Congr. Int. en Ing. Electrónica. Mem. ELECTRO, Vol. 39, pp. 217-222, Oct 2017, Chihuahua, Chih. México
http://electro.itchihuahua.edu.mx/memorias_electro/MemoriaElectro2017.zip
ISSN 1405-2172

IMPLEMENTACIÓN VLSI DEL ALGORITMO DE LA RAÍZ CUADRADA PARA NÚMEROS BINARIOS EN PUNTO FIJO DE 32 BITS

Chávez-Bracamontes Ramón, Cortés-Solares Xochiquetzatl
Tecnológico Nacional de México/Instituto Tecnológico de Ciudad Guzmán
Departamento de Eléctrica y Electrónica
Av. Tecnológico #100 Ciudad Guzmán, Mpio. de Zapotlán el Grande, Jalisco, México
Tel. 01 (341) 575 20 50, Fax 01 (341) 575 20 74
e-mail rachavez@itcg.edu.mx, xochiquetzatlm15290013@itcg.edu.mx

RESUMEN.

En este trabajo se presenta una implementación VLSI de un procesador de la raíz cuadrada de números fraccionarios. Los resultados alcanzados por la arquitectura propuesta permiten obtener errores relativos por debajo de 1x10E-5. El diseño del circuito fue implementado sobre tecnología CMOS AMI C5 con un proceso de 0.5 micras utilizando un área de 2.25 mm^2. La fabricación fue desarrollada por la empresa MOSIS.

Palabras Clave: Diseño VLSI, Raíz Cuadrada, Alliance CAD System, Algoritmo en Hardware.

ABSTRACT.

In this paper, a VLSI implementation of a square root processor of fractional numbers is presented. The results achieved by the proposed architecture allow to obtain relative errors below 1x10E-5. The circuit design was implemented on CMOS AMI C5 technology with a 0.5 micron process using an area of 2.25 mm^2. The manufacture was developed by the MOSIS company.

Keywords: VLSI Design, Square Root, Alliance CAD System, Algorithm on hardware.

1. INTRODUCCIÓN

El cálculo de la raíz cuadrada de un número ha estado presente desde los antiguos egipcios, babilonios y chinos hasta nuestros días. La determinación de la raíz cuadrada es una importante operación matemática en sistemas de procesamiento de señales digitales. En la literatura se han reportado varios algoritmos para el cálculo rápido de la raíz cuadrada, la mayoría de estos algoritmos son del tipo iterativos de restauración y no restauración [1]. Muchos de los algoritmos que se han desarrollado han sido implementados por software, pero en muchas ocasiones no son adecuados para su implementación en hardware. Los algoritmos de raíz cuadrada son mucho más complejos en su estructura y su función que los de funciones básicas como la suma, multiplicación y división, no siendo fácil de implementar en hardware donde lo más importante es la precisión, el rendimiento y el costo requerido en área de silicio.
Muchas de las aplicaciones tecnológicas de hoy en día no hubieran sido posible sin los avances en distintas áreas como los sistemas electrónicos analógicos y digitales, las nuevas técnicas de procesamiento de señales, la implementación de algoritmos en hardware y muchas otras áreas del conocimiento. En la actualidad es de gran importancia tener una mejora constante hacia la optimización de recursos usados en dichos sistemas.
El propósito principal de este artículo es presentar la implementación de un algoritmo de raíz cuadrada mejorando la precisión de un algoritmo clásico a través de desplazamientos de bits, ejecutado en hardware y la optimización del área requerida en silicio. Para sintetizar la arquitectura diseñada se utilizó un conjunto de herramientas CAD denominado Alliance © LIP6, que trabajan bajo un entorno de desarrollo de software libre. Finalmente se muestran los resultados obtenidos del diseño de la arquitectura de un circuito integrado (C.I.) de alta escala de integración (VLSI) en la solución de la raíz cuadrada de números fraccionarios en formato binario de punto fijo de 32 bits.

