# Diseño e Implementación de un CPU RISC Superescalar en un FPGA para Enseñanza e Investigación

Aurelio Morales Villanueva, *Member, IEEE*

***Resumen*—El presente trabajo de investigación muestra el diseño e implementación en hardware basado en un FPGA, de un CPU RISC superescalar de dos vías para la enseñanza e investigación, usando herramientas de diseño electrónico EDA y lenguaje VHDL. La estrategia de diseño seleccionada fue la de top-down. El diseño del CPU está basado en la arquitectura MIPS32 e incluye los bloques de adelanto de datos (Forwarding Unit), detección de dependencias de datos (Hazard Detection Unit), predicción de saltos (Branch Prediction Unit), entre otros. En la implementación del prototipo se usó la tarjeta educacional DE2 de Altera y con salida a video VGA. Se realizaron pruebas sobre el predictor de saltos, mostrando resultados aceptables, comparados a usar un simple predictor de 2 bits.**

***Palabras Claves*—Arquitectura MIPS, FPGA, VHDL, RISC superescalar, Branch Prediction Unit, Forwarding Unit, Hazard Detection Unit, pipeline.**

## I. INTRODUCCION

LOS fabricantes de FPGAs como Xilinx y Altera ofrecen soluciones de procesadores RISC escalares pipeline del tipo embebido como MicroBlaze y Nios II, respectivamente, conocidos como "*soft processor*" [1], [2]. Ambos fabricantes ofrecen un soporte completo de conexión de periféricos a sus procesadores y programación en lenguaje C/C++, pero es imposible depurar el hardware de estos procesadores, desde que el código que describe el hardware de los mismos está protegido.

Esto motivó el desarrollo de un proyecto que involucró el diseño e implementación de un CPU RISC superescalar a medida en un FPGA, y en el cual se pueda realizar depuración de hardware para ser usado en actividades académicas.

Las características principales de la implementación del CPU RISC superescalar en FPGA se resumen a continuación: CPU superescalar de dos vías, memoria embebida de instrucciones de 1024 x 64 bits (en dos memorias de 1024x32 bits), memoria embebida de datos de 1024x32 bits, dos colas de instrucciones aritmético/lógica (cada una con 3 etapas de pipeline), una cola de instrucción tipo Load/Store con 4 etapas de pipeline, dos colas FIFO capaces de almacenar cada una hasta 4 instrucciones, un conjunto de 32 registros para instrucciones enteras, soporte de un subjuego de instrucciones enteras de la arquitectura MIPS32 [3], envío de instrucciones es en orden, finalización de instrucciones es en orden, predicción de saltos dinámica de dos niveles usando historia local, e incluye las unidades de detección de dependencias de datos y de adelanto de datos.

A continuación se cubren conceptos fundamentales para un mejor entendimiento del trabajo de investigación. Se debe mencionar que los conceptos vertidos se focalizan hacia CPUs del tipo RISC (**R**educed **I**nstruction **S**et **C**omputer).

### *A. CPU escalar*

Un CPU del tipo RISC es un modelo de arquitectura de procesador en el que se facilita la segmentación y el paralelismo en la ejecución de instrucciones, reduciendo los accesos a memoria. Los diseños de CPU RISC utilizan la arquitectura Harvard, donde las instrucciones y los datos están en memorias separadas, las cuales se implementan hoy en día en memoria del tipo cache.

Un CPU RISC pipeline del tipo escalar permite que por cada nuevo pulso de reloj se termine de ejecutar una nueva instrucción, bajo el supuesto que no existan dependencias de datos o de control. Las etapas básicas de un CPU de este tipo son: búsqueda de instrucción (**I**nstruction **F**etch, IF), decodificación (**I**nstruction **D**ecode, ID), ejecución (**EXE**cute, EXE), lectura de memoria (**MEM**ory, MEM), y escritura en registro (**W**rite **B**ack, WB). Un detalle de funcionamiento de un CPU RISC pipeline escalar se puede encontrar en [4] y [5].

### *B. CPU superescalar*

Superescalar es el término utilizado para designar un tipo de arquitectura de procesador capaz de terminar de ejecutar más de una instrucción por ciclo de reloj. La arquitectura superescalar se basa en la arquitectura escalar, donde existen varias unidades funcionales del mismo tipo para realizar mas de una función a la vez. La estructura básica de un procesador superescalar contiene las siguientes etapas: búsqueda de instrucción (IF), decodificación (ID), lanzamiento (Dispatch), ejecución (EXE), memoria (MEM) y escritura en registro (WB). También es frecuente encontrar mas de una cola de instrucciones para cada tipo de instrucción, como por ejemplo

Manuscrito recibido el 16 de Marzo del 2011. Este trabajo fue financiado en parte por el Instituto de Investigación de la Facultad de Ingeniería Eléctrica y Electrónica (IIFIEE) de la Universidad Nacional de Ingeniería (UNI), Lima, Perú.

Aurelio Morales Villanueva es Profesor Asociado de la Facultad de Ingeniería Eléctrica y Electrónica (FIEE) de la Universidad Nacional de Ingeniería (e-mail: amorales@uni.edu.pe).

para instrucciones aritmético/lógicas enteras, para carga y almacenamiento, para operaciones de punto flotante, entre otras, y la longitud de las colas de instrucciones de diferente tipo no necesariamente es la misma.

En un procesador superescalar, el procesador lee más de una instrucción por ciclo de reloj, de no existir dependencias. El número máximo de instrucciones simultáneas que se lee desde la memoria de instrucciones por ciclo de reloj se denomina grado, así que, un procesador superescalar de grado 4 (conocido como "*4-way superscalar CPU*") es capaz de leer como máximo cuatro instrucciones por ciclo de reloj.

En lo que concierne a la emisión y finalización de instrucciones en un CPU superescalar, podemos tener emisión y finalización tanto en orden como en desorden, pudiéndose combinar estas opciones. Un detalle del funcionamiento de CPUs RISC superescalares se puede encontrar en [6] y [7].

