Congr. Int. en Ing. Electrónica. Mem. ELECTRO, Vol. 39, pp. 205-210, Oct 2017, Chihuahua, Chih. México
http://electro.itchihuahua.edu.mx/memorias_electro/MemoriaElectro2017.zip
ISSN 1405-2172

Diseño Comportamental de una Unidad Aritmético Lógica de M bits basada en VHDL e Implementada en un FPGA de Xilinx.

Área técnica: Sistemas Digitales y Computación
Jesús E. Valenzuela, Abimael Jiménez, David Chaparro, José Mireles y Ángel Sauceda
Instituto de Ingeniería y Tecnología, Universidad Autónoma de Ciudad Juárez
Av. Del Charro 450, Pronaf, C.P. 32310, Juárez, Chihuahua, Tel. (656) 6882100, Correo: jes.valenzuela1@gmail.com

Resumen

En este artículo se describe el diseño de una Unidad Aritmética Lógica con estructura de árbol de M bits. El diseño se realizó con el lenguaje de descripción de hardware VHDL, las herramientas de diseño Vivado 17 y la implementación en un FPGA Spartan 3E de Xilinx. La arquitectura propuesta de la ALU es parte de un procesador RISC simple. La característica más destacada de la ALU es que se puede escalar fácilmente y que se diseñó con un enfoque modular. La ALU fue simulada, sintetizada y se corroboraron todas las operaciones aritméticas, lógicas y banderas en un FPGA. Los resultados nos indican que, por un lado, es posible realizar la síntesis física de la ALU para diseñarse en un circuito integrado VLSI y; por otro lado, es una arquitectura factible para su implementación en un procesador tipo RISC.

1. Introducción

El cómputo reconfigurable es la arquitectura de computadoras, donde se asocia la flexibilidad del software con el alto rendimiento en hardware mediante el procesamiento de dispositivos electrónicos versátiles de alta velocidad como los arreglos de compuertas programables de campo o FPGAs (Field-Programmable Gate Arrays).

La importancia del procesador reconfigurable en el diseño de sistemas embebidos ha aumentado considerablemente durante las últimas décadas. Debido a la gran evolución de la tecnología FPGA, se ha alcanzado un punto donde la arquitectura del procesador se puede modificar mediante software [1]. La diferencia principal entre el procesador reconfigurable y los microprocesadores convencionales es la posibilidad de realizar cambios significativos en el propio datapath, además del control de flujo [2].

En arquitectura computacional, RISC (Reduced Instruction Set Computer) es una técnica de diseño utilizada para minimizar la cantidad de área requerida, la complejidad del conjunto de instrucciones, el ciclo de instrucción y el costo del diseño. Con el creciente desarrollo de la tecnología del silicio y la reducción del costo del circuito integrado, el uso del procesador RISC está aumentando ampliamente en todos los campos [3]. Las operaciones de carga y almacenamiento sólo se utilizan para acceder a la memoria en el procesador RISC. Las otras operaciones se realizan de registro a registro. Esta característica hace más claro el diseño del conjunto de instrucciones, ya que permite la ejecución de instrucciones a una velocidad de instrucción por ciclo. El cálculo rápido de direcciones de los operandos es posible debido a la transparencia en los modos de direccionamiento [4-5].

VHDL es un lenguaje utilizado en la industria electrónica para describir circuitos digitales. Este acrónimo se reconoce como la combinación de VHSIC (Very High Speed Integrated Circuit) y HDL (Hardware Description Language) [6]. VHDL es una herramienta muy útil con concurrencia que hace frente al paralelismo del hardware digital.

La unidad de aritmética-lógica ALU (Aritmetic-Logic Unit) es un circuito lógico

exclusivamente combinacional. En la Figura 1 se puede observar el símbolo electrónico de la ALU de M bits implementada en este trabajo. El diseño de la ALU en VHDL es de importancia significativa en sistemas digitales, ya que es una parte integral de la unidad central de procesamiento (Central Process Unit, CPU). La ALU es capaz de calcular los resultados de una amplia variedad de operaciones básicas aritméticas y lógicas [7-8]. Como se observa en la Figura 1, la ALU toma como entrada los datos A y B (llamados operandos) y un código de la unidad de control que indica la operación a realizar [9]. La salida *S* es el resultado del cálculo conjuntamente con las banderas de sobreflujo o overflow (O), cero (Z), negativo (N) y acarreo (C).

