ANÁLISIS DEL ALGORITMO DE CORRIMIENTO DE FASE DE CUATRO PASOS PARA IMPLEMENTACIONES VLSI

FOUR-STEP PHASE SHIFT ALGORITHM ANALYSIS FOR VLSI IMPLEMENTATIONS

Jorge Alfonso Flores Centeno
Universidad de Guadalajara / Centro Universitario de Ciencias Exactas e Ingenierías, México
jorgef415@gmail.com

Marco Antonio Gurrola Navarro
Universidad de Guadalajara / Centro Universitario de Ciencias Exactas e Ingenierías, México
marco.gurrola@cucei.udg.mx

José Antonio Muñoz Gómez
Universidad de Guadalajara, Centro Universitario de la Costa Sur, México
jose.munoz@cucsur.udg.mx

Ramón Chávez Bracamontes
Tecnológico Nacional de México / Instituto Tecnológico de Ciudad Guzmán, México
rachavez@itcg.edu.mx

Omar Aguilar Loreto
Universidad de Guadalajara / Centro Universitario de la Costa Sur, México
omar.aguilar@cucsur.udg.mx

Recepción: 16/septiembre/2019 **Aceptación:** 7/noviembre/2019

Resumen

En este artículo se realiza un análisis del algoritmo de corrimiento de fase de cuatro pasos para su implementación en circuito integrado digital. Como resultado de este análisis, estaremos en posibilidades de implementar un microprocesador de aplicación específica optimizado para ejecutar el algoritmo mencionado. Este tipo de algoritmos encuentran su campo de aplicación en reconstrucción tridimensional de objetos mediante la proyección de patrones de franjas corridos en fase. La idea es que siguiendo el procedimiento de análisis aquí presentado al final conoceremos cual es la cantidad mínima de registros que debe contener el procesador y la longitud mínima en bits para representar los datos sin perder precisión en el resultado final. Así mismo, conoceremos las operaciones que la unidad aritmético lógica debe ser capaz de realizar y la cantidad de memoria RAM externa necesaria

en nuestro sistema. Este tipo de análisis es un paso previo y necesario a la descripción del procesador mediante un lenguaje de descripción de hardware y la elaboración del plano del circuito integrado con la ayuda de herramientas de síntesis digital.

Palabras Clave: algoritmo CORDIC, fase envuelta, síntesis VLSI, reconstrucción 3D.

Abstract

This article analyzes the four-step phase shift algorithm for its implementation in a digital integrated circuit. As a result, we will be able to implement a specific application microprocessor optimized to execute the aforementioned algorithm. These types of algorithms find their field of application in 3D surface reconstruction by fringe projections techniques. The idea is that following the analysis procedure presented here at the end we will know what is the minimum number of records that the processor must contain and the minimum length in bits is to represent the data without losing precision in the final result. Likewise, we will know the operations that the logical arithmetic unit must be able to perform, and the amount of external RAM needed in our system. This type of analysis is a previous and necessary step to the description of the processor through a hardware description language and the elaboration of the integrated circuit layout with the help of digital synthesis tools.

Keywords**:** *3D-reconstruction, CORDIC algorithm, VLSI synthesis, wrapped phase.*

1. Introducción

La reconstrucción 3D es una técnica mediante la cual se recupera el relieve de un objeto tomando varias imágenes para observar la distorsión de una serie de patrones de franja corridos en fase proyectados sobre éste [Quan, 2010]. La cantidad de imágenes debe ser de por lo menos tres imágenes. Entre mayor número de imágenes se tiene la capacidad de reducir el efecto del ruido añadido inherente al proceso mismo de la toma de las imágenes [Flores, 2018]. Para el presente artículo, se emplean un algoritmo con cuatro imágenes.

Las aplicaciones del modelado 3D abarcan desde el análisis del relieve de un objeto en una versión tridimensional con sus propiedades físicas para la reconstrucción de artefactos para la industria, impresiones en 3D para diseño de prototipos, restauración de piezas, escáner biométrico, fabricación de prótesis médicas entre otros [Gorthi, 2010, Geng, 2011].

El proceso numérico de reconstrucción 3D a partir de las imágenes o fotografías originales se puede subdividir en dos grandes partes:

- Algoritmo para la obtención de la fase envuelta. Existen varias técnicas como el esquema AIA (Algoritmo de Iteración Avanzada), o los métodos de corrimiento de fase de 3 o más pasos. Para nuestro caso específico emplearemos el algoritmo de corrimiento de fase de 4 pasos. En el presente trabajo nos interesamos en la implementación de un algoritmo para la obtención de la fase envuelta llamado algoritmo de corrimiento de faso de cuatro pasos.
- Algoritmo de desenvolvimiento de fase. En este algoritmo actualmente ya se cuentan con soluciones altamente optimizadas.

La realización numérica de algoritmos es actualmente de uso corriente en la solución de problemas de ciencia e ingeniería y en el diseño de instrumentos electrónicos. Los algoritmos se pueden programar vía software para ejecutarse en microprocesadores (computadora o dispositivos móviles), en microcontroladores o en procesadores digitales de señales, todos ellos de propósito general. Este tipo de procesadores cuenta con una gran cantidad de recursos hardware para ejecutar un muy amplio conjunto de instrucciones máquina. En general ocurre que para implementar un algoritmo cualquiera vía software sólo se emplea una pequeña parte de estas instrucciones dejando al procesador subutilizado. Los registros, buses y la unidad aritmético lógica (ALU) del procesador también quedan generalmente subutilizados pues es común que, para representación de datos, no es necesaria la cantidad de bits incluida en los anchos de datos estándar, usualmente 32 o 64 bits. De manera alternativa se pueden proponer arquitecturas de hardware digital específicamente diseñadas para la ejecución de un algoritmo [Chávez, 2016, Kung,

1979, Ciaccio, 2014]. Empleando técnicas de síntesis digital, estas arquitecturas, llamadas procesadores de propósito específico, pueden mapearse a un circuito integrado (chip de silicio) o a un arreglo de compuertas programables en el campo (FPGA, por sus siglas en inglés) [Vlăduţiu, 2012, Slade, 2013].

El diseño de una arquitectura de propósito específico tiene un costo de ingeniería mucho más elevado que una implementación en software. Este costo se justifica si se requieren cientos de dispositivos, para el caso de FPGA, o decenas de miles de chips, para el caso de los circuitos integrados. En este último caso el precio por unidad puede bajar hasta unas cuantas decenas de pesos en comparación de los varios miles de pesos que cuesta una computadora o un dispositivo móvil [Barr, 2007]. La alternativa de un procesador de propósito específico también se justifica cuando se requiere un dispositivo de tamaño muy reducido, de muy bajo consumo de potencia o con tiempos de ejecución muy bajos. Por otra parte, en el diseño de cualquier arquitectura de circuito integrado es deseable emplear una baja cantidad de recursos hardware (p. ej., compuertas digitales, bits de memoria) pues esto repercute en el tamaño o área del circuito integrado que a su vez es proporcional al costo final del circuito integrado. Puesto que al mejorar alguna de las métricas mencionadas a veces sucede que otra empeora, comúnmente hay que buscar un balance entre ellas, sin descuidar satisfacer cabalmente los requerimientos mínimos del sistema [Barr, 2007].

