Implementación y optimización del uso de DPS en FPGA en diseño de circuitos a medida para calcular determinantes de orden 4

Francisco J. Plascencia Jauregui

Centro Universitario de Ciencias Exactas e Ingenierías, Universidad de Guadalajara,
Blvd. Marcelino García Barragán #1421, esq. Calzada Olímpica, C.P. 44430, Guadalajara, Jal., México
francisco.plascencia@alumno.udg.mx

Juan José Raygoza P.

Centro Universitario de Ciencias Exactas e Ingenierías, Universidad de Guadalajara,
Blvd. Marcelino García Barragán #1421, esq. Calzada Olímpica, C.P. 44430, Guadalajara, Jal., México
juan.raygoza@cucei.udg.mx

Edwin C. Becerra A.

Centro Universitario de Ciencias Exactas e Ingenierías, Universidad de Guadalajara,
Blvd. Marcelino García Barragán #1421, esq. Calzada Olímpica, C.P. 44430, Guadalajara, Jal., México
edwin.becerra@cucei.udg.mx

Susana Ortega Cisneros

Centro de Investigación y de Estudios Avanzados del I.P.N, CINVESTAV, Unidad Guadalajara
Av. del Bosque 1145, colonia el Bajío, Zapopan, 45019, Jalisco, México
susana.ortega@gdl.cinvestav.mx

Resumen

En este artículo se presenta el diseño e implementación de dos circuitos digitales a medida para el cálculo de determinantes de matrices de orden 4, mediante el algoritmo del Teorema de Laplace, utilizando números enteros de 8 bits. Se analizan los resultados de la implementación de los circuitos enfocados desde dos perspectivas, la

primera instanciando un módulo que calcula determinantes de orden 3, mientras que en la segunda, las multiplicaciones se realizan de manera directa en el mismo bloque, reduciendo así la cantidad de unidades DSP necesarios para obtener el resultado final. En ambos casos se comparan tanto la ocupación y los tiempos de respuesta. Por otro lado, la descripción del circuito se realizó en Lenguaje de Descripción de Hardware (HDL) en el software ISE de Xilinx.

Palabra(s) Clave(s): determinante, DSP, FPGA, teorema de Laplace.

1. Introducción

El trabajo con matrices ha resultado desde sus inicios de gran ayuda para el quehacer científico sobre todo por su gran adaptabilidad a diferentes ramas del saber; operaciones como el cálculo de determinantes, matrices inversas, transpuestas, entre otras son esenciales en áreas como la navegación espacial, visión artificial y procesamiento digital de imágenes, por mencionar algunas.

No obstante, con el avance y especialización de las ciencias, la cantidad de información a procesar crece cada vez más, y con ello la complejidad del algebra matricial [3], es por ello que en el ámbito académico se trabaja en el desarrollo de algoritmos cada vez más eficaces y eficientes [1, 2, 6, 7, 9].

Por otro lado, existen en la actualidad herramientas que facilitan el procesamiento de datos basadas en hardware y proporcionan al diseñador la flexibilidad de diseñar la arquitectura y gran capacidad de procesamiento, estos son los dispositivos reconfigurables FPGAs.

Utilizando los FPGAs como plataforma se abre la oportunidad de aplicar algoritmos enfocados al álgebra matricial. Además, es importante señalar que es posible reducir el consumo de energía, minimizar el uso de recursos, aumentar las velocidades de respuesta o la cantidad de información procesada [4, 5, 7, 8, 10, 11].

A pesar de esto, los dispositivos reconfigurables tienen ciertas limitantes, por ejemplo, la representación de los números decimales, que implica un gran consumo de recursos del FPGA.

2. Matrices y determinantes

Las matrices se definen como un arreglo bidimensional de datos y se les nombra con una letra mayúscula, mientras que sus elementos se enumeran con letras minúsculas. (ver Fig. 1).

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

Fig. 1. Matriz de 2x2.

En la Fig. 1 los subíndices señalan la posición del elemento respecto a las filas y las columnas, así pues, el elemento a_{21} se encuentra en la fila 2 y columna 1.

Por otro lado, una matriz que tiene la misma cantidad de filas y columnas se le denomina matriz cuadrada y se dice que es de orden n, en este sentido, la Fig. 1 muestra una matriz de orden 2.

Ahora bien, el determinante hace referencia a la expresión matemática que está intrínsecamente relacionado con la matriz cuadrada que le da origen, y posee varias propiedades, como la de establecer la singularidad de dicha matriz, es decir, indica que si a partir de esa matriz es posible obtener un determinante mediante ciertas operaciones, es decir, dicha matriz es no singular.

Sin embargo, dependiendo del tamaño de la matriz y sus elementos, se han desarrollado diferentes algoritmos para el cálculo de determinantes a través de los años, así pues, son varias las formas de señalar el determinante de una matriz. (ver Fig. 2).

$$det\ A = det\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

Fig. 2. Determinante de una matriz de orden 3.

