of art.

Currently, I cannot swing the wand in the air (I am still awaiting cables and hardware to make it more robust and can be made to add on other accessories such as the gyro unit and perhaps and extension.

The following photograph show hopefully the same effect as swinging the wand. I simply took a time exposure while quickly moving the camera side to side while the strip and FPGA are operating.



My Code

The following files are in the git repository:

1. doit.pl and doit_wand.pl; perl scripts to create the sine lookup tables

2. doled.v; module to take three primary colors and create the appropriate data set to be sent to the LED strip

3. spi.v; the SPI driver that sends the serial data and clock to the LED strip

4. dostring_test.v; test module to check the doled and spi modules

5. dostring_wave.v; the main module to create the color sine wave. All is done on the fly using counters and lookup tables. No large arrays or buffers are used

6. led_constraint.xdc; pin assignments for clock, scl (SPI clock), mosi (SPI data), board LED's for testing

7. outstuff.v; interim file generated by doit.pl and doit_wand.pl that is inserted into the dostring_wave.v

testbench1.v; simulation test bench

8. top.v; top module, this also calls the clock wizard module.

The clock wizard IP from xilinx is used. 100 MHZ input clock from board; 50 MHX output clock to rest of device. No other clocks are used. The wizard's locked output is used for the system reset.

Thank you all for helping!

Respectfully yours,

Mark Allyn

Bellingham, Washington