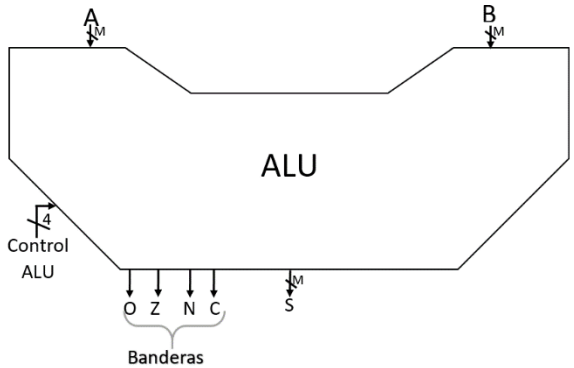

Figura1. Símbolo de una ALU de *M* bits con banderas de sobreflujo (O), cero (Z), negativo (N) y acarreo (C).

En este trabajo, se propone el diseño de una ALU de M bits con estructura de árbol como elemento principal del diseño de un procesador RISC simple. En las siguientes secciones se describe la metodología utilizada en el diseño comportamental de la ALU. Así como los resultados obtenidos en la simulación, síntesis e implementación de la arquitectura en un FPGA. Finalmente, se presentan las principales conclusiones del trabajo.

2. Metodología de diseño de la ALU

El diagrama esquemático completo de la ALU de *M* bits con cada uno de los módulos (codificados en VHDL) que la conforman, se muestra en la Figura 2. En la Tabla 1 se muestran las 16 operaciones aritmético-lógicas que realiza la ALU de M bits con la activación correspondiente de las banderas.

Tabla1. Lista de las 16 operaciones aritmético-lógicas de la ALU de M bits.

CtrlALU	Operación	Banderas
0000	No operación	O, Z, N, C
0001	A + B	O, Z, N, C
0010	A + (-B)	O, Z, N, C
0011	A + 1	O, Z, N, C
0100	A + (-1)	O, Z, N, C
0101	B + 1	O, Z, N, C
0110	B + (-1)	O, Z, N, C
0111	Operación AND	Z, N
1000	Operación OR	Z, N
1001	Operación NOT	Z, N
1010	Desplazamiento Izquierda	Z, N, C
1011	Rotación a la derecha	Z, N
1100	Salida igual a A	Z, N
1101	Salida igual a B	Z, N
1110	Salida igual a 0	Z
1111	Salida igual a 1	N, C

Además de las operaciones mostradas en la Tabla 1, la ALU considera la función “no operación”, en la cual la salida toma el valor de 0, pero sin modificar el valor de las banderas en la operación anterior.

2.1. Operaciones lógicas

Este bloque realiza varias operaciones lógicas. En la Tabla 1 y la Figura 2, se observa que la ALU realiza las tres operaciones lógicas básicas (AND, OR y NOT). Estas funciones lógicas se realizan entre los operandos de entrada *A* y *B*, excepto en la operación lógica NOT que se aplica únicamente al operando *A*. En este grupo también se encuentran operaciones que permiten transferir el dato de la entrada *A* o *B* a la salida *S* o cambiar todos los bits de *S* por unos o ceros lógicos. Así como operaciones para desplazar los datos a la izquierda o a la derecha. El desplazamiento a la izquierda mueve cada bit una posición

a la izquierda y el bit más significativo pasa como bit de acarreo y la posición del bit menos significativo se hace 0 lógico. Por otro lado, en el desplazamiento a la derecha, el bit menos significativo pasa a la posición del bit más significativo.

