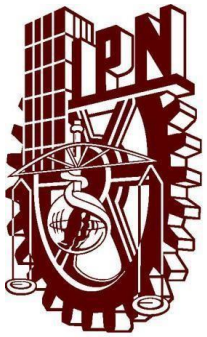


INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA

UNIDAD PROFESIONAL "ADOLFO LÓPEZ MATEOS" ZACATENCO

"DISEÑO DE UNA MAC EN UN MICROCONTROLADOR USANDO CORDIC PARA APLICAR PROCESOS DE PDS"

REPORTE TÉCNICO

PARA OBTENER EL TITULO DE:

INGENIERO EN COMUNICACIONES Y ELECTRÓNICA.

P R E S E N T AN

DANIEL DOMINGO BELTRAN MORALES

JUAN PABLO CARDOZA MAGAÑA

FERNANDO GRANADOS ALVAREZ

ASESORES

ING. JORGE ALBERTO GÓMEZ GARCIA

M.EN.C. GUILIBALDO TOLENTINO ESLAVA

CIUDAD DE MÉXICO, JULIO DE 2022

INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA
UNIDAD PROFESIONAL "ADOLFO LÓPEZ MATEOS"

R E P O R T E T É C N I C O

QUE PARA OBTENER EL TÍTULO DE **INGENIERO EN COMUNICACIONES Y ELECTRÓNICA**

POR LA OPCIÓN DE TITULACIÓN **CURRICULAR**

DEBERA (N) DESARROLLAR **C. DANIEL DOMINGO BELTRAN MORALES**

C. JUAN PABLO CARDOZA MAGAÑA

C. FERNANDO GRANADOS ALVAREZ

"DISEÑO DE UNA MAC EN UN MICROCONTROLADOR USANDO CORDIC PARA APLICAR PROCESOS DE PDS"

DISEÑAR UNA UNIDAD MAC COMPLEJA (MULTIPLY-ACCUMULATE OPERATION) EN EL MICROCONTROLODOR MSP430G2553 UTILIZANDO CORDIC PARA PROCESOS Y APLICACIONES DE PDS.

- **TEORÍA DEL PROCESAMIENTO DIGITAL DE SEÑALES**
- **MÓDULO DE MULTIPLICADOR-ACUMULADOR COMPLEJO (MAC-COMPLEJA)**
- **PROPUESTA DE DISEÑO DE LA MAC COMPLEJA**
- **VALIDACIÓN DE LA MAC: PRUEBAS Y RESULTADOS**

CIUDAD DE MÉXICO, A 01 DE JULIO DEL 2022.

A S E S O R E S

ING. JORGE ALBERTO GÓMEZ GARCIA

M. EN. C. GUILIBALDO TOLENTINO ESLAVA

DEPARTAMENTO DE I.C.E

M. EN C. ITZALÁ RABADÁN MALDA
JEFA DE LA CARRERA DE INGENIERÍA
EN COMUNICACIONES Y ELECTRÓNICA

Autorización de uso de obra

Instituto Politécnico Nacional

P r e s e n t e

Bajo protesta de decir verdad los que suscriben **BELTRAN MORALES DANIEL DOMINGO, CARDOZA MAGAÑA JUAN PABLO y GRANADOS ALVAREZ FERNANDO**, manifestamos ser autores y titulares de los derechos morales y patrimoniales de la obra titulada "**DISEÑO DE UNA MAC EN UN MICROCONTROLADOR USANDO CORDIC PARA APLICAR PROCESOS DE PDS**", en adelante "**La Tesis**" y de la cual se adjunta copia en dos CD'S, por lo que por medio del presente y con fundamento en el artículo 27 fracción II, inciso b) de la Ley Federal del Derecho de Autor, otorgamos al **INSTITUTO POLITÉCNICO NACIONAL**, en adelante **El IPN**, autorización no exclusiva para comunicar y exhibir públicamente total o parcialmente en medios digitales o en cualquier otro medio; para consulta y aportación del desarrollo del proyecto para futuros trabajos relacionados con el tema de **"La Tesis"** por un periodo de **4 AÑOS** contado a partir de la fecha de la presente autorización, dicho periodo se renovará automáticamente en caso de no dar aviso expreso a **El IPN** de su terminación.

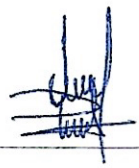
En virtud de lo anterior, **El IPN** deberá reconocer en todo momento nuestra calidad de autores de **"La Tesis"**.

Adicionalmente, y en nuestra calidad de autores y titulares de los derechos morales y patrimoniales de **"La Tesis"**, manifestamos que la misma es original y que la presente autorización no contraviene ninguna otorgada por los suscritos respecto de **"La Tesis"**, por lo que deslindamos de toda responsabilidad a **El IPN** en caso de que el contenido de **"La Tesis"** o la autorización concedida afecte o viole derechos autorales, industriales, secretos industriales, convenios o contratos de confidencialidad o en general cualquier derecho de propiedad intelectual de terceros y asumimos las consecuencias legales y económicas de cualquier demanda o reclamación que puedan derivarse del caso.

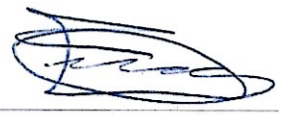
Ciudad de México, a 21 de Agosto de 2022

ATENTAMENTE

BELTRAN MORALES DANIEL DOMINGO



CARDOZA MAGAÑA JUAN PABLO



GRANADOS ALVAREZ FERNANDO

INDICE

pág.

INDICE DE TABLAS

Pág.

INDICE DE FIGURAS

Pág.

DISEÑO DE UNA MAC EN UN MICROCONTROLADOR USANDO CORDIC PARA APLICAR PROCESOS DE PDS.

INTRODUCCIÓN

DESCRIPCIÓN DEL PROBLEMA

Al tratar de implementar una Unidad Acumulador Multiplicador Compleja (MAC), la cual es el núcleo de todo procesamiento digital de señales e indispensable para que este pueda llevarse a cabo y a su vez, tiene múltiples aplicaciones en el campo del Procesamiento Digital de Señales (DSP), y también ,dado que el (DSP) es una herramienta indispensable para resolver problemas de la vida diaria y se diferencia de otras técnicas modernas en el campo de la computación, porque trabaja con señales del "mundo real". Es decir, señales que generalmente provienen directamente de sensores que monitorean fenómenos como: vibraciones sísmicas, imágenes, ondas de sonido, etc. Es por esta razón que muchas personas necesitan saber cómo implementar una MAC de manera adecuada y eficiente dado que sin una MAC el DSP no podría llevarse a cabo.

La comunidad estudiantil en general se enfrenta a un problema muy común al querer implementar una MAC, el cual es que los microcontroladores o tarjetas que tienen un módulo integrado para desarrollar esta, son de propósito específico, es decir, son tarjetas de desarrollo de un costo elevado y con un nivel de complejidad alto en cuanto al nivel de programación necesario para hacer que estas funcionen, ya que en la mayoría de las veces, la persona interesada debe aprender un lenguaje particular de programación para este tipo de tarjetas.

Gran cantidad de personas han realizado trabajos afines a esta problemática, pero todos ellos solo han desarrollado MAC´s en dispositivos de alto nivel y propósito específico como los DSP´s o los FPGA´s, esto quiere decir que únicamente han creado aplicaciones con los módulos que estas tarjetas de desarrollo contienen para implementar una MAC y con ella hacer múltiples operaciones en el área del procesamiento digital.

Algunas personas han abordado esta problemática implementando diversos lenguajes de programación, dentro de los lenguajes de programación de hardware, los más común en este tipo de aplicaciones son: el lenguaje VHDL (VHSIC -Very

High Speed Integrated Circuits- Hardware Description Lenguage) ya que se pueden realizar diseños en tres diferentes niveles de programación y existen diversos tipos de herramientas para el desarrollo de estas técnicas: ALTERA, XILINIX y VERIBEST. En cada nivel de este diseño jerárquico hay componentes que son usados para describir el diseño. A más alto nivel o niveles más abstractos, tenemos un número pequeño de componentes poderosos tales como sumadores y memorias. El lenguaje de descripción del hardware tal como VHDL se propone para la implementación de todo diseño jerárquico ya que provee algunos grados de uniformidad a través de varios niveles.

Por otro lado, también se ha utilizado el lenguaje c, c++. Java e incluso Python, Otro lenguaje de programación utilizado es la Herramienta llamada MATLAB, la cual es una herramienta de nivel medio muy poderosa y ligeramente fácil de entender. También se ha utilizado el algoritmo CORDIC para desarrollar aplicaciones en los FPGA's.

Dado que el multiplicador/acumulador (MAC) realiza multiplicación a alta velocidad, multiplicación con adición acumulativa, multiplicación con resta acumulativa, funciones de saturación y de puesta a cero y que normalmente, la multiplicación y suma se obtiene en un solo ciclo de reloj, entonces para el desarrollo de este trabajo se utilizará la herramienta de programación llamada MATLAB, ya que se desarrollara la MAC utilizando dos operaciones básicas, las cuales son la suma y la multiplicación, y MATLAB es un lenguaje de nivel medio, de fácil comprensión y poderoso para este tipo de aplicaciones, además de que es de fácil acceso para los estudiantes, de hecho, diversas escuelas de nivel publico dan licencias a los estudiantes para poder utilizar este lenguaje de programación.

Otro lenguaje que se utilizará es el lenguaje de programación llamado C++, ya que después de desarrollar todo el procedimiento en MATLAB se procederá a pasarlo a lenguaje C++ para así poderlo implementar en una tarjeta de desarrollo de uso comercial de fácil acceso para los estudiantes, también se utilizará el algoritmo CORDIC como apoyo para poder diseñar una MAC.

Al término de esta investigación se logrará tener un procedimiento claro y sencillo al cual tendrán acceso los estudiantes de ingeniería y carreras afines en las que se desarrolle una MAC desde cero para la tarjeta de desarrollo MSP430G2553, una ventaja de ello, es que estas tarjetas tienen muchas funciones y son baratas, así que cualquier alumno podría adquirirla. Pero no solo se lograra lo anterior, ya que con esta investigación también se garantizará que las personas interesadas en este tema

puedan desarrollar una MAC en cualquier tarjeta de trabajo de bajo nivel o nivel medio ya que lo que se realizará es un procedimiento donde se enlisten los pasos a seguir para implementar una MAC de principio a fin en cualquier tarjeta de trabajo.

OBJETIVO

Diseñar una unidad MAC compleja (Multiply–accumulate operation) en el microcontrolador MSP430G2553 utilizando el algoritmo CORDIC para procesos y aplicaciones de PDS.

OBJETIVOS PARTICULARES

- Revisión de antecedentes teóricos tales como: MAC, el formato binario punto fijo, convertidor analógico-digital y el convertidor digital-analógico.
- Revisión y simulación del algoritmo CORDIC.
- Simulación de una MAC compleja.
- Diseño en hardware de una MAC compleja.
- Aplicar la MAC compleja propuesta en alguna aplicación de PDS.

JUSTIFICACIÓN

Este proyecto se realizará porque como tal no hay una MAC implementada en un microcontrolador de bajo nivel o nivel medio con la cual los estudiantes de la materia de PDS o materias afines del área de comunicaciones puedan desarrollar prácticas a nivel físico en estas materias, lo cual provoca que el aprendizaje en estas áreas no se logre en su totalidad, es por ello que se realizará el procedimientopara desarrollar una MAC, la cual será de acceso libre, gratuito y muy importante, entendible para cualquier persona, en especial las del área de ingeniería y carreras afines para que ellos puedan implementarla en la tarjeta de desarrollo que ellos tengan a la mano.

Con esto se lograrán dos cosas muy importantes, en primer lugar, los estudiantes entenderán de una manera más clara las materias análogas al procesamiento digital de señales, y en segunda, se ayudará a las personas de bajos recursos a tener acceso a un material, que a veces solo se puede tener acceso comprando una tarjeta de desarrollo de propósito específico, la cual es muy costosa y en algunos casos difícil de programar, entonces con esta investigación se obtendrán múltiples beneficios.

Se debe tener presente que todo lo que conlleva el procesamiento digital de señales no se podría realizar sin una MAC, sin ella, las operaciones de PDS no se podrían llevar a cabo. Las aplicaciones que tiene una MAC son muchas, entre ellas, está la convolución, la correlación, la transformada rápida de Fourier (DFT) entre otras.

Entonces, si se logra implementar una MAC en un microcontrolador de gama baja, en consecuencia, se podrán implementar aplicaciones de PDS rápidas, sencillas, entendibles y eficientes consiguiendo así la posibilidad de hacer prácticas físicas en la materia de PDS y lograr un aprendizaje más sólido en estas materias.

La motivación Principal de este proyecto es apoyar a los futuros alumnos de la Licenciatura en Ingeniería en Comunicaciones y Electrónica para que puedan implementarlo en sus materias de procesamiento digital de señales o materias afines a esta rama, en sus proyectos personales o en cualquier otra aplicación que ellos consideren, la idea de ayudar a una gran parte de la población del Instituto Politécnico Nacional, así como a la población de carreras afines de bajos recursos da una gran motivación al equipo, con este proyecto se busca el ahorro y la eficiencia de los materiales que casi todos los estudiantes tienen a la mano, más que nada que estos son de fácil acceso en cuanto a precio, disponibilidad y manejo ligeramente sencillo.

Otra de las principales motivaciones es que hasta el momento nadie ha implementado una MAC en un microcontrolador de gama media o baja, a pesar de que una MAC es indispensable, por lo tanto, se buscara ser los primeros en implementarlo para un microcontrolador comercial como es el caso de la tarjeta de desarrollo MSP30G25

ALCANCE

Se proporcionará un procedimiento que ayude a generar una operación MAC, que puede ser utilizada por las futuras generaciones de estudiantes de la Licenciatura en Ingeniería en Comunicaciones y Electrónica (o semejantes) de la ESIME, así como en cualquier otra universidad donde se impartan este tipo de carreras, la cualserá una herramienta con la cual llevar un curso más didáctico de procesamiento digital de señales. Se podrá llevar a cabo el desarrollo profesional de los estudiantes de manera práctica. Al utilizar el procedimiento para generar la MAC en el microcontrolador de su preferencia sin la necesidad de adquirir uno sofisticado (gama alta) y por ende con mayor costo.

Debido a la optimización de la MAC compleja se podrán realizar aplicaciones como filtrado de señales, la operación de convolución donde se involucra la exponencial compleja y la correlación a partir de la cual se obtiene la operación de convolución o también un sintetizador de señales con el uso de la serie de Fourier pero es necesario aclarar que dichos objetivos ya serían parte de un trabajo extra por parte del alumno interesado ya que para realizarlo hay algunas otras cuestiones que se deben de tomar en cuenta, sin embrago es importante mencionar que el elemento principal para poder realizar estas aplicaciones es la MAC misma que se desarrollara en este trabajo.

Al realizar mediante el entorno de trabajo de programación C++, utilizando como base el software IAR para generar el código, depurar y programar, se dejan las bases para que pueda ser utilizado con microcontroladores de otras marcas con capacidades similares a las del que se utilizará siendo un microcontrolador de la marca Texas Instruments y su respectiva tarjeta de programación.

A su vez se implementará el algoritmo CORDIC para hacer uso de las funciones trigonométricas usuales en microcontroladores que no posean la capacidad de procesar este tipo de operaciones de tal manera que sólo se hagan sumas y restas y comparaciones, para hacer más eficiente el código y de esta forma también poder implementarlo en cualquier microcontrolador de propósito general, que, aunque el MSP430G2553 tiene la capacidad de procesar funciones trigonométricas, así como variables tipo flotante de 6 dígitos decimales, algunos otros utilizados en el área de licenciatura no poseen esta capacidad. Se documentarán las pruebas de que este procedimiento funciona correctamente.

Se utilizará la representación de números en formato Qnm para la obtención de resultados, ya que cuando se requiera implementar la MAC en microcontroladores de gama baja se tiene una desventaja, ya que algunos de ellos no pueden manejar funciones trigometricas y con la ayuda de la representación Qnm y del algoritmo CORDIC se puede solucionar este problema.

CAPÍTULO I:

TEORÍA DEL PROCESAMIENTO DIGITAL DE SEÑALES.

En este capítulo se presentan las bases teóricas que es necesario conocer para poder desarrollar una MAC, se tratara la representación digital de números en sus representaciones de formato binario y en punto fijo (Qnm), así como las diferentes operaciones que se pueden realizar con estas representaciones, se presentara lo que es la digitalización, la conversión de digital a analógico así como también las fases necesarias para llevar a cabo la digitalización, que en este caso es el muestreo, la cuantificación y la codificación. También se tratará la interpolación e interpolación de orden cero, así como de la definición básica acerca de lo que es una MAC.

1.1.-FORMATOS DE REPRESENTACIÓN DIGITAL (FORMATO BINARIO EN PUNTO FIJO).

La representación de números en un formato de punto fijo es una generalización de la familiar representación decimal de un número como una cadena de dígitos con un punto decimal. En esta notación, los dígitos a la izquierda del punto representan la parte entera del número, y los dígitos a la derecha del punto decimal, representan la parte fraccional del número.

Representación en complemento 1

Complemento 1 es una forma particular de representar números positivos y negativos. Su forma es simple y bastante directa de entender. Todo número positivo posee su bit más significativo igual a 0. Los números negativos se obtienen con sólo negar (o complementar) el número positivo correspondiente. Un ejemplo parael caso de tres bits es mostrado en la tabla 1-1.

Tabla 1-1: Números binarios en complemento 1 [1].

000	0	111	0
001	1	110	–1
010	2	101	–2
011	3	100	–3

De la tabla anterior se puede observar que el número 0 posee dos formas distintas de representación. Ello ha llevado a problemas con la comparación por cero en algoritmos, por lo cual este tipo de representación no es utilizado. El término “complemento 1” se debe a que el número negativo se obtiene sólo complementando el patrón de bits del número positivo que se quiere pasar a negativo [1].

Representación en complemento 2

La representación en complemento 2 es una forma eficiente de representar números con signo en microprocesadores de punto fijo. La propiedad fundamental de este formato es que permite representar números negativos, por lo cual se utiliza el bit más significativo de la palabra binaria que se está manejando. Esto lleva a que en un formato de palabra de n bits, la capacidad de representación sea de hasta 2^{n-1} para números positivos y $2^{n-1} - 1$ negativos. El bit de signo será siempre el más significativo. La figura 1-1 muestra un esquema para n bits [1].

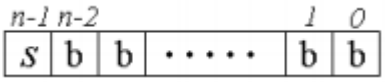


Figura 1-1: Palabra de n bits: S=bi de signo, b=bits que representan un número

Fuente: Huerta R, "Representación de números en punto fijo y flotante"

Con la forma de representación antes mostrada los números positivos tendrán el bit más significativo igual a 0. Si es un número negativo este bit será 1.

Para llegar a un número negativo en complemento 2 es necesario realizar las operaciones siguientes: al número positivo que se quiere convertir a negativo se le debe representar en formato binario, luego complementar cada bit (pasar los 1s a 0s y viceversa) y finalmente sumar 1. La tabla 1-2 muestra un ejemplo para el caso de tres bits.

Tabla 1-2: números binarios de 3 bits en complemento 2 [1].

Nº Positivo			Nº Negativo
1 = 001	110	110 + 1	111 = –1
2 = 010	101	101 + 1	110 = –2
3 = 011	100	100 + 1	101 = –3
4 = 100	011	011 + 1	100 = –4

Representación Qnm

Es una forma de representar números fraccionarios, es decir, números que tienen parte entera y parte fraccionaria. Qnm se denota a un número o a una secuencia de bits que representan a un número que tiene parte entera y parte fraccionaria [2].

Esta representación tiene un rango que está definido desde:

$$\boldsymbol{rango} = -2^{n-1}\ a\ 2^{n-1} - \frac{1}{2^m} \qquad [adimensional] \qquad (1\text{-}1)$$

Representación punto fijo $\boldsymbol{Q_{n.m}}$ con signo, donde:
n= representa "n" bits para la parte entera.

m= representa "m" bits para la parte decimal.

Donde: $\frac{1}{2^m}$: es la resolución que tiene esta representación y está en función de cuantos bits dedicamos para la parte fraccionaria [2].

Se le llama punto fijo porque no es posible tomar bits de la parte fraccionaria para ayudar a representar la parte entera y tampoco se pueden ocupar bits de la parte entera para ayudar a representar la parte fraccionaria, es decir, el punto permanece fijo, esta representación también tiene un rango, el cual fue descrito anteriormente, porque Qnm es un formato para representar números fraccionarios [1].

Ejemplo:

$Q_{1.2}$

Para esta representación se tendría un rango de:

$$\boldsymbol{n} = 1 \rightarrow -2^{n-1}\ a\ 2^{n-1} - \frac{1}{2^m} \qquad [adimensional] \qquad (1\text{-}2)$$

$$\boldsymbol{n} = -2^{1-1}\ a\ 2^{1-1} - \frac{1}{2^2} = -1\ a\ 0.75$$

$$\boldsymbol{m} = 2 \rightarrow \frac{1}{2^m} \qquad [adimensional] \qquad (1\text{-}3)$$

$$\boldsymbol{m} = \frac{1}{2^2} = 0.25$$

Por lo tanto, se tendrían tres bits y ocho posibles secuencias que se mostraran en la tabla 1-3: [2].

Tabla 1-3: secuencias válidas para 3 bits en las distintas representaciones (Formato Q1.2) [2].

Secuencia binaria	**representación entero sin signo**	**representación entero con signo**	**representación Qnm**
000	0	0	0
001	1	1	0.25
010	2	2	0.50
011	3	3	0.75
100	4	-4	-1
101	5	-3	-0.75
110	6	-2	-0.50
111	7	-1	-0.25

1.1.1.-Operaciones (suma, resta, multiplicación de números, pero en representación binaria en punto fijo).

Primero se definirán las 3 operaciones básicas con números binarios con signo y posteriormente se explican estas operaciones en formato Qnm.

Suma

Para la suma de dos o más números binarios debemos utilizar 4 reglas fundamentales que nos ayudan y facilitan el trabajo.

Reglas
0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0, con acarreo de 1 en la siguiente fila.

Otra regla muy importante por considerar es que dependiendo del número de bits con los que se representen los sumandos, dependerá la longitud del resultado, es decir, si el primer número binario a sumar es de 3 bits y el segundo de 3 bits, el resultado será un número de 4 bits, dado que se elige al número mayor de bits y se le suma uno, y de esa longitud será el resultado [3]. En la figura 1-2 se puede ver un ejemplo, en el que se aplican las reglas mencionadas anteriormente:

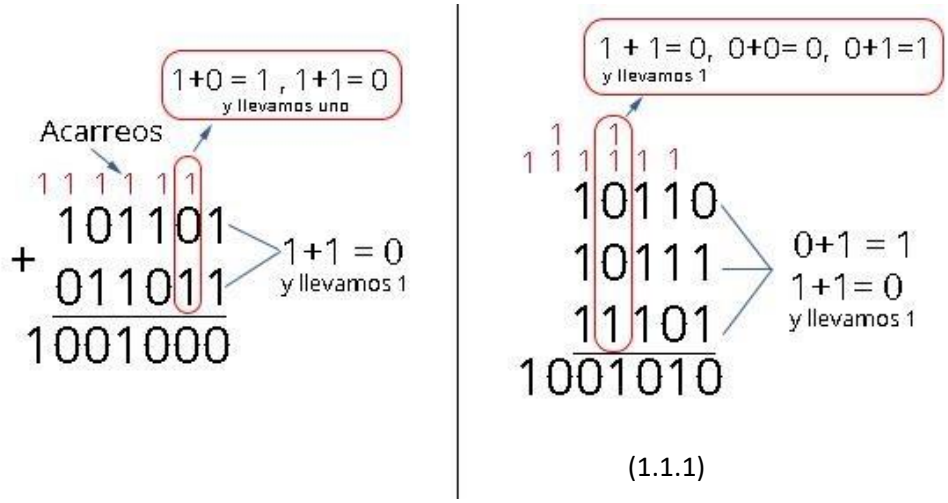


Figura 1-2: ejemplo para una suma con números sin signo.

Fuente: "Operaciones con números binarios y conversión a otros sistemas."

Como se utilizan números positivos sin signo el intervalo para representarlos será el siguiente:

$$0\ a\ 2^b - 1$$

Donde b es el número de bits, así que, si por ejemplo se tuvieran 3 bits.

$$0\ a\ 2^3 - 1 = 0\ a\ 8 - 1 = 0\ a\ 7$$

Entonces si se tienen 3 bits, se pueden representar 8 números, esto sería del 0 a 7, Esto se puede observar en la tabla 1-4: [2].

Tabla 1-4: Representación para números sin signo utilizando 3 bits [2].

representación entero sin signo	Código binario
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Resta

Para la resta se utiliza el complemento a 2, es decir, se hacen sumas, pero utilizando números con signo, para ello se siguen las mismas reglas mencionadas anteriormente para la suma, solo que ahora para saber el intervalo con el cual se pueden representar estos números con signo sería el siguiente:

$$-2^{b-1}\ a\ \ 2^{b-1} - 1 \qquad [adimensional] \tag{1-4}$$

Donde b es el número de bits, así que, si por ejemplo se tuvieran 3 bits.

$$-2^{3-1}\ a\ 2^{3-1}-1=-4\ a\ 3$$

Entonces si se tienen 3 bits, se pueden representar 8 números, esto sería del -4 a 3, Esto se puede observar en la tabla 1-5 [2].

Tabla 1-5: Representación para números con signo utilizando 3 bits [2].

representación entero sin signo	**Código binario**
0	000
1	001
2	010
3	011
-4	100
-3	101
-2	110
-1	111

Como ya se mencionó, para la resta se sigue las mismas reglas que la operación suma, solo que hay una ligera excepción , en una de las reglas mencionadas anteriormente, se dice que cuando se tiene un número de 3 bits y se suma con otro de 3 bits, el resultado será un número de 4 bits, es decir, se toma el máximo número de bits y se le suma uno para el resultado, en la resta ocurre lo mismo, solo que si se sabe que el resultado de la operación es de 4 bits , entonces se realiza un paso más , que se le llama la extensión se signó, lo que significa que el bit más significativo para ambos números se repite y se coloque al lado del bit más significativo, esto se puede observar con el siguiente ejemplo:

Si se suma 3+(-3):

De la tabla 1-3 se puede observar que estos números se representan como:

(3)→ 011

(-3)→ 101

Se sabe que esta suma debería de dar cero, así que se procede a realizarla:

011

+101

1000→ este número es 8 en decimal.

Por lo tanto, el resultado no es el correcto, entonces , aquí se aplica la extensión de símbolo, ya que se tienen dos números de 3 bits y por lo tanto el resultado debe de ser de 4 bits y entonces se realiza la extensión de símbolo que como se muestra a

continuación es repetir el bit más significativo, a continuación, se realizara la misma operación solo que ahora se colocara la extensión de símbolo:

0011

+1101

10000→ este número es 0 en decimal.

Se puede observar que en el resultado se obtienen 5 bits, pero solo se toman los 4 bits de resultado y el bit más significativo se desprecia [2].

Multiplicación binaria

En la escuela primaria se aprendió a multiplicar mediante la suma de una lista de multiplicandos trasladados, que se calculaban de acuerdo con los dígitos del multiplicador. Se puede utilizar el mismo método para obtener el producto de dos números binarios sin signo. La formación de los multiplicandos trasladados es trivial en la multiplicación binaria, puesto que los únicos valores posibles de los dígitos del multiplicador son 0 y 1 [4]. A continuación, en la figura 1-3 se puede ver un ejemplo:

11	1011	multiplicando
X 13	x 1101	multiplicador
33	1011	
11	0000	multiplicandos desplazados
143	1011	
	1011	
	10001111	producto

Figura 1-3: ejemplo de una multiplicación binaria.

Fuente: Martínez A, "Análisis de Arquitectura de multiplicadores digitales y su implementación en FPGA"

En general, cuando se multiplica un número de n bits por un número de m bits, para expresar el producto resultante se requieren como máximo n+m bits. [4]

Multiplicación binaria con signo

En la multiplicación con signo las reglas cambian ligeramente, la regla que se sigue conservando es que cuando se multiplica un número de n bits por un número de m bits, para expresar el producto resultante se requieren como máximo n+m bits. Lo que cambia ahora es que si, por ejemplo, se multiplica un número de 2 bits por otro de 4, el resultado que se obtendrá será un número de 6 bits, pero entonces al realizar la multiplicación se deben completar estos dos números haciendo la extensión de bits, es decir, el bit más significativo se repite tantas veces se requiera hasta llegar al número de bits esperados. Ya que se tiene el resultado, si por ejemplo el número esperado es de 6 bits, entonces solo se toman los 6 bits menos significativos como resultado y los demás se omiten. Después al bit más significativo de esos 6 bits elegidos se le da el valor negativo y se realiza una suma tomando en cuenta el valor de cada bit en decimal, al realizar esta suma se obtiene el resultado esperado. Todas estas reglas aplican para cualquier producto con números con signo y se mostrará en el ejemplo siguiente [2]:

Si se quiere multiplicar (3)x(-5)=-15

Se sabe que al número 3 se le puede expresar con 2 bits y al -5 con 4 bits, al sumar estas 2 cantidades, tendríamos un resultado de 6 bits por lo tanto se completan a estos números con la extensión se signó de la siguiente manera:

```
 x 000011
   111011
---------
   111011
  111011
---------
10110001
```

110001→ este número es -15 en decimal.

Después de obtener el resultado como se dijo anteriormente solo se toman 6 bits, ya que desde un inicio se sabía de cuantos bits sería el resultado, después de eso al bit más significativo de estos 6 bits se le da el valor negativo y se realiza una sumatoria de acuerdo con los valores de cada bit, esto se puede comprobar a continuación:

110001 → -32(1)+16(1)+8(0)+4(0)+2(0)+1(1)=-15

Después de explicar las operaciones básicas utilizando números enteros, ahora se procede a explicar estas operaciones en el formato Qnm.

1.1.2.-Operaciones (suma y multiplicación de números en formato Qnm)

Suma con Qnm

Cuando se tienen 2 números Qnm y se suman, suponiendo que los números Qnm son diferentes, se tendría:

$$Q_1 n_1 . m_1 + Q_2 n_2 . m_2 = Q_3 n_3 . m_3$$

Es decir:

$Q_1 n_1 . m_1 = Q_1 n_3 . m_3$

$Q_2 n_2 . m_2 = Q_2 n_3 . m_3$

Donde:

$n_3 = \max(n_1, n_2) + 1$

$m_3 = \max(m_1, m_2)$

En esta representación para la parte entera se hace la extensión de signo, es decir, suponiendo que n_3 es mayor. Que de echo así es, n_3 es mayora n_1 y a n_2, simplemente a través de la extensión de signo igualamos n_1 **y** n_2 a n_3 , y para la parte fraccionaria lo único que se hace es completar con ceros al número o a la derecha al número menor, es decir, si n_3 es menor a n_2 simplemente se le agregan los ceros faltantes para igualar las longitudes, ósea Q_1 y Q_2 se representa con $n_3 + m_3$ bits, se extiende el signo y se completa con ceros a la derecha del menor.

Ahora se mostrarán en las tablas 1-6 y 1-7 las representaciones para el formato Qnm para 1 bit, 4 bits respectivamente, la representación de 3 bits se puede observar en la tabla 1-3 mostrada anteriormente [2].

Tabla 1-6: secuencias válidas para 1 bits en las distintas representaciones (Formato Q1.1) [2].

Secuencia binaria	representación entero sin signo	representación Qnm
00	0	0
01	1	0.5
10	2	-1
11	3	-0.5

Tabla 1-7: secuencias válidas para 4 bits en las distintas representaciones (Formato Q2.2) [2].

Secuencia binaria	representación entero sin signo	Representación entera con signo	representación Qnm
0000	0	0	0
0001	1	1	0.25
0010	2	2	0.50
0011	3	3	0.75
0100	4	4	1
0101	5	5	1.25
0110	6	6	1.5
0111	7	7	1.75
1000	-8	-8	-2
1001	-7	-7	-1.75
1010	-6	-6	-1.5
1011	-5	-5	-1.25
1100	-4	-4	-1
1101	-3	-3	-0.75
1110	-2	-2	-0.5
1111	-1	-1	-0.25

Todo lo explicado anteriormente se puede observar en el siguiente ejemplo:

0.5+(-0.75)= -0.25

Como se puede observar el 0.5 según la tabla 1.5 es un formato Q1.1 y el -0.75, es un formato Q1.2, por lo tanto, el 0.5 y -0.75 se representan como:

0.5→0.1 y -0.75→1.01

Ahora recordando la propiedad de la suma que indica que la longitud del resultado es tomando el mayor número de bits de los sumandos y sumarle 1,se puede ver que el 0.5 se representa con dos bits y el -0.75 con 3 bits, por lo tanto el resultado sería de 4 bits, entonces estos números se completan para que ambos tengan esta longitud, para ello se siguen las reglas descritas anteriormente, se extiende el signo del punto decimal hacia la izquierda y hacia la derecha se completa con ceros según corresponda, así que la suma de estos números aplicándoles estas reglas quedarían de la siguiente manera [2]:

con la extensión se signó de la siguiente manera:

```
  00.10
+
  11.01
--------
```

11.11→ este número es un formato Q2.2 y si observamos en la tabla 1.6 si corresponde a --0.25 que era el resultado que se esperaba.

Multiplicación con Qnm

Cuando se multiplican dos números Qnm distintos, el resultado será un nuevo número Qnm donde:

$$Q_1n_1.m_1 * Q_2n_2.m_2 = Q_3n_3.m_3$$

De lo cual:

$n_3 = n_1 + n_2$

$m_3 = m_1 + m_2$

En la multiplicación ,si por ejemplo se multiplican 2 números de 2 bits cada uno, el resultado será de longitud 4, ya que se suman las longitudes de cada número en bits y de esa longitud tendrá que ser el resultado también, entonces , si a los númerosles faltan bits se completan únicamente con extensión de signo en el bit más significativo hacia la izquierda y algo muy importante es que el punto no se mueve, este permanece fijo. Al obtener el resultado, el punto se coloca sumando el número de posiciones en donde aparecía este en los números a multiplicar, después de eso, al bit más significativo se le asigna un valor negativo y después se procede a realizar la multiplicación de los valores de cada bit según su valor y así finalmente se comprueba el resultado. Todas estas reglas mencionadas anteriormente se pueden observar en el siguiente ejemplo [2]:

(-1)*(0.5)=-0.5

-1→formato Q1.1→1.0

0.5→formato Q1.1→0.1

Como son de 2 bits cada uno, el resultado será de 4 bits, por lo tanto, se hace laextensión de signo, esto sería:

111.0
+
000.1

11.10→ este número es un formato Q2.2 y si observamos en la tabla 1-6 si corresponde a -0.5 que era el resultado que se esperaba.

