Digital Design for Astrophysics Detectors

Collin Bradford (Homeschool Student)
Dr. Chris Stoughton (Fermi National Accelerator Laboratory)

Abstract

Sensors in particle detectors often have to sample at a rate that far outpaces typical computers today. One of the solutions for this is to use FPGAs or Field Programmable Gate Arrays to process and condense the signals from the sensors before sending the result to a computer for storage and further processing.

This paper reports on the project to design a data acquisition system to be used for the Nano Cam project. This project uses a phototube attached to a telescope to precisely measure the photons coming from a star. The FPGA, or Field Programmable Gate Array, will take the output from the phototube and process it, looking for spikes in the signal that denote a pulse.

The primary target for this experiment, the Crab Pulsar, is a neutron star that is an estimated 20 Km in diameter. (Crab Pulsar. n.d.) In addition to the regular pulses that happen at 30 Hz, there is an extremely fast pulse that happens randomly every few hours. The pulse comes from a part of the star about the size of a classroom and it pulses with the intensity of the sun. By measuring the star with the phototube, we hope to see if the intensity change is in the visible light spectrum.

Introduction

Background

To get an idea of what an FPGA is, it is helpful to explore the inner workings of a CPU or other digital logic circuit. A digital logic circuit is a device that uses electricity to turn wires on and off to produce signals. We usually refer to the on state as one, and the off state as zero. Scientists and engineers have been able to develop hardware that uses these signals to perform basic logic operations like: “and”, “or”, “not”, and “xor”. From the basics of digital logic and some more advanced hardware, we are able to produce digital logic circuits like processors.

Processors work by taking a basic set of instructions and following them sequentially. At the heart of a processor are registers. These each store information. By setting and changing registers we can add, subtract, multiply, and do any math calculation. An example of this would be taking a number stored in register location A45, adding it with location B46, and storing the result in register C78. (x86 Instruction Listing, Atmel 8-bit Instruction Set Listing) Luckily, this low level programming is all taken care of by compilers and we never usually have to do this when we program in compiled languages.

Because of how processors are designed, they are really good at following orders sequentially. They are designed so that they can run code that is different each time it is turned on. This allows for different programs to run on the processor without changing the internal design of the logic circuit. However, when it comes to processing large quantities of numbers extremely fast with the same algorithm, processors have a limit. For the nanoCam project, we are sampling at 1.5 gigasamples per second. There are a lot of processors that clock in at over 3 ghz, but since it takes multiple clock cycles for each datapoint, this is not a viable solution.

One alternative is to use FPGAs. FPGA stands for Field Programmable Gate Array. FPGAs are devices that contain thousands of CLBs or Configurable Logic Blocks. These are blocks of basic logic circuits like the ones mentioned above that can be connected together in different ways to produce a logic circuit for a specific purpose. To do this, a designer will design a circuit in code and schematics on the computer. The compiler will then take those designs and map out a logic circuit on the device using the CLBs. The design is then loaded onto a flash memory device that communicates with the FPGA. Upon startup, the FPGA will read from the flash memory, configure the logic inside, and start working. FPGAs can be programmed to take the place of almost any logic circuit. (Field-programmable Gate Arrays, Wikipedia) Many designs actually use a processor or multiple processors on the FPGA at the same time.

Context of this work

The data acquisition system I am helping build this summer is going to be used for the nanoCam project. The nanoCam is basically a photon counter that is going to be combined with a telescope that will be used to do high resolution photon detection for objects in space. The primary target for the nanoCam is the Crab Pulsar.

The Crab Pulsar one of the only neutron stars that emits light in the visible spectrum that we can see from earth. Most neutron stars can only be seen using a radio telescope. From observing the Crab Pulsar with a radio telescope, we can see the normal pulses that come from the rotation of the pulsar. In addition to those, however, we can also see a major pulse that happens once every few hours. We estimate the size of the pulsar to be around 20 Km. The major pulse, however, is coming from a region about the size of a classroom. When the major pulse happens every few hours, it changes brightness with around the intensity of the sun. By looking through the optical telescope, we hope to gain better knowledge as to why the Crab Pulsar has a major pulse every few hours.