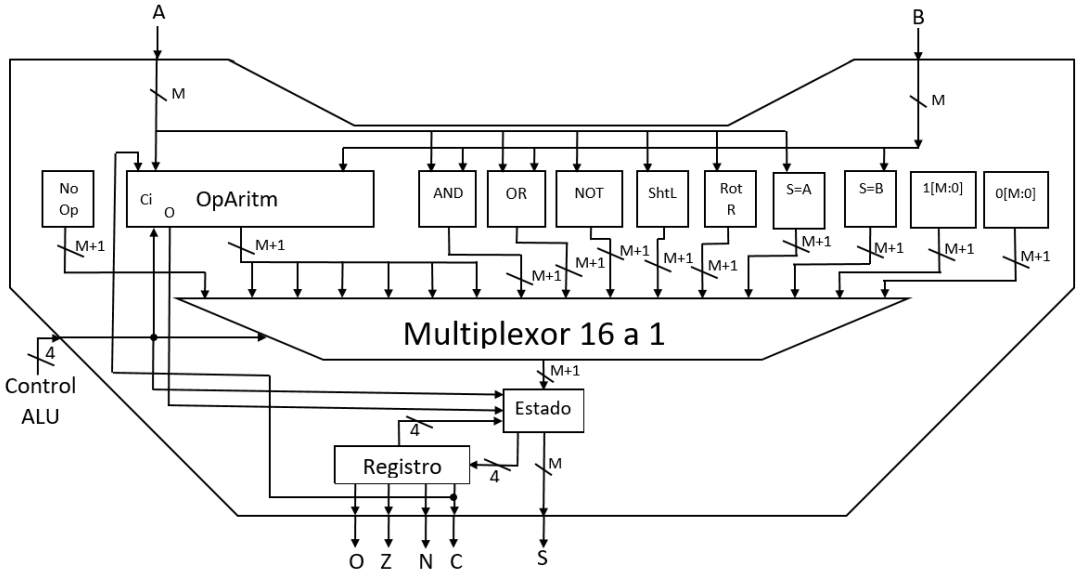

Figura 2. Diagrama completo de la ALU de *M* bits, donde se observan los módulos de las operaciones lógicas, el módulo de operaciones aritméticas, el multiplexor 16 a 1 y los registros donde se almacena el valor de las banderas O, Z, N, C.

2.2. Operaciones aritméticas

El módulo *OpAritm* de la Figura 2 corresponde a la unidad aritmética y se encarga de realizar las operaciones aritméticas: A+B, A-B, A+1, A-1, B+1, B-1 (ver Tabla 1). Debido a que todas estas operaciones utilizan el sumador completo como módulo básico; su arquitectura está formada por 4 bloques como se muestra en la Figura 3. En el bloque *SelOperaciones* se seleccionan los operandos, en el bloque *Ca2n* se complementa a 2 alguno de los operandos si es necesario, en el bloque *FAn* se realiza la suma y; finalmente, en el bloque *OverFlow* se genera la bandera de sobreflujo.

El bloque *SelOperaciones* recibe el código de la unidad de control correspondiente a la operación a realizar. En función de este código se seleccionarán los operandos de entrada *A* y *B* o enviará un 1.

Después de la selección de operandos, uno de ellos pasará directamente al sumador completo; mientras que, el otro pasará al bloque *Ca2n.* Dependiendo de la señal de habilitación del bloque *SelOperaciones* se determinará si el bloque *Ca2n* calcula el complemento a 2 o no.

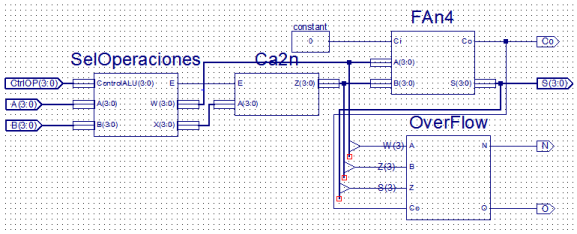

Figura 3 Diagrama esquemático de la unidad aritmética utilizada en la ALU.

Una característica importante de la ALU es que es genérica, es decir, se puede cambiar la cantidad de bits que puede manejar. Esto se logra a través de la instanciación de componentes, donde se utiliza la instrucción concurrente *port map* para conectar M elementos o bloques en un circuito. En el siguiente código de VHDL se observa cómo se utiliza un ciclo *for* para

realizar todas las conexiones de los sumadores completos. Como resultado, componente por componente es conectado hasta que el contador llega a M-1. Con *port map* se instancian directamente las entradas y salidas de los componentes. Mientras que, la instrucción *generate* repite la estructura las veces que sea necesario.

for *Contador in 0 to M-1* ***generate***
begin
FullAdder:FA ***port map****(A(Contador), B(Contador), Conexiones(Contador), S(Contador), Conexiones(Contador+1));*
end ***generate****;*

De esta manera se logra concatenar *M* sumadores completos como se muestra en la Figura 4. Por lo tanto, la cantidad de bits para implementar la suma binaria se puede incrementar o reducir fácilmente con el esquema adoptado.

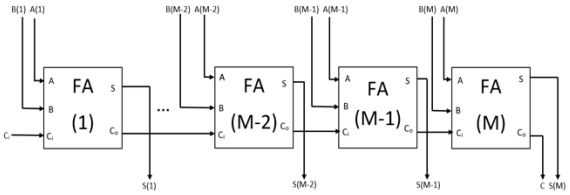

Figura 4. Instanciación de M sumadores completos obtenida con el uso de *generate* y *port map*.

