FPGALibre: Herramientas de Software Libre para diseño con FPGAs.

Salvador E. Tropea, Diego J. Brengi, Juan P. D. Borgna

Instituto Nacional de Tecnología Industrial, Electrónica e Informática,
Unidad Técnica Instrumentación y Control, Argentina
http://utic.inti.gov.ar.

Abstract. En este trabajo se presenta el proyecto FPGALibre. Este proyecto fue creado con el fin de facilitar el desarrollo de diseños que involucran FPGAs mediante el uso de software libre. El proyecto abarca no sólo las herramientas necesarias para el desarrollo sino que también contempla incluir *IP cores*. Se presentan los objetivos del proyecto, sus ventajas y las herramientas actualmente seleccionadas para cada una de las etapas de diseño. El proyecto pone énfasis en plataformas de Software Libre pero está abierto a usuarios de otro tipo de plataformas.

1 Introducción

El proyecto FPGALibre nace con la idea de poder compartir herramientas y *cores* desarrollados en la Unidad Técnica Instrumentación y Control perteneciente al centro de Electrónica e Informática del Instituto Nacional de Tecnología Industrial (Argentina).

Durante el congreso SPL2005[1] nos encontramos con mucha gente deseosa de compartir conocimientos sobre el apasionante tema de las FPGA y por eso decidimos crear este espacio como un lugar abierto para todos los que quieran unirse.

Actualmente el proyecto se encuentra alojado en *Source Forge*[2] que provee todos los recursos necesarios para el manejo de proyectos de Software Libre (S.L.). La página web del proyecto es http://fpgalibre.sf.net

2 Objetivo del Proyecto

El objetivo principal de este proyecto es el de facilitar el intercambio de los elementos necesarios para el desarrollo con FPGA.

El intercambio de medios físicos (*hardware*) es complejo debido al costo de replicación y el intercambio de programas bajo licencias es un delito. Por estas razones es que el proyecto pone énfasis en S.L. o gratuito y en *cores* que puedan ser redistribuidos sin restricciones.

Partiendo de estas bases se enuncian los siguientes objetivos formales:

- Impulsar el desarrollo con dispositivos FPGA utilizando herramientas de S.L. u open source.
- Fomentar el intercambio y desarrollo de cores IP con licencias que posean el mismo espíritu que las del S.L.

3 Ventajas del Uso de Software Libre

Entre las ventajas del uso de software libre podemos citar:

- La capacidad de aprender observando el código fuente de las aplicaciones y los cores.
- La posibilidad de adaptar a gusto según las necesidades particulares de cada interesado.
- La oportunidad de mejorar el código y brindar esas mejoras al resto de la comunidad.
- Bajo costo. Los productos propietarios de este rubro suelen tener altos costos de licencias, lo que limita y restringe su aplicación en forma masiva en proyectos e instituciones de bajos recursos y en países en desarrollo.

4 Preferencias de Lenguaje, Codificación y Trabajo

4.1 Lenguaje HDL usado

Debido a su extenso uso en instituciones gubernamentales y universitarias el proyecto tomó como lenguaje preferencial el VHDL. Esto no implica que el proyecto sólo acepte contribuciones en este lenguaje.

Al mismo tiempo se decidió adherir al estándar IEEE 1076-1993 e IEEE 1164 evitando el uso de extensiones no estandarizadas al lenguaje VHDL. En particular el proyecto apoya el uso del *package* numeric_std en lugar de las extensiones de Synopsys.

4.2 Convenciones de Codificación

El lenguaje VHDL, al igual que muchos otros, ofrece cierta libertad en cuanto al estilo del código, indentado, nombres de las variables, etc. Sin embargo cuando se pretende fomentar el trabajo en grupo, permitir que otras personas colaboren y aumentar la reusabilidad del código, es necesario definir pautas que complementen la sintaxis del lenguaje. Para abordar este tema se toman dos recomendaciones en cuanto a escritura de código VHDL: Recomendaciones de la Agencia Espacial Europea[3] y recomendaciones para proyectos en OpenCores[4]. En base estas dos recomendaciones, el proyecto FPGALibre define sus recomendaciones, también llamadas *guidelines*[5], para escritura de código VHDL.

