FrontPanel over IP

> ⚠ **ALPHA Release**
>
> FrontPanel Over IP (FPoIP) is currently an **ALPHA** feature and specification. It is not recommended for production use. Please keep an eye on our Newsletter for more information!

Introduction

FrontPanel versions 5.0.0 and newer support FrontPanel over IP (FPoIP). FPoIP enables a remotely connected client to establish a connection with a FrontPanel host (the "FPoIP server") and control FrontPanel devices connected to the host. The FrontPanel SDK handles all aspects of the protocol with relatively few application changes required to support this functionality.

Motivation

FrontPanel can manage and communicate with devices connected locally via USB and PCI Express. It is sometimes desirable to extend the reach of that connection over a longer distance such as a local area network (LAN) or wide area network (WAN) via the internet. This can allow a much wider range of clients to communicate with the device without a direct connection to the device.

From an application perspective, FPoIP enables very low-cost computing platforms such as Raspberry Pi or BeagleBone devices to put FrontPanel devices on the internet. The FPoIP server would run on the Linux machine and any authenticated client could communicate with the FrontPanel device (s) attached to it's USB port.

Features

FPoIP supports the following features:

- Enumeration of available remote devices
- Establishing an exclusive connection with one or more remote devices on one or more remote servers
- FPGA configuration
- FPGA communication via wires, triggers, and pipes
- Executing low-latency server-side scripts

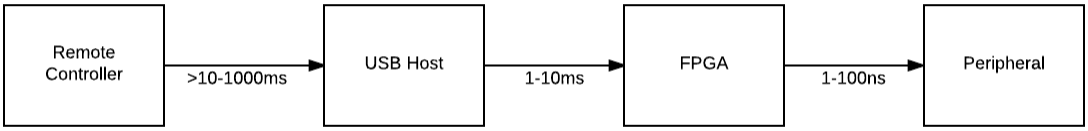
Known Issues and Limitations

- The Device Settings API is not presently supported over FPoIP. This is planned for the BETA release.
- The Device Sensors API is not presently supported over FPoIP. This is planned for the BETA release.
- ResetProfiles are not available for remote devices. This is planned for the BETA release.
- The current internet timeout for FPoIP is fixed at 30 seconds. This will be programmable in a future release.

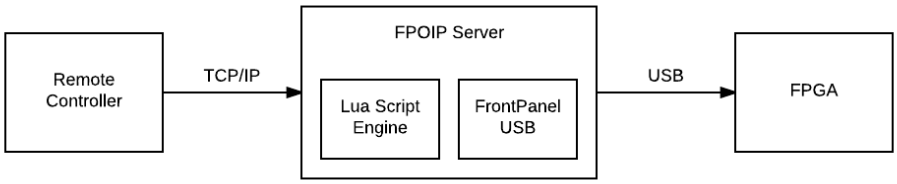
Server Side Scripting

One way to view a digital system is from the perspective of communication latency between a device and its controller. An FPGA commonly communicates at the nanosecond range in latency. In systems where Opal Kelly integration modules are often deployed, a System Host is added and communicates with the FPGA at the millisecond range. This system host may be required for any number of reasons including additional storage, post processing power, and user interface capability. Low-latency roles are still partitioned to the FPGA while slightly higher latency roles can be moved to the System Controller. Indeed, some register-transfer operations can take place between the System Host and the FPGA and still meet performance requirements. Examples include polling a state machine for completion or extracting a small amount of data, processing it, and transferring it back to the FPGA.

Some systems may require even greater distance from the FPGA and peripherals. In these cases, we refer to a Remote Controller that communicates with the System Host over media such as wired or wireless ethernet. At this greater distance, latencies can increase by orders of magnitude and connection reliability becomes a greater consideration. The increased latency between the Remote Controller and the FPGA makes register-transfer operations rather impractical. FrontPanel wires, triggers, and pipes were designed as register transfer (atomic) operations best suited to the relatively low latency environment of USB. Usage at higher latencies renders many applications impractical or completely unusable.



