Open Flow Controller and Switch Datasheet

California State University Chico

Alan Braithwaite

Spring 2013

Block Diagram

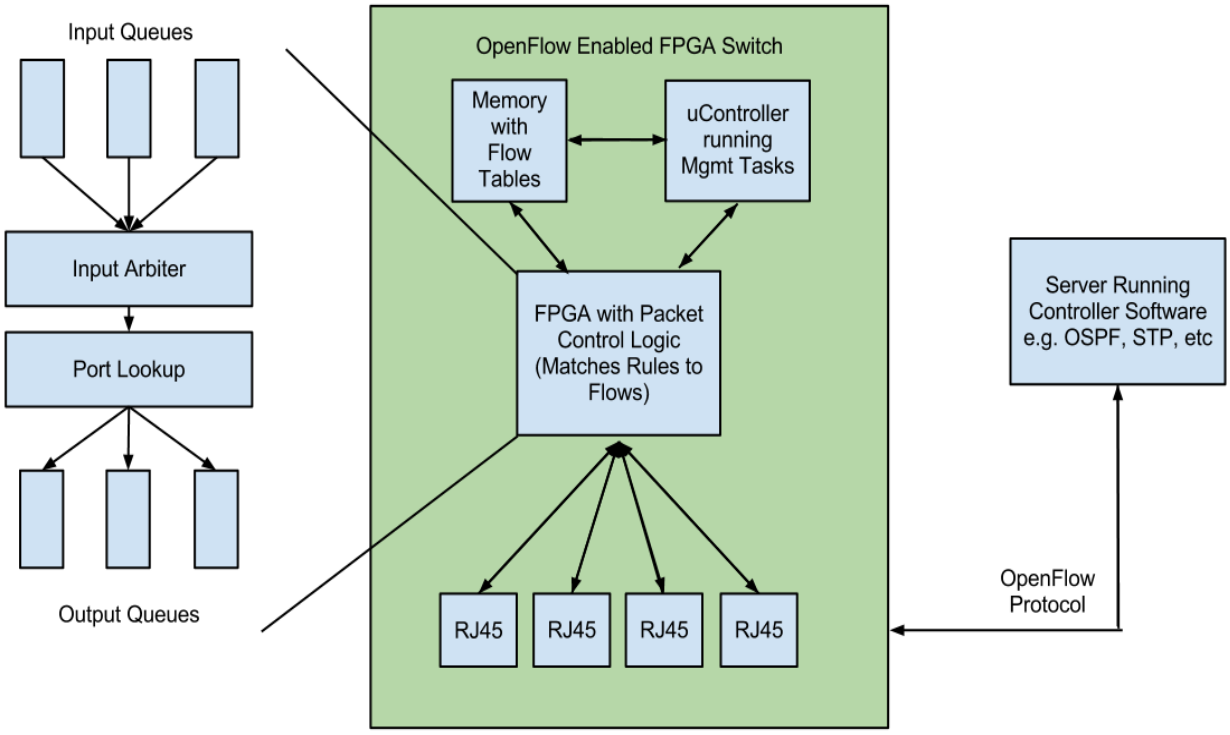


Figure 1. High Level Block Diagram

The project will consist of a network development board which has an FPGA and multiple network interfaces. It will be a PCI-E board for ease of programming and use. The board I will use in particular is the NetFPGA board developed by a team of researchers from Stanford, MIT, Google, and other major schools and corporations. The primary objective is to write a functional network switch in VHDL that fully supports the OpenFlow protocol. Once that's functional and tested, I plan on writing a custom controller and extending the capabilities of the switch. Figure 1 shows the high level block diagram of the system I will design and build.

System Schematic

An OpenFlow switch is a type of ethernet switch that contains flow tables which match packet headers to actions and apply them. I will be implementing the switch on a NetFPGA development board in VHDL. The NetFPGA development board provides all the necessary hardware for experimental development of network hardware. The switch logic will be implemented in VHDL on the

FPGA and has a minimum set of requirements defined by the OpenFlow protocol to be considered an OpenFlow compatible switch. Below is the high-level schematic of the development board.