4.3 Bus de Interconexión

Para aprovechar al máximo los desarrollos de *IP cores* y sacar mayor beneficio de la modularidad que estos brindan, es necesario definir y establecer un mecanismo de interconexión común. Siguiendo los objetivos del proyecto se buscó una especificación abierta y libre para cubrir esta necesidad. Tomando como referencia el proyecto OpenCores se selecciona la arquitectura de interconexión Wishbone[6] para *IP*s en *SoC*. Además de sus características y ventajas técnicas, esta especificación es de dominio público y permite su libre utilización sin ningún tipo de contrato, acuerdo, royalty o pago.

5 Herramientas Relacionadas con el Código HDL

En esta sección describimos las herramientas utilizadas para escribir el código HDL así como también herramientas auxiliares que permiten generar código, facilitar su escritura y/o validarlo.

5.1 Edición del Código Fuente

No son muchos los editores libres que poseen facilidades avanzadas para la edición de código VHDL. Uno de estos editores es el SETEdit[7], un editor pensado para programadores, con soporte para gran cantidad de lenguajes de programación. Estas son algunas de las características que lo hacen una buena elección para el trabajo con VHDL:

- Resaltado de sintaxis para VHDL.
- Macros específicas con construcciones típicas de VHDL.
- Búsqueda de packages, components, functions, etc. a nivel del código actual o a nivel del proyecto utilizando Exuberant C Tags (ECTAGS[8]) con soporte específico para VHDL
- Indentado coherente con los guidelines del proyecto.

5.2 Verificación de Estilo

Para facilitar el cumplimiento de los *guidelines* del proyecto se desarrolló un pequeño *lint* llamado bakalint[9]. El mismo permite verificar que el código escrito cumple con los lineamientos del proyecto. Así mismo, y en caso de que el código no cumpla, el programa sugiere que cambios realizar para que el código cumpla.

5.3 Generador de Interconexión

Cuando se utiliza el bus de interconexión Wishbone es necesario crear un componente denominado *intercon*. Este se encarga de implementar los detalles de bajo nivel de la interconexión tales como mapeo en el bus de direcciones, resolución de colisiones, técnicas para acceder al bus, etc. Este bloque puede escribirse manualmente pero si se desea experimentar con distintas posibilidades o se desean realizar cambios en el mapeo de direcciones la tarea se torna compleja.

Para evitar esto el proyecto sugiere el uso de la herramienta Wishbone Builder[10]. La misma fue adaptada para cumplir con lineamientos del proyecto y para resolver algunos problemas que se encontraron en la versión original.

5.4 Búsqueda en el Código HDL

Cuando se trabaja en proyectos de mediana complejidad el código se particiona estructuralmente y jerárquicamente. Cuando el número de archivos fuente crece realizar búsquedas se torna complejo. Para facilitar esta tarea el proyecto optó por agregar soporte para el lenguaje VHDL a la herramienta Exuberant C Tags[8].

C Tags es un programa muy usado en el mundo Unix para realizar búsquedas en proyectos. El mismo permite buscar elementos tales como funciones o tipos de datos. El programa Exuberant C Tags agrega soporte para un gran número de lenguajes de programación y facilidades para obtener más información de los elementos a buscar.

5.5 VHDLspp

En ocasiones es necesario generar ciertas estructuras en forma automática evitando transcribirlas cada vez que se realiza un cambio. Un ejemplo es cuando deseamos inicializar un bloque de memoria con una imagen o el código que correrá nuestra CPU.

En lenguajes como el C cuando se desea hacer esto basta con colocar los datos en un *header* (archivo de encabezado) y utilizar la directiva #include del preprocesador de C. En VHDL no existe el concepto de preprocesador y por lo tanto esto no es posible.

Con esta finalidad se creó vhdlspp[11]. Este script Perl permite reemplazar *tags* (marcadores) del tipo @nombre_archivo@ por el contenido del archivo indicado. A diferencia de C esto puede colocarse en cualquier lugar de nuestro código VHDL.

5.6 hex2vhdl

Cuando creamos nuestro core compatible con el PIC16C84 nos encontramos en la necesidad de incluir los .hex que contenían el código del PIC en nuestros fuentes VHDL para inicializar la memoria de programa del controlador.

Para solucionar este problema creamos esta herramienta que trabajando en conjunto con vhdlspp y GNU Make nos permitieron que bastara con modificar el código de ensamblador para el PIC para que nuestro bitstream se actualizara conteniendo el programa para el microcontrolador.

hex2vhdl[12] genera la inicialización de un array VHDL conteniendo el código de programa. En FPGAs de Xilinx esta definición se sintetiza como una ROM (Block RAM con la escritura deshabilitada).