2.3. Complemento a 2

La conversión de los operandos a complemento a 2 se realizó en el bloque *Ca2n* de la Figura 3. Este módulo contiene M bloques de 1 bit como se observa en la Figura 5. Cada bloque *Ca2* tiene 3 entradas (*A*, *B* y *H*) y dos salidas (*Y*, *Co*). Si la entrada *H*=1, entonces el bloque *Ca2n* se comportará como un buffer pasando el valor del operando *A* a la salida *Y* directamente. Si *H*=0, entonces *Ca2n* realizará el complemento a 2 del operando A.

El complemento a 2 se realiza complementando el número y sumándole 1, posteriormente. En cada bloque *Ca2* de la Figura 5 se complementa el dato de la entrada A, se le suma el dato de la entrada B, el resultado se obtiene en *Y* y el acarreo (si se genera alguno) se obtiene en la salida Co. A la entrada B del bloque *Ca2*(0), correspondiente al bit menos significativo, se le asigna el valor de 1 lógico. Todas las entradas *B* restantes son conectadas a la salida *Co* del bloque anterior, como se observa en la Figura 5.

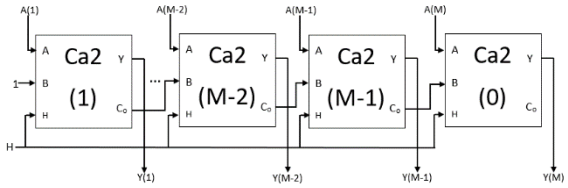

Figura 5. Módulo *Ca2n* de *M* bits utilizado para obtener el complemento a 2.

2.4. Banderas

Las banderas fueron diseñadas para indicar el estado de un resultado, debido a las limitantes que tiene la ALU. Por ejemplo, generar un acarreo (C) o un sobreflujo (O), un resultado negativo (N) o un resultado igual a cero (Z). Por lo tanto, es necesario contar con banderas que indiquen qué situaciones ocurrieron al realizar cierta operación para que sea reportada en la unidad de control.

El acarreo (C) se genera al realizar la suma y que el resultado sea mayor a M bits. El sobreflujo (O) ocurre cuando dos operandos del mismo signo dan como resultado un número con signo contrario. Es posible detectar el sobreflujo evaluando los bits más significativos de los operandos, el bit más significativo del resultado y el bit de acarreo. Si los bits más significativos de los operandos son iguales y el bit más significativo del resultado es diferente al bit de C; quiere decir que hay un sobreflujo y O=1.

La bandera de negativo (N) se activa siempre y cuando el bit más significativo sea igual a 1. Mientras que, la bandera de cero (Z) se activa cuando todos los bits del resultado son cero.

2.5. Multiplexor de 16 a 1

Todas las operaciones son realizadas al mismo tiempo de manera concurrente; por lo tanto, se utiliza un multiplexor 16 a 1 con entradas de *M* bits. Después de realizar una operación aritmética o lógica, en función del código de la unidad de control; el resultado se enviará al bloque *Estado* (ver Figura 2), el cual generará las banderas correspondientes. Éstas son almacenadas en un registro para después enviarlas junto con el resultado a la salida *S* de la ALU.

3. Resultados

Se definió un módulo esquemático de nivel superior que conecta todos los módulos del diseño en la lógica de transferencia de registros (Register Transfer Logic, RTL) [9,10]. Una vez identificados las entradas, salidas y módulos del sistema, las descripciones en VHDL se implementaron a través del simulador Isim de Vivado. Finalmente, el diseño se implementó en la tarjeta FPGA para su verificación [11,12].

Debido a las características de la tarjeta FPGA utilizada. En el diseño de la ALU se definió con *M*=4 para obtener una ALU de 4 bits. A continuación, se analizan 8 casos resultados específicos que se simularon con Isim y se comprobaron en la tarjeta FPGA Basys 2.

En la Figura 6 se muestran los primeros 4 casos y se observa que el valor de los operandos A y B se mantienen constantes todo el tiempo y solo cambia la operación de la ALU. En el primer caso (de izquierda a derecha) el control de la ALU (*CtrlALU*) muestra el valor 0001, el cual corresponde a la suma de 1001 + 1010 como el resultado excede la cantidad de bits se activan las banderas O y C con 1 lógico. En el segundo caso *CtrlALU*=0000 corresponde a la instrucción “no operación”, por lo que la salida será igual a 0 y las banderas conservan el valor del estado anterior. En el tercer caso *CtrlALU*=0111; por lo tanto, se realiza la operación lógica AND. En este caso solo se activa la bandera N debido a que esta bandera toma el valor del bit más significativo. En el cuarto caso CtrlALU=1001, lo que corresponde a la operación lógica NOT. Al realizarse esta operación no se debe activar ninguna bandera.

