MICROCONTROLADOR COMPATIBLE CON AVR, INTERFAZ DE DEPURACIÓN Y BUS WISHBONE

Tropea S. E., Caruso D. M.

Electrónica e Informática
Instituto Nacional de Tecnología Industrial
Buenos Aires, Argentina
email: salvador@inti.gob.ar

ABSTRACT

En este trabajo presentamos un microcontrolador compatible con la línea AVR de Atmel. Esta implementación puede ser configurada para ser compatible con los AVR de segunda (ej. ATtiny22), tercera (ej. ATmega103) y cuarta (ej. ATmega8) generación.

Este desarrollo incluye los siguientes periféricos compatibles: controlador de interrupciones, puertos de entrada y salida, temporizadores y contadores, UART y *watchdog*.

Para adaptarlo a distintas necesidades se incluyó una interfaz de expansión que utiliza el estándar de interconexión WISHBONE.

A los fines de facilitar el desarrollo de aplicaciones sobre esta plataforma se lo dotó de una unidad de depuración y se adaptó el software necesario para poder realizar depuración de alto nivel con una interfaz de usuario simple e intuitiva.

El diseño fue verificado usando simuladores y FPGAs de Xilinx (*Spartan* II y 3A).

1. INTRODUCCIÓN

Nuestro laboratorio realiza aplicaciones con microcontroladores embebidos de manera frecuente. Es por esta razón que al abordar la tecnología FPGA se deseó implementar microcontroladores compatibles con los usados en dichos desarrollos. De esta manera personas del equipo que no se dedicaran a esta nueva tecnología podrían participar en desarrollos con FPGAs.

Con esta finalidad se encaró el desarrollo de un microcontrolador compatible con el PIC 16C84 [1] y posteriormente se añadió una interfaz de depuración para el mismo [2]. Este microcontrolador mostró ser útil y fue transferido a la industria aero-espacial [3]. Sin embargo su arquitectura no es buena para la programación en lenguaje C. En los últimos años nuestro laboratorio se volcó al uso de la línea AVR de Atmel, que si fue diseñada para ser programada en lenguaje C. Por lo que se decidió encarar el desarrollo de un equivalente para FPGAs.

En este trabajo presentamos las características del microcontrolador desarrollado y de una interfaz de depuración que permite depurar programas escritos en lenguaje C desde una PC y utilizando un software simple e intuitivo.

2. MICROCONTROLADOR

2.1. Arquitectura

El AVR es un microcontrolador del tipo RISC de 8 bits con dos espacios de memoria completamente independientes: memoria de programa y memoria de datos.

En la memoria de programas se encuentra el código a ejecutar. Es una memoria de 16 bits y la mayor parte de las instrucciones son de este tamaño. Algunas instrucciones necesitan dos posiciones de memoria (32 bits).

La memoria de datos es de 8 bits y se divide en tres secciones. Existen instrucciones específicas para acceder a cada una de estas secciones de memoria, pero también hay instrucciones que pueden acceder a todo el espacio de memoria indistintamente. La parte más baja de esta memoria alberga 32 registros de 8 bits, seis de los cuales pueden agruparse de a pares para formar tres registros de 16 bits, usualmente usados como punteros. A continuación se encuentra el espacio de entrada/salida, con un total de 64 posiciones de 8 bits. El resto de la memoria es RAM.

A partir de la segunda generación la línea AVR implementa un *stack pointer*. El mismo es decrementado cada vez que se guarda algo en la pila, usualmente se lo inicializa con la dirección más alta de memoria RAM. Este recurso facilita el compilado de aplicaciones en lenguaje C.

La mayor parte de las instrucciones se ejecutan en un ciclo de reloj, pero algunas pueden tomar hasta cuatro ciclos, como en el caso del CALL.

El *stack pointer* y el registro de estados se encuentran mapeados en el espacio de memoria de entrada y salida.

La ALU implementa las operaciones básicas de suma,

resta y desplazamiento. A partir de la cuarta generación se introduce la multiplicación de enteros con o sin signo y en formato de punto fijo (1.7).

