Revista Digital del Departamento de Ingeniería e Investigaciones Tecnológicas de la Universidad Nacional de La Matanza

ISSN: *2525-1333*
Vol.:1-Nro.2 (Diciembre-2016)

*Comunicación en congreso*

# DESARROLLO DE MICROCONTROLADOR EMBEBIDO EN FPGA

# EMBEDDED MICROCONTROLLER DEVELOPMENT FOR AN FPGA

*Mauro CIPOLLONE, Carlos E. MAIDANA, Fernando I. SZKLANNY*

UNLaM, DIIT, Grupo de Investigación en Lógica Programable

mcipollone@unlam.edu.ar

**Resumen:**

En el presente trabajo se describe el proceso de desarrollo inicial de un microcontrolador descripto en VHDL para poder ser implementado sobre una FPGA. Esto comprende la descripción del núcleo del sistema y de sus elementos asociados para formar una unidad de cómputo completa, los bloques que permiten la programación externa y las herramientas necesarias para desarrollar los programas a ejecutar. El objetivo final de este trabajo es no solo obtener la descripción completa del microcontrolador, sino también adquirir experiencia en el campo, para encarar a futuro proyectos de mayor complejidad como sistemas de procesamiento de alta velocidad y procesadores de múltiples núcleos.

**Abstract:**

This paper describes the initial development process of a microcontroller coded in VHDL in order to be implemented into an FPGA. This it is a way to understand the description of the system's core and its associated elements in order to accomplish a complete computing unit, the modules that allow the external programming, and the tools used to develop the programs to be executed. The final objective of this job is

ISSN: 2525-1333 Vol.:1-Nro.2 (Diciembre-2016)

not only to obtain the complete description of the microcontroller, but also to acquire the experience in the field to face more complex projects in the future, like high speed processing systems and processors with multiple cores.

**Palabras Clave:** *Microcontrolador, sistema embebido, FPGA, VHDL, ensamblador*

**Key Words:** *Microcontroller, embedded system, FPGA, VHDL, assembler*

**Colaboradores:** *Martín FERREYRA BIRON, Federico ORTALDA*

ISSN: 2525-1333 Vol.:1-Nro.2 (Diciembre-2016)

## I. CONTEXTO

El presente proyecto es impulsado desde el DIIT y surgió a partir de la necesidad de contar con un módulo que se encargue de las tareas de control, comando y monitoreo dentro de dispositivos de hardware reconfigurable (por ejemplo, una FPGA) empleados en un conjunto de proyectos llevados a cabo por el Grupo de Investigación en Lógica Programable (GILP) de la UNLaM.

## II. INTRODUCCIÓN

A diferencia de trabajos anteriores presentados por los mismos autores [1][2], centrados en el rendimiento y la estructura de este tipo de procesadores, este trabajo se aborda desde el punto de vista del diseñador del sistema; estudiando y eligiendo las mejores alternativas en base a las propuestas que se pueden encontrar en la literatura relacionada y distintas opciones comerciales.

Hoy en día el desarrollo sobre dispositivos de hardware programable se encuentra en expansión debido a las ventajas que estos ofrecen, principalmente en referencia a los reducidos tiempos de demora para procesar grandes volúmenes de datos. Sin embargo, estos diseños puros en hardware adolecen de la capacidad de realizar en forma sencilla algunas tareas, por ejemplo ejecuciones cíclicas o tareas de baja velocidad y gran cantidad estados.

Con el objeto de solucionar este problema se presentan diseños híbridos que incluyen embebidos, la sección de lógica programable junto a un microcontrolador. De este modo, el usuario puede diseñar contando con las ventajas de ambos mundos: la alta velocidad de procesamiento de un sistema de hardware digital, y la flexibilidad multipropósito de un microcontrolador.

El alcance del presente proyecto es el desarrollo del microcontrolador completo junto a las herramientas necesarias para trabajar en forma adecuada con el mismo.

A continuación se realiza una breve reseña de las características principales de las que dispone el microcontrolador diseñado:

- Arquitectura Harvard RISC.
- Palabra de instrucción de 16 bits (salvo excepciones).
- Bus de ocho bits hacia la memoria de datos.
- ALU de ocho bits con incorporación de un "barrelshifter".
- 32 registros de ocho bits para propósito general.
- Ejecución de cualquier instrucción en ocho ciclos de reloj.
- Sin "pipeline" (segmentación).
- Todas las operaciones se realizan sobre los registros internos de la CPU.
- Modos de direccionamiento: implícito, inmediato, directo, indexado con autoincremento o autodecremento, relativo, de registro.
- Posibilidad de programación desde PC a través de puerto serie.