### C. *Dependencias en un CPU superescalar*

Un procesador superescalar es capaz de ejecutar más de una instrucción simultáneamente únicamente si las instrucciones no presentan algún tipo de dependencia. Los tipos de dependencias entre instrucciones son: estructural, de datos, anti-dependencia, de escritura y de control. La dependencia estructural (Structural Hazard), ocurre cuando dos instrucciones requieren el mismo tipo de unidad funcional y su número no es suficiente. La dependencia de Datos (Data Hazard o RAW, **R**ead **A**fter **W**rite dependency) ocurre cuando una instrucción necesita del resultado de una instrucción anterior para ejecutarse, por ejemplo add r1,r2,r3 (r1 ← r2+r3) y add r4,r1,r5 (r4 ← r1+r5). La anti-dependencia o falsa dependencia (WAR, **W**rite **A**fter **R**ead dependency) ocurre cuando una instrucción escribe en un registro que aun no ha sido leído previamente en una instrucción anterior, por ejemplo add r1,r2,r3 (r1 ← r2+r3) y add r2,r4,r5 (r2 ← r4+r5). La dependencia de escritura (WAW, **W**rite **A**fter **W**rite dependency) ocurre cuando una instrucción necesita escribir en el mismo registro que una instrucción anterior pero que aun no se concreta, por ejemplo add r1,r2,r3 (r1 ← r2+r3) y add r1,r4,r5 (r1 ← r4+r5). Por último, la dependencia de control se debe a que en el flujo de instrucciones se encuentran instrucciones de ramificación o salto, que impactan en el desempeño del CPU, generándose congelamiento en las colas de instrucciones o en el peor caso deshacer el contenido actual en varias etapas de las colas de instrucciones.

## II. Trabajos Relacionados

En esta sección se resumen los trabajos realizados por otros y que tienen relación con el presente proyecto.

Los autores en [8] diseñaron un CPU RISC escalar de 16 bits mostrando resultados de implementación en cuatro FPGAs, pero la implementación no era del tipo pipeline. En [9] los autores diseñaron un CPU RISC basado en la arquitectura MIPS en un FPGA de la familia XC4000 Xilinx para aplicaciones embebidas, y compararon los resultados con una implementación en un ASIC. En [10] los autores implementaron otro CPU RISC de 16 bits en un FPGA Spartan-II de Xilinx el cual constaba de un pipeline de 4 etapas, en el que se implementaba el manejo de dependencias estructurales, de datos y de control, obteniéndose una frecuencia de operación de 26 MHz.

Por otro lado, los autores en [11] diseñaron un CPU RISC pipeline de 32 bits (similar a la arquitectura MIPS descrita en [4]) en un FPGA Cyclone II de Altera con un enfoque en la reducción de consumo de potencia para aplicaciones embebidas.

Por último, los autores de [12] describen el comportamiento del hardware en VHDL del CPU RISC descrito en [4] pero en su implementación de un ciclo de reloj. A diferencia de los otros trabajos, en donde no se menciona si se provee una forma de depuración de hardware, lo cual no facilita hacer un seguimiento señales para la verificación funcional del diseño, los autores en [12] proveen una implementación muy básica para depuración con salida a monitor VGA.

## III. Diseño del CPU RISC Superescalar en FPGA

En esta sección se expone con detalle el diseño del CPU RISC superescalar de dos vías implementado en FPGA. El diseño del CPU superescalar se trabajó sobre la versión escalar de un CPU pipeline de 5 etapas expuesto en [4] el cual se basa en la arquitectura MIPS32 [3].

En la Fig. 1 se muestra el diagrama de bloques del CPU superescalar. El diseño considera la lectura simultánea de dos instrucciones e incluye tres colas de instrucciones. Dos de estas colas son para instrucciones tipo Registro (R) o que incluyen un valor Inmediato (I) y se conocen como *pipe0* y *pipe1*. La tercera cola de instrucciones es para instrucciones del tipo carga o almacenamiento (Load/Store) donde interviene un operando de memoria, y la cola se conoce como *pipe2*. La interacción entre las colas de instrucciones implementa el adelanto de datos de una etapa de pipeline de una cola hacia otra, lo cual es controlado por la Unidad de Adelanto de Datos (**F**orwarding **U**nit, FU). Igualmente, el congelamiento de una cola de instrucciones es manejado por la Unidad de Detección de Dependencias de Datos (**H**azard **D**etección **U**nit, HDU). El diseño incluye una Unidad de Predicción de Saltos (**B**ranch **U**nit, BU), una Unidad de Despacho de Instrucciones (**D**ispatch **U**nit, DU) entre otros bloques. La interacción entre etapas de las colas de instrucciones se establece con el intercambio de dos tipos de señales: de control y de datos.

En la Fig. 1 las señales de control están representadas por números, mientras que los datos están representados por letras. Por ejemplo, la señal "a" representa el dato del registro destino en la etapa WB de *pipe0* que se adelanta hacias las etapas ID y EXE de las colas *pipe0*, *pipe1* y *pipe2*, mientras que las señales 10, 11, 12, 1, 3, 5, 2 y 4 se aplican al bloque FU para decidir donde se adelanta la señal "a".

El diseño del CPU se especificó usando el lenguaje de descripción de hardware VHDL, y las unidades se interconectaron entre ellas usando la interfaz gráfica para la entrada de diseño de la herramienta de software Quartus II de Altera [15]. El detalle de funcionamiento y criterios de diseño

de las diferentes unidades se incluye en los siguientes apartados de la presente sección.

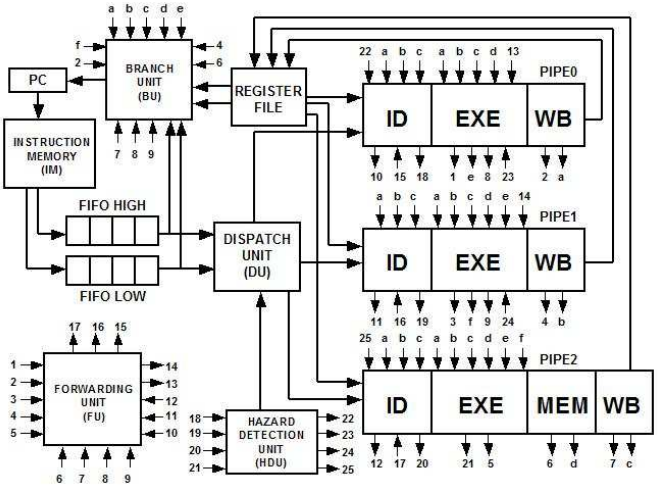


Fig. 1. Diagrama de bloques del CPU RISC superescalar de dos vías implementado en FPGA mostrando la interacción de las colas de instrucciones entre ellas y otros bloques.