2.2. Implementación

Las herramientas de desarrollo utilizadas fueron las recomendadas por el proyecto FPGALibre [4] [5]. Para este desarrollo se utilizaron estaciones de trabajo que corren Debian [6] GNU [7] /Linux. Siendo el lenguaje de descripción de *hardware* utilizado el VHDL.

A los fines de simplificar la tarea y comenzar por una base de código funcional se decidió basar el desarrollo en el proyecto AVR Core [8] de OpenCores.org. Primeramente se adaptó el código a los lineamientos del proyecto FPGALibre. El código VHDL original se encuentra escrito a muy bajo nivel, sin explotar toda la expresividad del lenguaje, por lo que se decidió reescribirlo para lograr un código más compacto y fácil de mantener. También se unificaron módulos que estaban separados, como la ALU y el procesador de operaciones orientadas a bits.

El proyecto original sólo implementa la tercer generación de AVR, en particular trata de modelar el ATmega103. Para lograr mayor flexibilidad se decidió hacer parametrizable a la CPU permitiendo seleccionar entre tres posibles generaciones. De esta manera es posible aprovechar mejor el área de FPGA seleccionando entre tres versiones diferentes de acuerdo a la complejidad del proyecto en cuestión.

Las principales diferencias entre la segunda y la tercera generación son el tamaño del *stack pointer* (8 bits en la segunda y 16 en las posteriores) y la falta de las instrucciones de salto absoluto (JMP y CALL). La cuarta generación agrega multiplicación, movimiento entre pares de registros (16 bits) y mejora la instrucción LPM (acceso a la memoria de programa).

2.3. Periféricos

A los fines de facilitar el uso de aplicaciones ya existentes para la línea AVR se decidió implementar algunos de los periféricos más comunes.

Controlador de interrupciones: el mismo permite configurar, enmascarar, etc. las interrupciones externas. Se realizaron dos implementaciones diferentes, una compatible con las líneas modernas (ej. ATtiny22 y ATmega8) y otro con las anteriores (ej. ATmega103).

Puertos de entrada y salida: permiten configurar pines como entrada o salida. Se realizó una implementación nueva y flexible que pudiera modelar todos los casos posibles.

Temporizador y contadores: se adaptó la implementación del proyecto original. La misma implementa los Timers 0 y 2 del ATmega103. Se modificó el código para permitir un mayor reuso ya que ambos timers son muy similares. Estos timers son de 8 bits y permiten su uso como contadores o generadores de PWM.

USART: se adaptó la implementación del proyecto original. La misma es muy flexible y permite la selección de distintas tasas de transmisión y largo de datos.

Watchdog: este periférico no se encontraba en la implementación original y debido a que se encuentra relacionado con una instrucción de la CPU se decidió implementarlo.

2.4. Bus de Expansión

En un microcontrolador los periféricos disponibles son fijos, pero en el caso de una implementación en una FPGA es deseable que los mismos puedan agregarse y/o quitarse fácilmente. Por esta razón se decidió implementar un bus de expansión.

Siguiendo los mismos criterios adoptados en el pasado [1] se seleccionó el estándar de interconexión WISHBONE [9]. El mismo posee las siguientes ventajas:

- Para casos simples (un maestro y uno o más esclavos) se reduce a poca o ninguna lógica adicional.
- Fue pensado para casos más complejos (más de un maestro, reintento, notificación de error, etc.).
- No posee *royalties* y puede ser usado sin costo alguno. La especificación completa se encuentra disponible en internet.

Para acceder al bus WISHBONE se implementaron dos registros en el espacio de entrada y salida. Debido a que nuestra implementación dejó de lado la memoria EEPROM de los AVR disponíamos de las direcciones 0x1C a 0x1F. Se decidió utilizar las direcciones 0x1E y 0x1F. El registro 0x1F se usa para indicar la dirección del periférico que deseamos utilizar en el bus WISHBONE. Posteriormente cualquier operación sobre el registro 0x1E se transfiere a través del bus WISHBONE. Esto permite acceder a hasta 256 registros de 8 bits en el bus WISHBONE.

El bus WISHBONE contempla la conexión de periféricos lentos por lo que si un periférico necesita más de un ciclo de reloj para realizar la operación puede detener al microcontrolador hasta que la misma haya concluido.