2. DESARROLLO

2.1. Raíz cuadrada

La raíz cuadrada de un número x, es el número y que al ser multiplicado por sí mismo resulta en x nuevamente, por tanto, $x = y^2$ sería una ecuación equivalente. Cualquier número real no negativo x tiene una única raíz cuadrada no negativa, llamada raíz cuadrada principal y denotada como $\sqrt{x}$ donde x es el radicando y $\sqrt{\ }$ es el símbolo raíz cuadrada.

2.2. Algoritmo de aproximación iterativo de la raíz cuadrada

Para calcular $y = \sqrt{x}$ se puede emplear el algoritmo de aproximación iterativo el cual es modelado con el código siguiente:

```
if x = 0 then return x
y = 1.0
y1 = 0
while abs (y - y1) > error
        y1 = y
        y = (y + x / y) / 2
```

end while
return *y*

3. HERRAMIENTAS DE ALLIANCE CAD SYSTEM

Los circuitos integrados digitales han estado en constante mejora debido a los requerimientos de los diseñadores para realizar tareas cada vez más complejas, que como consecuencia permiten el desarrollo de esquemas optimizados que facilitan el proceso de diseño con una mínima cantidad de errores. Debido a que el diseño de circuitos integrados VLSI [2] es una tarea que está al margen del error, existen metodologías que utilizan herramientas CAD semi-automatizadas que disminuyen estos errores. Alliance CAD System [3] es una herramienta para el diseño digital VLSI, el cual proporciona un compilador para el lenguaje de descripción de hardware (VHDL) y las herramientas de síntesis que incluyen librerías de celdas estandarizadas facilitando el diseño digital. Además, contiene herramientas de Place y Route [4-7] para la generación de los archivos denominados layout, que contienen la descripción física del circuito integrado. La fabricación del circuito integrado y el costo se financia mediante un programa educativo a través de los servicios que ofrece MOSIS (Metal Oxide Semiconductor Implementation Service) [8] a universidades públicas bajo convenio.

La metodología empleada en el diseño del C.I. una vez que se definen las especificaciones del sistema comienza con la definición de la arquitectura. El siguiente paso es la descripción lógica a través de un lenguaje de descripción de hardware como VHDL[9] o Verilog, que son los lenguajes estándar para la descripción lógica de un C.I., en el diseño lógico se describe la topología de interconexión entre puertas y la transferencia de datos entre registros. El resultado del diseño lógico es un archivo de texto que contiene una descripción estructural de un bloque o circuito. Posteriormente se lleva a cabo el flujo de síntesis que transfiere el diseño desde un nivel lógico a un nivel físico, esto se realiza para cada uno de los bloques definidos en la arquitectura. Una etapa muy importante en esta metodología es la validación del diseño desde la etapa de descripción lógica, para lo cual se utilizan herramientas de simulación que verifican la funcionalidad del circuito integrado. Una vez que se valida la arquitectura propuesta se generan los layouts que describen la forma física del circuito integrado y se verifica que cumpla con las reglas de diseño del fabricante. Los archivos finales son generados en formatos estándar (cif) que se envían a MOSIS para la fabricación.

Las herramientas de síntesis de Alliance CAD System para esta metodología son VASY, BOOM, BOOG, LOON, OCP, NERO y S2R, mientras que las herramientas ASIMUT, COUGAR Y LVX [2-6] se utilizan para la verificación del circuito integrado, las cuales nos permiten la detección de errores y la optimización de recursos como área de silicio y retardos de propagación. Finalmente la herramienta GENLIB [4] permite la descripción estructural de la arquitectura completa.

