Tutorial FPGALink

Revisión: *0.01* *revisión inicial*
0.02 *Cambios en aplicación del loop. Se agrega Lado FPGA y Lado PC para programar la FPGA con un loop adecuado. Correciones menores*

Autor: Marcos A. Cervetto, Edgardo Marchi[1]

Introducción

En este tutorial, se montará el sistema de comunicación USB basado en FPGALink[2] sobre el kit de desarrollo de nuestra querida Digilent **Nexys™2 Spartan-3E FPGA Board**[3]. Este kit consta con una FPGA Xilinx Spartan-3E FPGA 500K ó 1200K. Si bien el tutorial hará referencia a esta plataforma de hardware en particular, es aplicable con cambios mínimos a otros sistemas de similares características. También habrá notas sobre como se debe cambiar los comandos para que se puedan aplicar a la placa **Atlys Spartan-6 LX45 FPGA**[4]

FPGA Atlys = **xc6slx45-csg324-3**
FPGA Nexys2-500 =

¿Qué se necesita?

- Una conexión a internet (o haber descargado previamente las bibliotecas de FPGALink)
- Un kit Nexys 2 o similar
- Preferentemente alguna distribución de linux (no tendría que haber mayores incovenientes compilando la Librería en Windows pero el tutorial está orientado a una plataforma linux)

¿Qué es FPGALink?

Como el autor lo define en su pagína:

"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"

FPGALink provee 3 componentes de software:

- Un firmware para el microcontrolador USB ubicado en el kit
- Un firmware en HDL que permite la comunicación micro <-> FPGA
- Una API de alto nivel en el host (PC)

En definitiva, permite montar una conexión USB que, entre otras cosas, nos deja:

- Programar la FPGA mediante JTAG
- Enviar y recibir datos a alta velocidad (bueno, a velocidad USB 2.0 al menos)
- Una abstracción de 127 canales independientes para manejar distintos tipos de tráfico

---

1 *Correcciones y sugerencias a*: cervettomarcos@gmail.com, edgardomarchi@gmail.com
2 https://github.com/makestuff/libfpgalink/wiki/FPGALink
3 https://store.digilentinc.com/nexys-2-spartan-3e-fpga-trainer-board-retired-see-nexys-4-ddr/
4 https://digilent.com/reference/programmable-logic/atlys/start

Objetivos

En lo que sigue, cuando se habla de host, se hace referencia a la PC, ya que es la que controlará los procesos de escritura y lectura hacia la FPGA.

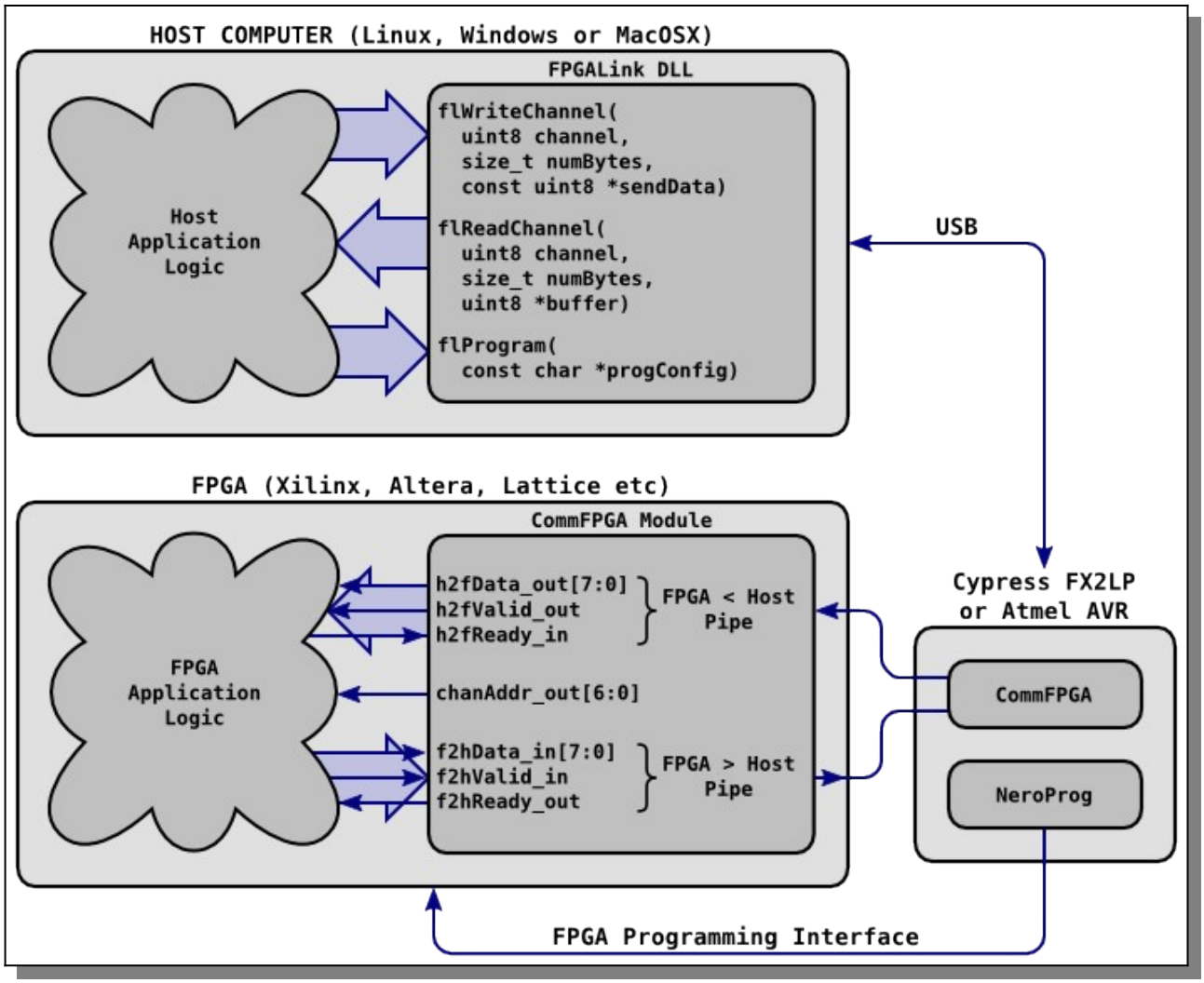
En este tutorial se realizan dos ejercicios:

- Una prueba de FPGALink con un ejemplo propio del autor
- Una prueba con un script escrito en Python, que además permite ver cómo se utiliza la API provista por FPGALink y abre las posibilidades a futuros diseños

Si bien el presente tutorial intenta ser autocontenido, se recomienda enfáticamente la lectura de la documentación provista por el autor, ya que es superior en profundidad al actual documento.

En la Figura 1 se puede observar la infraestructura provista por la librería. Se puede ver que provee soporte para varios sistemas operativos, FPGAs y microcontroladores.

*Figura 1 Se pueden observar las 3 partes constitutivas de **FPGALink***



Antes de empezar

Si usan la VM que nosotros les proveemos, este paso ya está hecho, pueden pasar a **"Instalando el FPGALink y probando flcli"**

Antes de empezar tenemos que setear algunas variables de entorno y hacer fácil la ejecución de las herramientas de xilinx en un entorno linux, así que vamos al ANEXO I.

¿Ya hiciste todo lo del ANEXO I? Bien. Si todavía no lo hiciste, malo, muy malo. Al ANEXO I ahora mismo.

Lo primero a tener en cuenta es que el kit sea programable por medio de las herramientas tradicionales de Xilinx, en particular, a través del software iMPACT (esto no es estrictamente necesario, ya que las herramientas de FPGALink permiten la programación directa de la FPGA).

Necesitamos:

- Digilent Adept[5]: *Communicate with Digilent system boards*

- Digilent Plugin for Xilinx Tools[6]: E*nables Xilinx software to directly use the Digilent USB-JTAG*

Vamos por pasos:

- **Digilent Adept**: bajar de la web la versión correspondiente a su sistema operativo. Para distribuciones basadas en debian, por ejemplo, bajar el .deb de la página, que tiene la forma:

digilent.adept.runtime_2.17.1-amd64.deb

Para instalarlo hacer:

```
$ sudo dpkg -i digilent.adept.runtime_2.17.1-amd64.deb
```

- **Digilent Plugin for Xilinx Tools**: bajar de la web la versión correspondiente a su sistema operativo. Para linux, por ejemplo, bajar el **.tar.gz** de la página, que tiene la forma:

libCseDigilent_2.5.2-x86_64.tar.gz

Descomprimir…

```
$ tar -xvf libCseDigilent_2.5.2-x86_64.tar.gz
```

Para instalar Digilent Plugin for Xilinx Tools, es necesario copiar los archivos *de la versión correspondiente de Xilinx*, por ejemplo:

```
$ cd libCseDigilent_2.5.2-x86_64/ISE14x/plugin
```

libCseDigilent.so
libCseDigilent.xml

al directorio

---

5 https://reference.digilentinc.com/reference/software/adept/start?redirect=1#software_downloads