Este resultado se obtuvo dándole al bit más significativo el valor negativo:

-2(1)+1(1)+0.5(1)=-0.5

Con esta operación se puede comprobar si los cálculos realizados fueron los correctos[2].

1.2.-DIGITALIZACIÓN Y CONVERSIÓN DE DIGITAL A ANALÓGICO.

1.2.1.-Digitalización.

La digitalización o conversión analógica-digital (conversión A/D) consiste básicamente en convertir una señal analógica en una señal digital, es decir, realizar de forma periódica medidas de la amplitud (tensión) de una señal (por ejemplo, la que proviene de un micrófono si se trata de registrar sonidos, de un sismógrafo si se trata de registrar vibraciones o de una sonda de un osciloscopio para cualquier nivel variable de tensión de interés), redondear sus valores a un conjunto finito de niveles preestablecidos de tensión (conocidos como niveles de cuantificación) y registrarlos como números enteros en cualquier tipo de memoria o soporte. La conversión A/D también es conocida por el acrónimo inglés ADC (analogue to digital converter) [5].

1.2.2.-Muestreo.

El muestreo consiste en la conversión de una señal continua en el tiempo en una señal discreta en el tiempo obtenida mediante la toma de "muestras" de la señal continua en el tiempo en instantes discretos de tiempo. Por tanto, si xa(t) es la entrada del muestreador, la salida será xa(nT) ≡ x(n), donde T es el intervalo de muestreo [6].

Uno de los métodos de muestreo y conversión a tiempo discreto más típicos consiste en realizar un muestreo periódico o uniforme, el cual se basa en la selección de muestras de la señal analógica en un intervalo de tiempo uniforme. Matemáticamente, el procedimiento consiste en sustituir la variable t = nT, donde T es el periodo de muestreo, obteniendo una secuencia de muestras [7]:

$$x(n) = x_a(nT) \qquad -\infty < n < \infty \quad [adimensional] \tag{1-5}$$

donde *x*(*n*) es la señal discreta en el tiempo obtenida "tomando muestras" de la señal analógica *xa*(*t*) cada *T* segundos. Este procedimiento se ilustra en la Figura 1-4 El intervalo de tiempo *T* entre muestras sucesivas es el período de muestreo o intervalo de muestreo y su recíproco 1/T =Fs se denomina tasa de muestreo (muestras por segundo) o frecuencia de muestreo (hercios).

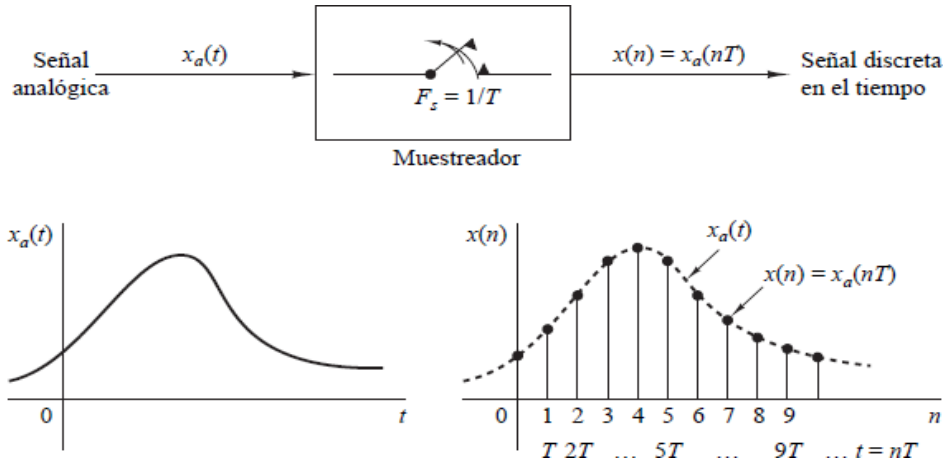


Figura 1-4: Muestreo periódico de una señal analógica.

Fuente: Proakis J y Manolakis D, "*Tratamiento Digital de Señales.*"

El muestreo periódico establece una relación entre las variables *t* y *n* de las señales continuas y discretas en el tiempo, respectivamente. Estas variables se relacionan linealmente a través del período de muestreo *T* o, de forma equivalente, a través de la tasa de muestreo *Fs* = 1/*T*, como [6]:

$$t = nT = \frac{n}{Fs} \qquad \left[\frac{n}{1/s} = s\right] \tag{1-6}$$

No se debe olvidar que la frecuencia de muestreo también se puede expresar en radianes por segundo, ωs = 2π/T. La diferencia más importante entre la señal analógica y la señal discreta es que la variable independiente de la secuencia es un índice entero donde se ha perdido la dimensión temporal, es decir, se ha producido una normalización temporal. Como veremos, esta relación entre la variable temporal analógica (t) y discreta (n) se transforma en una relación análoga entre el dominio de la frecuencia de tiempo continuo y de tiempo discreto.

En general, la operación de muestreo es no invertible porque se produce pérdida de información de la señal, ya que pueden existir varias señales analógicas diferentes que coincidan en los puntos de muestreo, generando la misma señal muestreada. Sin embargo, si imponemos alguna limitación a las señales analógicas, esta ambigüedad puede resolverse y el proceso pasa a ser invertible. Esta limitación viene dada por el teorema del muestreo [7].

Teorema de muestreo

Dada cualquier señal analógica, ¿cómo podemos seleccionar el período de muestreo *T* o, lo que es equivalente, la frecuencia de muestreo *Fs*? Para responder a esta pregunta, tenemos que disponer de alguna información sobre las características de la señal que se va a muestrear. En concreto, necesitamos conocer información general acerca del contenido en frecuencia de la señal. Normalmente, dicha información estará disponible.

Por ejemplo, se sabe que las principales componentes de frecuencia de una señal de voz se encuentran por debajo de los 3000 Hz. Por otro lado, las señales de televisión, generalmente, contienen componentes de frecuencia importantes hasta los 5 MHz. La información contenida en tales señales se encuentra en las amplitudes, frecuencias y fases de las diversas componentes de frecuencia, pero la información detallada de las características de dichas señales no estará disponible para nosotros antes de obtener las señales. De hecho, el propósito del procesamiento de señales es extraer dicha información detallada. Sin embargo, si se conoce la frecuencia máxima de la clase general de las señales (por ejemplo, la clase de las señales de vídeo, etc.), se podrá especificar la frecuencia de muestreo necesaria para convertir las señales analógicas en señales digitales. Supongamos que cualquier señal analógica puede representarse como una suma de sinusoides de diferentes amplitudes, frecuencias y fases, es decir [6]:

$$x_a(t) = \sum_{i=1}^{N} A_i \cos\left(2\pi F_i t + \theta_i\right) \quad [radianes \text{ o } grados] \qquad (1\text{-}7)$$

donde N indica el número de componentes de frecuencia.

Conocida Fmax, se puede seleccionar la apropiada frecuencia de muestreo. Se sabe que la frecuencia más alta de una señal analógica que puede reconstruirse sin ambigüedades cuando se muestrea la señal a una frecuencia Fs = 1/T es Fs/2. Cualquier frecuencia por encima de Fs/2 o por debajo de −Fs/2 produce muestras que son idénticas a las correspondientes frecuencias dentro del rango −Fs/2 ≤ F ≤ Fs/2. Para evitar las ambigüedades debidas al aliasing, tenemos que elegir una frecuencia de muestreo que sea suficientemente alta. Es decir, hay que seleccionar Fs/2 para que sea mayor que Fmax. Por tanto, para evitar el problema del aliasing, se selecciona Fs de modo que

$$F_s > 2F_{max} \qquad [H_z] \qquad (1\text{-}8)$$

donde Fmax es la componente de frecuencia más alta de la señal analógica. Seleccionando de este modo la frecuencia de muestreo, cualquier componente de

frecuencia, es decir, |Fi| < Fmax, de la señal analógica se corresponde con una sinusoide discreta en el tiempo con una frecuencia de:

$$-\frac{1}{2} \leq f_i = \frac{F_i}{F_s} \leq \frac{1}{2} \qquad \text{[adimensional]} \qquad (1\text{-}9)$$

$$-\pi \leq \omega_i = 2\pi f_i \leq \pi \qquad [radianes] \qquad (1\text{-}10)$$

Dado que | f | = ½ ó |ω | = π es la frecuencia más alta (única) de una señal discreta en el tiempo, elegir la frecuencia de muestreo de acuerdo con (ecuación 1-7) evita el problema del *aliasing*. En otras palabras, la condición $Fs > 2F\text{max}$ asegura que todas las componentes sinusoidales de la señal analógica se corresponden con las componentes discretas en el tiempo con frecuencias incluidas en el intervalo fundamental. Por tanto, todas las componentes de frecuencia de la señal analógica están representadas en la forma muestreada sin ambigüedades y, en consecuencia, la señal analógica puede reconstruirse sin distorsión a partir de los valores de las muestras, empleando el método "adecuado" de interpolación (conversión digital-analógica). La fórmula de interpolación "adecuada" o ideal se especifica mediante el teorema de muestreo.

Teorema de muestreo. Si la frecuencia más alta contenida en una señal analógica $xa(t)$ es $F\text{max} = B$ y la señal se muestrea a una frecuencia $Fs > 2F\text{max} \equiv 2B$, entonces $xa(t)$ puede recuperarse de forma exacta a partir de los valores de sus muestras utilizando la siguiente función de interpolación:

$$g(t) = \frac{sen(2\pi Bt)}{2\pi Bt} \qquad [adimensional] \qquad (1\text{-}11)$$

Luego, $x_a\,(t)$ puede expresarse como:

$$x_a = \sum_{n=-\infty}^{\infty} x_a\left(\frac{n}{F_s}\right) g\left(t - \frac{n}{F_s}\right) \qquad \text{[adimensional]} \qquad (1\text{-}12)$$

Donde $x_a\left(\frac{n}{F_s}\right) = x_a(nT) = x_a(n)$ son muestras de la $x_a(t)$.

Cuando el muestreo de xa(t) se realiza a la frecuencia mínima de muestreo Fs = 2B, la fórmula de reconstrucción (1.4.23) se convierte en:

$$x_a(t) = \sum_{n=-\infty}^{\infty} x_a\left(\frac{n}{2B_s}\right) \frac{sen2\pi B\left(t - \frac{n}{2B}\right)}{2\pi B\left(t - \frac{n}{2B}\right)} \qquad [adimensional] \qquad (1\text{-}13)$$

La frecuencia de muestreo FN = 2B = 2Fmax se denomina frecuencia de Nyquist. La Figura 1-5 ilustra el proceso de conversión D/A ideal utilizando la función de interpolación dada por (1-11) [6].

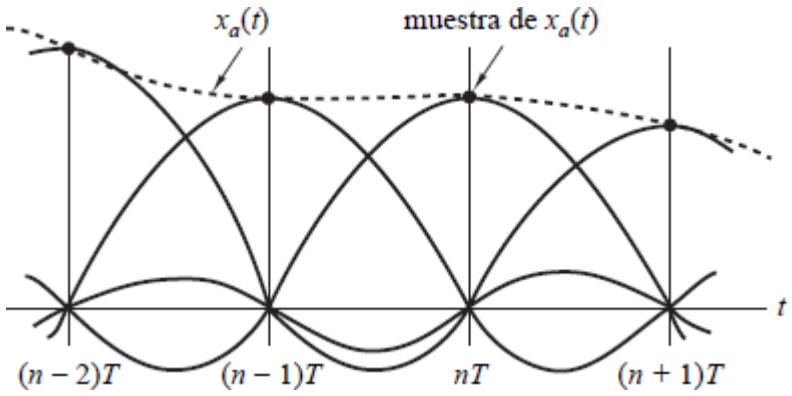


Figura 1-5: Conversión D/A ideal (interpolación)

Fuente: Proakis J y Manolakis D, "*Tratamiento Digital de Señales.*"

1.2.3.- Cuantificación.

Como se ha visto, una señal digital es una secuencia de números (muestras) en la que cada número está representado por un número finito de dígitos (precisión finita). El proceso de convertir una señal discreta en el tiempo con amplitud continua en una señal digital expresando cada valor de muestra como un número finito (en lugar de infinito) de dígitos se denomina **cuantificación**. El error introducido al representar la señal continua mediante un conjunto finito de niveles discretos es el error de cuantificación o ruido de cuantificación.

La operación de cuantificación de las muestras $x(n)$ se denota mediante $Q[x(n)]$ y se emplea $xq(n)$ para indicar la secuencia de las muestras cuantificadas a la salida del cuantificador. Por tanto,

$$xq(n) = Q[x(n)] \qquad \text{[adimensional]} \tag{1-14}$$

Luego el error de cuantificación es una secuencia $eq(n)$ definida como la diferencia entre el valor cuantificado y el valor de la muestra real. Por tanto, [6]:

$$eq(n) = xq(n) - x(n) \qquad \text{[adimensional]} \tag{1-15}$$

En la figura 1-6 se muestran los niveles, los intervalos, los valores de cuantificación y el escalón de cuantificación o resolución (Δ).

Figura 1-6: Niveles, intervalos y los valores de cuantificación.

Fuente: "Muestreo y recuperación se señales"

Como se puede observar, el espaciado de los escalones de cuantificación es uniforme, en este caso el cuantificador se denomina uniforme o lineal. Típicamente se usan cuantificadores uniformes para realizar un procesado digital de la señal, no obstante, para aplicaciones de almacenamiento y transmisión de señales puede no ser uniforme. La diferencia entre el máximo valor de la señal y el mínimo R = xL+1−x1 se denomina rango dinámico del cuantificador. Típicamente el cuantificador suele ser bipolar, es decir el valor máximo y el mínimo son opuestos uno del otro y la escala es simétrica respecto del 0, x1 = −xL+1. El funcionamiento de un cuantificador viene dado por su función característica, en la figura 1-7 se representa una función característica típica de un cuantificador uniforme. Como sepuede observar el valor del cuantificador viene dado por el redondeo del valor de la señal al valor de cuantificación más cercano. La cantidad de valores de cuantificación debe ser un número par (2b+1), si se requiere la codificación del 0 entonces el rango queda asimétrico [7].

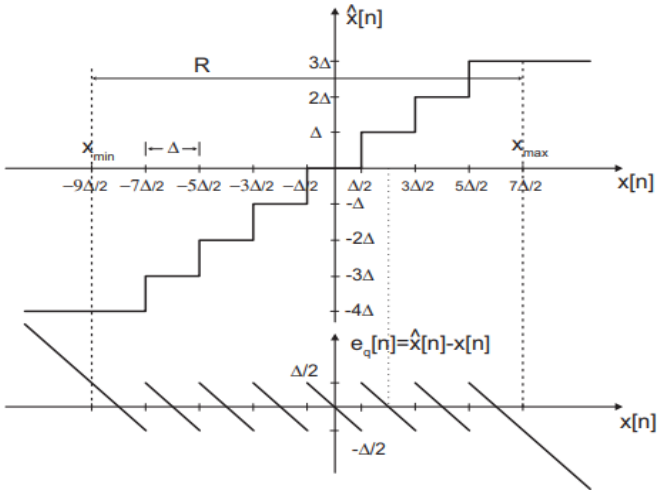


Figura 1-7: Función característica de un cuantificador uniforme.

En la figura 1-7 se muestran 8 valores de cuantificación, de los cuales uno es el cero, tres positivos y cuatro negativos. También puede observarse el error cometido por la cuantificación, el cual oscila entre: $-\Delta/2 < eq[n] \leq \Delta/2$ debido al redondeo que se produce en la señal [7].

Una vez cuantificados los valores, es necesario codificarlos y para ello, si la señal consta de L valores, es necesario una cantidad b+1 de bits tal que: L = 2b+1. Con esta consideración, el escalón de cuantificación o resolución se calcula como:

$$\Delta = \frac{R}{2^{b+1}} \qquad [adimensional] \tag{1-16}$$

donde R es el rango dinámico del cuantificador, entendiendo por rango dinámico como el intervalo que queremos discretizar. Merece la pena notar que cuando aumenta el número de bits, disminuye el escalón de cuantificación y por tanto disminuye el error cometido en la cuantificación [7].

También se debe mencionar que hay 2 tipos de cuantificadores donde la fórmula para calcular el escalón de la cuantificación o resolución cambia, estos son el cuantificador Midtread y el cuantificador Midrise.

Cuantificador Midtread

En este cuantificador el 0 es una amplitud, es decir, el 0 forma parte del conjunto y la resolución para este caso se calcula como:

$$\Delta = \frac{R}{2^b - 1} \ [adimensional] \tag{1-17}$$

Cuantificador Midrise

En este cuantificador una amplitud se ubica a $\frac{\Delta}{2}$, es decir, el 0 no forma parte del conjunto y la resolución para este caso se calcula como:

$$\Delta = \frac{R}{2^b} \ [adimensional] \tag{1-18}$$

La cuantificación uniforme normalmente es un requisito si la señal digital resultante va a ser procesada por un sistema digital. Sin embargo, en las aplicaciones de transmisión y almacenamiento de señales de voz, por ejemplo, frecuentemente se emplean cuantificadores variantes en el tiempo. Si se asigna el valor cero a un nivel de cuantificación, el cuantificador es de redondeo. Si se asigna cero a un nivel de decisión, se dice que el cuantificador es de truncamiento [6].

En el truncamiento siempre se elige el valor inmediato inferior, es decir se le asigna el valor que este debajo del valor de la muestra y para el redondeo se le asigna el valor más cercano a la muestra, ya sea el inferior o el superior, todo va a depender de donde cayó la muestra, algo que nos ayuda a decidir cuando es el caso de redondeo es lo que se conoce como los niveles de decisión, estos siempre van a la mitad del nivel de cuantificación y nos ayudan a decidir si se le asigna el valor superior o inferior a la muestra, cabe aclarar que si la muestra cae justamente en un nivel de decisión es indistinto a donde asignemos el valor de esa muestra.

Análisis de los errores de cuantificación

Para determinar los efectos de la cuantificación sobre el funcionamiento de un convertidor A/D, adoptamos un método estadístico. La dependencia del error de cuantificación de las características de la señal de entrada y la naturaleza no lineal del cuantificador hace el análisis determinista muy complejo, excepto en casos muy simples. En el método estadístico, suponemos que el error de cuantificación es aleatorio. Modelamos este error como ruido que se ha añadido a la señal original (no cuantificada). Si la señal de entrada analógica está dentro del rango del cuantificador, el error de cuantificación $eq(n)$ está limitado en módulo [es decir, $|eq(n)| < \Delta/2$], y el error resultante se denomina *ruido granular*. Cuando la entrada cae fuera del rango del cuantificador (recorte), $eq(n)$ no está limitado y da lugar al *ruido de sobrecarga*. Este tipo de ruido puede dar lugar a una severa distorsión de la señal. El único remedio consiste en aplicar un factor de escala a la señal de entrada de modo que su rango dinámico caiga dentro del rango del cuantificador. El siguiente análisis se basa en la suposición de que no existe ruido de sobrecarga.

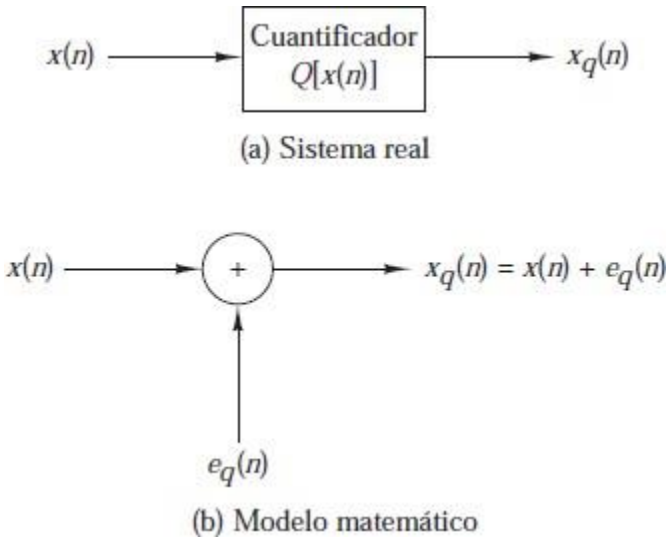


Figura 1-8: Modelo matemático del ruido de cuantificación

Fuente: Proakis J y Manolakis D, "*Tratamiento Digital de Señales.*"

El modelo matemático para el error de cuantificación $eq(n)$ se muestra en la Figura 1-8. Para realizar el análisis, hacemos las siguientes suposiciones acerca de las propiedades estadísticas de $eq(n)$:

1. El error eq(n) está distribuido uniformemente en el rango $-\Delta/2 < eq(n) < \Delta/2$.
2. El error {eq(n)} es un ruido blanco estacionario. En otras palabras, el error eq(n) y el error eq(m) para m _= n no están correlados.
3. El error {eq(n)} no está correlado con la señal x(n).
4. La señal x(n) tiene media cero y es estacionaria.

En general, estas suposiciones no se cumplen. Sin embargo, se cumplen cuando el

tamaño del escalón de cuantificación es pequeño y la señal x(n) atraviesa varios niveles de cuantificación entre dos muestras sucesivas. Bajo estas suposiciones, el efecto del ruido aditivo eq(n) en la señal deseada puede cuantificarse evaluando la relación señal-ruido de cuantificación (SQNR, signal-to-quantization-noise), que se puede expresar en una escala logarítmica (en decibelios, dB) como [6]:

$$SQNR = 10log_{10}\left(\frac{P_x}{P_n}\right)\ [adimensional] \tag{1-19}$$

donde $Px = \sigma 2x = E[x2(n)]$ es la potencia de la señal y $Pn = \sigma 2e = E[e2q(n)]$ es la potencia del ruido de cuantificación.

Si el error de cuantificación está distribuido uniformemente en el rango $(-\Delta/2, \Delta/2)$, como se muestra en la Figura 1-9, el valor medio del error es cero y la varianza (la potencia del ruido de cuantificación) es:

$$P_n = \sigma^2 e = \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} e^2 p(e)de = \frac{1}{\Delta}\int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} e^2 de = \frac{\Delta^2}{12}\ [adimensional] \tag{1-20}$$

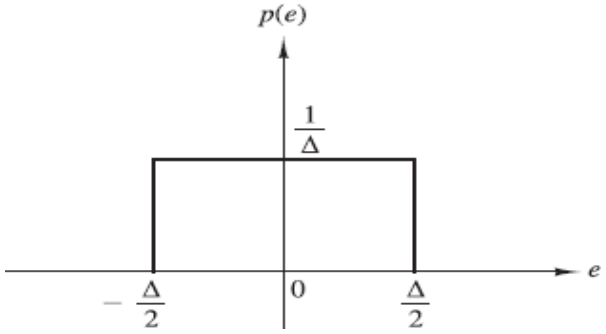


Figura 1-9: Función de densidad de probabilidad del error de cuantificación.

Combinando 1-16 con 1-20 y sustituyendo el resultado en 1-19, la expresión para la relación SQNR se convierte en:

$$SQNR = 10log\left(\frac{P_x}{P_n}\right) = 20log\left(\frac{\sigma_x}{\sigma_e}\right)\ [adimensional] \tag{1-21}$$

La fórmula dada en (1-21) se utiliza frecuentemente para especificar la precisión necesaria en un convertidor A/D. Quiere decir simplemente que cada bit adicional utilizado en el cuantificador aumenta la relación señal ruido de cuantificación en 6 dB [6].

1.2.4.-Codificación.

Existen diferentes representaciones para codificar los números, cada una de las cuales tiene sus ventajas e inconvenientes Típicamente, esta suele ser la última etapa de un convertidor ADC en la cual se realiza la codificación binaria de la señal.

Todos los números se pueden representar mediante una descomposiciónpolinómica en función de una determinada base, la más típico es la base decimal, aunque se puede utilizar cualquier otra. En particular, en sistemas digitales se utiliza una representación binaria, es decir, con base 2. Sea un número con parte entera y parte fraccional dado por:

$$X = (\, x_{-a} \ldots . . x_{-1}\, x_0, x_1 \ldots x_b)_r = \sum_{i=-a}^{b} x_i\, r^{-i}, 0 \le x_i \le r-1\, [adim.] \quad (1\text{-}22)$$

donde xi representa los dígitos o bits, r es la base, a es el número de dígitos enteros y b es el número de dígitos fraccionales. En el caso binario, los dígitos únicamente pueden tomar valores {0, 1}. El dígito situado más a la izquierda, x−a se denomina el bit más significativo (MSB, most significative bit) y el dígito situado más a la derecha, xb es el bit menos significativo (LSB, least significative bit). El punto o coma entre los bits x0 y x1 no existe físicamente en los computadores, para su gestión se diseñan los circuitos lógicos para asumir el punto o coma en la posición que se haya prefijado. Un número de bits disponible para representar un número hace que solo se puedan representar una cantidad de números finita. El proceso de cuantificar convierte las amplitudes de la señal en otras aproximadas que pueden ser representadas con un número determinado de bits. Todas las operaciones implicadas en el procesado digital también están sujetas a los efectos provocados por el hecho de tener un número finito de bits (longitud finita de la palabra o efectos de precisión finita), cualquier operación con números digitales puede generar un resultado que no se pueda representar con el número de bits disponible y en ese caso es necesario volver a realizar una cuantificación mediante redondeo o truncamiento del valor del número. Este procedimiento se le denomina aritmética de precisión finita y genera errores cuyo valor depende del número de bits utilizado.

Estos errores se propagan a lo largo de los procedimientos implicados en un algoritmo y pueden generar resultados erróneos, incluso divergencias e inestabilidades del propio algoritmo [7].

1.2.5.-Interpolación.

Un problema común en la teoría de aproximación es la reconstrucción de una función a partir de un conjunto de valores discretos de datos que dan información sobre la función misma. Esta información por lo general viene dada como valores puntuales o medias en celda de la función sobre un conjunto finito de puntos o celdas, respectivamente. La función entonces es aproximada por un interpolante, es decir, por otra función cuyos valores puntuales o medias en celda coinciden con los de la función original.

Este interpolante puede ser construido por interpolación lineal.

La interpolación es un procedimiento lineal sobre los valores de la función en un conjunto de puntos. En este caso la exactitud de la aproximación cerca de una singularidad está limitada y depende del orden de la singularidad, de modo que si construimos el polinomio interpolador basándonos en un conjunto de puntos (stencil) que cruzan la singularidad obtendremos una aproximación insatisfactoria. Esto significa que el aumento del grado del polinomio producirá regiones más grandes de mala aproximación alrededor de las singularidades.

Así pues, para aumentar la exactitud la solución no es aumentar el número de puntos que se utiliza en la interpolación, sino escoger los puntos de forma que el stencil quede dentro de la parte suave de la función, siempre que esto sea posible.

Consideremos la interpolación. El aumento de la frecuencia de muestreo por interpolación es un poco más complicado que el diezmado porque con la interpolación, es necesario calcular nuevos valores de muestra. Conceptualmente, la interpolación comprende la generación de una curva continua $y_c(t)$ que pasa a través de nuestro $X_{old}(n)$ valores muestreados, como se muestra en la Figura 1-10 (a), seguida de un muestreo de esa curva en la nueva frecuencia de muestreo f_{new} para obtener la secuencia interpolada. $X_{new}(n)$ en la Figura 1-10 (b). Por supuesto, las curvas continuas no pueden existir dentro de una máquina digital, por lo que solo se tendrá que obtener $X_{new}(n)$ directamente de $X_{old}(n)$. Para aumentar una tasa de muestra dada, o muestra superior, en un factor de M, se tiene que calcular valores intermedios M-1 entre cada muestra en $X_{old}(n)$.

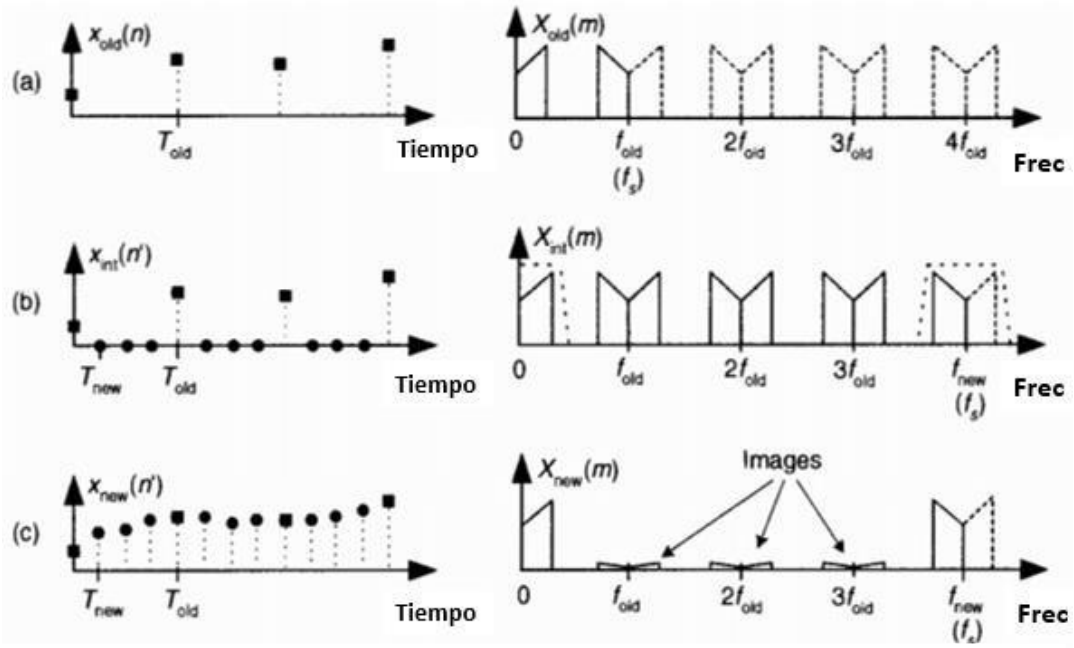


Figura 1-10: Interpolación por un factor de cuatro: (a) secuencia muestreada original y su espectro; (b) ceros insertados en la secuencia original y espectro resultante: (c) secuencia de salida del filtro de interpolación y espectro interpolado final.

Fuente: "*Muestreo y recuperación de las señales.*"

El subíndice 'int' significa intermedio. Observe $X_{int}(n') = X_{old}(n)$, cuando n '= 4n. Es decir, la secuencia anterior ahora está incrustada en la secuencia nueva. La inserción de ceros (un proceso llamado relleno de ceros) establece el índice de muestra para la nueva secuencia $X_{int}(n')$ se asignarán los valores interpolados.

El espectro de $X_{int}(n')$, $X_{int}(m)$, se muestra en la Figura 1-10(b) donde $f_{new}=4f_{old}$, las curvas sólidas en $X_{int}(m)$, centradas en múltiplos de f_{old}, llamadas imágenes. Loque se hizo al sumar los ceros es simplemente aumentar la frecuencia de muestreo efectiva a $f_s=f_{new}$ en la Figura 1-10(b). El último paso en la interpolación es filtrar la secuencia $X_{int}(n')$ con un filtro digital de paso bajo, cuya respuesta de frecuencia se muestra como líneas discontinuas alrededor de cero Hz y f_{new} Hz, en la Figura 1-10(b), para atenuar las imágenes espectrales. Este filtro de paso bajo se llama filtro de interpolación, y su secuencia de salida es la deseada $X_{new}(n')$ en la Figura 1-10(c) que tiene el espectro $X_{new}(m)$. El proceso no es muy complicado y se comprende mejor a modo de ejemplo.

Supongamos que se tiene la secuencia $X_{old}(n)$, parte de la cual se muestra en la Figura 1-11(a), y queremos aumentar su frecuencia de muestreo en un factor de M

= 4. El espectro de la secuencia $X_{old}(n)$ se proporciona en Figura 1-11(a), donde se muestra el espectro de señal entre cero Hz y $4f_{old}$. Tenga en cuenta que las curvas punteadas en $X_{old}(m)$ son réplicas espectrales. Para aumentar la muestra $X_{old}(n)$ por un factor de cuatro, normalmente insertamos tres ceros entre cada muestra, como se muestra en la figura 1-11 (b) para crear la nueva secuencia $X_{int}(n')$.

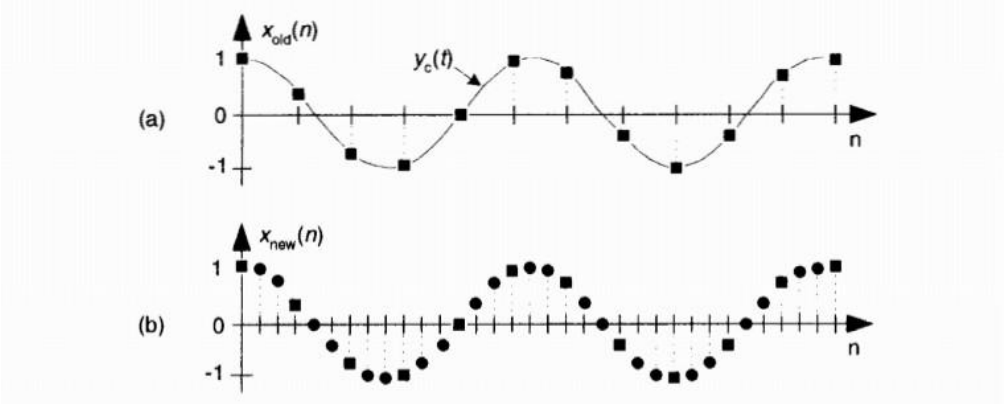


Figura 1-11: Conversión de la frecuencia de muestreo: (a) secuencia original; (b) interpolado por tres secuencias.
Fuente: "*Muestreo y recuperación de las señales.*"

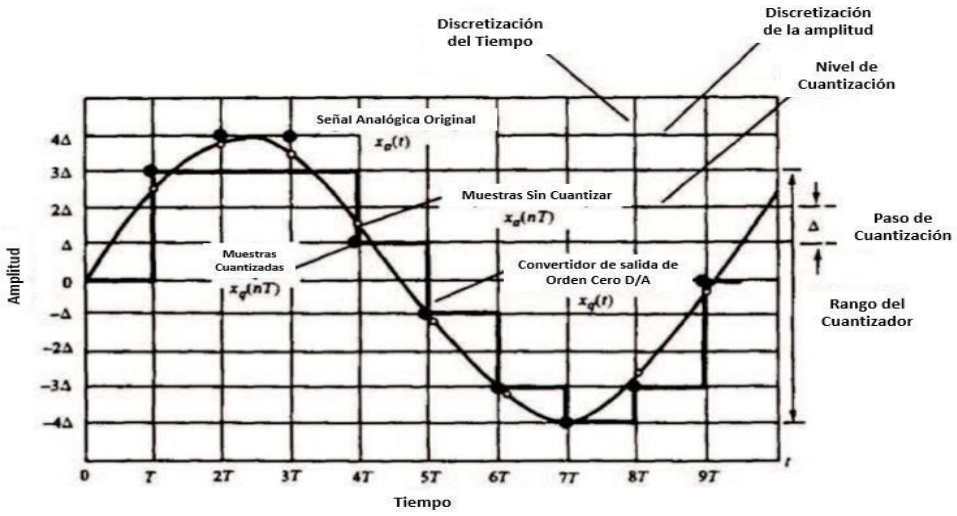
¿Eso es todo lo que hay que hacer con relleno, interpolación y filtrado a cero? Bueno, no del todo porque no se puede implementar un filtro de paso bajo ideal, $X_{new}(n')$ no será una interpolación exacta de $X_{old}(n)$. El error se manifiesta como las imágenes residuales dentro de $X_{new}(m)$. Con un filtro ideal, estas imágenes no existirían. Solo se puede aproximar a un filtro de interpolación de paso bajo ideal. El problema para recordar es que la precisión de todo el proceso de interpolación depende de la atenuación de la banda de estímulo de nuestro filtro de interpolación de paso bajo. Cuanto menor sea la tenencia, más precisa será la interpolación. Al igual que con el diezmado, la interpolación se puede considerar como un ejercicio de diseño de filtros de paso bajo. Se debe de tener en cuenta que el proceso de interpolación, debido a las muestras con valor cero, tiene un factor de pérdida de amplitud inherente de M. Por lo tanto, para lograr la ganancia unitaria entre las secuencias $X_{old}(n)$ y $X_{new}(n')$, el filtro de interpolación debe tener una ganancia de M.

