Creación de bancos de prueba complejos usando Software Libre

Tropea S., Borgna J. P.

Instituto Nacional de Tecnología Industrial,
Centro de Electrónica e Informática,
Unidad Técnica Instrumentación y Control,
Buenos Aires, Argentina,
salvador@inti.gov.ar - http://utic.inti.gov.ar/

Abstract. En este trabajo se presentan las conclusiones obtenidas al investigar la posibilidad de utilizar herramientas de Software Libre para la creación de bancos de prueba complejos. Esta técnica fue aplicada para el desarrollo de un microcontrolador compatible con el 16C84 que posee salida para monitor compatible con VGA. El desarrollo se realizó para FPGAs de la familia Spartan II utilizando sólo Software Libre para su verificación y GNU/Linux como plataforma de trabajo.

1 Introducción

Cuando se encaran desarrollos que manejan muchas señales con correlaciones temporales complejas es necesario utilizar herramientas poderosas para poder depurar y/o validar el diseño. No es posible el simple uso de herramientas que grafican las formas de onda de las señales en juego. En estos casos el uso de bancos de prueba complejos es una buena forma de solucionar el problema.

Las herramientas necesarias para esto suelen ser muy costosas y no siempre están disponibles para todos los sistemas operativos. En nuestro caso todas las tareas de desarrollo se llevan a cabo utilizando Debian[1] GNU[2]/Linux[3]. Por estas razones se buscó obtener herramientas de Software Libre para poder realizar los bancos de prueba.

Para probar dichas herramientas y buscar una metodología de trabajo adecuada se encaró el diseño de un microcontrolador compatible con el 16C84 de Microchip[4] con salida de video compatible con VGA en modo texto.

2 Herramientas Seleccionadas

Se buscaron herramientas para editar los fuentes VHDL, validar la sintaxis de los mismos, generar bancos de prueba complejos, procesar y representar los resultados generados por el banco de pruebas. A continuación se describen las herramientas seleccionadas.

2.1 Edición de los Fuentes

Se seleccionó el editor de texto SETEdit[5], que permite editar código fuente VHDL. El mismo es Software Libre, posee coloreado de sintaxis y características de edición avanzadas útiles para editar código fuente. Adicionalmente permite realizar saltos a entidades, arquitecturas, componentes, funciones, procedimientos y paquetes VHDL definidos en el fuente que se está editando.

2.2 Validación de Sintaxis y Simulación

Se probaron varias herramientas de esta categoría y sólo una cumplía con nuestros requisitos. Entre ellas el SAVANT 1.x[6] (creado por la Universidad de Cincinnati), el SAVANT 2.x[7] (ahora mantenido por Clifton Labs.), el simulador del proyecto FreeHDL [8] y el del proyecto Alliance[9] (muy orientado al desarrollo de circuitos integrados y no tanto al trabajo con FPGAs). Ninguno de estos era adecuado para nuestras tareas.

Finalmente seleccionamos al proyecto GHDL[10]. El mismo ofrece un frontend para el compilador GCC[11] del proyecto GNU. GHDL tiene un soporte muy amplio y estricto del lenguaje VHDL, superando en muchas áreas a herramientas de los fabricantes líderes de herramientas tipo EDA.

El GHDL permite generar binarios ejecutables que simulan el comportamiento de la descripción de hardware. Utilizando sentencias tipo *assert* es posible realizar verificaciones en los bancos de prueba. La velocidad de simulación de los binarios generados no es tan alta como la de los generados con herramientas costosas pero es aceptable para el tamaño de los proyectos actualmente manejados por nuestro laboratorio.

Impresión Formateada. Si bien el uso de assert es útil para detectar errores o validar si los resultados coinciden con los esperados nuestro objetivo era realizar un post procesamiento de los resultados para representarlos de una manera más conveniente. Si bien VHDL posee funciones se salida de texto estas no son cómodas para imprimir resultados de manera formateada. Para lograr esto encontramos una biblioteca[12] VHDL que implementa la mayor parte de las funciones estándares de C en VHDL. Entre ellas se encuentra la función *printf* que permite la impresión de valores con formato. Esta biblioteca fue presentada en las conferencias de la SNUG (Synopsys Users Group) y pueden ser usadas con GHDL haciendo sólo unos pequeños ajustes.

2.3 Procesamiento y Representación de los Resultados

Debido al gran volumen de datos generados por estos bancos de prueba se hace necesario poseer mecanismos especiales de validación y representación de los mismos. En nuestro caso decidimos crear un programa en lenguaje C que extrajera la información relevante de los datos generados por el banco de prueba y que realizara una representación gráfica de los mismos. A tales fines seleccionamos como compilador al GCC y como biblioteca gráfica a Allegro[13]. La biblioteca Allegro esta disponible para las plataformas más usadas: DOS, Unix (Linux, FreeBSD, Irix, Solaris, Darwin), Windows, QNX, BeOS y MacOS X.