Methods

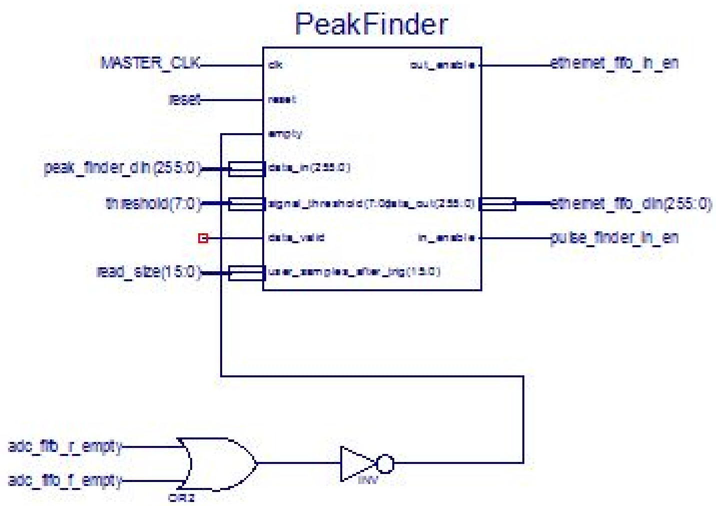
Equipment

To see the Crab Pulsar, we are using a photon counter mounted on the back of a telescope. The signal that comes out of the photon counter is an analog signal. In order to process the signal, it must be digitized. This is done with a high speed analog to digital converter. It samples the analog signal at 1.5 GSPS and sends that data out to the FPGA.

For signal processing, we are using the Captan board. This is an off the shelf data acquisition system designed by engineers here at Fermilab. It comes in a hexagonal shape with the ADC mounted directly below. The Captan board uses a Vertex 4 FPGA.

Captan board (left) with ADC (blue mostly hidden by Captan board. You can see a small blue square on the bottom of the image. That's the ADC board.) and the Xilinx JTag programming cable (red box to the right).

Firmware for the FPGA is developed using the ISE Design Suite provided by Xilinx, the manufacturer of the FPGA. The firmware is written in VHDl code as well as schematics. Some of the code used in the firmware design, like FiFo memory, is already written by Xilinx. To use it, we specify some parameters and it generates the firmware module. For anything that is not really common, however, the code has to be written out instead of generated. Once the code is written, the compiler creates a design that can be created on the physical chip.