3. Algoritmo para calcular determinantes

Para el cálculo de un determinante se emplea el algoritmo del Teorema de Laplace, mediante el cual es posible calcular este valor a partir del uso de menores y cofactores [3].

$$|A| = \sum_{i=1}^{n} \sum_{j=1}^{n} (-1)^{i+j} a_{ij}\, c_{ij} \quad (1)$$

Considerando la fórmula general planteada en (1), se selecciona una fila o columna, para en seguida obtener los menores de cada uno de los elementos de esa fila o columna. Además, es importente señalar que los menores son las matrices reducidas que se obtienen eliminando los elementos de la fila y columna del elemento elegido (ver Fig. 3).

$$\begin{vmatrix} a_{11} & \boldsymbol{a_{12}} & a_{13} \\ \boldsymbol{a_{21}} & a_{22} & \boldsymbol{a_{23}} \\ \boldsymbol{a_{31}} & a_{32} & \boldsymbol{a_{33}} \end{vmatrix} \rightarrow a_{12} = \begin{pmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{pmatrix}$$

Fig. 3. Menor del elemento a12.

Una vez que se tiene el menor se calcula su determinante, que se conoce como cofactor. Ahora, este cofactor se multiplica con el elemento seleccionado, y su signo depende de la posición del elemento, es decir, si la suma de los subíndices es par, el signo es positivo, mientras que si la suma es impar es negativo.

Por lo cual la aplicación del teorema a una matriz de orden 3 en la primera columna se observa en la Fig 4.

$$\begin{vmatrix} \boldsymbol{a_{11}} & a_{12} & a_{13} \\ \boldsymbol{a_{21}} & a_{22} & a_{23} \\ \boldsymbol{a_{31}} & a_{32} & a_{33} \end{vmatrix} = +a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{21}\begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} + a_{31}\begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix}$$

Fig. 4. Aplicación de Teorema de Laplace a una matriz de orden 3.

A continuación se realiza el desarrollo a nivel de operaciones aritméticas (ver Fig. 5).

$$+a_{11}\big((a_{22} * a_{33}) - (a_{32} * a_{23})\big) - a_{21}\big((a_{12} * a_{33}) - (a_{32} * a_{13})\big) + a_{31}\big((a_{12} * a_{23}) - (a_{22} * a_{13})\big)$$

Fig. 5. Desarrollo del determinante de orden 3.

Con el desarrollo de la Fig. 5 se requieren hasta ahora nueve multiplicaciones, cuatro restas y una suma.

Ahora bien, como el objetivo es llegar a matrices de orden cuatro, se hace uso de una de las bondades de este método, que es la recursividad.

Para ello, se plantea en primer lugar una matriz de orden 4 (ver Fig. 6).

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{vmatrix} = \begin{vmatrix} a_{15} & a_{14} & a_{13} & a_{12} \\ a_{11} & a_{10} & a_{9} & a_{8} \\ a_{7} & a_{6} & a_{5} & a_{4} \\ a_{3} & a_{2} & a_{1} & a_{0} \end{vmatrix}$$

Fig. 6. Matriz de orden 4.

Se aplica el algoritmo del Teorema de Laplace (ver Fig. 7).

$$+a_{15}\begin{vmatrix} a_{10} & a_{9} & a_{8} \\ a_{6} & a_{5} & a_{4} \\ a_{2} & a_{1} & a_{0} \end{vmatrix} - a_{11}\begin{vmatrix} a_{14} & a_{13} & a_{12} \\ a_{6} & a_{5} & a_{4} \\ a_{2} & a_{1} & a_{0} \end{vmatrix} + a_{7}\begin{vmatrix} a_{14} & a_{13} & a_{12} \\ a_{10} & a_{9} & a_{8} \\ a_{2} & a_{1} & a_{0} \end{vmatrix} - a_{3}\begin{vmatrix} a_{14} & a_{13} & a_{12} \\ a_{10} & a_{9} & a_{8} \\ a_{6} & a_{5} & a_{4} \end{vmatrix}$$

Fig. 7. Aplicación de Teorema de Laplace a un determinante de orden 4.

Por otro lado, para cuantificar el consumo total de operaciones se desarrolla las operaciones aritméticas (ver Fig. 8).

$$a_{15}\Big(a_{10}\big((a_5 * a_0) - (a_1 * a_4)\big) - a_6\big((a_9 * a_0) - (a_1 * a_8)\big) + a_2\big((a_9 * a_4) - (a_5 * a_8)\big)\Big)$$

$$-a_{11}\left(a_{14}\left((a_5 * a_0) - (a_1 * a_4)\right) - a_6\left((a_{13} * a_0) - (a_1 * a_{12})\right) + a_2\left((a_{13} * a_4) - (a_5 * a_{12})\right)\right)$$

$$+a_7\left(a_{14}\left((a_9 * a_0) - (a_1 * a_8)\right) - a_{10}\left((a_{13} * a_0) - (a_1 * a_{12})\right) + a_2\left((a_{13} * a_8) - (a_9 * a_{12})\right)\right)$$

$$-a_3\left(a_{14}\left((a_9 * a_4) - (5 * a_8)\right) - a_{10}\left((a_{13} * a_4) - (5 * a_{12})\right) + a_6\left((a_{13} * a_8) - (a_9 * a_{12})\right)\right)$$

Fig. 8. Desarrollo para obtener el determinante de orden 4.

De esta forma un determinante de orden cuatro se reduce a cuatro determinantes de orden 3; sin embargo, su consumo se incrementa a 40 multiplicaciones, 18 restas y 6 sumas.

No obstante lo anterior, se observa que hay algunas multiplicaciones que se repiten, por lo que en lugar de instanciar las operaciones mediante un bloque que calcule un determinante de orden tres, se propone un segundo diseño, realizando las operaciones individualmente para reducir las multiplicaciones necesarias.

Tomando como base las Figs. 5 y 8 se desarrollan los diagramas de flujo para cada uno de los circuitos que se implementarán (ver Fig. 9 y 10).