En el presente artículo abordamos el problema de analizar el algoritmo de corrimiento de fase de cuatro pasos para su implementación en un circuito integrado VLSI (Very Large Scale Integration), tal que, este último contenga únicamente los recursos de hardware necesarios para entregar los resultados con la máxima precisión posible dadas las condiciones limitantes inherentes a este tipo de aplicación.

2. Métodos

Algoritmo de corrimiento de fase de cuatro pasos

Para realizar una reconstrucción 3D existen diversos métodos como las técnicas donde se hace incidir luz estructurada sobre una superficie con un patrón de franjas

para adquirir múltiples imágenes fotográficas -corriendo la fase- y finalmente se analizan los patrones de franjas obtenidos. La superficie del objeto induce una distorsión en los patrones de franjas adquiridos por una cámara CCD, es decir, la topografía del objeto queda implícita en la fase. El patrón distorsionado se puede analizar por diferentes técnicas para calcular la profundidad del relieve. Un sistema de perfilometría básico, como se muestra en la figura 1, está conformado por un dispositivo activo (proyector) que proyecta un patrón de franjas sobre una superficie, un sensor de imagen (cámara CCD) para adquirir una captura de la superficie con la proyección, y un procesador (computadora) para diseñar digitalmente el patrón estructurado y ejecute el algoritmo para encontrar los datos de la superficie 3D.

Por medio de la perfilometría por corrimiento de fase, el cambio de fase nos permite sensar/medir reconstrucciones de alta resolución. Se tiene la limitante de que se requieren múltiples imágenes de superficie y requieren mayor tiempo de procesamiento para su reconstrucción [Tornero, 2018].

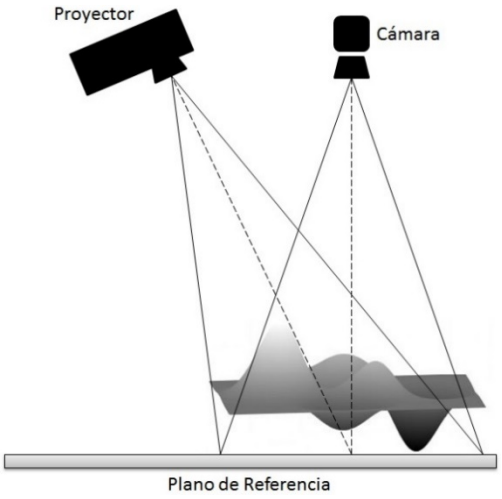
El problema de reconstrucción 3D por proyección de franjas consta de dos partes:

- Algoritmo para la recuperación de la fase envuelta (en el presente artículo se trabajó en esta parte del algoritmo).
- Algoritmo para el desenvolvimiento de fase.

Para la recuperación de la fase envuelta podemos emplear el algoritmo de AIA [Wang, 2007] o el algoritmo de tres o más pasos [Zhan, 2017], entre otros. El algoritmo de tres o más pasos tiene la ventaja de ser un método local (pixel por pixel) basado en la evaluación de una expresión matemática. Por otro lado, el algoritmo de AIA es un esquema iterativo que requiere estimaciones de parámetros globales (que abarcan la imagen completa). En el presente trabajo hemos seleccionado el algoritmo de 4 pasos respecto al algoritmo de 3 pasos, este esquema tiene una mejor relación señal a ruido aditivo, generado con la cámara, el proyector y demás partes del sistema óptico. Además, el esquema seleccionado nos permite filtrar el tercer armónico de los patrones capturados.

El sistema óptico para la reconstrucción 3D se muestra en la figura 1. Consta de un proyector que proyecta un patrón de franjas periódicas sinusoidales verticales en

tonos de gris. Las franjas verticales se proyectan sobre el objeto, siendo T el periodo en unidades de longitud que presentan las franjas en el plano de referencia a nivel del objeto. Mientras que el proyector y el objeto permanecen fijos, los patrones proyectados son corridos en fase y la cámara captura las imágenes con los patrones verticales distorsionados por la superficie del objeto analizado. Cada pixel de cada imagen entregada por la cámara es un valor de niveles de gris de 8 bits [Ordoñes, 2015].



Fuente: Elaboración propia.

Figura 1 Sistema óptico para la reconstrucción 3D mediante proyección de franjas.

El algoritmo de corrimiento de fase de cuatro pasos requiere de una serie de imágenes, I_n (ecuación 1), con patrones de franjas corridos en fase de $\delta n = n\pi T/2$, donde.

$$I_n(x,y) = A(x,y) + B(x,y)\cos(f(x,y) + d_n), \quad n = 0,1,2,3 \qquad (1)$$

Donde T es el periodo de las franjas, $A(x,y)$ intensidad de fondo, $B(x,y)$ amplitud, y el corrimiento equiespacionado está denotado por d_n . La información del objeto analizado se encuentra en la fase angular $\varphi(x,y)$. Las imágenes adquiridas $In(x,y)$ son de dimensiones x_{max} por y_{max} pixeles.

En particular, emplearemos cuatro capturas de imágenes ecuación 1, para poder estimar la fase envuelta del objeto [Okada, 1991]. Las diferencias entre pixeles

correspondientes de estas imágenes se comparan para calcular la fase envuelta empleando una función de arco-tangente.

La ecuación 2 corresponde al método de 4-pasos para estimar pixel a pixel la fase envuelta. Es decir, obtenemos la información 3D del objeto analizado envuelto en fase.

$$\varphi(x,y) = arctan\frac{I_4(x,y) - I_2(x,y)}{I_1(x,y) - I_3(x,y)} \tag{2}$$

El segundo y último paso en la reconstrucción 3D sería aplicar un algoritmo de desenvolvimiento de fase. Sin embargo, esta parte no será tratada en el presente trabajo. De lo expuesto en esta sección podemos ver que:

- El sistema deberá contar con memoria suficiente para almacenar las 4 imágenes I_n de x_{max} por y_{max} pixeles y cada valor de pixel requiere de 8 bits. Por lo tanto, se requiere un total de $4 \cdot x_{max} \cdot y_{max} \cdot 8$ bits para almacenar las 4 imágenes. Estos son los datos de entrada del algoritmo. Por ejemplo, para imágenes de 1000 por 1000 pixeles requerimos un aproximado de 4 MB. Para esta cantidad de memoria, en lugar de incluirla dentro del mismo circuito integrado que el procesador que estamos desarrollando, resulta más económico contar con una memoria RAM externa.
- La ecuación 2 constituye el algoritmo de corrimiento de fase de cuatro pasos. Esta ecuación deberá aplicarse un total de $x_{max} \cdot y_{max}$ veces. En cada vez deberán recuperarse de la memoria RAM externa 4 valores de niveles de gris, uno por cada imagen, y al final de cada pasada el algoritmo entregará un valor de ángulo de fase envuelta para el pixel (x, y) en turno.
- Nótese la ecuación 2 que la entrada del argumento para el arco tangente es un cociente de dos valores. Esta fracción puede interpretarse geométricamente como un valor de abscisa, el numerador, y un valor de ordenada, el denominador. De manera que este par de valores puede verse como las coordenadas de un punto en el plano cartesiano del cual queremos obtener el arco tangente. Este argumento de entrada en la forma de dos valores coordenados de un punto es el formato adecuado para obtener la

función ATAN2. Recordemos que ATAN2 nos entrega valores de ángulos en el rango $(-\pi, \pi)$, mientras que la función arco tangente normal nos entrega valores únicamente en la mitad de este rango.

- Una consecuencia positiva de lo explicado en el punto anterior es que no tenemos que realizar la división indicada en la ecuación 2, pues si lo hiciéramos ya no podríamos obtener la ATAN2.

Diseño de una arquitectura específica para un algoritmo

En este punto creemos conveniente explicar algunos lineamientos para la generación de arquitecturas específicas a un algoritmo. Estas recomendaciones son producto del diseño de circuitos integrados digitales:

- Determinar las operaciones requeridas. Lo primero que hay que hacer es una revisión de las operaciones requeridas en el algoritmo. Esto incluye operaciones aritméticas, de lectura y escritura, así como cualquier otro tipo de operación que requieran de algún periodo de tiempo para su ejecución. En un primer acercamiento conviene clasificar las operaciones en: 1) muy simples (requieren pocos recursos y poco tiempo de ejecución): desplazamiento de bits, incrementos y decrementos de registros, sumas, restas, comparaciones y transferencia de datos entre registros; 2) de mediana complejidad: multiplicación, división, inverso multiplicativo, raíz cuadrada; y 3) de alta complejidad: multiplicación matricial, inversa de una matriz, cálculo de logaritmos, funciones trigonométricas, etc.; Las operaciones de lectura y escritura de datos de una memoria RAM externa se pueden considerar no muy complejas (no requieren muchas compuertas) pero de velocidad media.
- Determinar la cantidad de operaciones. Con esta información podemos ir haciendo estimaciones de tiempos de ejecución del algoritmo conforme lo vamos refinando.
- Usar sólo lo necesario. La arquitectura debe incluir módulos para realizar todas las operaciones necesarias, pero no debe incluir recursos que no se vayan a usar. Para decirlo de manera más técnica, debemos ir

confeccionando una trayectoria de datos (o data-path) capaz de ejecutar todas las operaciones requeridas en nuestro algoritmo.

- Encontrar balance entre métricas de desempeño. Anteriormente hemos mencionado que en el diseño de una arquitectura podemos buscar optimizar alguna métrica de desempeño (área de silicio, velocidad de procesamiento, bajo consumo de potencia, tiempo de diseño reducido, etc.) y que, debido a que estas métricas en ocasiones son opuestas, debe llegarse a un balance aceptable entre ellas. Un ejemplo bastante común son la velocidad de procesamiento y el área de silicio requerida, pues se puede tratar de optimizar una de las dos a costa de castigar la otra.
- Operaciones complejas algorítmicamente o con hardware dedicado. Sí la velocidad no es tan crítica, lo siguiente es ver si algunas de las operaciones de complejidad media o elevada se pueden descomponer en operaciones más simples y ejecutarse de manera algorítmica. Por ejemplo, la multiplicación y la división se puede realizar secuencialmente como una serie de sumas y restas, respectivamente. (En el presente trabajo la función ATAN2 se implementará con el algoritmo CORDIC que requiere de una tabla de valores precargados y una secuencia de desplazamientos, sumas y restas [Schweers, 2002].) Cuando la velocidad es la métrica principal, entonces hay que considerar el incluir módulos hardware para acelerar las operaciones de complejidad media y alta.
- Punto fijo o punto flotante. Es más complejo diseñar módulos aritméticos para punto flotante que para punto fijo. De ahí que recomendamos emplear la representación de punto flotante únicamente cuando la representación de punto fijo no produce los resultados esperados.
- Cantidad de bits para los datos. Analizar el impacto de la cantidad de bits usados para representar los datos en la precisión final del resultado del algoritmo. Ya sea que intentemos usar punto fijo o punto flotante, hay que averiguar cuál es la mínima cantidad de bits para representar nuestros datos que cumpla con los requerimientos de precisión en los resultados. Recordemos que no estamos atados a tamaños de registro estándar como

32 o 64 bits. Estamos en libertad de escoger una cantidad de bits arbitraria para representar nuestros datos. Un método efectivo para ver el efecto que tiene la cantidad de bits elegida se explica a continuación: una vez que se tenga una versión tentativa de la arquitectura hay que realizar simulaciones del algoritmo en Matlab (o software similar) incluyendo todas las operaciones involucradas, pero en las cuales podamos emplear resoluciones de bits truncadas, iniciando con 8 bits, luego con 9 bits, y así sucesivamente hasta un máximo de 51 bits para punto fijo y 63 bits para punto flotante. Estos resultados los compararemos con los obtenidos al emplear resolución completa de 64 bits la cual consideraremos como nuestra referencia "exacta". En punto flotante de 64 bits en realidad se emplean sólo 52 bits para la mantisa de los números (51 bits más un bit '1' extra), por eso no tiene caso llevar las simulaciones de punto fijo a un truncamiento mayor a 51 bits.

- Truncamiento FLOOR o ROUND. Durante estas simulaciones hay que incluir también el efecto del tipo de truncado en las operaciones. Por ejemplo, una multiplicación de dos factores de 10 bits produce un producto de 20 bits, de ahí que hay que truncar los 10 bits menos significativos. Este truncamiento puede ser tipo FLOOR (que simplemente elimina los bits a partir de la posición de truncamiento) o de tipo ROUND (en la que se incrementa en 1 Bit Menos Significativo o LSB al resultado truncado si la cantidad truncada es igual o mayor a ½ LSB). Los módulos aritméticos para el truncamiento tipo FLOOR suelen ser más sencillos mientras que los del tipo ROUND que requieren ciertos tipos de ajustes y comparaciones para corregir el bit de más bajo orden en el valor truncado.
- Cantidad de registros. Analizar la cantidad de registros internos y de RAM externa necesarios para mantener en memoria todos los datos requeridos para efectuar las operaciones descritas en nuestro algoritmo. El sistema debe contar con registros internos suficientes para mantener en memoria los datos iniciales, los resultados parciales y el resultado final. Sin embargo, nótese que algunos de los datos pueden volverse innecesarios después de haberse empleado. Esto liberará registros o localidades de memoria.