5.7 xtracth

Un problema muy frecuente que aparece cuando se diseña un periférico que será mapeado en una dirección de memoria o entrada/salida es que cuando se realizan cambios en el orden de sus registros se hace necesario actualizar los cambios en el código fuente que accede al periférico.

Para facilitar esta tarea desarrollamos xtracth[13]. Xtrach sólo necesita el nombre del *package* donde se definieron las constantes VHDL que definen los registros en cuestión. A partir del *package* xtrach extrae las constantes y las declara en un .inc y en un .h. Usando xtrach en conjunto con GNU Make se puede automatizar por completo esta tarea.

Cuando se utiliza el bus Wishbone xtrach se complementa con WISHBONE Builder que puede generar archivos .inc y .h con las direcciones base de los periféricos.

5.8 tpl2file

El ISE de Xilinx incluye más de 600 ejemplos y plantillas para VHDL.

Si deseamos tener acceso a este recurso desde fuera del ISE mientras editamos nuestros fuentes con cualquier editor de texto es posible extraerlos y colocarlos en una estructura de directorios. De esta manera se los puede navegar con cualquier herramienta para navegar directorios, el mismísimo diálogo de abrir archivo de nuestro editor por ejemplo.

Para esto hemos creado una pequeña utilidad llamada tpl2file[14]. Es un pequeño script Perl. Basta ejecutar el script en el directorio data del ISE para obtener un directorio llamado VHDL conteniendo todos los ejemplos y plantillas.

5.9 Librería de C en VHDL

Las facilidades de VHDL para impresión con formato son poco adecuadas cuando se escriben bancos de prueba complejos que deben generar información formateada para que otra aplicación la procese[15]. Para resolver esto y cuestiones similares Francis G. Wolff y Michael J. Knieser[16] crearon una biblioteca VHDL que emula la biblioteca estándar de C. La misma fue presentada en las conferencias del Grupo de Usuarios de Synopsys y liberada bajo la licencia GPL.

Nuestro grupo tomó esta biblioteca y armó un makefile adecuado para compilarla y testearla usando el GHDL.

6 Herramientas Relacionadas con la Simulación

En esta sección describimos las herramientas utilizadas para simular el código HDL así como también herramientas auxiliares que permiten generar bancos de pruebas, visualizar las formas de onda y automatizar el proceso.

6.1 Simulador

Se evaluaron varias alternativas teniendo siempre como objetivo el desarrollo en lenguaje VHDL para su aplicación principal en dispositivos FPGA. Dándole siempre prioridad a las alternativas bajo licencias de S.L. u open source, se evaluaron las siguientes herramientas: SAVANT[17], FreeHDL[18], Alliance[19] y GHDL[20]. Se seleccionó a GHDL que utiliza la tecnología del GCC[21], el compilador de S.L. más utilizado.

GHDL es un excelente simulador para VHDL. El mismo puede usarse para simular testbenches (bancos de prueba).

Es posible indicarle a GHDL que genere un archivo con las formas de onda para luego analizarlas en busca de algún problema.

GHDL soporta los estándares IEEE 1076-1987 y IEEE 1076-1993 con mucha más fidelidad que muchas herramientas comerciales.

6.2 Generador de Bancos de Prueba

Muchas veces la escritura de *testbenches* sencillos donde un patrón de entrada debe corresponderse con un patrón de salida puede resultar tediosa y monótona. Para estos casos se ha creado la herramienta llamada natebege[22], que genera automáticamente un *testbench* a partir de la descripción de los patrones de entrada y salida deseados.

La idea de natebege es que el usuario sólo necesita escribir una especie de tabla de verdad del componente y luego natebege genera automáticamente el banco de pruebas. Al ejecutar el banco de pruebas el mismo informará si alguna de las condiciones falló.

6.3 Visualizadores de formas de onda

Si bien no es posible encontrar errores en proyectos medianos o grandes usando este tipo de herramientas si es posible usarlas para entender cual es el problema una vez que hemos encontrado el error a través del uso de un buen *testbench*.
Existen varios visores de ondas libres uno muy útil es el GTKWave[23]. Las versiones actuales (1.3.69 al momento de esta publicación) no tienen soporte para los estados extendidos de VHDL (std_logic: L, H, W, U y -). Por esta razón es que creamos una versión modificada del mismo con este soporte. En versiones futuras estos cambios serán parte del programa oficial.