Un último problema con respecto a la interpolación. Se puede caer en la trampa de pensar que la interpolación nació de las actividades modernas de procesamiento de señales (por ejemplo, cuando interpolamos / muestreamos una señal musical antes de aplicarla a un convertidor de digital a analógico (D / A) para enrutarlo a amplificador y altavoz en reproductores de discos compactos. Ese aumento de muestreo reduce el costo del filtro analógico que sigue al convertidor (D / A). Por favor no lo hagas. Antiguas tablillas astronómicas cuneiformes (originadas en Uruk y Babilonia 200 años antes del nacimiento de Jesús) indican que se utilizó la interpolación lineal para rellenar las posiciones tabuladas faltantes de los cuerpos celestes en aquellos momentos en que las condiciones atmosféricas impedían la observación directa [8]. Se ha utilizado la interpolación desde entonces, para completar los datos faltantes.

1.2.6.-Interpolación de orden cero.

La aproximación de interpolación de orden cero (Zero Order Hold) consiste en mantener el valor de la muestra hasta la llegada de la siguiente muestra, de manera que queda una aproximación de la señal en escalera. La aproximación de orden uno consiste en unir las muestras con una línea recta, y así sucesivamente.

Un ejemplo de señal sinusoidal de tiempo continuo que ha sido muestreada y cuantificada se muestra en la figura 1-12. Las líneas verticales muestran los instantes de muestreo de la señal. Las líneas verticales situadas dentro del rango del cuantificador (±4Δ) indican los niveles permitidos de cuantificación de la señal. La señal cuantificada viene dada por los puntos negros xq(nT) estos puntos se pueden interpolar por un interpolador de orden cero generando la señal en escalera que viene dada por la señal xq(t).



Fuente: "*Muestreo y recuperación de las señales.*"

Figura 1-12: Señal de tiempo continúo muestreada y cuantizada.

Una vez finalizado el procesado de señal analógica a digital, suele ser necesario una conversión de la señal digital al dominio analógico mediante un convertidor Digital Analógico (DAC). El diagrama de bloques de DAC viene representado en la figura 1- 13.

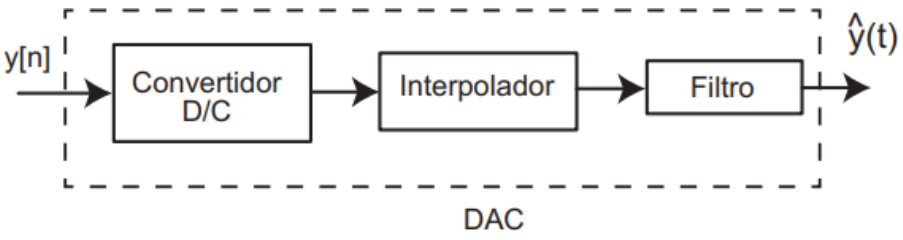


Figura 1-13: Convertidor DAC, bloques básicos.
Fuente: "Problemas de Tratamiento Digital de Señales"

Este sistema realiza una conversión de una señal digital a una señal de tiempo continuo (D/C), posteriormente es necesario realizar una interpolación de la señal para aproximar el valor ésta en los instantes de tiempo entre muestras consecutivas. Teóricamente existe un proceso de interpolación óptima, también denominada interpolación ideal que permite recuperar la señal analógica original, de la cual proviene la secuencia de entrada al conversor. Este proceso de recuperación se produce de forma exacta, siempre que la señal esté limitada en banda y se haya muestreado con una frecuencia de muestreo suficientemente alta. En la práctica este método resulta excesivamente complejo y se suelen utilizar otros métodos de interpolación: interpolación de orden cero, lineal, cuadrática, entre otras. Estas técnicas suelen generar ciertas distorsiones en la señal y normalmente requieren de una última etapa de filtrado [14].

1.3.-CONVERSIÓN DE DIGITAL A ANALÓGICO.

1.3.1.-Conversores digitales-analógicos.

Un convertidor digital a analógico (DAC) es un dispositivo que convierte los datos digitales en una señal analógica que es una tensión, corriente o carga eléctrica. Los datos digitales generalmente son una secuencia de impulsos de tiempo finitos que se procesan y convierten en una señal analógica física continua. La resolución, tasa de muestra y linealidad son los parámetros de rendimiento clave que describen la calidad del DAC. La resolución se refiere al número de bits digitales por muestra que se pueden convertir con precisión en una señal analógica. La tasa de muestra es la frecuencia en la que se prueban los datos digitales de entrada. La linealidad, que está relacionada con la resolución, describe qué tan uniformemente el DAC responde a los cambios de entrada digitales incrementales. La no linealidad diferencial (DNL) y no linealidad integral (INL) se utilizan normalmente para caracterizar la linealidad del DAC.

Existen muchos tipos diferentes de DAC y la escala de resistores es la más simple. La red de resistores forma un promedio ponderado de todos los bits de entrada digital; el bit más significativo (MSB) de la palabra de entrada digital recibe la ponderación más alta, mientras que el bit menos significativo (LSB) recibe la más baja. Un DAC sigma-delta de primer orden utiliza solo la lógica digital para producir una salida de un solo bit sobre muestreada que se puede filtrar a paso bajo para revelar la forma de onda analógica deseada. Los DAC requieren un filtro de reconstrucción para filtrar los términos de aliasing causados por los pasos constantes rectangulares de función definida a trozo que ocurren en la tasa de muestreo. Un filtro de paso bajo facilita estos pasos y atenúa los términos de aliasing.

Muchos DAC incorporan una funcionalidad específica de la aplicación y pueden ser un proveedor específico según el diseño. Un ejemplo de ello son los códecs G.711, que incorporan la conversión u-law: un método de codificación de datos que sacrifica la linealidad por el rango dinámico. Estos expanden los datos de ancho de banda de audio de 8 bits ejecutados en circuitos telefónicos digitales en una tensión con el rango dinámico original. Otro ejemplo es un TxDAC. Estos pasan datos digitales a través de filtros FIR de interpolación integrados para aumentar la tasa de muestra a fin de reducir los requisitos del filtro anti-aliasing analógico. También incorporan canales I/Q de cuadratura intercalada y la corriente de salida en lugar de tensión [12].

1.4.-MAC.

1.4.1.-¿Que es una MAC?

Las señales digitales están en alta demanda en el mundo actual debido al crecimiento reciente y rápido en multimedia y comunicación de sistemas. La unidad MAC es un componente esencial para la generación de estas señales digitales en tiempo real. Se utilizan para aumentar la productividad y el rendimiento de sistemas DSP y tienen numerosas aplicaciones como filtrado, convolución y productos internos. Una unidad MAC realiza ambas funciones de multiplicar y sumar. Estas funciones están en dos etapas. En la primera etapa, la multiplicación de dos números se calcula y en la siguiente etapa, el resultado de la primera etapa es utilizado para sumar / acumular. Si tanto la multiplicación como suma se ejecuta en un solo redondeo, entonces se dice que es una unidad fusionada de acumulación múltiple (MAC).

En la comunicación digital, el procesador de señales digitales (DSP) es un bloque importante que realiza varias aplicaciones de procesamiento de señales digitales, como la convolución, la transformada de coseno discreta (DCT), la transformada de Fourier, etc. Cada procesador de señal digital contiene una unidad MAC. La unidad MAC facilita el cálculo de la convolución que se necesita en los filtros, realiza procesos de multiplicación y acumulación repetidamente para realizar operaciones continuas y complejas en el procesamiento de señales digitales. La unidad MAC contiene un reloj y reinicio para controlar su funcionamiento [9].

1.4.2.- Teoría de MAC (Acumulador multiplicador).

La tecnología avanza rápidamente y, por lo tanto, el desarrollo y la mejora de la electrónica se ha vuelto obligatoria. La unidad MAC es una parte integral de un circuito DSP. Muchos papeles tienen usos diferentes, combinaciones de multiplicadores y sumadores en orden para crear una unidad MAC que tenga baja potencia y alta velocidad.

La mayoría de las señales digitales son producidas por el proceso repetido de multiplicación y suma, que controla la velocidad de todo el procedimiento, mientras que la acumulación de la multiplicación y la suma controlan el rendimiento y velocidad del proceso de cálculo. La ventaja de usar la unidad MAC es que ayudan

en la reducción del consumo de energía y retrasos que a su vez reducen el área del sistema en general.

La unidad MAC básica contiene multiplicador, sumador y acumulador. El multiplicador multiplica las entradas y da el resultado al sumador, que suma el resultado del multiplicador al anterior resultado acumulado. El diagrama de bloques de la unidad MAC básica está ilustrado en la figura 1-14:

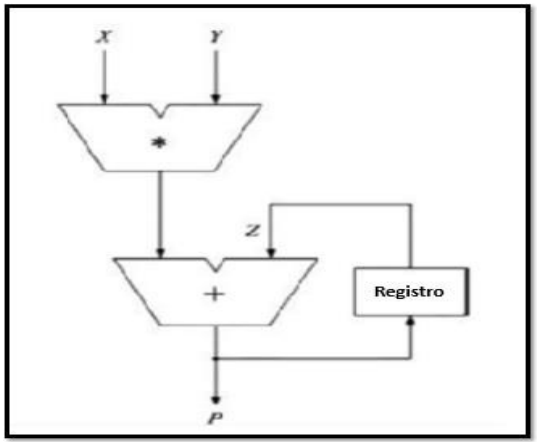


Figura 1-14: Diagrama de Bloques de una unidad básica MAC

Fuente: Rakesh S. and Grace K. S. V. "A survey on the design and performance of various MAC unit architectures"

Se dan dos entradas al multiplicador, el resultado final del multiplicador se le da al sumador, y la segunda entrada del sumador proviene del acumulador. El resultado final es de nuevo almacenado en el acumulador.

Básicamente, una unidad MAC utiliza un multiplicador rápido instalado en la forma de información y el aumento de la producción del multiplicador se nutre en un sumador rápido que se pone a cero al principio. La consecuencia de la expansión se guarda en un acumulador (ver figura 1-15).

Se han realizado muchas investigaciones comparativas para potencia, velocidad, retraso y área de diferentes multiplicadores como Árbol de Wallace, Array, Booth, Vedic y sumadores como la mitad sumador, sumador completo, sumador detransporte ondulado, sumador de avance están diseñados y comparados. El multiplicador juega un papel importante, en muchas aplicaciones de procesamiento de señales digitales (DSP) en convolución, filtros digitales y otro procesamiento de datos unidad [10].

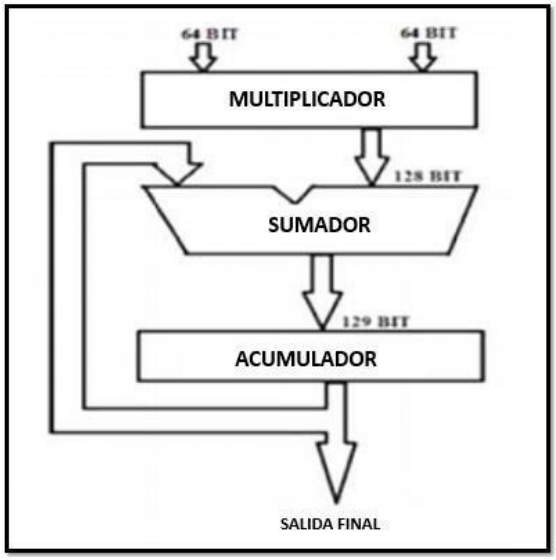


Figura 1-15: Arquitectura de una unidad básica MAC

Fuente: Rakesh S. and Grace K. S. V. "A survey on the design and performance of various

CAPÍTULO II: **MÓDULO MULTIPLICADOR -ACUMULADOR COMPLEJO (MAC COMPLEJA).**

En este capítulo se habla de las bases teóricas que son necesarias para poder desarrollar una MAC, se tratan tópicos de suma importancia para comprender mejor toda la teoría y las bases necesarias para cumplir con el objetivo, el cual es la implementación de una MAC en un microprocesador, los temas que se analizan son: la convolución, el filtrado digital, la correlación, la cual es un caso particular de la convolución, la transformada y serie de Fourier, así como la comparación entre una MAC y una MAC compleja, de todos estos temas antes mencionados se expone su definición, para que se usan, sus principales propiedades y características así como un ejemplo de cada uno de ellos, también se presenta el algoritmo CORDIC, del cual se expondrá su funcionamiento y ecuaciones necesarias para poder desarrollarlo.

2.1.- CONVOLUCIÓN: FILTRADO DIGITAL.

2.1.1 Filtro digital

Un filtro digital, es un filtro que opera sobre señales digitales. Es una operación matemática que toma una secuencia de números (la señal de entrada) y la modifica produciendo otra secuencia de números (la señal de salida) con el objetivo de resaltar o atenuar ciertas características. Puede existir como una fórmula en un papel, un loop en un programa de computadora, como un circuito integrado en un chip. [15]

2.1.2 Caracterización de un filtro

Hay tres formas equivalentes de caracterizar un filtro (ver figura 2-1):

- Respuesta al impulso
- Respuesta en frecuencia
- Respuesta al escalón

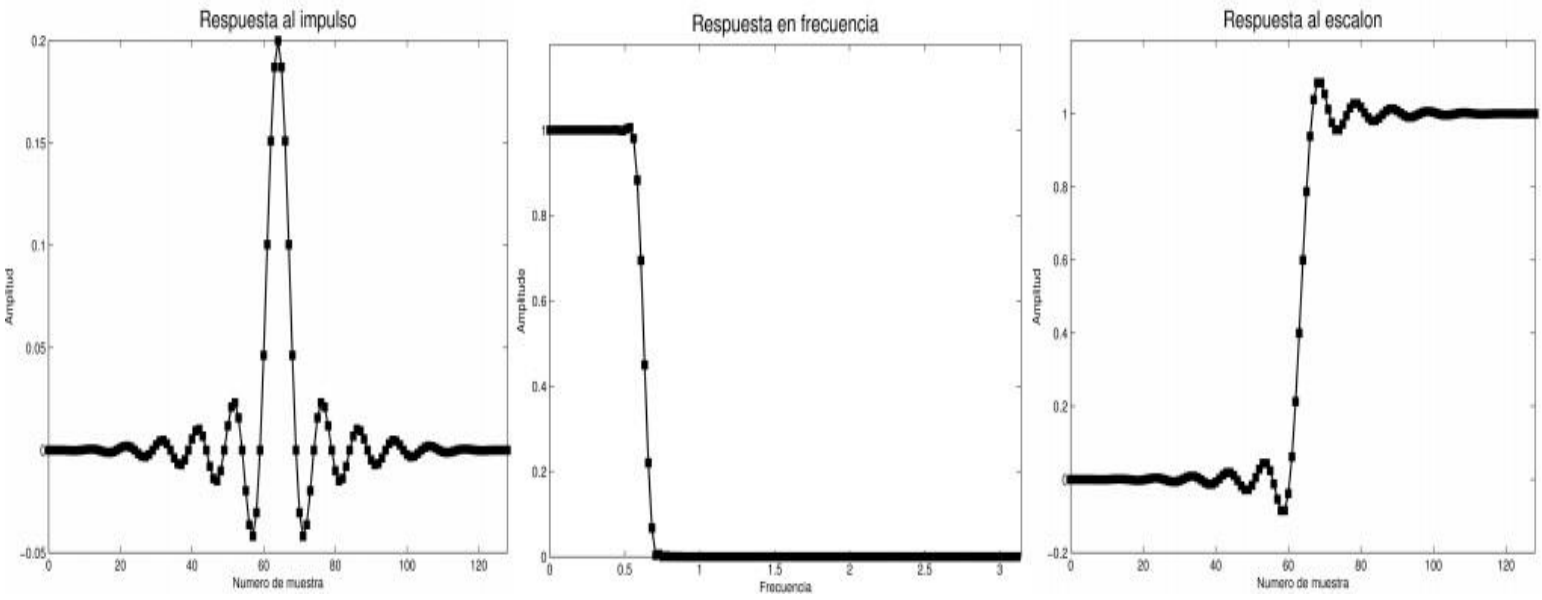


Figura 2-1: formas equivalentes de caracterizar un filtro

Fuente: "Introducción a los Filtros Digitales"

Caracterización de un filtro

Respuesta al impulso

Conociendo la respuesta al impulso, se puede calcular la respuesta del filtro a cualquier entrada (principio de superposición, ver figura 2-2) [15].

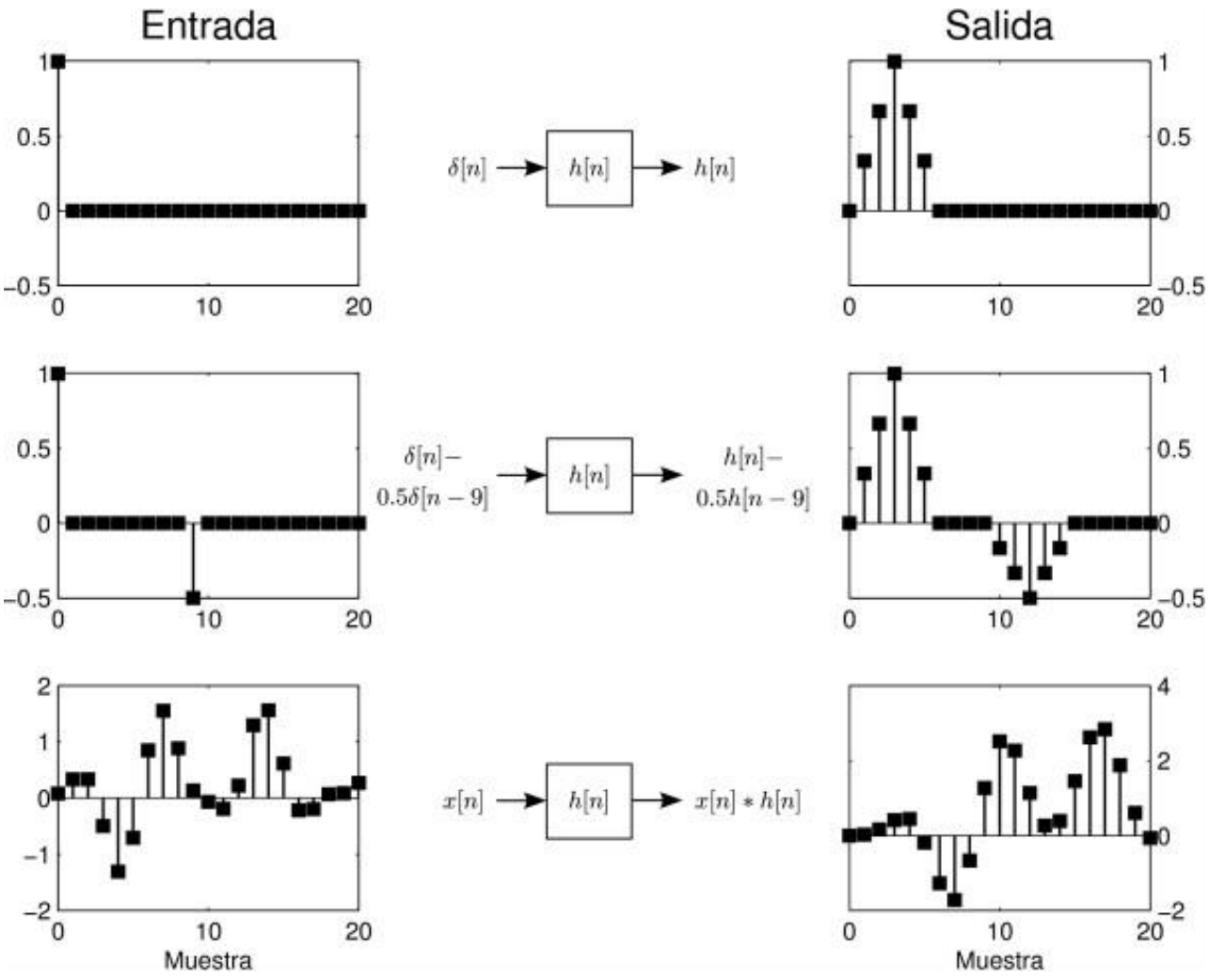


Figura 2-2: principio de superposición

Fuente: "Introducción a los Filtros Digitales"

Implementación de un filtro

Convolución

Convolución de la señal de entrada con la respuesta al impulso del filtro. En este caso, la salida del filtro en cada instante es un promedio ponderado de la muestra actual y muestras pasadas de la entrada (ver figura 2-3) [15].

$$y[n] = (x * h)[n] = \sum_k x[k]h[n-k]$$

Figura 2-3: Respuesta al impulso finita (FIR)

Fuente: "Introducción a los Filtros Digitales"

2.1.3 Función delta de Dirac o función impulso unitario.

Esta función es una abstracción matemática creada por el físico inglés Paul Dirac. Se aplica en muchas ramas de la ciencia en las cuales se describen procesos mediante modelación matemática. Esta función es el límite cuando T tiende a cero del pulso rectangular mostrado en la figura 2-4 [16]:

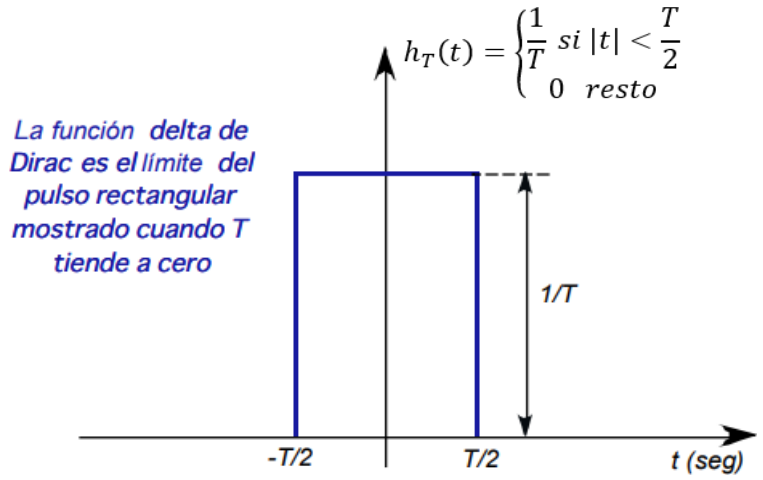


Figura 2-4: Función delta de Dirac o función impulso unitario

Fuente: Olvera J. "Convolución: Un proceso natural en los sistemas lineales e invariantes en el tiempo"

Cuando T tiende a cero se tiene un pulso rectangular con las siguientes características:

• Su duración tiende a cero.

• Su amplitud tiende a infinito.

• Su área permanece constante y es igual a 1.

La representación matemática para esta función es δ (t) y se dibuja como una flecha vertical donde su altura es proporcional al área bajo el impulso tal como se muestra en la figura 2-5 [16].

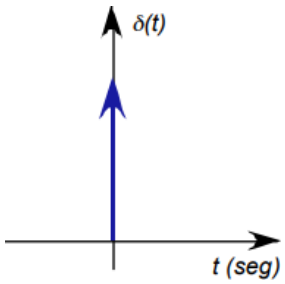


Figura 2-5: Representación de la función delta de Dirac o función impulso unitario

Fuente: Olvera J. "Convolución: Un proceso natural en los sistemas lineales e invariantes en el tiempo"

Respuesta al impulso. Suponga que un sistema lineal continuo e invariante en el tiempo no tiene energía almacenada internamente y se le aplica la excitación x(t)=δ(t), es decir, se la aplica como excitación un impulso unitario en tiempo cero con condiciones iniciales cero para el sistema. La respuesta del sistema para este caso se conoce como respuesta al impulso y se denominará h(t). A continuación, se muestran varias excitaciones y sus correspondientes respuestas, sabiendo que el sistema es lineal e invariante en el tiempo (ver figura 2-6) [16]:

Excitación	Respuesta
$\delta(t)$	$h(t)$
$a\,\delta(t)$	$a\,h(t)$
$\delta(t-t_0)$	$h(t-t_0)$
$a\,\delta(t-t_0)$	$a\,h(t-t_0)$
$a\,\delta(t-t_1)+b\,\delta(t-t_2)$	$a\,h(t-t_1)+b\,h(t-t_2)$
$x(t)$	¿?

Figura 2-6: Respuesta al impulso unitario

2.1.4 Técnicas para el análisis de los sistemas lineales

Existen dos métodos básicos que permiten analizar el comportamiento o respuesta de un sistema lineal a una determinada señal de entrada. Un método está basado en la resolución directa de la ecuación de entrada–salida del sistema, que, en general, tiene la forma:

$$y(n) = F[y(n-1), y(n-2), \dots, y(n-N), x(n), x(n-1), \dots, x(n-M)]\ [adimen.]$$

donde F[·] indica alguna función de las magnitudes encerradas entre corchetes. Específicamente, para un sistema LTI, veremos más adelante que la forma general de la relación entrada–salida es [6]:

$$y(n) = -\sum_{k=0}^{N} a_k y(n-k) + \sum_{k=0}^{N} b_k x(n-k) \qquad (2\text{-}1)$$

Donde {ak} y {bk} son parámetros constantes específicos del sistema y son independientes de *x*(*n*) e *y*(*n*). La relación de entrada–salida dada por (2-1) es una ecuación en diferencias y representa una forma de caracterizar el comportamiento del sistema LTI discreto en el tiempo.

El segundo método para analizar el comportamiento de un sistema lineal ante una señal de entrada determinada consiste en descomponer primero la señal de entrada en una suma de señales elementales. Estas señales elementales se seleccionan de manera que la respuesta del sistema a cada componente de señal se determine fácilmente. A continuación, utilizando la propiedad de linealidad del sistema, las respuestas de este a las señales elementales se suman para obtener la respuesta total del sistema a la señal de entrada dada. Este segundo método es el que hemos descrito en esta sección. Supongamos que la señal de entrada x(n) se descompone en una suma ponderada de componentes elementales de la señal {xk(n)}, de modo que [6]:

$$x(n) = \sum_{k} C_k\, x_k(n)\ [adimensional] \qquad (2\text{-}2)$$

donde los {ck} hacen referencia al conjunto de amplitudes (coeficientes de ponderación) de la descomposición de la señal x(n). Ahora suponemos que la respuesta del sistema a la señal elemental xk(n) es yk(n). Por tanto,

$$y_k(n) = \mathcal{F}[x_k(n)] \quad [adimensional] \tag{2-3}$$

Suponiendo que el sistema está en reposo y que la respuesta a ckxk(n) es ckyk(n), como consecuencia de la propiedad de escalado del sistema lineal. Por último, la respuesta total a la entrada x(n) es [6]:

$$y(n) = \mathcal{F}[x(n)] = \mathcal{F}\left[\sum_k C_k\, x_k(n)\right] = \sum_k C_k\, \mathcal{F}[x_k(n)] \tag{2-4}$$

$$= \sum_k C_k\, y_k(n) \quad [adimensional]$$

En (2-4) hemos utilizado la propiedad aditiva del sistema lineal. Aunque en principio parece que la elección de las señales elementales es arbitraria, en realidad dicha selección es extremadamente dependiente de la clase de señales de entrada que deseemos considerar. Si no imponemos ninguna restricción a las características de las señales de entrada, entonces su descomposición en una suma ponderada de secuencias de impulsos unitarios es matemáticamente conveniente y completamente general. Por el contrario, si nos restringimos a una subclase de señales de entrada, puede existir otro conjunto de señales elementales que sea más adecuado matemáticamente para la determinación de la salida [6].

2.1.5 Descomposición en impulsos de una señal discreta en el tiempo

Suponga que disponemos de una señal arbitraria x(n) que deseamos descomponer en una suma de impulsos unitarios. Para aplicar la notación establecida en la sección anterior, seleccionamos las señales elementales xk(n) de modo que:

$$x_k(n) = \delta(n-k) \quad [adimensional] \tag{2-5}$$

donde *k* representa el retardo de la secuencia de impulsos. Para poder manejar una señal arbitraria *x*(*n*) que tenga valores distintos de cero de duración infinita, el conjunto de impulsos unitarios tiene que ser infinito, para contener el número infinito de retardos. Ahora supongamos que multiplicamos las dos secuencias *x*(*n*) y δ (*n*−*k*). Puesto que δ (*n*−*k*) es cero siempre, excepto para *n* = *k*, que es igual a la unidad, el resultado de esta multiplicación es otra secuencia que es igual a cero siempre excepto en *n* = *k*, cuyo valor es *x*(*k*) [6].

$$X(n)\delta(n-k) = X(k)\delta(n-k)\ [adimensional] \tag{2-6}$$

es una secuencia que es cero siempre excepto en *n* = *k*, que es igual a *x*(*k*). Si repetimos la multiplicación de *x*(*n*) por δ (*n*−*m*), siendo *m* otro retardo (*m* _= *k*), el resultado sería una secuencia que es cero para todos los puntos excepto en *n* = *m*, donde toma el valor *x*(*m*). Luego

$$x(n)\delta(n-m) = x(k)\delta(n-k) \quad [adimensional] \tag{2-7}$$

En otras palabras, cada multiplicación de la señal *x*(*n*) por un impulso unitario desplazado un cierto *k*, [es decir, δ (*n*−*k*)], extrae el valor *x*(*k*) de la señal *x*(*n*) en elinstante en que el impulso unitario es distinto de cero. Por tanto, si repetimos esta multiplicación para todos los posibles desplazamientos, −∞ < *k* < ∞, y sumamos todos los productos, el resultado será una secuencia igual a *x*(*n*), es decir,

$$x(n) = \sum_{k=-\infty}^{\infty} X(k)\delta(n-k) \quad [adimensional] \tag{2-8}$$

Vamos a fijarnos en el lado derecho de la Ecuación (2-8), que es el sumatorio de un número infinito de impulsos unitarios desplazados, donde el impulso unitario δ (*n*−*k*) tiene una amplitud *x*(*k*). Por tanto, el lado derecho de dicha ecuación proporciona la descomposición de cualquier señal arbitraria *x*(*n*) en una suma ponderada (escalada) de impulsos unitarios desplazados.

2.1.6 Respuesta de los sistemas LTI a entradas arbitrarias: la convolución

Una vez descompuesta una señal de entrada arbitraria x(n) en una suma ponderada de impulsos, estamos preparados para determinar la respuesta de cualquier sistema lineal en reposo a cualquier señal de entrada. En primer lugar, designamos la respuesta y(n,k) del sistema al impulso unitario de entrada en n = k mediante el símbolo especial h(n,k), $-\infty < k < \infty$. Es decir [6],

$$\mathrm{y(n,k)} = \mathrm{h(n,k)} = \mathcal{F}[\delta(n-k)]\ [adimensional] \tag{2-9}$$

Observe en la Ecuación (2-9) que *n* es el índice de tiempos y *k* es un parámetro que muestra la posición del impulso de entrada. Si se cambia la escala del impulso que se encuentra a la entrada en una cantidad $ck \equiv x(k)$, la escala de la respuesta del sistema cambiará en la misma magnitud; es decir

$$c_k h(n,k) = x(k)\ h(n,k)\ \ [adimensional] \tag{2-10}$$

Por último, si la entrada es la señal arbitraria $x(n)$ expresada como una suma de impulsos ponderados:

$$x(n) = \sum_{k=-\infty}^{\infty} x(k)\ \delta(n-k) \quad [adimensional] \tag{2-11}$$

entonces la respuesta del sistema a $x(n)$ será la correspondiente suma de salidas ponderadas, luego

$$y(n) = \mathcal{F}[x(n)] = \mathcal{F}\left[\sum_{k=-\infty}^{\infty} x(k)\ \delta(n-k)\right] = \sum_{k=-\infty}^{\infty} x(k)\mathcal{F}[\delta(n-k)] \tag{2-12}$$

$$= \sum_{k=-\infty}^{\infty} x(k)\ h(n,k)\ \ [adimensional] \tag{2-13}$$

Evidentemente, la expresión (2-12) cumple la propiedad de superposición de los sistemas lineales y se conoce como sumatorio de superposición. Observe que (2-

12) es una expresión para la respuesta de un sistema lineal a cualquier secuencia de entrada arbitraria x(n). Esta expresión es una función tanto de x(n) como de las respuestas h(n,k) del sistema a los impulsos unitarios δ (n−k) para $-\infty < k < \infty$. Para obtener la Ecuación (2-12) hemos utilizado la propiedad de linealidad del sistema, pero no su propiedad de invarianza en el tiempo, ya que dicha expresión puede aplicarse a cualquier sistema lineal (e invariante en el tiempo) en reposo. Si además el sistema es invariante en el tiempo, la fórmula dada por (2-12) se simplifica considerablemente. De hecho, si la respuesta del sistema LTI al impulso unitario δ (n) se denota como h(n), es decir [6],

$$h(n) = \mathcal{F}[\delta(n)]\ [adimensional]$$

Entonces aplicando la propiedad de invarianza en el tiempo, la respuesta del sistema a la secuencia de impulsos unitarios desplazados δ ($n-k$) es

$$h(n-K) = \mathcal{F}[\delta(n-K)]\ [adimesnsional] \qquad (2\text{-}14)$$

Por tanto, la fórmula dada por (2-12) se reduce a

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)\, h(n-k) \quad [adimensional] \qquad (2\text{-}15)$$

Queda claro entonces que el sistema LTI en reposo queda completamente caracterizado por una única función h(n), es decir, su respuesta al impulso unitario δ (n). Por el contrario, la caracterización general de la salida de un sistema lineal e invariante en el tiempo requiere un número infinito de funciones de respuesta al impulso unitario, h(n,k), una para cada posible desplazamiento. La fórmula dada por (2.3.17) que proporciona la respuesta y(n) del sistema LTI como una función de la señal de entrada x(n) y de la respuesta al impulso h(n) se **denomina suma de convolución**. Decimos que la entrada x(n) se convoluciona con la respuesta al impulso h(n) para proporcionar la salida y(n). A continuación, vamos a explicar el procedimiento para calcular la respuesta y(n), tanto por medios matemáticos como gráficos, conocidas la entrada x(n) y la respuesta al impulso h(n) del sistema. Supongamos que deseamos calcular la salida del sistema en un determinadoinstante de tiempo, por ejemplo, para n = n0. De acuerdo con (2-15), la respuesta en n = n0 está dada por [6]:

$$y(n_0) = \sum_{k=-\infty}^{\infty} x(k)\, h(n_0-k) \quad [adimensional] \qquad (2\text{-}16)$$

La primera observación que se tiene que hacer es que el índice del sumatorio es k, y, en consecuencia, tanto la señal de entrada x(k) como la respuesta al impulso h(n0−k) son funciones de k. En segundo lugar, observe que las secuencias x(k) and h(n0 −k) se multiplican para formar una secuencia producto. La salida y(n0) es simplemente la suma de todos los valores de la secuencia producto. La secuencia h(n0−k) se obtiene a partir de h(k), reflejando primero h(k) respecto de k = 0 (el origen de tiempos), lo que da como resultado la secuencia h(−k). La secuencia reflejada se desplaza entonces n0 para proporcionar h(n0−k). En resumen, el proceso de calcular la convolución entre x(k) y h(k) implica los pasos siguientes:

1. Reflexión. Se refleja h(k) respecto de k = 0 para obtener h(−k).

2. Desplazamiento. Se desplaza h(−k) una cantidad n0 hacia la derecha (o la izquierda) si n0 es positivo (negativo), para obtener h(n0−k).