A fin de facilitar la depuración en la ejecución de programas y verificación del funcionamiento de cada bloque mostrado en la Fig. 1, las señales de control y datos mencionadas anteriormente se extraen y se muestran en una pantalla VGA. Asimismo, la estrategia de diseño seguida para el proyecto fue la top-down.

### *A. Memoria de Instrucciones (IM)*

Este bloque forma parte de la arquitectura Harvard seleccionada, en donde se almacenan las instrucciones que el CPU ejecutará. El ancho del bus de datos es tal que contiene dos instrucciones, las cuales son leídas simultáneamente por cada ciclo de reloj del CPU. Cada instrucción es de 32 bits, con lo que el bus de datos es de 64 bits. La memoria de instrucciones almacena 1024 palabras de 64 bits con lo que el bus de direcciones de esta memoria tiene 10 bits. El tipo de memoria considerada es del tipo ROM, desde que las instrucciones no se modifican. La memoria implementada dentro del FPGA se conoce como memoria embebida, la cual usa una señal de reloj de 27 MHz, y el contenido de la memoria de instrucciones se puede cambiar cada vez que se quiera ejecutar un nuevo programa.

La memoria de instrucciones está organizada en dos bloques de 1024 palabras de 32 bits. Una de las memorias almacena los 32 bits de instrucción menos significativos (*imemory_low*) mientras que la otra memoria almacena los 32 bits más significativos (*imemory_high*). Las instrucciones de *imemory_low* se ejecutan antes que las que vienen de *imemory_high*. El ordenamiento de cada instrucción es little-endian. Por otro lado, se incluyó un bloque llamado *imemory_select* que sirve para seleccionar la salida de datos de la memoria *imemory_low* o un contenido de 32 bits iguales a "0". Este último caso ocurre para los casos en que una dirección de ramificación (producto de instrucciones del tipo branch) o salto (producto de instrucciones del tipo jump) genera una dirección para la memoria de instrucción cuyos 3 bits menos significativos son "100". Por defecto, las direcciones que se generarán terminan en "000" en los 3 últimos bits del bus de direcciones de la memoria de instrucciones.

### *B. Memoria de Datos*

Este bloque forma parte de la arquitectura Harvard seleccionada, en donde se almacenan los datos que usan las instrucciones. El tipo de memoria considerada es del tipo RAM, donde se podrán leer o escribir datos desde o hacia ella. En esta memoria se almacenan 1024 palabras de 32 bits cada una. La memoria de datos forma parte de la etapa MEM de la cola de instrucciones tipo Load/Store. La memoria implementada dentro del FPGA se conoce como memoria embebida, la cual usa una señal de reloj de 27 MHz, y cuyo contenido se pueda cambiar cada vez que se quiera ejecutar un nuevo programa. El ordenamiento de datos por cada palabra de 32 bits es little-endian.

### *C. Register File*

Este bloque almacena el conjunto de registros internos del tipo entero del CPU. Se consideraron 32 registros de 32 bits c/u para las instrucciones del tipo entero. El registro r0 no puede verse afectado en su contenido y siempre tendrá un contenido de 00000000H. Todos los demás registros se actualizarán con el flanco de subida de la señal de reloj.

Desde que existen tres colas de instrucciones, dos para instrucciones tipo R o I (colas *pipe0* y *pipe1*) y la otra para instrucciones tipo Load/Store (cola *pipe2*), es posible que se presente el caso en que las tres colas existan tres instrucciones que leen dos registros, por lo que el Register File debe permitir que se pueda leer a la vez seis registros (dos por cada instrucción). Adicionalmente, para instrucciones de ramificación (branch) el Register File proporciona el contenido de dos registros y que se usan en la unidad BU.

Por otro lado, el Register File debe ser capaz de actualizar su contenido ante la presencia de tres señales de escritura, proveniente de las tres colas de instrucciones. En el diseño de este bloque se consideró la posibilidad de escrituras simultáneas hacia un mismo registro, con resultados diferentes que vengan de dos o más colas, lo cual debe evitarse. Para esto se usó la Unidad de Detección de Dependencias de Datos (**H**azard **D**etection **U**nit, HDU).

### *D. Unidad Aritmético/Lógica para Instrucciones Tipo R o I*

Este bloque contiene a la Unidad Aritmética y Lógica (ALU) reservada para las instrucciones enteras y relacionadas a las instrucciones de tipo R o I que se puedan implementar. En este caso se implementaron dos ALUs, una para la cola *pipe0* y otra para la cola *pipe1*, para evitar contención por encolamiento de instrucciones. Para mantener las cosas simples, se restringió el juego de instrucciones del CPU, a fin de que las operaciones aritméticas y lógicas puedan realizarse fácilmente en las ALUs. Las instrucciones aritmético/lógicas implementadas, y compatibles con la arquitectura MIPS32 [3] incluyen las siguientes: ADD, ADDU, ADDI, ADDIU, SUB, SUBU, AND, ANDI, OR, ORI, XOR, XORI, NOR, y SLT.

### E. *Unidad Aritmética para Cálculo de Direcciones*

Este bloque contiene a la Unidad Aritmética y Lógica (ALU) para el cálculo de las direcciones de datos de aquellas instrucciones del tipo Load o Store (cola *pipe2*). Esta ALU es sencilla, desde que solamente realiza una operación de suma con dos operandos: el contenido de un registro del Register File (32 bits), y un valor extraído de la instrucción del tipo Load/Store para formar 32 bits. Este bloque genera una señal a la salida de la ALU de 32 bits, de los cuales se usan los 10 menos significativos para direccionar a la memoria de datos.

### F. *Colas FIFO de Instrucciones*

Estos bloques constan de un almacenamiento tipo cola FIFO (**F**irst-**I**n, **F**irst-**O**ut), de tal forma que la primera instrucción que entra será la primera en salir. El diseño incluye dos FIFOs, uno lee instrucciones desde *imemory_low* (*fifo_low*) y el otro desde *imemory_high* (*fifo_high*). Además, se provee un control para el ingreso y retiro de instrucciones en cada cola. El CPU superescalar trabaja como un CPU del tipo que emite y finaliza las instrucciones en orden. Se tratará que las colas FIFO de instrucciones estén ocupadas, a no ser que se ejecute una instrucción de ramificación (branch) o de salto (jump), lo que origina que se vacíen las colas FIFO.