3 Uso de las Herramientas

Las herramientas antes mencionadas se utilizaron para el desarrollo de un microcontrolador compatible con el 16C84 de Microchip con salida de video compatible con VGA en modo texto.

Fig. 1. Imagen obtenida luego de procesar la información generada por el banco de prueba. La misma predice la imagen obtenida al conectar la FPGA a un monitor VGA

Primero se procedió a desarrollar el generador de video. Este circuito es simple, pero las señales generadas son de compleja interpretación y por lo tanto durante esta parte del desarrollo fue cuando más se necesitó de herramientas que permitieran representar los datos de una manera adecuada. A tales fines se creó un banco de prueba que imprimiera la información más relevante y se compiló con GHDL. Dicho banco de prueba imprime cada línea de barrido en una línea de texto separada y un muestreo de cada punto en la pantalla indicando el color en forma numérica. La información es generada utilizando la función *printf*. Estos valores se guardan en un archivo de texto el cual alimenta a un programa escrito en C que analiza dicha información y como resultado imprime las frecuencias de barrido horizontal y vertical, la resolución en puntos y finalmente grafica una imagen representando lo que debería verse al conectar la FPGA a un monitor VGA (Fig. 1 muestra un ejemplo). Adicionalmente el programa realiza verificaciones tales como asegurarse de que todas las líneas de barrido tengan la misma duración.

Durante el desarrollo ayudó mucho el uso de la herramienta GNU make[14] que permite procesar automáticamente archivos de acuerdo con su fecha de modificación. De esta manera si se cambiaba el fuente en assembler del programa para el microcontrolador el GNU make automáticamente invocaba el ensamblador, luego convertía el archivo .hex en un fuente VHDL (inicialización de la memoria de programa) y finalmente incorporaba los cambios al banco de prueba.

Disponíamos de un kit de desarrollo con una FPGA Spartan II de Xilinx por lo que para la etapa de síntesis se usaron las herramientas de Xilinx (XST, map, etc.). Como la versión para Linux del *ISE Webpack* no es gratuita utilizamos el programa Wine[15] para ejecutar la de Windows.

3.1 Datos Técnicos del Proyecto Desarrollado

El proyecto desarrollado implementa casi toda la funcionalidad del 16C84 incluyendo el stack de ocho direcciones, interrupciones, *watch dog*, temporizador/contador, entrada externa de interrupciones e interrupción por cambio en el puerto B. Adicionalmente el procesador posee 464 registros de RAM (sólo 36 en el original) y 24 pines de entrada/salida (sólo 13 en el original). El generador de caracteres se diseño de manera tal de aprovechar los *Block RAM* ofrecidos por la FPGA y por lo tanto se decidió acotarlo. El mismo posee 64 caracteres diferentes y 8 colores, tanto para seleccionar el color de fondo como el de primer plano. La resolución es de 40 columnas y 25 líneas.

El generador de video se incorporó al microcontrolador agregando dos registros especiales que permiten direccionar hasta 256 dispositivos de entrada/salida. Dichos dispositivos se conectan directamente al bus interno de registros del PIC permitiendo utilizar operaciones de tipo Lectura/Modificación/Escritura con los mismos.

Utilizando una FPGA XC2S100-5-PQ208 (Spartan II) las herramientas de Xilinx reportaron un uso de 390 slices (sobre 1200) y una frecuencia máxima de clock de 30 MHz (el PIC 16C84 soporta sólo 10 MHz).

4 Conclusiones

Se verificó que cuando se encaran proyectos que generan señales muy complejas es imposible utilizar herramientas comunes de representación de formas de onda y es necesario recurrir a bancos de prueba complejos y a software que interprete y represente adecuadamente la información generada por dichos bancos de prueba. Estas tareas pueden ser llevadas a cabo utilizando Software Libre reduciendo los costos de desarrollo y permitiendo el uso de plataformas de desarrollo confiables como es el caso de Debian GNU/Linux.

Referencias

1. Debian http://www.debian.org/
2. GNU Project http://www.gnu.org/
3. Linux http://www.linux.org/
4. Microchip Technology Inc. http://www.microchip.com/
5. SETEdit http://setedit.sourceforge.net/
6. SAVANT http://www.ececs.uc.edu/~paw/savant/
7. SAVANT 2 http://www.cliftonlabs.com/savant.htm
8. FreeHDL http://www.freehdl.seul.org/
9. Alliance http://www-asim.lip6.fr/recherche/alliance/
10. GHDL http://ghdl.free.fr/
11. GCC (GNU Compiler Collection) http://gcc.gnu.org/
12. Public Domain VHDL packages http://bear.ces.cwru.edu/vhdl/
13. Allegro http://www.talula.demon.co.uk/allegro/
14. GNU make http://www.gnu.org/software/make/make.html
15. Wine http://www.winehq.com/