4. ARQUITECTURA DEL C.I. SQRTFRAC

Se le denominó SQRTFRAC al circuito integrado que realiza la operación matemática de la raíz cuadrada. SQRTFRAC recibe el valor a extraerle la raíz cuadrada por un bus de entrada denominado DATAIN y el resultado lo entrega por el bus de salida DATAOUT. Para la operación también son necesarias una serie de señales de control como WRITE, HL, RESET, CLK, STAR y STATEOUT que se describen más adelante. El diseño de la arquitectura VLSI comienza con la definición del bloque principal al que se le se denominó SQRT32, ya que este bloque realiza la función de la raíz cuadrada para números enteros a través del método bit a bit no restaurador [10], el cual se implementó por el método de corrimientos para raíces de punto fijo. De forma general la arquitectura está conformada por 4 bloques como se muestra en la Fig.1. 1) SHA, 2) SQRT32, 3) SHB y 4) CONTROLLER. SHA es el bloque encargado de recibir dos datos de 16 bits en un registro de 32 bits que contendrá el argumento de la raíz. SQRT32 es el bloque principal que realiza la extracción de la raíz cuadrada (Fig. 2). El bloque SHB es un registro de 32 bits que almacena el resultado, además multiplexa el dato de salida de 32 bits para su lectura en un bus de 16 bits. Finalmente, CONTROLLER permite hacer un corrimiento a la derecha e izquierda con la finalidad de reducir los errores en la solución.

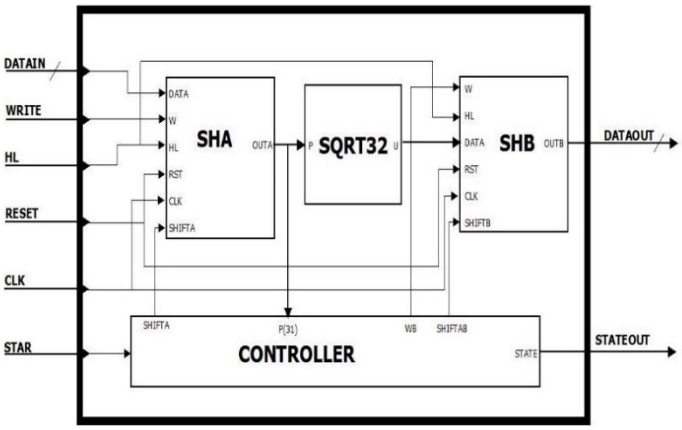

Fig. 1. Arquitectura del SQRTFRAC

4.1. Funcionamiento de SQRTFRAC

La primera acción que realizar es almacenar el valor del argumento al cual se le extraerá la raíz cuadrada. Para ellos se utiliza un formato de punto fijo de 32 bits, debido a los limitantes en cuanto al número de terminales de los circuitos

integrados fabricados a través de MOSIS en encapsulados de 40 terminales fue necesario limitar el bus de entrada DATAIN a 16 bits por lo cual es necesario en almacenar en dos etapas la parte alta y la parte baja. El bloque SHA como se menciona anteriormente contiene un registro de 32 bits, el cual recibe el dato de entrada mediante el bus DATAIN de 16 bits, el registro de 32 bits se divide en parte baja 0-15 bits y parte alta de 16-31 bits. Con la ayuda de la señal de control HL se selecciona respectivo registro, para elegir parte alta o baja de los registros se utiliza la señal HL, si HL es '1' se selecciona los bits 16-31 del registro, si HL es "0" selecciona los bits 0-15 del registro, la escritura se hace almacenando el dato de entrada en la parte alta y baja con la señal de control WRITE. Para seleccionar SHA o SHB se utiliza el bloque de CONTROLLER, con la señal SHIFTA se selecciona SHA y con SHIFTB se selecciona SHB.

El bloque SQRT32 recibe un dato de 32 bits, el cual procesa de forma combinacional y de acuerdo con el esquema de la Fig.2 para determinar la raíz cuadrada, calcula raíces de números enteros. Este esquema de la Fig.2 que corresponde a la extracción de una raíz de 8 -bits [9-10] y para un número de 32 bits se expandió dicho esquema, teniendo como entrada un dato de 32 bits y una salida (U) de 16 bits.