Ambas colas FIFO almacenan hasta cuatro instrucciones. Si las colas FIFO se encontraran vacías, no es necesario esperar cuatro pulsos de reloj para que el primer par de instrucciones aparezca a la salida de los FIFOs, lo cual sería ineficiente. Esto se evita con una implementación conocida como "first word fall through", y al leerse el primer par de instrucciones desde las memorias embebidas *imemory_low* e *imemory_high*, éstas aparecen a la salida de ambos FIFOs en un único pulso de reloj. Las siguientes instrucciones leídas desde *imemory_low* e *imemory_high* se colocarán a continuación del primer par de instrucciones colocadas en los FIFOs.

### G. *Unidad de Búsqueda de Instrucciones*

Esta unidad se encarga de realizar la búsqueda de dos instrucciones que residen en la memoria de instrucciones. Cada instrucción se ubica en una posición de memoria que es múltiplo de 4 (instrucciones alineadas) y con un ordenamiento de datos little-endian. El PC (que es el registro para direccionar a la memoria de instrucciones) debiera incrementarse de manera normal en 4, pero al ser este CPU del tipo superescalar de 2 vías, el incremento normal es de 8.

Esta unidad genera los 10 bits que se usan para direccionar a las memorias *imemory_low* e *imemory_high*, los cuales se mapean con los bits 12 a 3 del PC[31..0]. Por otro lado, esta unidad recibe las señales branch y jump, que se usan para ramificar o saltar, consiguiendo que el PC[31..0] se modifique, siempre que branch o jump sean "1" (condición verdad). Además, esta unidad recibe las señales *push_fifo_low* y *push_fifo_high* que se utilizan para controlar el incremento del valor del PC, si será +8 o +4 o se mantiene en su valor. Por otro lado, las señales de entrada *full_fifo_low* y *full_fifo_high* indican si las colas FIFO de instrucciones están llenas ("1") o no ("0"). Si alguna de estas últimas señales es "1", el valor del PC se mantiene en su último valor.

### H. *Unidad de Despacho de Instrucciones (DU)*

Una vez que se tienen dos instrucciones a las salidas de las colas FIFO (*fifo_high* y *fifo_low*), debe de tomarse la decisión a cual cola de instrucciones debe enviarse, bien sea la cola *pipe0*, *pipe1*, o *pipe2*. La decisión estará basada básicamente en los códigos de operación que se extraen de las salidas de las colas FIFO *fifo_low* y *fifo_high*. Debe de contemplarse todas las posibilidades de combinación de dos instrucciones que salen de las colas FIFO. De no haber dependencias de datos entre dos instrucciones tipo R o I, la que viene de la cola FIFO *fifo_low* se encamina a la cola *pipe0* y la otra a la cola *pipe1*. Si llegan 2 instrucciones tipo Load/Store, la que viene por la cola FIFO *fifo_low* se encamina a la cola *pipe2* y la otra espera. Si la instrucción que viene de la cola FIFO *fifo_low* es una instrucción tipo R o I, y la otra es tipo Load/Store, la de la cola FIFO *fifo_low* se encamina a la cola *pipe0*, y la otra a la cola *pipe2*. De haber dependencias de datos (RAW o WAW), alguna de las colas se congela, con lo que no necesariamente dos instrucciones se encaminan hacia las colas.

### I. *Cola pipe0 para Instrucciones Tipo R o I*

La cola *pipe0* consta de un pipeline para la ejecución de instrucciones aritméticas y/o lógicas del tipo R o I. A partir del contenido de la instrucción que recibe esta cola se identifican dos registros fuentes (instrucción tipo R), o un registro fuente y un dato inmediato (instrucción tipo I), y para ambos casos, un registro destino. Una instrucción del tipo R o I necesitará pasar por tres etapas para realizar la escritura en un registro, las que se consideran a continuación: lectura de registros (etapa ID), cálculo en ALU (etapa EXE), y escritura en registro destino (etapa WB). La Fig. 2 muestra las señales entrantes y salientes de la cola *pipe0*.

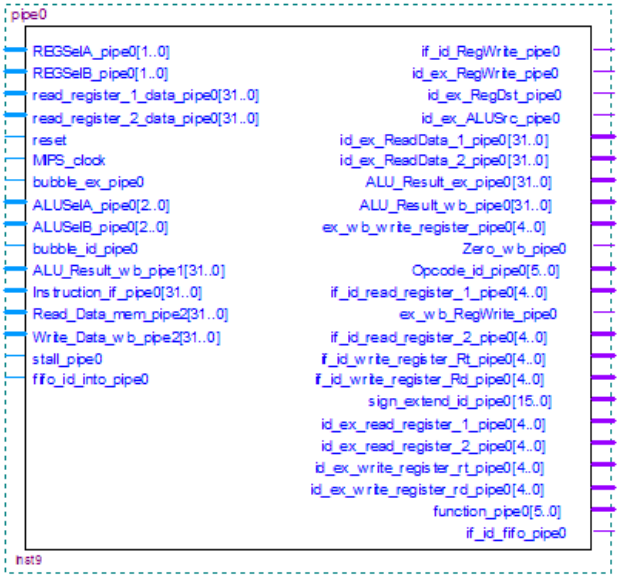


Fig. 2. Entradas y salidas de la cola pipe0 para instrucciones tipo R o I del CPU RISC superescalar. Incluye las etapas pipeline ID, EXE y WB.

Para esto, entre etapa y etapa se coloca un registro de interfaz, con lo que son necesarios 3 pulsos de reloj para que

el resultado de una instrucción que entra a esta cola se escriba en un registro destino.

### J. *Cola pipe1 para Instrucciones Tipo R o I*

La cola *pipe1* tiene la misma estructura básica que la cola *pipe0*. Las variaciones vienen como consecuencia de las dependencias de datos del tipo RAW y que se resuelven vía adelanto de datos por medio de la unidad FU. Los cambios se deben a que la ejecución de las instrucciones en el CPU superescalar es en orden, por lo que siempre las instrucciones del tipo R o I que se ejecuten en el *pipe0* son anteriores a las que se ejecuten en el *pipe1*. La Fig. 3 muestra las señales entrantes y salientes de la cola *pipe1*.