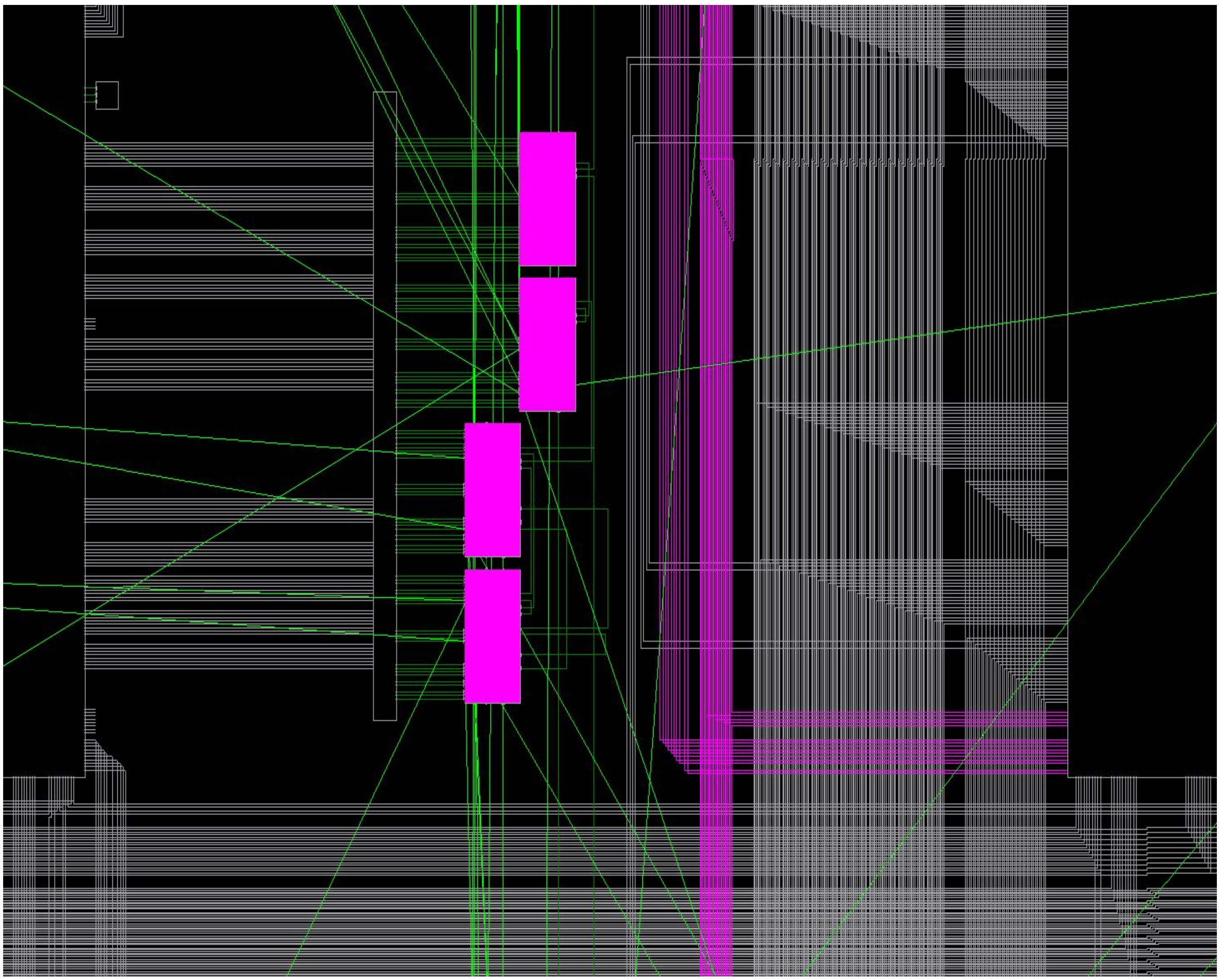


Schematic view of the peak finder module I developed.

```
entity PeakFinder is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           data_in : in  STD_LOGIC_VECTOR (255 downto 0);
           signal_threshold : in STD_LOGIC_VECTOR(7 downto 0);
           user_samples_after_trig : in std_logic_vector(15 downto 0);
           empty : in  STD_LOGIC;
           data_valid : in std_logic;
           data_out : out  STD_LOGIC_VECTOR (255 downto 0);
           out_enable : out  STD_LOGIC;
           in_enable : out std_logic);
end PeakFinder;

architecture Behavioral of PeakFinder is
   type data_array is array (0 to 15) of unsigned(7 downto 0);
   signal data : data_array;

   signal threshold : unsigned( 7 downto 0 );
   signal samplesSinceTrig : unsigned(15 downto 0);
   signal userSamplesSinceTrig : unsigned(15 downto 0);
   signal out_en_sig : std_logic;
begin

   threshold <= unsigned(signal_threshold);
   userSamplesSinceTrig <= unsigned(user_samples_after_trig);
   out_enable <= out_en_sig;

   process(clk)

   begin
      in_enable <= '1';

      if(reset = '0') then--reset is low
         if(rising_edge(clk)) then--rising edge of clk and reset is low
            if(empty = '0') then
                  data_out <= data_in;
                  out_en_sig <= '1';
               else
                  if(samplesSinceTrig >= userSamplesSinceTrig)then--Our sample count matches t
                     out_en_sig <= '0';
                     samplesSinceTrig <= (others => '0');
                  end if;
               end if;

               if(out_en_sig = '1')then --We took another sample.  Increase the sample count
                  samplesSinceTrig <= samplesSinceTrig + 1;
               end if;
```

