Pico Computing

M-Series Getting Started Guide

December 16, 2014

Contents

1 Overview

Thank you for choosing Pico Computing FPGA products. Pico offers an easily implemented, massively scalable FPGA-based approach to high-performance computing. This manual will help you get started using your Pico M-series module(s). Once you are finished with the installation instructions, you will have:

- A fully assembled, working hardware system.
- Device driver software for the FPGAs.
- The Pico API, a high-level C++ library for communicating with the FPGAs.
- The Pico Framework firmware, including DMA and multi-port memory components.
- A set of sample programs for testing your system and creating new projects.
- Full documentation on all of the above.

1.1 System Requirements

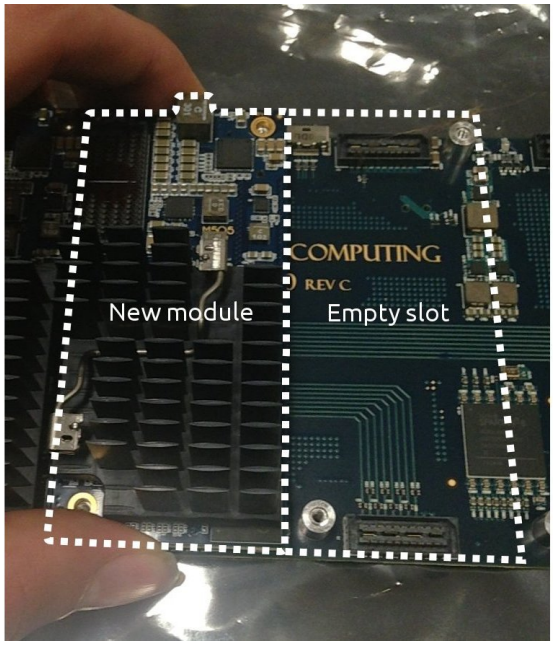
In order to operate a Pico Computing FPGA product, your system must meet the following requirements:

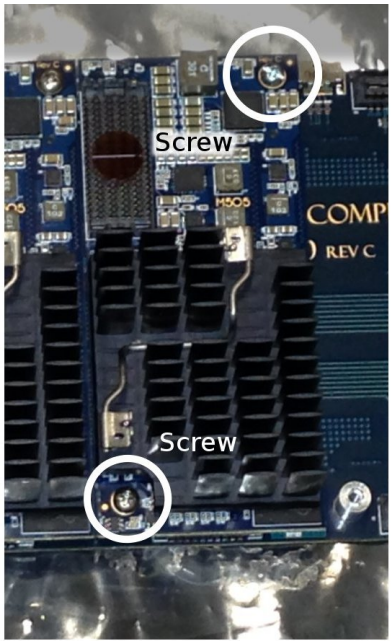
- System must have at least one available x16 PCIe slot for each backplane.
- Power supply must have 1 available PCIe power cable for each backplane (GPU style connector).
- Power supply must be able to deliver at least 240 W per backplane.
- System must have cooling fans pointed at the FPGA(s) in order to keep them under 85 C.
- System must be running the Ubuntu 64-bit desktop OS. We recommend version 14.04. If you need to run the Pico Framework on a different operating system, please contact Pico Computing support.
- We recommend an Intel motherboard chipset, but this is not a requirement.

2 Installing the Hardware

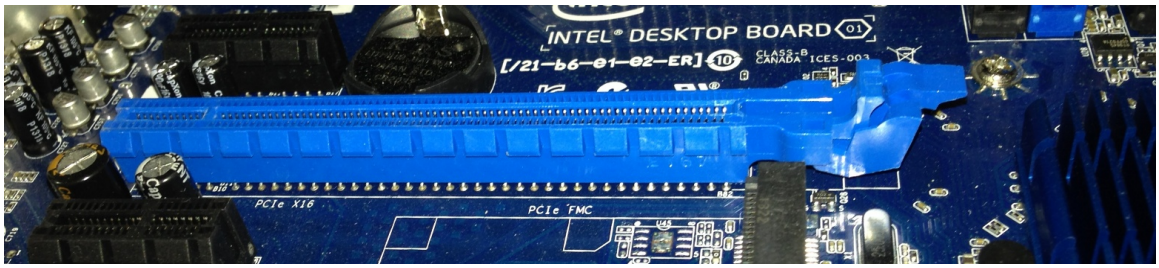
Take the boards out of their ESD bags, being careful to avoid electrostatic discharge. The small M-series boards, which hold the FPGAs, are called *modules* or sometimes *cards*. We call the large EX-series boards *backplanes*. Their purpose is to hold the modules and provide a physical medium for the modules to communicate with the rest of the system over PCIe.

First, completely power down your system. If your M-series modules were not shipped attached to the backplane(s), place each of the modules onto one of the available slots on a backplane and push down gently. Fasten in the modules with the screws provided.