2.5. Configuraciones Equivalentes

A los fines de proveer al usuario final de configuraciones que sean similares a microcontroladores ya existentes se implementaron tres microcontroladores, uno por cada generación implementada:

- **ATtiny22**: de segunda generación, un único puerto de entrada y salida de 5 bits, un timer de 8 bits, bus WISHBONE, *stack pointer* de 8 bits, *watchdog*, una fuente de interrupción externa, 128 bytes de memoria RAM, 1024 words de memoria de programa.

- **ATmega103**: de tercera generación, 6 puertos de entrada y salida, UART, bus WISHBONE, dos timers de 8 bits, *stack pointer* de 16 bits, *watchdog*, ocho fuentes de interrupción externa, 4096 bytes de memoria RAM, hasta 65536 words de memoria de programa.

- **ATmega8**: de cuarta generación, 3 puertos de entrada y salida, UART, bus WISHBONE, dos timers de 8 bits, *stack pointer* de 16 bits, *watchdog*, dos fuentes de interrupción externa, 1024 bytes de memoria RAM, hasta 65536 words de memoria de programa.

Todas las configuraciones son parametrizables, pudiéndose eliminar los periféricos no utilizados.

3. HERRAMIENTAS DE DESARROLLO

Debido a que esta implementación incluye todo el set de instrucciones del procesador original, y a que se implementaron tres configuraciones equivalentes a microcontroladores existentes, es posible utilizar la mayor parte de las herramientas disponibles para la línea AVR.

Existen herramientas de software libre de muy buena calidad disponibles para la línea AVR y que pueden utilizarse con nuestro microcontrolador.

3.1. Ensamblador

Para compilar fuentes en *assembler* escritos para el ensamblador de Atmel (avrasm) es posible utilizar avra [10]. Se distribuye bajo licencia GPL y se puede compilar para las plataformas más populares. Además de ser compatible con el ensamblador de Atmel ofrece un soporte de macros mejorado y ensamblado condicional.

3.2. Compilador de C/C++

A pesar de tratarse de una plataforma de 8 bits es posible obtener una versión del compilador de C del proyecto GNU [11] capaz de generar código para el AVR. A esta versión del gcc se la conoce como gcc-avr y es capaz de generar código altamente optimizado para AVRs de segunda a quinta generación.

3.3. Biblioteca Estándar de C

Una implementación muy completa de la biblioteca estándar de C especialmente diseñada para los AVR se encuentra disponible en el proyecto avr-libc [12]. Esta implementación se encuentra especialmente optimizada para los AVRs y permite realizar aplicaciones flexibles y compactas.

Uno de los detalles interesantes de esta biblioteca es que es posible redireccionar la entrada y salida estándar (stdin y stdout) a cualquier dispositivo, por ejemplo el puerto serie.

3.4. Depurador

El depurador del proyecto GNU, gdb [13], puede compilarse con soporte para AVR. A esta versión del gdb se la conoce como avr-gdb. El mismo es capaz de depurar programas escritos para los AVR. Debido a que gdb es una aplicación enorme la misma corre en una PC comunicándose con un simulador de AVR o bien con el microcontrolador.

GDB es una aplicación de línea de comandos, pero existen varias interfaces de usuario disponibles para hacer más simple su uso.

3.5. Simulador

Un simulador capaz de simular el comportamiento de un AVR es el simulavr [14]. El mismo corre en una PC y es posible utilizarlo desde el avr-gdb para realizar una depuración del código simulado sin necesidad de disponer de un AVR real.

4. INTERFAZ DE DEPURACIÓN

4.1. Introducción

El desarrollo de sistemas electrónicos basados en sistemas embebidos presenta un desafío a la hora de eliminar errores de implementación. Esto se debe a que los microcontroladores utilizados para estas tareas suelen ser pequeños y no es posible que ejecuten las complejas tareas involucradas en la depuración de errores. La tarea se dificulta aún más cuando dichos dispositivos no poseen una interfaz de usuario amistosa, sin salida de vídeo donde conectar un monitor ni entradas de teclado o similares para ingresar datos.