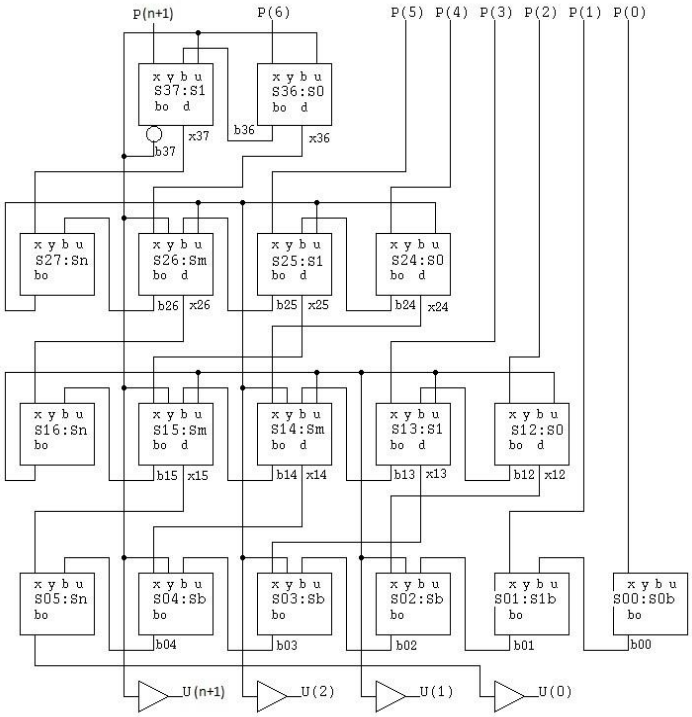

Fig. 2. Bloque SQRT32

Debido a que el bloque SQRT32 realiza la extracción de la raíz cuadrada dando resultados solo números enteros, esto genera errores muy significativos ya que trunca la parte fraccionaria del resultado, por tal motivo fue necesario implementar el algoritmo de la Fig.3 cuyo objetivo es recuperar la parte fraccionaria del resultado, disminuyendo así el error del SQRT32. El bloque CONTROLLER realiza escalamientos con corrimientos lógicos tanto a la derecha como a la izquierda con la finalidad de tener una aproximación más exacta, esta técnica se basa en escalar el dato de entrada eliminando los ceros de la izquierda llevando a cabo la extracción de un número escalado y su resultado volviéndolo a escalar. Esto reduce sustancialmente el error mejorando el resultado. Una vez que se introduce el valor por medio de DATAIN, se escribe el registro y se envía al bloque de CONTROLLER el cual lee el dato, detecta el bit más significativo si es '0' desplaza a la izquierda hasta que sea el bit más significativo '1' (si el bit más significativo es '1' envía el dato a SQRT32), a la par va acumulando los corrimientos realizados, si es par envía a SQRT32 que sacara la raíz del número ya desplazado, si no es par hará un corrimiento a la izquierda y enviara el dato a SQRT32. El dato obtenido de SQRT32 será enviado a SHB, es un dato de 16 bits, SHB recibe el dato obtenido y lo almacena en la parte alta de un registro de 32 bits, con la ayuda de CONTROLLER se realiza el desplazamiento a la derecha el cual será la mitad de corrimientos a la izquierda esto para no alterar el número ni el resultado.

El bloque CONTROLLER es una máquina de estados, a través SHIFTA selecciona el bloque SHA y con la señal STAR = '1' checar el bit más significativo P(31) , si P(31) es '0' desplaza a la izquierda el dato del registro SHA, a la par va acumulando cada desplazamiento en la variable Q, si Q es par manda el dato a la entrada de SQRT32 (P), si no hace un corrimiento a la derecha y manda el dato a SQRT32.