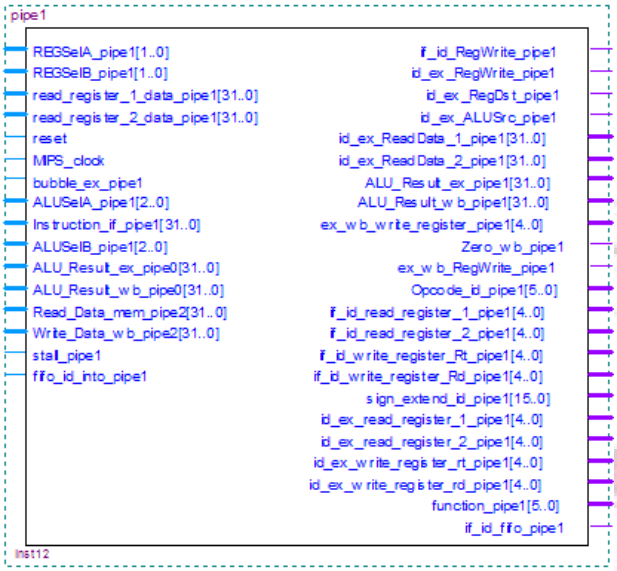


Fig. 3. Entradas y salidas de la cola pipe1 para instrucciones tipo R o I del CPU RISC superescalar. Incluye las etapas pipeline ID, EXE y WB.

### K. *Cola pipe2 para Instrucciones Tipo Load/Store*

La cola *pipe2* consta de un pipeline que permite la ejecución de instrucciones tipo Load o Store exclusivamente. Una instrucción del tipo Load necesitará pasar por cuatro etapas para realizar su cometido en etapa WB, las que se consideran a continuación: lectura de registros (de dirección y destino de datos, etapa ID), cálculo de dirección de memoria en ALU (etapa EXE), lectura de operando de memoria (etapa MEM), y escritura en registro destino (etapa WB).

Por otro lado, una instrucción del tipo Store necesitará pasar por tres etapas para realizar su cometido, es decir escribir en memoria, las que se consideran a continuación: lectura de registros (uno de dirección y el otro de datos, etapa ID), cálculo de dirección de memoria en ALU (etapa EXE), y escritura de registro hacia memoria (etapa MEM).

Entre etapa y etapa se coloca un registro de interfaz, con lo que serán necesarios cuatro pulsos de reloj para una instrucción Load y tres para una instrucción Store.

Las instrucciones implementadas, y compatibles con la arquitectura MIPS32 [3] son LW y SW. La Fig. 4 muestra las señales entrantes y salientes de la cola *pipe2*.

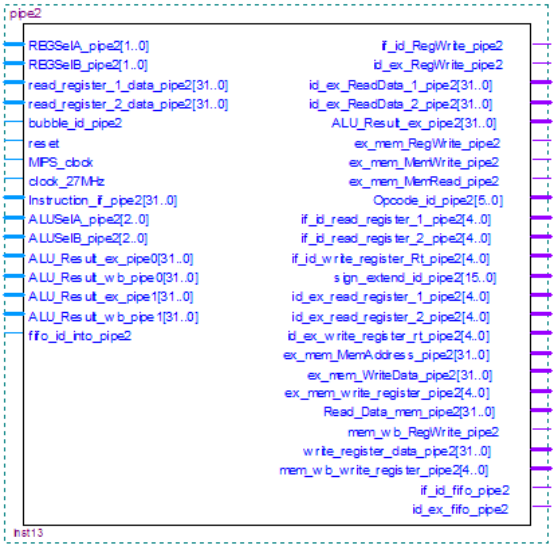


Fig. 4. Entradas y salidas de cola pipe2 para instrucciones tipo Load/Store del CPU RISC superescalar. Incluye las etapas pipeline ID, EXE, MEM y WB.

### L. *Unidad de Detección de Dependencias de Datos (HDU)*

La Unidad HDU es mucho más compleja que aquella para un CPU escalar, por la cantidad de casos y simultaneidad de los mismos. La detección de una dependencia se hace evaluando cuando una instrucción entra en su etapa ID y ésta depende de otra instrucción que se encuentra en la misma etapa o en etapas posteriores (EXE, MEM o WB) del mismo pipeline o de los otros. De haber una dependencia, se deberá congelar la instrucción en la etapa ID que tiene problemas (cuando no pueda ser resuelta por adelanto de datos) tantos pulsos de reloj sean necesarios hasta resolver la dependencia.

Como se cuenta con tres colas de instrucciones, dos para instrucciones tipo R o I (*pipe0* y *pipe1*), y una para instrucciones tipo Load/Store (*pipe2*), las dependencias de datos tipo RAW en las colas *pipe0* y *pipe1* pueden resolverse por técnicas de adelanto de dato. Las dependencias tipo RAW en la cola tipo R o I por instrucciones en la cola Load/Store ocurre cuando la instrucción tipo R o I está detrás de la instrucción tipo Load/Store (una instrucción independiente en el flujo de instrucciones está en la otra cola tipo R o I), y se deberá resolver por un congelamiento de la cola tipo R o I.

Por otro lado, para dependencias tipo WAW entre las colas *pipe0* y *pipe1* pueden ocasionar congelamiento de la cola *pipe1* de no poder resolverse la dependencia por adelanto de datos. Asimismo, para dependencias tipo WAW entre instrucciones en las colas *pipe0* o *pipe1* que siguen a instrucciones en la cola *pipe2* debe resolverse por congelamiento de las colas *pipe0* o *pipe1*.

### M. *Unidad de Predicción de Saltos (BU)*

Existen varias posibilidades para la detección temprana de las instrucciones de ramificación, las que incluyen: detección paralela, detección anticipada, y detección integrada en la captación [6]. Para propósitos del proyecto se implementó una solución que fusiona las dos primeras opciones.

Las alternativas para resolver las instrucciones de

ramificación condicional incluyen las siguientes: bloqueo de la instrucción de ramificación, procesamiento especulativo de la instrucción de ramificación, y ejecución de múltiples caminos [6]. Para fines del proyecto, se implementó la segunda alternativa con ciertas restricciones.

