Encouraging Reusable Network Hardware Design

G. Adam Covington, Glen Gibb, Jad Naous, John W. Lockwood, Nick McKeown

Stanford University
{gcoving, grg, jnaous, jwlockwd, nickm}@stanford.edu
http://netfpga.org

Abstract— **The NetFPGA platform is designed to enable students and researchers to build networking systems that run at line-rate, and to create re-usable designs to share with others. Our goal is to eventually create a thriving developer-community, where developers around the world contribute reusable modules and designs for the benefit of the community as a whole. To this end, we have created a repository of "User Contributed Designs" at NetFPGA.org. But creating an "open-source hardware" platform is quite different from software oriented open-source projects. Designing hardware is much more time consuming–and more error prone–than designing software, and so demands a process that is more focussed on verifying that a module really works as advertised, else others will be reluctant to use it. We have designed a novel process for contributing new designs. Each contributed design is specified entirely by a set of tests it passes. A developer includes a list of tests that their design will pass, along with an executable set of tests that the user can check against. Through this process, we hope to establish the right expectations for someone who reuses a design, and to encourage sound design practices with solid, repeatable and integrated testing. In this paper we describe the philosophy behind our process, in the hope that others may learn from it, as well as describe the details of how someone contributes a new design to the NetFPGA repository.**

I. Introduction

The NetFPGA platform is an "open-source" platform for teaching and research in networking hardware [1] [6]. The NetFPGA has two defining characteristics:

1) **Line-rate.** A student in a class, or a network researcher, can create new applications that run at line-rate. The current NetFPGA board has four ports of Gigabit Ethernet, and it is common for students to write applications (e.g. an Internet router or intrusion-detection system) that run at the full line-rate.
2) **Open-source Hardware.** There have been few, if any, successful "open-source hardware" platforms in which users contribute hardware design for reuse by others. Our goal is to create a thriving developer community, where students and researchers contribute reusable designs.

Unlike in the open-source *software* community, where standard practices have evolved, there isn't a standard practice for "open-source hardware". So what's different? The primary difference is the huge time it takes to write a hardware module (e.g. in Verilog and VHDL) that meets timing, and fits within the target device.[1] While the benefits of re-use are enormous (in fact, even greater in hardware than in software because of the potential time saved), there is greater skepticism as to whether a hardware module actually works because of the difficulty of getting it right, and the time taken to fix a broken design. This makes designers very wary of using modules contributed by others. Re-use is even less likely for entire hardware system designs, unless the system is very well specified. For example, how do we know if a contributed Internet router design implements IPv6, ICMP messages, Ethernet spanning tree, and so on...? While in a software system it is relatively straightforward to understand the functionality by reading the code, it is generally much less obvious when reading hardware descriptions.

So how can we get designers comfortable reusing designs from others? This is the key problem we are addressing here, and the purpose of this paper.

Our approach, which we call *Test-Driven Design*, is very familiar to hardware designers: Specify reusable designs and modules by the set of tests they pass. In other words, the developer of a reusable design is expected to clearly document the set of tests their design passes (e.g. "IPv4 packets from 64-1500bytes long"), and supply the set of tests that can be repeated by the designer who is reusing their code. Our experience so far has been that this creates unambiguous specifications of a reusable design, sets clear expectations of how complete (or not) the design is, and instills greater confidence for whoever is reusing the code.

We have noticed two common "traps" that hardware designers fall into when creating a reusable design. First, they describe a set of features of their design, which may or may not be implemented correctly. For example, if a designer says: "My design implements ICMP", how do we know exactly what ICMP messages it really supports, or under what conditions? If we find the design lacking, how do we know what support to expect from the designer? If, instead, the designer includes a set of tests that the design will pass, along with the test itself, then we can be quite sure what to expect, and will feel justified when going back to the designer to ask for help when a test no longer passes. Second, designers tend to mistakenly think they are contributing to a "community tree", rather than contributing a reusable module or design.

[1]For comparison, students in our classes design the forwarding path of a router in software in 7 days, whereas it takes them 6 weeks to create the same design in hardware.

While each developer might maintain their own local tree, we find that standalone designs, along with the tests they pass, leads to a cleaner process and more reusable designs. And so in summary, we conclude that in order to build a developer community, everything in our process should follow from the premise that all contributed designs are specified by the set of tests that they pass.

