6º Jornadas de Desarrollo e Innovación Tecnológica | 2007
Instituto Nacional de Tecnología Industrial
ELECTRÓNICA
Transferencia de tecnología de apropiación colectiva

FPGALibre: Herramientas de Software Libre para diseño con *FPGAs*

Tropea, S. E.[(i)]; Brengi, D. J.[(i)]; Borgna, J. P. D.[(i)]; Gwirc, S. N.[(i)]

[(i)]INTI-Electrónica e Informática

Introducción

En este trabajo se presenta el proyecto FPGALibre[1]. Este proyecto fue creado con el fin de facilitar el desarrollo de diseños que involucran *FPGAs* (*Field Programmable Gate Arrays* = Arreglo de Compuertas Programable en Campo) mediante el uso de software libre. Esto abarca no sólo las herramientas necesarias para el desarrollo sino que también se 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.

El proyecto FPGALibre nace con la idea de poder compartir herramientas y *IP cores* desarrollados en nuestra Unidad Técnica. Durante el congreso SPL2005[2] 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[3] 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

Descripción y características del proyecto

Las *FPGAs* son circuitos integrados reconfigurables y son los miembros más avanzados de la familia de circuitos lógicos programables. Una *FPGA* está compuesta por lógica combinacional, registros y mecanismos de interconexión para unir estos dos últimos. Todos estos recursos son reconfigurables de manera tal que se puede modificar el funcionamiento para que se ajuste a nuestras necesidades. Estos elementos se complementan con celdas de entrada y salida que permiten conectar nuestro circuito con el exterior.

Las *FPGAs* presentan una alternativa muy interesante cuando es necesario desarrollar un circuito integrado a medida. Los circuitos integrados realizados a medida poseen costos muy elevados y sólo se justifican cuando el volumen de producción supera las decenas de miles. Esto es válido aún para los *ASIC* (*Application Specific Integrated Circuit* = Circuito Integrado de Aplicación Específica) que utilizan técnicas de fabricación de las conocidas como *semi-custom* en las que sólo una parte del proceso es específica del cliente y el resto son estándares. Cuando los volúmenes de producción son pequeños las *FPGAs* aparecen como una excelente alternativa. Las mismas no poseen grandes gastos iniciales o del tipo *NRE* (*Non-Recurring Engineering*) como en el caso de los *ASICs*. Por otra parte si se detectara un error en el diseño, o fuera necesario introducir otro tipo de cambio o mejora, las *FPGAs* pueden reconfigurarse sin ser necesario reemplazar el circuito integrado. Como contrapartida el costo por unidad de las *FPGAs* es superior, su velocidad es inferior y el consumo de energía es mayor cuando se las compara con los *ASICs*.

Dentro de una *FPGA* se puede incluir la funcionalidad de varios circuitos integrados. Esta funcionalidad puede ser desarrollada por el mismo equipo de trabajo o adquirida a través de un tercero. Debido a que estas funcionalidades son como componentes electrónicos, pero sin su parte física, se los suele llamar componentes virtuales. En la industria se los conoce como bloques de propiedad intelectual o *IP cores*.

El desarrollo de un *IP core* para una *FPGA* es muy similar al de un *ASIC*. En el caso de la *FPGA* el proceso se encuentra simplificado gracias a que nuestro circuito utilizará recursos ya probados y algo sobredimensionados. Este diseño se realiza utilizando lo que se conoce como lenguajes de descripción de hardware. Los mismos tienen cierto parecido con los lenguajes de programación de computadoras pero poseen diferencias conceptuales muy importantes. No se trata de programar un dispositivo sino de describir el

comportamiento del mismo. Luego la descripción se convierte en una configuración para la *FPGA* utilizando herramientas de síntesis.

Cuando nos referimos a S. L. hablamos de software que otorga libertades a quien lo usa. Un claro ejemplo de S. L. es el creado por el proyecto GNU[4] y que se encuentra distribuido bajo la licencia GPL[5].

S. L. se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. De modo más preciso, se refiere a cuatro libertades de los usuarios del software:

—La libertad de usar el programa, con cualquier propósito (libertad 0).

—La libertad de estudiar el funcionamiento del programa, y adaptarlo a las necesidades (libertad 1). El acceso al código fuente es una condición previa para esto.

—La libertad de distribuir copias, con lo que puede ayudar a otros (libertad 2).

—La libertad de mejorar el programa y hacer públicas las mejoras, de modo que toda la comunidad se beneficie (libertad 3). De igual forma que la libertad 1 el acceso al código fuente es un requisito previo.

Debido a su extenso uso en instituciones gubernamentales y universitarias el proyecto tomó como lenguaje preferencial el *VHDL* (*Very high speed integrated circuit Hardware Description Language* = Lenguaje de Descripción de Hardware para Circuitos Integrados de Muy Alta Velocidad). 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.

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[6] y recomendaciones para proyectos en OpenCores[7]. En base estas dos recomendaciones, el proyecto FPGALibre define sus recomendaciones, también llamadas *guidelines*[8], para escritura de código *VHDL*.

