FPGALink Documentation

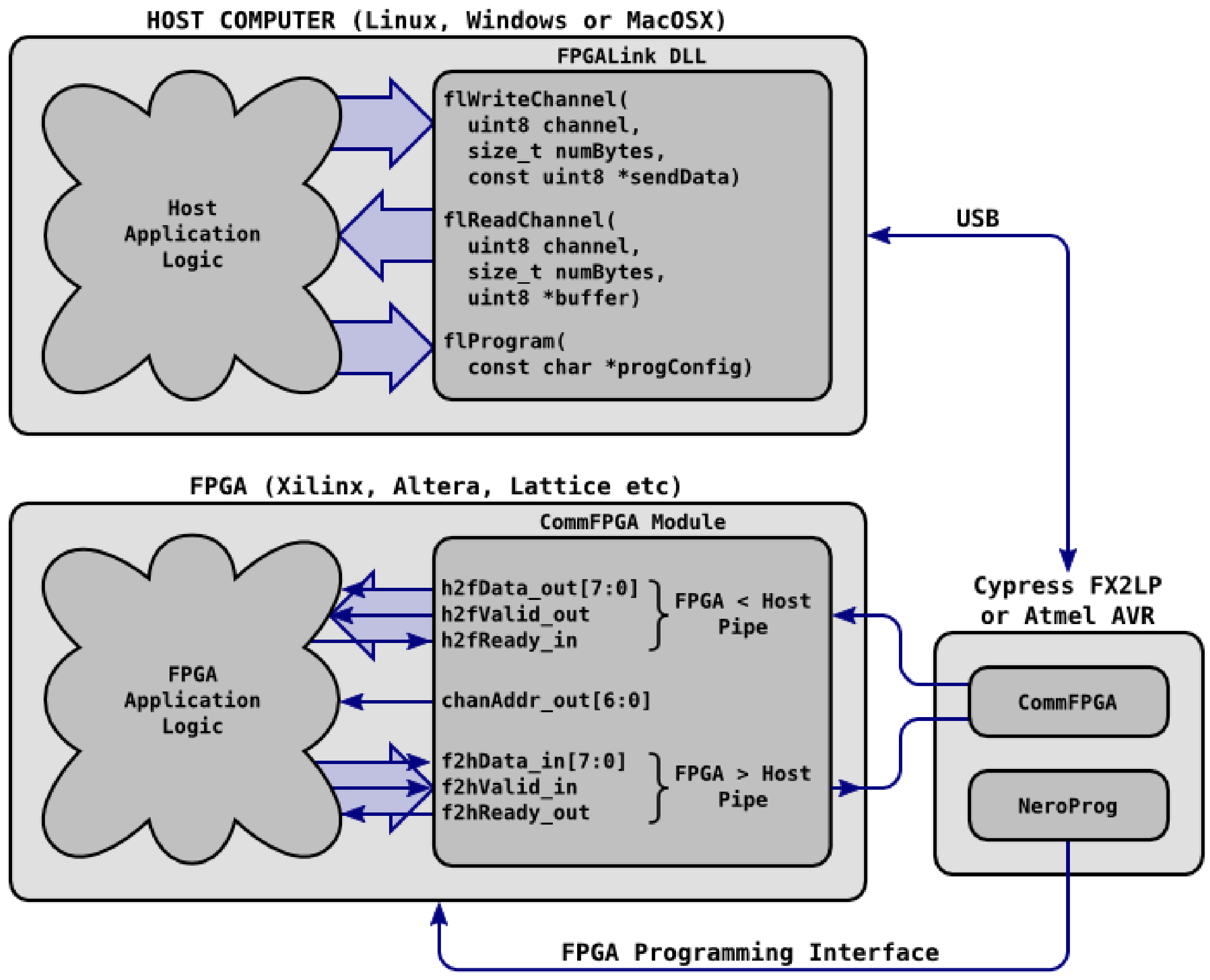
Release 20140311

Chris McClelland

March 13, 2014

Contents

The aim of the FPGALink project is to provide a hardware abstraction layer for hardware involving an FPGA connected to a computer over USB, to abstract core functionality like FPGA-programming and subsequent host-FPGA communication. The result is that you only have to learn how to use one set of tools, APIs and HDL interfaces because that knowledge is transferrable to many different host platforms and many different commercial FPGA development kits.