- Cantidad de memoria RAM externa. La RAM se emplea en los casos dónde se maneja un elevado número de datos, como procesamiento de imágenes o de video. Analizar el algoritmo y manipularlo para buscar en primer lugar que requiera pocos registros internos y en segundo lugar que requiera una baja cantidad de memoria RAM.

Lo anterior nos permite plantearnos una arquitectura general del sistema. Al final del proceso anterior debemos terminar con una propuesta de arquitectura que incluye un conocimiento de:

- Representación numérica a emplear (punto fijo o punto flotante) y su cantidad de bits.
- Tipo de truncamiento que se usará ROUND o FLOOR.
- Cantidad de registros y de RAM requeridos.
- Lista de operaciones que el sistema debe ser capaz de hacer:
    - ✓ Operaciones que tendrán un módulo aritmético dedicado.
    - ✓ Operaciones que se realizarán de manera algorítmica empleando los módulos de operaciones más simples.
- Buses de entrada y salida, y señales de control del sistema.
- Secuencia completa de operaciones para completar el algoritmo.

El siguiente paso es la implementación a detalle del algoritmo empleando las técnicas usuales de organización o microarquitectura de computadoras. Esto incluye el uso de un lenguaje de descripción de hardware como VHDL o Verilog y el uso de un set de herramientas CAD para la síntesis de circuitos digitales. En este caso, estas técnicas se usarán para implementar un microprocesador de propósito específico.

Operaciones en el algoritmo de corrimiento de fase de 4 pasos

El algoritmo de corrimiento de fase por el método de cuatro pasos consiste en la aplicación de la ecuación 2 para cada conjunto de cuatro valores (x, y) en las imágenes de entrada. Las entradas del algoritmo son los valores de nivel de gris

en cuatro imágenes de dimensiones x_{max} por y_{max}. La salida está compuesta por $x_{max} \cdot y_{max}$ valores de ángulos en el rango de $-\pi \; a \; \pi$.

Observando la ecuación 2 podemos ver que se realizan las siguientes operaciones en cada una de las $x_{max} \cdot y_{max}$ pasadas del algoritmo:

- 4 operaciones de lectura de 1 byte cada uno. Leer la RAM externa los 4 datos $I_1(x,y)$, $I_2(x,y)$, $I_3(x,y)$, $I_4(x,y)$ de 8 bits cada uno, correspondientes al pixel en turno.
- Escritura en memoria RAM de un ángulo en el rango de $-\pi \; a \; \pi$ (cuyo ancho de bits en este momento aún no conocemos pero que al final del presente análisis resultará ser de 16 bits).
- 2 operaciones de resta de datos de 9 bits (pues las restas entre los datos I1 a I4, aunque son de 8 bits, pueden generar resultados de 9 bits).
- Cálculo del arco tangente que por facilidad de diseño y economía de hardware hemos decidido emplear el método CORDIC según se explica a más detalle en la siguiente sección.

Arco tangente mediante algoritmo CORDIC

Buscando economía de hardware decidimos emplear el algoritmo CORDIC (Coordinate Rotation Digital Computer) para obtener la función ATAN2, pues este método no emplea multiplicación ni división, únicamente desplazamientos (equivalentes a multiplicaciones o divisiones por potencias de 2), sumas y restas [Chávez, 2018]. A diferencia de otros métodos como series de potencias, CORDIC requiere de una carga de valores precargada que incluye aproximadamente un valor por cada bit de resolución empleado en la representación del ángulo. Como se mencionó antes, la división indicada dentro del argumento del arcotan de la ecuación 2 no se realiza, sino que se emplea para hacer una adaptación del método CORDIC que resuelve el ATAN, para resolver el ATAN2. El numerador y denominador del argumento se consideran como las coordenadas (abscisa y ordenada) de un punto en el plano cartesiano. Para adaptar el método, al principio se verifica si el punto se encuentra en el cuadrante II o III y si es así se cambia de signo a ambas coordenadas para llevar el punto a los cuadrantes IV o I y se guarda

un término de corrección de ángulo con valor de π o de $-\pi$, respectivamente. Este término de corrección se añade al valor obtenido con el algoritmo CORDIC. Si el punto se encontraba originalmente en el cuadrante I o IV, entonces no se hace cambio de signo en los valores de las coordenadas iniciales y el término de corrección es cero. El rango del valor final obtenido en este caso es un ángulo que va de $-\pi\ a\ \pi$. Los primeros 24 valores de los ángulos elementales se muestran en la tabla 1.

Tabla 1 Ángulos elementales para CORDIC con 24 iteraciones.

Iteración	Valor θ_n en decimal	θ_n en binario con resolución de 26 bits
1	0.78539816	000.11001001000011111101101
2	0.46364761	000.01110110101100011001110
3	0.24497866	000.00111110101101101110101
4	0.12435499	000.00011111110101011011101
5	0.06241881	000.00001111111110101010110
6	0.03123983	000.00000111111111110101010
7	0.01562373	000.00000011111111111110101
8	0.00781234	000.00000001111111111111110
9	0.00390623	000.00000000111111111111111
10	0.00195312	000.00000000011111111111111
11	0.00097656	000.00000000001111111111111
12	0.00048828	000.00000000000111111111111
13	0.00024414	000.00000000000011111111111
14	0.00012207	000.00000000000001111111111
15	0.00006104	000.00000000000000111111111
16	0.00003052	000.00000000000000011111111
17	0.00001526	000.00000000000000001111111
18	0.00000763	000.00000000000000000111111
19	0.00000381	000.00000000000000000100000
20	0.00000191	000.00000000000000000010000
21	0.00000095	000.00000000000000000000111
22	0.00000048	000.00000000000000000000011
23	0.00000024	000.00000000000000000000001
24	0.00000012	000.00000000000000000000001

Fuente: Elaboración propia.

Tenemos que guardar tantos valores en una ROM en nuestro procesador como iteraciones deseemos realizar. Truncaremos los valores de la tabla a la cantidad de bits que resulte en el análisis de la siguiente sección. Una vez truncados, conservaremos únicamente los valores de la tabla que sean diferentes de cero. (De antemano comentamos que la cantidad de bits será de 14 y que la tabla contendrá

12 valores diferentes de cero. Es decir que el algoritmo CORDIC contará con 12 iteraciones. Tómese en cuenta que en el truncamiento se hace un redondeo tipo ROUND.)

Las ecuaciones 3 y 4 son las que se emplearon para la obtención de estos valores.

$$tan\theta_n = \frac{1}{2^n} \tag{3}$$

$$\theta_n = \text{atan}(tan\theta_n) \tag{4}$$

Donde n es el número de iteración (Tabla 1) de 1 a 24.

