Plataforma de hardware reconfigurable

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

Luis A. Guanuco

Marzo 2013

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. Programador JTAG

Los *scripts* utilizados se obtuvieron del manual de usuario del *software Open On-Chip Debugger* (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.

2.1. *Software* OpenOCD

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

- Instalación del *software* OpenOCD
- *Interface*

2.1.1. 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

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

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].

2.1.2. *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
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-S
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.

2.2. 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.

2.2.1. Agregar un *Test Access Port* (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
Debug: 131 359 core.c:1323 jtag_tap_init(): Created Tap: auto0.tap @ abs position 0, irle
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
```

2.2.2. 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.

2.2.3. Programación *Complex Programmable Logical Device* (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.

2.3. *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. 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`.