3. Multiplicación. Se multiplica x(k) por h(n0−k) para obtener la secuencia producto vn0(k)≡x(k)h(n0−k).

4. Suma. Se suman todos los valores de la secuencia producto vn0 (k) para obtener el valor de la salida en el instante n = n0.

Observe que este procedimiento proporciona la respuesta del sistema en un determinado instante de tiempo, por ejemplo, en n = n0. En general, se estará interesado en evaluar la respuesta del sistema en todos los instantes de tiempo del intervalo $-\infty < n < \infty$. Por tanto, los pasos 2 hasta 4 deben repetirse para todos los posibles valores del desplazamiento temporal, $-\infty < n < \infty$.

Con el fin de comprender mejor el procedimiento de evaluación de la convolución, se va a mostrar el proceso gráficamente. Las gráficas nos ayudan a explicar los cuatro pasos necesarios en el cálculo de la convolución [6].

Ejemplo

La respuesta al impulso de un sistema lineal invariante en el tiempo es:

$$h(n) = \{1,2,1,-1\}\ [adimensional] \qquad (2\text{-}17)$$

Determine la respuesta del sistema a la siguiente señal de entrada

$$\mathrm{X(n)}=\{1,2,3,1\}[adimensional] \qquad (2\text{-}18)$$

Solución. Se va a calcular la convolución empleando la fórmula (2-15), pero se utilizarán gráficas de las secuencias para poder apoyarse en los cálculos. En la Figura 2-7(a) se ilustra la secuencia de entrada $x(k)$ y la respuesta al impulso $h(k)$ del sistema, usando k como el índice de tiempos para ser coherentes con la expresión (2-15). El primer paso para calcular la convolución consiste en reflejar $h(k)$. La secuencia reflejada $h(-k)$ se muestra en la Figura 2-7(b). A continuación, podemos calcular la salida en $n = 0$, aplicando la fórmula (2-15), la cual es:

$$y(0) = \sum_{k=-\infty}^{\infty} x(k)h(-k) \quad [adimensional] \qquad (2\text{-}19)$$

Dado que $n = 0$, utilizamos $h(-k)$ directamente sin desplazarla. La secuencia producto:

$$V_0 = X(k)h(-k) \quad [adimensional] \qquad (2\text{-}20)$$

También se muestra en la Figura 2-7(b). Por último, la suma de todos los términos de la secuencia producto es:

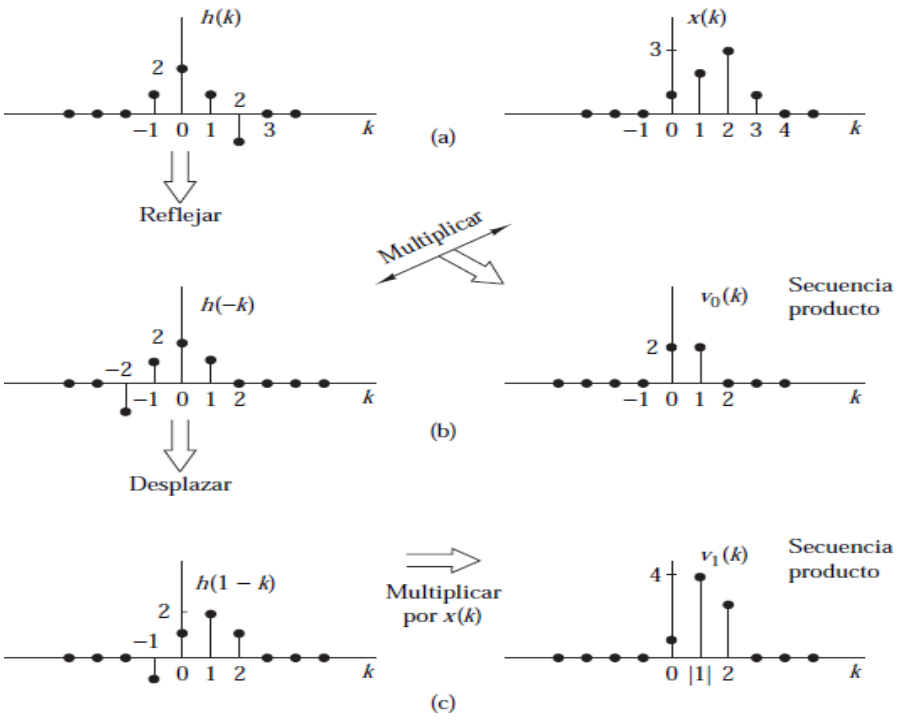
$$y(0) = \sum_{k=-\infty}^{\infty} h = -\infty V_0(k) = 4 \quad [adimensional] \qquad (2\text{-}21)$$

Continuando el cálculo evaluando la respuesta del sistema en n = 1. Según (2-15), y(1)

$$y(1) = \sum_{h=-\infty}^{\infty} x(k)h(1-k) \quad [adimensional] \tag{2-22}$$

La secuencia h(1−k) es simplemente la secuencia reflejada h(−k) desplazada hacia la derecha una unidad de tiempo. Esta secuencia se ilustra en la Figura 2-7(c). La secuencia producto [6]

$$V_1(k) = x(k)h(1-k)\ [adimensional] \tag{2-23}$$



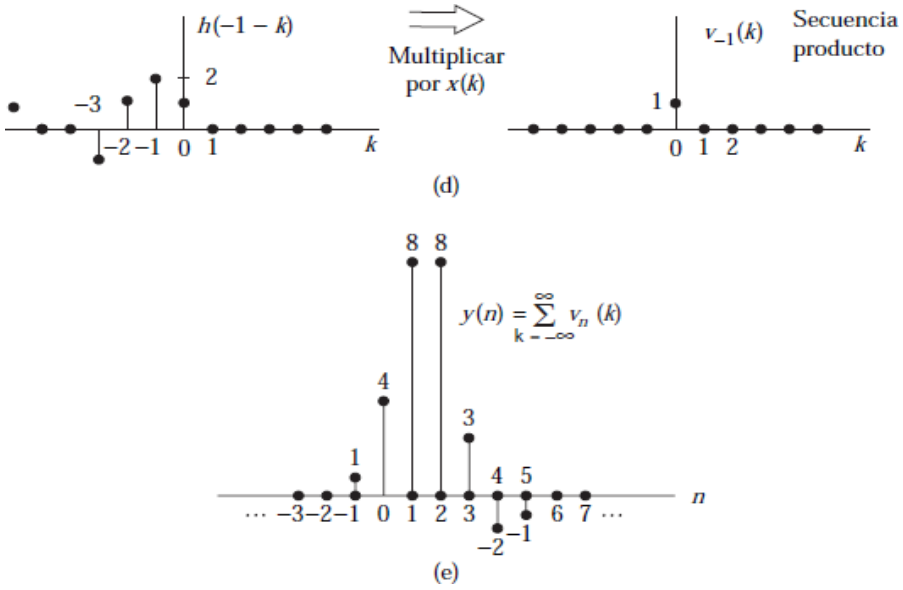


Figura 2-7: Cálculo de la convolución aplicando métodos gráficos

Fuente: Proakis J y Manolakis D, "*Tratamiento Digital de Señales.*"

Se muestra en la Figura 2-7(c). Por último, la suma de todos los valores de la secuencia producto es:

$$y(1) = \sum_{k=-\infty}^{\infty} V_1(k) = 8 \ [adimensional] \qquad (2\text{-}24)$$

De forma similar, se obtiene y(2) desplazando h(−k) dos unidades de tiempo hacia la derecha, para obtener la secuencia producto v2(k) = x(k)h(2−k) y luego sumamos todos los términos de la secuencia producto obteniendo y(2) = 8. Desplazando h(−k) hacia la derecha sucesivamente, multiplicando la secuencia correspondiente y sumando todos los valores de las secuencias producto resultantes, obtenemos y(3) = 3, y(4) = −2, y(5) = −1. Para n > 5, tenemos que y(n) = 0 porque las secuencias producto continúen únicamente ceros. Por tanto, hemos obtenido la respuesta y(n) para n > 0. A continuación deseamos evaluar y(n) para n < 0. Comenzamos con n = −1, luego [6]:

$$y(-1) = \sum_{k=-\infty}^{\infty} x(k)h(-1-k) \ [adimensional] \qquad (2\text{-}25)$$

Ahora la secuencia h(−1−k) es simplemente la secuencia reflejada h(−k) desplazada una unidad de tiempo hacia la izquierda. La secuencia resultante se muestra en la Figura 2-7(d). La secuencia producto correspondiente también se ilustra en la Figura 2-7(d). Por último, sumando todos los valores de la secuencia producto, obtenemos:

$$y(-1) = 1$$

Si se observan las gráficas de la Figura 2-7, es evidente que cualquier desplazamiento ulterior de h(−1−k) hacia la izquierda siempre da como resultado una secuencia producto cuyos valores son todos igual a cero, y, por tanto

$$y(n) = 0 \qquad para\ n \leq -2$$

Ahora se tiene la respuesta completa del sistema para $-\infty < n < \infty$, la cual resumimos como sigue

$$y(n) = \{\ldots,0,0,1,4,8,8,3,-2,-1,0,0,\ldots\} \quad [adimensional] \qquad (2\text{-}26)$$

2.1.7 Propiedades de la convolución

En este apartado se verán algunas propiedades importantes de la convolución y vamos a interpretar estas propiedades en función de la interconexión de los sistemas lineales invariantes en el tiempo. Veremos que estas propiedades se conservan para cualquier señal de entrada. Es cómodo simplificar la notación empleando un símbolo de asterisco para designar la operación de convolución. Por tanto [6],

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) \quad [adimensional] \qquad (2\text{-}27$$

Con esta notación, la secuencia que sigue al asterisco [es decir, la respuesta al impulso *h*(*n*)] se refleja y se desplaza. La entrada al sistema es *x*(*n*). Por otro lado, demostraremos también que:

$$y(n) = h(n) * x(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k) \quad [adimensional] \qquad (2\text{-}28)$$

En esta forma de la fórmula de la convolución, es la señal de entrada la que se refleja. Alternativamente, podemos interpretarla como el resultado de intercambiar los papeles que desempeñan x(n) y h(n). En otras palabras, podemos ver x(n) como la respuesta al impulso del sistema y h(n) como la excitación o señal de entrada.

LaFigura 2-8 ilustra esta interpretación [6].

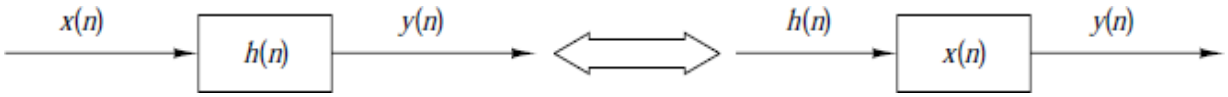


Figura 2-8: Interpretación de la propiedad conmutativa de la convolución

Fuente: Proakis J y Manolakis D, "*Tratamiento Digital de Señales.*"

Propiedades de identidad y desplazamiento. Se puede ver también que el impulso unitario δ (*n*) es el elemento identidad de la operación de convolución, es decir:

$$y(n) = x(n) * \delta(n) = x(n)$$

Si se desplaza δ (*n*) una cantidad *k*, la secuencia de convolución se desplaza también una cantidad *k*, luego

$$x(n) * \delta(n-k) = y(n-k) = x(n-k)$$

Se puede apreciar la convolución de forma más abstracta como una operación matemática entre dos señales, por ejemplo, *x*(*n*) y *h*(*n*), que satisface una serie de propiedades. La propiedad indicada en las expresiones (2-27) y (2-28) **es la ley conmutativa.**

Ley conmutativa

$$x(n) * h(n) = h(n) * x(n) \quad [adimensional] \qquad (2\text{-}29)$$

Desde el punto de vista matemático, la operación de convolución también satisface la ley asociativa, la cual puede enunciarse como sigue [6].

Ley asociativa

$$[x(n) * h_1(n)] * h_2(n) = x(n) * [h_1(n) * h_2(n)] \quad [adimensional] \qquad (2\text{-}30)$$

Desde el punto de vista físico, podemos interpretar x(n) como la señal de entrada a un sistema lineal invariante en el tiempo con una respuesta al impulso h1(n). La salida de este sistema, designada por y1(n), se convierte en la entrada a un segundo sistema lineal invariante en el tiempo con una respuesta al impulso h2(n). Así, la salida

$$y(n) = y_1(n) * h_2(n) = [x(n) * h_1(n)] * h_2(n)$$

Es precisamente el lado izquierdo de la Ecuación (2-30), la cual corresponde a dos sistemas lineales invariantes en el tiempo conectados en cascada. El lado derecho de la Ecuación (2-30) indica que la entrada x(n) se aplica a un sistema equivalente que tiene una respuesta al impulso, como, por ejemplo, h(n), que es igual a la convolución de las dos respuestas al impulso; es decir,

$$h(n) = h_1(n) * h_2(n)$$

$$y(n) = x(n) * h(n)$$

Además, puesto que la operación de convolución satisface la propiedad conmutativa, es posible intercambiar el orden de los dos sistemas con respuestas h1(n) y h2(n) sin alterar la relación global de entrada–salida. La Figura 2-9 ilustra gráficamente la propiedad asociativa [6].

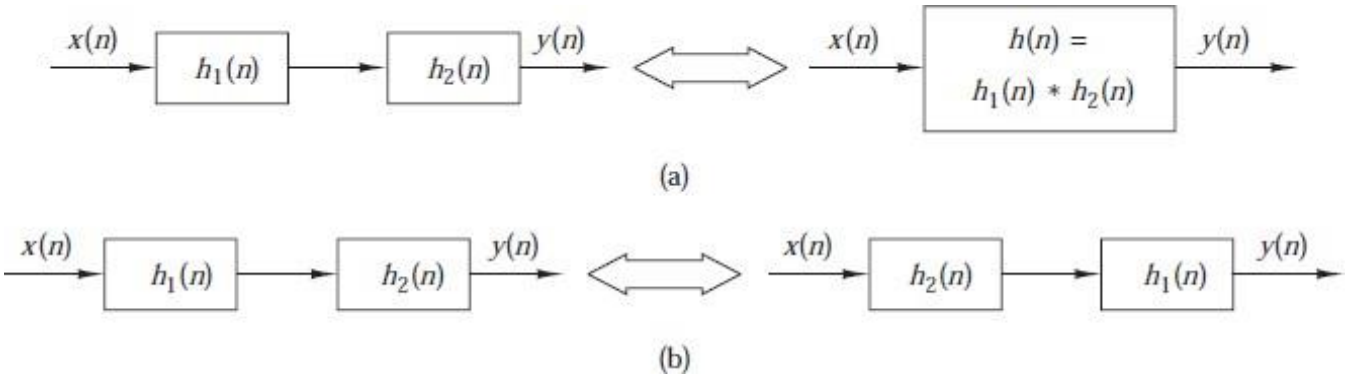


Figura 2-9: Implicaciones de las propiedades de la convolución (a) asociativa y (b) asociativa y conmutativa.

Fuente: Proakis J y Manolakis D, "*Tratamiento Digital de Señales*."

Es posible generalizar la ley asociativa a más de dos sistemas conectados en cascada fácilmente a partir de la exposición anterior. Así, si tenemos L sistemas lineales invariantes en el tiempo conectados en cascada con respuestas al impulso h1(n), h2(n), . . ., hL(n), existe un sistema lineal invariante en el tiempo que tiene una respuesta al impulso igual a (L−1) convoluciones sucesivas de las respuestas al impulso. Es decir,

$$h(n) = h_1(n) * h_2(n) * \ldots . * h_L(n) \ [adimensional] \qquad (2\text{-}31)$$

La ley conmutativa implica que el orden en que se efectúen las convoluciones es indiferente. A la inversa, cualquier sistema lineal invariante en el tiempo puede descomponerse en una interconexión en cascada de subsistemas [6].

Ley distributiva.

$$x(n) * [h_1(n) + h_2(n)] = x(n) * h_1(n) + x(n) * h_2(n)\ [adimensional] \quad (2\text{-}32)$$

Por tanto, el sistema completo es una combinación en paralelo de los dos sistemas lineales invariantes en el tiempo, como se ilustra en la Figura 2-10.

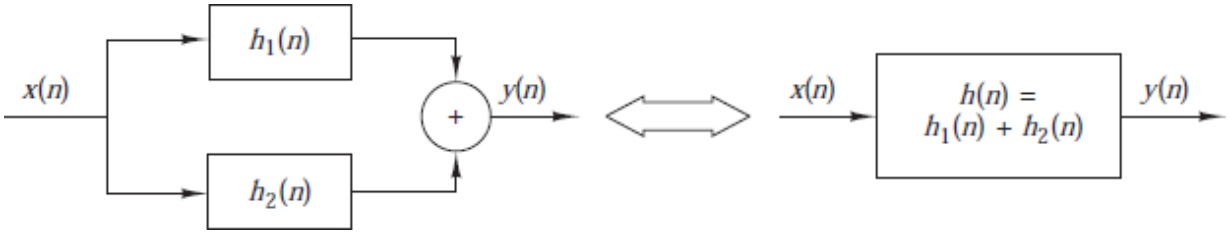


Figura 2-10: Interpretación de la propiedad distributiva de la convolución: dos sistemas LTI conectados en paralelo pueden reemplazarse por un único sistema h(n)=h1(n)+h2(n).

Fuente: Proakis J y Manolakis D, "*Tratamiento Digital de Señales*."

La generalización de la expresión (2-32) a más de dos sistemas lineales invariantes en el tiempo conectados en paralelo puede obtenerse fácilmente por deducción matemática. Por tanto, la interconexión de L sistemas lineales invariantes en el tiempo en paralelo con respuestas al impulso h1(n), h2(n), . . . ,hL(n) y excitados por la misma señal de entrada x(n) es equivalente a un sistema global con una respuesta al impulso [6]:

$$h(n) = \sum_{j=1}^{L} h_j(n)\ \ [adimensional] \quad (2\text{-}33)$$

Inversamente, cualquier sistema lineal invariante en el tiempo se puede descomponer en una interconexión paralelo de subsistemas.

2.1.8 Aplicaciones de la convolución

La convolución y las operaciones relacionadas se encuentran en muchas aplicaciones de ingeniería y matemáticas.

- En estadística, como un promedio móvil ponderado.
- En teoría de la probabilidad, la distribución de probabilidad de la suma de dos variables aleatorias independientes es la convolución de cada una de sus distribuciones de probabilidad.
- En óptica, muchos tipos de *manchas* se describen con convoluciones. Una sombra (p. ej. la sombra en la mesa cuando se tiene la mano entre esta y la fuente de luz) es la convolución de la forma de la fuente de luz que crea la sombra y del objeto cuya sombra se está proyectando. Una fotografía desenfocada es la convolución de la imagen correcta con el círculo borroso formado por el diafragma del iris.
- En acústica, un eco es la convolución del sonido original con una función que represente los objetos variados que lo reflejan.
- En ingeniería eléctrica, electrónica y otras disciplinas, la salida de un sistema lineal (estacionario o bien tiempo-invariante o espacio- invariante) es la convolución de la entrada con la respuesta del sistema a unimpulso (ver animaciones).
- En física, allí donde haya un sistema lineal con un principio de superposición, aparece una operación de convolución [17].

Al final de este capítulo podemos darnos cuenta de que la convolución es una suma y acumulación de productos, es decir, la convolución es una MAC.

2.2.-CORRELACIÓN: ES UN CASO DE CONVOLUCIÓN.

2.2.1 Correlación de señales discretas en el tiempo

Una operación matemática muy similar a la convolución es la correlación. Al igual que en el caso de la convolución, la correlación implica dos señales. Sin embargo, a diferencia de la convolución, el objetivo al calcular la correlación entre dos señales es medir el grado de semejanza entre ambas señales y, por tanto, extraer alguna información que dependa de forma importante de la aplicación. La correlación de señales se encuentra a menudo en áreas de la ciencia y la ingeniería, como el radar, el sonar, las comunicaciones digitales, la geología y otras muchas.

Suponga que disponemos de dos señales $x(n)$ e $y(n)$ que deseamos comparar. En las aplicaciones de radar y sonar, $x(n)$ puede representar la versión muestreada de la señal transmitida e $y(n)$ puede representar la versión muestreada de la señal recibida en la salida del convertidor analógico–digital (A/D). Si hay un blanco en el espacio en el que el radar o el sonar están barriendo, la señal recibida $y(n)$ estará formada por una versión retardada de la señal transmitida, reflejada desde el blanco, y distorsionada por efecto del ruido aditivo. La Figura 2-11 describe el problema de la recepción de la señal del radar [6].

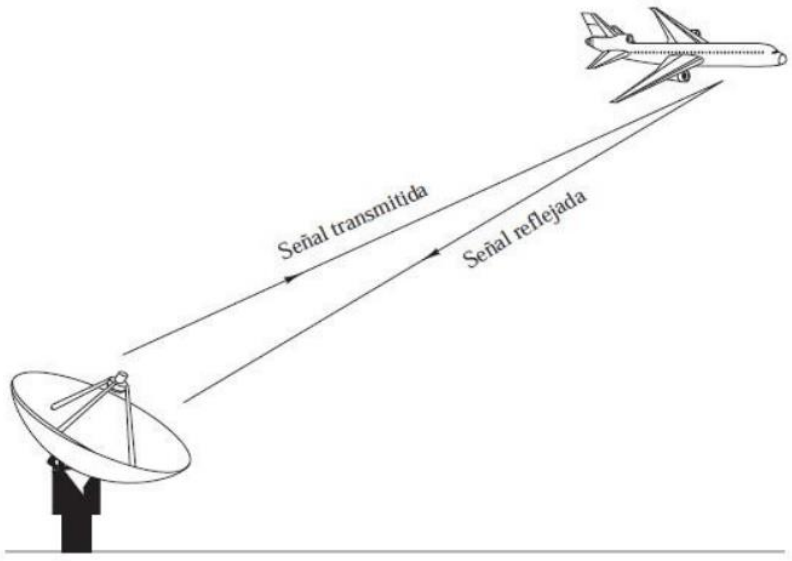


Figura 2-11: Detección de blancos mediante radar.

Fuente: Proakis J y Manolakis D, "*Tratamiento Digital de Señales*."

$$y(n) = \alpha x(n - D) + w(n)\ [adimensional] \qquad (2\text{-}34)$$

Donde α es un factor de atenuación que representa la pérdida de señal que se produce en la transmisión de ida y vuelta que sigue la señal x(n), D es el retardo de ida y vuelta, que se supone que es un múltiplo entero del intervalo de muestreo y w(n) representa el ruido aditivo que capta la antena y cualquier ruido generado por los componentes electrónicos y amplificadores contenidos en la entrada del receptor. Por el contrario, si no hay un blanco en el espacio barrido por el radar o el sonar, la señal recibida y(n) es solamente ruido. Disponiendo de las dos señales, x(n), que se conoce como señal de referencia o señal transmitida e y(n), que es la señal recibida, el problema de la detección por radar o sonar consiste en comparar y(n) y x(n) para determinar si hay presente un blanco y, en caso afirmativo, determinar el tiempo de retardo D y calcular la distancia al blanco. En la práctica, la señal x(n−D) se ve fuertemente distorsionada a causa del ruido aditivo, por lo que una inspección visual de y(n) no revela la presencia o ausencia de la señal reflejada deseada desde el blanco. La correlación nos proporciona un medio para extraer esta importante información de y(n).

Las comunicaciones digitales constituyen otra importante área en la que a menudo se emplea la correlación. En las comunicaciones digitales, la información que se va a transmitir de un punto a otro habitualmente se convierte a formato binario, es decir, una secuencia de ceros y unos, la cual se transmite entonces al receptor. Para transmitir un 0, podemos transmitir la secuencia de señal x0(n) para $0 \leq n \leq L-1$, y para transmitir un 1 podemos transmitir la señal x1(n) para $0 \leq n \leq L-1$, donde L es un entero que designa el número de muestras en cada una de las secuencias. Muya menudo, x1(n) se elige para que sea el valor negativo de x0(n). La señal recibida por el receptor se puede representar como [6]

$$y(n) = x_i(n) + w(n), \qquad i = 0{,}1 \; 0 \leq n \leq L-1 \; [adimensional] \quad (2\text{-}35)$$

Donde la incertidumbre ahora se encuentra en si x0(n) o x1(n) es la componente de señal de y(n), y w(n) representa el ruido aditivo y otras interferencias inherentes a cualquier sistema de comunicación. De nuevo, este ruido tiene origen en los componentes electrónicos de la etapa de entrada del receptor. En cualquier caso, el receptor conoce las posibles secuencias transmitidas x0(n) y x1(n) y se enfrenta a la tarea de comparar la señal recibida y(n) con x0(n) y x1(n) para determinar con cuál de las dos señales se corresponde mejor y(n) [6].

2.2.2 Secuencias de correlación cruzada y autocorrelación

Supongamos que tenemos dos secuencias de señal reales x(n) e y(n), teniendo cada una de ellas energía finita. La correlación cruzada de x(n) e y(n) es una secuencia rxy(l), la cual se define como:

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n)y(n-l), \quad l = 0, \pm 1, \pm 2, \ldots \; [adimensional]$$

0 de forma equivalente, como:

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n+l)y(n), \quad l = 0, \pm 1, \pm 2, \ldots \; [adimensional] \qquad (2\text{-}37)$$

El índice l es el parámetro de desplazamiento (tiempo) (o retardo) y el subíndice empleado en la secuencia de correlación cruzada rxy(l) indica las secuencias que se van a correlar. El orden de los subíndices, cuando x precede a y, indica la dirección en que se desplaza una secuencia respecto a la otra. En concreto, en la expresión (2-36), la secuencia x(n) no se desplaza e y(n) se desplaza l unidades de tiempo hacia la derecha para l positivo y hacia la izquierda si l es negativo. Del mismo modo, en la expresión (2-37), la secuencia y(n) no se desplaza y x(n) se desplaza l unidades de tiempo hacia la izquierda para valores de l positivos y hacia la derecha para valores de l negativos. Pero desplazar x(n) hacia la izquierda l unidades de tiempo respecto a y(n) es equivalente a desplazar y(n) hacia la derecha l unidades respecto de x(n). Por tanto, las expresiones (2-36) y (2-37) generan secuencias de correlación cruzada idénticas. Si invertimos los papeles de x(n) e y(n) en (2-37), invirtiendo por tanto el orden de los índices xy, obtenemos la secuencia de correlación cruzada [6].

$$r_{yx}(l) = \sum_{n=-\infty}^{\infty} y(n)x(n-l) \; [adimensional] \qquad (2\text{-}38)$$

O, lo que es equivalente,

$$r_{yx}(l) = \sum_{n=-\infty}^{\infty} y(n+l)x(n) \; [adimensional] \qquad (2\text{-}39)$$

Comparando (2-36) con (2-39) o (2-37) con (2-38), podemos concluir que:

$$r_{xy}(l) = r_{yx}(-l)[adimencional] \qquad (2\text{-}40)$$

Por tanto, ryx(l) es simplemente la versión reflejada de rxy(l), donde la reflexión se efectúa respecto a l = 0. Por tanto, ryx(l) proporciona exactamente la misma información que rxy(l), en lo que respecta a la similitud de x(n) e y(n).

Ejemplo

Determine la secuencia de correlación cruzada rxy(l) de las secuencias

$$x(n) = \{\ldots,0,0,2,-1,3,7,1,2,-3,0,0,\ldots\}$$

$$x(n) = \{\ldots,0,0,1,-1,2,-2,4,1,-2,5,0,0,\ldots\}$$

Solución: Utilizamos la definición dada en (2-36) para calcular rxy(l). Para l = 0, tenemos:

$$r_{xy}(0) = \sum_{n=-\infty}^{\infty} x(n)y(n)$$

La secuencia producto v0(n) = x(n)y(n) es

$$V_0(n) = \{\ldots,0,0,2,1,6,-14,4,2,6,0,0,\ldots\}$$

Y por lo tanto la suma para todos los valores de n es

$$r_{xy}(0) = 7$$

Para l > 0, simplemente desplazamos y(n) hacia la derecha l unidades respecto de x(n), calculamos la secuencia producto vl (n) = x(n)y(n−l) y, por último, sumamos para todos los valores de la secuencia producto. Luego obtenemos:

$$r_{xy}(1) = 13, \quad r_{xy}(2) = -18, \quad r_{xy}(3) = 16, \quad r_{xy}(4) = -7$$

$$r_{xy}(5) = 5, \quad r_{xy}(6) = -3, \quad r_{xy}(l) = 0, \quad l \geq 7$$

Para l < 0, desplazamos y(n) hacia la izquierda l unidades respecto de x(n), calculamos la secuencia producto vl (n) = x(n)y(n−l) y sumamos para todos los valores de la secuencia producto. Obtenemos entonces los valores de la correlación cruzada:

$$r_{xy}(-1) = 0, \quad r_{xy}(-2) = 33, \quad r_{xy}(-3) = -14, \quad r_{xy}(-4) = 36$$

$$r_{xy}(-5) = 19, \quad r_{xy}(-6) = -9, \quad r_{xy}(-7) = 10, \quad r_{xy}(1) = 0, \quad l \leq 8$$

Por tanto, la secuencia de correlación cruzada entre *x*(*n*) e *y*(*n*) es:

$$r_{xy}(1) = \{10, -9, 19, 36, -14, 33, 0, 7, 13, -18, 16, -7, 5, -3\}$$

Las similitudes entre el cálculo de la correlación cruzada de dos secuencias y la convolución de dos secuencias es evidente. En el cálculo de la convolución, una de las secuencias se refleja, luego se desplaza, a continuación, se multiplica por la otra secuencia para generar la secuencia producto correspondiente a dicho desplazamiento y, por último, se suman los valores de la secuencia producto. Excepto en lo que respecta a la operación de reflexión, el cálculo de la secuencia de correlación cruzada implica las mismas operaciones: desplazamiento de una de las secuencias, multiplicación de las dos secuencias y suma para todos los valores de la secuencia producto. En consecuencia, si disponemos de un programa para computadora que realice la convolución, podremos emplearlo para obtener la correlación cruzada proporcionando como entradas al programa la secuencia x(n) y la secuencia reflejada y(−n). Así, la convolución de x(n) e y(−n) proporciona la correlación cruzada rxy(l), es decir [6].

$$r_{xy}(l) = x(l) * y(l) \quad [adimensional] \qquad (2\text{-}41)$$

Observe que la ausencia de reflexión hace de la correlación cruzada una operación no conmutativa. En el caso especial en que y(n) = x(n), tenemos la autocorrelación de x(n), que se define como la secuencia:

$$r_{xx}(l) = \sum_{n=-\infty}^{\infty} x(n)x(n-l) \quad [adimensional] \qquad (2\text{-}42)$$

O de forma equivalente, como:

$$r_{xx}(l) = \sum_{n=-\infty}^{\infty} x(n+l)x(n) \quad [adimensional] \qquad (2\text{-}43)$$

Cuando se trabaja con secuencias de duración finita, es costumbre expresar la

autocorrelación y la correlación cruzada en función de los límites finitos del sumatorio. En concreto, si x(n) e y(n) son secuencias causales de longitud N [es decir, x(n) = y(n) = 0 para n < 0 y n ≥ N], las secuencias de la correlación cruzada y de la autocorrelación se pueden expresar del siguiente modo:

$$r_{xy}(l) = \sum_{n=1}^{N-|k|-1} x(n)y(n-l) \qquad [adimensional] \tag{2-44}$$

y

$$r_{xx}(l) = \sum_{n=i}^{N-|k|-1} x(n)x(n-l) \qquad [adimensional] \tag{2-45}$$

donde $i = 1$, $k = 0$ para $l \geq 0$ e $i = 0$, $k = 1$ para $1 < 0$.