## III. DESARROLLO

### *A. Definición de la arquitectura*

#### *1. Von Neumann vs. Harvard*

Debido a que se busca que este microcontrolador pueda trabajar rápido sin por ello perder la simplicidad, se ha optado por emplear arquitectura Harvard. De este modo, se pueden realizar operaciones sobre la memoria de datos al mismo tiempo que se está buscando la próxima instrucción a ejecutar, contribuyendo a que todas las instrucciones se ejecuten en un solo ciclo de instrucción (ocho ciclos de reloj) [3].

#### *2. CISC vs. RISC*

Para definir el tipo de set de instrucciones se presentan dos enfoques: una arquitectura de tipo CISC (juego de instrucciones complejo) o una arquitectura RISC (juego de instrucciones reducido). Nuevamente buscando la simplicidad y eficiencia en el diseño, se optó por un set de instrucciones de tipo RISC. De esta manera la CPU requiere menor cantidad de hardware para ejecutar todas las instrucciones y se puede lograr que estas se ejecuten en igual cantidad de ciclos [3].

*3. Tamaño de los buses*

Desde el punto de vista de la memoria de datos, como el proyecto se refiere a un microcontrolador de pequeñas o medianas prestaciones, se decidió que el bus de datos de la misma fuese de ocho bits. Por otro lado, desde el punto de vista de la memoria de instrucciones se decidió que el bus de datos fuese de 16 bits (teniendo en cuenta una palabra de instrucción de 16 bits) para que de este modo se pueda traer una instrucción completa en un solo ciclo de reloj.

*4. Set de instrucciones*

Para diagramar el set de instrucciones se analizaron microcontroladores de prestaciones similares a las que se buscan. Entre estos los más destacados son: el PIC16F84 de Microchip [4] y ATmega32 de Atmel [5].

Se decidió que el tamaño de la palabra de instrucción fuese de 16 bits, fijo para todas las instrucciones, con los seis primeros bits dedicados a codificar el código de operación (máximo 64 instrucciones). Solo unas pocas instrucciones han requerido ser codificadas en palabras de 32 bits (sin alterar la condición de que un ciclo de instrucción se resuelva en ocho de reloj).

*B. Implementación*

La implementación del sistema se encuentra dividida en dos grupos: la estructura interna y la estructura externa. La estructura interna comprende al microcontrolador propiamente dicho y está formada por todos aquellos bloques que tienen contacto con los buses del sistema. Por otro lado, la estructura externa se encuentra formada por aquellos bloques que sirven de apoyo para la operación del microcontrolador y no tienen contacto directo con los buses.

En la Fig. 1 se presenta un diagrama en bloques que muestra la interconexión de los bloques principales del sistema. La estructura interna se encuentra dentro del bloque "MCU", mientras que la estructura externa está formada por los bloques controlador (Controller) y UART.

*C. Estructura interna*

A continuación se describe cada uno de los bloques que componen la estructura interna del sistema. En la Fig. 2 se ejemplifica la interconexión entre estos.

*1. CPU – Unidad Central de Proceso*

La CPU incluye tanto la unidad de control (UC) como el "datapath" (ruta de datos). Dos de los bloques del datapath, la Unidad Aritmético Lógica y el banco de registros, fueron descriptos en archivos separados para facilitar su edición y sus características principales se detallan a continuación:

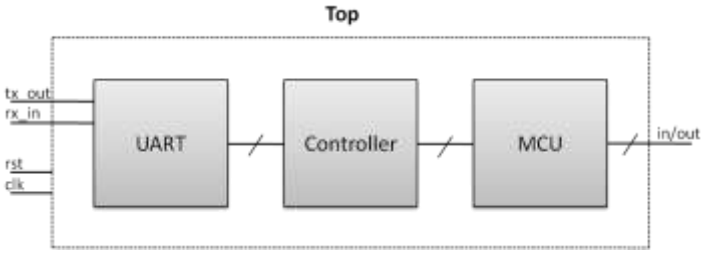


Fig. 1. Diagrama en bloques de interconexión de bloques principales.