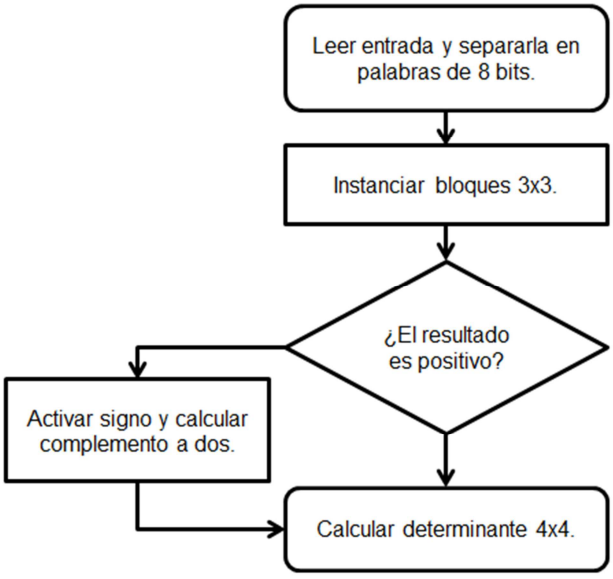


Fig. 9. Diagrama de flujo del circuito 1.

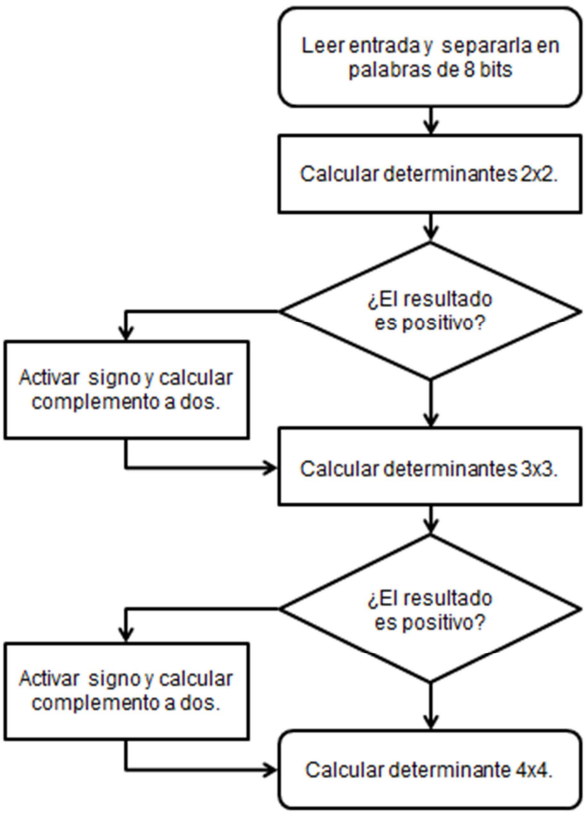


Fig. 10. Diagrama de flujo para el circuito 2.

4. Diseño e implementación de la unidad aritmética para calcular el determinante de una matriz de orden 4, con instanciación de bloques

Los valores de entrada para cada elemento de la matriz se ingresan mediante un vector de datos binarios, que contiene la representación de cada número con valores entre 0 y 255, este conjunto de unos y ceros se separan para representar los valores individuales de la matriz, y así poder realizar las operaciones necesarias y obtener el determinante de dicha matriz.

En Lenguaje de Descripción de Hardware se declaran las señales que reciben estos valores (ver Fig. 11).

```
signal a15, a14, a13, a12,
       a11, a10,  a9,  a8,
        a7,  a6,  a5,  a4,
        a3,  a2,  a1,  a0: std_logic_vector(x-1 downto 0);
```

Fig. 11. Señales declaradas para contener los valores del vector de entrada.

Posteriormente, a cada señal se le asignan 8 bits de la cadena inicial de datos.

4.1. Instanciación del bloque para calcular determinantes de orden 3

A continuación, se llama el bloque al que se le envía como parámetros las señales con las que se forman los cuatro menores y del cual se obtiene como salida el valor del cofactor y la señal de signo que es negativa cuando su valor es 1 y 0 en caso contrario (ver Fig. 12).

```
det_3x3_1: entity work.Laplace_3x3_DSP_8bits port map
(A=> (a10&a9&a8&a6&a5&a4&a2&a1&a0), signo=>signo1, det=>det1);
```

Fig. 12. Instanciación del bloque calculador de determinantes de orden 3.

4.1.1. Señal de signo

Esta señal recibe su origen al momento de operar un determinante de orden 3, como se observa en la Fig. 4, el primer término se origina a partir del producto de uno de los elementos por la diferencia de otros dos productos $(a_{11}*(a_{22}*a_{33})-(a_{32}*a_{23}))$, es justamente aquí dónde se pueden generar números negativos, pero al ser el resultado de una diferencia de productos, este número tiene el doble de longitud de cada elemento, es decir, 16 bits. Por otro lado, al multiplicar esta cadena por un elemento de 8 es posible recibir resultados indeseados como se muestra en la Fig. 13 donde el resultado correcto es la segunda opción.

Debido a esta situación, se considera de gran importancia el poder detectar si el resultado de una operación es negativo o positivo para continuar con el resto de las operaciones.

$$255 * (-254) = 11111111 * 1111111100000010 = 11111110000000101111110$$

$$-(255 * 254) = 11111111 * 0000000011111110 = 11111110000000101111110$$

Fig. 13. Origen de la necesidad de implementar la señal de signo.