To solve this issue while still maintaining the convenience and ease of use of FrontPanel, we introduce Server Side Scripting (SSS). A SSS is a program written in the Lua scripting language that is transferred from the Remote Controller to the USB Host. It then runs on the USB Host with the benefits of the low-latency FPGA communication. Results of the script are transferred back to the Remote Controller upon completion.



Examples

Image Capture

Opal Kelly's Image Sensor Evaluation Boards are a great way to evaluate several Opal Kelly modules. The optional EVB100X Developer's Release includes full source code to the software camera application and the FPGA IP that communicates with the image sensor, buffers the images, and conveys the captured images to the PC over the FrontPanel SDK. This is a real application that benefits greatly from the tight coupling allowed by Server Side Scripting. Communication latencies and buffer polling latencies are greatly reduced.

The latest version of the EVB100X Developer's Release includes an updated software suite supporting FPoIP Server Side Scripting.

I2C Controller

TBD

Opal Kelly's open-source I2C Controller IP is a full-featured I2C controller that allows a software application to communicate with I2C devices attached to pins of the FPGA. Originally designed for the tightly-coupled USB FrontPanel connection, performance suffers over long-latency links. This controller will be updated shortly with Server Side Scripting to improve performance.

Lua Scripting Language

From the Lua About page:

```
Lua is a powerful, efficient, lightweight, embeddable scripting language. It supports procedural
programming, object-oriented programming, functional programming, data-driven programming, and
data description.

Lua combines simple procedural syntax with powerful data description constructs based on associative
arrays and extensible semantics. Lua is dynamically typed, runs by interpreting bytecode with a
register-based virtual machine, and has automatic memory management with incremental garbage
collection, making it ideal for configuration, scripting, and rapid prototyping.
```

These characteristics make Lua very attractive to platform designers looking to integrate a scripting language into their platform. It is used extensively in many games as well as many applications to extend functionality such as Adobe Photoshop Lightroom.

The Lua syntax is clean and simple and should not be difficult to follow even for those unfamiliar with the language. We provide some examples with our server side scripting examples and there is a lot of information available online. If you're interested in learning more about Lua, please check out the Lua website.

Security

> ⚠ **ALPHA Release**
>
> In the ALPHA release, only password client authentication is supported. Additional authentication methods will be supported in a future release.

Encryption

TLS encryption is supported and, in fact, required.

Client Authentication

Client authentication enables the FPoIP server to grant access to only allowed clients. FPoIP supports two types of client authentication:

- Password

- Client Certificates

Password Authentication

With **password authentication**, the server maintains a list of usernames and passwords. A client must provide credentials with the initial connection request in order to be granted access. The passwords are protected by the well-respected bcrypt hashing function and a random salt is used. The command-line application `fpoip-passwd` is used to manage users and passwords for the server.

Client Certificates

With **client certificates**, the server is given one or more public keys. A client must have a corresponding private key in order to be granted access.

Note: Client certificates are not presently supported.

Server Authentication

Server authentication is used to verify that the server is who the client expects it to be.

Performance

> ⚠ **ALPHA Release**
>
> Please note that there are still improvements to be made in performance as we are still in the ALPHA phase.

Generally, you should expect lower performance with FPoIP when comparing it to a USB connection. Internet and intranet connections are subject to many more factors that affect performance such as latency, intermediate devices, dropped frames, and reduced bandwidth. It is impossible to quantify these performance differences for all conditions. A direct connection between two fast computers with Gigabit ethernet to a switch will most likely perform better than two computers communicating over WiFi, several internet hops, and transatlantic cable.