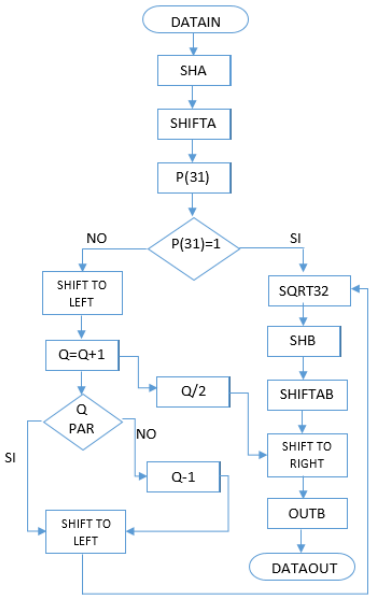

Fig. 3. Diagrama de flujo CONTROLLER

Para obtener el resultado correcto en el bus de salida DATAOUT es necesario desplazar el registro SHB a la derecha Q/2 posiciones, se hace el corrimiento a la derecha de los datos obtenidos de SQRT32 (U) dependiendo de Q, ya que se le aplicó un corrimiento, el corrimiento a la derecha depende que Q, ya realizado el corrimiento se envía a SHB en la cual el dato obtenido se guarda en un registro de 32 bits y el dato es multiplexado para la lectura en el bus de salida DATAOUT.

5. RESULTADOS

La implementación de algoritmos por software es muy flexible, ya sea utilizando lenguajes de alto nivel o bajo nivel. Con la finalidad de evaluar el tiempo de ejecución del algoritmo de la raíz cuadrada descrito en la sección 2.2, se realizó la implementación en el lenguaje "C" con las herramientas como MPLAB de Microchip, para posteriormente compararlo con los resultados de la implementación en VLSI del algoritmo de la raíz cuadrada con el chip SQRTFRAC.

5.1. Implementación por software de la raíz cuadrada

Para medir el tiempo de ejecución del algoritmo de la raíz cuadrada se empleó un compilador de 'C' embebido para microcontroladores de la compañía Microchip. El código para la implementación por software del algoritmo de la raíz cuadrada se muestra en la Fig. 4, el código se desarrolló sin el uso de librerías matemáticas y a través de las herramientas de depuración del software MPLAB se estimó que el tiempo de ejecución para calcular la raíz cuadrada de un número arbitrario (m=9) es de 373 µs, empleando una frecuencia de oscilación de 1MHz y el resultado proporcionado por este algoritmo es de 3.0999992, presentando un error considerable del 3%.

```
#include <stdio.h>
#include <stdlib.h>

/*
*
*/
int main (int argc, char** argv)

{
    float i=0;
    int j=0;
    int m=9;
    float x1,x2;
    while((i*i) <= m )
            i+=0.1;
    x1=i;
    for (j=0;j<10;j++)
      {
       x2=m;
       x2/=x1;
       x2+=x1;
       x2/=2;
       x1=x2;
                        }

     return (1);
}
```

Fig. 4. Algoritmo iterativo de la raíz cuadrada

Otra solución por software se realizó utilizando la librería math.h como se muestra en la Fig. 5. Para este caso se midió la velocidad de ejecución resultando que se necesitan 72µs a la misma frecuencia de 1Mhz. Como se puede observar al utilizar la función sqrt() de la librería math.h se consigue una aproximación más exacta y un menor tiempo de ejecución.

```
float main ()     {
    float z;
    int x=9;

z=sqrt (x);
    return    (1);
}
```

Fig. 5. Raíz cuadrada usando la librería math.h

5.2. Implementación en hardware del algoritmo de la raíz cuadrada utilizando SQRTFRAC

El diseño del SQRTFRAC se llevó acabo de manera jerárquica, de tal modo que se realizó la síntesis de forma independiente para cada uno de los bloques. En la Fig. 6 se observa una simulación de la arquitectura con ayuda de las herramientas CAD.

El dato de entrada que es colocado en DATAIN con formato hexadecimal en la cual se observa la parte alta es "0000" y la parte baja es "0002". Este es el argumento a calcular, también se muestras las entradas de control (WRITE, CLK, RST) y salida (DATAOUT) que se utilizaron para comprobar el correcto funcionamiento del extractor de raíz cuadrada de 32 bits.