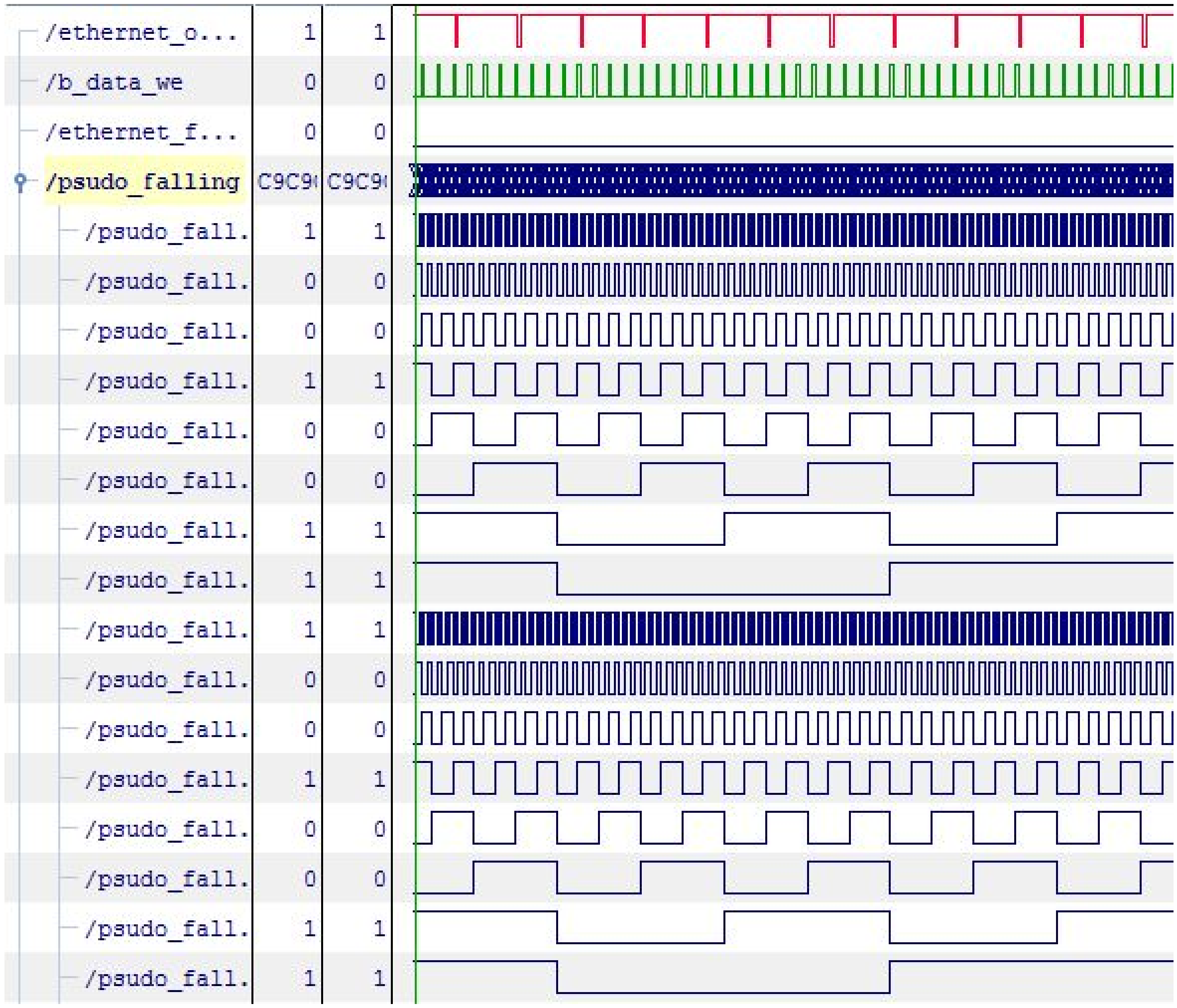
Sample code from the peak finder module.

