Plataforma de hardware reconfigurable

JTAG – Configuración OOCD-Links, (*Hardware & Software*)

Luis A. Guanuco

Marzo 2013

Índice

1. Introducción

El presente reporte desarrolla la continuación del armado, testeo de las distintas placas que conformarán la *Plataforma de Hardware Reconfigurable (PHR)*. Además, gran parte del informe se basa en el *software* que permitirá la comunicación entre la PC y el programador *Joint Test Action Group* (JTAG).

2. *Software* OpenOCD

Los *scripts* utilizados se obtuvieron del manual de usuario del *software* OpenOCD que se encuentra publicado en la web[1]. Anteriormente al uso de OpenOCD se intentó utilizar el *software* UrJTAG[2] pero debido a la limitaciones que presentaba la versión estable se lo descartó.

La instalación y puesta en funcionamiento del *software* OpenOCD se realiza en tres etapas:

- Instalación del *software* OpenOCD
- *Interface*

2.1. Instalación del *software* OpenOCD

Se recomienda descargar la última versión estable de OpenOCD y compilarlo manualmente con los correspondientes argumentos necesarios para el *driver* a utilizar. En nuestro caso utilizaremos el *driver* libftdi[3]. Además en la documentación oficial de OpenOCD se hace referencia a los requerimientos para cada sistema operativo (SO). Como ya se mencionó en anteriormente reportes, se utilizará herramientas libres, por lo que se realizará la instalación en un SO GNU/Linux Debian "Squeeze" 6.0. Como así también los drivers para el *hardware* están licenciados como *General Public License* (GPL). Un documento de referencia muy útil es la Nota Técnica *Entorno de desarrollo de firmware sobre arquitecturas ARM Cortex-M3, basado en herramientas libres*[1].

2.2. *Interface*

Interface hace referencia al *hardware* que realiza el enlace entre el puerto JTAG del dispositivo al que se quiere acceder y la PC. En éste caso el *interface* será nuestra placa OOCD-Links. OpenOCD dispone de *scripts* para varios *interface* entre los cuales se encuentra nuestra placa programadora. La información que proporciona éste código hace a que tipo de driver se utilizará como así también identificación del dispositivo central. A continuación se puede ver la estructura del archivo `oocdlink.cfg`.

```
#
# Joern Kaipf's OOCDLink
#
# http://www.joernonline.de/contrexx2/cms/index.php?page=126
#

interface ft2232
```

[1] http://openocd.sourceforge.net/doc/html/index.html
[2] http://urjtag.org/
[3] http://www.intra2net.com/en/developer/libftdi/

```
ft2232_device_desc ``OOCDLink``
ft2232_layout oocdlink
ft2232_vid_pid 0x0403 0xbaf8
jtag_khz 5
```

Para obtener el VIP (vendor ID) y el PID (product ID) se lo puede obtener una vez conectada el *interface* a la PC y listando los dispositivos conectados al puerto USB, por ejemplo,

```
luis@luis-laptop:openocd$ lsusb
Bus 008 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 006 Device 002: ID 0403:6010 Future Technology Devices International,
    Ltd FT232 USB-Serial (UART) IC
Bus 006 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 002: ID 0bda:0158 Realtek Semiconductor Corp. USB 2.0
   multicard reader
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

En la salida del `lsusb` se puede ver que tenemos el dispositivo *0403:6001 Future Technology Device International*. Luego utilizamos éste dato para modificar o crear nuestro archivo `openocd.cfg` que será argumento de OpenOCD.

2.3. Documentación de comandos y *scripts*

Una vez que se logra comunicar el *interface* JTAG con el *software* OpenOCD, se crean los archivos necesarios para acceder a los dispositivos que se encuentren conectados en el protocolo JTAG. Para correr OpenOCD correctamente se necesita pasarle algunos simples argumentos,