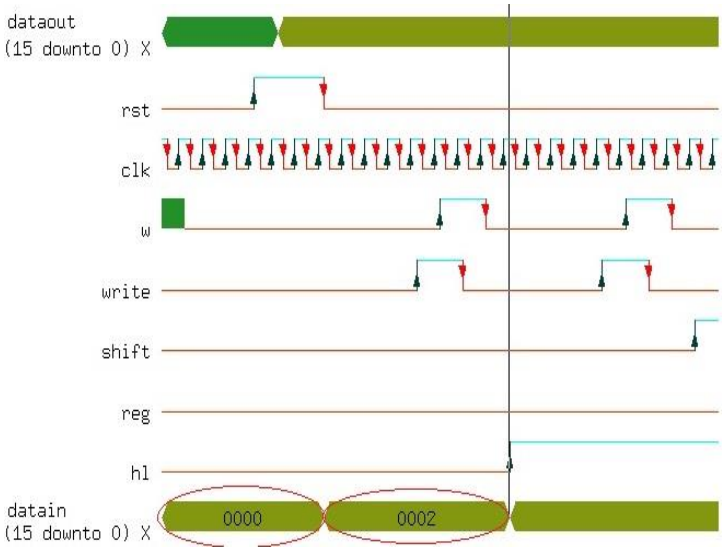

Fig. 6. Simulación del SQRTFRAC para almacenamiento del dato de entradas.

La simulación bloque SQRT32 por si solo obtiene la raíz cuadrada con un margen de error bastante alto como se muestra en la cuarta columna de la Tabla 1.

Tabla I
RESULTADOS DE SIMULACIÓN SQRT32 Y SQRTFRAC

RAÍZ CUADRADA					
NÚM	CALCULADORA	SQRT32	ERROR%	SQRTFRAC	ERROR %
2	1,414213562	1	29,2893219	1,41418457	0,00205005
10	3,16227766	3	5,13167019	3,16222314	0,00172394
120	10,95445115	10	8,71290708	10,9543457	0,0009626
350	18,70828693	18	3,78595291	18,7080078	0,00149197
1100	33,1662479	33	0,50125629	33,1660156	0,00070035
1800	42,42640687	42	1,00505063	42,4257813	0,0014746
2000	44,72135955	44	1,61300899	44,7207031	0,00146781
3500	59,16079783	59	0,27179794	59,1601563	0,00108447
4020	63,40346994	63	0,63635308	63,4033203	0,00023599
5500	74,16198487	74	0,21842036	74,1601563	0,00246567

El bloque de SQRT32 da resultados con números enteros, es por eso se vio en la necesidad de realizar el método de corrimientos de bits para escalar el número, tanto en el dato de entrada como en el dato de salida, el dato de entrada debe tener *n* corrimientos a la izquierda y el resultado debe ser *n*/2 corrimientos a la derecha, para que no se altere el resultado, esté método es implementado por SQRTFRAC cuyos resultados de simulación se muestran en la Fig.7, en esta imagen se muestra los corrimientos realizados hasta obtener nuestro resultado de la raíz del número 2. Se observa en la parte superior de la Fig.7 la señal DATAOUT en la cual se representa nuestra salida, la parte alta contiene los enteros y la parte baja los decimales. Este registro esta en formato hexadecimal. convirtiendo este valor a decimal es 1.41418457 dando un error mínimo de 0.00205005 comparando el valor con el resultado de una calculadora estándar (1.414213562).

Para la validación fue necesario un archivo de patrones con diferentes valores a calcular. Al realizar la simulación con la herramienta ASIMUT [4] se obtuvieron los datos mostrados en la Tabla I. La columna SQRT32 corresponde al resultado únicamente del bloque SQRT32 la columna SQRTFRAC los datos generados por la arquitectura del chip implementado. La columna calculadora, los resultados generados por una calculadora científica. Al comparar los resultados de la solución del SQRT32 con el error de la arquitectura propuesta, se observan errores por debajo de 1x10E-5, validando una muy buena respuesta de la arquitectura SQRTFRAC. También se puede observar en la Fig.8, la comparación de error tanto de SQRT32 como de SQRTFRAC, en la cual SQRT32 presenta un mayor error que el obtenido por SQRTFRAC logrando disminuir el error y manteniéndose en la tendencia a 0.