II. NetFPGA

The NetFPGA platform was originally built at Stanford University with funds from NSF to support teaching of a networking class, *CS344 Build an Internet Router* [3]. A first version of the hardware was designed in 2001 based on 10Mb/s Ethernet. Prototypes were used to teach a graduate course in 2003, and courseware for the class was made available on the web. A second version of the NetFPGA that used Gigabit Ethernet was developed in 2005-2006 [1] [7]. The current version of the hardware was made operational after manufacturing cards in 2007 and tested by a group of alpha users from ten different universities.

The NetFPGA beta program was started in January of 2008. With support from industrial partners that included Cisco, Google, Huawei, Juniper, Agilent, Micron, Cypress, and Broadcom; additional NetFPGA cards were built for the program and distributed through Digilent Inc. The open-source gateware, software, and hardware were made freely available on the NetFPGA website [6].

Eleven hands-on tutorials were delivered on four continents to over 300 professors and students [2]. For each tutorial, a laboratory with ten NetFPGA systems was deployed to provide the students with a hands-on experience using the NetFPGA. Courseware is available online.

About 1000 NetFPGA boards are in use at over 150 universities and research institutions across the North America, South America, The United Kingdom, Europe, India, China, and Australia. We recently released Version 2.0 of the gateware and software. Major changes include: Simpler installation using yum, improved regression tests, and improved documentation.

A. Contributed Designs

The NetFPGA includes two types of Contributed Designs:

1) Reference designs. These are basic designs maintained by a core group of developers, are designed to run at line-rate, and are expected to be re-used by many. To-date, the NetFPGA release includes five main reference designs: (1) The IPv4 router, (2) A Traffic Generator, (3) A Programmable 4-port NIC, (4) An OpenFlow switch, and (5) A hardware-accelerated Linux router.
2) User-contributed designs. These are modules and complete designs contributed by the NetFPGA developer community, including our own students. We expect this repository to grow rapidly; to-date it includes a Netflow probe, an RCP router, a buffer-sizing router, a router with DRAM packet buffers (instead of SRAM), a wireless interface, a router with support for deficit-round-robin scheduling (DRR), and so on. A full list can be found at [5].

III. Gateware and Software

One of appealing aspects of the NetFPGA Platform is the availability of the Verilog gateware, and the accompanying software. The packages that are released on the website, NetFPGA.org, contain the source code for projects, and an environment that allows researchers to synthesize, simulate, and verify in hardware the applications and features they create. The gateware itself is designed in a modular fashion to allow users to create and connect modules in new configurations. Most designs use the reference pipeline.

The NetFPGA release has three major components. The first is the kernel module that is used to communicate with the NetFPGA hardware. It allows the bitfiles for the FPGA to be loaded through the PCI bus of a Linux based PC. In addition, it allows software programs to communicate with the NetFPGA through a register interface implemented using shared memory. The reference systems use the DMA to send and receive packets from the card. The register system is used to read statistics counters and to write control data from software running on the host to the hardware. The second part of the NetFPGA release are the common utilities used to communicate with the card. These utilities include but are not limited to a bitfile download utility and register read and write programs. The third part of the release is the reference pipeline, described next.

IV. Reference Pipeline

Figure 1 shows the NetFPGA reference pipeline. Modules in the pipeline are interconnected using two buses: the *packet bus* and the *register bus* [4].

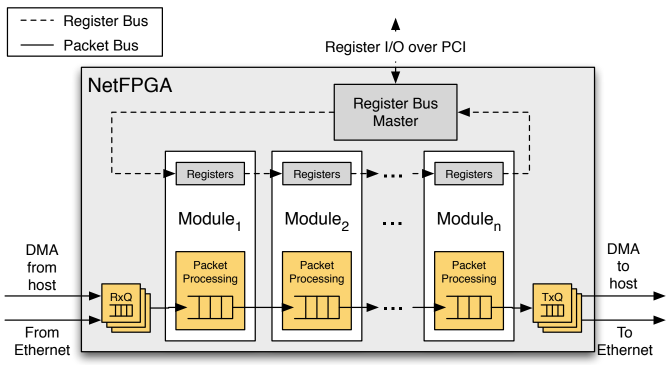

Fig. 1. The modular pipeline utilizes two buses: the Packet Bus and the Register Bus.