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[9] para *IP cores*. 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.

Para la implementación de una descripción de hardware en lenguaje *VHDL* se utiliza un editor de texto. No son muchos los editores libres que poseen facilidades avanzadas para la edición de código *VHDL*. Uno de estos editores es el SETEdit[10], 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[11]) con soporte específico para *VHDL*.

—Indentado coherente con los *guidelines* del proyecto.

Para facilitar el cumplimiento de los *guidelines* del proyecto se desarrolló un pequeño *lint* llamado bakalint[12]. 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 los cumpla.

Cuando se utiliza el estándar 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[13]. La misma fue adaptada para cumplir con lineamientos del proyecto y para resolver algunos problemas que se encontraron en la versión original.

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. 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.

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[14]. 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*.

Cuando creamos nuestro *IP core* compatible con el PIC16C84[15][16][17] 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[18] 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).

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[19]. Xtracth sólo necesita el nombre del *package* donde se definieron las constantes *VHDL* que definen los registros en cuestión. A partir del *package* xtracth extrae las constantes y las declara en un .inc y en un .h. Usando xtracth en conjunto con GNU Make se puede automatizar por completo esta tarea. Cuando se utiliza el estándar de interconexión WISHBONE xtracth se complementa con WISHBONE Builder que permite generar archivos .inc y .h con las direcciones base de los periféricos.

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[20]. 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.

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[21]. Para resolver esto y cuestiones similares Francis G. Wolff y Michael J. Knieser[22] 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.

Una vez terminada la descripción de hardware se procede a la primer verificación que se lleva a cabo utilizando un 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[23], FreeHDL[24], Alliance[25] y GHDL[26]. Se seleccionó a GHDL que utiliza la tecnología del GCC[27], 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 e IEEE 1076-1993 con mucha más fidelidad que muchas herramientas comerciales.

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[28], 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.

Una vez simulado nuestro modelo es probable que encontremos errores. Una de las herramientas que permiten ayudarnos a encontrar estos errores son los visores de forma de onda que permiten analizar el estado de todas las señales de nuestra descripción. 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[29].

Para ahorrar tiempo y evitar la repetición de tareas triviales el proyecto recomienda el uso de la herramienta GNU Make[30]. 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.

Terminada la tarea de simulación y depuración se procede a convertir nuestra descripción de hardware en una configuración adecuada para la *FPGA* (*bitstream*). Esta tarea se lleva a cabo con un sintetizador. Lamentablemente no conocemos ningún S.L. que permita realizar esta operación para *FPGAs*. Existe S.L. que permite llegar al diseño de chips (Alliance) pero no es útil para *FPGAs*. Afortunadamente *Xilinx,* uno de los fabricantes más importantes de *FPGAs,* 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. Las versiones posteriores a la 7 del ISE WebPack están disponibles para Linux. Utilizando las aplicaciones de línea de comandos es posible sintetizar nuestro proyecto y obtener un *bitstream*.

El *bitstream* obtenido luego de la síntesis debe transferirse a la *FPGA* o a la memoria de 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 (*Electronic Design Automation*) de software libre llamada KICAD. Todos los archivos de diseño y la documentación asociada están disponibles y son de libre uso.

KICAD[31][32] 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.

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 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[33] incluye ambos aspectos, pero se encuentra limitada a dispositivos específicos. La herramienta GNU JTAG[34] 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[35] es configurable de manera tal que es posible agregar nuevos dispositivos.

El componente final en esta cadena de desarrollo es el circuito impreso que contiene la *FPGA* a utilizar. El proyecto FPGALibre desarrolló una placa[36][37] con *Spartan* II[38] en encapsulado PQ208 utilizando KICAD. Sus circuitos esquemáticos e impresos se encuentran disponibles.

Conclusiones

—Se puede realizar el ciclo completo de diseño con dispositivos *FPGA*, utilizando software libre y/o gratuito y trabajando sobre un sistema operativo libre, como por ejemplo Debian GNU/Linux.

—Se dispone actualmente en el laboratorio de herramientas muy útiles y fácilmente automatizables que han permitido realizar importantes desarrollos de *IP cores*, tanto para uso interno como para terceros.

—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 que el mismo cumpla con el estándar *VHDL* y sólo los comandos de síntesis sean específicos de un fabricante. Esto se aplica tanto a las *EDA tools* como a las *FPGAs*.