ISSN: 2525-1333 Vol.:1-Nro.2 (Diciembre-2016)

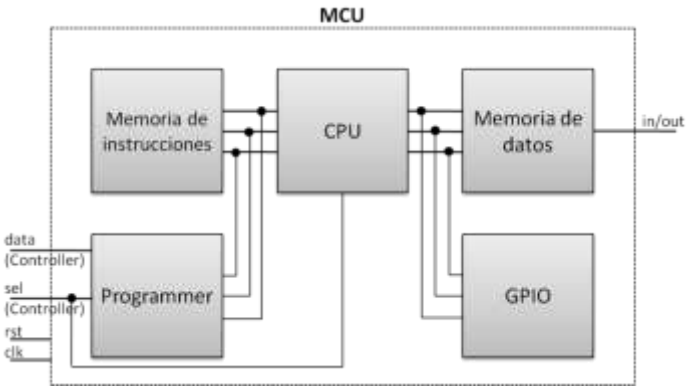


Fig. 2 - Diagrama en bloques de interconexión entre bloque de estructura interna.

- ALU (Unidad Aritmético-Lógica): Bloque encargado de llevar a cabo todas las operaciones aritmético-lógicas en un solo ciclo de reloj, inclusive los desplazamientos gracias a la incorporación del barrel shifter. Las entradas son los dos operandos, la operación a ejecutar y las señales de control. Las salidas son el resultado y el registro de estado.
- Banco de registros: Este bloque dispone de 32 registros de ocho bits cada uno. Esta cantidad de registros se debe al hecho de que todas las operaciones de la ALU tienen como origen y destino los registros internos de la CPU, por lo tanto, es necesario disponer de una buena cantidad de registros para minimizar los accesos a memoria.

Por otro lado, el datapath incluye los registros de uso específico, los cuales se mencionan a continuación:

- Status register (Registro de estado): Este registro de ocho bits incorpora las banderas de cero, overflow, carry, negativo e interrupciones globales, junto a tres bits que permiten indicar distintos estados del microcontrolador.
- Address Pointers (Punteros): Son dos registros de 16 bits que contienen la dirección sobre la cual se va a operar para las instrucciones de load y store que emplean el modo de direccionamiento indirecto.
- Stack pointer (Puntero de pila): Registro de 16 bits que apunta a la última dirección libre dentro de la zona de memoria reservada para la pila.
- StackLimit (Límite de pila): Registro de 16 bits que contiene la dirección de la última posición de la pila y es comparado en todo ciclo de instrucción con el puntero de pila para detectar si se produjo un desborde.
- ProgramCounter (contador de programa): Registro de 16 bits que apunta a la dirección de la próxima instrucción a recuperar de la memoria de instrucciones.
- Memory Buffer Register: Este registro contiene el último dato leído de la memoria.

Finalmente, la Unidad de Control se implementó como un secuenciador de ocho etapas (un ciclo de instrucción = ocho ciclos de reloj). En cada una de estas se describen las acciones a ejecutar sobre el datapath.

### *2. Memoria de instrucciones*

Este es un bloque sencillo, dimensionado para poder contener hasta 8192 instrucciones de 16 bits cada una (8Kx16).

### *3. Memoria de datos*

Esta memoria tiene exactamente la misma estructura que la memoria de instrucciones pero su tamaño es de 1Kx8 y de todas estas direcciones las últimas se encuentran reservadas para la pila y las primeras están reservadas para los registros de los periféricos. Es decir, se implementó un modelo de E/S mapeada en memoria, el que si bien inutiliza una porción del espacio de direccionamiento, trae como ventaja el poder utilizar las mismas instrucciones de carga y almacenamiento para escribir en memoria como para escribir en los registros de los periféricos.

*4. Programador*

Antes de comenzar con la descripción de este bloque se debe hacer una introducción sobre el sistema de programación. El microcontrolador recibe los datos a programar con una determinada estructura de trama a través de la UART. Luego, el Controlador es el encargado de reconocer el tipo de comando a ejecutar y transmitir los datos correspondientes. En principio el MCU puede recibir dos órdenes principales:

- Modo ejecución: El Programador pone en estado de alta impedancia todas sus salidas hacia los buses de la memoria de instrucciones y la CPU controla dicha memoria.
- Modo programación: La CPU pone en alta impedancia todas sus salidas hacia la memoria de instrucciones y el Programador es quien la controla.

Habiendo definido esto es más sencillo comprender la función del bloque en cuestión. Este es el encargado de grabar la memoria de instrucciones con el programa a ejecutar, y es por esta causa que debe tomar la posesión de los buses de dicha memoria.