Ademas, la corrección del problema se logra al comparar los resultados de las primeras multiplicaciones, si el primer producto es menor al segundo se genera un número negativo, por lo que es necesario operarlo de forma positiva mediante su complemento a dos, una vez obtenido el resultado se devuelve a su forma negativa mediante el complemento antes mencionado (ver Fig. 14).

Estas mismas operaciones se realizan con los otros dos términos de la Fig. 4, con lo que se generan los tres valores necesarios para calcular el determinante de orden 3. Durante la ejecución de estos procedimientos se generan además las señales que permiten establecer el signo del mencionado determinante.

```
mult1<=a4*a0;
mult2<=a1*a3;
process (mult1, mult2, datos1, comp2_1, comp2_2, comp2_3, a8)
begin
if (mult1<mult2) then
  comp2_1<=mult1-mult2;
  comp2_2<= (not (comp2_1))+1;
  comp2_3<= a8*comp2_2;
  datos1<= (not (comp2_3))+1;
  sign1<='1';
else datos1<= a8*(mult1-mult2); sign1<='0';
end if;
end process;
```

Fig. 14. Sección de código para detectar números negativos.

Una vez que se logra obtener los valores y signos de los tres términos se realiza el análisis de ellos para determinar el signo final. De acuerdo con la Fig. 4, al primer dato se resta el segundo, posteriormente se suma el tercero.

De esta forma, se estudian las ocho posibles combinaciones pertenecientes a los signos de los tres datos, dentro de estos casos se profundiza en los absolutos (complemento a dos, cuando son negativos) de cada dato.

Por ejemplo, al tratarse del caso en que los tres son positivos, el signo del determinante final será positivo cuando el valor del primer dato sea mayor que el segundo. En caso contrario, el tercer valor debe ser superior al valor absoluto de la resta (señal pre_resta y su complemento a dos) de los primeros dos (ver Fig. 15).

```
pre_resta<=datos1-datos2;
comp2_pre<= (not (pre_resta))+1;

process (sign1, sign2, sign3, datos1, datos2, datos3,
         pre_resta, comp2_pre, comp2_1, comp2_2, comp2_3,
         comp2_4, comp2_5, comp2_6, comp2_7, comp2_8, comp2_9)
begin
   if (sign1='0' and sign2='0' and sign3='0') then
       if (datos1>datos2) then signo<='0';
            elsif (datos1<datos2) then
                        if (comp2_pre<datos3) then signo<='0';
                        else signo<='1';
                        end if;
                 end if;
```

Fig. 15. Señal de pre_resta y estudio del primer caso de signos.

Al concluir estos ocho casos se tienen como resultados un determinante de una matriz de orden 3 y una señal de signo que comunica si este número es positivo o negativo, esto facilitará su instanciación y aplicación en el cálculo de un determinante de orden 4.

5. Diseño e implementación de la unidad aritmética para calcular el determinante de una matriz de orden 4, con multiplicaciones internas

En la Fig. 8 se observan las multiplicaciones que se repiten, una vez eliminadas sólo quedan doce operaciones y con esto se minimiza el uso de DSPs (ver Tabla 1).

Nombre	Producto	Nombre	Producto
mult1	a5*a0;	mult7	a13*a0;
mult2	a1*a4;	mult8	a1*a12;
mult3	a9*a0;	mult9	a13*a4;
mult4	a1*a8;	mult10	a5*a12;
mult5	a9*a4;	mult11	a13*a8;
mult6	a5*a8;	mult12	a9*a12;

Tabla 1. Nombre de las señales y valores a multiplicar.

Se analiza cada par de multiplicaciones para obtener los valores para el cofactor, considerando a la vez, los casos que generan números negativos (ver Fig. 16).

```
mult1<=a5*a0;
mult2<=a1*a4;
process (mult1, mult2, datos1, comp2_1, comp2_2, comp2_3, a10)
begin
if (mult1<mult2) then
  comp2_1<=mult1-mult2;
  comp2_2<= (not (comp2_1))+1;
  comp2_3<= a10*comp2_2;
  datos1<= (not (comp2_3))+1;
  signo1<='1';
else datos1<= a10*(mult1-mult2); signo1<='0';
end if;
end process;
```

Fig. 16. Análisis en cada par de multiplicaciones.

Posterior al análisis que lleva a obtener cada dato que ha de sumarse o restarse, para generar cada uno de los cofactores, se realiza también el correspondiente estudio de los signos para determinar si se trata de un número positivo o negativo (ver Fig. 14).

Una vez calculados los cuatro cofactores se estudian las multiplicaciones de dichos valores con el elemento de la matriz original, todo esto considerando siempre los casos en que el cofactor sea positivo o negativo (ver Fig. 17).

```
if (signo13='1') then --Signo del primer cofactor
  comp2_det11<=pre_det1;
  comp2_det12<= (not (comp2_det11))+1;
  comp2_det13<= a15*comp2_det12;
  det1<= (not (comp2_det13))+1;
  signo_det1<='1';
else det1<= a15*pre_det1; signo_det1<='0';
end if;
```

Fig. 17. Multiplicación del cofactor por el elemento de la matriz original.

Una vez que se tienen los cuatro determinantes se realizan las sumas y restas necesarias para obtener el resultado final, sin embargo, aún falta establecer los criterios para definir el valor de la señal de signo. Para este caso, al tratarse de cuatro valores se separan en pares, las restas de los determinantes uno y dos, y los determinantes tres y cuatro, estas comparaciones se envían a un par de señales parciales que se suman para dar una señal final (ver Figs. 18 y 19).