Para solucionar estos problemas se suele incluir funcionalidad en el microcontrolador que permite realizar las tareas de depuración en forma remota utilizando una computadora personal, donde se encuentran disponibles los recursos antes mencionados.

4.2. Selección de la Arquitectura

Los dispositivos modernos de la línea AVR poseen facilidad de depuración a través de un puerto JTAG (Joint Test Action Group). Una posible solución a este problema hubiera sido implementar una interfaz compatible. La ventaja de esta solución es que se podría haber utilizado cualquier software ya disponible, sin modificaciones. La desventaja es que las PCs no poseen interfaz JTAG, esto implica un cable especial. Este no es el único problema, en la práctica lo que soportan los programas no es el manejo directo de JTAG sino un protocolo especial que normalmente se implementa utilizando un microcontrolador. Así, estos cables en realidad implican el uso de un microcontrolador que es el que realmente se comunica por JTAG con el microcontrolador

que deseamos depurar. Una solución posible era implementar esta segunda CPU en la misma FPGA.

Por otro lado era necesario implementar una unidad de depuración compatible con la del AVR y someterse a sus limitaciones. Nuestro equipo ya poseía experiencia en el desarrollo de una unidad de depuración [2], más flexible que la de los AVR. Por lo que se decidió adaptar nuestra unidad de depuración y evitar la necesidad de un segundo microcontrolador. La desventaja de este mecanismo es que es necesario adaptar el software para que funcione con nuestro microcontrolador.

4.3. Comunicación con la PC

Nuestra unidad de depuración original es un periférico que soporta el estándar de interconexión WISHBONE. Para controlar este tipo de periféricos es necesario poder acceder al bus WISHBONE, que se encuentra dentro de la FPGA. Una forma de lograr este acceso es utilizando algún tipo de puente. En nuestro trabajo anterior usamos un puente de puerto paralelo en modo EPP a WISHBONE [15], desarrollado por nuestro equipo. En este caso y debido a que el puerto paralelo ha sido reemplazado casi por completo por el USB optamos por utilizar un puente de USB a WISHBONE [16] [17], también desarrollado por nuestro equipo.

4.4. Características

Nuestra unidad de depuración permite:

- Detener/Reanudar la ejecución del microcontrolador en cualquier momento.
- Ejecutar su programa paso a paso.
- Detener la ejecución cuando se alcanzó una posición de memoria determinada, punto de parada o *breakpoint*. La cantidad de *breakpoints* es configurable entre 1 y 256.
- Reinicializar el microcontrolador.
- Acceder a todos los registros, incluyendo el contador de programa.
- Acceder al espacio de entrada y salida.
- Inspeccionar la pila de llamadas (*calling stack*).
- Detener la ejecución cuando se accede a una posición de memoria de datos, *watchpoint*. Los accesos pueden seleccionarse para detenerse por lectura, escritura o ambos. El número de *watchpoints* es configurable entre 1 y 256.
- Alterar la memoria de programa.
- Detectar desbordes en la pila y detener la ejecución cuando esto sucede.

4.5. Software complementario

Para poder depurar programas corriendo en microcontroladores AVR es posible utilizar el avr-gdb, pero este programa necesita comunicarse con otro programa que es el que realmente controla al microcontrolador. Un programa muy usado es el AVaRICE [18] y al ser software libre fue posible modificarlo. Se agregó soporte al AVaRICE para controlar nuestra unidad de depuración utilizando el puerto USB.

Como interfaz de usuario para controlar al avr-gdb se seleccionó el programa SETEdit [19]. El mismo es el entorno de trabajo recomendado por el proyecto FPGALibre. SETEdit implementa el protocolo GDB/MI para la depuración de programas en C/C++ y *assembler* por lo que no fueron necesarios cambios importantes para lograr que el mismo se adaptara a la depuración de programas escritos en lenguaje C o *assembler* corriendo en dicho microcontrolador.