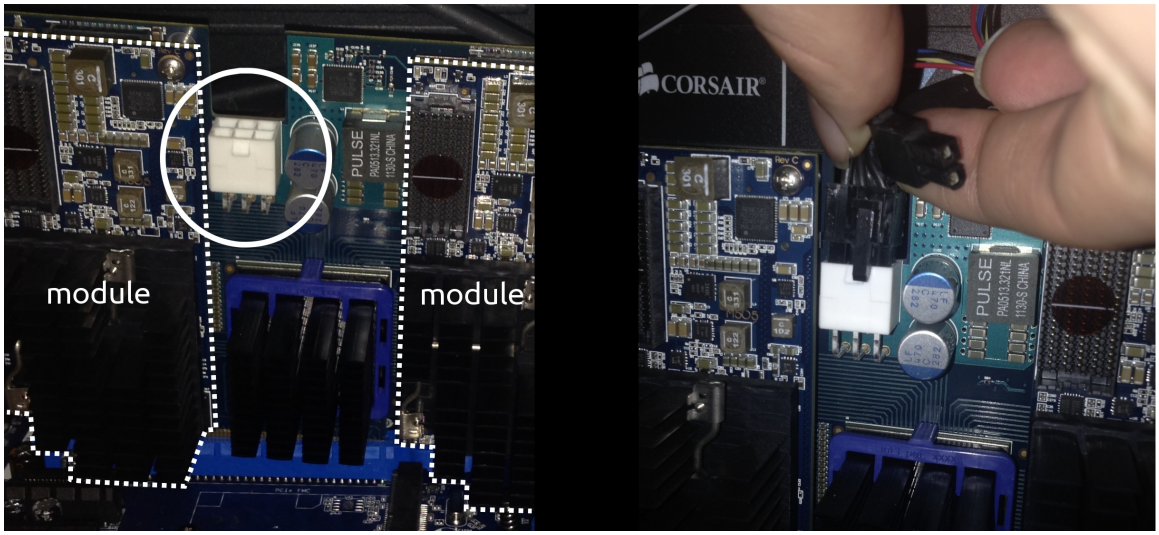




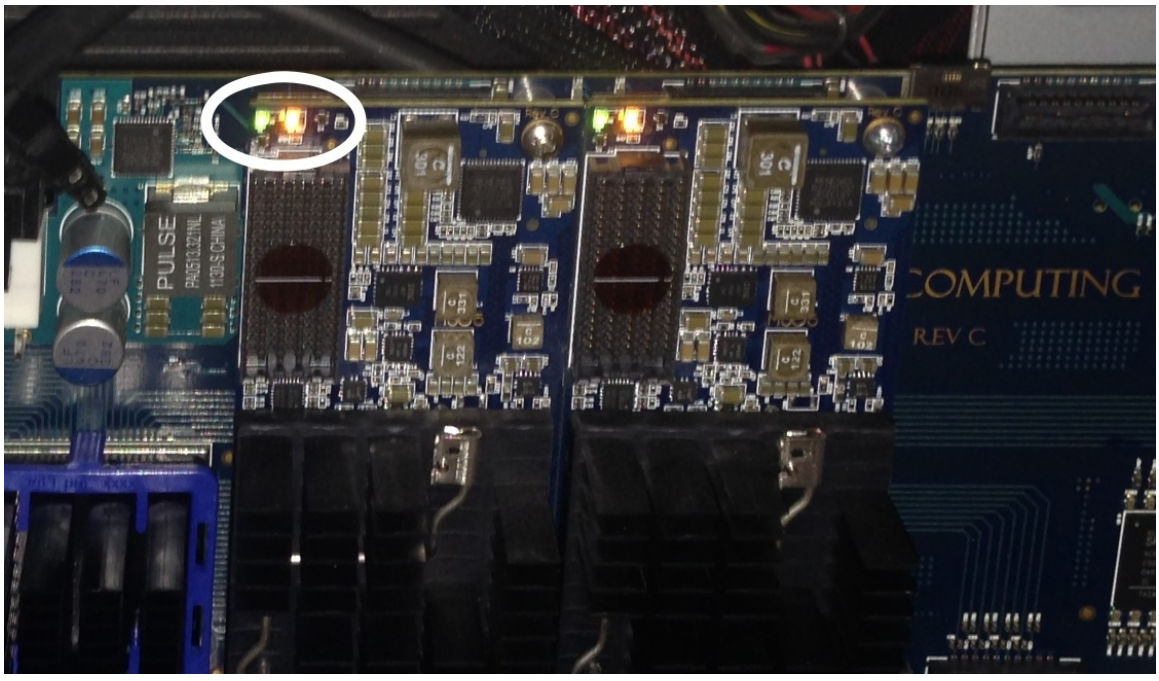
When you are done with a backplane, insert it into a PCIe slot on your motherboard (an x16 PCIe slot should be shaped like the blue object in the first picture below, although it can be a different color). Make sure there is a cooling fan near the FPGA(s) and that it is blowing directly on the heat sinks.



Plug in the PCIe power cable.



Finally, turn the system back on. The FPGAs should now have power and you should see a steady green and orange LED on each module, showing that the FPGA has power and is programmed. For example, the photo below shows how these LEDs look on an M-505 LX-325T module.



3 Installing the Software

This section describes the procedure for installing the Pico software on an Ubuntu Linux system.

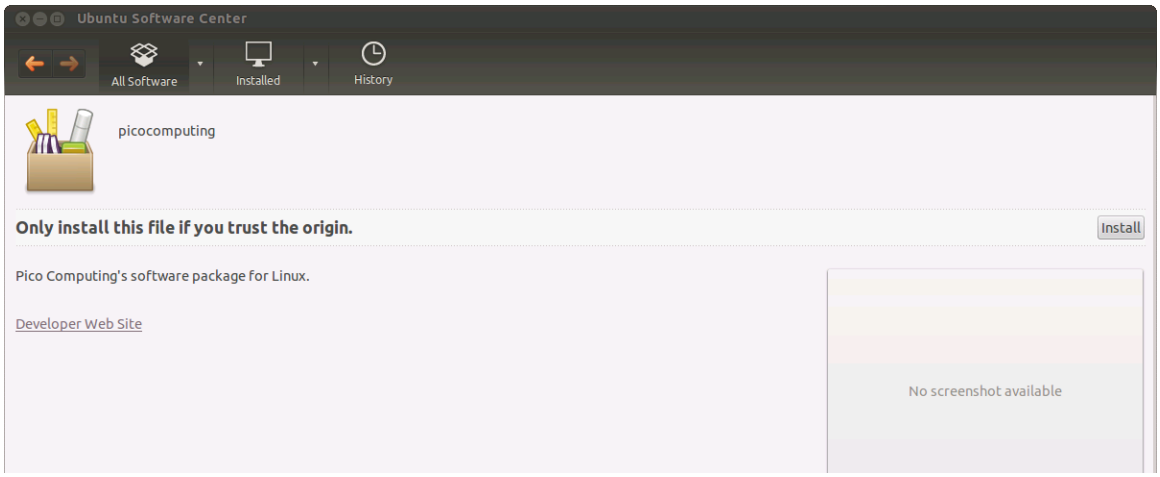
The Pico Framework is distributed as an Ubuntu package named picocomputing_W.X.Y.Z.deb, with W.X.Y.Z replaced by the current Framework version number. This file is available from the "Linux installer" link at `www.picocomputing.com/support/`. Download it and save it to an easily accessible

location. You can install the software either from the command line or from the GUI. We will describe both methods in the next two sections.