En lo que respecta a la predicción de salto, se tienen las siguientes opciones: predicción fija, predicción estática, y predicción dinámica [6]. La opción implementada para el proyecto estuvo basada en una predicción dinámica, basada en el historial de la instrucción de ramificación, aunque con ciertas restricciones. La historia de saltos ejecutados de las instrucciones de ramificación se almacena en una memoria. En este caso se escogió guardar una historia de las tres últimas veces que una instrucción de ramificación se leyó. La historia de ramificaciones se almacena en la tabla de historia de saltos o **B**ranch **H**istory **T**able (BHT). Cada vez que un salto se toma, se modifica el dato de la BHT de la dirección de la instrucción de ramificación, haciendo un desplazamiento a la izquierda introduciendo un "1". Por ejemplo, si el dato original era "000", ahora tendrá "001", si era "010", ahora será "101". Si el salto no se toma se desplaza un bit igual a "0".

Para efectos de mejorar la predicción de salto, se hace uso de una máquina de estados con dos bits tipo contador saturado, como se muestra en la Fig. 5. Cada vez que nos encontremos con una instrucción de ramificación condicional, la máquina de estados predecirá si el salto se toma o no. Al inicio, para todos los saltos se asume el estado "Not Taken Weak" (NTW). En función de la historia de saltos de cada instrucción, la máquina de estados predice el salto.

Cada estado de la máquina se codificará con dos bits. Para que la predicción no se base solamente en el estado presente de la máquina, lo cual sería común a todas las instrucciones de ramificación, se usa una tabla de patrones de salto que almacena el estado de la máquina por cada instrucción de este tipo. El bloque es conocido como el **P**attern **H**istory **T**able (PHT). El conjunto de los bloques BHT, PHT (implementados en memoria embebida del FPGA) y máquina de estado constituye lo que se conoce como una unidad de predicción dinámica de saltos de dos niveles usando historia local, o PAg (**P**er-address **A**daptive Branch Prediction using one **g**lobal pattern history table) [13]. Este esquema proporciona un nivel de acierto en la predicción de salto de al menos 90% [14].

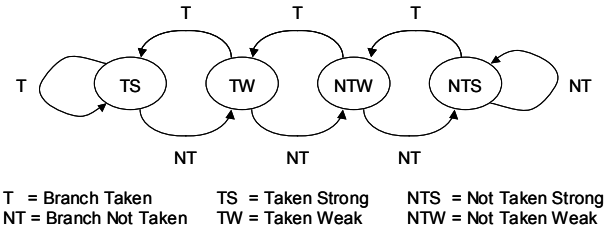


Fig. 5. Máquina de estados para predicción de saltos, la cual forma parte de la Unidad de Predicción de Saltos (BU).

Como las instrucciones de ramificación trabajan sobre registros, la unidad BU debe leer el contenido de dos registros desde el File Register. Pero si hay un registro que se va a modificar a futuro y cuyo nuevo valor está en algunas de las etapas EXE o WB de las colas *pipe0* o *pipe1*, o en las etapas MEM o WB de la cola *pipe2*, estos valores deben ser adelantados hacia la unidad BU. En este caso, se establece una prioridad, bajo el criterio de retiro y ejecución en orden de las instrucciones, ya que un nuevo valor de registro puede estar circulando en mas de una cola de instrucciones.

Además, la unidad BU genera las direcciones de destino en caso la instrucción que se detecta es de salto. Las instrucciones de ramificación implementadas en el diseño, y compatibles con la arquitectura MIPS32 [3] son BEQ, BNE, BLEZ, y BGTZ, mientras que las instrucciones de salto implementadas de la misma arquitectura son J y JAL. La Fig. 6 muestra las señales entrantes y salientes de la unidad BU.

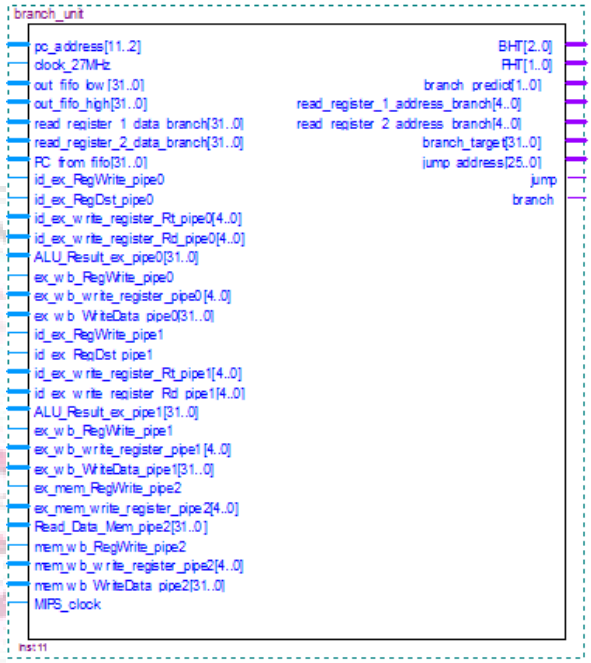


Fig. 6. Entradas y salidas de la Unidad de Predicción de Saltos (BU) del CPU RISC superescalar. Incluye a los bloques BHT, PHT, y máquina de estados de predicción de saltos.

### *N. Unidad de Adelanto de Datos (FU)*

La Unidad FU debe permitir el adelanto de datos dentro de una cola de instrucciones, y también entre colas. Esta unidad sirve para resolver los casos de dependencia de datos del tipo RAW, recibiendo las siguientes señales de datos: los números de registros a ser leidos del Register File para las instrucciones en las colas *pipe0*, *pipe1* y *pipe2*, tanto en la etapa ID como en la etapa EXE. También recibe los números de los registros a ser actualizados para las colas *pipe0* y *pipe1*, cuando las instrucciones en estas colas están tanto en la etapa EXE como en la etapa WB. Asimismo, recibe al número de registro destino en la cola *pipe2*, cuando la instrucción se encuentra en la etapa MEM o WB.

Igualmente, la unidad FU recibe las siguientes señales de control: comando de escritura en la etapa EXE y WB para las instrucciones de las colas *pipe0* y *pipe1*, así como las señales de comando de escritura en las etapas MEM y WB para las instrucciones de la cola *pipe2*.

Las señales de salida de la unidad FU incluyen las siguientes: ALSelA, ALUSelB, para seleccionar el valor a ser adelantado en las etapas EXE de las colas *pipe0*, *pipe1* y *pipe2*. Las salidas REGSelA y REGSelB sirven para adelantar datos hacia las etapas ID en las colas *pipe0*, *pipe1* y *pipe2*. Como en cada instrucción en la etapa ID se pueden leer hasta dos registros, se puede adelantar hacia esta etapa dos valores.