4.6. Hardware complementario

El microcontrolador en cuestión no implementa la funcionalidad de reprogramación incluida en el original. Esto no es un problema importante debido a que las FPGAs son reconfigurables y por lo tanto basta con volver a sintetizar el diseño para modificar el programa ejecutado por el microcontrolador. Es común que los depuradores remotos puedan modificar el programa ejecutado por el sistema embebido, por lo que para complementar este desarrollo se diseñó un periférico WISHBONE capaz de acceder a la memoria de programa del microcontrolador. Esto permitió reconfigurar dicha memoria sin necesidad de reconfigurar toda la FPGA.

La Fig. 1 ilustra la interconexión entre los distintos componentes de *hardware* antes mencionados, los bloques ilustrados con relleno sólido corresponden a los desarrollos descriptos en este trabajo. En la Fig. 2 se muestra el flujo de datos dentro de la computadora, los datos ingresan a través del puerto USB, son tomados por el sistema operativo (Linux) utilizando operaciones básicas de entrada/salida y enviados al espacio de usuario a través de la biblioteca *libusb*, estos datos son procesados por AVaRICE y traducidos al protocolo remoto de gdb y finalmente el depurador (*avr-gdb*) los envía a la interfaz de usuario (SETEdit) utilizando el protocolo GDB/MI.

5. RESULTADOS

La Fig. 3 muestra una sesión de depuración utilizando SETEdit. En la misma se observa el código fuente del programa, el código desensamblado y una ventana utilizada para monitorizar el valor de una variable.

Este desarrollo fue verificado utilizando FPGAs *Spartan* II y *Spartan* 3A de Xilinx. Para la síntesis se utilizó el programa XST 10.1.02 K.37.

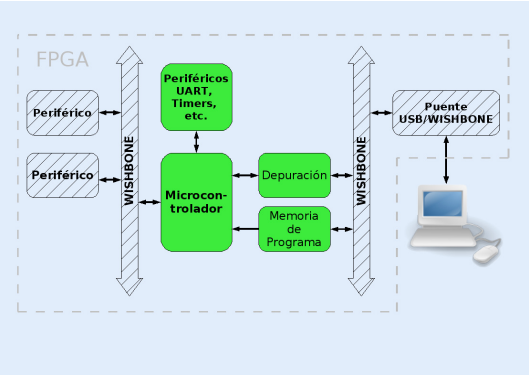

Fig. 1. Diagrama de conexiones de los bloques de hardware.

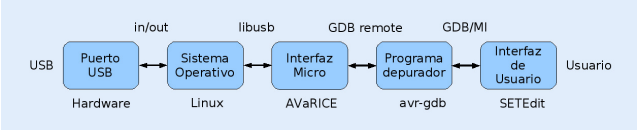

Fig. 2. Flujo de datos dentro de la computadora.

El área ocupada depende de varios parámetros, a continuación se describen algunos ejemplos.

- Configuración ATmega103 con memoria de programa de 1024x16, sin periféricos internos y sólo una UART pequeña en el bus WISHBONE: 644 *slices* (245 *FFs* y 1124 *LUTs*) 3 *BRAMs*.
- Configuración ATmega8 con memoria de programa de 1024x16, sin periféricos internos y sólo una UART pequeña en el bus WISHBONE: 707 *slices* (275 *FFs* y 1224 *LUTs*) 2 *BRAMs* y un multiplicador.
- Configuración ATtiny22 con memoria de programa de 1024x16, sin periféricos internos y sólo una UART pequeña en el bus WISHBONE: 606 *slices* (237 *FFs* y 1053 *LUTs*) 2 *BRAMs*.
- Configuración ATmega8 con memoria de programa de 1024x16, puerto B habilitado, una UART pequeña en el bus WISHBONE, interfaz de depuración con 3 breakpoints y 3 watchpoints (USB): 1548 *slices* (902 *FFs* y 2477 *LUTs*) 4 *BRAMs* y un multiplicador. Aproximadamente 500 de los *slices* son necesarios para la implementación del puente de USB a WISHBONE.

Sólo en el último caso se le pidió a la herramienta que buscara cumplir con una frecuencia de trabajo fijada de 24 MHz para todo el circuito, salvo para el PHY de USB que debía correr a 48 MHz. En el resto de los casos no se pidió ningún requisito en particular y la herramienta reporto frecuencias de trabajo de entre 30 y 37 MHz para una *Spartan 3A* grado 4.