Una vez concluido el trabajo de la describir los algoritmos de instanciación y de multiplicaciones internas, se usó el software ISE de Xilinx para sintetizar los bloques (ver Fig. 20).

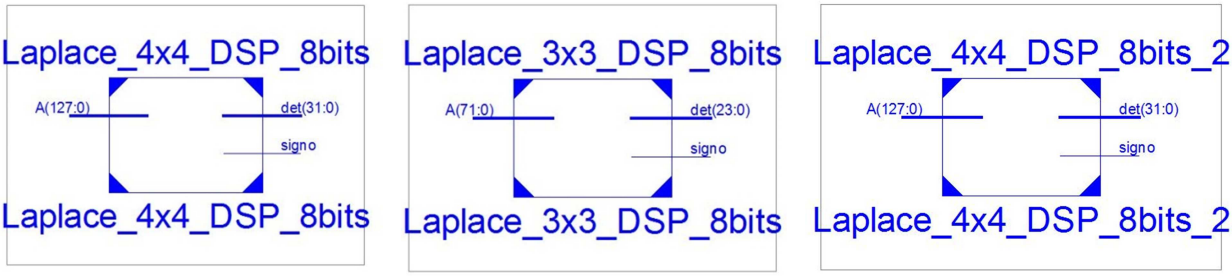
Con el fin de medir tiempos de respuesta y ocupaciones, se selecciona a la FPGA XC6VLX240T de la tarjeta de pruebas ML605, dicho dispositivo contiene un total de 768 unidades DSP [11, 12, 13].

```
if (signo_det1='0' and signo_det2='0') then
    if (det1>det2) then SS1<='0'; --Señal parcial SS1
            else SS1<='1';
         end if;

  elsif (signo_det1='0' and signo_det2='1') then SS1<='0';

  elsif (signo_det1='1' and signo_det2='0') then SS1<='1';

  elsif (signo_det1='1' and signo_det2='1') then
        if (comp2_det13>comp2_det23)  then SS1<='0';
               else SS1<='1';
            end if;
    else SS1<='1';
end if;
```

Fig. 18. Análisis de los signos de los dos primeros determinantes.

```
pre_resta5<=det1-det2;
comp2_pre5<= (not (pre_resta5))+1;
pre_resta6<=det3-det4;
comp2_pre6<= (not (pre_resta6))+1;

   if (SS1='0' and SS2='0') then signo<='0';

     elsif (SS1='0' and SS2='1') then
           if (pre_resta5>comp2_pre6)  then signo<='0';
                  else signo<='1';
               end if;

     elsif (SS1='1' and SS2='0') then
           if (comp2_pre6<pre_resta5)  then signo<='0';
                  else signo<='1';
               end if;

     elsif (SS1='1' and SS2='1') then signo<='1';

        else SS2<='1';
   end if;
```

Fig. 19. Generación de la señal final de signo.



a) **b)** **c)**

Fig. 20. a) Determinante 4x4 con instanciación. b) Determinante 3x3. C) Determinante 4x4 con multiplicaciones internas.

6. Resultados de calcular determinantes de orden 4 con instanciación

Se declaran los vectores de entrada que representan a las matrices utilizadas para poner a prueba el desempeño del diseño en el dispositivo reconfigurable seleccionado, cabe mencionar que el determinante de dichas matrices son 12, 1278, -6312 y -3784, respectivamente (ver Fig. 21).

$$\begin{vmatrix} 250 & 251 & 252 & 253 \\ 254 & 255 & 254 & 252 \\ 252 & 252 & 252 & 255 \\ 252 & 253 & 254 & 255 \end{vmatrix} \begin{vmatrix} 250 & 251 & 251 & 253 \\ 254 & 255 & 254 & 252 \\ 252 & 252 & 252 & 255 \\ 252 & 253 & 254 & 255 \end{vmatrix} \begin{vmatrix} 250 & 251 & 251 & 253 \\ 254 & 255 & 254 & 252 \\ 252 & 252 & 255 & 255 \\ 252 & 250 & 254 & 255 \end{vmatrix} \begin{vmatrix} 250 & 251 & 255 & 253 \\ 254 & 255 & 254 & 252 \\ 252 & 252 & 252 & 254 \\ 252 & 253 & 254 & 255 \end{vmatrix}$$

Fig. 21. Matrices utilizadas para poner a prueba los diseños en la FPGA.

Además, cabe mencionar que debido a la longitud de la cadena de bits de entrada, ésta no puede ser representada en la ventana de simulación. A partir de los resultados que se obtienen al calcular el determinante de cada matriz planteada anteriormente, se infiere que hay un tiempo promedio de respuesta de 42.19575 ns (ver Fig. 22).