6 https://reference.digilentinc.com/reference/software/digilent-plugin-xilinx-tools/start?s[]=digilent&s[]=plugin

```
$XILINX/lib/lin/plugins/Digilent/libCseDigilent   ←- 32 bits
$XILINX/lib/lin64/plugins/Digilent/libCseDigilent ←- 64 bits
```

haciendo, en nuestro caso (hacer **sudo** si es necesario):

```
$ mkdir -p $XILINX/lib/lin64/plugins/Digilent/libCseDigilent
$ cp libCseDigilent.* $XILINX/lib/lin64/plugins/Digilent/libCseDigilent
```

Si estos archivos ya existen en el directorio, reemplazarlos.

Para comprobar su funcionamiento, iniciamos iMPACT[7] y comprobamos que se pueda conectar correctamente

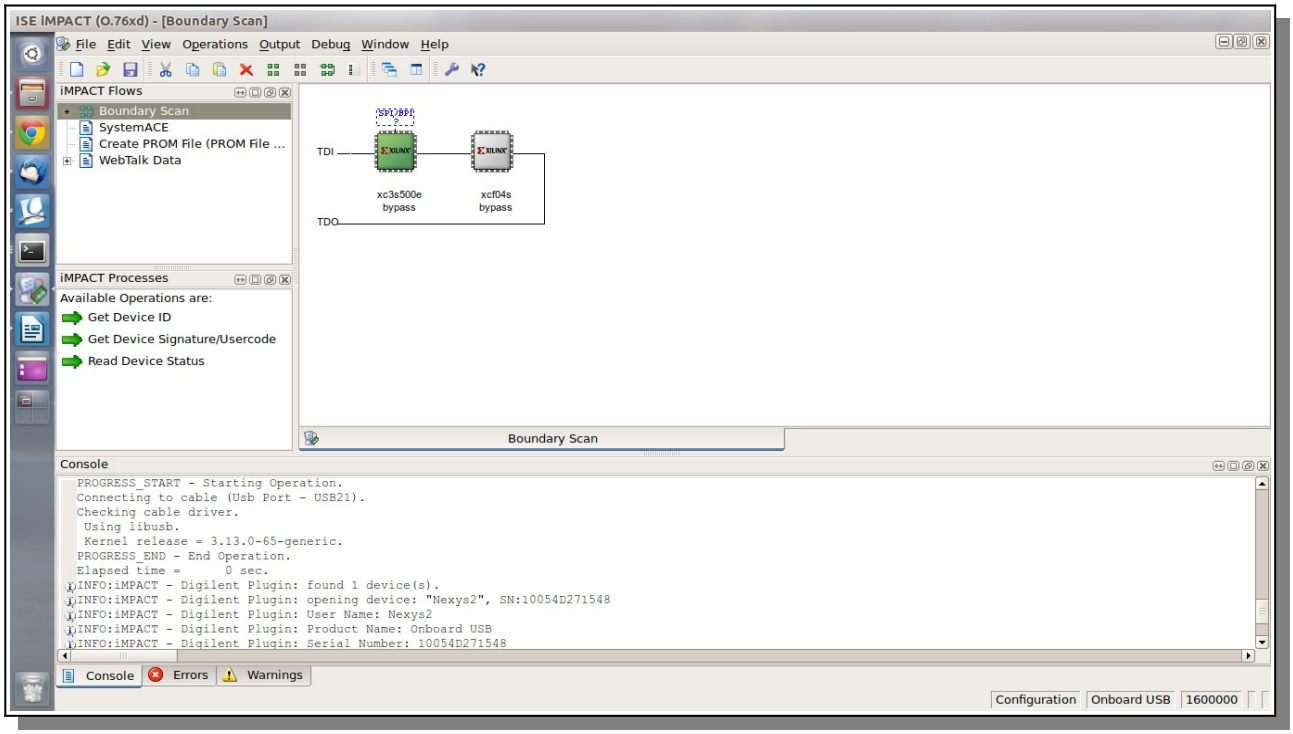


Figura 2 iMPACT detectando correctamente el kit Nexys2

Si vemos eso, genial! Ya Xilinx puede ver nuestro kit. Pero no festejemos todavía: *winter is coming*

Para poder acceder al kit mediante FPGALink es necesario editar un archivo *rules* de *udev*

```
$ sudo vim /etc/udev/rules.d/nexys.rules
```

y pegar el siguiente contenido, teniendo en cuenta:

```
ACTION=="add", SUBSYSTEM=="usb", ATTR{idVendor}=="1443",
ATTR{idProduct}=="0005", GROUP="pong", MODE="0660"
```