```
luis@luis-laptop:trunk$ openocd -h
Open On-Chip Debugger 0.6.1 (2012-12-08-02:21)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.sourceforge.net/doc/doxygen/bugs.html
Open On-Chip Debugger
Licensed under GNU GPL v2
--help       | -h       display this help
--version    | -v       display OpenOCD version
--file       | -f       use configuration file <name>
--search     | -s       dir to search for config files and scripts
--debug      | -d       set debug level <0-3>
--log_output | -l       redirect log output to file <name>
--command    | -c       run <command>
```

Los más importantes de éstos son `--file`, que permite definir que archivo de configuración OpenOCD utilizará. Dentro de éste archivo se podría agregar cualquiera de los otros comandos. Es recomendable que la estructura de archivos sea la siguiente,

- Directorio raíz de trabajo (por ej.: `openocd_dir`)
  - → `core.cfg`, contiene los comandos básicos para que OpenOCD reconozca el *interface*
  - → `openocd.cfg`, contiene los comandos que configura el acceso al protocolo JTAG
  - → `device.cfg`, contiene los comandos propios de los dispositivos a conectar en la cadena JTAG, por ejemplo ID, cantidad de bits de los registros, etc.

→ `files/...`, también se debe conciderar tener un subdirectorio en el cual almacenar archivos que se transerirán a los dispositivos mediante JTAG, por ejemplo: archivos .svf, archivos .elf, etc.

A continuación se describen comandos útiles que se pueden aplicar en forma general a cualquier dispositivo a conectar.

2.4. Agregar un TAP

Para acceder a un dispositivo mediante el protocolo JTAG, se debe proporcionar especificaciones del mismo. Muchas veces ésta información no se encuentra fácil de acceder en la hoja de datos de los dispositivos conectados al *interface*. Para situaciones como la descrita, OpenOCD dispone de un modo llamado *Autoprobing*[2].

En definitiva, para capturar la información del dispositivo conectado se debe generar una archivo base `openocd.cfg` como el que se muestra a continuación,

```
# OpenOCD configuration script
# 2013-03-25 lguanuco
# Hardware:
# - OOCD-Link-s

# (``3'': max. verbosity level)
debug_level 3
# (set log file)
log_output out.log

source [find core.cfg]

# 3. Other configs

reset_config trst_and_srst
jtag_rclk 8
```

Luego de correr OpenOCD, se debe acceder al archivo de salida con el nombre que se asignó en el archivo `openocd.cfg`, `out.log`. Este archivo registra todos los mensajes de salida que genera OpenOCD, a continuación se muestra parte de éste archivo donde se puede observar información útil para generar el TAP de dicho dispositivo.

```
...
Debug: 129 273 core.c:323 jtag_call_event_callbacks(): jtag event: TAP
    reset
Warn : 130 359 core.c:1145 jtag_examine_chain(): AUTO auto0.tap - use ``
    jtag newtap auto0 tap -expected-id 0x4f1f0f0f ...''
Debug: 131 359 core.c:1323 jtag_tap_init(): Created Tap: auto0.tap @ abs
    position 0, irlen 0, capture: 0x1 mask: 0x3
Debug: 132 359 core.c:1208 jtag_validate_ircapture(): IR capture
    validation scan
Warn : 133 369 core.c:1244 jtag_validate_ircapture(): AUTO auto0.tap -
    use ``... -irlen 4''
Debug: 134 369 core.c:1267 jtag_validate_ircapture(): auto0.tap: IR
    capture 0x01
Debug: 135 369 openocd.c:145 handle_init_command(): Examining targets...
...
```

Una vez obtenida éstas lineas se puede rescatar los datos más importantes que son,

- *id*

- *irlen*
- *capture*
- *mask*

Por lo tanto, con estos datos se puede generar nuestro propio *script* en el cual agregar manualmente el TAP, por ejemplo `lpc2124.cfg`