A. Packet Bus

The packet bus interface sends data between packet processing modules. A series of 64-bit words, consisting of the header(s) and payload of a packet, are processed as data streams through each module. Modules can split the header and payload, but there is no requirement to do so. Likewise, modules can use FIFOs to store some or all parts of a packet, but they usually do not. The total latency of the system can be easily determined by adding the number of clocks required for a packet to stream through each module in the pipeline.

B. Register Bus

A common register interface exposes the hardware's registers, counters, and tables to the software and allows software to modify them. This is achieved by memory-mapping the internal hardware registers. The memory-mapped registers appear as I/O registers to software. The software can then access the registers using ioctl calls.

The register bus pipelines reads and writes through a ring of modules. One module in the chain initiates and responds to requests that arrive as PCI register requests from software. Any stage on the chain is allowed to issue register access requests, allowing information to trickle backwards in the pipeline. This allows module i to get information from module $i + k$.

V. Test-Driven Design

The most critical part of any NetFPGA project or module are the tests. This follows from our test-driven design approach for creating open-source hardware. The regression tests are created by the developer and have two specific tasks: 1) define the operation of their project and 2) verify the installation of a project. Regression tests allow the community as a whole to have a high level of confidence about the design, functionality, and installation of a project. Without tests it is difficult to verify the operation of hardware modules created by other developers. Without simple verification of modules and projects, users are reluctant to re-use hardware designs.

Every function of a module or project must have a regression test associated with it. Each test is defined by the overall goal and the process used to test the function. The NetFPGA package provides Perl modules that allow users to send/receive live packets and read/write registers of the NetFPGA. As an example, the Packet Generator has four regression tests written using the Perl libraries. These tests run live network traffic to verify the ability of the hardware to send PCAP data, utilize the rate limiter, perform iterations, and capture packets. Below is the description of the Packet Generator iteration test.

Name
    test_iterations

Description
    Load and send two PCAP files from nf2c0 and nf2c1 with a specified number of iterations for each. Verify the packet sent counters.

Process
1. Initialize NetFPGA hardware
2. Load two PCAP files for nf2c0 and nf2c1
3. Set 10 iterations for nf2c0 and 100 iterations for nf2c1
4. Run Packet Generator (send traffic)
5. Check counters to verify the number of packets sent

Within the reference router project there are a total of 16 tests that vary from testing ARP misses to longest prefix match hits. All of the regression tests are documented on the Wiki page associated with each project.

VI. Project Directory Structure

The directory structure for contributed projects follow the structure of the reference projects.

NF2	{base directory}
projects	{project directory}
<project_name>	{contributed project}
doc	{documentation}
include	{project.xml, project specific module XML}
lib	{perl and C headers}
src	{non library verilog}
synth	{all .xco files}
regress	{regression tests}
verif	{simulation tests}
sw	{project software}

Documentation for each project is located in the project's documentation directory (doc). The include directory contains a project XML file (project.xml) that names and describes the project, lists all of the Verilog library modules used by the project, and defines the location of all module registers within the project. The include directory also contains all module XML files needed for project-specific modules. All project specific modules (i.e. new or modified library modules) are located in the source directory, src. The makefiles for the NetFPGA platform ensure that the modules in the src directory of a project will override any module with the same name that is included from the library modules. For instance, the Packet Generator uses a modified output queues module that is found in the NetFPGA library. This modified library module is found in the src directory of the Packet Generator's project directory. The include directory also contains the Packet Generator's files specifying the registers that the additional Verilog modules utilize.

The library directory (lib) is used by the NetFPGA scripts to store the Packet Generator's register files for both C and Perl. This directory and the register files are automatically generated when running simulations or synthesis. The register files allow software to utilize registers by name rather than by address. This enables the ability to modify register addresses without having to worry about changing hard coded address values.

The synth directory is where all Xilinx .xco files are located. These files are used to create the necessary cores from the Xilinx Core Generator. This is also where the project is synthesized. After synthesizing and creating the project's bitfile (i.e. running `make` in the synth directory) the regression tests can be performed on the newly generated bitfile.

The regression tests, found in the regress directory, are used to specify a projects capabilities and verify the functionality of a project as it is running in the NetFPGA hardware. Example regression tests can be found in all of the base NetFPGA package's projects.