2.2.3 Propiedades de la autocorrelación y de la correlación cruzada

Las secuencias de autocorrelación y correlación cruzada poseen una serie de importantes propiedades que vamos a ver a continuación. Para desarrollar estas propiedades, suponemos que disponemos de dos señales, x(n) e y(n), de energía finita a partir de las cuales formamos la combinación lineal [6]:

$$ax(n) + by(n - l)$$

donde a y b son constantes arbitrarias y l es un cierto desplazamiento de tiempo. La energía de esta señal es:

$$\sum_{n=-\infty}^{\infty} [ax(n) + by(n-l)]^2 = a^2 \sum_{n=-\infty}^{\infty} x^2(n) + b^2 \sum_{n=-\infty}^{\infty} y^2(n-l) + 2ab \sum_{n=-\infty} x(n)y(n-l) = a^2 r_{xx}(0) + b^2 r_{yy}(0) + 2abr_{xy}(l) \quad (2\text{-}46)$$

En primer lugar, observamos que rxx(0) = Ex y ryy(0) = Ey, que son las energías de x(n) e y(n), respectivamente. Es obvio que

$$a^2 r_{xx}(0) + b^2 r_{yy}(0) + 2abr_{xy}(l) \geq 0 \ \ [adimensional] \quad (2\text{-}47)$$

Ahora supongamos que b≠0, entonces podemos dividir (2-47) entre b^2 para obtener

$$r_{xx}\left(\frac{a}{b}\right) 2 + 2r_{xy}\ (l) + 2r_{xy}(1)\left(\frac{a}{b}\right) + r_{yy}(0) \geq 0 \ \ [adimensional] \quad (2\text{-}48)$$

Se puede considerar esta ecuación como una ecuación cuadrática de coeficientes rxx(0), 2rxy(l) y ryy(0). Dado que esta ecuación nunca es negativa, se deduce que su discriminante no será positivo, es decir

$$4[r_{xy}^2(1) - r_{xx}(0)r_{yy}(0)] \leq 0 \quad (2\text{-}49)$$

Por tanto , la correlación cruzada satisface la siguiente condición:

$$|r_{xy}(1)| \leq \sqrt{r_{xx}(0)r_{yy}(0)} = \sqrt{E_x E_y}\ [adimensional]$$

En el caso especial en el que y(n)=x(n), se reduce a

$$|rxx(1)| \leq rxx(0) = Ex\ [adimensional] \qquad (2\text{-}50)$$

Esto significa que la secuencia de autocorrelación de una señal alcanza su valor máximo para un retardo de cero. Este resultado es coherente con la idea de que una señal se corresponde de forma perfecta consigo misma para un retardo igual a cero. En el caso de la secuencia de la correlación cruzada, la cota superior de sus valores está dada por la expresión (2-49).

Observe que, si se aplica un factor de escala a una o ambas señales implicadas en la correlación cruzada, la forma de la secuencia de la correlación cruzada no varía, únicamente se modificarán sus amplitudes en el mismo factor de escala. Puesto que el cambio de escala no es importante, en la práctica se suelen normalizar las secuencias de correlación cruzada y autocorrelación en el rango comprendido entre −1 y 1. En el caso de la autocorrelación, simplemente dividimos entre *rxx*(0). Por tanto, la autocorrelación normalizada se define como [6]:

$$p_{xx}(l) = \frac{r_{xx}(l)}{r_{xx}(0)} \quad [adimensional] \qquad (2\text{-}51)$$

Del mismo modo, definimos la correlación cruzada normalizada como

$$p_{xx}(l) = \frac{r_{xy}(l)}{\sqrt{r_{xx}(0)r_{yy}(0)}} \quad [adimensional] \qquad (2\text{-}52)$$

Así, |ρxx(l)| ≤ 1 y |ρxy(l)| ≤ 1 y, por tanto, estas secuencias son independientes del cambio de escala que se pueda aplicar a la señal. Por último, como ya hemos demostrado, la correlación cruzada satisface la propiedad

$$r_{xy}(l) = r_{yx}(-l)$$

Si *y*(*n*) = *x*(*n*), esta relación se convierte en la siguiente importante propiedad para

la secuencia de autocorrelación

$$r_{xx}(l) = r_{xx}(-l) \ \ [adimensional] \tag{2-53}$$

Luego la función de autocorrelación es una función par. En consecuencia, basta con calcular rxx(l) para l ≥ 0.

2.2.4 Secuencias de correlación de entrada–salida

En esta sección vamos a deducir dos relaciones de entrada–salida para los sistemas LTI en el "dominio de la correlación". Supongamos que una señal x(n) con una autocorrelación conocida rxx(l) se aplica a un sistema LTI con una respuesta al impulso h(n), generando la señal de salida

$$y(n) = h(n) * x(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k)$$

La correlación cruzada entre la señal de salida y la de entrada es:

$$r_{yx}(l) = y(l) * x(-l) = h(l) * [x(l) * x(-l)]$$

Ó

$$r_{yx}(l) = h(l) * r_{xx}(l) \ \ [adimensional] \tag{2-54}$$

donde hemos *utilizado* $r_{xy}(l) = x(l) * y(-l)$ y las propiedades de la convolución. Por tanto, la correlación cruzada entre la entrada y la salida del sistema es la convolución de la respuesta al impulso con la autocorrelación de la señal de entrada.

Alternativamente, ryx(1) puede interpretarse como la salida del sistema LTI cuando la secuencia de entrada es rxx(l). Esto se ilustra en la Figura 2-12 [6].

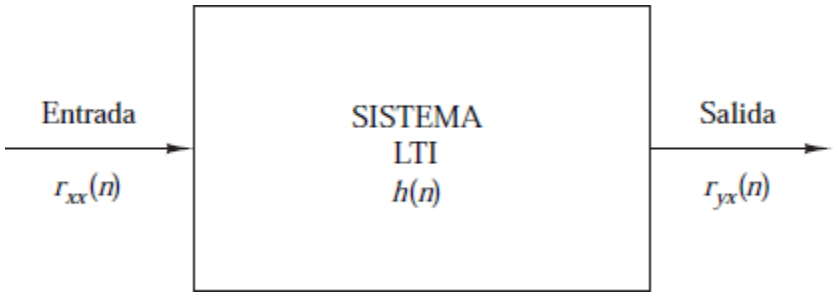


Figura 2-12: Relación de entrada-salida para la correlación cruzada $r_{yx}(n)$.

Fuente: Proakis J y Manolakis D, "*Tratamiento Digital de Señales*."

Si reemplazamos 1 por −1en (2-54), obtenemos

$$r_{xy}(l) = h(-l) * r_{xx}(l)$$

La autocorrelación de la señal de salida se puede obtener aplicando $r_{xy}(l) = x(l) * y(-l)$ con $x(n)=y(n)$ y las propiedades de la convolución. Por tanto, tenemos

$$\begin{aligned} r_{yy}(l) &= y(l) * y(-l) \\ &= [h(l) * x(l)] * [h(-l) * x(-l)] \\ &= [h(l) * h(-l)] * [x(l) * x(-l)] \\ &= r_{hh}(l) * r_{xx}(l) \quad [adimensional] \end{aligned} \tag{2-55}$$

La autocorrelación rhh(l) de la respuesta al impulso h(n) existe si el sistema es estable. Además, la estabilidad asegura que el sistema no cambiará el tipo (energía o potencia) de la señal de entrada. Evaluando (2-55) para l = 0, obtenemos:

$$r_{yy}(0) = \sum_{k=-\infty}^{\infty} r_{hh}(k) r_{xx}(k) \quad [adimensional] \tag{2-56}$$

Que proporciona la energía (o la potencia) de la señal de salida en términos de las autocorrelaciones. Estas relaciones se cumplen tanto para señales de energía como de potencia [6].

En conclusión, como podemos observar, la correlación es un caso particular de la convolución, es decir, esta también es una acumulación de productos y por lo tanto podemos decir que la correlación también es una MAC.

2.3.-TRANSFORMADA DE FOURIER: TRANSFORMADA Y SERIE DE FOURIER.

2.3.1 Que es la Transformada y la Serie de Fourier

Durante los últimos 200 años se han desarrollado distintos métodos y técnicas de procesamiento digital, para la detección y evaluación de funciones que posteriormente se aplicaron al tratamiento de señales. El análisis espectral de una señal pretende analizar en detalle el comportamiento y aporte de sus componentes armónicas en el dominio de la frecuencia. Para determinar el espectro más simple de una función se puede recurrir a la Transformada de Fourier (FT).

Las series e integrales de Fourier constituyen un tema clásico del análisis matemático. Desde su aparición en el siglo XVIII en el estudio de las vibraciones de una cuerda, las series de Fourier han sido una piedra de toque para el desarrollo de los conceptos básicos del análisis –función, integral, serie, convergencia...–, y la evolución de estos conceptos ha ido abriendo a su vez nuevos rumbos en el análisis de Fourier. Así lo expresa Zygmund en el prólogo de su famoso libro sobre series trigonométricas (1958):

Esta teoría ha sido una fuente de nuevas ideas para los analistas durante los dos últimos siglos y probablemente lo será en los próximos años. Muchas nociones y resultados básicos de la teoría de funciones han sido obtenidos por los matemáticos trabajando sobre series trigonométricas. Es concebible pensar que estos descubrimientos podían haber sido realizados en contextos diferentes, pero de hecho nacieron en conexión con la teoría de las series trigonométricas. No fue accidental que la noción de función aceptada ahora generalmente fuera formulada en la celebrada memoria de Dirichlet (1837) que trata de la convergencia de la serie de Fourier, o que la definición de integral de Riemann en su forma general apareciese en el Habilitationsschrift de Riemann sobre series trigonométricas, o que la teoría de conjuntos, uno de los desarrollos más importantes de las matemáticas del siglo XIX, fuera creada por Cantor en su intento de resolver el problema de los conjuntos de unicidad para series trigonométricas. En épocas más recientes, la integral de Lebesgue se desarrolló en estrecha conexión con la teoría de series de Fourier y la teoría de funciones generalizadas (distribuciones) con la de las integrales de Fourier.[27]

Se llama serie trigonométrica de periodo 2π a toda serie de funciones de la forma:

$$\frac{a_o}{2} + \sum_{k=1}^{\infty}(a_k \cos kx + b_k \operatorname{sen} kx)\ [adimensional]. \qquad (2\text{-}57)$$

Se llama polinomio trigonométrico de grado N y periodo 2π a toda expresión de la forma:

$$\frac{a_o}{2} + \sum_{k=1}^{N}(a_k \cos kx + b_k \sin kx)\ [adimensional]. \qquad (2\text{-}58)$$

Si al menos uno de los coeficientes aN y bN es distinto de cero se dice que el grado del polinomio es N. Obsérvese que las sumas parciales de las series trigonométricas (2-57) son polinomios trigonométricos.

Si suponemos que la serie (2-57) converge uniformemente en [−π, π] a la función f, escribimos la igualdad:

$$f(x) = \frac{a_o}{2} + \sum_{k=1}^{\infty}(a_k \cos kx + b_k \sin kx)\ [adimensional]. \qquad (2\text{-}59)$$

Y la integramos en [−π, π]:

$$\int_{-\pi}^{\pi} f(x)dx = \pi a_o$$

De donde sale el valor de a0. Del mismo modo, si multiplicamos la igualdad (2-59) por cos kx e integramos en [−π, π], la propiedad de ortogonalidad da:

$$\int_{-\pi}^{\pi} f(x)\cos kx\, dx = \pi a_o$$

Haciendo lo mismo con sin kx llegamos a:

$$\int_{-\pi}^{\pi} f(x)\sin kx\, dx = \pi a_o$$

Los valores de ak, bk que se obtienen son los siguientes:

$$a_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos kx\, dx \qquad b_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin kx\, dx \quad [adimensional]. \qquad (2\text{-}60)$$

Definición. Dada una función integrable f, los números {ak, k = 0, 1, 2, . . . } y {bk ,k = 1, 2, . . . } dados por las fórmulas (2-60) se llaman coeficientes de Fourier de f. La serie trigonométrica (2-57) construida con estos coeficientes se llama serie de

Fourier de f.

Los coeficientes dependen de la función y cuando intervienen simultáneamente coeficientes de varias funciones distintas, conviene hacer explícita esta dependencia; en esos casos escribiremos ak(f) y bk(f).

Observemos que, si f es un polinomio trigonométrico, el intercambio de sumas e integrales está perfectamente justificado por la linealidad de la integral y deducimos que los coeficientes del polinomio trigonométrico (2-58) vienen dados por las fórmulas (2-60).[27]

2.3.2 Expresiones de la Transformada de Fourier

Hemos considerado funciones periódicas de periodo 2π. Si el periodo es:

$$\frac{a_o}{2} + \sum_{k=1}^{\infty}(a_k \cos\frac{\pi kx}{l} + b_k \sin\frac{\pi kx}{l}) \ [adimensional]. \qquad (2\text{-}61)$$

Las fórmulas de los coeficientes también deben adaptarse convenientemente y quedan:

$$a_k = \frac{1}{l}\int_{-l}^{l} f(x)\cos\frac{\pi kx}{l}\,dx, \qquad b_k = \frac{1}{l}\int_{-l}^{l} f(x)\sin\frac{\pi kx}{l}\,dx \quad [adimensional].$$

La función real con valores complejos e^{it} se define como:

$$e^{it} = cost + isint$$

Y cambiando t por -t se tiene también:

$$e^{-it} = cost - isint$$

Sumando y restando estas expresiones se deduce:

$$cost = \frac{e^{it} + e^{-it}}{2}, \qquad sint = \frac{e^{it} - e^{-it}}{2i}$$

Entonces, la serie de Fourier de una función se puede escribir en forma compleja

como:

$$\sum_{k=-\infty}^{\infty} c_k e^{ikt} \ [adimensional]. \tag{2-62}$$

$$c_k = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(t)e^{-ikt}dt$$

Nótese que el coeficiente Cn generaliza los coeficientes a y b respectivamente para la identidad de Euler.

Fourier demostró que prácticamente cualquier función periódica se puede representar como una suma de Senos y Cosenos asignándole a cada uno un coeficiente de ponderación. Como ejemplo, se ha implementado una función cuadrada de periodo 2Π y amplitud 1 con tres términos y seis coeficientes, cuya función se analiza a continuación: Los coeficientes de la función periódica son 1, 0, 1/3, 0, 1/5, 0,1/7 generando la siguiente f(t):

$$f(t) = \sin(t) + \frac{1}{3}\sin(3t) + \frac{1}{5}\sin(5t) + \frac{1}{7}\sin(7t) \tag{2-63}$$

( ) ( )

Si se hace que el período de la función a transformar tienda a infinito, entonces se obtiene la transformada de Fourier de la función.

$$C(w) = \frac{1}{2}\int_{-\infty}^{\infty} f(t)e^{-iwt}dt$$

La integral impropia que aparece en estos coeficientes (conocida como la transformada de Fourier), resulta ser de gran importancia en el análisis de Fourier y en muchas otras aplicaciones. La transformada de Fourier, como se aprecia en la Figura 2 se utiliza en el estudio de señales y sistemas, así como en óptica; aparece en los equipos más modernos y sofisticados como los usados para realizar una tomografía, también surge en técnicas analíticas como la resonancia magnética nuclear, y en general, en todo tipo de instrumentación científica que se use para el análisis y presentación de datos.[18]

$$F(w) = \int_{-\infty}^{\infty} f(t)e^{-iwt}dt$$

De igual forma, se propone la transformada inversa de Fourier para todo t:

$$F^{-1}\{F(w)\} = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(w)e^{iwt}dw$$

Transformada Discreta de Fourier: Análogo a la serie, la transformada de Fourier descompone la señal en senos y cosenos de diferentes frecuencias y amplitudes. Es de destacar que el uso de esta transformada implica la solución de integrales que hacen el análisis continuo para todo tiempo. En la práctica, no siempre es posible por el consumo de tiempo o el desconocimiento de la función original, puesto que solo se poseen datos discretos resultantes de una captura. En las aplicaciones de ingeniería y tratamiento de señales, resulta más útil considerar el proceso de manera discreta y no continua, ya que los sistemas de adquisición de datos no pueden obtener ni analizar la totalidad de la información.[27]

$$X(w) = \sum_{n=-\infty}^{\infty} x[n]e^{-jwn} \qquad (2\text{-}65)$$

Y su inversa es:

$$x[n] = \frac{1}{2\pi}\int_{-n}^{n} X(w)e^{jwn}dw$$

El cálculo de la DFT requiere la suma compleja de N multiplicaciones complejas para cada una de las salidas. En total, N2 multiplicaciones complejas y N(N-1) sumas complejas para realizar un DFT de N puntos.

La Transformada Rápida de Fourier FFT: La evaluación directa de la DFT requiere (n^2) operaciones aritméticas. Mediante un algoritmo FFT se puede obtener el mismo resultado con sólo (n log n) operaciones. La FFT es el algoritmo que se utiliza para realizar la DFT de una forma eficiente y rápida. Lo que se consigue con este algoritmo es simplificar enormemente el cálculo de la DFT introduciendo “atajos” matemáticos para reducir drásticamente el número de operaciones. La idea que permite esta optimización es la descomposición de la transformada a tratar en otras más simples y así sucesivamente hasta llegar a transformadas de 2 elementos donde k puede tomar los valores 0 y 1. Una vez resueltas las transformadas más simples

se agrupan en otras de nivel superior que deben resolverse de nuevo y así sucesivamente hasta llegar al nivel más alto. Al final de este proceso se ordenan los resultados obtenidos. Dado que la transformada discreta de Fourier inversa es análoga a la transformada discreta de Fourier, con distinto signo en el exponente y un factor 1/n, cualquier algoritmo FFT se puede adaptar fácilmente para el cálculo de la transformada inversa.

2.3.3 Para que sirve la Transformada de Fourier

A principios del siglo XIX, Joseph Fourier indica que toda función periódica puede ser expresada como una suma infinita de senos y cosenos en distintas frecuencias.

En funciones de Variable Compleja, la transformada de Fourier es una aplicación que hace corresponder a una función f, con valores complejos y definida en la recta, con otra función g definida de la manera siguiente:

$$F(w) = \int_{-\infty}^{\infty} f(t)e^{-iwt}dt$$

La transformada de Fourier es básicamente el espectro de frecuencias de una función. Un buen ejemplo de eso es lo que hace el oído humano, ya que recibe una onda auditiva y la transforma en una descomposición en distintas frecuencias (que es lo que finalmente se escucha). El oído humano va percibiendo distintas frecuencias a medida que pasa el tiempo, sin embargo, la transformada de Fourier contiene todas las frecuencias contenidas en todos los tiempos en que existió la señal; es decir, en la transformada de Fourier se obtiene un sólo espectro de frecuencias para toda la función.[20]

2.3.4Aplicaciones de la Transformada de Fourier

El poder extraordinario y la flexibilidad de las series y transformadas de Fourier se ponen de manifiesto en la asombrosa variedad de las aplicaciones que ellas tienen en diversas ramas de la matemática y de la física matemática, desde teoría de números y geometría hasta mecánica cuántica. En esta sección presentamos algunas de las más importantes aplicaciones del análisis de Fourier. Comenzamos dando una hermosa y elegante solución al que demostró ser uno de los más complejos problemas de la geometría plana: el famoso **problema isoperimétrico**.

Si C es una curva cerrada simple de clase C 1 y de longitud 1, entonces el área A encerrada por C satisface la desigualdad $A \leq \frac{1}{4}\pi$. La igualdad se satisface si y sólo

sí C es una circunferencia. En consecuencia, entre todas las curvas cerradas simples de longitud 1 la que encierra mayor área es la circunferencia.

Demostración. Supongamos que la curva C está parametrizada en la forma (x(t), y(t)), donde el parámetro t representa la longitud de arco. En virtud del teorema de Stokes

$$\begin{aligned} A &= \frac{1}{2}\int_{\partial\mathcal{C}}(x\,dy - y\,dx) = \frac{1}{2}\int_0^1 (x\dot{y} - y\dot{x})\,dt \\ &= \frac{1}{2}((\hat{x},\hat{\dot{y}})_{l_2} - (\hat{y},\hat{\dot{x}})_{l_2}) \quad \text{(Parseval)} \\ &= \frac{1}{2}\sum_{n\in\mathbb{Z}}(\hat{x}\overline{\hat{\dot{y}}} - \hat{y}\overline{\hat{\dot{x}}})(n) = \sum_{n\in\mathbb{Z}} i\pi n(\overline{\hat{x}}\hat{y} - \hat{x}\overline{\hat{y}})(n) \\ &= \sum_{n\in\mathbb{Z}} i\pi n 2i\mathrm{Im}(\overline{\hat{x}(n)}\hat{y}(n)) = -2\pi\sum_{n\in\mathbb{Z}} n\mathrm{Im}(\overline{\hat{x}(n)}\hat{y}(n)). \end{aligned}$$

Por otra parte, puesto que la curva C tiene perímetro 1, tenemos que:

$$\begin{aligned} 1 &= \int_0^1 (\dot{x}^2 + \dot{y}^2) = \|\hat{\dot{x}}\|_{l_2} + \|\hat{\dot{y}}\|_{l_2} = \sum_{n\in\mathbb{Z}} |\hat{\dot{x}}(n)|^2 + |\hat{\dot{y}}(n)|^2 \\ &= \sum_{n\in\mathbb{Z}} 4\pi^2 n^2(|\hat{x}(n)|^2 + |\hat{y}(n)|^2) = 4\pi^2\sum_{n\in\mathbb{Z}} n^2(|\hat{x}(n)|^2 + |\hat{y}(n)|^2). \end{aligned}$$

Luego,

$$\frac{1}{\pi}\left(\frac{1}{4\pi} - A\right) = \sum_{n\in\mathbb{Z}}\left[n^2(|\hat{x}|^2 + |\hat{y}|^2) + 2n\mathrm{Im}(\overline{\hat{x}(n)}\hat{y}(n))\right].$$

Escribamos $\hat{x} = \alpha + i\beta$, $\hat{y} = \gamma + i\delta$, entonces:

$$\begin{aligned} \frac{1}{\pi}\left(\frac{1}{4\pi} - A\right) &= \sum_{n\in\mathbb{Z}} n^2(\alpha^2 + \beta^2 + \gamma^2 + \delta^2) + 2n(\alpha\delta - \beta\gamma)) \\ &= \sum_{n\neq 0}\left[(n\alpha + \delta)^2 + (n\beta - \gamma)^2 + (n^2 - 1)(\delta^2 + \gamma^2)\right] \geq 0. \end{aligned}$$

Luego $A \leq \frac{1}{4\pi}$ que es precisamente el área encerrada por una circunferencia de longitud 1.

Supongamos ahora que se satisface la igualdad $A \leq \frac{1}{4\pi}$ y demostremos que la curva C es una circunferencia. En efecto, si $A \leq \frac{1}{4\pi}$, entonces

$$\sum_{n\neq 0} \left[(n\alpha+\delta)^2+(n\beta-\gamma)^2+(n^2-1)(\delta^2+\gamma^2)\right]=0.$$

Esta última relación implica que si |n| ≥ 2, entonces δ = γ = α = β = 0, y sí |n| = 1, entonces α(±1) = ∓δ(±1) y β(±1) = ±γ(±1). Por otra parte, $\alpha(1) = Re(x(1)) = \int_0^1 x(t)\cos(2\pi t) = \alpha(-1)$, similarmente β(1) = $Im(x(1)) = \int_0^1 x(t)\sin(2\pi t) = -\beta(-1)$. Análogamente, γ(1) = γ(−1) y δ(1) = −δ(−1).

Desarrollando las funciones x(t) y y(t) en series de Fourier, obtenemos entonces que:

$$x(t)=\sum_{n\in\mathbb{Z}}\hat{x}(n)e^{2\pi in}=\hat{x}(0)+2\alpha(1)\cos(2\pi t)-2\beta(1)\sin(2\pi t)$$

$$y(t)=\sum_{n\in\mathbb{Z}}\hat{y}(n)e^{2\pi in}=\hat{y}(0)+2\beta(1)\cos(2\pi t)+2\alpha(1)\sin(2\pi t).$$

Entonces deducimos que:

$$(x(t)-x(0))^2+(y(t)-y(0))^2=4(\alpha(1)^2+\beta(1)^2).$$

En otras palabras, la curva C es una circunferencia.[28]

OFDM (Orthogonal Frequency Division Multiplexing) es un esquema de modulación de banda ancha capaz de hacerle frente a los problemas de la recepción multitrayectoria, transmitiendo muchas señales digitales de banda angosta en paralelo y traslapadas dentro de una banda amplia (Dusan, 1998). Este aumento del número de canales de transmisión paralelos reduce la tasa de datos que cada portadora individual debe transportar y alarga el período de símbolo. Como resultado, el tiempo de retardo de las ondas reflejadas es comprimido dentro de un tiempo de símbolo. Este concepto surgió gracias a la multiplexación por división de frecuencia (FDM) publicada a mediados de los sesenta (Chang, 1966, p. 1775) y (Salzberg, 1967, p. 805). La idea era utilizar secuencias de datos paralelos y

subcanales traslapados para evitar el uso de ecualizadores de alta velocidad y combatir el ruido impulsivo y la distorsión multitrayectoria, así como para utilizar eficientemente el AB disponible. En los años 80, OFDM fue usado en módems de alta velocidad (desarrollados para redes telefónicas) comunicaciones móviles digitales y grabación de alta densidad con Codificación Trellis. En los años 90, OFDM fue explotado en comunicaciones de datos de banda ancha sobre canales móviles de radio FM, líneas de suscriptor digital de alta tasa de bit (HDSL, 1.6 Mb/s), asimétricas (ADSL, 1.536 Mb/s) y de muy alta velocidad (VHDSL, 100 Mb/s) además de difusión digital de audio (DAB) y de TV (HDTV). En OFDM, cada portadora es ortogonal al resto de portadoras, siendo la versión óptima de los esquemas de transmisión multiportadora ya conocidos. La diferencia más importante entre FDM y OFDM es que el primero asigna cada canal a un usuario mientras que el segundo asigna todos los canales a un usuario. Para una gran cantidad de subcanales, los arreglos de generadores sinusoidales y demoduladores coherentes requeridos en un sistema para lelo pueden llegar a ser desmesuradamente costosos y complejos. Para esto, el receptor necesita precisar la fase de las portadoras demoduladas y los tiempos de muestreo para mantener así, una interferencia entre subcanales aceptable.

(Weinstein and Ebert, 1971, p.628) Aplicaron la Transformada de Fourier Discreta (DFT) a los sistemas de transmisión de datos paralelos como parte del proceso de modulación y demodulación. De esta forma, para eliminar los bancos de osciladores de las subportadoras y de los demoduladores coherentes requeridos por FDM, una implementación completamente digital puede ser construida alrededor de un hardware de propósito especial que pueda realizar la Transformada Rápida de Fourier (FFT). Los avances recientes en tecnología VLSI permiten la fabricación de chips de alta velocidad que pueden realizar FFT de gran tamaño a un precio razonable.

Los principales objetivos que se persiguen con la aplicación de filtros son:

*Suavizar la imagen: reducir la cantidad de variaciones de intensidad entre píxeles vecinos.

* Eliminar ruido: eliminar aquellos píxeles cuyo nivel de intensidad es muy diferente al de sus vecinos y cuyo origen puede estar tanto en el proceso de adquisición de la imagen como en el de transmisión.

* Realzar bordes: destacar los bordes que se localizan en una imagen.

* Detectar bordes: detectar los píxeles donde se produce un cambio brusco en la función intensidad.

Por tanto, se consideran los filtros como operaciones que se aplican a los píxeles de una imagen digital para optimizarla, enfatizar cierta información o conseguir un efecto especial en ella.

El proceso de filtrado puede llevarse a cabo sobre los dominios de frecuencia y/o espacio.

Los **filtros de frecuencia procesan una imagen** trabajando sobre el dominio de la frecuencia en la Transformada de Fourier de la imagen. Para ello, ésta se modifica siguiendo el Teorema de la Convolución correspondiente:

1. Se aplica la Transformada de Fourier,

2. Se multiplica posteriormente por la función del filtro que ha sido escogido,

3. Para concluir re-transformándola al dominio espacial empleando la Transformada Inversa de Fourier, donde tenemos que:

$$F(t) = \frac{1}{2\pi} V.P.C \int_{-\infty}^{\infty} f(t) e^{iwt} dw$$

F(u,v): transformada de Fourier de la imagen original.

H(u,v): filtro atenuador de frecuencias.

Como la multiplicación en el espacio de Fourier es idéntica a la convolución en el dominio espacial, todos los filtros podrían, en teoría, ser implementados como un filtro espacial (ver figura 2-13).

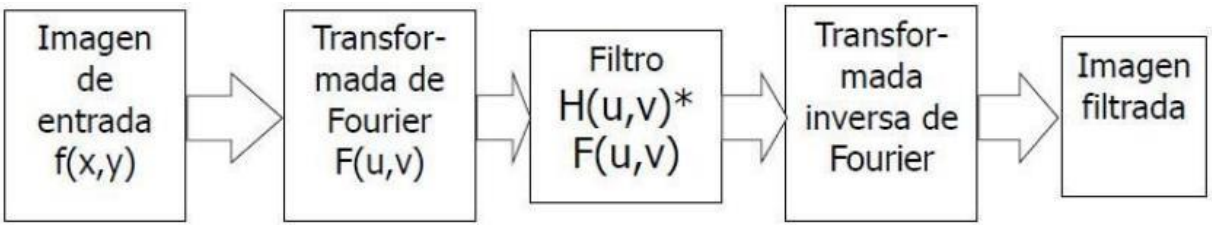


Figura 2-13: Implementación de un filtro espacial.

Fuente: Vela F, "*Transformada de Fourier en procesamiento digital de Imágenes Funciones de Variable Compleja*

El análisis de Fourier de una señal permite determinar sus frecuencias, pero a costa de perder la información de tipo temporal. (No dice cuando aparece cada frecuencia) En el caso de las imágenes, las "señales" corresponden a los niveles de

gris o intensidad de las diferentes filas o columnas de la matriz imagen (ver figura 2-14).

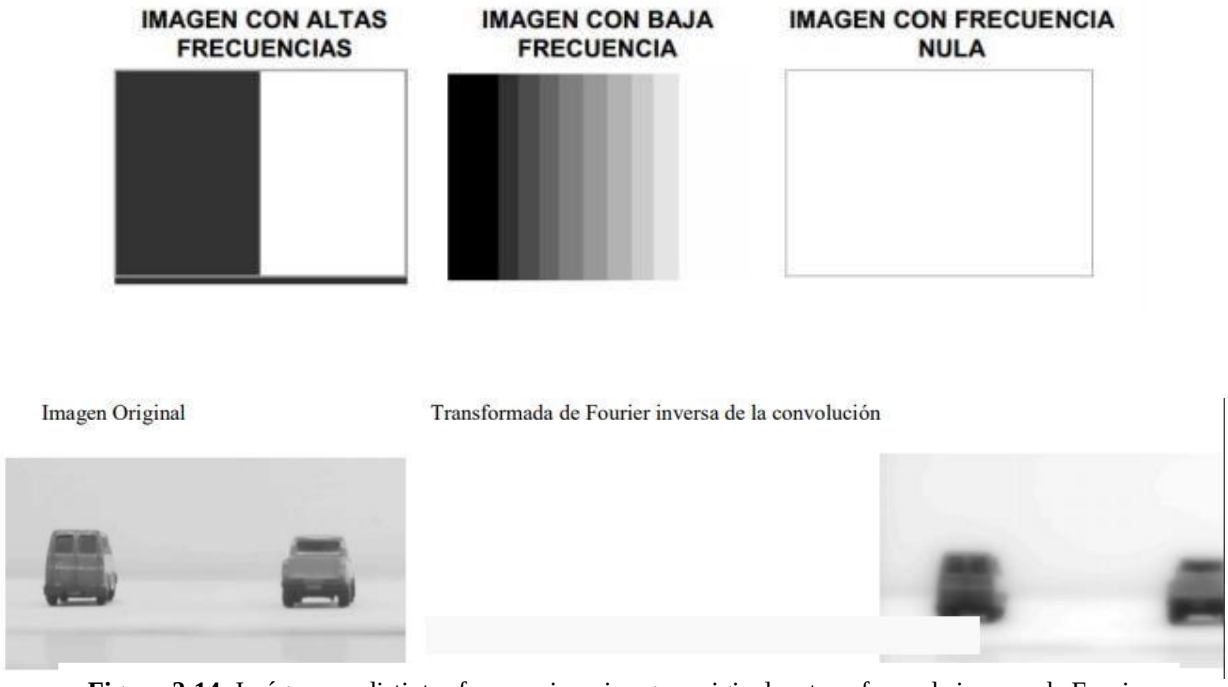


Figura 2-14: Imágenes a distintas frecuencias e imagen original vs transformada inversa de Fourier de la convolución.

Fuente: Vela F, "*Transformada de Fourier en procesamiento digital de Imágenes*

La transformada de Fourier se utiliza para pasar al dominio de la frecuencia una señal para así obtener información que no es evidente en el dominio temporal. Por ejemplo, es más fácil saber sobre qué ancho de banda se concentra la energía de una señal analizándola en el dominio de la frecuencia.

También sirve para resolver ecuaciones diferenciales con mayor facilidad y, por consiguiente, se usa para el diseño de controladores clásicos de sistemas realimentados si conocemos la densidad espectral de un sistema y la entrada podemos conocer la densidad espectral de la salida. Esto es muy útil para el diseño de filtros de radiotransistores.

La tomografía computarizada (TC) es un gran avance tecnológico utilizado en medicina para diagnosticar enfermedades. La misma técnica es usada en otras áreas, tales como química aeroespacial, radioastronomía, etc. El principal objetivo de la TC es reconstruir la parte interna de un objeto sin tener la necesidad de abrirlo, dicho procedimiento es conocido como un ensayo no destructivo. Para lograr este

propósito, la TC usa la atenuación que sufren los rayos X al atravesar por el objeto con dicha información, se reconstruye la parte interna del objeto, es decir la TC se trata de un problema inverso.

Se utiliza la transformada de Fourier para la reconstrucción de la parte interna, para ello se parte directamente de un modelo continuo, el método de Fourier es más eficiente.[20]

2.3.3 Relación entre la Transformada de Fourier y la MAC

En general, los procesadores de señales digitales se utilizan para realizar las operaciones de procesamiento de señales digitales como convolución, correlación, transformación y filtrado. Todas las operaciones de procesamiento de señales digitales mencionadas anteriormente están en forma de multiplicación y suma repetida. Por lo tanto, el circuito de acumulación y multiplicación (MAC) es el corazón del procesador de señales digitales. El filtro de respuesta de impulso finito digital general (FIR) [1] representado como (1), donde x [n] y y [n] son secuencias de señales de entrada y salida respectivamente. Aquí h [n] es la respuesta al impulso del filtro y norte es la longitud del filtro. Las secuencias de señales se pueden representar como números complejos de punto fijo / flotante.

Los números complejos juegan un papel vital en la electrónica y el procesamiento de señales digitales (DSP), porque son una forma fácil de representar y manipular las formas de onda sinusoidales más útiles del mundo real. Los atributos de la señal, como la amplitud y la fase, se pueden revelar fácilmente mediante números complejos que con números reales. Por ejemplo, los números complejos se utilizan en la transformada rápida de Fourier (FFT).

$$y[n] = \sum_{k=0}^{N-1} x[n-k]h[k] \tag{2-66}$$

2.4.-MAC VS MAC COMPLEJA.