```
jtag newtap arm_lpc2124 tap -irlen 4 -ircapture 0x01 -irmask 0x3 -
    expected-id 0x4f1f0f0f
```

2.5. Multiples TAP

Muchas plataformas o kits de desarrollo tienen multiples TAPs. Por ejemplo, en nuestro caso tenemos conectados dos dispositivos a la cadena JTAG que son la *Field Programmable Gate Array* (FPGA) y la memoria *Programmable Read-Only Memory* (PROM) de programación del mismo. Si la conexión es tal que,

- OpenOCD TDI(output) → XC3S50A pin TDI(BS input).
- XC3S50A pin TDO(BS input) → XCF01S pin TDI.
- XCF01S pin TDO(BS output) → OpenOCD TDO(input).

Los comandos para agregar éstos TAPs deberán respetar él orden con el que fueron conectados físicamente,

```
...
jtag newtap xcf01s tap -irlen ...
jtag newtap xc3s50a tap -irlen ...
...
```

Es decir, se agrega los TAPs recorriendo la cadena JTAG desde el pin TDO al pin TDI desde el interface, aquí sería desde la placa OOCDLink.

2.6. Programación CPLD

Para los CPLD, primero se debe tener el archivo *Serial Vector Format* (SVF), que es generado por la herramienta con la que se diseña y sintetiza el código VHDL o Verilog. En éste caso se utiliza el *software* ISE Xilinx. Una vez que se tiene el archivo SVF, desde OpenOCD, se corre el siguiente comando;

```
svf -tap tap_name file.svf quiet
```

Lo que generaría una salida como la que sigue,

```
svf processing file: ‘‘file.svf’’
500 kHz
...
...
Time used: 0m7s478ms
svf file programmed successfully for 8468 commands
```

En el caso de que se tenga problemas en la transferencia del archivo, se podría probar bajando la frecuencia de la comunicación JTAG.

Algo interesante se puede observar en el archivo `file.svf`, pues en una de sus lineas tiene el IDCODE del dispositivo al que se va a transferir, que debería coincidir con el que se le pasa cuando se agrega el TAP al código de OpenOCD. O quizá diferenciarse con el valor hexadecimal más significativo, que indica la versión del dispositivo.

```
...
//Loading device with 'idcode' instruction.
SIR 8 TDI (fe) SMASK (ff) ;
SDR 32 TDI (00000000) SMASK (ffffffff) TDO (f9604093) MASK (0fffffff) ;
//Check for Read/Write Protect.
SIR 8 TDI (ff) TDO (01) MASK (e3) ;
//Boundary Scan Chain Contents
//Position 1: xc9572xl
...
```

La placa OOCD-Links cuenta con un dispositivo central, FT2232D. Éste dispositivo posee dos alimentaciones, por una parte la alimentación de núcleo del circuito integrado y por otro lado la alimentación de los interfaces de salida. La primera alimentación es tomada desde el puerto USB al que se encuentra conectado, mientas que la alimentación de sus puertos de conexión lo toma del conector JTAG, del *Pin 1*. De tal manera que la tensión de los puertos de conexión JTAG del *interface* siempre tiene la misma tensión que maneja las señales JTAG de la placa conectada. Es decir, si se hace una conexión entre la placa OOCD-Links y una placa con un CPLD XC9572XL, ambas tendrán la misma tensión en sus puertos JTAG, en éste caso 3.3V.

Un inconveniente podría ser el consumo de corriente, pues el regulador de la placa OT-CPLD se tendría que proporcionar energía a la placa OOCD-Links, pero como es en el proceso de programación y como proceso de testeo, no debería haber inconvenientes. Igualmente se debe tener en cuenta.

Se encontró en la nota de aplicación de Xilinx [*ug445.pdf*] que para los pines TDI y TMS de los CPLD XC9572XL, se dispone de resistores *pull-ups* internos. Por lo que no es necesario colocar resistores externos en el diseño del PCB.

2.7. *Debugging*

Para el *debugging* se utiliza varias herramientas y todas ellas corren sobre terminales *bash* de nuestro SO. Entre éstos *software*, tenemos

- *terminator*, múltiples terminales en una sola ventana (escritorio GNOME).
- *tailf*, sigue la escritura de un archivo(por ej.: archivos de .log).
- *ccze*, un robusto visor de archivos log con la posibilidad de colorear las líneas.
- *grep*, imprime las líneas similares a un patrón.

Todos éstos comandos son utilizados a la vez. La forma de trabajo es correr *terminator* y luego dividir la pantalla en varios terminales al vez, eso será útil para ingresar comandos al puerto telnet 4444 donde se puede acceder al *interface* JTAG. Los otros *software* se utilizar a modo de *debugger*, pues nos permiten resaltar mensajes que genera OpenOCD. Por ejemplo, una forma estándar de ver la salida es,