GTKWave permite visualizar las formas de onda de las señales usadas en nuestro proyecto.

6.4 Automatización del Proceso

Para ahorrar tiempo y evitar la repetición de tareas triviales el proyecto recomienda el uso de la herramienta GNU Make[24]. La misma permite invocar programas de acuerdo con reglas de dependencias entre los archivos y utilizando la fecha de modificación de los mismos para saber si es necesario volver a generarlos.

A través de las reglas se le puede indicar a GNU Make como crear un archivo a partir de otro.

La ventaja de GNU Make es que es altamente configurable. De esta manera se puede automatizar la generación del ejecutable usado para simular el proyecto como así también de otras cosas auxiliares.

En nuestro caso hemos creado reglas de manera tal que al modificar un código fuente en assembler la herramienta utiliza al ensamblador (gpasm) para generar un archivo en formato hexadecimal de Intel (.hex), luego convierte este archivo en un array VHDL usando hex2vhdl y a continuación incluye el mismo en la descripción de la memoria de programa del procesador usando vhdlspp, luego de esto analiza la descripción con GHDL y finalmente elabora el ejecutable con el simulador. No se trata de un proceso en lotes (batch) sino de reglas que permiten que sólo se procesen las partes necesarias debido a un cambio en una de ellas.

7 Herramientas de Síntesis

Lamentablemente no conocemos ningún S.L. que permita realizar esta operación para FPGA. Existe S.L. que permite llegar al diseño de chips (Alliance) pero no para FPGA.

Afortunadamente Xilinx (uno de los fabricantes más importantes de FPGA) ofrece un entorno de desarrollo completo gratuito y que corre tanto en Windows como en Linux. El ISE incluye herramientas de línea de comando que nos permiten automatizar este proceso.

La versión 6 del ISE WebPack se puede ejecutar utilizando Wine y la versión 7 del ISE WebPack está disponible para Linux. Utilizando las versiones de línea de comandos es

posible sintetizar nuestro proyecto y obtener un bitstream.

8 Herramientas Relacionadas con el Hardware

En esta sección describimos las herramientas utilizadas para transferir los resultados de la síntesis a la FPGA así como también para crear los circuitos impresos definitivos.

8.1 Hardware Para Grabación de la Configuración

Los dispositivos FPGA pueden poseer varios mecanismos para la programación de su configuración. Uno de los mecanismos que garantizan compatibilidad de hardware entre diferentes fabricantes y modelos es el estándar IEEE 1149.1 del JTAG (Joint Test Action Group). De esta forma un hardware de interfase para programación JTAG será útil para más de un fabricante de FPGA y memorias de configuración.

Basados en varias notas de aplicación de Xilinx y en circuitos publicados en la web, hemos desarrollado una interfase JTAG de muy bajo costo que se conecta al puerto paralelo. El circuito se ha desarrollado con una herramienta EDA de software libre llamada KICAD. Todos los archivos de diseño y la documentación asociada están disponibles y son de libre uso.

8.2 Software para Grabación de la Configuración

La transferencia de la configuración al hardware involucra dos tipos de conocimientos. Por un lado el protocolo de transferencia, que en nuestro caso es el JTAG, y por otro lado el conocimiento de las características específicas del dispositivo a usar.
La herramienta Impact de Xilinx resuelve ambos aspectos pero tiene dos desventajas muy importantes. El más importante es que su versión para GNU/Linux requiere del uso de una distribución en particular (Red Hat Enterpise) y de módulos del kernel propietarios. Por lo que se descartó su uso.

La herramienta xilinx_jtag[25] incluye ambos aspectos, pero se encuentra limitada a dispositivos específicos.

La herramienta GNU JTAG[26] sólo resuelve uno de los dos aspectos (el JTAG). Por lo que el proyecto desarrolló una herramienta que permite complementarla agregando la información necesaria para los dispositivos a usar. Esta parte de la herramienta denominada JBit[27] es configurable de manera tal que es posible agregar nuevos dispositivos.

8.3 Esquemático y PCB

KICAD[28] es una muy buena herramienta para generar esquemáticos y circuitos impresos (PCB). La funcionalidad de KICAD es comparable con la que se encontraba en los programas OrCAD y Autotrax hace unos años atrás. Si bien no posee algunas características de los programas más modernos y avanzados es posible realizar proyectos importantes.