- cambiar el grupo (**GROUP**) por aquel que accederá al dispositivo
- Los números 1443:0005 son los de nexys2 y 1443:0007 para atlys

Para saber VID:PID de nuestra placa, la conectamos y escribimos **sudo dmesg** en la terminal. Algo así aparecerá:

```
New USB device found, idVendor=1443, idProduct=0005
```

---

7 Si siguieron los pasos del ANEXO I, esto implica ejecutar **xilinx_bash** y tipear **impact**

indicándonos el vendor id y el product id

y reiniciamos el daemon udev

```
$ sudo service udev restart
```

Desconectamos y reconectamos el kit. Hecho esto, estamos listos para comenzar con la instalación y prueba de la librería FPGALink.

Instalando el FPGALink y probando flcli

Disponemos de 2 scripts:

install_fpgalink.sh: esto compila e instala en el sistema la biblioteca necesaria para poder utilizar FPGALink en nuestras aplicaciones.

test_fpgalink.sh: prueba un ejemplo del autor de FPGALink con la nexys2-500 o atlys

Se usa de esta manera:

test_fpgalink.sh <board>

Donde **<board>** puede ser una de las siguientes: **nexys2-500** | **atlys**

Para los curiosos que quieran entender un poco lo que hacen los scripts, vayan al ANEXO II

Para más información sobre las opciones de flcli tipear **flcli -help**

(la aplicación quedó en el directorio **~/FPGALink/makestuff/apps/flcli/lin.x64/rel**)

Creación de un loop con FPGALink. Aplicación en Python

La idea es generar un loop interno a la fpga con fpgalink, si es posible con fifos de escritura y lectura.

Lado FPGA

Por el lado de la FPGA, vamos a programar un loop simple, basado en los archivos ya mencionados y agregando una FIFO de lectura y otra de escritura.

Crear un proyecto en ISE o planAhead y luego sintetizar y generar el archivo de programación .bit

--- **Para crear el proyecto utilizamos los archivos en el directorio "rtl"**

Agreguen de utils:

funciones.vhd, mod_m_counter.vhd, tictoc.vhd

funciones.vhd en librería "comun" (no en "work")

Agreguen de loopFPGALink:

top_level.vhd, comm_fpga_fx2.vhd

Agreguen los cores FIFOS:

fifo8t32.xco, fifo32to8.xco

Agreguen el archivo ucf de la placa correspondiente

loopFPGALink.ucf

Sintetizar, Implementar y generar el bitstream. Luego lanzar tenemos que convertir el archivo a formato xsvf

Para eso disponemos de un script que convierte el archivo .bit a .xsvf:

convert_to_xsvf.sh <board> <bit_file> <xsvf_file>

IMPORTANTE: Las FIFOs, así como están conectadas en el proyecto deben ser First Word Fall Through (FWFT), para que el sistema funcione correctamente.

Lado PC

Loop con python

Creamos un archivo random.dat para enviar a la FPGA

```
$ dd if=/dev/urandom of=random.dat bs=512 count=64
```

Hacemos ejecutable el script:

```
$ chmod +x flLoop.py
```

y ejecutamos:

```
$ ./flLoop.py -b <board> random.dat -x path/to/xsvf
```

Si el loop internamente está bien realizado, debería generar una copia idéntica a **random.dat** en **out.dat**

Podemos chequear eso con

```
$ diff random.dat out.dat
```

ANEXO I

Crear un script para iniciar las herramientas de Xilinx

Recomendamos crear un script para inicializar una terminal con todas las variables de xilinx ya seteadas. Este archivo se podría llamar **xilinx_bash** (el nombre no es relevante), con el siguiente contenido:

```
#!/bin/bash
cd ~/Xilinx/workspace
export LANG="en_US.UTF-8"
export LC_CTYPE="en_US.UTF-8"
export LC_NUMERIC="en_US.UTF-8"
export LC_TIME="en_US.UTF-8"
export LC_COLLATE="en_US.UTF-8"
export LC_MONETARY="en_US.UTF-8"
export LC_MESSAGES="en_US.UTF-8"
export LC_PAPER="en_US.UTF-8"
export LC_NAME="en_US.UTF-8"
export LC_ADDRESS="en_US.UTF-8"
export LC_TELEPHONE="en_US.UTF-8"
export LC_MEASUREMENT="en_US.UTF-8"
export LC_IDENTIFICATION="en_US.UTF-8"
source /opt/Xilinx/14.7/ISE_DS/settings64.sh
bash
```

Hacemos que este archivo sea ejecutable y lo ponemos en algún lugar que nos guste:

```
$ chmod a+x xilinx_bash
```

Luego, desde el $HOME hacemos:

```
$ mkdir -p Xilinx/workspace
```

Este será el directorio de ejecución de Xilinx. Las aplicaciones de Xilinx suelen tirar logs, journals y demás en el directorio donde se ejecuta.