```
luis@luis-laptop:~$ tailf  codigo/jtag/openocd/find_tap/out.log | ccze -A
    -o scroll
```

También se podría utilizar *grep* para filtrar o diferenciar las líenas en función si son *Error, Warning, Debug & User*.

3. *Software xc3sprog*

Si bien el OpenOCD se consideró como el *software* a utilizar, se encontraron algunos inconvenientes a la hora de interactuar con la placa PHR. El problema, en principio, se debía a que el OpenOCD no puede manejar archivos binarios que pesaran más de 2Mb[REF]. Existe la posibilidad de modificar OpenOCD de forma tal que se pueda transferir datos binarios con tamaños superiores a 2Mb pero quizá genere otro tipos de errores como consecuencia.

Frente al inconveniente planteado en el párrafo anterior, se busco herramientas alternativas. Mediante el contacto con otros desarrolladores/usuarios de plataformas basadas en PLDs se encontró el *software xc3sprog*. Este programa permite con unos simples comandos programar varias FPGAs, memorias PROM y hasta microcontroladores mediante el interfaz JTAG.

En una aplicación típica, xc3sprog lee un archivo `.bit` generado por la herramienta de diseño de la FPGA y programará el chip PROM de la placa que contiene la FPGA. Como su nombre lo indica, xc3sprog fue diseñado originalmente para la familia de FPGA Sparta-3 de Xilinx. Sin embargo se ha extendido el manejo a varios otros tipos de dispositivos que incluyen otras FPGAs, CPLDs, XCF flash PROMs, microprocesadores AVRs de Atmel y memorias flash SPI. El *software* xc3sprog soporta varios cables JTAG, incluyendo cables de puerto paralelo y programadores USB.

3.1. Instalación del *software* xc3sprog

Para obtener el xc3sprog, primero se debe descargar el código fuente desde SourceForge[4]. El proceso se realizará sobre un terminal del SO GNU/Linux que estemos utilizando.

```
luis@luis-laptop:~$ cd ~/
luis@luis-laptop:~$ mkdir sourceforge
luis@luis-laptop:~$ cd sourceforge
luis@luis-laptop:~$ svn co https://xc3sprog.svn.sourceforge.net/svnroot/
    xc3sprog/trunk xc3sprog
```

Luego se debe compilare e instalar el programa

```
luis@luis-laptop:~$ cd  xc3sprog
luis@luis-laptop:~$ cmake .
luis@luis-laptop:~$ make
luis@luis-laptop:~$ make install
```

En esta sección no se hace referencia al *driver* de la placa OOCDLink pues los pasos para la instalación del controlador son los mismos que se describieron en la sección 2.1.

[4] http://sourceforge.net/projects/xc3sprog/

3.2. Documentación de comandos y *scripts*

Al igual que el *software* OpenOCD, xc3sprog recibe varios argumentos en función de como se quiera interactuar con los dispositivos a programar. Para listar las opciones que se tienen, se debe correo el programa desde el directorio donde se construyó el proyecto, llamado `build`,