Actualmente el proyecto FPGALibre se encuentra desarrollando una placa con Spartan II en encapsulado PQ208 utilizando KICAD. Cuando la misma haya sido probada y validada los circuitos esquemáticos e impresos estarán disponibles.

9 Conclusiones

- Se puede realizar el ciclo completo utilizando software libre y/o gratuito.
- Se puede realizar el ciclo completo trabajando sobre un sistema operativo libre (Ejemplo: Debian GNU/Linux).
- Se dispone de herramientas muy útiles y fácilmente automatizables.
- Es posible enseñar y desarrollar aplicaciones comerciales con las mismas herramientas y sin costos extras de licencias.
- Es posible enseñar los conceptos del desarrollo con FPGAs sin necesidad de comprometerse con un fabricante. Al verificar el código con GHDL nos aseguramos de que sea estándar y sólo los comandos de síntesis son específicos. Se aplica a las EDA tools y a las FPGAs.

Referencias

1. 1st Southern Conference on Programmable Logic - SPL 2005, Mar del Plata, 14-18 marzo 2005 http://www.ii.uam.es/~mcts/Frames/Proj_SCH_UAM_2005_IntroWS_Fset.htm
2. Source Forge.net http://www.sourceforge.net/
3. Sinander, P: VHDL Modelling Guidelines. European Space Agency. ASIC/001 Issue 1 September 1994. ftp://ftp.estec.esa.nl/pub/vhdl/doc/ModelGuide.pdf
4. Amitay, Y., Khatib, J., Lampret, D.: OpenCores Coding Guidelines. http://www.opencores.org/cvsget.cgi/common/opencores_coding_guidelines.pdf
5. Tropea, S.E.: FPGALibre guidelines. http://fpgalibre.sourceforge.net/vhdl.html#guidelines
6. Herveille, R.: WISHBONE System on Chip (SoC) Interconnection Architecture for Portable IP Cores. Revision B.3. September 7, 2002. http://prdownloads.sourceforge.net/fpgalibre/wbspec_b3-2.pdf?download.
7. Tropea, S. E., *et al*: http://setedit.sf.net/
8. Hiebert, D.: Exuberant CTAGS. http://ctags.sourceforge.net/
9. Tropea, S. E.: http://fpgalibre.sourceforge.net/vhdl.html#bakalint
10. Unneback, M., Tropea, S. E.: http://fpgalibre.sourceforge.net/vhdl.html#WBB
11. Tropea, S. E.: http://fpgalibre.sourceforge.net/vhdl.html#vhdlspp
12. Tropea, S. E.: http://fpgalibre.sourceforge.net/vhdl.html#hex2vhdl
13. Tropea, S. E.: http://fpgalibre.sourceforge.net/vhdl.html#xtracth
14. Tropea, S. E.: http://fpgalibre.sourceforge.net/editor.html#tpl2file
15. Tropea S. E., Borgna J. P. D.: Creación de bancos de prueba complejos usando Software Libre. SPL 2005, Mar del Plata. http://utic.inti.gov.ar/publicaciones/Bancos_de_prueba_usando_SL.pdf
16. Wolff, F. G., Knieser, M. J., *et al*: C/UNIX Functions for VHDL Testbenches. SNUG, San Jose, 2002. http://bear.ces.cwru.edu/vhdl/
17. SAVANT http://www.ececs.uc.edu/~paw/savant/
18. FreeHDL http://www.freehdl.seul.org/
19. Alliance http://www-asim.lip6.fr/recherche/alliance/
20. Gingold, T.: GHDL http://ghdl.free.fr/
21. GCC (GNU Compiler Collection) http://gcc.gnu.org/
22. Tropea, S.E.: http://fpgalibre.sourceforge.net/simul.html#natebege
23. Bybell, T.: http://home.nc.rr.com/gtkwave/
24. GNU Make: http://www.gnu.org/software/make/
25. Usselmann, R., Lamberts R.: http://fpgalibre.sourceforge.net/hard.html#xilinx_jtag
26. Telka, M., *et al*: http://openwince.sourceforge.net/jtag/
27. Borgna, J. P. D.: http://fpgalibre.sourceforge.net/hard.html#JBit
28. Charras, J-P.: http://www.lis.inpg.fr/realise_au_lis/kicad/