2.4.1 Unidades MAC en serie y paralelo

En esta sección discutimos la operación de multiplicación compleja que emplea unidades MAC en serie y en paralelo. Una multiplicación compleja requiere cuatro veces más multiplicadores que la multiplicación de números reales. Considere un ejemplo de dos números complejos, la operación de multiplicación producirá dos salidas reales e imaginarias como se muestra en la Figura 2-15. Este modelo requiere cuatro multiplicadores para una salida compleja usando una sola señal de reloj.

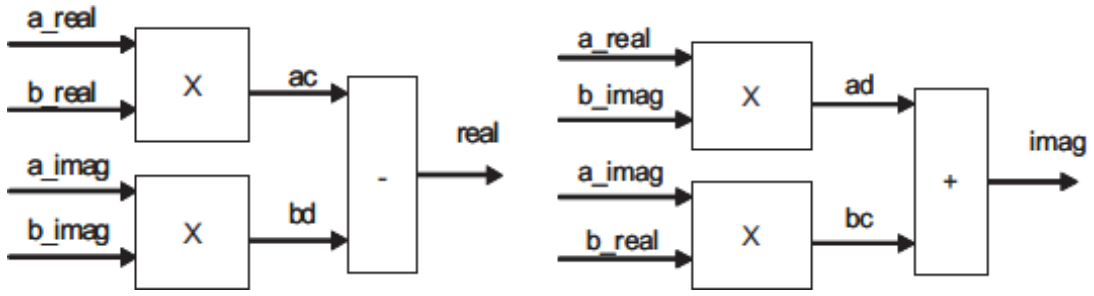


Figura 2-15: Arquitectura paralela para la multiplicación de dos números complejos.

Fuente: T. Salim, "Una arquitectura MAC en serie para la implementación FPGA de un complejo adaptativo".

Por otro lado, esta estructura se puede reemplazar con un modelo que utilice datos en serie y emplee un solo multiplicador. En esta arquitectura se realizan cuatro operaciones de multiplicación utilizando una única unidad MAC. En la Figura 2-16 se muestra un diagrama de bloques para dicho sistema.

Las estructuras en serie requieren multiplexación y demultiplexación y producen dos productos de salida iguales como tasa de entrada.

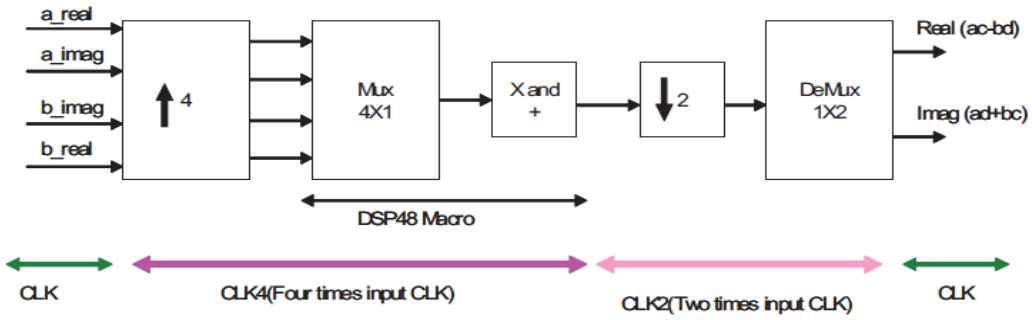


Figura 2-16: Arquitectura en serie para la multiplicación de dos números complejos.

Fuente: T. Salim, "Una arquitectura MAC en serie para la implementación FPGA de un complejo adaptativo".

Empleamos unidades MAC en serie y en paralelo en formadores de haz adaptativos y comparamos su rendimiento y recursos de hardware.

MAC es el bloque de construcción básico en todas las operaciones de procesamiento de señales digitales, como el filtro de convolución, correlación y respuesta de impulso finito (FIR). Las funciones DSP se implementan generalmente en procesadores DSP de propósito general donde se utilizan motores de acumulación múltiple (MAC) integrados para realizar operaciones matemáticas.

Los números complejos son muy importantes en las operaciones de procesamiento de señales digitales (DSP), principalmente DFT y FFT, porque son una forma fácil de representar y manipular las formas de onda sinusoidales más útiles del mundo real. Los atributos de la señal, como la amplitud y la fase, pueden revelarse fácilmente mediante números complejos que con números reales.

En general, la salida del sistema y [n] se puede obtener usando la convolución como:

$$y[n] = \sum_{k}^{\infty} x[n]h[n-k] \tag{2-67}$$

Donde x [n] y h [n] son la señal de entrada y la respuesta al impulso del sistema, respectivamente. El bloque de construcción básico de la unidad MAC se muestra en la figura 2-17, donde x y h son las dos entradas que se multiplican. El resultado de la multiplicación se guarda en el acumulador después de sumarlo al resultado anterior de la multiplicación. Si x y h tienen n bits de ancho, entonces el resultado de la multiplicación es 2n bits de ancho, por lo tanto, el acumulador puede constar de unos pocos bits más que 2n para evitar el desbordamiento. [25]

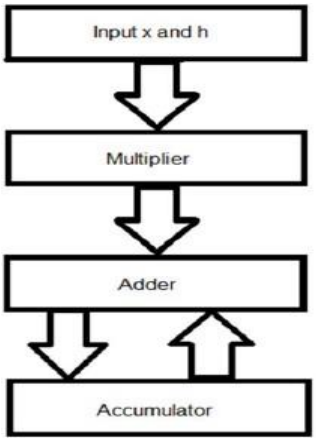


Figura 2-17: Unidad MAC.

Fuente: Mohamed A. & Noor M. “Un diseño MAC eficiente basado en hardware en filtros digitales con números

2.4.2 Diagrama a bloques de una unidad MAC compleja

En general, la señal digital (x) con amplitud (a) y fase (θ) se puede representar mediante un número complejo.

$$x = a\angle\theta = x + jy \tag{2-68}$$

Donde $a = x^2 + y^2$ y $\theta = \tan^{-1}(\frac{y}{x})$. Dos números complejos se representan como (a+jb) y (c+jd) la multiplicación de dos números complejos se dan como:

(a+jb)(c+jd) = (ac-bd) +j(ad+bc)

Por lo tanto, para calcular el resultado de la multiplicación anterior se requieren cuatro multiplicadores y dos sumadores. Por lo tanto, la arquitectura del multiplicador de números complejos se muestra en la figura 2-18.

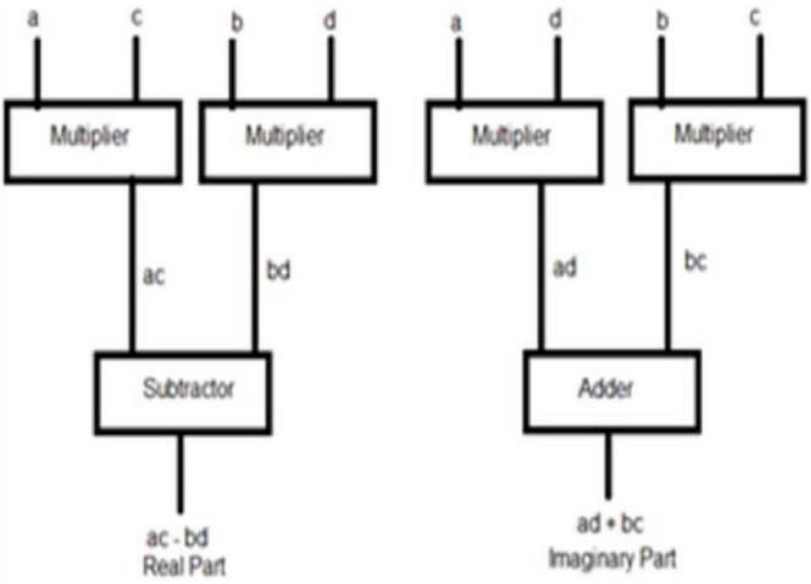


Figura 2-18: Unidad MAC Compleja.

Fuente: Mohamed A. & Noor M. "Un diseño MAC eficiente basado en hardware en filtros digitales con números complejos".

Los bloques básicos de MAC se muestran en la Fig.2-19, donde las entradas A y B se multiplican, el resultado de la multiplicación se suma al resultado MAC anterior. Si A y B están norte bits de ancho, el resultado de la multiplicación tendrá 2norte bits de ancho. Entonces, para evitar el desbordamiento durante la acumulación, el registro de acumulación tendrá k bits adicionales con su longitud real de 2n bits.

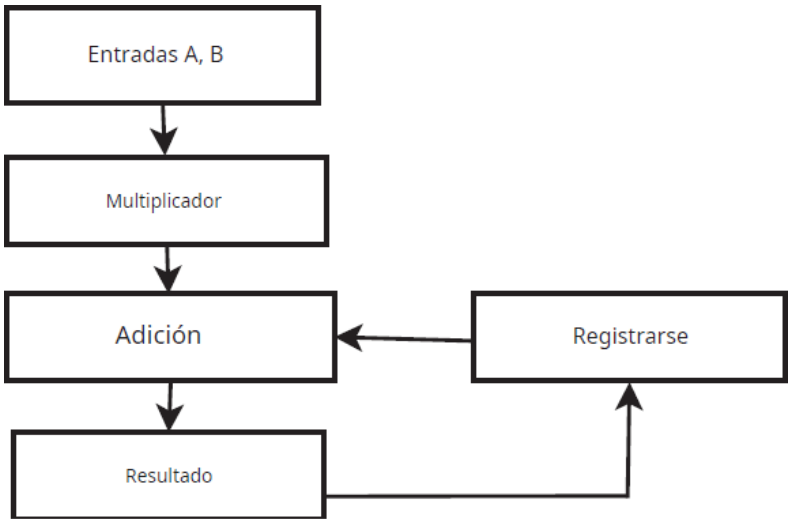


Figura 2-19: Bloques básicos de MAC.

Fuente: Mohamed A. & Noor M. "Un diseño MAC eficiente basado en hardware en filtros digitales con números complejos".

El multiplicador es la parte del MAC que se puede diseñar de muchas formas. El multiplicador de matriz y el multiplicador de árbol de Wallace son los multiplicadores populares que se utilizan en la implementación de hardware.

La segunda parte del MAC es un acumulador que se puede diseñar de varias formas, a saber, sumador de acarreo de ondulación y sumador de avance de acarreo.

El acarreo y la suma de la última etapa de almacenamiento de acarreo del multiplicador se envían a la primera etapa junto con resultado.

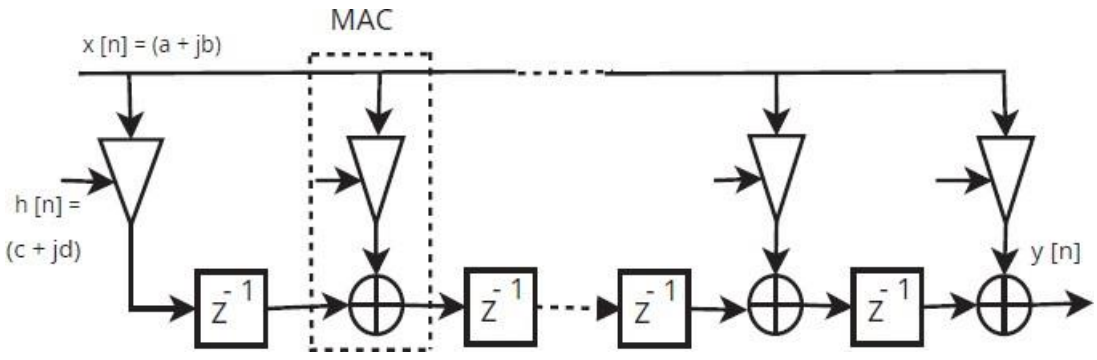


Figura 2-20: Estructura de filtro FIR básica con valores numéricos complejos.

Fuente: Mohamed A. & Noor M. "Un diseño MAC eficiente basado en hardware en filtros digitales con números complejos".

La figura 2-20 muestra la estructura básica del filtro FIR basada en la multiplicación constante múltiple con coeficiente de filtro de número complejo (h [n]) y valores de muestra de la señal de entrada (x [n]), donde el rectángulo punteado muestra la unidad MAC.[23]

2.5.- ALGORITMO CORDIC.

2.5.1 Teoría cordica: un algoritmo para la rotación vectorial

Todas las funciones trigonométricas pueden calcularse o derivarse de funciones usando rotaciones vectoriales, como se discutirá en las siguientes secciones. La rotación vectorial también se puede utilizar para conversiones polares a rectangulares y rectangulares a polares, para la magnitud del vector y como un bloque de construcción en ciertas transformaciones como la DFT y la DCT. El algoritmo CORDIC (Coordinate Rotation Digital Computer) proporciona un método iterativo para realizar rotaciones vectoriales mediante ángulos arbitrarios utilizando solo cambios y adiciones. Los algoritmos CORDIC generalmente producen un bit adicional de precisión para cada iteración. Los algoritmos trigonométricos CORDIC se desarrollaron originalmente como una solución digital para problemas de navegación en tiempo real. El trabajo original se le atribuye a Jack Volder. Las extensiones de la teoría CORDIC basadas en el trabajo de John Walther y otros proporcionan soluciones a una clase más amplia de funciones. El algoritmo, atribuido a Volder, se deriva de la transformación de rotación general (Givens):

$$x' = x\cos\phi - y\sin\phi \tag{2-69}$$

$$y' = y\cos\phi + x\sin\phi \tag{2-70}$$

Que gira un vector en un plano cartesiano en el ángulo ϕ. Estos se pueden reorganizar para que:

$$x' = x\cos\phi * [x - y\tan\phi]$$

$$y' = y\cos\phi * [y + x\tan\phi]$$

Hasta ahora, nada está simplificado. Sin embargo, si los ángulos de rotación están restringidos de modo que $tan(\phi) = \pm 2^{-i}$, la multiplicación por el término tangente se reduce a una operación de desplazamiento simple. Se pueden obtener ángulos de rotación arbitrarios realizando una serie de rotaciones elementales sucesivamente más pequeñas. Si La decisión en cada iteración, i, es en quédirección rotar en lugar de rotar o no, entonces el termino $\cos(\delta i)$ se convierte enuna constante $(porque\ \cos(\delta i) = \cos(-\delta i))$ la rotación iterativa ahora se puede expresar como:

$$x_{i+1} = k\,[x_i - y_i * d_i * 2^{-i}] \quad (2\text{-}71)$$

$$y_{i+1} = k_i[y_i + x_i * d_i * 2^{-i}] \quad (2\text{-}72)$$

Donde:

$$k_i = \cos(tan^{-1}2^{-i}) = \frac{1}{\sqrt{1 + 2^{-2i}}} \quad (2\text{-}73)$$

$$d_i = \pm 1$$

Al eliminar la constante de escala de las ecuaciones iterativas se obtiene un algoritmo de cambio-suma para la rotación de vectores. El producto de Ki´s, se puede aplicar en cualquier otra parte del sistema o se puede tratar como parte de una ganancia de procesamiento del sistema. Ese producto se acerca a 0,6073 cuando el número de iteraciones llega al infinito. Por tanto, el algoritmo de rotacióntiene una ganancia, An de aproximadamente 1,647. La ganancia exacta depende del número de iteraciones y obedece a la relación:

$$A_n = \prod_{n} \sqrt{1 + 2^{-2i}} \quad (2\text{-}74)$$

El ángulo de una rotación compuesta se define de forma única por la secuencia de las direcciones de las rotaciones elementales. Esa secuencia se puede representar mediante un vector de decisión.

El conjunto de todos los vectores de decisión posibles es un sistema de medición angular basado en las conversiones entre este sistema angular y cualquier otro mediante una búsqueda. Un mejor método de conversión acumula los ángulos de rotación elementales en cada iteración. Los ángulos elementales se pueden expresar en cualquier unidad angular conveniente. Esos valores angulares son

proporcionados por una pequeña tabla de búsqueda (una entrada por iteración) o están cableados, dependiendo de la implementación. El acumulador de ángulos agrega una tercera ecuación en diferencias al algoritmo CORDIC:

$$z_{i+1} = z_i - d_i * tan^{-1}(2^{-i}) \tag{2-75}$$

Obviamente, en los casos en los que el ángulo es útil en la base arco tangente, este elemento adicional no es necesario.

El rotor CORDIC normalmente se opera en uno de dos modos. El primero, llamado rotación por Volder, rota el vector de entrada en un ángulo específico (dadocomo argumento). El segundo modo, llamado vectorización, rota el vector de entrada hacia el eje x mientras registra el ángulo requerido para hacer esa rotación.En el modo de rotación, el acumulador de ángulos se inicializa con el ángulo de rotación deseado. La decisión de rotación en cada iteración se toma para disminuir la magnitud del ángulo residual en el acumulador de ángulos. Por tanto, la decisiónen cada iteración se basa en el signo del ángulo residual después de cada paso. Naturalmente, si el ángulo de entrada ya está expresado en la base arco tangente binaria, el acumulador de ángulos puede eliminarse. Para el modo de rotación, las ecuaciones CORDIC son:

$$x_{i+1} = x_i - y_i * d_i * 2^{-i} \tag{2-76}$$

$$y_{i+1} = y_i + x_i * d_i * 2^{-i}$$

$$z_{i+1} = z_i - d_i * tan^{-1} * (2^{-i}) \tag{2-77}$$

Donde:

$$d_i = -1 \ \ si \ z_i < 0, +1 \ de \ lo \ contrario$$

Que proporciona el siguiente resultado:

$$x_n = A_n[x_0 \cos z_0 - y_0 \sin z_0] \tag{2-78}$$

$$y_n = A_n[y_0 \cos z_0 + x_0 \sin z_0] \tag{2-79}$$

$$z_n = 0$$

$$A_n = \prod_{n} \sqrt{1 + 2^{-2i}}$$

En el modo de vectorización, el rotador CORDIC rota el vector de entrada en cualquier ángulo que sea necesario para alinear el vector de resultado con el eje x.

El resultado de la operación de vectorización es un ángulo de rotación y la magnitud escalada de el vector original (el componente x del resultado). La función de vectorización trabaja buscando minimizar el componente del vector residual en cada rotación. El signo del componente y residual se usa para determinar en qué dirección rotar a continuación. Si el acumulador de ángulos se inicializa con cero, contendrá el ángulo atravesado al final de las iteraciones. En modo vectorial, las ecuaciones CORDIC son:

$$x_{i+1} = x_i - y_i * d_i * 2^{-i}$$

$$y_{i+1} = y_i + x_i * d_i * 2^{-i}$$

$$z_{i+1} = z_i - d_i * tan^{-1} * (2^{-i})$$

Donde:

$$d_i = -1 \ \ si \ z_i < 0, +1 \ de \ lo \ contrario$$

Entonces:

$$x_n = A_n \sqrt{x_0^2 + y_0^2} \qquad (2\text{-}80)$$

$$y_n = 0$$

$$z_n = z_0 + tan^{-1}\left(\frac{y_0}{x_0}\right) \qquad (2\text{-}81)$$

$$A_n = G_n \sqrt{1 + 2^{-2i}} \qquad (2\text{-}82)$$

Los algoritmos de vectorización y rotación CORDIC como se indica se limitan a ángulos de rotación entre $-\frac{\pi}{2}$ y $\frac{\pi}{2}$ .Esta limitación se debe al uso de 2 ° para la tangente en la primera iteración. Para ángulos de rotación compuestos mayores que a $\frac{\pi}{2}$, se requiere una rotación adicional. Volder describe una rotación inicial $\pm\frac{\pi}{2}$. Esto da la iteración de corrección:

$$x' = -d * y \qquad (2\text{-}83)$$

$$y' = -d * x \qquad (2\text{-}84)$$

$$z' = z + d * \frac{\pi}{2} \quad (2\text{-}85)$$

$$donde \;\; d = +1 \;\; si \; y < 0, -1 \; de \; lo \; contrario$$

No hay crecimiento para esta rotación inicial. Alternativamente, se puede hacer una rotación inicial de π o 0, evitando la reasignación de los componentes **x** y **y** a los elementos rotadores. Nuevamente, no hay crecimiento debido a la rotación inicial:

$$x' = d * x$$

$$y' = d * y$$

$$z' = z \; si \; d = 1, o \; z - \pi \; si \; d = -1$$

$$d = -1 \; si \; x < 0, +1 \; de \; lo \; contrario$$

Ambas formas de reducción asumen una representación módulo 2π del ángulo de entrada. El estilo de la primera reducción es más consistente con las rotaciones sucesivas, mientras que en la segunda La reducción puede ser más conveniente cuando el cableado está restringido, como suele ser el caso de FPGAS. El rotador CORDIC descrito se puede utilizar para calcular indirectamente varias funciones trigonométricas. La elección acertada de los valores y modos iniciales permite el cálculo directo de seno, coseno, arco, magnitud vectorial y transformaciones entre coordenadas polares y cartesianas.

2.5.2 Seno y coseno

La operación CORDIC en modo rotacional puede calcular simultáneamente el seno y el coseno del ángulo de entrada. Establecer el componente y del vector de entrada en cero reduce el resultado del modo de rotación a:

$$x_n = A_n * x_0 \cos z_0 \quad (2\text{-}86)$$

$$y_n = A_n * x_0 \sin z_0 \quad (2\text{-}87)$$

Al establecer x_o igual a $1/A_n$, la rotación produce el seno y el coseno sin escalar del argumento del ángulo, Zo. Muy a menudo, los valores de seno y coseno modulan un valor de magnitud. El uso de otras técnicas (por ejemplo, una tabla de consulta) requiere un par de multiplicadores para obtener la modulación. La técnica CORDIC realiza la multiplicación como parte de la operación de rotación y, por lo tanto, elimina la necesidad de un par de multiplicadores explícitos.

La salida del rotador CORDIC se escala por la ganancia del rotador. Si la ganancia no es aceptable, una sola multiplicación por el recíproco de la constante de ganancia colocada antes del rotador CORDIC producirá resultados sin escala. Vale la pena señalar que la complejidad del hardware del rotador CORDIC es aproximadamenteequivalente a la de un solo multiplicador con el mismo tamaño de palabra.

2.5.3 Transformación polar a rectangular

Una extensión lógica de la computadora de seno y coseno es un transformador de coordenadas polar a cartesiano. La transformación del espacio polar al cartesiano se define por:

$$x = r\cos\theta \tag{2-88}$$

$$y = r\sin\theta \tag{2-89}$$

Como se señaló anteriormente, la multiplicación por la magnitud se obtiene de forma gratuita con el rotador CORDIC. La transformación se logra seleccionando el modo de rotación con x_0 =magnitud polar, z_0 =fase polar y $y_0 = 0$. El resultado vectorial representa la entrada polar transformada en espacio cartesiano. La transformación tiene una ganancia igual a la ganancia del rotador, que debe tenerse en cuenta en algún lugar del sistema. Si la ganancia es inaceptable, La magnitud se puede multiplicar por el recíproco de la ganancia del rotador antes de que se presente al rotador CORDIC.

2.5.4 Rotación vectorial general

El rotor CORDIC en modo de rotación también es útil para realizar rotaciones vectoriales generales, como se encuentra en los sistemas de control y corrección de movimiento. Para la rotación general, el vector de entrada bidimensional se presenta a las entradas del rotor. El rotador rota el vector en el ángulo deseado. Lasalida se escala por la ganancia del rotor CORDIC, que debe tenerse en cuenta en otras partes del sistema. Si la escala es inaceptable, se requiere un par de multiplicadores constantes para compensar la ganancia. Los rotores CORDIC se pueden conectar en cascada en una arquitectura de árbol para una rotación generalen n dimensiones. Es posible cierta optimización de la rotación multidimensional para permitir ahorros computacionales sobre el caso general de n dimensiones.

2.5.5 Arco tangente

El arco tangente, θ= Atan (y / x), se calcula directamente usando el rotador CORDIC en modo de vectorización si el acumulador de ángulos se inicializa con cero. El argumento debe proporcionarse como una razón expresada como un vector (x, y). Presentar el argumento como una razón tiene la ventaja de poder representar el infinito (estableciendo x=0). Dado que el resultado de arco tangente se toma del acumulador de ángulo, el crecimiento del rotador CORDIC no afecta el resultado.

$$z_n = z_0 + tan^{-1}\left(\frac{y_0}{x_0}\right) \tag{2-88}$$

2.5.6 Magnitud del vector

El rotador CORDIC del modo de vectorización produce la magnitud del vector de entrada como un subproducto del cálculo del arco tangente. Después de la rotación del modo de vectorización, el vector se alinea con el eje x. Por lo tanto, la magnitud del vector es la misma que la componente x del vector girado. Este resultado es evidente en el resultado de las ecuaciones para el rotador en modo vectorial:

$$x_n = A_n\sqrt{x_0^2 + y_0^2} \tag{2-89}$$

El resultado de la magnitud se escala por la ganancia del procesador, que debe tenerse en cuenta en otra parte del sistema. Esta implementación de magnitud vectorial tiene una complejidad de hardware de aproximadamente un multiplicador del mismo ancho. La implementación de CORDIC representa un ahorro de hardware significativo en comparación con un procesador pitagórico equivalente. La precisión del resultado de la magnitud mejora en 2 bits por cada iteración realizada.

2.5.7 Transformación cartesiana a polar

La transformación cartesiana a polar consiste en encontrar la magnitud ($r = sqrt(x^2 + y^2)$) y el ángulo de fase ($\phi = atan\left(\frac{y}{x}\right)$) del vector de entrada,

(x, y). El lector reconocerá inmediatamente que ambas funciones son proporcionadas simultáneamente por el rotor CORDIC del modo de vectorización. La magnitud del resultado será escalada por el CORDIC ganancia del rotador, y debe tenerse en cuenta en otras partes del sistema. Si la ganancia es inaceptable,

puede corregirse multiplicando la magnitud resultante por el recíproco de la constante de ganancia.

2.5.8 Funciones CORDIC inversas

En la mayoría de los casos, si una función puede ser generada por una computadora de estilo CORDIC, también se puede calcular su inversa. A menos que se pueda calcular la inversa cambiando el modo del rotador, su cálculo normalmente implica comparar la salida con un valor objetivo. El inverso CORDIC se ilustra con la función Arco seno.

2.5.9 Arcoseno y Arco coseno

El Arcoseno se puede calcular comenzando con un vector unitario en el eje x positivo y luego rotándolo de modo que su componente y sea igual al argumento de entrada. El arcoseno es entonces el ángulo subtendido para hacer que la componente y del vector girado coincida con el argumento. La decisión de La función en este caso es el resultado de una comparación entre el valor de entrada yel componente y del vector girado en cada iteración:

$$x_{i+1} = x_i - y_i * d_i * 2^{-i} \quad (2\text{-}88)$$

$$y_{i+1} = y_i + x_i * d_i * 2^{-i} \quad (2\text{-}89)$$

$$z_{i+1} = z_i - d_i * tan^{-1} * (2^{-i}) \quad (2\text{-}90)$$

Donde:

$$d_i = +1 \ \ si\ y_i < c, -1\ de\ lo\ contrario\ y$$

$$c = argumento\ de\ entrada.$$

$$la\ rotación\ produce\ los\ siguientes\ resultados:$$

$$x_n = \sqrt{(A_n * x_0)^2 - c^2} \quad (2\text{-}91)$$

$$y_n = c$$

$$z_n = z_0 + arcsin\left(\frac{c}{A_n * x_0}\right) \quad (2\text{-}92)$$

$$A_n = \prod_n \sqrt{1 + 2^{-2i}} \quad (2\text{-}93)$$

La función de arcoseno, como se indicó anteriormente, devuelve ángulos correctos para las entradas $-1 < \frac{c}{A_n x_0} < 1$, aunque la precisión sufre a medida que la entrada

se acerca a $\pm$ 1 (el error aumenta rápidamente para las entradas mayores de aproximadamente 0.98). Esta pérdida de precisión se debe a la ganancia del rotador. Para ángulos cerca del eje y, la ganancia del rotador hace que el vector girado sea más corto que la referencia (entrada), por lo que las decisiones se toman incorrectamente. Los problemas de ganancia se pueden corregir utilizando un "algoritmo de doble iteración" [9] a costa de un aumento de la complejidad.

El cálculo de Arco-coseno es similar, excepto que la diferencia entre el componente x y la entrada se utiliza como función de decisión. Sin modificaciones,el algoritmo de arco-coseno funciona solo para entradas inferiores a $1/A_n$, por lo que el algoritmo de doble iteración es una necesidad. El arco-coseno también podría calcularse utilizando la función arcoseno y restando $\frac{\pi}{2}$ del resultado, seguido de una reducción angular si el resultado está en el cuarto cuadrante.

2.5.10 Extensión a funciones lineales

Una simple modificación de la ecuación CORDIC permite el cálculo de funciones lineales:

$$x_{i+1} = x_i - 0 * y_i * d_i * 2^{-i} = x \quad (2\text{-}94)$$

$$y_{i+1} = y_i + x_i * d_i * 2^{-i} \quad (2\text{-}95)$$

$$z_{i+1} = z_i - d_i * (2^{-i}) \quad (2\text{-}96)$$

$$para\ el\ modo\ de\ rotacion\ (d_i = -1\ si\ z_i < 0, +1\ de\ lo\ contrario\ )$$

La rotación lineal produce:

$$x_n = x_0 \quad (2\text{-}97)$$

$$y_n = y_0 + x_0 z_0 \quad (2\text{-}98)$$

$$z_n = 0 \quad (2\text{-}99)$$

Esta operación es similar a la implementación de cambio-suma de un multiplicador y, a medida que avanzan los multiplicadores, no es una solución óptima. La multiplicación es útil en aplicaciones donde ya está disponible una estructura CORDIC. El modo de vectorización ($d_i = +1\ si\ y_i < 0, -1$ en caso contrario) es más interesante, ya que proporciona un método para evaluar las proporciones (radios):

$$x_n = x_0 \tag{2-100}$$

$$y_n = 0$$

$$z_n = z_0 - \frac{y_0}{x_0} \tag{2-101}$$

Las rotaciones en el sistema de coordenadas lineales tienen una ganancia unitaria, por lo que no se requieren correcciones de escala.

CAPÍTULO III:

PROPUESTA DE DISEÑO DE LA MAC COMPLEJA.

En este capítulo se presenta la propuesta de diseño para La MAC compleja, se mencionaran los requerimientos que requiere la MAC compleja, entre ellos se hablara de las operaciones, funciones requeridas y consideraciones de hardware necesarios para crear la MAC compleja, posteriormente se realizara un diseño de una MAC, primeramente se realizara de manera sencilla y después se diseñara una MAC compleja, esta se programara en MATLAB y en el pic MSP430G2553, se hablara un poco de la descripción del problema y la solución que se sugiere para este y por último se trataran las especificaciones técnicas del diseño de la MAC compleja.

3.1.- REQUERIMIENTOS DE LA MAC COMPLEJA

Primero se debe recordar que una unidad MAC realiza ambas funciones, las cuales son multiplicar y sumar y que estas funciones están en dos etapas. En la primera etapa, la multiplicación de dos números se calcula y en la siguiente etapa, el resultado de la primera etapa es utilizado para sumar / acumular. Si tanto la multiplicación como suma se ejecuta en un solo redondeo, entonces se dice que es una unidad fusionada de acumulación múltiple (MAC). Entonces la unidad MAC básica contiene multiplicador, sumador y acumulador. El multiplicador multiplica las entradas y da el resultado al sumador, que suma el resultado del multiplicador al anterior resultado acumulado, este diagrama se puede consultar más a detalle en el capítulo 1, figura 1-14.

Teniendo de referencia el funcionamiento y la composición de una MAC común, ahora, para poder crear una MAC compleja se requieren cuatro multiplicadores y dos sumadores, es decir, ahora en lugar de tener dos entradas, como se tuvo en la MAC común, ahora se tendrán ocho entradas producidas a raíz de trabajar con números complejos, por lo tanto, para una MAC compleja se requieren cuatro multiplicaciones complejas y dos sumas reales, en donde con dos multiplicaciones y una suma se obtiene la parte real y con las dos multiplicaciones y la suma restante se obtiene la parte imaginaria. Esto se puede observar con más a detalle en el capítulo dos, figura 2-18.

También es de suma importancia mencionar que para la MAC compleja en algunas aplicaciones se requieren las funciones trigonométricas, las cuales son: $e^{j\theta}$, en donde se debe de calcular el valor de $e^{j\theta}$, donde teta puede ser cualquier valor arbitrario dependiendo de la aplicación que se le vaya a dar o incluso del propio diseñador.

3.1.1.-Operaciónes y funciones requeridas

Operaciones requeridas

Las operaciones requeridas para poder desarrollar una MAC compleja, básicamente son la suma y la multiplicación compleja. Estas dos operaciones se realizan respetando las leyes básicas de los números complejos y se explicaran a continuación.

Antes de operar con números complejos se debe de recordar algunos conceptos básicos de estos, los cuales son que:

- Un número complejo en su forma general está definido como: $\boldsymbol{z = x + jy}$
- $\boldsymbol{x} = Re(z)$, es decir, x es la parte real de z.
- $\boldsymbol{y} = Im(z)$, es decir, y es la parte imaginaria de z.
- El número complejo conjugado de z es, $\bar{z} = x - jy$
- j , es conocida como la unidad imaginaria.
- $j^2 = -1 \quad y \quad j = \sqrt{-1}$

Las operaciones requeridas para poder desarrollar una MAC compleja básicamente son la suma y la multiplicación compleja. Estas dos operaciones se realizan respetando las leyes básicas de los números complejos y se explicaran a continuación.

Ahora que ya se tienen estos conceptos básicos, se procederá a explicar de manera sencilla las dos operaciones que se utilizan en la MAC compleja.

Suma compleja

Para sumar dos números complejos , se debe sumar la parte real a la parte real y la parte imaginaria a la parte imaginaria.

Ejemplo:

Dados dos números complejos: $z_1 = a + jb \;\; y \;\; z_2 = c + jd$

$$z_1 + z_2 = (a + jb) + (c + jd) = a + c + jb + jd$$

$$z_1 + z_2 = (a + c) + j(b + d)$$

Multiplicación compleja

El producto de los números complejos se realiza aplicando la propiedad distributiva del producto respecto de la suma y teniendo en cuenta que $\boldsymbol{j^2 = -1}$. No se debe olvidar que cuando se multiplica un binomio por otro binomio, se debe usar la propiedad distributiva repetidamente.