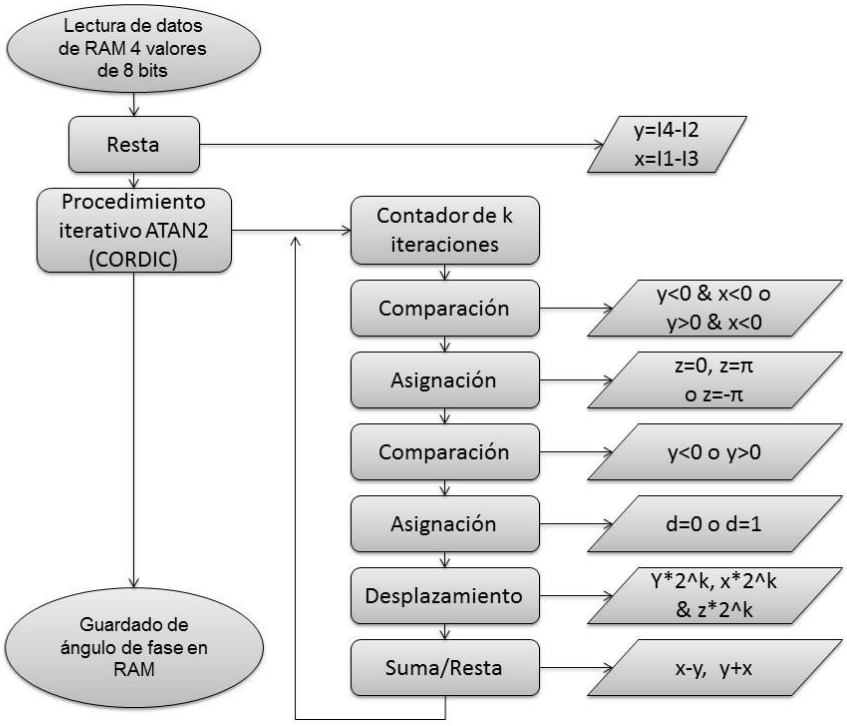
Mayores detalles para la implementación en hardware del algoritmo CORDIC se muestran en [Dinechin, 2015, Duprat, 1993]. Después de analizarlo, encontramos que en cada cálculo del algoritmo CORDIC para calcular el arco tangente con k iteraciones requerimos de las siguientes operaciones básicas:

- k comparaciones
- 2·k desplazamientos
- 2·k sumas

3. Resultados

Análisis de precisión

Tras haber analizado el algoritmo de corrimiento de fase de cuatro pasos contamos ahora con el tipo y la cantidad exacta de operaciones involucradas incluido el algoritmo CORDIC. Además, ya contamos con el procedimiento exacto paso a paso para implementar el mencionado algoritmo. Este procedimiento se muestra en la figura 2. Puesto que estamos diseñando hardware “a la medida” no estamos atados a emplear representaciones de dato estándar (que generalmente vienen en anchos de bits en potencias de dos, p.ej., 16, 32 o 64 bits), sino que podemos emplear anchos de bits arbitrarios. Por ahorro de hardware deseamos no emplear una cantidad de bits excesiva en la cual los bits de más bajo orden no tengan ningún efecto en la precisión final. Deseamos averiguar cuál es la representación mínima que nos permita explotar el nivel de precisión máximo a partir de los datos de las imágenes recibidos por las cámaras.



Fuente: Elaboración propia.

Figura 2 Procedimiento para ejecutar el algoritmo.

Si es posible, hay que escoger un rango adecuado para nuestra representación de punto fijo para que no queden bits de alto orden sin utilizar. En el caso que nos ocupa, los resultados de ángulos de fase estarán en el rango de $-\pi\ a\ \pi$, es decir, de -3.1416 a 3.1416, por lo tanto sólo es necesario conservar con dos bits para la parte entera y uno para el signo y con eso nos alcanza para representar números en el rango de -3.5 a 3.5 (tomando en cuenta que contamos con al menos un bit en la parte fraccionaria para representar el 0.5).

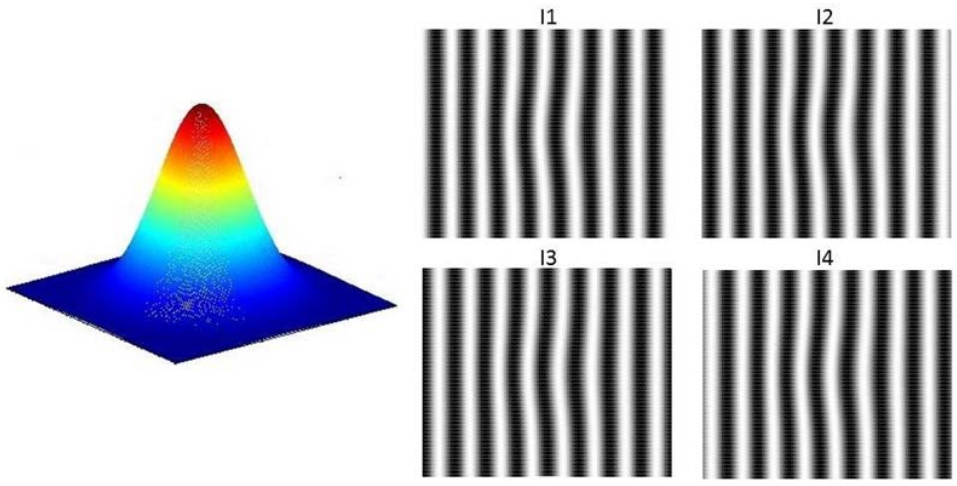
Las fuentes de error en nuestro sistema abarcan:

- La resolución finita de en los niveles de gris provenientes de las cámaras. Esta resolución usualmente es de sólo 8 bits.
- El ruido presente en los niveles de gris que usualmente va desde 3 hasta 5 bits menos significativos (LSB).
- La cantidad finita de bits, *d*, empleada internamente en el procesador para la representación de los valores numéricos (incluida la tabla de valores precargados).

- Tipo de truncado FLOOR o ROUND empleado en las operaciones de desplazamiento a la derecha (divisiones por potencias de 2) así con en los valores de la tabla de valores precargados.