5.1. Control de Motores de DC

Con la finalidad de verificar el correcto funcionamiento del microcontrolador y la capacidad de la unidad de depuración, se encaró el desarrollo de un control de posición para motores de corriente continua. Para dicho control se decidió implementar un PID. Dicho desarrollo utiliza los siguientes periféricos conectados al bus WISHBONE:

- Modulador PWM de 15 bits de resolución.
- Decodificador de encoder relativo con 16 bits de resolución.
- UART pequeña trabajando a 115200 baudios.

La configuración usada es compatible con el ATmega8 con una memoria de programa de 4096x16 y la unidad de depuración habilitada. De los periféricos internos sólo el puerto B se encuentra habilitado. Dicha configuración insumió 1732 *slices* (1009 *FFs* y 2786 *LUTs*) 7 *BRAMs* y un multiplicador (*Spartan 3A*).

Habiéndose ya obtenido los primeros resultados exitosos queda por refinar el mecanismo de determinación de las constantes del PID.

6. CONCLUSIONES

La elección de la arquitectura AVR permitió contar con una amplia gama de herramientas y bibliotecas. Al mismo tiempo se comprobó que programar este dispositivo es tan fácil como programar un AVR comercial.

La unidad de depuración obtenida es poderosa, capaz de realizar la mayor parte de las operaciones realizadas por depuradores utilizados en computadoras personales, y que es de gran ayuda a la hora de buscar errores en sistemas funcionando en tiempo real.

El uso de FPGAs posee como ventaja el hecho de que la unidad de depuración puede removerse en la versión definitiva del diseño con lo que el mismo no ocupa recursos en el dispositivo final. En el caso en que el diseño ocupe prácticamente el total de la FPGA basta con usar una FPGA más grande durante la etapa de desarrollo, a los fines de incluir unidades de depuración como esta.

La selección del estándar de interconexión WISHBONE permitió el reuso de un puente de puerto USB y abre la posibilidad a la implementación de otro tipo de mecanismos de comunicación, como podrían ser RS-232 o Ethernet.

La elección de modificar un programa como AVaRICE permitió acelerar notablemente el desarrollo y reusar interfaces de usuario ya existentes y con las cuales nuestro equipo ya se encontraba familiarizado.

La utilización de las herramientas propuestas por el proyecto FPGALibre mostró ser adecuada para este desarrollo.