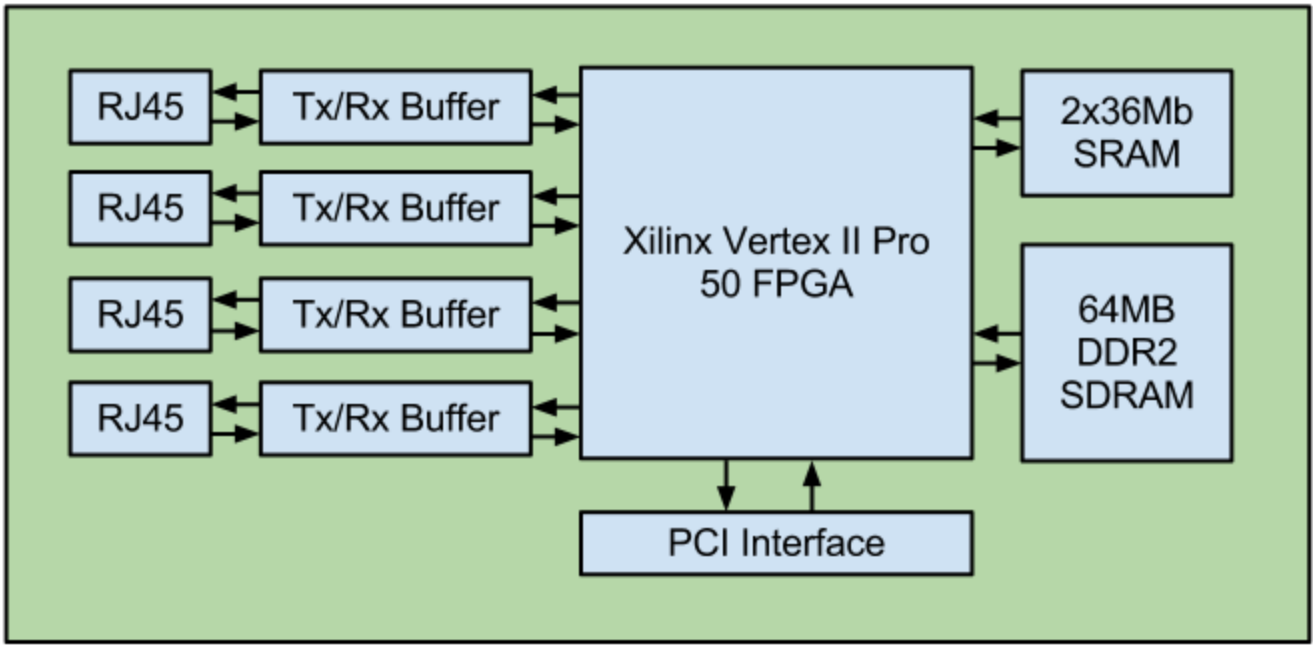


Figure 2. NetFPGA Schematic

The board communicates entirely via parallel synchronous address and data busses with all the interfaces to the Xilinx FPGA. The PCI interface makes interaction with the host computer easy and incredibly versatile compared to serial interfaces like USB. The ethernet buffers are managed by a Broadcom BCM5464SR Transceiver chip. All devices are memory mapped from the perspective of the FPGA which makes writing the VHDL easy as all the programmer has to worry about is reading and writing to the right buffer in memory. The board has linux drivers written for it that make programming the FPGA and communicating with the software running on the computer clean and simple.

A. Hardware

The hardware used in this project will be the NetFPGA board to build the switch. It has all the necessary components to allow versatile experimentation and development of network switch processors. The hardware requirements of the project dictate that I have an FPGA fully capable of handling all the logic required by an OpenFlow switch and should be fast enough to process packets at line rate. This board is uniquely designed to fit that bill. The following is a list of the hardware on the board taken from the NetFPGA website describing the hardware. [4]

- Field Programmable Gate Array (FPGA) Logic
    - Xilinx Virtex-II Pro 50
    - 4,176 Kbit block RAM
    - 53,136 logic cells
    - 2 PowerPC Cores
    - Fully programmable
- Gigabit Ethernet networking ports
    - Broadcom: BCM5464SR
    - Connector block on left of PCB interfaces to 4 external RJ45 plugs
    - Interfaces with standard Cat5E and Cat6 network cables
    - Wire-speed (Line rate) processing on all ports at all time using FPGA logic.