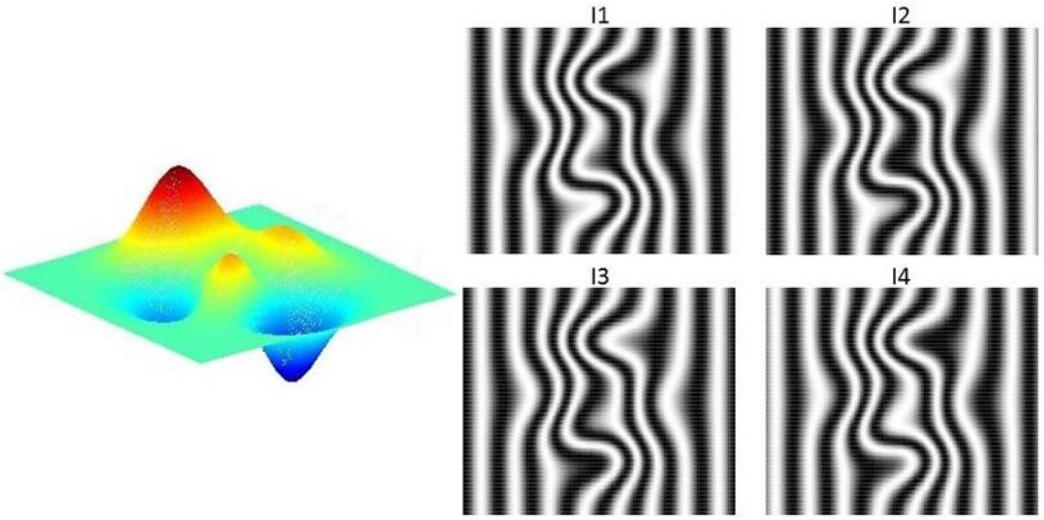
El procedimiento que seguimos para realizar el análisis de la precisión en el resultado de ángulo final consiste en codificar el procedimiento de la figura anterior en Matlab (o software similar). Sin embargo, las variables (que se almacenarán en registros) así como los resultados de las operaciones de desplazamiento, de suma/resta y de los valores de la tabla precargada deben tener la opción de poderse establecer o redondear a *d* bits.

Preparamos un banco de prueba de dos imágenes: una campana Gaussiana de revolución y la función peaks de Matlab que se muestran en figuras 3 y 4. Con base en la ecuación 1, simulamos un patrón con 8 periodos de franjas cuyo nivel de gris tiene un comportamiento sinusoidal. Conforme a los datos simulados, el relieve queda implícito en el argumento del coseno, obtenemos cuatro imágenes desplazando la fase de forma equiespaciada. Las imágenes tienen un total de x_{max} por y_{max} pixeles. Los niveles de gris obtenidos se convierten a un formato de 8 bits en un rango de 0 a 255. A cada punto de las imágenes se añade ruido con distribución normal con desviaciones estándar de 3 y 5 LSB de niveles de gris (es decir 3/256 y 5/256 del rango total de niveles de gris).



Fuente: Elaboración propia.

Figura 3 Función “Gaussian” y las tres imágenes sin ruido.



Fuente: Elaboración propia.

Figura 4 Función “peaks” y las tres imágenes sin ruido.

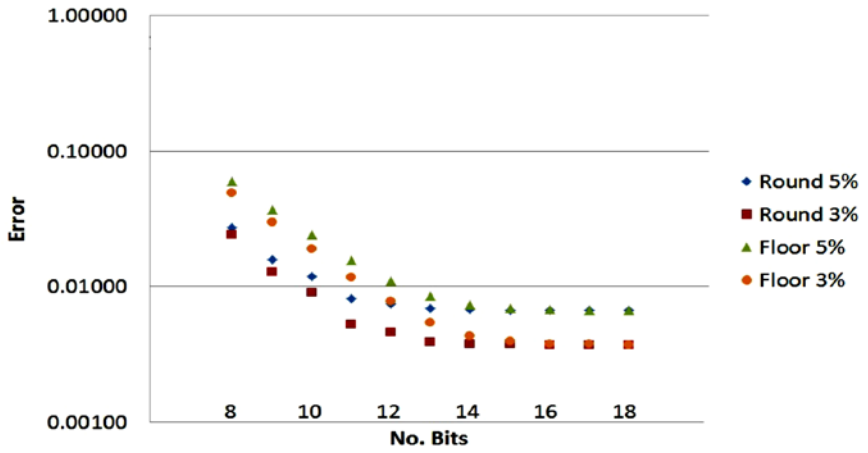
Se realiza una simulación con la precisión máxima posible en Matlab de 64 bits y con las cuatro imágenes de franjas sin ruido añadido. Aplicamos el algoritmo con la precisión máxima de 64 bits. A los valores de fase envuelta así obtenidos les aplicaremos alguna de las conocidas funciones para el desenvolvimiento de fase [Okada. 1991]. Los valores de fase desenvuelta así obtenidos lo consideraremos como los valores “exactos” de referencia. Respecto a este conjunto de valores “exactos” compararemos los valores con cantidades de bits truncadas y con ruido añadido. Ahora procederemos a realizar pruebas a partir de $d = 8$ bits de resolución empleando las imágenes con ruido añadido. Iremos incrementando *d* hasta notar que nuestra precisión final ya no mejore. Para cada valor de d, se harán simulaciones para las desviaciones estándar en el ruido de las imágenes de 3 y de 5 LSB, y se probarán los métodos de redondeo FLOOR y ROUND.

Para cuantificar la precisión total y poder comparar las diferentes pruebas entre sí, emplearemos la siguiente definición del error absoluto medio en la ecuación 5.

$$E_{\text{abs}} = \frac{\sum_{i=1}^{x_{\max}} \sum_{j=1}^{y_{\max}} \left| \varphi_{(i,j)} - \varphi'_{(i,j)} \right|}{x_{\max}\, y_{\max}} \tag{5}$$

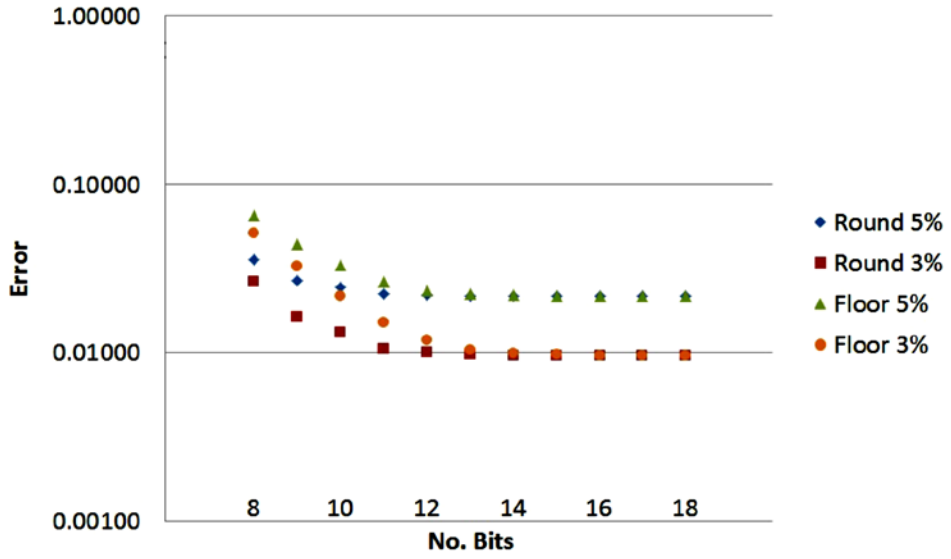
Donde $\varphi(i,j)$ es el valor de fase obtenido con el algoritmo de precisión a *d* bits, $\varphi'(i,j)$ es el valor de referencia “exacto” obtenido con precisión a 64 bits, x_{max} y y_{max} son las dimensiones horizontal y vertical de la imagen.