3.1 From the GUI

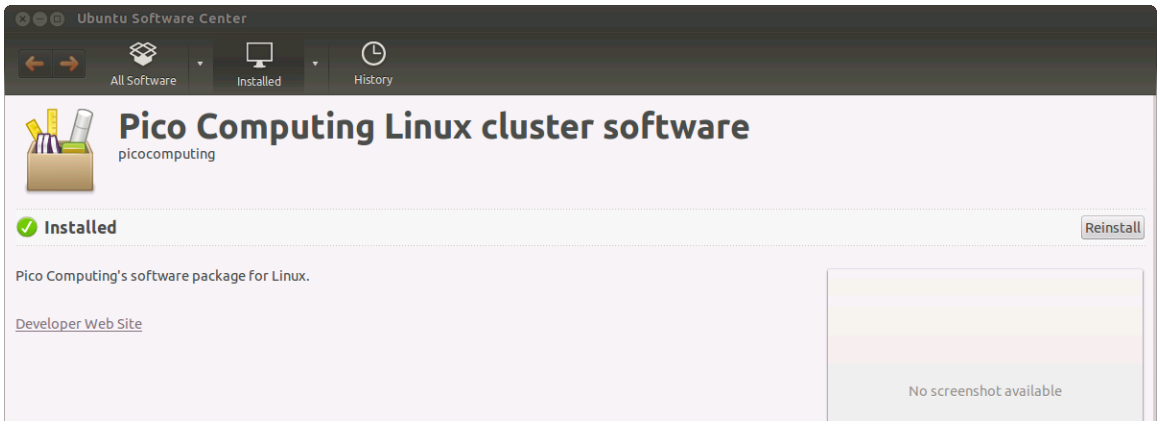
To install the software from the GUI, open the file you just downloaded. If it's a zip or other archive file, extract the files inside. Then, double-click on the .deb package. Ubuntu will show you a summary of the package information; it may say that it needs to download and install several dependencies. However, if it says something like "linux-headers is missing," you probably do not have the correct version of Ubuntu - make sure that you have Ubuntu 12.04.1 installed and running.

Click the *Install* button to start the installation.



Because you are changing the software on your computer, Ubuntu will ask for your password. It will then download any required packages and install them along with the Pico Framework. This process may take a minute or two.

When Software Center's work is complete, the screen should look like this:



Now, proceed to "Finishing the Installation."

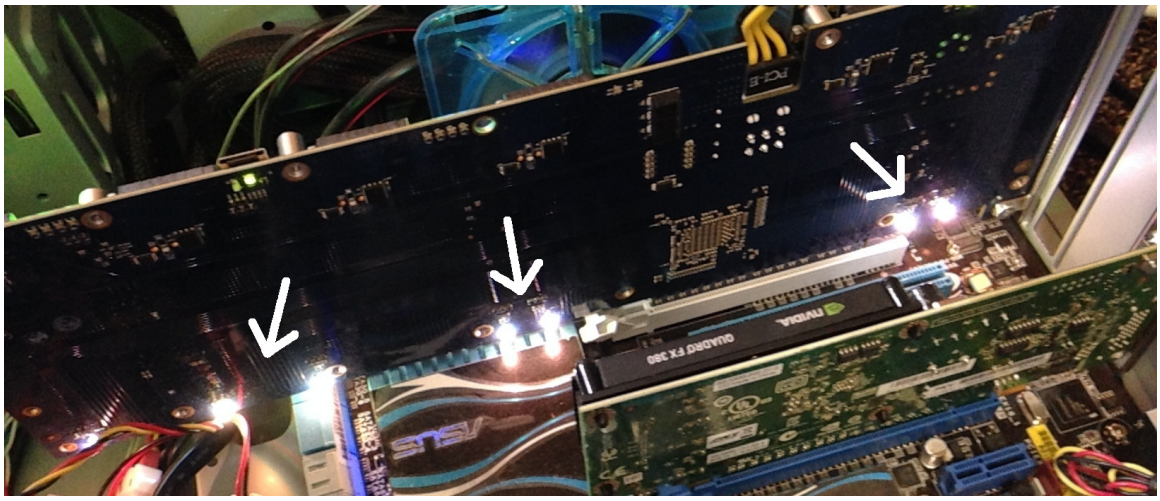
3.2 From the Command Line

To install the software from a terminal, cd to the location of the software, unzip it if necessary, and run dpkg on the resulting .deb file. For example, if you have downloaded version 5.2.0.0 of the Pico Framework as a zip package and stored it in ~/Downloads, you can install it with the commands:

```
cd ~/Downloads
unzip picocomputing_5.2.0.0_all.zip
sudo dpkg -i picocomputing_5.2.0.0_all.deb
```

3.3 Finishing the Installation

Once you have installed the .deb package, power cycle the system by shutting it all the way down and starting it up again. The software should now be fully installed. Within one minute of powerup, you should see bright, flashing multicolored LEDs confirming this. Here is how these lights look on an EX-500 test system in our office:

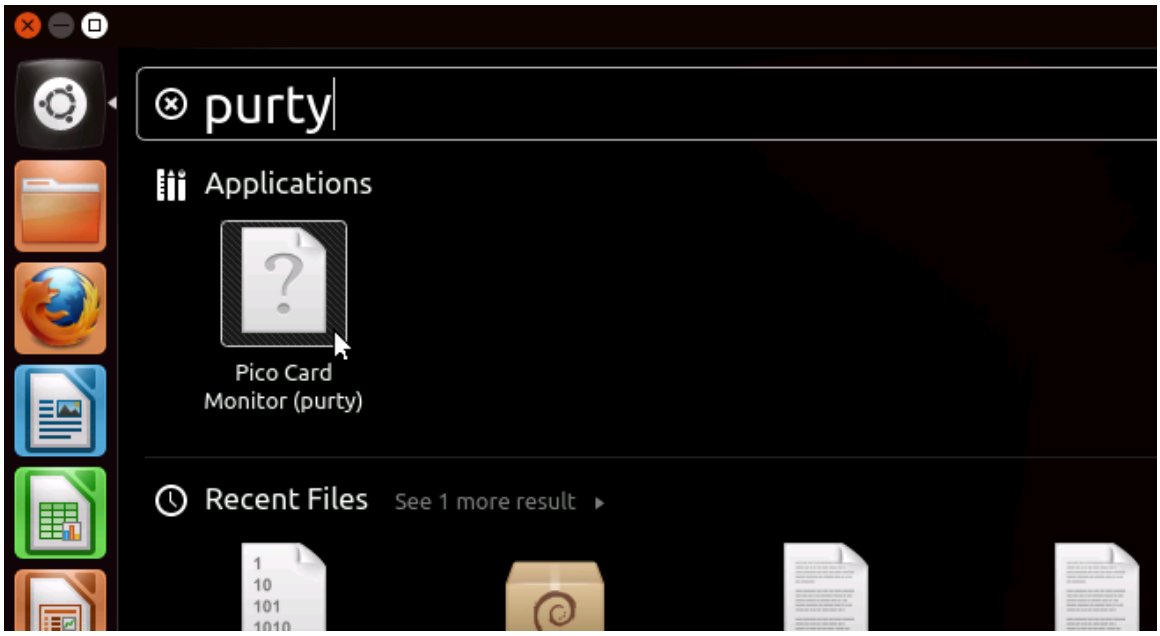


You should now verify that the firmware and software are correctly installed by running Pico's card monitoring program, called "Purty," as described in the next section.

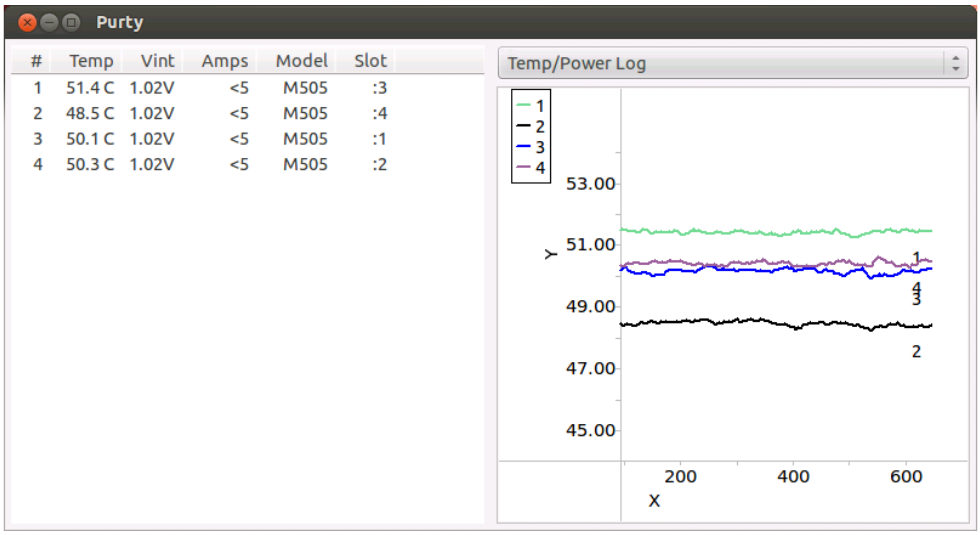
4 Monitoring Modules with Purty

Pico provides a program for monitoring the state of Pico modules, called Purty (an acronym for "Pico Utility for Reliability and Testing"). Purty reports the status and health of all the modules in the system, including each module's temperature, voltage, installation status, serial number, and physical location.

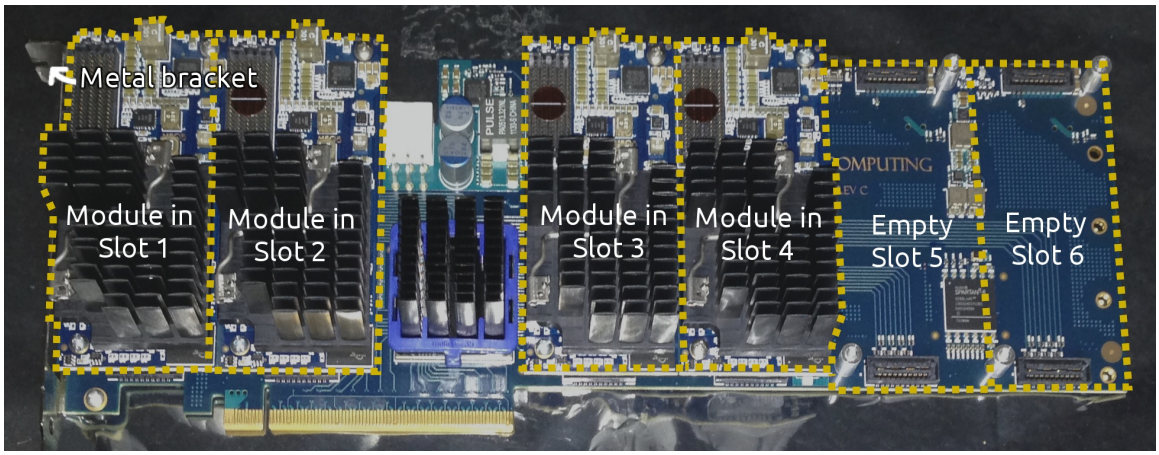
To start purty, click on "Dash home" at the top left corner of the screen and search for *purty*. You can also launch purty from the command line. It's located in `$PICOBASE/bin/purty`.



The Purty window should look something like the one shown below. On the left side, you can see a list of the modules in your system along with various information about their status (from the left, temperature, voltage, current, model, and physical location on the backplane). On the right, we have opened the temperature/power log from the selection menu in the upper right corner. By choosing appropriately from this menu, you can also access console output from each of the FPGAs.



This picture shows the physical locations of the six slots on an EX-500 backplane. As you can see, we have installed modules in slots 1, 2, 3, and 4 (thus the slot numbers in the Purty window above). Your system may look different.



5 Learning More About the Pico Framework

5.1 Documentation

After you've finished going through this getting-started guide, you will probably want to learn more about the details of the Pico Framework. Full documentation on the software and firmware parts of the framework can be found in the *PicoAPI* documentation file, which is located at `$PICOBASE/doc/` along with all of the other distributed documentation for the Pico Framework. This document provides a full description of the Pico Application Programming Interface (API) and how you can use it to deploy the capabilities of the Pico Framework. The software side of the Framework is written in C++.

Depending on your project goals, you may also be interested in the PicoMemory documentation, which explains how to use the module's off-chip memory (if applicable). Further documentation files in `$PICOBASE/doc` discuss various other features of the Framework.