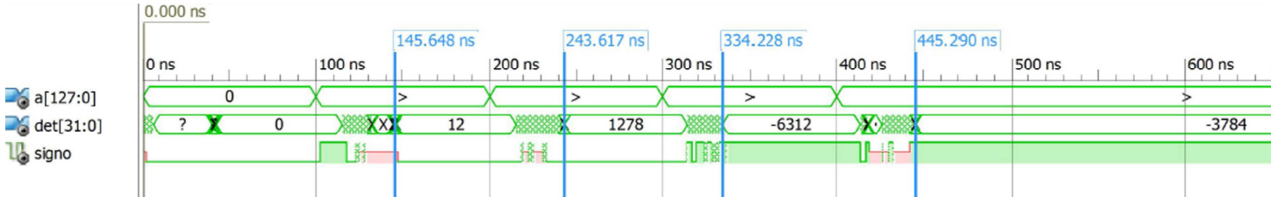


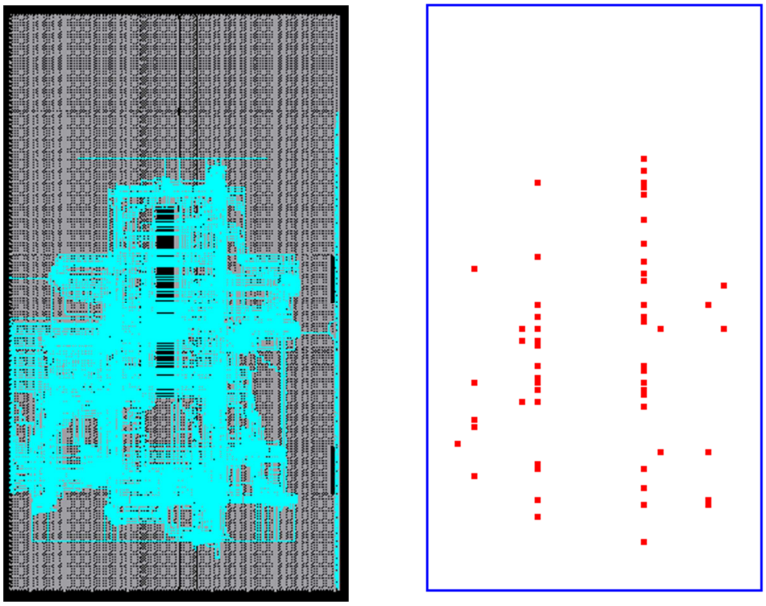
Fig. 22. Tiempo de respuesta para cada una de las matrices ingresadas.

En cuanto a la ocupación, este circuito consume un total de 56 DSP's (ver Tabla 2).

Resumen de utilización del dispositivo			
Utilización de Slices Lógicos	**Usados**	**Disponibles**	**Porcentaje**
Número de Slices Registros	1,025	301,440	1%
Número de Slices LUTs	2,130	150,720	1%
Número de Slices ocupados	773	37,680	2%
Número de IOBs	263	600	43%
Número de DSP48E1s	56	768	7%

Tabla 2. Recursos utilizados por el primer circuito.

Por otro lado, se obtiene el ruteo y localización de los DSP's en la FPGA para apreciar la utilización de la misma (ver Fig. 23).



a) **b)**

Fig. 23. a) Ruteo del circuito 1. b) Localización de los 56 DSP en la PGFA.

7. Resultados de calcular determinantes de orden 4 con multiplicaciones internas

El tiempo promedio de respuesta del segundo circuito es de 38.06975 ns (ver Fig. 24).

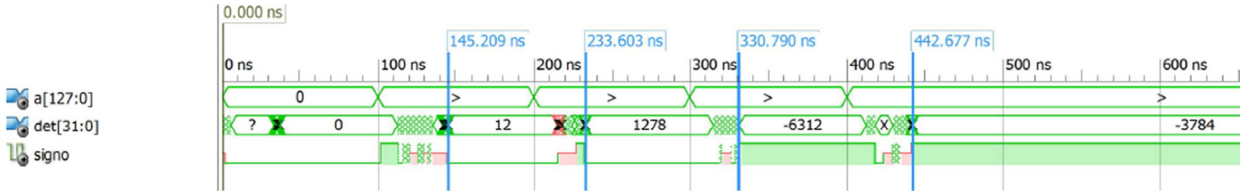


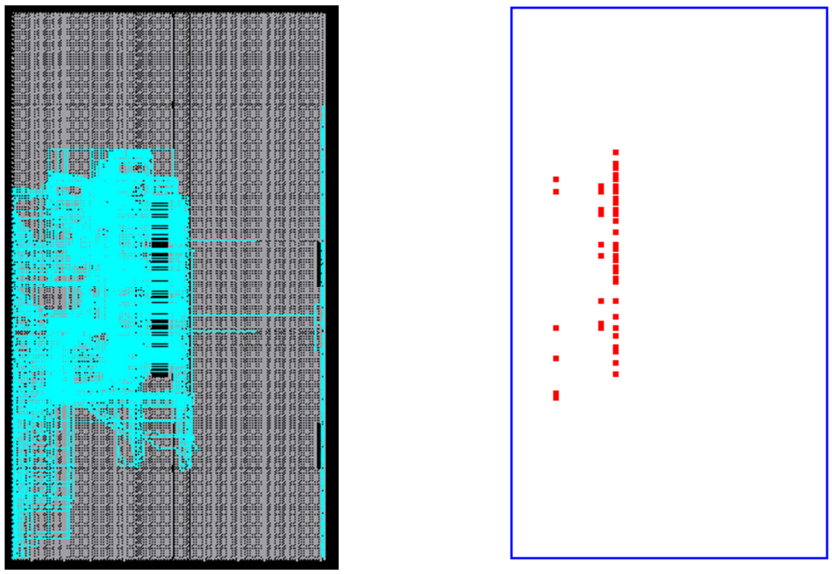
Fig. 24. Tiempos de respuesta del segundo circuito para cada matriz.