Los resultados de las pruebas se muestran en figuras 5 y 6, Gaussian y Peaks. Anticipamos que, dado que se obtuvieron buenos resultados con la representación numérica de punto fijo, no fue necesario hacer pruebas con representación numérica de punto flotante:



Fuente: Elaboración propia.

Figura 5 Error absoluto medio de la función Gaussian.



Fuente: Elaboración propia.

Figura 6 Error absoluto medio de la función Peaks.

Cómo puede observarse, si se está empleando truncado ROUND bastan 14 bits para alcanzar el máximo de precisión con cualquiera de las dos figuras y de los dos niveles de ruido. En cambio, si se emplea el truncado tipo FLOOR se requiere de

16 bits para lograr explotar el máximo de precisión que el sistema puede otorgar en las pruebas con ambos niveles de ruido. El redondeo ROUND permite ahorrar dos bits en la representación de datos para este problema. La única operación que requiere de truncamiento es el desplazamiento y no es muy complejo implementar ROUND en este caso. Así que preferimos emplear el truncamiento ROUND sobre el truncamiento FLOOR para este problema.

Concluimos que nuestro sistema empleará 14 bits para la representación de los datos y empleará tipo de redondeo ROUND tanto en las operaciones de desplazamiento como en los valores de la tabla precargada (en formato binario). A continuación, mostramos los valores de la tabla 2, truncados a 14 bits con redondeo tipo ROUND. Nótese que a partir del valor número 13 los valores se vuelven 0 por lo tanto la tabla a guardar sólo requiere de 12 valores y el algoritmo CORDIC para resolver el arco tangente se resuelve en 12 iteraciones.

Los registros necesarios para almacenar los valores temporales y el sumador/restador deberán ser de 14 bits. Mientras que el registro de desplazamiento deberá ser capaz de desplazar hasta un total de 12 posiciones a la izquierda, pero contar con capacidad de redondeo tipo ROUND. Esto implica que el data-path deberá tener la capacidad de añadirle un 1 al resultado desplazado si el bit de más alto orden de la parte truncada es 1.

Tabla 2 Ángulos elementales para CORDIC con 14 iteraciones.

Iteración	Valor θ_n en decimal	Valor de θ_n en binario (truncada a 14 bits)
1	0.78539816	000.11001001000
2	0.46364761	000.01110110110
3	0.24497866	000.00111110110
4	0.12435499	000.00011111111
5	0.06241881	000.00010000000
6	0.03123983	000.00001000000
7	0.01562373	000.00000100000
8	0.00781234	000.00000010000
9	0.00390623	000.00000001000
10	0.00195312	000.00000000100
11	0.00097656	000.00000000010
12	0.00048828	000.00000000001
13	0.00024414	000.00000000000
14	0.00012207	000.00000000000

Fuente: Elaboración propia.

Arquitectura general propuesta

A continuación, se realiza un análisis de los registros requeridos para conservar los datos empleados en cada pasada del algoritmo como se muestra en la tabla 3 (es una pasada por cada uno de los xy pixeles que conforman la imagen). Se requieren 5 registros de 14 bits: X, Y, Z, XD y YD; y un registro de 4 bits. Por simplicidad únicamente se muestra el algoritmo para las operaciones del algoritmo en un solo pixel. Se requieren otros 4 registros de 11 bits cada uno para hacer los bucles externos para recorrer la totalidad de los $x_{max} \cdot y_{max}$ pixeles. (Con registros de 11 bits podemos hacer bucles para poder analizar imágenes de hasta 2048 por 2048 pixeles.)

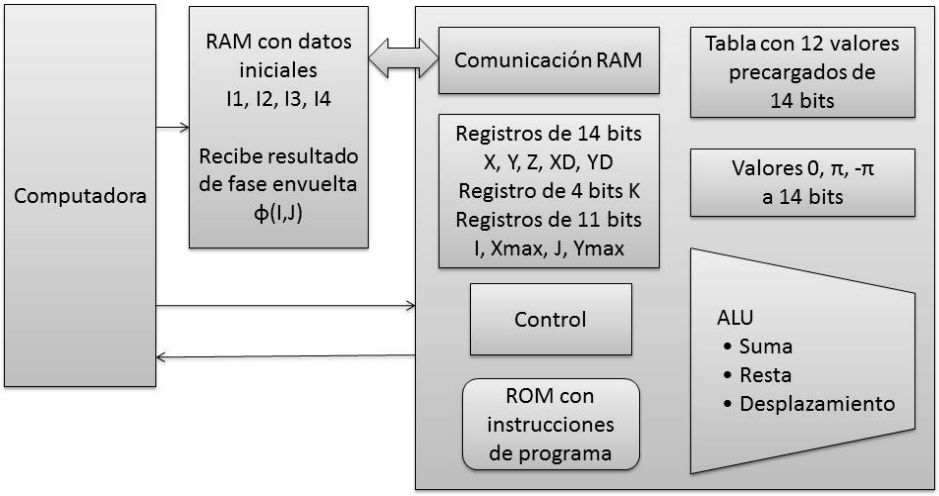
Tabla 3 Descripción del funcionamiento de la arquitectura.

Descripción general	Operación detallada	Registros afectados
Carga de valor inicial	Leer I4 de RAM y ponerlo en Y (un multiplexor a la salida de Y permite elegir entre Y o –Y)	Y
Carga de valor inicial y resta	Leer I2 de RAM y restarlo a Y	Y
Carga de valor inicial	Leer I1 de RAM y ponerlo en X (un multiplexor a la salida de X permite elegir entre X o –X)	X
Carga de valor inicial y resta	Leer I3 de RAM y restarlo a X	X
Corrección de valores iniciales e inicialicación de ángulo	Iniciar Z con valor 0, π o –π dependiendo del cuadrante del punto (X,Y), e Invertir valores de X y Y si X es negativo	X, Y, Z
Inicialización de contador	Inicializar K en ceros	K (contador de 4 bits)
Inicio del Bucle CORDIC	Incrementar K en 1 (el valor de K es el número de iteración)	K
Bucle CORDIC	Desplazar Y a la derecha K posiciones y guardar en YD	YD
Bucle CORDIC	Desplazar Y a la derecha K posiciones y guardar en XD	XD
Bucle CORDIC	Guardar en X la resta X–YD	X
Bucle CORDIC	Guardar en Y la suma Y+XD	Y
Bucle CORDIC	Guardar en Z la resta de Z menos el elemento K de la tabla (un multiplexor a la salida de la tabla permite elegir entre el valor K o su negativo)	Z
Fin del bucle CORDIC	Comparar registro K con el número 12, sí aún no es igual repetir el bucle.	
Guardar Resultado	Guardar en la RAM el resultado de ángulo de fase en formato de punto fijo de 14 bits	

Fuente: Elaboración propia.