5.2 Running a Sample Program

For a more hands-on introduction to the Pico Framework, we suggest that you try running one of our sample programs, located in

```
$PICOBASE/samples
```

One of our most often-used sample programs is StreamLoopback128. In this program, the software on the host computer sends the FPGA an increasing sequence of numbers via an input stream. The firmware on the FPGA sums these data and returns a running total in an output stream. You can test it out by typing:

```
cp -r $PICOBASE/samples/StreamLoopback128 ~/
cd ~/StreamLoopback128/software
make
./StreamLoopback $PICOBASE/samples/StreamLoopback128/firmware/M50*_******
    _StreamLoopback128.bit
```

where, in the last command, you should substitute the programming file appropriate to your FPGA. For example, on our M-505 FPGA, we used the bit file M505_LX325T_StreamLoopback128.bit.

The program's output should look like this:

```
File  Edit  View  Search  Terminal  Help
Loading FPGA with '../firmware/M505_LX325T_StreamLoopback128.bit' ...
Opening streams to test counter
8192 bytes of room in stream to firmware.
Wrote 4096 B
4096 B available to read from firmware.
Reading 512 B
Data received back from firmware:
0x42424242_deadbeef_42000000_42000000
0x42424242_deadbeef_84000004_42000004
0x42424242_deadbeef_c600000c_42000008
0x42424242_deadbeef_8000018_4200000c
0x42424242_deadbeef_4a000028_42000010
0x42424242_deadbeef_8c00003c_42000014
0x42424242_deadbeef_ce000054_42000018
0x42424242_deadbeef_10000070_4200001c
0x42424242_deadbeef_52000090_42000020
0x42424242_deadbeef_940000b4_42000024
0x42424242_deadbeef_d60000dc_42000028
0x42424242_deadbeef_18000108_4200002c
0x42424242_deadbeef_5a000138_42000030
0x42424242_deadbeef_9c00016c_42000034
0x42424242_deadbeef_de0001a4_42000038
0x42424242_deadbeef_200001e0_4200003c
0x42424242_deadbeef_62000220_42000040
0x42424242_deadbeef_a4000264_42000044
0x42424242_deadbeef_e60002ac_42000048
0x42424242_deadbeef_280002f8_4200004c
0x42424242_deadbeef_6a000348_42000050
0x42424242_deadbeef_ac00039c_42000054
0x42424242_deadbeef_ee0003f4_42000058
0x42424242_deadbeef_30000450_4200005c
0x42424242_deadbeef_720004b0_42000060
0x42424242_deadbeef_b4000514_42000064
0x42424242_deadbeef_f600057c_42000068
0x42424242_deadbeef_380005e8_4200006c
0x42424242_deadbeef_7a000658_42000070
0x42424242_deadbeef_bc0006cc_42000074
0x42424242_deadbeef_fe000744_42000078
0x42424242_deadbeef_400007c0_4200007c
All tests successful!
thomas@kunk-pico:~/StreamLoopback128/software$
```

As you can see, this sample is self-checking and reports a simple success message upon completion if everything went according to plan. More details about the StreamLoopback128 sample can be found in the StreamLoopback128 documentation. Likewise, several other fully documented samples are also available in individual subdirectories of `$PICOBASE/samples`.

6 Creating and Building Your Own Application

You are now ready to begin creating your own projects. A complete project using the Pico Framework requires both firmware for the FPGAs and software for the host computer. This section describes how to create the firmware for a new project. Several of the sections below will be divided into two halves, one using Xilinx tools and the other using Altera tools. In most cases, however, the procedures described will be very similar. *(Note: The sample projects were written and tested with Xilinx Vivado version 2014.2 or Altera Quartus II version 13.1 in mind. Newer software versions should work, but older ones may not.)*

To get started, copy the sample folder from `$PICOBASE/samples` that best fits your application. The sample projects provide an easy starting point from which to write your own firmware. In this

guide, we will show you how to open and build one of the more commonly used sample projects, the StreamLoopback128 example that we introduced above. We recommend that you familiarize yourself with that sample before continuing on to the next section. Full documentation is available in a PDF file located in `$PICOBASE/samples/StreamLoopback128`.

6.1 Copying a Sample Project Directory

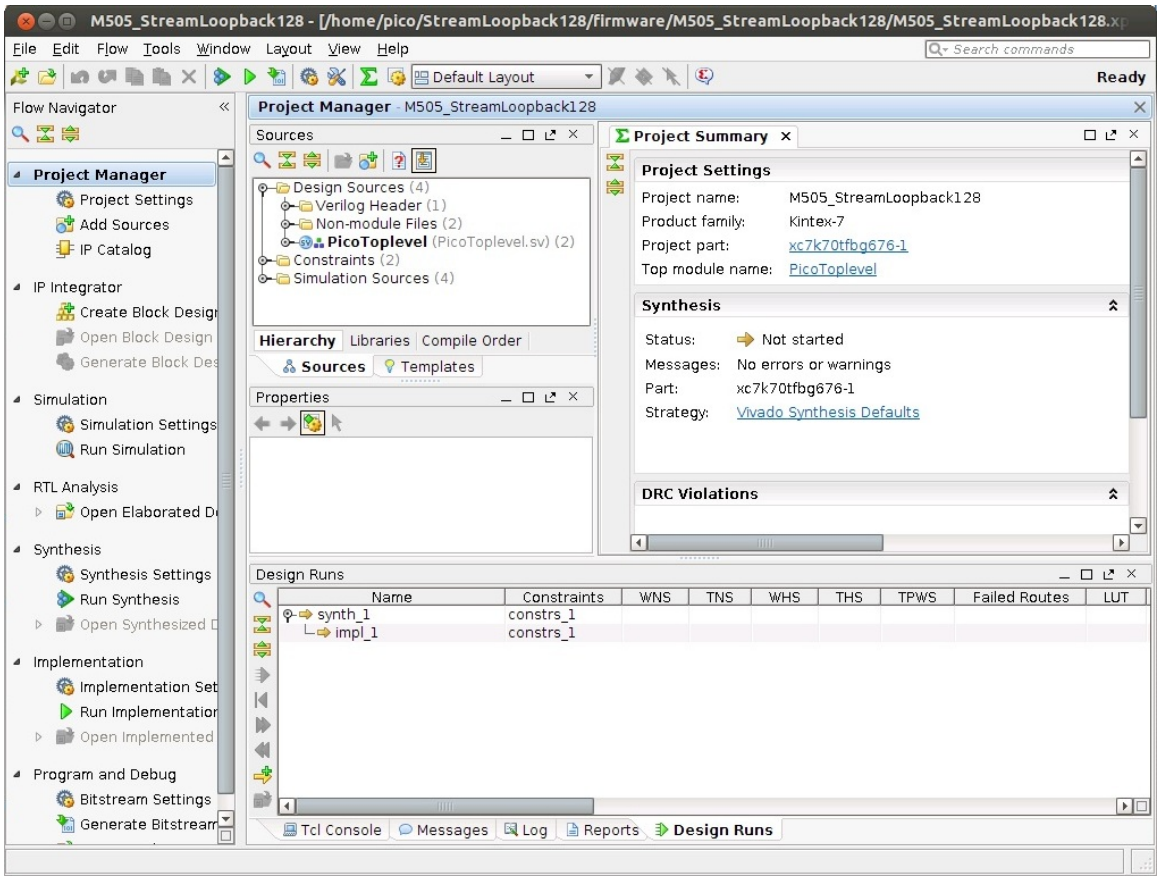
Start by copying the samples/StreamLoopback128 directory to your home directory, where we'll work on it. For example, run:

```
cp -r $PICOBASE/samples/StreamLoopback128 ~/
```