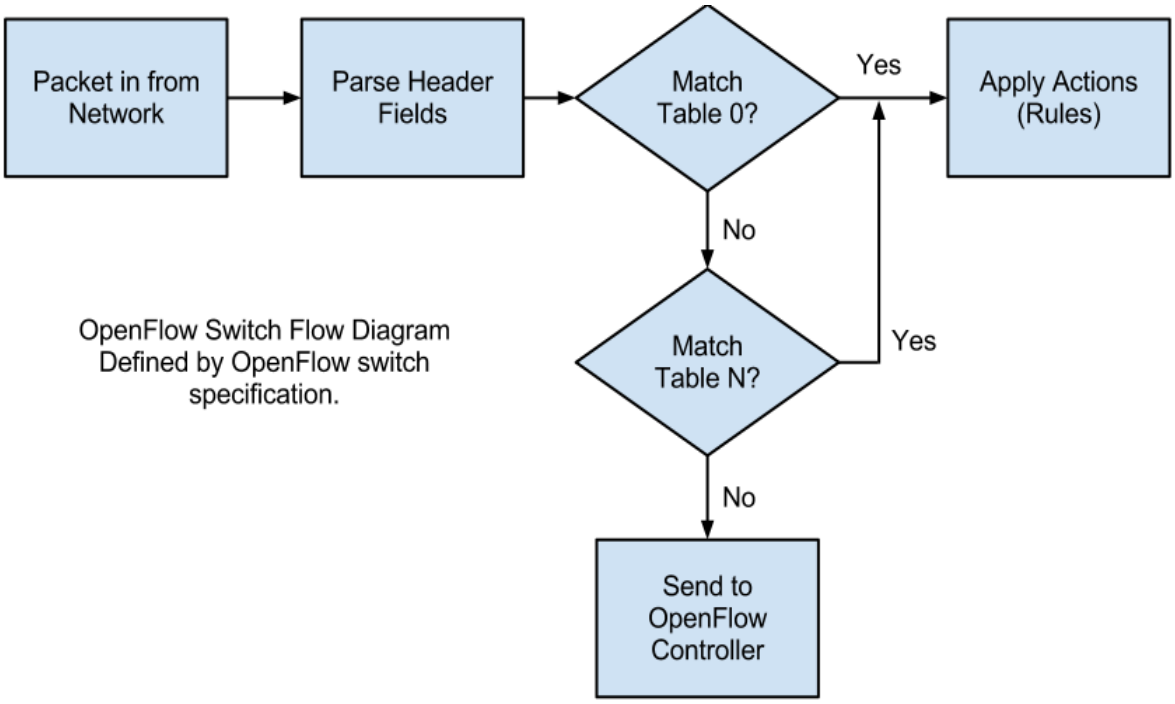
- Static Random Access Memory (SRAM)
    - Cypress: CY7C1370D-167AXC
    - Zero-bus turnaround (ZBT), synchronous with the logic
    - Two parallel banks of 18 MBit (2.25 MByte) ZBT memories
    - Total capacity: 4.5 MBytes
    - Suitable for storing forwarding table data
- Double Data Rate 2 RAM (DDR2 RAM)
    - Micron: MT47H16M16BG-5E
    - 400 MHz Asynchronous clock
    - 25.6 Gbps peak memory throughput
    - Total capacity: 64 MBytes
    - Suitable for packet buffer storage
- Standard PCI Bus Interface
    - Enables fast reprogramming/reconfiguration without use of JTAG
    - Provides CPU access to memory-mapped registers and memory on the NetFPGA hardware
- JTAG Debugging Port
    - Allows for standard debugging through Xilinx Software

B. Software

The primary focus of my project will be implementing the OpenFlow switch in VHDL. OpenFlow is a protocol that was devised by scholars at institutions across the country as a way of creating a network infrastructure more open to experimentation of new protocols. They have already defined minimum requirements for an OpenFlow switch but leave the implementation details up to the

switch designers. At the highest level, the switch must be able to process OpenFlow packets and update the flow tables internally according to those packets. Figure 3 shows what must be implemented in VHDL for the switch.