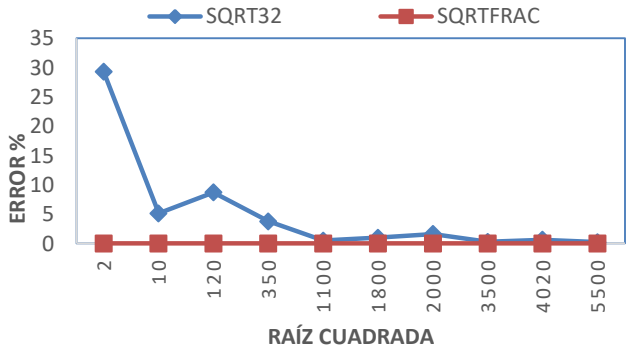

Fig. 8. Error de SQRT32 vs SQRTFRAC

Una vez validada la arquitectura propuesta se preparó el layout con dimensiones de 1.5mm x 1.5mm, el cual se muestra en la Fig.9 llevando a cabo la verificación de reglas de diseño y colocación de pads para su fabricación.

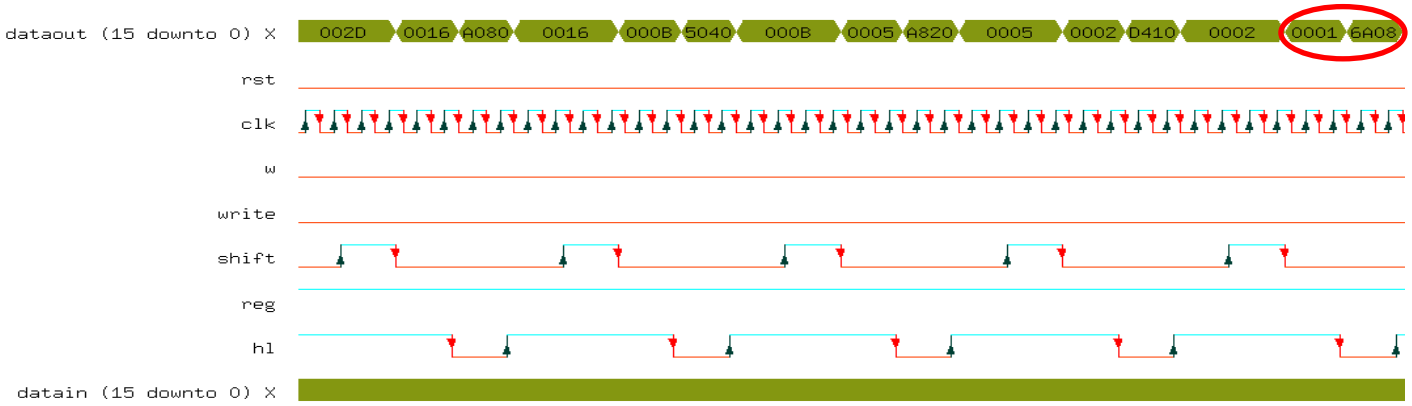

Fig. 7. Resultado de simulación del SQRTFRAC

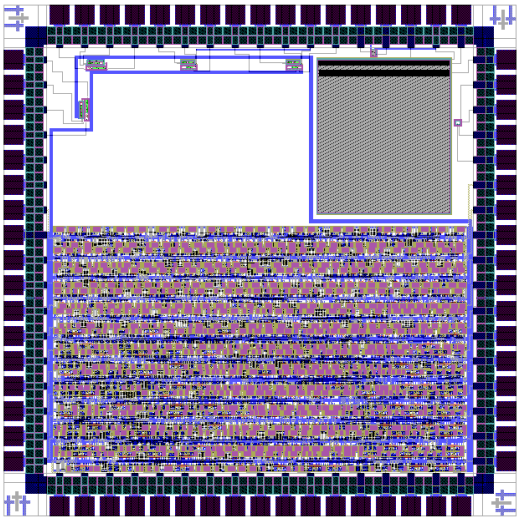

Fig. 9. Layout para fabricación del C.I.