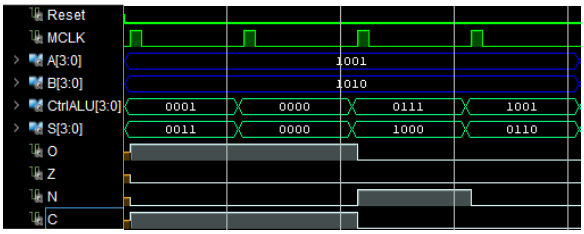

Figura 6. Resultados de simulación de la ALU en la herramienta Isim de Vivado. Los resultados mostrados son del caso 1 al 4.

En la Figura 7 se muestran los casos 5 a 8. A diferencia de los primeros 4 casos ahora el valor de los operandos A y B es el mismo “0101” y las operaciones de la ALU se han cambiado. En quinto caso *CtrlALU*=0010 que corresponde a la resta aritmética y como resultado se han activado las banderas Z y C. En el sexto caso *CtrlALU*=1000, lo que corresponde a la operación lógica OR y no se debe activar ninguna bandera.

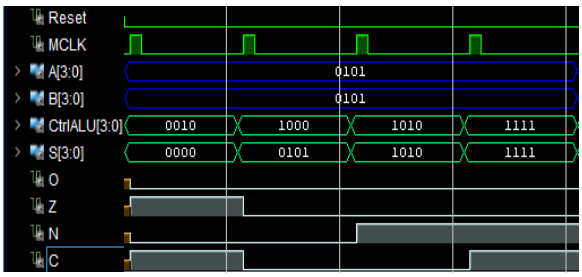

Figura 7. Resultados de simulación de la ALU en la herramienta Isim de Vivado. Los resultados mostrados son del caso 5 al 8.

El séptimo caso *CtrlALU*=1010, corresponde al desplazamiento a la izquierda donde el bit más significativo pasa a la posición del acarreo, pero como el bit más significativo es 0 entonces la bandera de acarreo no debe activarse. La única bandera activa es N debido a su naturaleza de activación. Por último, se realiza una

operación que modifica la salida y manda todo a 1 (incluyendo el acarreo), por lo que las banderas que se activan son N y C.

4. Conclusiones

El diseño de la ALU genérica de M bits se desarrolló satisfactoriamente. El diseño se podría implementar en un diseño de circuito integrado VLSI o en un procesador RISC reconfigurable. El enfoque modular del diseño nos permite modificar fácilmente la arquitectura de la ALU e inclusive analizar parámetros como área, consumo de potencia y velocidad en trabajos futuros.

Referencias

[1] J. Ball, J. Nurmi, *Designing Soft-Core Processors for FPGAs Processor Design" in Processor Design: System-on-Chip Computing for ASICs and FPGAs*, Netherlands: Springer, pp. 229-256, 2007.

[2] K. Compton, S. Hauck, "Reconfigurable computing: a survey of systems and software", *ACM Computing Surveys*, vol. 34, no. 2, pp. 171-210, 2002.

[3] Sivarama P. Dandamudi, "Guide To RISC Processor For Programmers And Engineers, Springer", Springer-Verlag New York, 2005.

[4] Y. Fei Guan, H. Lin, X. Guan, "Hierarchical Design of an Application Specific Instruction Set Processor for High-Throughput and Scalable FFT Processing", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, pp. 551-563, 2011.

[5] Tom Wada, "Small Risc Processor (SPR) design specification v1.0", 12th Design Contest in OKINAWA, pp. 1-17, 2009.

[6] *Behavioural languages - Part 1-1: VHDL Language Reference Manual. 2011*, IEC 61691-1-1:2011, ISBN 978-0-7381-6605-6.

[7] T. Esther Rani, M.A. Rani, R. Rao, "AREA optimized low power arithmetic and logic unit", *IEEE International Conference on Electronics Computer Technology*, pp. 224-228, April 2011.

[8] B. Stephen Brown, V. Zvonko, *Fundamentals of digital logic with VHDL Design*, 2nd Edition, Mc Graw Hill International Edition, 2005.

[9] Charles H.Roth, Jr., *Digital System Design using VHDL*, PWS Publishing Company, 2006.

[10] Douglas L. Perry, *VHDL*, third edition, McGraw-Hill, pp. 60-63, 238, July 1999.

[11] Mark Zwolinski, *Digital System Design with VHDL*, 2nd Edition, Prentice Hall, 2000.

[12] M. Morris Mano, *Arquitectura de Computadoras*, 3ra Edición, Pearson Education, 1994, Estado de México, pp. 85-86, ISBN: 968-880-361-8.