*5. GPIO*

Este bloque es un puerto de E/S clásico de un microcontrolador de 8 bits y es el primero de todos los periféricos a incluir dentro de la estructura interna.

*D. Estructura externa*

A continuación se describen los bloques que componen la estructura externa del sistema. Si bien su función no tiene relevancia durante la ejecución del programa, estos permiten cargar desde una PC el programa dentro de la memoria, lo cual evita que el mismo deba ser embebido en la síntesis del proyecto en VHDL.

Por otro lado, esta configuración permite una sencilla depuración del programa corriendo el mismo sobre la plataforma real.

*1. UART*

Este bloque es una UART genérica cuya operación se encuentra fija para trabajar a 9600 baudios en formato 8N1. Su función es la de recibir los bytes que llegan desde la PC y transmitirlos al bloque controlador, así como también la de recibir los bytes que le llegan desde el controlador y transmitirlos hacia la PC.

*2. Controlador*

Este es el encargado de recibir los comandos desde la PC y actuar en forma correspondiente. Estos comandos le llegan como tramas que responden a la estructura que se observa en (1).

$$\text{“a”} + \text{tipo} + \text{dato\_h} + \text{dato\_l} + \text{“z”} \qquad (1)$$

Dónde:

- “a”: Carácter de inicio.
- tipo: Tipo de comando a ejecutar: “modo programación”, “modo ejecución” o “dato a grabar”.
- dato_h/dato_l: Parte alta y parte baja del dato a grabar en la memoria de instrucciones.
- “z”: Caracter de fin.

Los caracteres de inicio y fin permiten validar las tramas para evitar errores como, por ejemplo, un falso inicio o la pérdida de algún byte en la transmisión. Por otro lado, a futuro se planea establecer mayor cantidad de comandos en el campo “tipo” los que permitirán una depuración en tiempo real del sistema y la escritura de la memoria en forma no secuencial (necesario para la incorporación de subrutinas y rutinas de interrupción).

*E. Ensamblador*

Hasta el momento se ha desarrollado un ensamblador que toma el código assembler y lo transforma en código

de máquina. Este programa se desarrolló en lenguaje C++ y las instrucciones en el archivo fuente deben responder a la estructura señalada en (2).

ADD R0,R1; (2)

Es decir, primero el código nemotécnico de la instrucción, luego uno o más espacios seguidos de los operandos correspondientes separados por comas y finalmente un punto y coma señalizando el fin de la instrucción.

*F. Carga del programa*

Para transmitir los datos desde la PC al microcontrolador se puede utilizar cualquier tipo de programa que permita controlar el puerto serie, siendo el RealTerm el elegido en este proyecto.

## IV. RESULTADOS

En un primer momento el sistema sólo estaba formado por el módulo programador, la memoria de instrucciones y la CPU, a la cual se le había incorporado una salida al mundo exterior ya que no se disponía de ningún periférico.

Ya que no se disponía de la UART y el Controlador para poder transmitir las instrucciones desde la PC, todas las pruebas se ejecutaban con simuladores.

A modo de ilustración, se presenta a continuación el resultado de una simulación donde se ejecuta el programa de la Fig. 3.

```
LDK R0,160;
INC R0;
BRA -1;
```

Fig. 3. Programa ejemplo

La primera instrucción es una carga sobre el registro cero del valor constante 160, luego se incrementa dicho registro en uno y, finalmente, se entra en un lazo sin fin en el cual se repite el incremento del registro.

En la Fig. 4 se observa el tiempo que demora la ejecución de una instrucción, el cual es de ocho ciclos de reloj como se mencionó previamente.

Por otro lado, en la Fig. 5 se observa el detalle de un tramo de ejecución de programa. Los opcode 49, 14 y 24 corresponden a las instrucciones LDK, INC y BRA respectivamente. En esta imagen se observa que las tres instrucciones siguen manteniendo el tiempo de ejecución

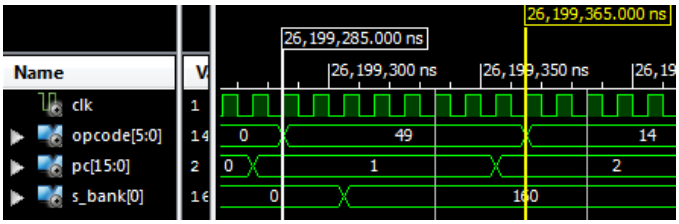


Fig. 4.Medición de un ciclo de instrucción.

de ocho ciclos de reloj cada una; el PC avanza en forma creciente conforme pasa el tiempo (hasta el momento del salto); y el resultado es el esperado, se incrementa en pasos de uno el registro cero (s_bank[0] en la simulación) conforme pasa el tiempo.