HOST COMPUTER (Linux, Windows or MacOSX)
FPGALink DLL
Host Application Logic
flWriteChannel(uint8 channel, size_t numBytes, const uint8 *sendData)
flReadChannel(uint8 channel, size_t numBytes, uint8 *buffer)
flProgram(const char *progConfig)
USB
FPGA (Xilinx, Altera, Lattice etc)
CommFPGA Module
FPGA Application Logic
h2fData_out[7:0]
h2fValid_out
h2fReady_in
FPGA < Host Pipe
chanAddr_out[6:0]
f2hData_in[7:0]
f2hValid_in
f2hReady_out
FPGA > Host Pipe
Cypress FX2LP or Atmel AVR
CommFPGA
NeroProg
FPGA Programming Interface

CHAPTER 1

Contents

1.1 INTRODUCTION

1.1.1 Justification

Development kits for Field Programmable Gate Arrays (FPGAs) are ubiquitous, with offerings from a plethora of manufacturers, with prices ranging from the tens of dollars to well into the thousands, and featured FPGAs ranging from a few thousand logic cells to a few million. Whereas the high-end boards tend to be PCIx plug-in cards, the cheaper boards tend either to be designed around a USB interface chip (e.g Cypress FX2LP, Atmel AVR, Microchip PIC or FTDI chip), or lack direct host interfacing altogether, requiring a standalone JTAG cable for programming. Unfortunately, even for those boards designed around a USB interface, there is a general lack of good integrated solutions for exchanging arbitrary data between the host computer and the FPGA, once it has been programmed.

1.1.2 Overview

FPGALink is an end-to-end, open-source, cross-platform solution designed to do a few simple jobs, and do them well:

- Directly program the FPGA during development, using JTAG or one of the vendor-specific serial and parallel programming algorithms.
- Program onboard flash chips used by the FPGA to boot on power-on.
- Allow the host and/or microcontroller to exchange arbitrary binary data with the FPGA, using a variety of connection methods.

It provides a host-side API, firmware for several USB interface microcontrollers, and 128 addressable eight-bit read/write FIFOs on the FPGA side.

- On the host side there is a dynamic-link library with a straightforward API. The libraries and example application code will build on MacOSX (x64 & x86), Windows (x64 & x86) and Linux (x64, x86, ARM & PowerPC). Bindings are provided for C/C++, Python, Perl, Java and Excel/VBA; binding other languages is straightforward.
- For the USB interface there are firmwares for the Cypress FX2LP (used on most Digilent, Aessent, KNJN, ZTEX and Opal Kelly boards) and Atmel AVR (used by some AVNet, Digilent, Embedded Micro and Lattice boards). There's also an NXP LPC firmware which unfortunately lacks an active maintainer.
- The Cypress FX2LP firmware supports a synchronous FIFO interface with a sustained bandwidth of around 42MiB/s. The Atmel AVR firmware supports an asynchronous parallel interface (using the IEEE1284 EPP Protocol) and a synchronous serial interface, both of which have a sustained bandwidth of around 500KiB/s.
- On the FPGA side there are several simple interface modules with separate implementations in VHDL and Verilog. Each module gives the host a FIFO-style read/write interface into your FPGA design, supporting up to

128 separate logical "channels" with flow-control. A couple of fully-functional example designs are provided to get you started.

Everything is licensed under the GNU Lesser General Public Licence; you are therefore free to distribute unmodified copies of FPGALink with your products. The library has no commercial or hardware usage restrictions, so you can prototype your design with an inexpensive devkit, and then use the same software tools on your custom-built PCBs. In this way you can easily distribute updated FPGA designs to your customers just as you would with regular firmware updates, with no special programming cables required, making your FPGA truly "field-programmable".

1.1.3 Conventions

- 1MB = 1 megabyte = 10^6 bytes = 1,000,000 bytes.
- 1MiB = 1 mebibyte = 2^{20} bytes = 1,048,576 bytes.
- 1Mb = 1 megabit = 10^6 bits = 1,000,000 bits.
- 1Mib = 1 mebibit = 2^{20} bits = 1,048,576 bits.