Ejemplo:

Dados dos números complejos: $z_1 = a + jb \;\; y \;\; z_2 = c + jd$

$$z_1 + z_2 = (a + jb) + (c + jd)$$

$$z_1 + z_2 = ac + ajd + cjb + jbjd$$

$$z_1 + z_2 = ac + j(ad + cb) + j^2bd$$

$$z_1 + z_2 = (ac - bd) + j(ad + cb)$$

Funciones requeridas

Las funciones requeridas para una MAC compleja son las funciones trigonométricas, las cuales se basan en la exponencial compleja ($e^{j\theta}$), En Matemáticas y más específicamente Análisis Matemático es la versión de la función exponencial sobre el conjunto de los números complejos. Esta misma se basa en la representación exponencial de los números complejos y su relación con la forma trigonométrica. Es de vital importancia en la formulación de otras funciones complejas como el logaritmo, las funciones trigonométricas complejas, las funciones trigonométricas hiperbólicas complejas y otras.

Definición:

A la función $e^x;\ \mathbb{C} \rightarrow \mathbb{C},\quad donde\ x = a + jb$, expresada en forma de cálculo por:

$$e^x = e^{a+jb} = e^a e^{jb} = e^a(\cos b + j \sin b)$$

Se le denomina **función exponencial compleja** y también se denota como: exp(x) donde x en un argumento complejo.

Propiedades

La definición mencionada anteriormente, que no es otra que la relación entre las representaciones exponencial y trigonométricas de los complejos, cumple todas las propiedades que cumplía en los reales, excepto aquellas relacionadas con el ordenamiento porque los números complejos no son un conjunto ordenado. Entonces de debe contemplar que [30]:

- e^x está definida para todos los complejos
- $e^{x+y} = e^{x+y}$
- $(e^x)^y = e^{xy}$
- Para todo número complejo $z = ae^{jb}$ su conjugado es $z = ae^{-jb}$
- $e^{-x} = e^{\frac{x}{n}}; n \in setN; n \geq 2$
- $lne^x = x$

La ley de Euler

La ley de Euler establece un importante vínculo entre la trigonometría y el cálculo, al asociar la exponenciación de la parte imaginaria compleja con la representación de un complejo cuyo módulo es 1.

Esta es la base para la representación exponencial de complejos y como base demostrativa para la Ley de Moivre; sin excluir el hecho de clarificar el procedimiento para exponenciar complejos, calcular raíces, determinar logaritmos de argumentos negativos y facilitar el trabajo del cálculo computacional de las funciones trigonométricas.

La ley de Euler se define como:

$$e^{ix} = \cos x + \mathrm{j}\sin x$$

$$e^{-ix} = \cos x - \mathrm{j}\sin x$$

Donde expresado en términos de θ, la identidad quedaría como:

$$e^{i\theta} = \cos\theta + \mathrm{j}\sin\theta$$

$$e^{-i\theta} = \cos\theta - \mathrm{j}\sin\theta$$

Relación con la trigonometría

La fórmula de Euler también permite interpretar las funciones seno y coseno como meras variaciones de la función exponencial:

$$\cos x = \frac{e^{jx} + e^{-jx}}{2}$$

$$\sin x = \frac{e^{jx} - e^{-jx}}{2j}$$

Expresadas en función de teta seria:

$$\cos\theta = \frac{e^{j\theta} + e^{-j\theta}}{2}$$

$$\sin\theta = \frac{e^{j\theta} - e^{-j\theta}}{2j}$$

A partir de estas igualdades, es posible definir las funciones trigonométricas para los números complejos:

$$\cos z = \frac{e^{jz} + e^{-jz}}{2}$$

$$\sin z = \frac{e^{jz} - e^{-jz}}{2j}$$

$$\tan z = \frac{e^{jz} - e^{-jz}}{e^{jz} + e^{-jz}} \, x \frac{1}{j}$$

Siendo $z \in \mathbb{C}$, es decir, que pertenece al conjunto de números complejos. Estas funciones trigonométricas cumplen las leyes de sus similares aplicadas a los números reales, sean los números complejos z y w, es decir, z A $w \in \mathbb{C}$, entonces son validad las siguientes igualdades [31]:

$$cos^2 z + sin^2 z = 1$$

$$\cos(-z) = \cos(z)$$

$$\cos(-z) = -\sin(z)$$

$$\sin(z + w) = \sin(z)\cos(w) + \sin(w)\cos(z)$$

$$\cos(z + w) = \cos(z)\cos(w) - \sin(w)\sin(z)$$

3.1.2.-Consideraciones de Hardware

Si se desea implementar una MAC compleja en algún otro dispositivo que no sea el MSP430, se deben de tener en cuenta las consideraciones a nivel hardware que latarjeta de desarrollo o el pic en cuestión deben de cumplir para que se pueda implementar la MAC compleja, es decir, se necesita que el hardware sobre el cual se va a programar la MAC cuente con las operaciones que se necesitan y las funciones requeridas. Estas operaciones y funciones requeridas se mencionarán a continuación:

- El hardware debe ser capaz de trabajar con números flotantes.
- El hardware debe ser capaz de calcular la raíz cuadrada y el cuadrado de un número flotante.
- El hardware debe de ser de preferentemente de 16 bits, ya que con ello este podrá realizar más operaciones, es decir, podrá ser más preciso. Pero también se puede trabajar con hardware que sean de 8 bits.
- El hardware debe de poder trabajar con las funciones trigonométricas (seno, coseno, tangente inversa).
- El Hardware debe de poder trabajar con las funciones modulo.

- El hardware debe de ser capaz de manejar las funciones de tangente inversa,tangente cuadrada.

Nota1: algunas de estas pruebas mencionadas anteriormente se realizaron en la tarjeta de desarrollo MSP430 para corroborar que efectivamente este cumplía con las características necesarias para poder realizar una MAC, algunas de estas pruebas fueron: obtener el valor de la función seno, coseno, suma de ambos y tangente inversa, también cabe aclarar que en estas pruebas se trabajó con números flotantes y el MSP430 lo pudo realizar sin problemas, estas pruebas se pueden encontrar en el anexo 1, donde se mostrará el código en C++ y el resultado obtenido en la LCD.

3.2.- DISEÑO DE UNA MAC COMPLEJA EN EL PIC MSP430

Teniendo en cuenta todos los requerimientos necesarios para poder llevar a cabo una MAC, ahora se procederá a realizar el diseño de una MAC compleja en el MSP430, para ello se va a partir del diseño de una MAC sencilla, es decir, recordemos que la unidad MAC básica contiene un multiplicador, un sumador y un acumulador. El multiplicador multiplica las entradas y da el resultado al sumador, que suma el resultado del multiplicador al anterior resultado acumulado, ósea que, en la primera etapa, la multiplicación de dos números se calcula y en la siguiente etapa, el resultado de la primera etapa es utilizado para sumar / acumular. Si tanto la multiplicación como suma se ejecuta en un solo redondeo, entonces se dice que es una unidad fusionada de acumulación múltiple (MAC). Con esto presente, entonces se diseñará un programa en el MSP430, el cual realice una MAC, en esencia se definirán dos vectores (a y b) de la misma magnitud, los cuales contendrán números flotantes, después de ello con ayuda de los ciclos for y while se diseñará la MAC, que básicamente es multiplicar elemento a elemento los componentes de cada vector, sumarlos e irlos acumulando. (Este programa se puede observar a detalle en el anexo 2).

Pero hay que tener en cuenta que con este diseño únicamente se está trabajando con número reales, pero, ¿Qué pasaría si los valores de a y b fueran complejos?, la respuesta a esta pregunta seria que entonces se tendría que diseñar una MAC capaz de trabajar con números complejos, es decir, una MAC compleja, para diseñar una MAC compleja se tiene que considerar que para ello se requiere el uso de las funciones trigonométricas, y luego contemplar que como los resultados son complejos, también se requiere el uso de la función modulo, la función raíz cuadrada y la función tangente inversa para poder expresar el resultado, ya sea en forma trigonométrica, rectangular como se le conoce o en forma polar, entonces,

¿Dónde se podría tener una mac compleja?, la respuesta es, en una transformada de Fourier, la cual se puede observar en la ecuación 3-1.

$$X[K] = \sum_{n=0}^{N-1} X[n] e^{-j2\pi \frac{K}{N} n} \qquad (3\text{-}1)$$

Entonces lo que se hará, será diseñar una MAC compleja partiendo de un ejemplo muy común en la ingeniería como lo es la transformada de Fourier, ya que la transformada de Fourier es un claro ejemplo donde se lleva a cabo el uso de la MAC compleja.

3.2.1.-Diagrama a bloques

El diagrama a bloques de una MAC se presentará de forma clara y concreta en este apartado, en el diagrama a bloques de una MAC únicamente tendríamos, lo siguiente:

Una MAC con dos entradas, a y b, se necesita un reloj, ósea, algo que le esté diciendo a la MAC cada cuando debe de tomar el valor a y b, multiplicarlos e irlos acumulando. También debe de contar con una salida, y debe de tener un reinicio (Reset), básicamente este sería el diagrama a bloques de una MAC, y ahora la pregunta es: ¿Para qué me sirve el Reset en una MAC?, la respuesta es que este sirve para reiniciar la MAC, es decir, cuando se termine de calcular un valor y se requiera calcular otro, pero, que ahora los valores de a y b sean distintos, entonces el acumulador se inicia con cero, básicamente esa es la función del reset, el acumulador que es el que va acumulando se inicia en cero, y se empieza a acumular otra vez, este reloj es el que le va diciendo a la MAC cada cuando tomar los valores a y b, multiplicarlos y sumarlos al acumulador y entonces, cada vez que se van sumando se tiene un resultado de salida, donde la salida es el acumulado, cuando hay un reset, entonces el valor acumulado se vuelve cero y se reinicia la MAC , por consiguiente el diagrama a bloques de una MAC seria el siguiente (ver figura 3-1):

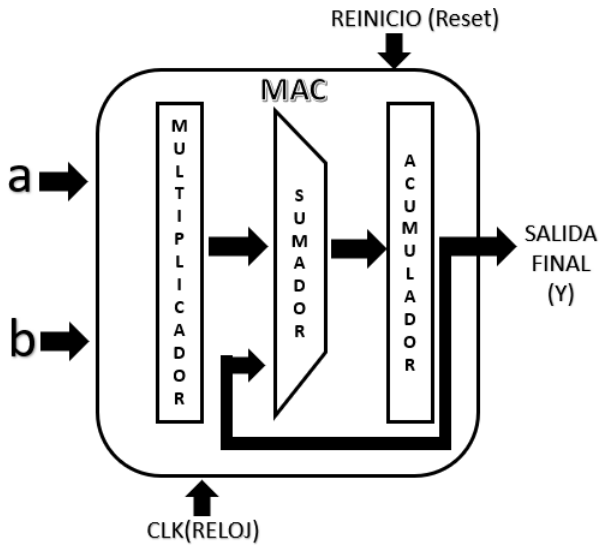


Figura 3-1: Diagrama a bloques de una MAC.

Fuente: autoría propia.

3.2.2.-Programación de la MAC compleja

En este apartado se explicara a detalle cómo se llevó a cabo la programación de la MAC compleja en el MSP430, para ello lo primero que se hará será diseñar la MAC compleja en MATLAB, ya que este software permite realizar operaciones con números complejos, manejo de vectores y de funciones trigonométricas, esto será de mucha ayuda para que a partir de ello se realice el diseño en el MSP430, aparte de que se tendrá como referencia los resultados que se obtengan en MATLAB y los resultados que se obtengan en el MSP430, ya que si las cosas se hacen correctamente estos deben de ser prácticamente los mismos.

Entonces el diseño en MATLAB para la MAC compleja se realizó de la siguiente manera:

Como se comentó anteriormente, para probar la MAC vamos a basarnos en una transformada de Fourier, recordando como se define (ecuación 3-1), la cual contiene una sumatoria que va desde n=0 hasta n-1, esta se muestra a continuación:

$$X[K] = \sum_{n=0}^{N-1} X[n] e^{-j2\pi \frac{K}{N} n}$$

Donde: X[n] =es un vector de números, el cual se manejará con números reales, aunque también se puede manejar con números complejos en su definición más general.

N=es el tamaño del vector de X[n].

$\sum_{n=0}^{N-1} X[n]e^{-j2\pi \frac{K}{N} n}$ = esto es una MAC, es una suma de productos, es decir, un acumulador multiplicador. Donde un acumulador es la sumatoria, y el multiplicador es lo que está dentro de la sumatoria. Ahora el factor de la exponencial como se va a crear primero en MATLAB, lo único a colocar seria:

$$\exp\left(\frac{-j * 2pi * k * n}{N}\right)$$

Para crear la MAC compleja se utilizarán 2 for, donde el primer for se utiliza para barrer k, para cada valor de k se aplica una MAC, y el segundo for se utiliza para barrer el valor de la constante n. entonces la programación de una MAC compleja a grandes rasgos estaría estructurada como se muestra a continuación:

```
for k=1: N1,

Acum=0;

For n=1: N

Producto= X[n]*exp((-j*2pi*(K-1)*(n-1)) /N);

Acum=Acum+Producto;

end;

X[K]=Acum;

end;
```

for k=1: N1, este sería el primer for (o for externo), que es el que va a ir obteniendo cada valor de X[K], para cada valor de K, es importante tomar en cuenta que debemos igualar N=N1.

Ahora para obtener el valor de la variable X[K], se necesita barrer la variable n, entonces se necesitaría otro for (segundo for o for interno), el cual sería: for n=1: N.

Se debe recordar que cuando se trabaja en MATLAB siempre se empieza en uno porque Matlab no permite empezar en cero.

Ahora, en X[K], primero se debe de hacer una suma de productos, entonces primero se realiza el producto y después se suma, por lo tanto, hay que definir, primeramente, un acumulador=0, es decir, es el inicio del acumulador. Donde se coloca el (K-1) para empezar en cero y el (n-1) también para empezar el barrido en n=0, así como lo indica la expresión.

En la expresión de Acum=Acum+Producto está es en esencia la MAC, entonces el primer for es el que barre las Ks, es decir, para cada valor de K se realiza todo el proceso descrito anteriormente, donde todo el proceso en su conjunto es una MAC, ósea, se tiene que aplicar el valor de K varias veces, lo que significa que, para cada valor de K se aplica una MAC. Todo esto se trabajó definiendo un vector llamado X_1 de 16 elementos. El segundo for es donde se barren todos los valores de n, es decir, donde para cada valor de n se hace un producto, y luego para cada producto, este se debe ir acumulando, ya que realiza un barrido completo, y se obtiene el primer valor de X[K] y ya solo se imprime este valor.

Nota 2: el código de este programa se puede consultar de manera más detallada en el anexo 3.

Después de programar la MAC compleja en MATLAB, ahora se procederá a programarla en el MSP430, para ello se creó un diagrama de flujo donde se puede observar cuales son los pasos necesarios para poder crear la MAC compleja, este diagrama de flujo está enfocado al diseño en el MSP430, ya que como se sabe MATLAB muestra los resultados en su misma interface, es decir en el comand window, y en el caso del MSP430 no es así, ya que el software que se utiliza para programar esta tarjeta de desarrollo no nos permite hacerlo y para ello se tiene que buscar alguna otra alternativa para mostrar los datos obtenidos, así que para esta ocasión se hará el diseño enfocado a la pantalla LCD de 16x32, la cual nos ayudara a mostrar los datos de salida, este diagrama se puede observar en la figura 3-2.

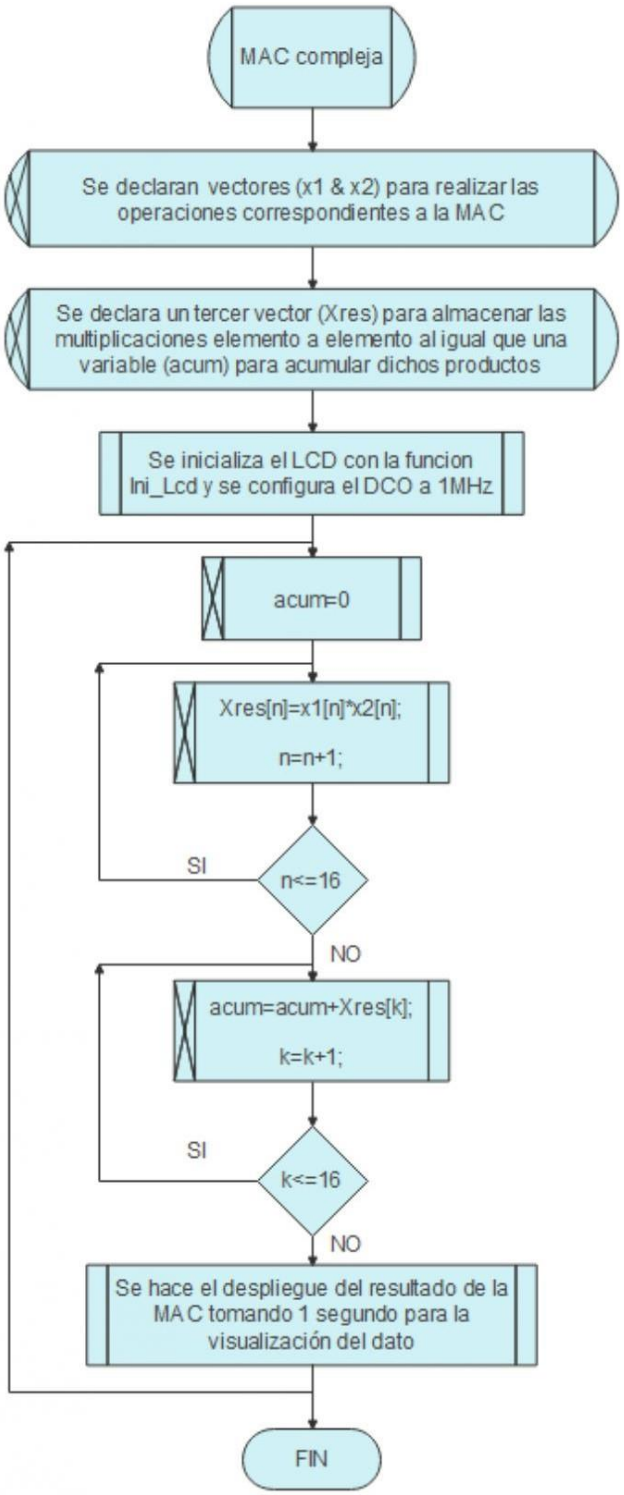


Figura 3-2: Diagrama de flujo para la programación de una MAC compleja.

Fuente: autoría propia.

La programación de la MAC compleja en el MSP430 se realizó de manera muy similar a la programación que se realizó en MATLAB, de echo también se utilizan 2 for en esencia para poder realizar La MAC compleja, entonces para poder crear la MAC compleja primero se agrega la librería llamada “complex.h” al entorno de programación, esta librería hace posible que podamos utilizar números complejos en nuestro programa, después de define un vector llamado X_1, el cual contiene 16 valores en formato flotante. Después de ello se coloca un for, el cual es un bucle externo definido como: for (char k=0 ; k<=15 ; k++), este bucle se encarga de realizar el barrido de las Ks, es decir, para cada valor de K se realiza todo el proceso para obtener una MAC. Después de ello se crean 2 acumuladores, los cuales se inicializan en cero para cada iteración de los distintos valores del vector inicial.

Después de esto se crea el segundo for (for interno), el cual se define como: for(char n=0; n<=15; n++), este ciclo se encarga de realizar la acumulación y los productos correspondientes del vector inicial. Después de esto se coloca acum=X1[n]*cexp(-I*2*3.141592*k*n/16), con esta expresión se realiza la MAC compleja con la exponencial compleja, posteriormente se realiza el cálculo del módulo de cada acumulación de productos y esos productos se almacenan en un vector para su posterior despliegue. Con ello se tendría la programación de una MAC compleja en el MSP430, ya solo al final se colocan 2 for para la impresión de resultados en la LCD.

Nota 3: el código de este programa se puede consultar de manera más detallada en el anexo 4.

En general se puede concluir que la MAC compleja necesita dos for para poderse llevar a cabo, donde uno de ellos es para ir haciendo la suma acumulada, el cual sería el for más interno, y el for externo simplemente es para ir calculando más valores de y, es decir, el for externo, ejecuta varias veces la MAC, ósea, cada vez que entra al for externo es como si entrara el reset, es decir, es como si se reiniciara, porque en esta parte se limpia al acumulador, por lo tanto se tienen dos for porque la MAC se ejecuta varias veces, pero en si la MAC solo necesita de un for, pero como se está tomando de ejemplo la transformada de Fourier, requiere echar a andar varias veces la MAC, por ello se coloca un for para poder reproducir varias veces la MAC, es por ello que hay dos for, entonces la explicación de ello es porque la MAC se ejecuta en varias ocasiones según el tamaño del vector que se defina.

3.2.3.-Validación de la MAC programada

La validación de la MAC compleja se hace comparando los resultados obtenidos con el programa realizado en MATLAB y los resultados ofrecidos por el programa realizado en el MSP430, estos dos programas fueron descritos anteriormente y se pueden revisar más a detalle en los anexos pertinentes a cada uno de ellos, lo que ahora se hará será exponer los resultados de cada uno de ellos y compararlos, si todo salió bien, los resultados obtenidos en MATLAB y en el IAR para el MSP430 deberían de ser prácticamente idénticos, entonces, como se sabe y como ya se mencionó anteriormente MATLAB muestra estos resultados en su misma interface de trabajo en la parte del comand window, pero el MSP430 no puede hacer lo mismo, entonces se había comentado anteriormente que los resultados se desplegarían por medio de la LCD y de echo el programa realizado para el diseño de la MAC compleja se realizó con base a este criterio, pero para fines más prácticos y que se tuviera una manera más sencilla de visualizarlos, dado que son 16 resultados y sería un poco tedioso mostrar 16 fotos con estos resultados contra una de MATLAB, que los muestra en forma de lista, entonces la solución a esto fue mostrar los datos obtenidos para el MSP430 de forma serial, para poder realizar una comunicación serial en esta tarjeta de desarrollo se realizó lo siguiente:

Para hacer uso del **emulador Tera Term** y sincronizarlo con la tarjeta MSP430G2553 LaunchPad de Texas Instruments con la cual se realizó la comunicación serial (UART), es muy importante mencionar que esta comunicación se debe llevar a cabo de manera asíncrona y además de ello se debe revisar que el microcontrolador, pic o tarjeta de trabajo con la que se esté trabajando cuente con el sistema de comunicación UART.

Para este caso, el microcontrolador MSP430G2553 si tiene el hardware para la comunicación UART, algunos micros no tienen esta facilidad y para emularlo hay que realizarlo por medio de software. Entonces para poder llevar a cabo una comunicación serial exitosa Se siguen los siguientes pasos:

1.- Descargar de https://osdn.net/projects/ttssh2/releases/ el programa de Tera Term, este es un emulador de terminal de software gratuito de código abierto compatible con el protocolo UTF-8. Este emulador es compatible con Windows 7,8, Windows 95/98 / ME, Windows NT / 2000 y posteriores.

2.-Es indispensable tener los controladores de la tarjeta MSP430G2553 LaunchPad actualizados, de no ser así hay que ingresar a la parte de administrador de dispositivos y actualizar los correspondientes a esta.

3.-En la tarjeta de desarrollo se deben de puentear las terminales de Tx y Rx de manera horizontal para que haya transmisión y recepción por parte del emulador y el puerto virtual de la tarjeta a través del puerto USB (ver figura 3-3).

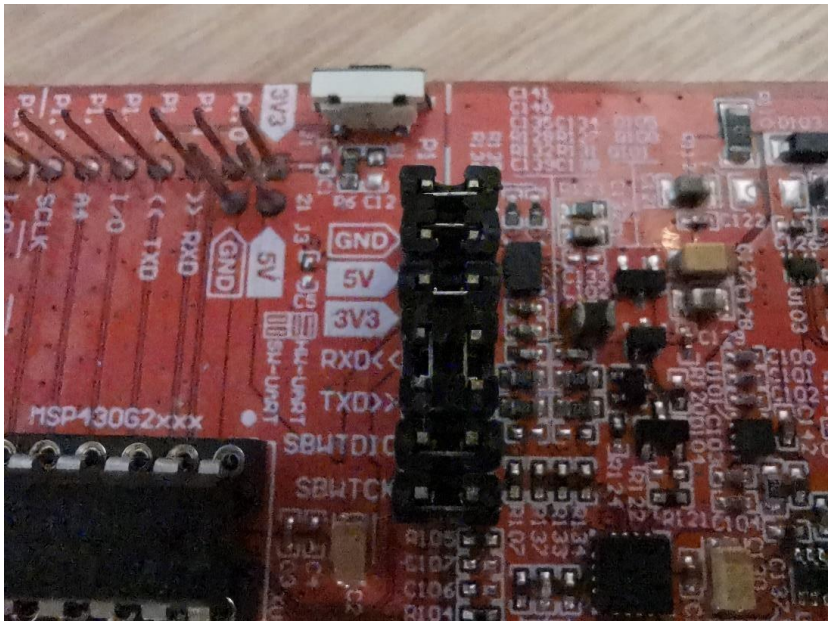


Figura 3-3: Puenteo de las terminales Tx y Rx de manera horizontal en la tarjeta de desarrollo MSP430G2553.

Fuente: autoría propia.

4.- Se configura la terminal P1.1 y P1.2 como transmisión y recepción para que de esta manera se pueda configurar el módulo secundario (UART).

5.-Para elegir la velocidad de transmisión hay que consultar el manual de usuario del MSP430G2553 para configurar los registros del UART, en este caso como se está trabajando con un reloj de 1[MHz] y una tasa de 9600[bauds(baudios)] se seleccionan los valores correspondientes para UCA0BR0 y UCA0BR1, los cuales se pueden observar en la figura 3-4 y están sombreados en amarillo.

Frecuencia [Hz]	Tasa de baudios [bauds]	UCBRx	UCBRSx	UCBRFx	Error máximo de TX [%]		Error máximo de RX [%]	
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
1,000,000	9600	104	1	0	-0.5	0.6	-0.9	1.2
1,000,000	19200	52	0	0	-1.8	0	-2.6	0.9
1,000,000	38400	26	0	0	-1.8	0	-3.6	1.8
1,000,000	56000	17	7	0	-4.8	0.8	-8.0	3.2
1,000,000	115200	8	6	0	-7.8	6.4	-9.7	16.1
1,000,000	128000	7	7	0	-10.4	6.4	-18.0	11.6
1,000,000	256000	3	7	0	-29.6	0	-43.6	5.2
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

Figura 3-4: seleccionando los valores de UCA0BR0 y UCA0BR1 en la tarjeta de desarrollo MSP430G2553.

Fuente: MSP430x2xx Family, User's Guide.

6.- Para la transmisión, se envía el dato por el buffer de transmisión y se testea una bandera (IFG2&UCA0TXIFG) para que mientras esta esté activada no envíen más datos y se pueda dar una colisión de datos por ello, en este caso se perderían los datos previos enviados.

7.- Para la configuración del Tera Term se debe conectar mediante USB el microcontrolador para que se habilite la opción de puerto serial en la interfaz, se selecciona el puerto COM para aplicación de UART, en este caso es el COM5 (ver figura 3-5).

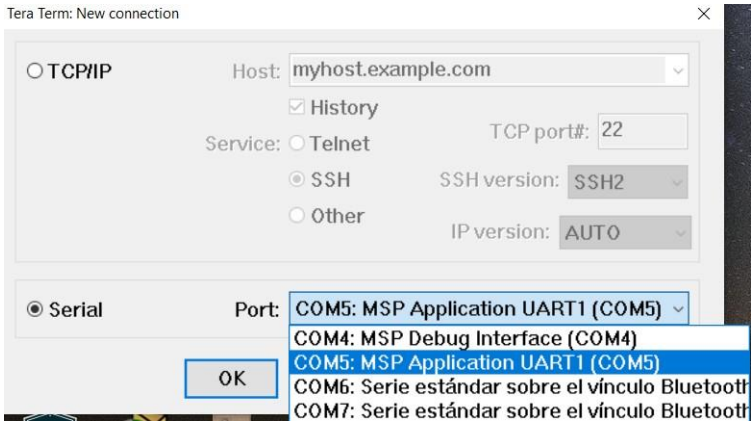


Figura 3-5: seleccionando el COM5 para poder establecer la comunicación serial en el MSP430G2553

Fuente: autoría propia.

8.-En las configuraciones del puerto serial se selecciona la tasa de transmisión, los bits de datos, la paridad y el control de flujo como se muestra en la figura 3-6.

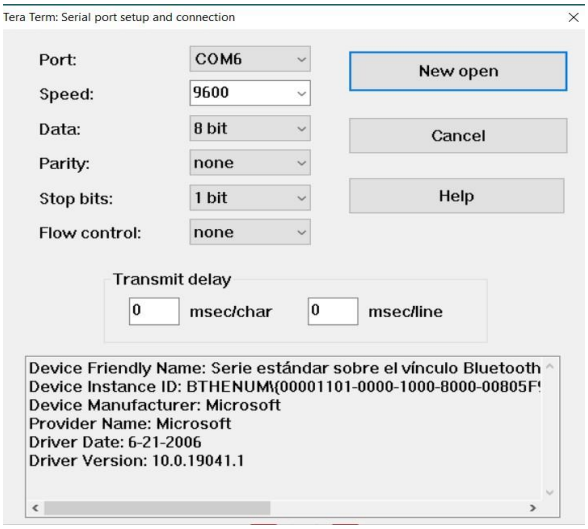


Figura 3-6: seleccionando las características necesarias para establecer la comunicación serial en el MSP430G2553
Fuente: autoría propia.

Con lo descrito anteriormente se podrá realizar una comunicación serial exitosa, entonces ahora mostraremos los resultados obtenidos en MATLAB (ver figura 3-7) y los resultados obtenidos por medio de la comunicación serial en el Tera term (ver figura 3-8).

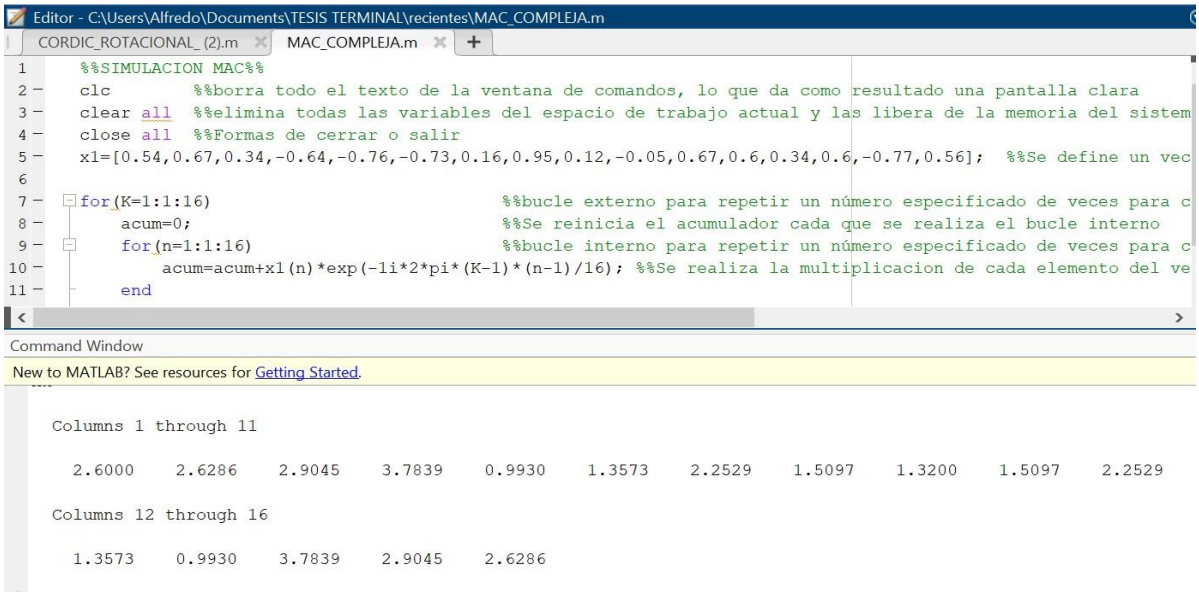


Figura 3-7: Mostrando los datos obtenidos por el programa hecho en MATLAB para la MAC compleja.

Fuente: autoría propia.

Figura 3-8: Mostrando los datos obtenidos por el programa hecho en el IAR para la MAC compleja en el MSP430G2553.

Fuente: autoría propia.

Con los resultados anteriores se puede concluir que todo lo que se realizó se hizo de forma adecuada ya que se puede observar que los resultados obtenidos tanto en MATLAB como en el IAR para el MSP430G2553 fueron prácticamente idénticos, es decir, con ello se puede validar la MAC compleja y decir que los procedimientos descritos anteriormente fueron correctos.

Entonces se puede decir que aquí se realizó una validación de la MAC compleja, más no se está llevando a cabo la aplicación de la MAC como se explicó en el capítulo 2, ya que como se sabe, esta tiene varias aplicaciones y aquí solamente se está validando que efectivamente acumula productos y realiza las funciones descritas en los capítulos anteriores.

3.3.- DESCRIPCIÓN DE LA PROBLEMÁTICA

La MAC compleja programada en el pic MSP430 fue posible realizarla porque este hardware cumple con los requerimientos necesarios para crear una MAC, los cuales se mencionaron anteriormente, y que básicamente es que el MSP430 permite realizar las funciones trigonométricas, las funciones modulo, las funciones tangente inversa y tangente cuadrada y eso es lo que se necesita para poder realizar la MAC y en este caso el MSP430 cuenta con ellas y eso hace que se pueda realizar la MAC sin ningún inconveniente. Pero, ¿Qué se necesita hacer cuando hay un pic que no cumple esos requerimientos?

Esta pregunta es muy acertada, y aquí se encuentra la problemática en sí, dado que varios de los microcontroladores y tarjetas de desarrollo de gama baja y media baja no cuentan con estas características en hardware mencionadas anteriormente, entonces no les sería posible poder crear una MAC. también se debe recordar que uno de los objetivos de esta investigación fue el poder proporcionar herramientas necesarias para que los alumnos con un poder adquisitivo bajo pudieran hacer aplicaciones de PDS, entonces , sabiendo que la mayoría de alumnos cuentan con pics sencillos de propósito general, la MAC que se creó en el MSP430 no la podrían reproducir los alumnos a menos de que compraran esta tarjeta de desarrollo y eso genera un gasto, entonces la problemática radica en este aspecto, es por ello que se buscara realizar una MAC considerando que nuestro dispositivo no cumple con todos los requerimientos necesarios para crearla y así poder abordar esta problemática empleando algunas herramientas matemáticas y algoritmos que nos permitan desarrollar una MAC funcional para los dispositivos que no cumplan con las características necesarias para poder crear una MAC.