Windows 8.1 client:

Server	1 MiB Write	4 MiB Write	16 MiB Write	1 MiB Read	4 MiB Read	16 MiB Read	iperf (control)
Windows 8.1 (localhost)	79 MiB/s	125 MiB/s	125 MiB/s	119 MiB/s	126 MiB/s	133 MiB/s	4.4 Gbit/s
Windows 10 (gigabit)	15 MiB/s	32 MiB/s	32 MiB/s	48 MiB/s	52MiB/s	64 MiB/s	946 Mbit/s
MacMini (gigabit)	13 MiB/s	32 MiB/s	34 MiB/s	22 MiB/s	53 MiB/s	67 MiB/s	949 Mbits/s
nVidia Tegra K1 (gigabit)	7 MiB/s	12 MiB/s	13 MiB/s	10 MiB/s	16 MiB/s	22 MiB/s	942 Mbit/s
Raspberry Pi 3 (10/100)	5 MiB/s	7 MiB/s	7 MiB/s	7 MiB/s	7 MiB/s	8 MiB/s	95 Mbits/s

For consistent results all performance tests were completed using XEM6310-LX45 boards. All tests were performed using the PipeTest sample on an iMac client running Windows 8.1 through bootcamp. Measurements using the iPerf tool have been provided as a baseline to give an idea of the ideal network performance between the client and server machines.

MacOS client (Mac Mini):

Server	1 MiB Write	4 MiB Write	16 MiB Write	1 MiB Read	4 MiB Read	16 MiB Read	iperf (control)
MacOS (localhost)	85 MiB/s	92 MiB/s	92 MiB/s	96 MiB/s	98 MiB/s	100 MiB/s	20.1 Gbit/s
Windows 10 (gigabit)	55 MiB/s	65 MiB/s	67 MiB/s	40 MiB/s	40 MiB/s	40 MiB/s	903 Mbit/s
nVidia Tegra K1 (gigabit)	13 MiB/s	16 MiB/s	20 MiB/s	18 MiB/s	24 MiB/s	26 MiB/s	932 Mbit/s
Raspberry Pi 3 (10/100)	7 MiB/s	8 MiB/s	8 MiB/s	7 MiB/s	8 MiB/s	8 MiB/s	95 Mbit/s

For consistent results all performance tests were completed using XEM6310-LX45 boards. All tests were performed using the PipeTest sample on an MacMini client running MacOS 10.12 through bootcamp. Measurements using the iPerf tool have been provided as a baseline to give an idea of the ideal network performance between the client and server machines.

Migrating Legacy Software to use FPoIP

Some modification to software written for FrontPanel versions prior to 5.0.0 is necessary to use the new functionality provided by FPoIP. Use of FPoIP functionality requires opening devices through the new `OpalKelly::FrontPanelDevices` or `FrontPanelManager` classes. Older methods to open devices will still work but are deprecated and only allow use of local (USB connected) devices.

The new `FrontPanelDevices` class allows the user to connect to both local and remote devices, defined by the FPoIP realm setting. This setting defaults to local devices, though it can be overwritten by setting the "`okFP_REALM`" environment variable prior to software execution. Further, the realm can be set definitively by passing a specific realm URI as a string to the `FrontPanelDevices` constructor when instantiating the object.

A typical example of opening a device for use with FrontPanel prior to 5.0.0 is as follows:

```
okCFrontPanel* dev = new okCFrontPanel;
if (dev->OpenBySerial() != okCFrontPanel::NoError) {
    ... handle the error ...
    delete dev;
    return;
}
... use the device, e.g. call dev->ConfigureFPGA() ...
delete dev;
```

The minimal change necessary to the above code to take advantage of new FPoIP features is:

```
OpalKelly::FrontPanelDevices devices;
OpalKelly::FrontPanelPtr dev = devices.Open();
if (!dev) {
    ... handle the error ...
    return;
}
... use the device, e.g. call dev->ConfigureFPGA() ...
```

Note that there is no need to explicitly call "delete" on `dev` using the updated API. This is because the `FrontPanelPtr` object is an C++ `auto_ptr` and will automatically be destroyed when it goes out of scope. As before, a serial number can be passed to the `devices.Open()` function to open a specific device.

Using the method above for opening a device will default to use of local devices directly attached to a computer via USB. This can be changed by setting the "`okFP_REALM`" environment variable to a valid FPoIP URI pointing to a running FPoIP server.

```
# Windows
set okFP_REALM=fpoip://user:password@host:port

# Mac / Linux
export okFP_REALM=fpoip://user:password@host:port
```

See the FPoIP server documentation for more information on configuring and running the FPoIP server.