La tabla de utilización de recursos muestra una disminución de DSP empleados para proporcionar una respuesta, sin embargo hay un incremento en el número de IOB's puestos en marcha (ver Tabla 3).

Resumen de utilización del dispositivo			
Utilización de Slices Lógicos	**Usados**	**Disponibles**	**Porcentaje**
Número de Slices Registros	929	301,440	1%
Número de Slices LUTs	1,938	150,720	1%
Número de Slices ocupados	716	37,680	1%
Número de IOBs	395	600	65%
Número de DSP48E1s	44	768	5%

Tabla 3. Recursos utilizados por el segundo circuito.

De igual forma se genera el ruteo y localización de los DSP's para observar la ocupación de la FPGA (ver Fig. 25).



a) **b)**

Fig. 25. a) Ruteo del circuito 2. b) Localización de los 44 DSP en la FPGA.

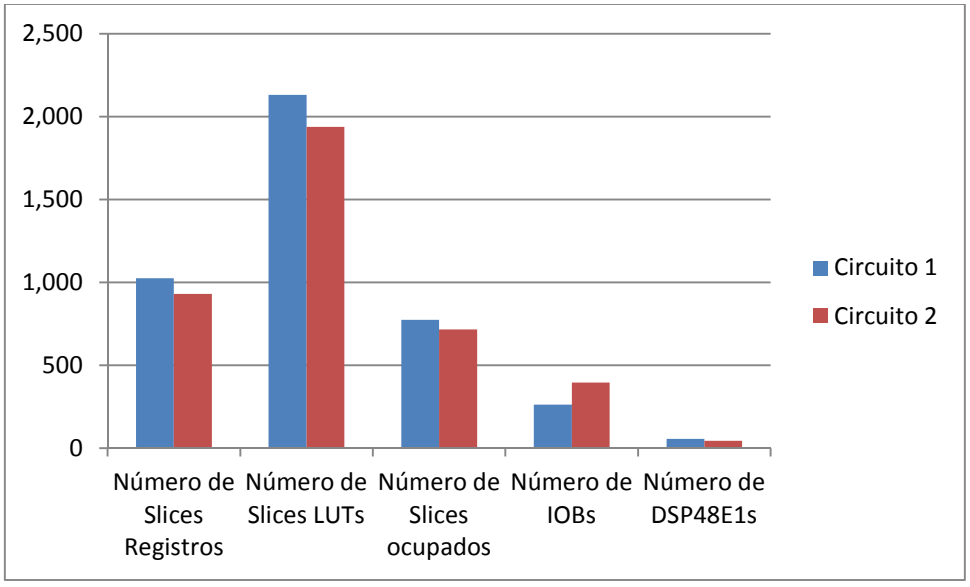
8. Comparación de los resultados de las implementaciones de ambos circuitos

Al tratarse de dos circuitos diferentes es lógico pensar que debe haber diferencias en cuanto a los recursos utilizados de la FPGA (ver Tabla 3).

Utilización del dispositivo			
Utilización de Slices Lógicos	**Circuito 1**	**Circuito 2**	**Disponibles**
Número de Slices Registros	1,025	929	301,440
Número de Slices LUTs	2,130	1,938	150,720
Número de Slices ocupados	773	716	37,680
Número de IOBs	263	395	600
Número de DSP48E1s	56	44	768

Tabla 3. Comparación de utilización de recursos en ambos circuitos.

Esta información se observa más claramente en la gráfica 1, donde se observa que el segundo circuito consume menos recursos.



Gráfica 1. Comparación de los recursos usados en la FPGA.

Hablando del número de Slices hay una disminución de 96 unidades, lo que representa un 9.3%; tratándose de los Slices LUT la reducción es de 192 bloques, que es el 9%. Mientras que los Slices ocupados que se ahorraron son 57 que equivale a 7.3%. En cuanto a los DSP, que es nuestro objetivo, hay un ahorro de 12 bloques, lo que corresponde a un 21.4%.

Por otro lado, el único caso en que hay un incremento es en los IOB, estimados en 132 unidades, lo que implica un 50%.

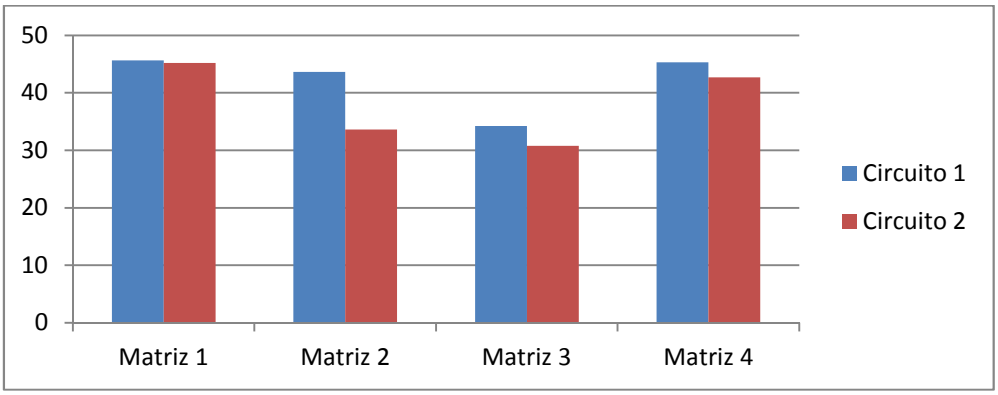
8.1. Comparación de los tiempos de respuesta de ambos circuitos

Ahora se analizan los tiempos de respuesta a cada una de las matrices insertadas, se puede notar que también hay reducciones (ver Tabla 4).

