FlueNT10G Documentation

Andreas Oeldemann

Feb 04, 2020

Contents

FlueNT10G is an open-source, free-to-use FPGA-based network tester. FlueNT10G can precisely replay pre-recorded or synthetically created network traces, capture incoming network traffic and accurately record network latencies on a per-packet basis. The hardware design has been developed for the popular NetFPGA-SUME FPGA board. The goal of this documentation is to get you started as fast as possible and have you perform your measurements in no time.

Note: This documentation is work-in-progress.

All the information you need:

CHAPTER 1

Getting Started - What do I need?

FlueNT10G tries to exploit the flexibility and low cost of software wherever possible. However, to allow the generation of precise traffic patterns, to perform nanosecond packet timestamping and to determine accurate per-packet network latencies, we utilize the power of specialized hardware. Therefore, FlueNT10G's open-source software library is complemented by an (equally open-source) FPGA design, which performs all these time-sensitive operations.

Here's what you'll need (you'll find more details below):

- Standard PC with a free PCIe Gen3 slot (eight lanes will do)
- NetFPGA-SUME or (with some development effort) another Xilinx-based FPGA board
- License for Xilinx Vivado Design Suite
- License for Xilinx 10 Gigabit Ethernet MAC IP core

1.1 Computer Hardware

Most standard PCs with a free PCIe Gen3 slot will do. If you are using FlueNT10G with the NetFPGA-SUME board, here's some more information for you: the board utilizes eight PCIe lanes, so most slots on your motherboard should be fine. To ensure stable operation of the hardware, the board's *PCI Express Auxiliary Power* connector should be connected to your computer's power supply. The NetFPGA-SUME guys provide a list of motherboards, which have been tested with their FPGA.

1.2 FPGA board

FlueNT10G has been developed for the NetFPGA-SUME FPGA board, which is a popular open-source platform for FPGA-based network processing. Since the NetFPGA-SUME is so widely used in academia, chances are that there is one in your lab already and you can get started out of the box without a steep up-front investment.

If you do not own a NetFPGA-SUME board (and don't want to buy one), FlueNT10G can be ported to other Xilinx-based FPGA boards without too much development efforts. The design utilizes standard (*AXI4-MM*, *AXI4-Lite*, *AXI4-*

Stream) interfaces between IP cores, so ideally all you'll need to do is adjusting some constraint files. Please let us know if you decide to go down that road. Other people may benefit from this as well!

1.3 Xilinx Vivado Design Suite

To build the FPGA firmware, you'll need to use the Xilinx Vivado Design Suite tool flow. If you are in academia, Xilinx may be able to donate a license for your research work. FlueNT10G can be built with the latest Vivado 2018.3 release.

1.4 Xilinx 10 Gigabit Ethernet MAC

Unfortunately, Vivado does not come with a license for the Xilinx 10 Gigabit Ethernet MAC IP core. You'll need to obtain a license separately. Again, if you are in academia: contact Xilinx and hope for a donation!

CHAPTER 2

System Preparation

Before installing FlueNT10G, the host system (both hard- and software) must be prepared. We'll start with the hardware and then move on to the software.

2.1 Hardware

This documentation assumes that you are in the posession of a NetFPGA-SUME FPGA board. The board must be plugged into a PCI Express Gen3 (at least eight lane) slot. The NetFPGA guys provide a list of motherboards, which have been tested with their board. Make sure to attach the FPGA board's *PCI Express Auxiliary Power* connector to ensure stable operation.

2.2 Software

FlueNT10G is developed and tested on Ubuntu 18.04 LTS (Bionic Beaver). Please install the following dependency packages before continuing:

```
sudo apt update
sudo apt install -y unzip wget build-essential linux-headers-generic golang
```

2.2.1 Environment Variables

The following sections describe how to set the *FLUENT10G* and *GOPATH* environment variables.

FLUENT10G

To make things easy, we choose a working directory for FlueNT10G and set up an environment variable pointing to it. In this documentation, our working directory is *~/fluent10g*, but feel free to adjust. After creating the directory, we set up *$FLUENT10G* to point to the working directory. Updating the *.bashrc* will make that configuration persistent. If you are using a different shell, please update the commands accordingly.

```
mkdir ~/fluent10g
echo "export FLUENT10G=~/fluent10g" >> ~/.bashrc
source ~/.bashrc
```

GOPATH

The measurement applications, which control the behavior of the FlueNT10G network tester, are written in *Go*. When working with Go, we must set the *GOPATH* environment variable to point to the directory holding the source code and binary files of all Go projects. In this documentation, we set the *GOPATH* to *~/go* (which is actually the default value). Feel free to use any other path. You can find more information on the *GOPATH* environment variable here. Again, if you are not using bash, please update the commands accordingly.

```
mkdir ~/go
echo "export GOPATH=~/go" >> ~/.bashrc
source ~/.bashrc
```

2.2.2 Xilinx Vivado

To generate the FPGA bitstream, please install Xilinx Vivado 2018.3. Please ensure that a Xilinx 10G Ethernet MAC license is installed (separate purchase/donation required), otherwise you will encounter problems during synthesis/implementation.