Luego, una vez que se implementaron una gran cantidad de instrucciones, se incorporaron el UART y el controlador. A partir de este punto los primeros programas eran cargados en forma manual para probar el correcto funcionamiento del sistema de programación y a continuación se procedió a desarrollar el ensamblador.

Finalmente, otro aspecto a remarcar de este proyecto es que ocupa un porcentaje muy bajo de los recursos de una FPGA promedio. De la FPGA Spartan-6 LX45 el sistema solo ocupa el 1% de los Slice Registers (Flip-Flops), el 3% de las LUTs, el 1% de los multiplexores y el 5% de los Slices totales. Cada CLB de esta arquitectura está compuesta por dos bloques llamados Slices y cada Slice está conformada por un conjunto de elementos lógicos [6].

ISSN: 2525-1333 Vol.:1-Nro.2 (Diciembre-2016)

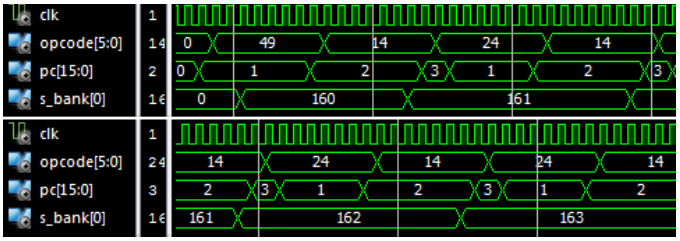


Fig. 5. Diagrama temporal de la ejecución del programa ejemplo.

## VI. CONCLUSIONES

Actualmente, la próxima funcionalidad a incluir en el sistema es el manejo de interrupciones, el cual exige tanto reformas en la estructura interna, la estructura externa y en el ensamblador.

Finalizado el desarrollo de todos los periféricos, se procederá al desarrollo de un compilador C/C++ junto a herramientas de depuración para lograr un flujo de trabajo estándar, propio de las soluciones comerciales.

En etapas posteriores (finalizado el desarrollo tal como se presentó en este artículo) se analizará la posibilidad y conveniencia de:

- Incorporar el controlador y la UART dentro de la estructura interna (es decir, conectarlos directamente a los buses del microcontrolador).
- Adaptar la CPU para trabajar con pipeline. Si bien esta incorporación plantea una profunda reforma, la misma es factible ya que todas las instrucciones se ejecutan en igual tiempo e igual cantidad de etapas, necesario para trabajar con pipeline [7], y la modularidad del diseño permite la reutilización de una buena proporción de código.

## VII. REFERENCIAS Y BIBLIOGRAFÍA

*A. Referencias bibliográficas:*

[1] Mauro Cipollone, Ing. Fernando I. Szklanny, Lic. Carlos E. Maidana. "Implementación de un microprocesador embebido sobre un FPGASpartan 6", *IV Congreso de MicroElectrónica Aplicada – UEA 2013*, ISBN: 978-987-1896-18-9.

[2] Mauro Cipollone, Ing. Fernando I. Szklanny, "Implementación de Linux sobre un microprocesador embebido en una FPGASpartan 6",*V Congreso de MicroElectrónica Aplicada – UEA 2014",* ISBN: 987-24680-5-2.

[3] Wikibooks, "Microprocessor Design", en https://upload.wikimedia.org/wikipedia/commons/7/71/MicroprocessorDesign.pdf - 02/01/2017

[4] "PIC16F8X – /Data Sheet",en /http://pdf1.alldatasheet.com/datasheet-pdf/view/74975/MICROCHIP/PIC16F84.html - 02/01/2017

[5] "ATmega32 - Data Sheet", enhttp://pdf1.alldatasheet.com/datasheet-pdf/view/77378/ATMEL/ATMEGA32.html - 02/01/2017

[6] Xilinx Inc,"Spartan-6 FPGA Configurable Logic Block – User Guide(UG384)", 02/2010.

[7] John L. Hennesy y David A. Patterson, "Computer Architecture, a quantitative approach", ISBN:9780123838728, Quinta edición, McGraw-Hill.

**Recibido:** 2016-12-12
**Aprobado:** 2016-12-29
**Datos de edición:** Vol. 1-Nro. 2-Art. 2
**Fecha de edición:** 2016-12-30