Tiempos de respuesta del dispositivo			
Matriz	**Circuito 1**	**Circuito 2**	**Diferencia**
1	45.648 ns.	45.209 ns.	0.439 ns
2	43.617 ns.	33.603 ns.	10.014 ns.
3	34.228 ns.	30.79 ns.	3.438 ns.
4	45.29 ns.	42.677 ns.	2.613 ns.
Promedio	42.19575 ns.	38.06975 ns.	4.126 ns.

Tabla 4. Comparación de los tiempos de desempeño de la FPGA.

Al igual que en el caso de las ocupaciones, esta información se aprecia de mejor manera en la gráfica presentada (ver gráfica 2).



Gráfica 2. Comparación de los tiempos de respuesta de ambos circuitos en nanosegundos.

9. Comparación con trabajos previos

En la literatura existen trabajos enfocados al algebra matricial implementados en FPGAs como plataforma. Estos se observan en perspectiva con los diseños aquí presentados en la Tabla 5.

Referencias	**[7]**	**[8]**	**Diseño 1 (instanciación)**	**Diseño 2**
FPGA	Virtex-6 LX240T	Artix 7 C7A100T	Virtex 6 LX240T	Virtex 6 LX240T
Algoritmo	Laplace	Laplace	Laplace	Laplace
Matriz	4x4	3x3	4x4	4x4
Tipo de número	Punto flotante de 32 bits.	Entero de 2 bits	Entero de 8 bits	Entero de 8 bits
Tiempo (ns)	7130.00	15.45	42.20	38.07
Slices	610	0	1025	929
LUTs	1615	12	2130	1938
DSP	28	25	56	44

Tabla 5. Comparación de los resultados con trabajos previos.

Se resalta el hecho de que los tiempos de respuesta tanto del diseño 1 y 2 son menores comparados con [7]; en el caso de [8] el tiempo es aún menor, sin embargo, los números son de 2 bits y su implementación se realiza en una FPGA con mayores prestaciones. Se observa también que en el diseño 2 el uso de DSPs optimiza doce bloques en contraste con el diseño 1.

10. Conclusiones

A través de la realización de este trabajo se constata el hecho de que efectuar operaciones aritméticas a nivel binario implica un gran consumo de recursos, por lo que la necesidad de diseñar circuitos que proporcionen respuestas en el menor tiempo posible, así como la utilización de la menor cantidad de bloques operacionales, se convierte cada día en una realidad más prioritaria.

En este caso en particular, el detectar que doce multiplicaciones se repiten, llevó a la optimización de los tiempos de respuesta y los recursos de la FPGA seleccionada, en especial, los bloques DSP's, cuya disminución va de 56 a 44 lo que representa un ahorro del 21%.

Como trabajos futuros se propone el realizar este ejercicio con matrices de orden mayor, con números que utilicen mayor cantidad de bits para su representación o la implementación de otras operaciones matriciales.

11. Referencias

[1] S. Almalki, "New parallel algorithms for finding determinants of NxN matrices". World Congress on Sousse Computer and Information Technology (WCCIT). 2013.

[2] W. Eberly, “On Computing the Determinant and Smith Form of an Integer Matrix”. 41st Annual Symposium on Foundations of Computer Science. 2000.

[3] S. I. Grossman. Álgebra lineal. 6a. ed. 2008. McGraw-Hill. México. 169-172 pp.

[4] B. Holanda, “An FPGA-Based Accelerator to Speed-Up Matrix Multiplication of Floating Point Operations”. IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW). 2011. 306-309 pp.

[5] Z. Jovanovic, “FPGA accelerator for floating-point matrix multiplication”. Computers & Digital Techniques, IET. Vol. 6. No.4. 249-256 pp.

[6] X. Lei, “Cloud Computing Service: the Case of Large Matrix Determinant Computation”. IEEE Transactions on Services Computing. Vol. X. No. X. 2014.

[7] J. D. Quesenberry, “Communication Synthesis for MIMO Decoder Algorithms”. Faculty of the Virginia Polytechnic Institute and State University. 2011.

[8] Fco. Plascencia, J. J. Raygoza, “Implementación de un circuito custom DSP en FPGAs para cálculo de determinantes 3x3, y matriz inversa de matrices ortogonales 3x3”. ReCIBE. Año 4. No. 2. Mayo 2015.

[9] Y. G. Tai, “Accelerating Matrix Operations with Improved Deeply Pipelined Vector Reduction”. IEEE Transactions on Parallel and Distributed Systems- Vol. 23 No. 2. 2012. 202-210 pp.

[10] X. Wang. “Performance Optimization of an FPGA-Based configurable multiprocessor for matrix operations”. IEEE International Conference on Field-Programmable Technology (FPT). 2003 303-306 pp.

[11] H. Yang. “FPGA-based Vector Processing for Matrix Operations”. Fourth International Conference on Information Technology. 2007.

[12] Virtex-6 Family Overview. http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf. Mayo de 2015.

[13] Virtex-6 FPGA Data Sheet: DC and Switching Characteristics. http://www.xilinx.com/support/documentation/data_sheets/ds152.pdf. Mayo de 2015.

[14] Virtex-6 FPGA DSP48E1 Slice. http://www.xilinx.com/support/documentation/user_guides/ug369.pdf. Mayo de 2015.