1.1.4 How to get help

The only place you're *guaranteed* to get a response to FPGALink-related queries is the FPGALink Users Group.

1.1.5 Licences & Disclaimers

The FPGALink library, firmware & HDL code is licensed under the LGPLv3:

What does it mean to apply a free software licence like the LGPLv3 to FPGALink's VHDL and Verilog modules? Broadly, the intent is to enable you to use them *unmodified* in your own (perhaps commercial & proprietary) products without invoking the copyleft clause. However, if you *modify* any FPGALink VHDL or Verilog source file, the result is a derived work which must only be distributed under the LGPLv3 licence. In practice it's sufficient to email your modifications to the fpgalink-users mailing list. In summary, I will never ask you to reveal *your* intellectual property; I only ask that you reveal your improvements to *my* intellectual property.

The FLCLI utility is licensed under the GPLv3:

The Gordon flash tool is conceptually based on ideas from the flashrom project and is thus licensed under the GPLv2:

If you have specific licensing concerns, just ask on the mailing list. Don't worry, I won't bite!

1.2 INSTALLATION

1.2.1 Linux Installation

Prerequisites

You'll need a gcc compiler and the development packages **ABCDEFGHIJKLMNOPQRSTUVWXYZ** for **libusb** and **libreadline**. For example, on a Debian-derived Linux distribution (e.g Debian, Ubuntu & Mint), thus:

```
$ sudo apt-get install build-essential libreadline-dev libusb-1.0-0-dev python-yaml
```

If you want to use the Atmel AVR firmware, you'll also need the AVR toolchain:

```
$ sudo apt-get install gcc-avr avr-libc dfu-programmer
```

The Cypress FX2LP firmware is provided as a pre-built **.hex** file, so you don't *need* to build it from source, but if you want to do so, you'll need **sdcc**:

```
$ sudo apt-get install sdcc
```

For building the VHDL and/or Verilog examples for loading into your FPGA you'll need to install the FPGA vendor tools. For Xilinx FPGAs you'll need ISE WebPACK and for Altera FPGAs you'll need Quartus II Web Edition.

Note: Pay close attention to the supported device families before you download the software: Altera especially have a habit of dropping tooling support for devices fairly rapidly, so unless your FPGA is very recent you may need to download an older software release.

On Debian/testing amd64 I did this:

If you want to run the VHDL in simulation you'll need GHDL and GTKWave. You can get a recent GHDL from Joris van Rantwijk:

```
$ wget http://jorisvr.nl/debian/ghdl/ghdl_0.30~svn20130213-2_amd64.deb
$ sudo dpkg --install ghdl_0.30~svn20130213-2_amd64.deb
$ sudo apt-get -f install  # fix dependencies
```

And you can install GTKWave from the standard repository:

```
$ sudo apt-get install gtkwave
```

1.2.2 Blah

Blah blah blah.

Blah.

1.2.3 FooBar

Note: Blah note!

Foo bar!

Code:

```
<h1>code block example</h1>
```

Foo:

```
// 4567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
#include <stdio.h>

int main(void) {
    printf("Hello World\n");
}
```