6.2 Opening the Project

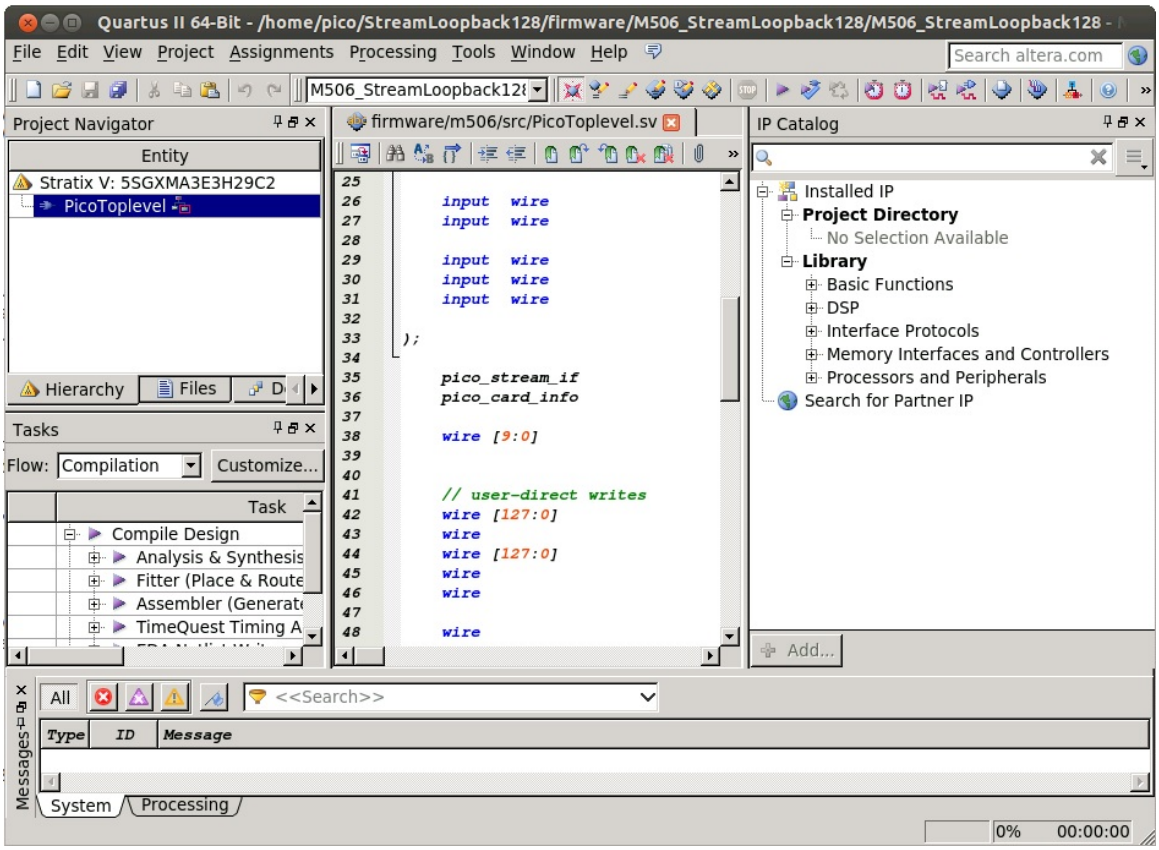
6.2.1 Xilinx

Launch Vivado. Click "Open Project". The Vivado project file will be located in `StreamLoopback128/firmware` in a subdirectory appropriate to your FPGA (e.g. open the file `~/StreamLoopback128/firmware/M505_StreamLoopback128/M505_StreamLoopback128.xpr` for an M-505). In this subdirectory, double-click on the .xpr file. When the project is open, your screen should look like this:



6.2.2 Altera

Launch Quartus II and click "File->Open Project". The Quartus project file will be located in `StreamLoopback128/firmware` in a subdirectory appropriate to your FPGA (e.g. M506_StreamLoopback128 for an M-506). Once you're there, double-click on the .qpf file. When the project is open, your screen should look like this:



6.3 A Brief Overview of the Pico Architecture

In the small *Sources* pane at the top of the Vivado window, you can see the design sources, constraints, and simulation sources for the project. Under *Design Sources*, *Verilog Header* contains the header PicoDefines.v that names the project and defines stream widths, information about the FPGA, and other important details as needed for a given project. The top-level module of the Pico architecture is called *PicoToplevel*. This top-level module instantiates the modules required for every project, the most important of which are the PicoFramework and your HDL modules. You should only modify your own UserModule and files below it in this hierarchy. The only exception is the include file PicoDefines.v, which you should change as mentioned below so that your project files can be included and built.

The rest of the hierarchy is made up of a variety of modules and other files that generally operate behind the scenes. The core logic used for PCIe communication with the host CPU, including a DMA engine, is located in PicoFramework. The UserWrapper instantiates logic for connecting the DMA engine to the streaming and PicoBus interfaces in the UserModule. Other modules found under PicoModules handle streams, the PicoBus, and memory access. Constraint files located under

Constraints convey pin placement and timing information to the Vivado synthesis tool, which is used during the mapping and the place and route phases of generating a bitfile.