Windows, Linux, and Mac applications built with FPoIP support must provide `okimpl_fpoip.dll, okimpl_fpoip.so, and okimpl_fpoip.bundle` respectively. These files can be found in the appropriate `FrontPanelUSB/API/lib` folder for a given architecture. These DLL files are in addition to the `okFrontPanel.dll` file already required for FrontPanel applications.

FPoIP Server

The FPoIP server handles communication between a FrontPanel USB device connected to the server machine and a networked client machine. The server is currently distributed as a command line application.

Quick Start

In order for an FPoIP client to access remote devices, the FPoIP server must be running on the host computer where the devices are attached. The FPoIP server requires a TLS certificate and a TLS key file to enable TLS encryption for the link. There is no option to run FPoIP without encryption. The server also requires a password file to authenticate the client.

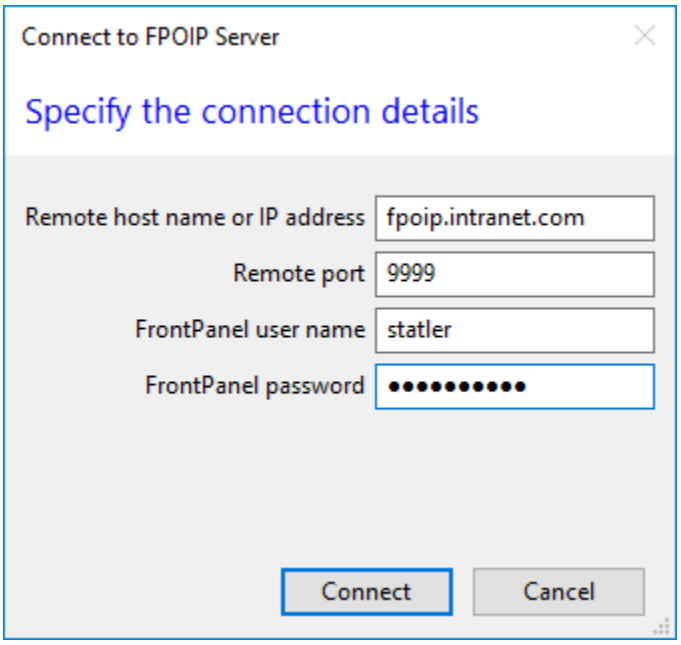
We strongly encourage users to use their own certificate and key file. However, in the interest of getting up and running quickly, we have provided example files in the `certs` folder. An example password file is also provided with a default user "User" with password "Test".

On the FPoIP server, run the following in a command prompt:

```
$ ./fpoip-server --tlscert=FPoIP/certs/snakeoil.crt --tlskey=FPoIP/certs/snakeoil.key --password=FPoIP
/example.passwd
```

Run the Client

In the FrontPanel application, click on **FrontPanel > Connect to remote server...**



For command line applications, the environment variable `okFP_REALM` can be set to tell the application how to find the FPoIP server.

```
# Mac and Linux (Bash)
export okFP_REALM=fpoip://User:Test@10.10.1.5:9999
pipetest ...command line arguments here...

# Windows
set okFP_REALM=fpoip://User:Test@10.10.1.5:9999
pipetest.exe ...command line arguments here...
```

Security and Authentication

In the QuickStart, we used a self-signed TLS certificate and TLS key that are included with FrontPanel. These will allow you to use TLS encryption, but they will not provide any identity verification. For more advanced security, you will want to generate your own certificate or purchase one from an existing **certificate authority** (CA).

Certificate	Certificate Authority	Encryption?	Server ID	Notes
`snakeoil.crt`	none	Yes	No	This is the QuickStart example.
Self-signed	none	Yes	No	
Self-signed	Local	Yes	Yes	Running a local CA means you can control server identification.
Commercial	Commercial	Yes	Yes	Best security, but commercial certificates are not free.