2.2.3 Xilinx PCIe DMA driver

FlueNT10G depends on the Xilinx PCI Express DMA driver for data transfers between host system and the FPGA. We first clone the repository (containing different kinds of DMA drivers), which is maintained by Xilinx:

```
cd $FLUENT10G
git clone https://github.com/Xilinx/dma_ip_drivers
```

Compile and install the driver (we need the XDMA driver):

```
cd dma_ip_drivers/XDMA/linux-kernel/xdma/
sudo make install
sudo depmod
```

Load the driver and configure the system to automatically load the driver after reboot (poll mode is enabled to increase transfer performance):

```
sudo modprobe xdma poll_mode=1
sudo sh -c "echo 'options xdma poll_mode=1' >> /etc/modprobe.d/xdma.conf"
```

2.2.4 ZeroMQ (optional)

If you are planning to use the FlueNT10G Agent for communication with the device-under-test, the ZeroMQ messaging library needs to be installed. Please follow the instructions below to compile and set up the library:

```
mkdir $FLUENT10G/zeromq
cd $FLUENT10G/zeromq
wget https://github.com/zeromq/libzmq/releases/download/v4.3.1/zeromq-4.3.1.tar.gz
tar xfz zeromq-4.3.1.tar.gz
cd zeromq-4.3.1
./configure --prefix=`pwd`/install
make
make install
sudo cp ./install/lib/pkgconfig/libzmq.pc /usr/share/pkgconfig
```

CHAPTER 3

Installation

Warning: Before proceeding with the installation, make sure to complete the steps in *System Preparation*.

3.1 Cloning the Repositories

To clone the Git repository containing the hard- and software implementation of FlueNT10G to the working directory defined by the environment variable set in the *System Preparation* steps, execute the following commands:

```
git clone https://github.com/aoeldemann/fluent10g.git $FLUENT10G/src
cd $FLUENT10G/src
git submodule init
git submodule update
```

The Go package containing the software source code is located in a separate repository, which is included in the main repository as a Git submodule. The last two commands initialize this submodule repository and fetch the contents from GitHub.

3.2 Hardware

Moving on to the hardware . . .

3.2.1 FPGA Bitstream Generation

Before proceeding with the bitstream generation, make sure that Xilinx Vivado 2018.3 is installed on your system and that a license for the Xilinx 10G Ethernet IP core is available. The bitstream can then be generated using the following command:

```
make -C $FLUENT10G/src hw
```

Synthesis and implementation will take a while, so please be patient! :-) If everything goes well, the bistream file should be located at *$FLUENT10G/src/hardware/project/fluent10g.runs/impl_1/fluent10g_wrapper.bit*.

3.2.2 Programming the FPGA

First start a Xilinx Hardware Server:

```
hw_server
```

Open a new terminal and program the FPGA:

```
FPGA_HOST=localhost make -C $FLUENT10G/src/hardware program
```

Note: The *FPGA_HOST* environment variable value specified when running the *make* command determines the hostname of the machine running the Xilinx Hardware Server. Change the value if the FPGA that shall be programmed is plugged into a remote machine.

Note: Alternatively, the FPGA can be programmed using the hardware manager of the Xilinx Vivado GUI. To do so, open the generated project file located at *$FLUENT10G/src/hardware/project/fluent10g.xpr* in Vivado.

Finally, reboot the machine containing the FPGA board (PCI Express device reenumeration may be sufficient, however it did not work for our host system).

```
shutdown -r now
```

3.3 Software

All measurement applications, which control the behavior of the FlueNT10G network tester, rely on the *gofluent10g* Go package. To allow Go applications to use the package, it must be located in the *GOPATH* (see its configuration in the *System Preparation* section). Execute the following commands to create a symbolic link pointing to the *gofluent10g* package, which can then be found by other Go applications:

```
mkdir -p $GOPATH/src/github.com/aoeldemann
ln -s $FLUENT10G/src/software/gofluent10g $GOPATH/src/github.com/aoeldemann/
↪gofluent10g
```

Then, change into the package's directory and install its dependencies:

```
cd $GOPATH/src/github.com/aoeldemann/gofluent10g
go get
```

Done! Both hard- and software should now be ready to go.

CHAPTER 4

Examples

Currently, there is no API documentation ready yet. Please have a look at the following three example measurement applications, which should give you a solid idea on how FlueNT10G can be used:

4.1 Trace Replay

Source Code

This example measurement application replays a constant-bit-rate network trace (10 Gbps, 64 bytes packets) on all four interfaces of the network tester simultaneously.

4.2 Packet Capture

Source Code

This measurement application captures arriving network traffic on all four network interfaces for 10 seconds. It then writes the recorded data into four separate output network traces and prints out the number of packets captured on each interface. Each packet is accurately timestamped upon reception at the network interface, the resulting PCAP trace contains nanosecond resolution timestamps.

4.3 Latency Measurements

Source Code

This measurement application replays a trace file (constant-bit-rate traffic, 10 Gbps, 64 byte packets) on network interface 0. It expects the traffic to arrive back at the network tester at interface 1. Therefore, make sure that these two ports are either directly looped-back or configure your attached device-under-test accordingly. The network tester captures the latencies (= time from packet transmission until reception back at the tester) for every packet and writes the values to an output text file after the measurement (6.4 ns time resolution).