### *O. Unidades de Interfaz con el Usuario*

Las unidades que nos permitan una adecuada interfaz con el usuario incluyen los siguientes: unidad anti-rebotes para ingreso de comandos vía teclas, unidad de caracteres a visualizar en pantalla VGA, unidad para generar señales de sincronismo de video VGA, unidad para separar señales de video de texto, unidad para evaluar el contenido de memorias embebidas, y la unidad para visualizar señales internas del CPU hacia video VGA. La resolución de la señal de video VGA es de 1024x768 píxeles (horizontal x vertical). Para la generación de la señal VGA de una resolución de 1024x768 se necesitó una frecuencia de reloj de 65 MHz. Una vista parcial del estado de las señales de interés del CPU visualizadas a través de una pantalla VGA se muestra en la Fig. 7.

## IV. Resultados Experimentales

### *A. Implementación de prototipo en FPGA*

La implementación del CPU RISC superescalar se realizó sobre el dispositivo FPGA EP2C35F672C6 de la familia Cyclone II de Altera, usando para esto la tarjeta de desarrollo DE2 de Altera [16], [17]. La Fig. 8 muestra una vista panorámica de la implementación del prototipo del CPU superescalar. La tarjeta DE2 se conecta a una pantalla VGA para poder visualizar las señales de mayor interés del diseño. Adjunto a la tarjeta DE2 se visualiza una laptop donde se muestra el entorno gráfico de entrada de diseño de la herramienta Quartus II de Altera [15].

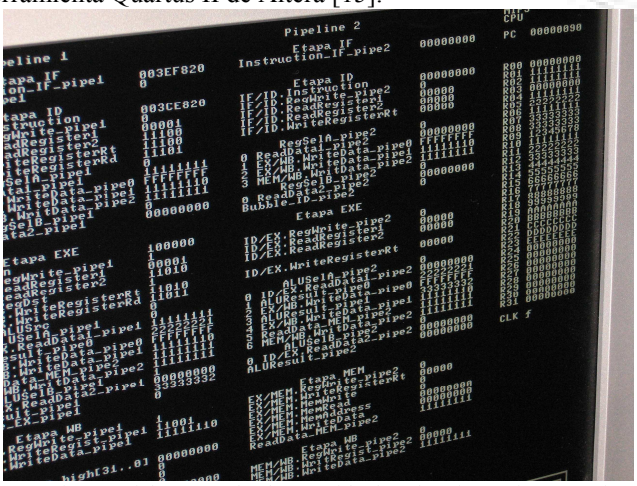

Fig. 7. Vista parcial mostrando el contenido de colas pipe1, pipe2 y registros del Register File del CPU RISC superescalar en una pantalla VGA.

### *B. Resultados de la Implementación del diseño en FPGA*

La Tabla I muestra un resumen de los recursos empleados en la implementación del CPU superescalar en el dispositivo FPGA EP2C35F672C6 de Altera, mientras que la Fig. 9 muestra una vista de cómo se han dispuesto los recursos empleados físicamente al interior del FPGA. De la Tabla I y Fig. 9, puede apreciarse que el diseño del CPU RISC superescalar es bastante denso. El CPU recibe una frecuencia de reloj de 27 MHz para las memorias embebidas, mientras que el reloj para el resto de unidades es manual. El consumo de potencia de la implementación del prototipo es de 3.7 W.

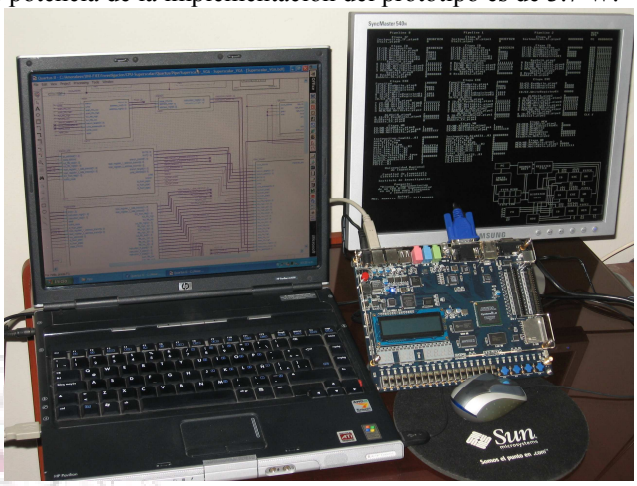

Fig. 8. Vista panorámica del prototipo de CPU RISC superescalar de dos vías en FPGA, implementado en la tarjeta de desarrollo DE2 de Altera y con visualización a pantalla VGA.

TABLA I
Resultados de implementación del CPU superescalar en FPGA

| Recursos | Utilización |
|---|---|
| Total de Elementos Lógicos (LE) | 16,447 / 33,216 (50 %) |
| Total de Memoria Embebida (bits) | 286,976 / 483,840 (59 %) |
| Total de I/O pins | 42 / 475 (9 %) |
| Total de PLLs | 1 / 4 (25 %) |


Fig. 9. Vista de planta (floorplanning) de la implementación del CPU RISC superescalar en FPGA EP2C35F672C6 de Altera.

La Fig. 10 muestra los resultados de simulación del esquema de predición seleccionada para el diseño (PAg), comparados con el uso de un simple predictor basado en una máquina de estados de 2 bits (2bit). Las pruebas fueron las siguientes: ordenación de 100 valores usando el método de la

burbuja (Bubble), generación de secuencias Hailstone de los primeros 1000 enteros positivos (Hail), y conversión de datos de 100 números enteros a su representación binaria (Conv).

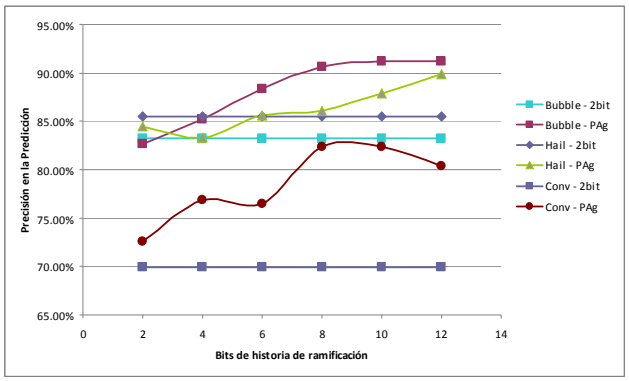


Fig. 10. Resultados de simulacion de la precisión en la predicción de ramificaciones para diferentes pruebas.