To flash firmware to an FPGA, a JTag programming cable is used. This clever device actually contains an FPGA of its own, a Spartan 3A, as well as a microcontroller. From the computer, we can send it a design file, and it will flash the flash memory on the board. On startup, the FPGA will read the flash memory and implement the design.



A snapshot of a small portion of the design viewed after the compiler has routed the connections to the different logic resources.

For debugging, we can use the programming cable to see some of the signals inside the actual FPGA. When the firmware is designed, we include a module that samples certain signals and sends them over the JTag connection. After we flash the firmware, we can open a logic analyzer program on the computer and see what is actually going on inside of the FPGA.



Signals from the FPGA as viewed through the logic analyzer. The red one shows that the FIFO memory is overflowing (As to be expected based on the current parameters.) The green signal is showing when data is being written to the computer. The blue signals are the sawtooth wave being sent into the pulse finder.

Data

The data that comes into the FPGA comes in on every rising and falling edge of a 375 Mhz clock. It comes over a 32 bit bus with four samples on every rising and falling edge. The firmware on the FPGA analyzes the data at 100 Mhz so to keep up, it analyzes 32 samples each clock cycle. When it finds a peak, it reads the next few samples and sends them to the computer. The computer can set registers that define the threshold that it compares the data to and how many samples are read after it detects an event.

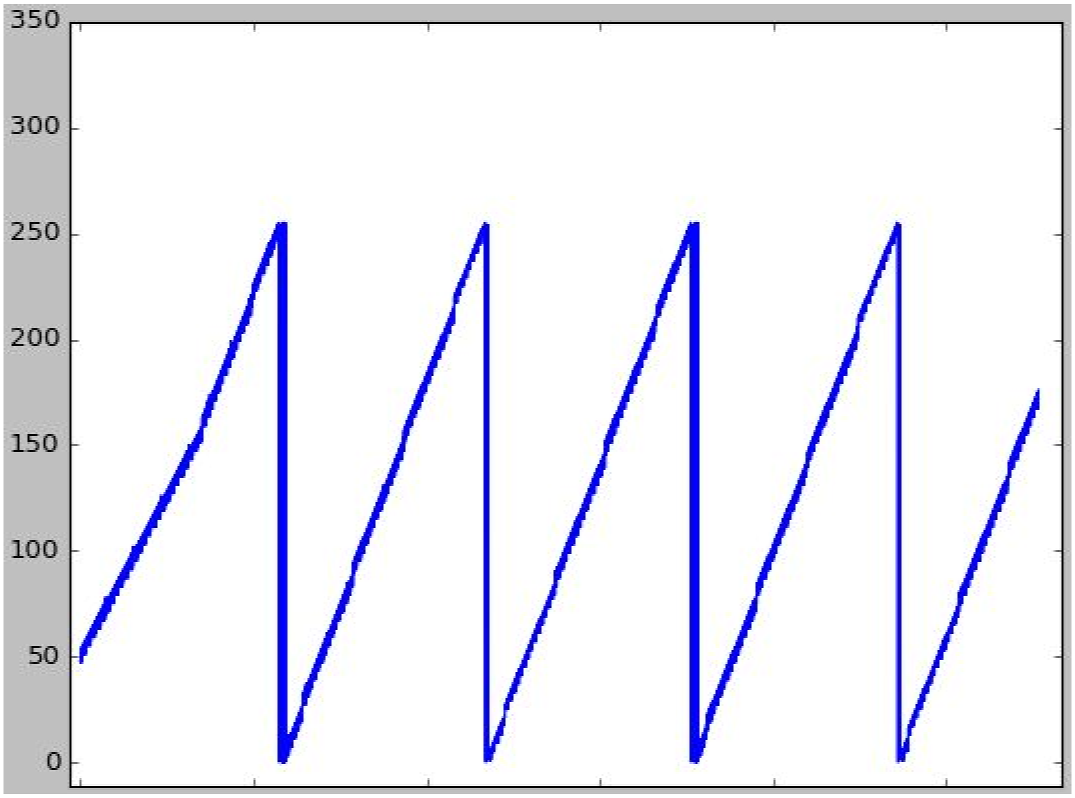
Results

So far, I have been able to use the provided ethernet module in the FPGA firmware to communicate with the computer allowing me to send data to the computer and set registers inside the FPGA that change parameters in the program.