Specifying a Certificate Authority

You can specify a CA in your environment with the `FPOIP_CA` environment variable. For example,

ENVAR	Value
`FPOIP_CA`	`/etc/certs/ca/certs/ca.cert.pem ./TestClient/TestClient`

Managing Passwords

FPoIP Password Files are managed using the provided `fpoip-passwd` utility. The password file consists of username/hashed password pairs separated by a colon (:), each pair is placed on a separate line. A sample password file (`example.passwd`) is provided and includes an entry for a user with username "User" and password "Test".

Some examples are shown below.

```
# Create a new password file
# fpoip-passwd -c <filename> <username> <password>
fpoip-passwd -c example.passwd User Test

# Add a new user to an existing password file
fpoip-passwd example.passwd NewUser Test

# Remove a user's entry from a password file
# fpoip-passwd -D <filename> <username>
fpoip-passwd -D example.passwd DeleteMe
```

Creating a Certificate

A new SSL certificate must be created by generating a key and then self-signing it, signing it with a local certificate authority (CA), or by going through a commercial certificate signing process.

The following commands will generate a self-signed certificate. Note that while this certificate can be used to encrypt FPoIP communication, it will not provide any server identity verification.

```
# Generate and self-sign the key
openssl req -newkey rsa:2048 -nodes -keyout snakeoil.key \
    -x509 -days 3655 -out snakeoil.crt \
    -subj "/C=US/ST=CA/L=Newport Beach/O=Bluth Company/CN=server.bluth.com"
```

A CA-signed certificate will provide both encryption and identity verification. To obtain a CA-signed certificate, the following commands can be used to generate a new key and a certificate signing request.

```
# Generate a new key file
openssl genrsa -aes256 -out server.key 2048
# Enter a password to protect this key when prompted

# Create a certificate signature request for the new key
openssl req -key server.key -new -sha256 -out server.csr \
    -subj "/C=US/ST=CA/L=Newport Beach/O=Bluth Company/CN=server.bluth.com"
```

The certificate request file (`server.csr`) can then be used to generate a signed SSL certificate through either a local or commercial CA.

When using a commercial CA, follow their instructions for signing the request.

Information on creating a local CA can be found in the OpenSSL documentation or in this tutorial by Jamie Nguyen. A signature with a local CA can be accomplished with the command below.

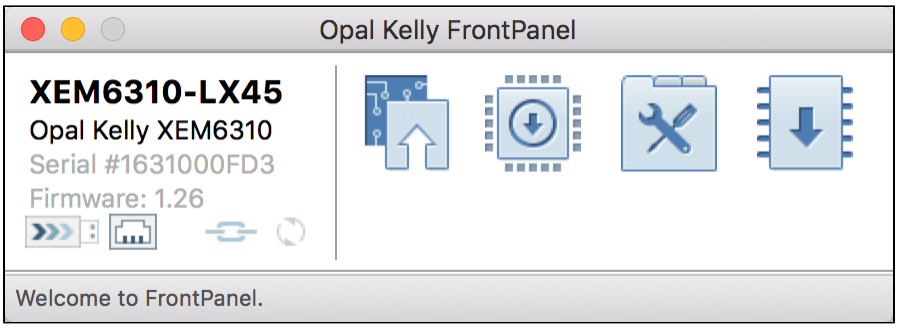
```
# Sign a csr with a local CA
openssl ca -config ca/openssl.conf -extensions server_cert \
    -days 3655 -notext -md sha256 -in server.csr -out server.crt
# Confirm the signature when prompted
```

When complete, the certificate request file (`server.csr`) can be deleted. The server.key and server.crt files will be used by the FPoIP server.

FrontPanel Application

The FrontPanel Application supports connections to a remote FPoIP server. To connect to a remote server, click on **FrontPanel > Connect to remote server...**

Remote devices are indicated in the corresponding device panel by the ethernet plug icon next to the USB icon.



Known Issues / Limitations

- FrontPanel typically polls the device intermittently to determine if FrontPanel is still enabled. This can detect "de-configuration" conditions if the device is, for example, reset via JTAG. This polling is disabled for FPoIP remote devices.
- Device Sensors are not available for remote devices. This is planned for the BETA release.
- Device Settings are not available for remote devices. This is planned for the BETA release.
- ResetProfiles are not available for remote devices. This is planned for the BETA release.