```
luis@luis-laptop:~$ cd ~/sourceforge/xc3sprog/build
luis@luis-laptop:~$ ./xc3sprog
XC3SPROG (c) 2004-2011 xc3sprog project $Rev: 747 $ OS: Linux
Free software: If you contribute nothing, expect nothing!
Feedback on success/failure/enhancement requests:
        http://sourceforge.net/mail/?group_id=170565
Check Sourceforge for updates:
        http://sourceforge.net/projects/xc3sprog/develop

usage:  xc3sprog -c cable [options] <file0spec> <file1spec> ...
        List of known cables is given with -c follow by no or invalid
            cablename
        filespec is filename:action:offset:style:length
        action on of 'w|W|v|r|R'
        w: erase whole area, write and verify
        W: Write with auto-sector erase and verify
        v: Verify device against filename
        r: Read from device,write to file, don't overwrite existing file
        R: Read from device and write to file, overwrite existing file
        Default action is 'w'

        Default offset is 0

        style: One of BIT|BIN|MCS|IHEX|HEX
        BIT: Xilinx .bit format
        BIN: Binary format
        MCS: Intel Hex File, LSB first
        IHEX: INTEL Hex format, MSB first (Use for Xilinx .mcs files!)
        HEX:  Hex dump format
        Default for FPGA|SPI|XCF is BIT
        Default for CPLD is JED
        Default for XMEGA is IHEX
        Default length is whole device
```

xc3sprog utiliza dos archivos más que ofrecen mayor información y configuración tanto al *interface* JTAG como también sobre los dispositivos que vayamos a programar. Estos archivos se encuentran en el mismo directorio `build` donde se compiló el programa, estos son: `cablelist.txt` y `devlist.txt`.

3.2.1. Configuración del *interface*

El archivo `cablelist.txt` contiene, por cada línea, diferentes tipos de configuraciones para *interface* de programación JTAG. Si bien son varios, a continuación se listan algunos de los más reconocidos en los sistemas embebidos.

```
luis@luis-laptop:build$ cat cablelist.txt
# Alias       Type  OptString Max_Freq
# OptString for ftdi:
# VID:PID:PRODDESC:INTERFACE:DBUS_DATA:DBUS_EN:CBUS_DAT:ACBUS_EN
# OptString for pp:
# OptString for xps:  VID:PID
```

```
# Max_Freq == 0 mean use maximum speed of device
# Use 1500000 for all cable connected cables and max for all on board
    cables

ftdi            ftdi    1500000 0x0403:0x6010:
ftdiphr         ftdi    6000000 0x0403:0x6010:
papilio         ftdi    6000000 0x0403:0x6010:
ft232h          ftdi    1500000 0x0403:0x6014:
amontec         ftdi    1500000 0x0403:0xcff8::1:0x00:0x10:0x00:0x00
olimex          ftdi    1500000 0x15b1:0x0003::1:0x00:0x10:0x00:0x08
knob2usb        ftdi    0       0x0403:0x6010:KNOB2USB:0:0x00:0x10:0x00:0
    x40
xpc             xpc     0       0x03fd:0x0008
llbbc08         fx2     0       0xfffe:0x0018
pp              pp      0       NULL
dlc5            pp      0       NULL
jtaghs1         ftdi    1500000 0x0403:0x6010:Digilent Adept USB Device:0:0
    x80:0x80:0x00:0x0
turtelizer      ftdi    1500000 0x0403:0xbdc8:Turtelizer JTAG/RS232 Adapter
    :0:0x00:0x10:0x00:0x0
arm-usb-ocd-h ftdi      1500000 0x15ba:0x002b::1:0x00:0x10:0x00:0x08
```

Se puede ver que el formato de este documento consiste en

- El nombre con que se define el *inteface*
- El *hardware* que se utiliza, es decir, si se utiliza el puerto paralelo, el usb a través de un dispositivo FTDI, etc.
- La frecuencia en Hertz a la que trabajará el *interface* JTAG
- El identificador del *hardware* según el SO GNU/Linux.

Se toma como ejemplo algunas de estas configuraciones que utilice como base el dispositivo FTDI y se agrega una nueva línea con la configuración de nuestra placa OOCDLink. Es la segunda configuración disponible del archivo `cablelist.txt`,