Referencias

[1] S. E. Tropea, D. J. Brengi, J. P. D. Borgna, "FPGALibre: Herramientas de Software Libre para diseño con *FPGAs*", *FPGA* Based Systems, ISBN 84-609-8998-4, pp 173-180, 2006.
[2] 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
[3] Source Forge.net http://www.sourceforge.net/
[4] GNU project, http://www.gnu.org/
[5] General Public License, http://www.gnu.org/copyleft/gpl.html
[6] P. Sinander, "*VHDL* Modelling Guidelines. European Space Agency", *ASIC*/001 Issue 1 September 1994. ftp://ftp.estec.esa.nl/pub/vhdl/doc/ModelGuide.pdf
[7] Y. Amitay, J. Khatib, D. Lampret, "OpenCores Coding Guidelines" http://www.opencores.org/cvsget.cgi/common/opencores_coding_guidelines.pdf
[8] S. E. Tropea, "FPGALibre guidelines" http://fpgalibre.sourceforge.net/vhdl.html#guidelines
[9] Silicore and OpenCores.Org, "WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores", http://prdownloads.sourceforge.net/fpgalibre/wbspec_b3-2.pdf?download
[10] S. E. Tropea y otros, "SETEdit, un editor de texto amigable", http://setedit.sourceforge.net.
[11] D. Hiebert,"Exuberant CTAGS", http://ctags.sourceforge.net/
[12] S. E. Tropea, "BakaLint", http://fpgalibre.sourceforge.net/vhdl.html#bakalint
[13] M. Unneback, S. E. Tropea, "WISHBONE Builder", http://fpgalibre.sourceforge.net/vhdl.html#WBB
[14] S. E. Tropea, "*VHDL* Simple PreProcessor", http://fpgalibre.sourceforge.net/vhdl.html#vhdlspp
[15] Microchip Technology Inc., "8 bits CMOS EEPROM Microcontroller", p 8 (http://ww1.microchip.com/downloads/en/devicedoc/30445c.pdf, verificado 27/06/2007).
[16] S. E. Tropea, J. P. D. Borgna, "Microcontrolador compatible con PIC16C84, bus Wishbone y video", *FPGA* Based Systems, ISBN 84-609-8998-4, 2006.
[17] S. E. Tropea, "Microntrolador compatible con *PIC* 16C84, descripción de hardware", 6° Jornadas de Desarrollo e Innovación Tecnológica, Instituto Nacional de Tecnología Industrial, Argentina, 2007.
[18] S. E. Tropea, ".HEX to *VHDL* converter", http://fpgalibre.sourceforge.net/vhdl.html#hex2vhdl
[19] S. E. Tropea, "eXTract Headers", http://fpgalibre.sourceforge.net/vhdl.html#xtracth
[20] S. E. Tropea, "TemPLate to FILEs converter", http://fpgalibre.sourceforge.net/editor.html#tpl2file
[21] S. E. Tropea, J. P. D. Borgna, "Creación de bancos de prueba complejos usando Software Libre", 1st Southern Conference on Programmable Logic - SPL 2005, Mar del Plata, 2005 (http://utic.inti.gov.ar/publicaciones/Bancos_de_prueba_usando_SL.pdf)
[22] F. G. Wolff, M. J. Knieser y otros "C/UNIX Functions for *VHDL* Testbenches", SNUG, San Jose, 2002. (http://bear.ces.cwru.edu/vhdl/)
[23] SAVANT, http://www.ececs.uc.edu/~paw/savant/
[24] FreeHDL, http://www.freehdl.seul.org/
[25] Alliance, http://www-asim.lip6.fr/recherche/alliance/
[26] T. Gingold, "GHDL", http://ghdl.free.fr/
[27] "GNU Compiler Collection", http://gcc.gnu.org/
[28] S. E. Tropea, "NAive TEstBEnch GEnerator", http://fpgalibre.sourceforge.net/simul.html#natebege
[29] T. Bybell, "GTKWave", http://home.nc.rr.com/gtkwave/
[30] GNU Make, http://www.gnu.org/software/make/
[31] J-P Charras, "KICAD", http://www.lis.inpg.fr/realise_au_lis/kicad/
[32] D. J. Brengi, S. E. Tropea, "Experiencia de migración hacia herramienta de diseño de circuitos impresos de software libre", 6° Jornadas de Desarrollo e Innovación Tecnológica, Instituto Nacional de Tecnología Industrial, Argentina, 2007.
[33] R. Usselmann, R. Lamberts, "*Xilinx* JTAG", http://fpgalibre.sourceforge.net/hard.html#xilinx_jtag
[34] M. Telka y otros, "GNU JTAG", http://openwince.sourceforge.net/jtag/
[35] J. P. D. Borgna, S. E. Tropea, "JBit", http://fpgalibre.sourceforge.net/hard.html#JBit

[36] D. J. Brengi, S. E. Tropea, J. P. D. Borgna, "Tarjeta de Diseño Abierto para Desarrollo y Educación", III Southern Conference on Programmable Logic - SPL 2007, Designer Forum Proceedings (ISBN 978-84-611-4716-8), pp 57-60, 2007.
[37] J. P. D. Borgna, D. J. Brengi, S. E. Tropea, "Tarjeta de aplicaciones generales con *FPGA*", 6° Jornadas de Desarrollo e Innovación Tecnológica, Instituto Nacional de Tecnología Industrial, Argentina, 2007.
[38] *Xilinx Spartan* II *FPGA*, http://www.xilinx.com/products/silicon_solutions/fpgas/spartan_series/spartan2_fpgas/index.htm

Para mayor información contactarse con:

Ing. Salvador E. Tropea - salvador@inti.gov.ar