En la Fig.10 se muestra el diagrama de bonding con las conexiones hacia las terminales exteriores para un encapsulado tipo LCC68 de montaje superficial con 68 terminales.

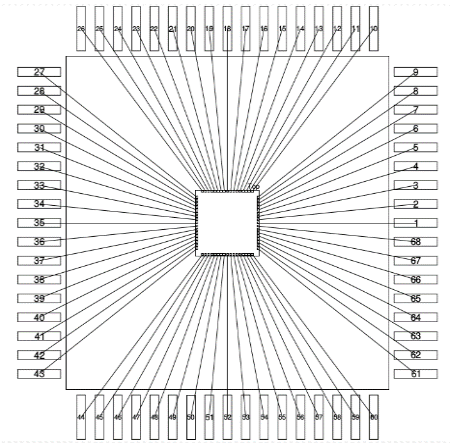

Fig. 10. Cableado con las terminales del encapsulado

6. CONCLUSIONES

Se implementó la arquitectura VLSI del algoritmo de la raíz cuadrada en 32 bits para números en punto fijo como un módulo de propiedad intelectual que nos facilitará la implementación de arquitecturas más complejas en el desarrollo de circuitos integrados con tecnología propia y con altas prestaciones para la implementación de algoritmos en hardware con aplicación al procesamiento avanzado de señales. Se comprobó que la arquitectura propuesta SQRTFRAC cumple su cometido con un margen de error que oscila en el rango de 1x10E-6 a 1x10E-5. Los resultados obtenidos mostraron la factibilidad en la implementación de la arquitectura optimizadas de operaciones aritméticas en algoritmos de control avanzado.

7. REFERENCIAS

[1] Omondi A.,Computer-Hardware evaluation of mathematical functions, Hackensack, NJ : Imperial College Press, 2015, 411

[2] Weste N-H. E., Harris D-M., CMOS VLSI Design a Circuit and Systems Perspective, 4th ed., U.S.A., Addison Wesley, 2011, pp. 615-657.

[3] LIP6, Alliance VLSI Cad System, [en línea], Pierre & Marie Curie University, Paris, France, 2014, Disponible en: https://soc-extras.lip6.fr/en/alliance-abstract-en/

[4] Chaput J-P., Pétrot F. GenLib User's Manual [en línea], Pierre & Marie Curie University, LIP6 ASIM Department, 2002. Disponible en://ftp.lip6.fr/lip6/softs/alliance/latest-checkout/alliance/src/genlib/doc/genlib.pdf

[5] Lam K-S. Ak F. Alliance Tutorial, Part 1 VHDL Modeling and Simulation [en línea], Pierre & Marie Curie University, France, 2004. Disponible en: http://www-asim.lip6.fr/pub/alliance/distribution/latest/

[6] Lam K-S. Ak F. Alliance Tutorial, Part 2 Logic Synthesis [en línea], Pierre & Marie Curie University, France, 2004. Disponible en : http://www-asim.lip6.fr/pub/alliance/distribution/latest/

[7] Lam K-S. Ak F. Alliance Tutorial, Part 3 Place and Route [en línea], Pierre & Marie Curie University, France, 2004. Disponible en: http://www-asim.lip6.fr/pub/alliance/distribution/latest/

[8] The MOSIS Service, USC Information Sciences Institute, 2017, disponible en: http://www.mosis.com/pages/products/mep/mep-about.

[9] Department of Computer Science and Electrical Engineering, VHDL reference material [en linea], 2017, Disponible en: https://www.csee.umbc.edu/portal/help/VHDL/samples/samples.shtml

[10] Tole Sutikno, "An Efficient Implementation of the Non Restoring Square Root Algorithm in Gate Level" Member, IACSIT, IEEE, International Journal of Computer Theory and Engineering, Vol.3, No.1, February, 2011.