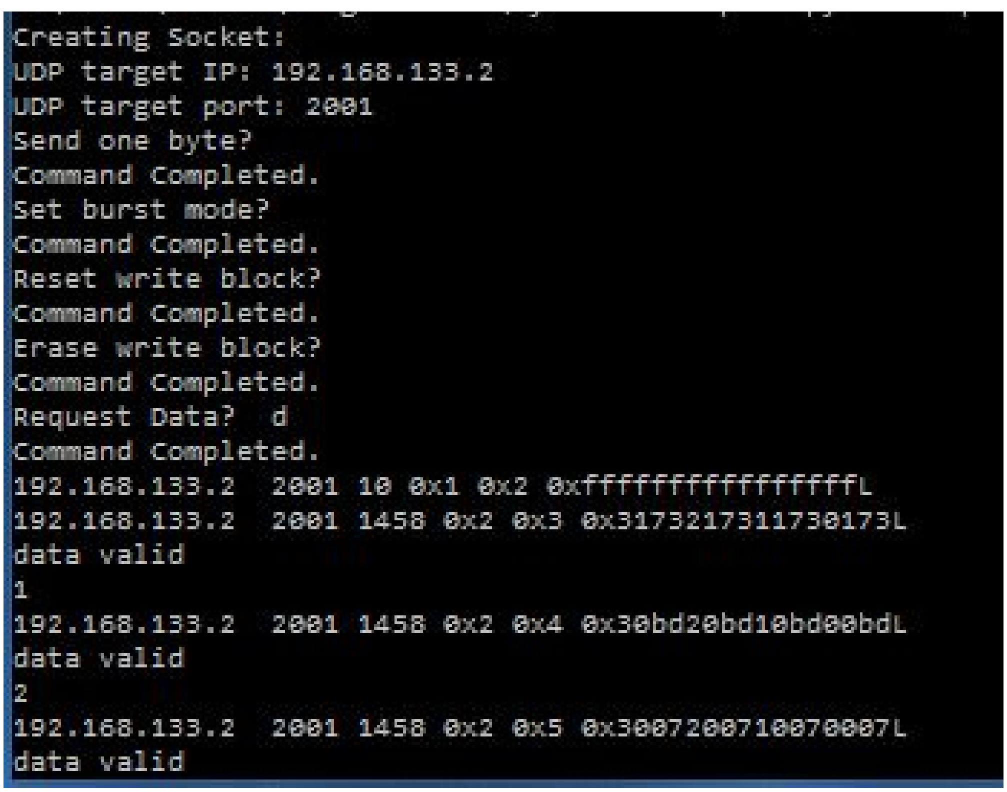
I have also developed a peak finder module that analyzes the incoming data for peaks and sends those portions out to the computer. By setting registers in the FPGA, the user can set the threshold point for a peak and select the minimum number of samples that should be sent back after a peak.

On the computer end, I have written a python script that reads the data from the FPGA and plots it on a graph to show the waveform. Python is a scripting language which means that the program is stored as a text file and compiled when it is being run. This is different than a compiled language where the program is compiled once and stored in an executable file. To read data from the Captan board, I send a small packet that tells the FPGA where to send the data. I then set a register that tells it to turn on the data sending module. Once the module is enabled, it will then start sending data packets and all I have to do is choose a few, read the values, and plot them on a graph.

Right now, I am currently testing the design using a firmware module I developed that sends a sawtooth signal. By setting the registers for the pulse finder and looking at the output from the python script, I am able to see how well my program is functioning.