En figura 7, se bosqueja un diagrama con la propuesta de arquitectura la cual debe incluir todas las operaciones y características obtenidas en el presente análisis y que aquí enumeramos:

- Memoria RAM externa suficiente para $4 \cdot x_{max} \cdot y_{max}$ bytes. En cada pasada del algoritmo, una vez utilizados los 4 valores de imagen de entrada las localidades de memoria RAM correspondientes quedan desocupadas para poder guardar el ángulo de fase resultante (de sólo 14 bits).
- Interfaz para leer y escribir en la memoria RAM externa.
- 5 registros de 14 bits, 4 registros de 11 bits y 1 registro de 4 bits.
- ALU de 14 bits con capacidad de sumar, restar y desplazar hasta 12 posiciones a la derecha. El desplazamiento debe contar con truncamiento tipo ROUND.
- Tabla precargada con 12 valores de 14 bits cada uno, 11 bits para la parte fraccionaria.
- Valores de 0, $-\pi$ y π en punto fijo redondeados a 11 bits para la parte fraccionaria.
- Módulo de control.
- Memoria ROM interna para microprograma. El ancho de bits de las instrucciones se determina en una etapa más adelante cuando se analice la arquitectura minuciosamente para la síntesis del procesador.
- Interfaz de entrada, salida del sistema.



Fuente: Elaboración propia.

Figura 7 Diagrama a bloques general de la arquitectura.

A partir de este punto la implementación de esta arquitectura en circuito integrado puede hacerse con ayuda de algún lenguaje de descripción de hardware, como VHDL o Verilog, en conjunto con herramientas de síntesis digital. Para ello hay que aplicar las técnicas de diseño y verificación habituales que vienen en los textos de organización o microarquitectura de computadoras, con la salvedad de que en este caso se implementará un microprocesador de aplicación específica.

4. Discusión

En el presente artículo se ha mostrado a detalle la metodología para analizar el algoritmo de corrimiento de fase de 4 pasos. Este algoritmo es importante para la reconstrucción 3D de objetos. El procedimiento incluye: análisis manuales para encontrar el tipo y la cantidad de operaciones básicas involucradas en el algoritmo; la implementación de la operación arco tangente mediante operaciones de desplazamiento y sumas/restas; así como simulaciones en computadora para determinar la cantidad mínima de bits tal que no exista pérdida en el resultado final del algoritmo. Los resultados encontrados se pueden emplear para la obtención de un plano de circuito integrado fabricable siguiendo los procedimientos de descripción en lenguaje VHDL o Verilog y el empleo de herramientas de síntesis digital. Además, se presentaron lineamientos generales que pueden servir para el análisis de otros algoritmos con miras a su implementación en VLSI.

5. Bibliografía y Referencias

[1] Quan, C., Chen, W. & Tay, J. (2010). Phase-retrieval techniques in fringe-projection profilometry. Opt. Lasers Eng., N°48, pag. 235-243.

[2] Flores, J., Stronik, M., Muñoz, A., Torales, G., Ordoñez, S. & Cruz, A. (2018). Dynamic 3D shape measurements by iterative phase shifting algorithms and colored fringe patterns. Optics Express., N°10, pag. 12403-12414.

[3] Gorthi, S. & Rastogi P. (2010). Fringe projection techniques: whither we are?. Optics and Lasers in Engineering., N°48, pag. 133-140.

[4] Geng, J. (2011). Structured-light 3D surface imaging: a tutorial. Advances in Optics and Photonics. N°2, pag. 128-160.

[5] Chávez, R., Gurrola, M., Jiménez, H. & Bandala, M. (2016). VLSI architecture of a Kalman filter optimized for real-time applications. Electronics Express, N° 13, pag. 1-11.

[6] Kung, H. (1979). Let's design algorithms for VLSI system. Caltech Conference on VLSI. pag. 65-90.

[7] Ciaccio, E., Biviano, A. & Garan, H. (2014). Software algorithm and hardware design for real-time implementation of new spectral estimator. Biomedical Engineering OnLine.

[8] Vlăduţiu, M. (2012). Computer arithmetic. Algorithms and hardware implementations. Springer.

[9] Slade, G. (2013). The fast fourier transform in hardware: A tutorial based on an FPGA implementation.

[10] Barr, K. (2007). ASIC Dising in the silicon sandbox: A complete guide to building mixed-signla integrated circuits. The McGraw-Hill Companies.

[11] Tornero, N., Trujillo, G., Anguiano, M., Mendoza, P., Salas D. & Corral, L. (2018). Color profilometry techniques: A review. Optica Pura y Aplicada, N° 51, pag. 1-26.

[12] Wang, Z. & Han, B. (2007). Advanced iterative algorithm for randomly phase-shifted interferograms with intra- and inter-frame intensity variations. Optics and Lasers in Engineering, N° 45, pag. 274-280.

[13] Zhan, G., Tang, H., Zhong, K., Li, Z., Shi, Y. & Wang C. (2017). High-speed FPGA-based phase measuring profilometry architecture. Optics Express, N° 25.

[14] Ordoñes, S., Muñoz, A. & Flores, J. (2015). An efficient computational phase extraccion from arbitrary phase-shifted fringes patterns.

[15] Chávez, R., Vidrios, V., Villaseñor, J., Bracamontes, H., Molinar, J. & Gurrola, M. (2018). Implementación en chip VLSI del algoritmo CORDIC para la solución de funciones trigonométricas. Pistas Educativa, N° 40, pag. 1500-1511.

[16] Schweers, R. (2002). Descripción en VHDL de arquitecturas para implementar el algoritmo CORDIC.

[17] Chen, H. & Hsu, S. (2016). An Efficient FPGA-Based Parallel Phase Unwarpping Hardware Architecture, IEEE Transactions on Computational Imaging.

[18] Okada, K., Sato, A. & Tsujiuchi, J. (1991). Simultaneous calculation of phase distribution and scanning phase shift in phase shifting interferometry. Optics Communications N° 84, pag. 118-124.

[19] Dinechin, F. & Istoan, M. (2015). Hardware implementations of fixed-point Atan2, HAL.

[20] Duprat, J. & Muller, J. (1993). The CORDIC algorithm: new results for fast VLSI implementation. IEEE Transactions on Computers, N° 42, pag. 168-178.