```
  ftdiphr       ftdi    6000000 0x0403:0x6010:
```

3.2.2. Dispositivos soportados

El archivo `devlist.txt` contiene información de los diferentes dispositivos soportados por xc3sprog. Este documento se presente a modo descriptivo pues no tenemos por que modificar ninguna de las líneas de este archivo. El programa xc3sprog utiliza este documento para obtener fundamentalmente el tamaño de algunos registros mediante la identificación del *IDCODE* que proporciona todo los dispositivos JTAG cuando son escaneados. Como se podrá apreciar, se ha evitado incluir todos los dispositivos ya que son varias líneas, por lo que se utilizó los puntos $\cdots$ para acortar el documento.

```
luis@luis-laptop:build$$ cat devlist.txt
# IDCODE IR_len ID_Cmd Text
...
04001093      6    0x9 XC6SLX9
04002093      6    0x9 XC6SLX16
04004093      6    0x9 XC6SLX25
...
02210093      6    0x9 XC3S50A
```

```
02218093      6    0x9 XC3S200A
02220093      6    0x9 XC3S400A
02228093      6    0x9 XC3S700A
02230093      6    0x9 XC3S1400A
...
05044093      8    0xfe XCF01S
05045093      8    0xfe XCF02S
05046093      8    0xfe XCF04S
...
#XC95XL
09602093      8  0xfe XC9536XL
09604093      8  0xfe XC9572XL
09608093      8  0xfe XC95144XL
09616093      8  0xfe XC95288XL
...
#unsupported
#Virtex2
01020093      6     5 XC2V500
#XC95
09502093      8  0xfd XC9536
09504093      8  0xfd XC9572
09506093      8  0xfd XC95108
...
##Atmel
#list should not care for the version part
0978203f      4   0x1 AT90USB
0958103f      4   0x1 AT90CAN32
0968103f      4   0x1 AT90CAN64
0978103f      4   0x1 AT90CAN128
...
#ARM 7 in ICE Mode (e.g. AT91SAM7X)
3f0f0f0f      4   0xe  ARM-ICE-MODE
4f1f0f0f      4   0xe  ARM-ICE-MODE(ARM7TDMI-S rev4)

#AVR32
01e8203f     5   0x1 AT32AP7000

#STM
06410041     5   0x1 STM32L1_Med_density
...
#ARM Debug
3BA00477     4   0xe ARM_Cortex-M3_r1p1-01rel0
4BA00477     4   0xe ARM-Cortex-M4F_r0p1

#Unsure about IR_Len...
0270817f     11  0x1 BCM2835

#Manufacturer list
00000093        99  0 Xilinx_Unknown
0000003f        99  0 Atmel_Unknown
```

3.2.3. Comandos básicos

En el *usage* del xc3sprog define una estructura básica para correr comandos. El comando se debe correr desde el directorio `build` donde se ha compilado el programa.