El esquema PAg es mas preciso con respecto al esquema 2bit sobretodo cuando se usan mas bits de historia de ramificación. Cabe mencionar que la implementación del bloque BU solo incluye 3 bits de historia por la limitación de tamaño de la memoria embebida del FPGA seleccionado.

## V. Conclusiones

Fue posible implementar un diseño básico de un CPU RISC superescalar de 32 bits de 2 vías en un FPGA. El prototipo desarrollado es adecuado para la enseñanza, desde que el diseño del sistema se hizo de manera modular, de manera que sea fácil una modificación e integración de los archivos de diseño. Igualmente, se provee visualización de una gran cantidad de señales hacia monitor VGA para depuración y verificación funcional de cada bloque del diseño. La capacidad ocupada de recursos del dispositivo FPGA seleccionado es considerable (ver Tabla I), por lo que para futuras modificaciones y/o ampliaciones debe pensarse en cambiar a un dispositivo de mayores prestaciones. La implementación del CPU RISC superescalar en FPGA no pretende ser una competencia a procesadores superescalares comerciales, desde que estos últimos tienen mayores recursos de hardware (mayor profundidad de pipelines, mayor grado de paralelismo, ejecución fuera de orden, memorias cache de varios niveles, soporte de instrucciones de punto flotante, entre otras características). A pesar de las restricciones mencionadas, el proyecto implementado sirve de incentivo para desarrollo de procesadores a medida sobre FPGAs. Posibles proyectos incluyen: desarrollo de CPUs multicore, computación multithread, CPUs configurables.

## VI. Trabajos Futuros

Dentro de las mejoras al proyecto, se incluyen (pero no se limitan) a las siguientes: ampliación del repertorio de instrucciones del tipo entero a soportar, inclusión de soporte a instrucciones de punto flotante, mejora en la implementación hardware para la predicción de saltos, aumento en el grado de paralelismo de lectura y ejecución de instrucciones.

## Referencias


[1] Xilinx Inc., MicroBlaze Processor Reference Guide - EDK 13.1, 2011: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/mb_ref_guide.pdf

[2] Altera Corporation, Nios II Processor Reference Handbook, 2010: http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf

[3] MIPS32 Architecture for Programmers Volume II - The MIPS32 Instruction Set, Rev 2.50, MIPS Technologies, Inc., USA, 2005.

[4] D. Patterson, J. Hennessy, Computer Organization and Design: The Hardware/Software Interface, 4th Edition, Morgan Kaufmann Publishers, USA, 2009, pp. 330-384.

[5] J. Hennessy, D. Patterson, Computer Architecture: A quantitative Approach, 4th Edition, Morgan Kaufmann Publishers, USA, 2007, pp. A-1 – A-37.

[6] J. Ortega, M. Anguita, A. Prieto, Arquitectura de Computadores, Thomson, 2005, pp. 101-224.

[7] J. P. Shen, M. H. Lipasti, Arquitectura de Computadores: Fundamentos de los procesadores superescalares, Mc Graw Hill, 2005, pp. 179-283.

[8] J. D. Luker, V.B. Prasad, "RISC system in an FPGA", *Proceedings of the 44th IEEE 2001 Midwest Symposium on Circuits and Systems*, MWSCAS 2001, pp. 532-536 vol.2, 2001.

[9] M. Gschwind,V. Salapura, D. Maurer, "FPGA Prototyping of a RISC Processor Core for Embedded Applications", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 9, No. 2, pp. 241-250, Abr 2001.

[10] P. S. Mane, I. Gupta, M. K. Vasantha, "Implementation of RISC Processor on FPGA", *IEEE International Conference on Industrial Technology*, ICIT 2006, pp. 2096-2100, 2006.

[11] N. Joseph, S. Sabarinath, K. Sankarapandiammal, "FPGA based Implementation of High Performance Architectural level Low Power 32-bit RISC Core", *International Conference on Advances in Recent Technologies in Communication and Computing*, ARTCom '09, pp. 53-57, 2009.

[12] J. O. Hamblen, T. S. Hall, M. D. Furman, Rapid Prototyping of Digital Systems - Quartus II Edition, Springer, 2006, pp. 256-275.

[13] T. Y. Yeh, Y.N. Patt, "Alternative implementations of two-level adaptive branch prediction", in *Proc. 19th Annual International Symposium on Computer Architecture*, Australia, pp. 124–134, 1992.

[14] T. Y. Yeh, Y.N. Patt, "A comparison of dynamic branch predictors that use two levels of branch history", in *Proc. 20th Annual International Symposium on Computer Architecture*, California, pp. 257–266, 1993.

[15] Altera Corporation, "Quartus II software for education", USA: http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html

[16] Altera Corporation, DE2 Development and Education Board, USA: http://www.altera.com/education/univ/materials/boards/de2/unv-de2-board.html

[17] Altera Corporation, "DE2 Development and Education Board User Manual", ver 1.4, USA, 2006: ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf


**Aurelio Morales Villanueva** (M'09) nació en Lima en 1961. Obtuvo el grado de Bachiller en Ciencias con mención en Ingeniería Electrónica en 1985, el título de Ingeniero Electrónico en 1988 y el grado de Maestro en Ciencias con mención en Ingeniería Electrónica en 1991, de la Universidad Nacional de Ingeniería (UNI), en Lima, Perú. Obtuvo el grado de Master of Science en Electrical Engineering de State University of New York, Buffalo, USA en 1994 gracias a una beca Fulbright. Su campo de especialización es ingeniería de computadores, con un énfasis en arquitectura de computadores, sistemas digitales, y lógica reconfigurable.

Actualmente es profesor asociado de la Facultad de Ingeniería Eléctrica y Electrónica de la UNI, y tuvo a su cargo cursos de antegrado y postgrado. Su campo de investigación actual incluye la aplicación de lógica reconfigurable en arquitectura de computadores y procesamiento de señales.

El MSc. Morales Villanueva se encuentra de licencia en la UNI desde Agosto del 2009, y está realizando estudios de doctorado en el Department of Electrical and Computer Engineering de University of Florida, gracias a una beca FINCyT/LASPAU.