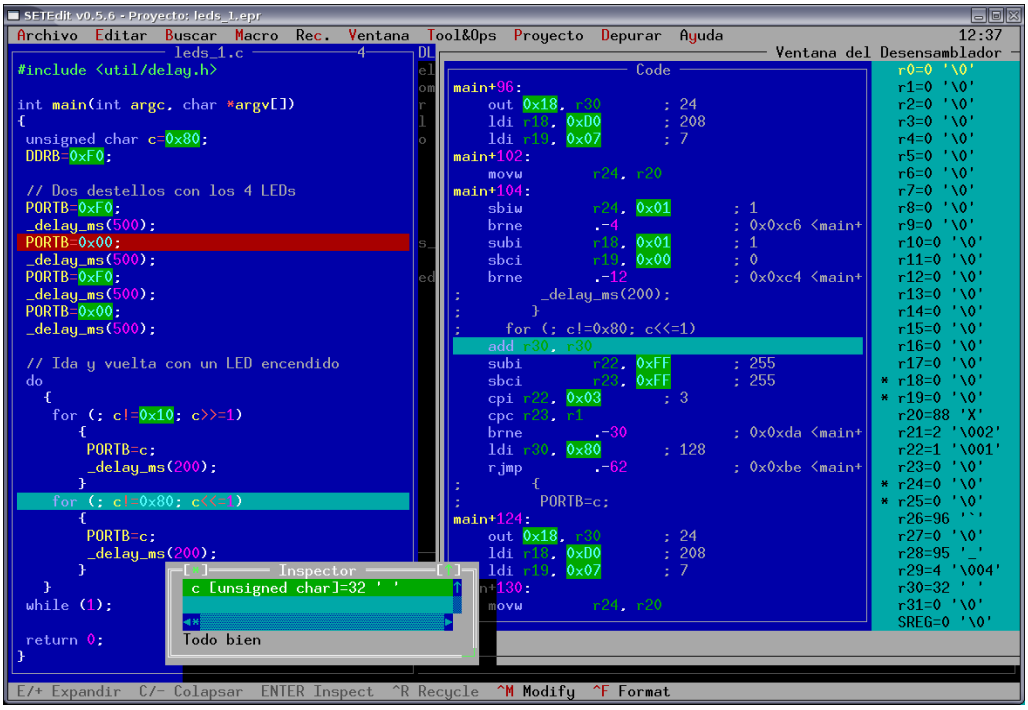

Fig. 3. Sesión de depuración.

7. REFERENCIAS

[1] S. E. Tropea and J. P. D. Borgna, "Microcontrolador compatible con PIC16C84, bus WISHBONE y video," in *FPGA Based Systems*. Mar del Plata: Surlabs Project, II SPL, 2006, pp. 117–122.

[2] S. E. Tropea, "Interfaz de depuración para microcontrolador," in *2008 4th Southern Conference on Programmable Logic Designer Forum Proceedings*, Bariloche, 2008, pp. 105–108.

[3] R. M. Cibils, A. Busto, J. L. Gonella, R. Martinez, A. J. Chielens, J. M. Otero, M.Ñuñez, and S. E. Tropea, "Wide range neutron flux measuring channel for aerospace application," in *Space Technology and Applications International Forum-STAIF 2008 Proceedings*, vol. 969, New México, 2008, pp. 316–325.

[4] S. E. Tropea, D. J. Brengi, and J. P. D. Borgna, "FPGAlibre: Herramientas de software libre para diseño con FPGAs," in *FPGA Based Systems*. Mar del Plata: Surlabs Project, II SPL, 2006, pp. 173–180.

[5] INTI Electrónica e Informática *et al.*, "Proyecto FPGA Libre," http://fpgalibre.sourceforge.net/.

[6] Debian, "Sistema operativo Debian GNU/Linux," http://-www.debian.org.

[7] "GNU project," http://www.gnu.org/.

[8] R. Lepetenok. (2009, Nov.) AVR Core. OpenCores.org. [Online]. Available: http://www.opencores.org/project,avr_core

[9] Silicore and OpenCores.Org, "WISHBONE System-on-Chip (SoC) interconnection architecture for portable IP cores," http://prdownloads.sf.net/fpgalibre/wbspec_b3-2.pdf?download.

[10] (2009, Nov.) Avr assembler (avra). [Online]. Available: http://avra.sourceforge.net/

[11] (2009, Nov.) GCC, the GNU compiler collection. [Online]. Available: http://gcc.gnu.org/

[12] M. Michalkiewicz, J. Wunsch *et al.* (2009, Nov.) AVR C runtime library. [Online]. Available: http://www.nongnu.org/-avr-libc/

[13] (2009, Nov.) GDB: The GNU project debugger. [Online]. Available: http://www.gnu.org/software/gdb/

[14] T. A. Roth *et al.* (2009, Nov.) Simulavr: an AVR simulator. [Online]. Available: http://savannah.nongnu.org/-projects/simulavr/

[15] A. Trapanotto, D. J. Brengi, and S. E. Tropea, "Puente IEEE 1284 en modo EPP a bus WISHBONE," in *FPGA Based Systems*. Mar del Plata: Surlabs Project, II SPL, 2006, pp. 257–264.

[16] R. A. Melo and S. E. Tropea, "IP core puente USB a WISHBONE," in *XV Workshop Iberchip*, vol. 2, Buenos Aires, 2009, pp. 531–533.

[17] S. E. Tropea and R. A. Melo, "USB framework - IP core and related software," in *XV Workshop Iberchip*, vol. 1, Buenos Aires, 2009, pp. 309–313.

[18] S. Finneran. (2009, Nov.) AVR in circuit emulator. [Online]. Available: http://avarice.sourceforge.net/

[19] Salvador E. Tropea *et al.*, "SETEdit, un editor de texto amigable," http://setedit.sourceforge.net.