```
xc3sprog -c cable [options] <file0spec> <file1spec> ...
```

En este comando, lo principal es definir que *cable* se utilizará de la lista que tenemos en el archivo `cablelist.txt`. Para esto se debe indicar luego del `-c` el nombre de la lista de *interfaces*. Una vez definido el cable, se concatenan demás tareas que debe procesar el xc3sprog, pero ya teniendo referencia a que tipo de *interface* está conectado. Los tareas son,

`-j` Detecta la cadena JTAG e imprime una lista de dispositivos encontrados. La numeración de la posición del dispositivo en la cadena se cuenta desde el dispositivo conectado al pin TDO del *interface*.

`-p val` Define a que dispositivo de la cadena JTAG se está por acceder, donde `val` es el número que ocupa en la cadena y es visualizado por el comando `-j`. El valor por defecto de `val` es 0.

`-T n` Testea la cadena JTAG `n` veces. Cuando se corre en modo ISF, se testea la conexión SPI.

- Si `n` no se específica, el valor por defecto es de 10,000 veces.
- Si `n = 0` se mantendrá en un testeo para siempre.

`-J freq` Ejecuta a una frecuencia específica el clock del JTAG (`freq` se encuentra en Hz). Si no se proporciona este dato, o `freq = 0`, el *interface* trabajará a la máxima frecuencia que soporte y se encuentre definido en el archivo `cablelist.txt` (véase 3.2.1).

`-e` Borra el dispositivo seleccionado por el parámetro `-p`.

`-I file` Define si se trabaja en modo ISF para programar una memoria serial *flash* interna. La memoria *flash* está junta al dispositivo JTAG primario, pero no es accesible directamente mediante la cadena JTAG. El *interface* JTAG primario es usado como un *proxy* para enviar datos sobre SPI a la memoria *flash*. Si se especifica el argumento `file`, se comenzará por programar el bitfile especificado en el *interface* JTAG primario (típicamente una FPGA).

`-R` Envía un comando de reconfiguración al dispositivo de memoria del *hardware* utilizado (por ejemplo: XVC, XCF, XCFP para las FPGA o directamente a los dispositivos XC3S, XC6S, XC2V).

`-m mapdir` Búsqueda de archivos de mapeo (XC2C) en el directorio especificado por `mapdir`. Los archivos de mapeo (*map files*) son requeridos para manejar archivos JEDEC durante la programación de los CPLDs. Si se especifica el directorio, por defecto el valor que se toma es `$XC_MAPDIR`.

`-d /dev/parportN` Especifica el puerto paralelo a utilizar. Si no es especificado, por defecto se utiliza el valor `$XCPORT` o `/dev/parport0`.

`-s serialnum` Utiliza el puerto USB con *string* de número de serie específico.

`-L` Usa el controlador *libFTD2XX* en luag de *libftdi* para acceder a un *interface* basados en los dispositivos FTDI.

`-D` Genera los archivos `cablelist.txt` y `devlist.txt` en el directorio actual. Si ya existiesen, xc3sprog intentará generar los mismos archivos pero agregando al nombre de los archivos un número que incrementará.

`-v` Habilita la salida formal detallada (*verbose*).

`-h` Imprime un texto de ayuda para el programa.

3.3. Programación CPLD

3.4. *Debugging*

4. Documentación

La documentación resulta fundamental en ésta etapa del desarrollo. Si bien se quiere lograr el correcto funcionamiento de las placas, la documentación sirve para realizar correciones a las versiones futuras de cada placa. Otro objetivo es documentar el funcionamiento de cada dispositivo que sirvan al reporte final como así también a los usuarios de la *Plataforma de Hardware Reconfigurable*.

Referencias

[1] Sebastián García, "Entorno de desarrollo de firmware sobre arquitectura ARM Cortex-M3, basado en herramientas libres", 31 de Julio del 2013, Versión 0

[2] OpenOCD, "OpenOCD User's Guide", 25 de Noviembre del 2012, 10.7 Autoprobing, 58 p., Versión 0.7.0-dev

A. Acrónimos

PHR Plataforma de Hardware Reconfigurable

OpenOCD *Open On-Chip Debugger*

JTAG *Joint Test Action Group*

TAP *Test Access Port*

SVF *Serial Vector Format*

CPLD *Complex Programmable Logical Device*

FPGA *Field Programmable Gate Array*

PROM *Programmable Read-Only Memory*

SO sistema operativo

GPL *General Public License*

UTN-FRC Universidad Tecnológica Nacional – Facultad Regional Córdoba

B. Repositorio de proyecto

El proyecto se encuentra alojado en los servidores de *OpenCores*. Por lo que se puede acceder a los repositorios mediante el siguiente link, `http://opencores.org/project,phr` De todas formas se pueden comunicar por correo, `guanucoluis@gmail.com`.