The verification directory, verif, contains simulation tests used to verify the functionality of the gateware. The tests are contained in separate directories within the verification directory. The naming scheme of the test directories is test_major_minor, where the major and minor labels can be named whatever the project designer decides. The simulation script uses the major and minor in order to determine what simulation tests to run. If the script is invoked with only the major option specified then all tests that have a major equal to the one specified are run sequentially. If the major and minor options are both specified then only the test matching those labels will be run. Each test contains a file called make_packets.pl that uses Perl libraries included in the NetFPGA package to create packets and read/write registers for the simulation.

The software directory includes all software (functional and diagnostic) for that project. The Packet Generator has a Perl script called packet_generator.pl that is used to load PCAP files into the NetFPGA and start sending packets. Also located in this directory are example PCAP files that are used by the regression tests to verify the functionality of a generated bitfile.

VII. Contributing a Project/Application

Contributing a package back to the NetFPGA community is a simple process that involves four simple steps. These steps are: create a WordPress page on the NetFPGA website, create a NetFPGA Wiki page, create the package, and post the package.

A. Create a Project Summary Page

The projects and applications available for the NetFPGA platform are listed and indexed on the NetFPGA website using WordPress pages. A project page consists of a small icon that can be used to identify the project and a few lines that describe the features of the project.

B. Create a Wiki Page

Each project has its own Wiki page. This page describes the project, how to download the project, run regression tests, and use the hardware and software. In addition, each project can have additional sections that provide additional information such as references. This is were external users (i.e. users other than the project creator) will search for information regarding a project.

C. Creating a Package

The nf2_make_release.pl script is run to create a standard NetFPGA tarball package. This script is found in the bin directory of the NetFPGA tree. It parses an XML file that determines what to include in the package. As can be seen in the Packet Generator's project XML code (below) the bitfile is specified in addition to the name of the project to include in the package. Since each contributed project should only have one project per tarball, there should be only one project specified. The standard open-source BSD-style license text is also included in the package.

```
<build>
  <base name="LICENSE" />
  <base name="bitfiles/packet_generator.bit" />
  <release name="packet_generator" version="1.0">
     <project name="packet_generator" />
  </release>
</build>
```

D. Post the Package

The last step in contributing a project is posting the package online. Authors are encouraged to post their package on their own website. It is important that the project is publicly available and the instructions in the wiki page describes the process of obtaining the package.

VIII. Conclusion

Encouraging module reuse in an open-source hardware platform is a difficult process. Hardware designers are reluctant to reuse modules contributed by others unless the modules are well defined and work as expected. To achieve higher confidence and encourage sound design practices, we have created a test-driven design process. This process requires hardware designers create tests that define the features of their designs. Although the creation of tests add to the hardware designers burden, it is not an insurmountable task. Given the added benefit of the test-driven design process, the additional work is minimal. This design process, in combination with an open-source platform, helps create and sustain a healthy user community that contributes open-source designs.

References

[1] G. Gibb, J. W. Lockwood, J. Naous, P. Hartke, and N. McKeown. NetFPGA: An open platform for teaching how to build gigabit-rate network switches and routers. In *IEEE Transactions on Education*, August 2008.

[2] J. W. Lockwood, G. Gibb, J. Naous, G. A. Covington, N. McKeown, A. W. Moore, and D. Miller. Building Gigabit-rate Routers with the NetFPGA. www.netfpga.org/php/events/php, 2007-2008.

[3] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. NetFPGA - an open platform for gigabit-rate network switching and routing. In *International Conference on Microelectronic Systems Education*, 2007.

[4] J. Naous, G. Gibb, S. Bolouki, and N. McKeown. NetFPGA: reusable router architecture for experimental research. In *PRESTO '08: Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, pages 1–7, New York, NY, USA, 2008. ACM.

[5] NetFPGA Development Team. NetFPGA Applications website. http://netfpga.org/wordpress/category/applications/.

[6] NetFPGA Development Team. NetFPGA website. http://netfpga.org/.

[7] G. Watson, N. McKeown, and M. Casado. NetFPGA - a tool for network research and education. In *2nd Workshop on Architecture Research using FPGA Platforms (WARFP)*, Febuary 2006.