6.3.1 Creating Your Own UserModule

For now, we won't edit anything in the sample. When you are ready to create a new design, you should create a new UserModule file (or copy one from a sample) and edit the USER_MODULE_NAME definition in PicoDefines.v (once again, this is located under *Verilog Header*) to match the name of your UserModule. Do not modify any of the files above the UserModule in the hierarchy other than PicoDefines.v.

6.4 Altera

6.4.1 Copying and Opening a Sample Project

1. Start by copying the samples/StreamLoopback128 directory to your home directory, where we'll work on it. For example, run:

```
cp -r $PICOBASE/samples/StreamLoopback128 ~/
```

6.4.2 A Brief Overview of the Pico Architecture

In the small *Project Navigator* pane in the top left corner of the Quartus window, there should be a project hierarchy with one top-level module, "PicoToplevel". This module instantiates the modules required for every project. Specifically, it creates the basic PicoFramework firmware module and your module, which in turn instantiates memory, stream, and user firmware among other things.

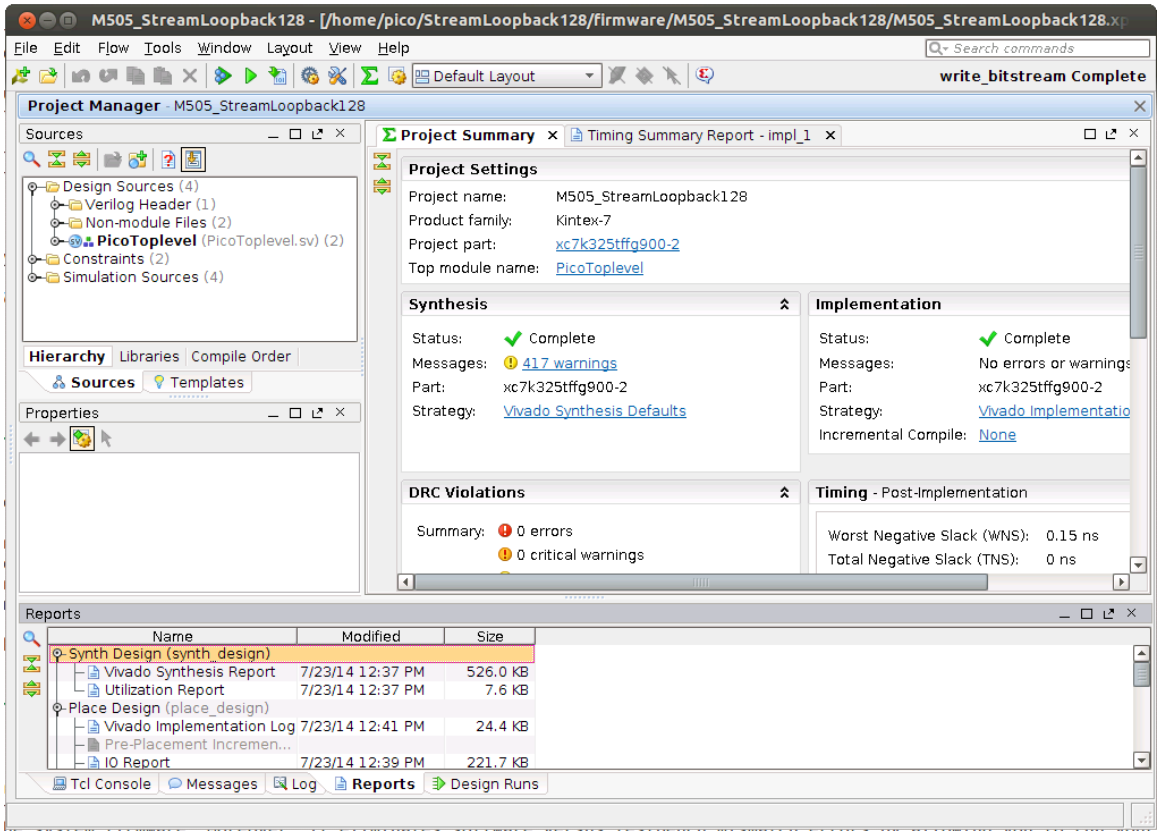
6.4.3 Creating Your Own UserModule

For now, we won't edit anything in the sample. When you are ready to create a new design, you should create a new user module file (or copy one from a sample) and edit the USER_MODULE_NAME definition in PicoDefines.v (probably near the bottom of the list in the *Files* tab) to match the name of your new module. Do not modify any other files in the project.

7 Building the Project

7.1 Xilinx

Once you are finished writing the initial version of your design, it's time to build it and test it on a real system. To build the project, click *Generate Bitstream* in the *Flow Navigator* pane on the left side of the Vivado window. The build process is computationally intensive and may take anywhere from 5 to 10 minutes for our simple samples to several hours for a more complex project, depending on the speed of your computer.