Sawtooth waveform from a firmware test. Some of the inconsistency is from the FIFO memory overflowing.



```
Creating Socket:
UDP target IP: 192.168.133.2
UDP target port: 2001
Send one byte?
Command Completed.
Set burst mode?
Command Completed.
Reset write block?
Command Completed.
Erase write block?
Command Completed.
Request Data?  d
Command Completed.
192.168.133.2  2001 10 0x1 0x2 0xffffffffffffffffL
192.168.133.2  2001 1458 0x2 0x3 0x3173217311730173L
data valid
1
192.168.133.2  2001 1458 0x2 0x4 0x30bd20bd10bd00bdL
data valid
2
192.168.133.2  2001 1458 0x2 0x5 0x3007200710070007L
data valid
```

Output from the python script that obtains data from the FPGA.

248	28.358413	192.168.133.2	192.168.133.199	UDP	1500	2001 → 49387	Len=1458
249	28.358414	192.168.133.2	192.168.133.199	UDP	1500	2001 → 49387	Len=1458
250	28.358418	192.168.133.2	192.168.133.199	UDP	1500	2001 → 49387	Len=1458
251	28.358608	192.168.133.2	192.168.133.199	UDP	1500	2001 → 49387	Len=1458
252	28.358610	192.168.133.2	192.168.133.199	UDP	1500	2001 → 49387	Len=1458

Packets arriving to the computer from the FPGA as seen in the Wireshark network analyzer.

```
10111001 11101010 01100011 00101010 10001000 11001100 00000010 00000111
00000100 00000000 00100110 10111001 11101010 01100011 00101010 00000100
00000111 00000011 00000000 00100110 10111001 11101010 01100011 00101010
00000110 00000010 00001110 00010001 11111110 00001001 00000000 00010010
00001111 00000001 00000011 00000000 00000001 00000000 00000000 11111110
00000111 00000000 00010010 10111011 00000001 00000000 00000001 00000001
00000000 00000000
```

Raw data from the FPGA.

Getting Started Programming with FPGAs

Getting into the world of FPGAs takes some getting used to. It is a lot different from programming computers since you are designing a logic circuit instead of a collection of code that runs sequentially. One of the best resources out there in my opinion, is the Mojo Development Environment from Embedded Micro. (www.embeddedmicro.com) I consider it to be the “Arduino” of FPGAs. The development board is based off of the Spartan six and comes with a Arduino compatible microcontroller. The microcontroller acts as the JTag cable eliminating the need for an expensive programmer from Xilinx. In addition to the development board, there is an IDE very similar to the Arduino IDE. You can program the board from the IDE and it even comes with a basic logic analyzer. On the Embedded Micro websites there are plenty of tutorials to get started with as well as a selection of shields and add-ons.

Acknowledgements

Fermilab is operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy. QuarkNet is an educational program sponsored by the National Science Foundation and the Department of Energy whose aim is to support science education in schools by establishing a nationwide network of science teachers. This work was supported by CRADA FRA-2014-0022-02 agreement between Fermi Research Alliance LLC and the University of Notre Dame for the 2015-16 QuarkNet summer program.

Dr. Chris Stoughton
Ryan Rivera
Laura Thorpe
George Dzuricsko
The Quarknet Students
The Holometer Team

References

Atmel AVR 8-bit Instruction Set [Instruction set for Atmel 8-bit microcontrollers]. (2016). From http://www.atmel.com/images/Atmel-0856-AVR-Instruction-Set-Manual.pdf

Crab Pulsar. (n.d.). Retrieved July 21, 2016, from https://en.wikipedia.org/wiki/Crab_Pulsar

Field-programmable gate array. (n.d.). Retrieved July 22, 2016, from https://en.wikipedia.org/wiki/Field-programmable_gate_array

Kumar, N. S., Saravanan, M., & Jeevananthan, S. (2010). *Microprocessors and microcontrollers*. New Delhi: Oxford University Press.

X86 instruction listings. (n.d.). Retrieved July 22, 2016, from https://en.wikipedia.org/wiki/X86_instruction_listings