An OpenFlow switch consists of the forwarding element and the secure channel (used to communicate with a controller). The forwarding element of the switch consists of the pipeline and the flow tables which are used to map packet headers to actions. The secure channel will be managed by the host system but will operate through the switch via OpenFlow packets.



Source: www.openflow.org/documents/openflow-spec-v1.0.0.pdf

Figure 3. OpenFlow Switch Pipeline

The VHDL code will require a number of different modules that each represent a piece of the switch pipeline. I have defined the modules below without any consideration for how much time each takes for the time being. Each module has a specific task that it's responsible for before passing on the packet.

- Input Arbiter
    - Determine which packet buffer to read next packet from

- Header Parser
    - Parse the headers off the packet
    - Determine which flow tables to look at

- Table Lookup
    - Loop through tables looking for matching table
    - Queue in Action controller to be processed
- Action Controller
    - Apply actions defined in flow table for currently processed packet
    - Queue in Output Arbiter
- Output Arbiter
    - Queue packet to port defined by Action Controller

i. Flow Tables

A flow table row must contain at least the following three fields:

- Packet Header to match the flow
- The rules/actions to apply to that flow
- A counter to track the statistics of that flow

Match Fields	Counters	Instructions

Table 1. An Example of a Flow Table

There are three basic actions a switch must implement before it can be considered an OpenFlow switch:

- Packet Forwarding at Line Rate
- Encapsulate and forward un/matched packets to the Controller.
- Drop un/matched packets

Whether or not a unmatched packet gets dropped or forwarded to the controller is a configuration option that can be set on the switch.

Openflow does not define what headers are necessary to be able to read and match, but in order to have any kind of useful switch the ability to read the following headers should be implemented:

- Ethernet
- IP
- TCP
- VLAN

Without support for these four basic headers, it would be extremely difficult to make a useful switch or router given the current model of networking. Implementing more headers is optional and encouraged, but for the scope of my senior project, I will stick to getting these four working and operational first, then expand upon it.

ii. Secure Channel

The method for communication between the controller and the switch is defined by the OpenFlow protocol which is a special type of packet that gets sent via the secure channel to manage the switch or process packets. These packets contain the control signals necessary to update/modify flow tables, establish a connection, forward packets to process and gather statistics data. Figure 4 is an example of how the switch communicates with the controller via the secure channel.

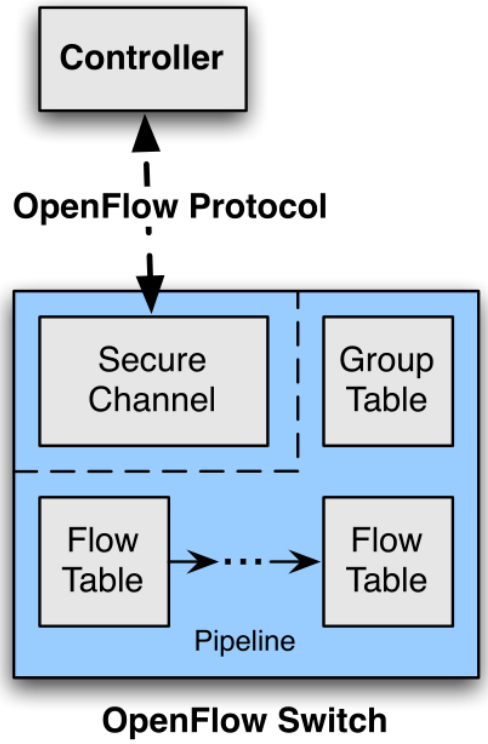


Figure 4. Switch Communication with Controller. [2]

The secure channel is a TLS encrypted channel that goes between the OS/Firmware on the switch and the controller. In my NetFPGA implementation, the a packet that needs forwarding to the controller will be passed to the host system through PCI/DMA, prepared for transport by linux, then requeued into the switch pipeline. The switch will then recognize that it is an OpenFlow packet and must forward that packet to the external controller.