Perfecto! Ya tenemos todo seteado. Podemos ejecutar nuestro script dirigiéndonos al directorio donde se encuentra el script **xilinx_bash** y haciendo:

```
$ ./xilinx_bash
```

Desde ahí ya podemos iniciar las herramientas de xilinx, tipeando su nombre: **ise, planAhead, impact**, etc.

ANEXO II

Antes que nada, empiecen con una consola de bash limpia, SIN HABER EJECUTADO **xilinx_bash**

Para empezar, vamos a seguir el proceso detallado en el *Quickstart de Nexys2*[8]*:*

Obtenemos la infraestructura primero:

```
$ sudo apt install build-essential libreadline-dev libusb-1.0-0-dev python3-yaml 2to3
$ mkdir -p $HOME/FPGALink/20170708
$ cd $HOME/FPGALink/20170708
$ wget -O- http://tiny.cc/msbil | tar zxf -
```

y luego obtenemos y compilamos la aplicación flcli

```
$ cd makestuff/apps
$ ../scripts/msget.sh makestuff/flcli/20170708
$ cd flcli
$ make deps
$ cd ../../
```

Ahora ya tenemos compilado flcli, lo cual nos permite acceder a la FPGA desde el host.

Nos falta compilar el firmware que va en la FPGA. Descargamos y compilamos la versión VHDL de cksum para nexys2-500 (reemplazar por nexys2-1200 de ser necesario)

Vamos a necesitar

```
$ scripts/msget.sh makestuff/hdlmake
$ cd hdlmake/apps
$ 2to3 --write ../bin/hdlmake.py
$ sed -i '157s/file/open/; 107s/file/open/; 191s/file/open/' ../bin/hdlmake.py
$ ../bin/hdlmake.py -g makestuff/swled
$ cd makestuff/swled/cksum/vhdl
```

Sintetizamos el diseño, para eso debemos tener las herramientas de Xilinx cargadas en bash

```
$ source $(HOME)/Xilinx/14.7/ISE_DS/settings64.sh
$ ../../../../../bin/hdlmake.py -t ../../templates/fx2all/vhdl -b <board> -p fpga
```

Donde **<board>** es “atlys” ó “nexys2-500”

La siguiente línea carga el firmware FPGALink en el microcontrolador y habilita USB power (sólo en nexys2-500):

NEXYS2-500:

```
$ ../../../../../../apps/flcli/lin.x64/rel/flcli -v 1443:0005 -i 1443:0005 -d D7+
```

8 https://gist.github.com/makestuff/7829484

ATLYS:

```
$ ../../../../../../apps/flcli/lin.x64/rel/flcli -v 1443:0007 -i 1443:0007
```

IMPORTANTE: Si queremos que renumere con otro VID:PID(-v) distinto del de Digilent(1443:0005, para nexys2, 1443:0007 para atlys), no tenemos que olvidarnos de agregar esos permisos a udev como hicimos anteriormente.

Ahora programamos la FPGA:

```
$ ../../../../../../apps/flcli/lin.x64/rel/flcli -v <vid>:<pid> -p J:D0D2D3D4:fpga.xsvf
```

En este momento tendrían que prenderse los display de siete segmentos del kit y marcar (sólo en la nexys2-500 esto) 0000

Si anduvo todo esto, vayan a tomar algo. Probablemente no se entiende demasiado lo que está pasando. No se preocupen, es parte de crear la infraestructura. Pronto nos independizaremos de las herramientas y scripts de FPGALink y nos quedaremos sólo con lo que nos interesa.

Ahora escribimos 64 KiB de datos aleatorios al kit:

```
$ dd if=/dev/urandom of=random.dat bs=1024 count=64
```

```
$ ../../../../../../apps/flcli/lin.x64/rel/flcli -v <vid>:<pid> -a 'w0 "random.dat";r1;r2' -b
```

Y leemos 64KiB (0x10000 bytes) del kit y lo guardamos en "out.dat":

```
$ ../../../../../../apps/flcli/lin.x64/rel/flcli -v <vid>:<pid> -a 'r0 10000 "out.dat"'
```