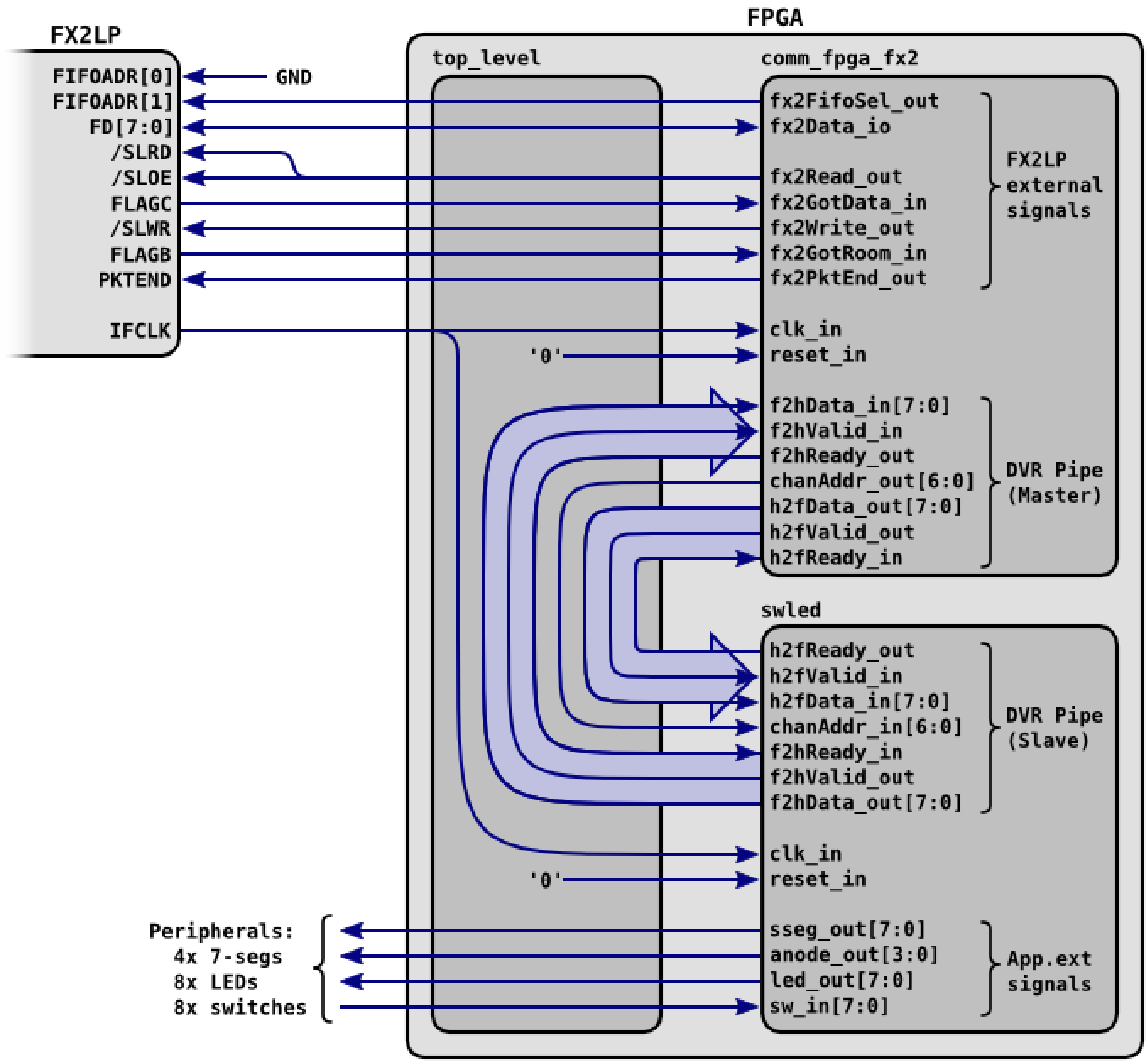
See?

```
package mem_ctrl_pkg is
    type MCCmdType is (
        MC_NOP,
        MC_RD,
        MC_WR,
        MC_REF
    );
    component mem_ctrl is
        generic (
            INIT_COUNT     : unsigned(12 downto 0);  -- cycles to wait during initialisation
            REFRESH_DELAY  : unsigned(12 downto 0);  -- gap between refresh cycles
            REFRESH_LENGTH : unsigned(12 downto 0)   -- length of a refresh cycle
        );
        port(
            clk_in         : in    std_logic;
            reset_in       : in    std_logic;

            -- Client interface
            mcAutoMode_in  : in    std_logic;
            mcCmd_in       : in    MCCmdType;
            mcAddr_in      : in    std_logic_vector(22 downto 0);
            mcData_in      : in    std_logic_vector(15 downto 0);
            mcData_out     : out   std_logic_vector(15 downto 0);
            mcRDV_out       : out   std_logic;
            mcReady_out     : out   std_logic;

            -- SDRAM interface
            ramCmd_out      : out   std_logic_vector(2 downto 0);
            ramBank_out     : out   std_logic_vector(1 downto 0);
            ramAddr_out     : out   std_logic_vector(11 downto 0);
            ramData_io      : inout std_logic_vector(15 downto 0);
            ramLDQM_out     : out   std_logic;
            ramUDQM_out     : out   std_logic
        );
    end component;
end package;
```

See?



And that's all.