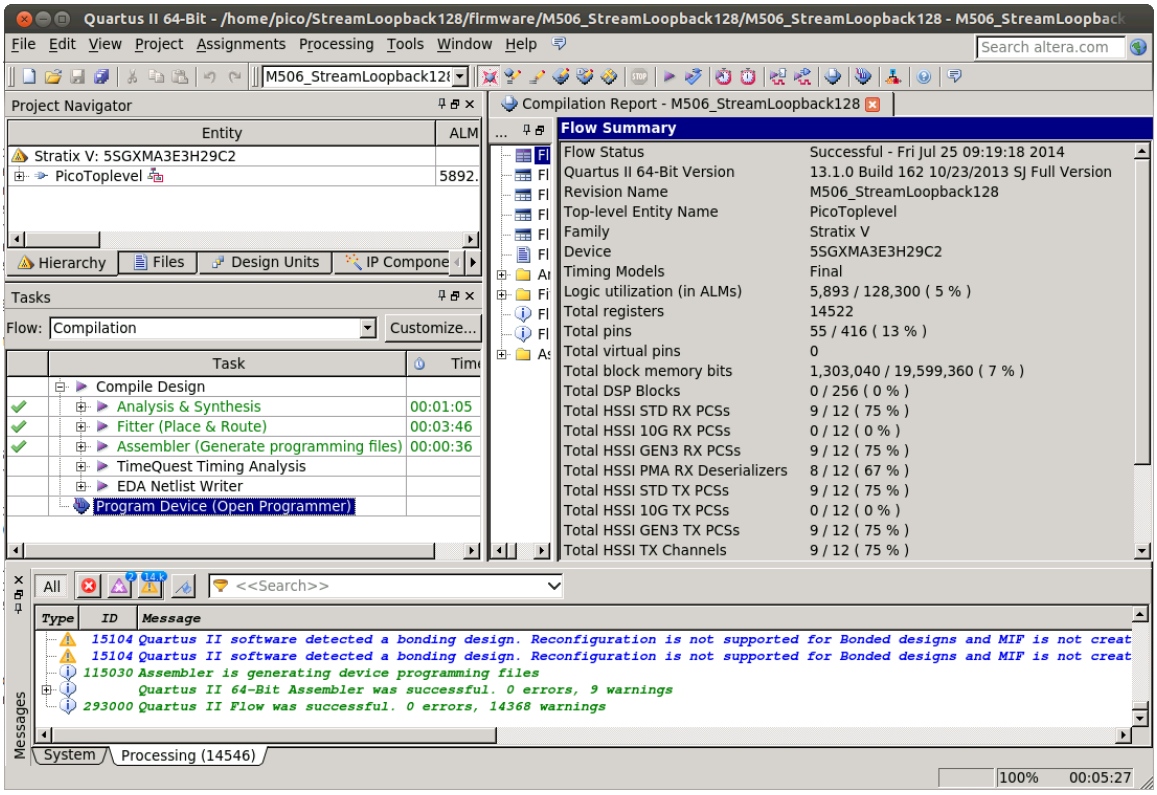


When the process is finished, it's a good idea to check and make sure the project meets timing by checking the numbers reported under Timing, just outside the screenshot area. "Number of Failing Endpoints" and "Total Negative Slack" should both be zero. (For ISE developers, Vivado's "Number of Failing Endpoints" is similar to ISE's timing score.) Also worthy of note is the fact that this entire sample project used just 3% of the flip-flops and 3% of the LUTs in the FPGA, and most of that was spent on the Pico Framework. (You can see this information by scrolling down a bit farther in the Project Summary.) This means that the FPGA is wide open for you to use in other parallel applications should you desire to do so.

You should now run through the StreamLoopback128 software again, this time using the new bit file instead of the precompiled one provided in the distribution. This checks that your Vivado installation is operating correctly.

7.2 Altera

Once you are finished with your design, it's time to build it and test it on a real system. To build the project, double-click *Assembler (Generate Programming Files)* in the *Tasks* pane on the left side of the Quartus window. The build process is computationally intensive and may take anywhere from 5 to 10 minutes for our simple samples to several hours for a more complex project, depending on the speed of your computer.



You should now run through the StreamLoopback128 software again, this time using the new rbf file instead of the precompiled one provided in the distribution. This checks that your Vivado installation is operating correctly.

8 Other Sample Projects

In order to take advantage of other features of the Pico Framework, you may wish to start with another sample project rather than StreamLoopback128. Each sample has full documentation, explaining how it works and what it does, included in the Pico Framework distribution.

- StreamLoopback128 demonstrates several of the most useful features of the Framework, including streams and DMA transactions with the host.
- The PicoBus128_HelloWorld sample illustrates the PicoBus, the other possible communication scheme (besides streams) allowed by the Pico architecture. For more information on the PicoBus, see the Pico API documentation.
- The DDR3_MovingAverage sample shows how to interact with the DDR3 off-chip memory on an M-series module. More information on memory systems can be found in the PicoMemory documentation file.
- The MultiStream example shows how to do computation using multiple FPGAs by programming them to communicate with streams.

- Two of the programs (HelloWorld and StreamLoopback) have both 32- and 128-bit versions in order to demonstrate both sizes of streams and the PicoBus. If you wish to use 32-bit streams in your design, it is easier to start with the 32-bit version of these projects.

9 Troubleshooting

9.1 General Suggestions

- Be sure to verify that your system meets all system requirements.
- Is your system fully turned on? Check to see that the green and orange LEDs on your modules are turned on. If they aren't, your backplane(s) may not be plugged in to a PCIe power cable.
- Power cycling the system by shutting it down completely, waiting a few seconds, and then restarting may help.

9.2 Module not shown in purty

If you have just finished installing the Pico software, you may simply need to power cycle your system. Try shutting it down and restarting.

If the Pico software is fully installed and your module still does not show up in the monitor program, open a command prompt and type (case sensitive):

```
sudo lspci -nn | grep Pico
```

If you don't see anything, then the system does not see the module at all. There is nothing that can be done in software. If your board is a module on the EX-500, such as the M-503 or M-505, please check the LEDs to ensure the board is getting power. See the hardware manual for LED locations.

If, however, the lspci program does report a Pico module, the problem is likely with the driver. To see if the driver is loaded, run (case sensitive):

```
lsmod | grep pico
```

If you see no output, the driver is not loaded. Try loading the driver by running the following as root:

```
depmod; modprobe -v pico
```

Now use dmesg to look at the system log to see if the driver has reported any errors:

```
dmesg
```

If the modprobe command didn't fix the problem, please run these commands:

```
dmesg > dmesg.log
sudo lspci -nn -vvvvv > lspci.log
```

and contact Pico with the resulting log files.

9.3 Unable to compile programs

Check to make sure the PICOBASE environment variable is pointing to the latest Pico installation. This is set when the Pico package is installed. You should see something like this:

```
echo $PICOBASE
[output] /usr/src/picocomputing-5.2.0.0
```

If you don't see PICOBASE pointing to a valid directory, you can set it with:

```
export PICOBASE=/usr/src/picocomputing-5.2.0.0
```

or a similar command appropriate to the version of the Pico software you have installed. Now rebuild the program with the the PICOBASE variable set correctly.

9.4 Software-generated Errors

For a detailed list of error codes produced by the Pico software, please consult the Pico API error code appendix.

10 Support

For further support using Pico products, both firmware and software, please see our customer service website at `https://picocomputing.zendesk.com/home`.