3.3.1.-Problema a resolver

Primero se debe olvidar u omitir que existe un pic o un dispositivo hardware que cumple con todas las características necesarias para crear una MAC y debemos concentrarnos en el problema a resolver, el cual es que hay muchos pics o tarjetas de desarrollo de nivel bajo o medio que no cumplen con las características necesarias para crear una MAC, es decir, se debe contemplar que estos solo manejan los elementos de suma y multiplicación y que además muchos de ellos no manejan operaciones con números flotantes, es decir, manejan únicamente números enteros, en si eso es la problemática que se tiene, además de que estos pics son de uso muy común en las escuelas de ingeniería por su bajo costo y fácil acceso, además son los que se ocupan en la mayor parte del nivel medio superior y en la licenciatura también, los cuales son los pics de Microchip o Texas instruments y que estos en la gran mayoría son de 8 bits. Se debe volver a reiterar que el objetivo de esta investigación es programar una MAC sobre un pic de propósito general, los cuales son de fácil acceso y a un bajo costo, además de que como se mencionó anteriormente son los más comerciales y son los que la mayoría de estudiantes utiliza en las escuelas de nivel medio superior y superior, así que considerando esto, el problema a resolver seria el crear una MAC en un dispositivo de gama baja, el cual carezca de las características necesarias para crear la MAC, entonces se debe hacer uso de diversas herramientas, algoritmos y técnicas que nos permitan poder suplir a las funciones necesarias para crear la MAC y hacerla funcionar en un pic de propósito general.

3.3.2.- Solución propuesta

La solución propuesta para resolver la problemática planteada anteriormente es implementar una MAC basada en los principios comentados en el apartado 3.2, pero considerando que no se tienen las funciones trigonométricas, las funciones modulo, la función de tangente inversa, las operaciones con números flotantes, etc. Entonces para poder abordar esta problemática se ocupara el algoritmo llamado CORDIC, el cual nos va permitir generar o tener las funciones trigonométricas sin ocupar senos y cosenos, ya que como se mencionó en el capítulo dos, todas las funciones trigonométricas pueden calcularse o derivarse de funciones usando rotaciones vectoriales, es decir, la rotación vectorial también se puede utilizar para conversiones polares a rectangulares y rectangulares a polares, para la magnitud del vector y como un bloque de construcción en ciertas transformaciones como la DFT y la DCT. El algoritmo CORDIC (Coordinate Rotation Digital Computer) proporciona un método iterativo para realizar rotaciones vectoriales mediante ángulos arbitrarios utilizando solo cambios y adiciones. Los algoritmos CORDIC generalmente producen un bit adicional de precisión para cada iteración. Entonces, con ayuda de CORDIC se podrá reemplazar la ausencia de funciones

trigonométricas, ya que se generarán mediante las operaciones que realiza este algoritmo.

El otro problema es que el hardware no maneja números flotantes, este maneja únicamente números enteros y para ello se hará uso de la representación Qnm, con la cual por medio de diversas operaciones que se programaran en el hardware y siguiendo los principios de Qnm se podrán desarrollar operaciones con números flotantes, pero sin necesidad de que el hardware maneje esta característica, con ello este problema quedaría solucionado.

Y entonces en la solución propuesta en términos generales seria diseñar una MAC en el mismo pic MSP430, pero considerando que no se tienen las funciones que se requieren para poder crear una MAC, las cuales se mencionaron anteriormente, ósea que nada más se tienen las operaciones de números enteros y reales y que no hay números flotantes, entonces esa sería la solución, diseñar una MAC sobre un dispositivo que carezca de estos requerimientos y hacerla funcionar sin problema alguno.

Nota 5: el código del programa que lleva a cabo el algoritmo CORDIC se realizó en MATLAB, este es muy importante, ya que de él nos apoyamos para dar solución a la problemática planteada, este programa se puede consultar de manera más detallada en el anexo 5.

3.4.-PROPUESTA DE DISEÑO DE UNA MAC COMPLEJA

Ya con la suma, queda claro de cuantos elementos debe ser el resultado de salida cuando de hace una suma de n elementos, además aún falta la multiplicación. Basándose en una transformada de Fourier:

$$Y = \sum ab$$

En el cual la sumatoria es el acumulador y el multiplicador es el producto de “a” con “b”, donde “a” y “b” son números con representación Q_{nm}

Para saber el número de bits que se requieren para realizar la operación se toma la longitud del valor mayor que se esté utilizando para representar cualquier número y se le suma el número de acarreos que tenga el cual se obtiene de la siguiente operación:

$$Ceil(\log_2 S)$$

Donde:

S=Número de sumandos

Ceil=Redondeo al número entero inmediato superior

Un ejemplo si se tiene que cada "X" de la siguiente operación tiene una longitud de 2 bits:

$X_1+ X_2+ X_3+ X_4=Y$

Luego, $Ceil(\log_2 4)=2$

El cuatro es debido a que son cuatro sumandos, por lo que, si cada "X" tiene una longitud de 2 bits y quedan 2 bits de acarreo, se llega a la conclusión de que son necesarios 4 bits para representar el resultado de la operación, cabe aclarar que todas las operaciones se van a trabajar con variables enteras signadas, por ejemplo:

(3)+(-5)+(2)+(4) = 4

Ya que se necesitan 3 bits para representar el número 4 no quiere decir que el resultado sea necesariamente de 3 bits y el número de acarreos será de 2 por la operación $Ceil(\log_2 4)=2$, además se tiene un límite el cual dice que si tenemos "n" bits entonces el rango de representación para la operación es:

$\boldsymbol{rango}=-2^{n-1}\ a\ 2^{n-1}-1$

En este caso, $-2^{4-1}\ a\ 2^{4-1}-1 \rightarrow -8\ a\ 7$ y este es el rango de representación el cual claramente abarca hasta el número que más bits requiere siendo el -5. Ahora, ya que se necesitan 4 bits para representar el -5 y se tienen 2 acarreos, el número de bits necesarios para realizar esta operación es de 6 bits.

Ya que se hizo el producto hay que ir acumulandolo, las sumas serán en representación Q_{nm}, ahora tomando en cuenta la MAC:

$$Y = \sum_{i=0}^{a} a_i b_i$$

"Y" será del número de bits que dependa del número de sumas que habrá en la MAC y el número de acarreos que va a requerir en base a estas sumas, ya que se está utilizando el microcontrolador MSP30G2553, ahora, es un requerimiento el número de sumas que hará la MAC ya que es lo que determinará la cantidad de acarreos a considerar, en caso de requerir un número diferente de acarreos, solamente se cambia este mismo en el algoritmo.

En síntesis, para conocer el número de bits que se requieren para el resultado de salida de la MAC se deben tener en consideración los siguientes puntos:

1.-Cantidad de bits del sumando más grande.

2.-Número de bits de acarreo tomado de la cantidad de sumandos.

La MAC realizada en este caso realiza 16 sumas lo que quiere decir que el número de bits de acarreo será de 4 bits.

Ahora, el sumando más grande que se podrá tomar en cuenta se determina mediante el hecho de que los valores de "a" y "b" serán $Q_{1.5}$ por lo tanto se utiliza 1 bit para la parte entera y 5 bits para la parte fraccionaria. Dado que lo que la MAC está realizando es la suma de los productos de dos valores, los cuales son del tipo Q_{nm} y cuando se multiplican dos números Q_{nm} distintos, el resultado será un nuevo número Q_{nm} donde:

$$Q1n1.m1 * Q2n2.m2 = Q3n3.m3$$

De lo cual:

$n3=n1+n2$

$m3=m1+m2$

En la multiplicación de Q_{nm}, se suman las longitudes de los bits de "n" y de "m" y de esa longitud tendrá que ser el resultado también. Dado que para esta MAC se tienen números de 6 bits cada uno en representación $Q_{1.5}$, cada multiplicación dará como resultado un número $Q_{2.10}$, 2 bits para la parte entera y 10 bits para la parte fraccionaria.

Este es el diseño de una MAC fija donde no se puede cambiar la longitud máxima de bits disponibles para la representación de los resultados a la salida de la MAC, el diseño no es universal, ya que depende de la aplicación, pero como ya se mencionó antes si se puede configurar para poder adaptarla a la necesidad del usuario.

3.4.1.-Requerimientos

Para poder obtener una MAC funciónable y práctica para su uso en los microcontroladores de cualquier gama, la manera en que se van a representar "a" y "b" es en valores de 6 bits cada uno, para poder desplegar valores de números que realizan la representación Q_{nm} al cual se le asigna precisamente un bit a "n" que es la parte que representa los números enteros y 5 bits a "m" siendo la parte que representa a los números fraccionarios, además, la MAC es capaz de acumular hasta 16 productos, cantidad de productos ya previamente programada, con el fin de que se utilicen 4 bits de acarreo para la salida, estos debidos a los sumandos y ya que cada producto genera un nuevo valor en representación Q_{nm} donde las "n" de "a" se suman con las "n" de "b" y a su vez las "m" de "a" se suman con las "m" de "b", lo que es:

$$Q1_{n1.m1} * Q2_{n2.m2} = Q3_{n3.m3}$$

De lo cual:

$n3=n1+n2$

$m3=m1+m2$

$$Q1_{1.5} * Q2_{1.5} = Q3_{n3.m3}$$

Entonces:

$n3=1+1$

$m3=5+5$

$$Q3_{n3.m3} = Q3_{2.10}$$

Este nuevo valor generado por los productos es de 12 bits, teniendo 2 bits disponibles para la representación de los números enteros y 10 bits para la representación de los números fraccionarios. Ahora, si a estos 12 bits se le suman los bits de acarreo de las 16 sumas los cuales se obtienen de la siguiente operación previamente explicada:

Ceil($\log_2 16$)=4 bits de acarreo

Por lo tanto, la salida va a ser de 12 bits más el logaritmo base dos de 16 que son las sumas, esto da por resultado 4 más los 12 que ya se tenían, da un resultado de 16 bits que serán los disponibles para la representación del número de la salida de la MAC.

3.4.2.-Diagrama a bloques

Básicamente se retoma el último diagrama a bloques de la MAC, pero ahora con la diferencia de que se le agrega “a” y “b” en representación Qnm y el resto del diagrama de la MAC queda de la misma forma que el diseño original.

Además, como era de esperarse, ya que las operaciones se están realizando con números en representación Q_{nm}, la salida a su vez, también será un número representado en Q_{nm}, que esta mostrado en el diagrama de la MAC.

Como se puede apreciar en el diagrama de la MAC, el vector “a” de entrada para las operaciones tiene una longitud de 6 bits, el vector “b” de igual forma tiene una longitud de 6 bits, al pasar a la etapa en donde se realizan los productos de “a” con “b”, el resultado obtenido son valores en Q_{nm} que pueden llegar a ser de hasta 12 bits, lo siguiente es sumar producto tras producto e irlos acumulando, para este punto en el acumulador ya se tendrán valores en representación Q_{nm} de hasta 16 bits, lo que es lo mismo para la salida, un solo valor en representación Q_{nm} que puede ser de hasta 16 bits, esto por motivo de los 4 bits de acarreo ya mencionados antes a razón de los sumandos (ver figura 3-9).

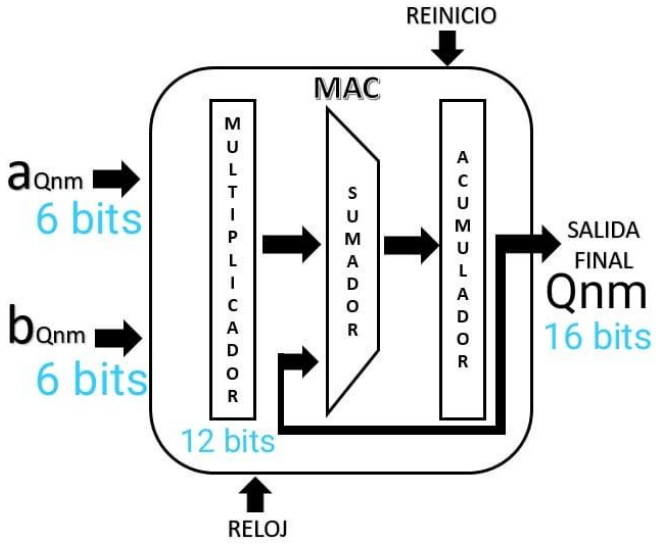


Figura 3-9: Diagrama a bloques de una MAC.

Fuente: autoría propia.

Ahora, lo siguiente es agregar un bloque llamado Cordico, el cual arroja un dato que será “b” o “a”, que se puede dar en un dato de forma polar o en forma rectangular, quiere decir que puede introducir parte real con parte compleja o modulo y ángulo, de donde se va a utilizar la forma rectangular que es introducir parte real e imaginaria a la MAC, en la entrada donde irá “a” es exactamente lo mismo, este se ocupará para cuando se vaya a generar un número complejo donde se necesite generar la siguiente expresión: $e^{j\theta_k}$

Normalmente el Cordico solamente se requiere para una entrada “a” o “b” porque uno de los datos es el que se quiere procesar y ya se conoce, ya está introducido, pero el otro puede estar cambiando y para lo que se utiliza Cordico.

Con esto “a” y ”b” se pueden predefinir los cuales, si son complejos expresados de forma rectangular, por lo que el resultado será un número complejo, debido a esto

se debe tener un bloque más de CORDIC a la salida de la MAC para representara la salida el resultado complejo (ver figura 3-10).

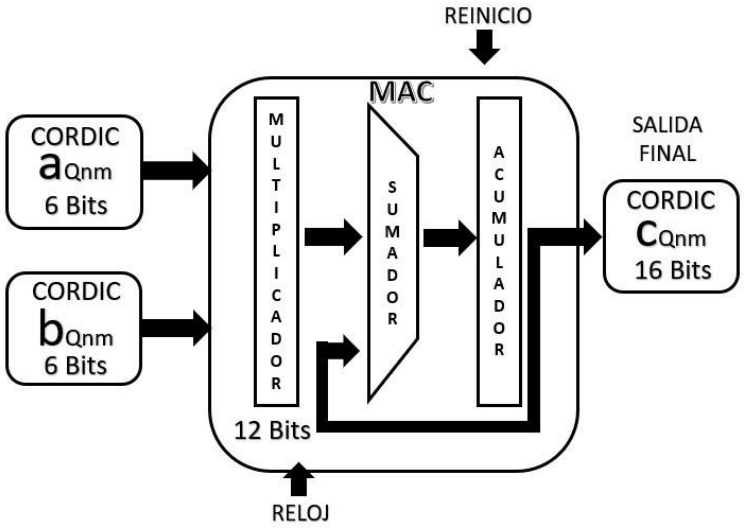


Figura 3-10: Diagrama a bloques de una MAC con Cordic.

Fuente: autoría propia.

3.4.3.-Especificaciones técnicas de diseño

Para comenzar con la descripción de estás especificaciones técnicas, primero se indica que los valores de "a" y "b" se toman extrayéndolos de un par de arreglos a los cuales uno corresponde al vector de "a" y el otro al vector de "b", estos arreglos ya están preprogramados, cada uno con sus respectivos valores.

Para determinar el máximo valor que se podría obtener en la MAC, se realiza lo siguiente, cuando cada valor de "a" y "b" tengan su valor de mayor exigencia de bits siendo -1, se estarán acumulando 16 multiplicaciones, por lo que, si cada una tiene una representación de -1 a 0.999 debido a:

$$\boldsymbol{rango}=-2^{n-1}\ a\ 2^{n-1}-\frac{1}{2^m}=-2^{1-1}\ a\ 2^{1-1}-\frac{1}{2^{10}}==-1\ a\ 0.999$$

Donde $1/2^m$, es la resolución que tiene esta representación y está en función de Cuantos bits se dedican para la parte fraccionaria.

Entonces, en el caso de que los 16 productos sean -1 multiplicado por -1, se obtiene un resultado de 1 y este resultado sumado las 16 veces que lo hace el acumulador

nos entrega como posible máximo valor de entrega a la salida de 16. Por consiguiente, un número $Q_{2.10}$ que es el resultado de los productos, utiliza 12 bits más los 4 bits de acarreo resultado de las 16 sumas, da como salida un valor que ocupe 16 bits, con el que se obtiene un número $Q_{6.10}$, el rango de representación que tiene esta variable $Q_{6.10}$ es la siguiente:

$$\boldsymbol{rango}=-2^{n-1}\ a\ 2^{n-1}-1=-2^{6-1}\ a\ 2^{6-1}-\frac{1}{2^{10}}==-32\ a\ 31.999$$

Valor que cabe perfectamente dentro del intervalo de representación Q_{nm} del diseño de la MAC con 16 sumandos, cada uno siendo un producto $Q_{2.10}$ de entre dos variables con valores $Q_{1.5}$.

Por último, cuando "a" o "b" sea llenado con la función llamada Cordic, el cual básicamente entrega las funciones trigonométricas, después se señala de que cantidad de bits serán estas magnitudes de "a" y "b" por lo que dependiendo de la cantidad de bits que se utilicen para la representación de las entradas a su vez también cambiara el número de bits que tendrá la salida.

A continuación, se muestra una gráfica realizada en MATLAB donde se aprecia el comportamiento de la parte de la redundancia que se controla con la cantidad de bits otorgados al valor de "m" en el número Q_{nm} contra el grado de error al representar el valor deseado con lo que el resultado es el que sigue (ver figura 3-11):

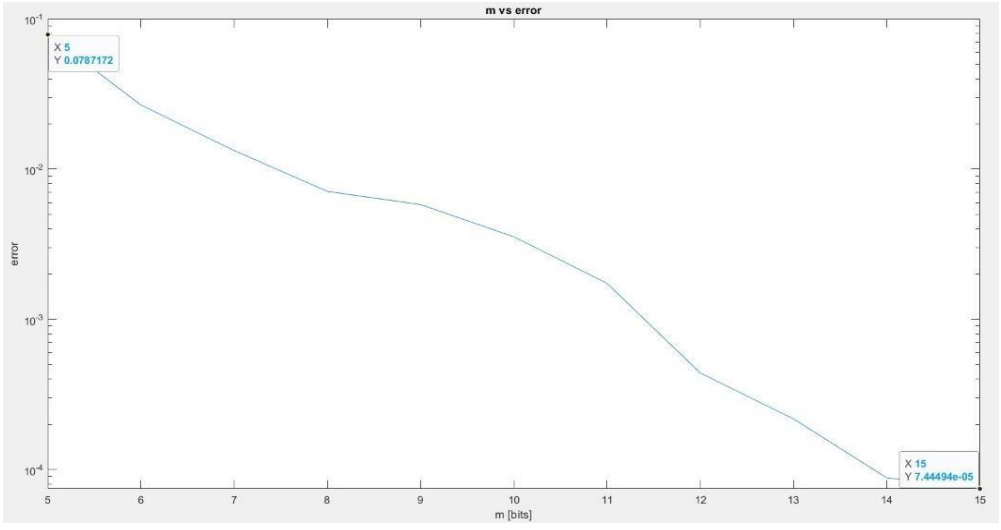


Figura 3-11: Grafica de "m" vs Error

Fuente: autoría propia.

Nota 6: El código del programa que lleva a cabo la gráfica de la magnitud del error debido al número de bits utilizados para la redundancia de la parte fraccionaria de los valores se encuentran en el anexo 6.

3.5 Validación del diseño

En un principio ya teniendo la MAC simulada, está se implementa en el MSP430G2553, pero ahora utilizando números flotantes.

Finalmente, luego de la realización de los respectivos códigos en MATLAB y además también en IAR con lenguaje C se validan, en ambos códigos han sido propuestos dos vectores que son la representación de "a" y "b", valores que se muestran a continuación.

Siguiendo con la validación, se realiza una comparación con la multiplicación que arroja el MSP430G2553 en flotante, luego de que se tiene el valor flotante se multiplica elemento a elemento "a" y "b" sin decodificar y se prosigue a sumar los productos y por último se decodifica.

Verificando que los resultados obtenidos del código del MSP430G2553 coincida con los resultados del código hecho en MATLAB (ver figura 3-12):

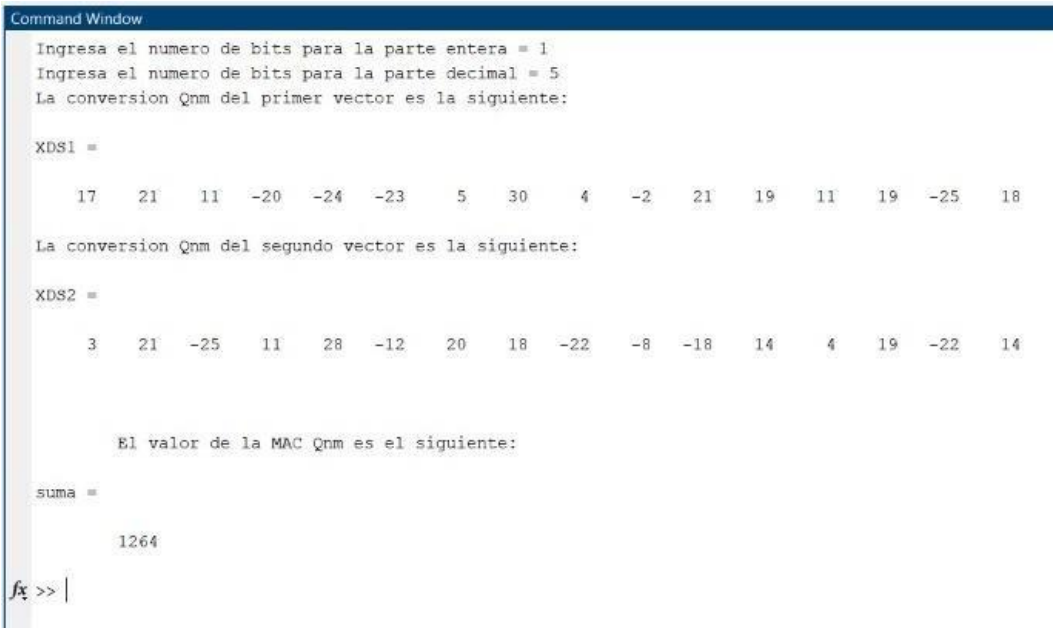


Figura 3-12: Resultados de la MAC en Matlab.

Fuente: autoría propia.

Ahora se muestra la figura donde se obtiene el mismo resultado en el MSP430G2553 desplegándolo en un LCD (ver figura 3-13):

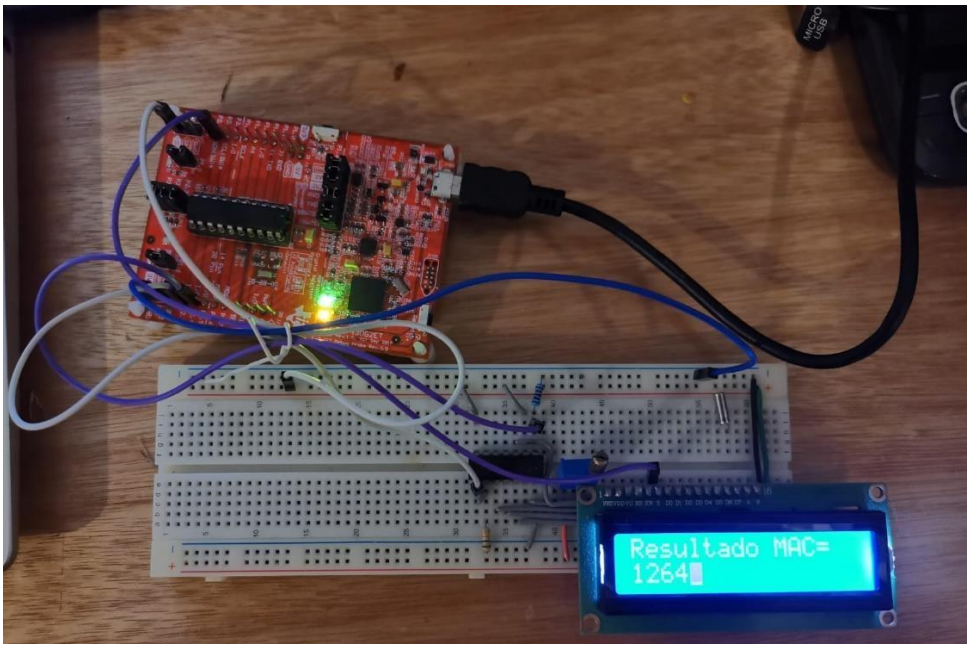


Figura 3-13: Resultados de la MAC en el MSP430G2553.

Fuente: autoría propia.

De igual manera se verifican los resultados al hacer que los vectores introducidos en la MAC sean mediante Cordic con lo que se obtienen los siguientes resultados, primero se muestran los obtenidos en MATLAB (ver figura 3-14):

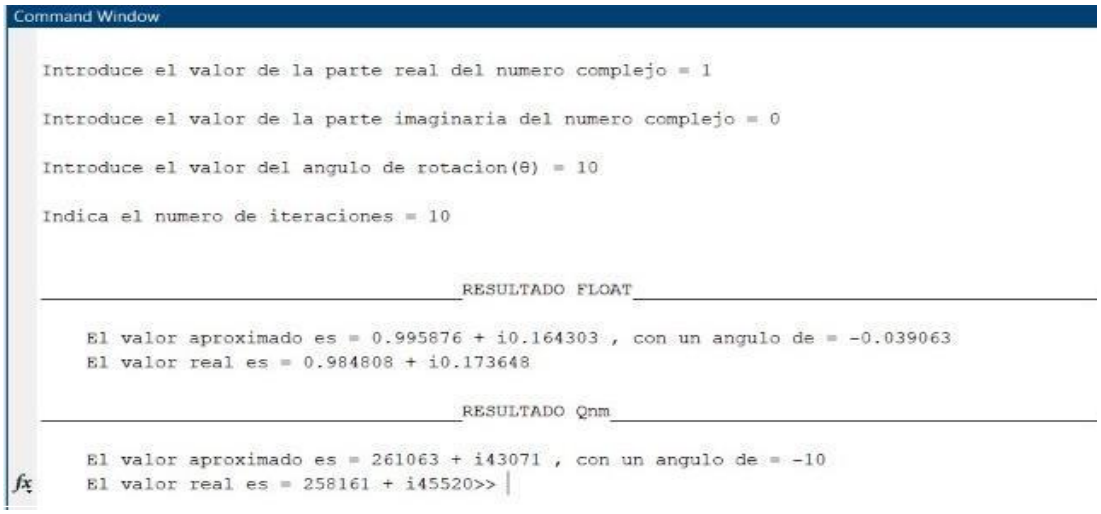


Figura 3-14: Resultados de la MAC utilizando el algoritmo Cordic en el MATLAB.

Fuente: autoría propia.

Nota 7: El código del programa que lleva a cabo la MAC con valores en representación Q_{nm} tanto en MATLAB como en IAR para ocupar el MSP430G2553 se pueden consultar de manera más detallada en el anexo 7.

A continuación, siguen los resultados de la MAC ahora también utilizando en algoritmo Cordic utilizando en MSP430G2553 siendo desplegados en un LCD (ver figura 3-15):

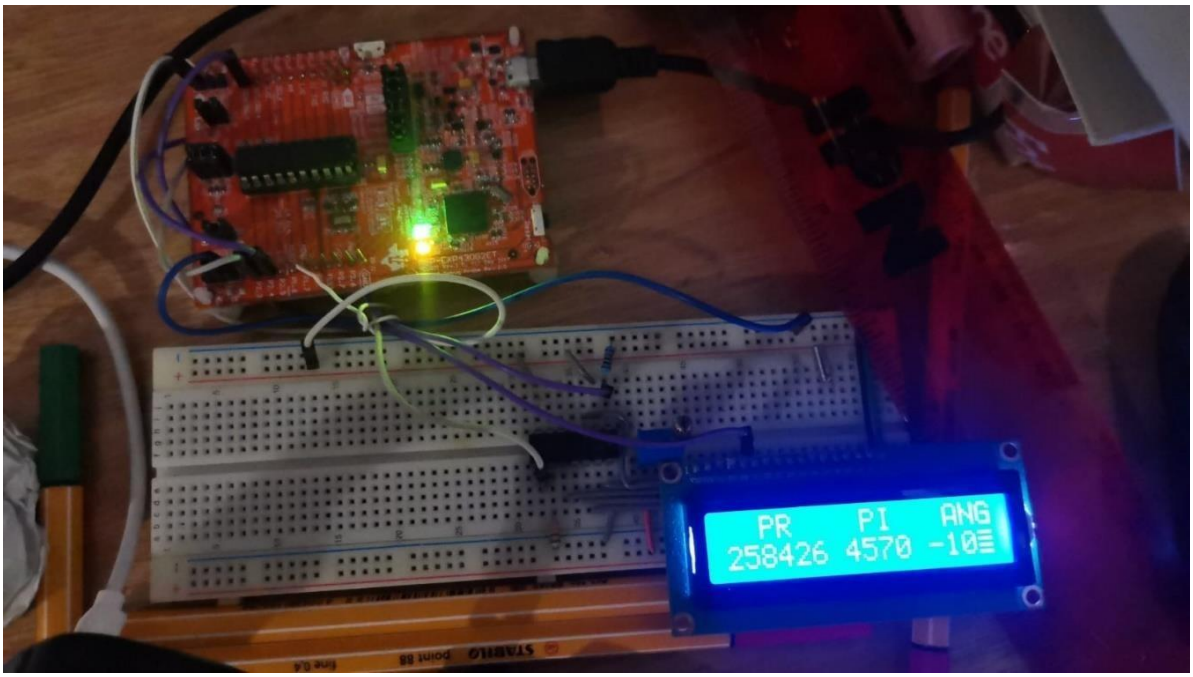


Figura 3-15: Resultados de la MAC utilizando el algoritmo Cordic en el MSP430G2553.

Fuente: autoría propia.

A su vez se adjunta de igual forma los resultados de la MAC utilizando Cordic a través de TERATERM (ver figura 3-16):

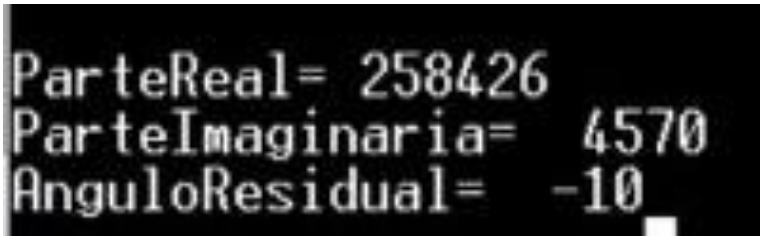


Figura 3-16: Resultados de la MAC utilizando el algoritmo Cordic en TERATERM.

Fuente: autoría propia.

Como se puede apreciar los resultados entre la MAC con Cordic en MATLAB no es exactamente el mismo que con el MSP430G2553, esto es debido a que los datos tipo int llegan hasta 32767 y el valor Q_{nm} de la parte real llega hasta 258426, por lo que para obtener el resultado deseado se debe usar datos tipo long int que son ya de 32 bits, con esto se obtienen ya los resultados deseados.

Todo esto se puede implementar en cualquier pic que maneje números enteros con signo ya que no se necesitan números fraccionarios para realizar las operaciones.

Una vez que se tenga está validación de diseño se logra tener una opción para tener un hardware portátil y cómodo de usar. Siendo un hardware que el alumno puede ocupar para hacer prácticas de PDS sin necesidad de ocupar el laboratorio y tener equipo especializado (el alumno podrá realizar prácticas desde su propia casa, no se necesita material extra más que el propio micro, sin necesidad de uso de osciloscopio ni un generador de funciones), con lo que lo único que el alumno tiene que apreciar son los resultados de la MAC. Al mismo tiempo puede servir para realizar operaciones como la convolución o transformada de Fourier y serie de Fourier y para desplegar los datos, se utiliza UART.

Nota 8: El código del programa que lleva a cabo la MAC con valores en representación Q_{nm} junto con el código utilizado para que las entradas fueran introducidas mediante Cordic, se realizaron en MATLAB y en IAR con lenguaje C, estos programas se pueden consultar de manera más detallada en el anexo 8.

3.6 Alcancé de diseño

En este apartado se plantea la siguiente interrogativa, ¿Qué pasa si se incrementa n? Hablando del tamaño o número de sumandos que se van a realizar en la MAC, o lo que quiere decir, si se incrementa, ¿Qué es lo que hay que cambiar en el diseño de la MAC?, para ello se toma en cuenta el número de bits que tiene en el acumulador el cual es de 16, dado que el número de sumandos o sumas determina cuantos bits se tendrán de acarreo, cosa que cambia el tamaño de la longitud de bits hábiles para el valor de salida de la MAC. Un ejemplo de forma que se hace más sencillo de entender:

En lugar de realizar 16 sumas de los productos de los vectores “a” con “b”, ahora se va a cambiar el número de sumas, siendo:

$$Y = \sum_{n=0}^{N=20} a_i b_i$$

Como se sabe, para conocer el número de bits que se requieren para realizar la operación se toma la longitud del valor mayor que se esté utilizando para representar cualquier número, en este caso se refiere al Q_{nm} (1.5) o $Q_{1.5}$ y se le suma el número de acarreos que tenga el cual se obtiene de la siguiente operación donde el argumento "S" es el número de sumas que se están realizando:

$Ceil(\log_2 S)$= Ceil(20)=4.321928

El Ceil es una función que permite redondear al número entero superior inmediato del resultado, por lo que el número de bits de acarreo que resultan es de 5 bits. Sumando los 5 bits de acarreo más los 12 bits que quedan de cada producto de "a" con "b", quedan 17 bits, de manera que permite tener un número Q_{nm} con un rango más amplio de representación o de ser necesario un valor de resolución más preciso, quedando como se muestra por ejemplo con una mejor resolución Q_{nm} (6.11) o Q6.11:

$$\boldsymbol{rango}=-2^{n-1}\ a\ 2^{n-1}-\frac{1}{2^m}=-2^{6-1}\ a\ 2^{6-1}-\frac{1}{2^{11}}==-32\ a\ 31.9995$$

Así que solo va a cambiar la resolución, pero no el hardware.

Otro aspecto para considerar es ¿Qué pasa si los vectores "a" y "b" mantienen los mismos bits, y se ocupa una representación diferente? Para dar respuesta a esta incógnita, se maneja otro ejemplo:

En lugar de una representación 1.5 de Q_{nm} o $Q_{1.5}$ para cada vector, ahora se usarán Q_{nm} (2,7) o $Q_{2.7}$, entonces ahora esto va a ocasionar que cambie el rango en el cual se van a representar los valores para Q_{nm} en la MAC:

$$\boldsymbol{rango}=-2^{n-1}\ a\ 2^{n-1}-\frac{1}{2^m}=-2^{2-1}\ a\ 2^{2-1}-\frac{1}{2^7}==-2\ a\ 1.999$$

El máximo valor que se podría obtener en la MAC, ahora cuando cada valor de "a" y "b" tengan su valor de mayor exigencia de bits será -2, se estarán acumulando 16 multiplicaciones, en el caso de que los 16 productos sean -2 multiplicado por -2, se obtiene un resultado de 4 y este resultado sumado las 16 veces que lo hace el acumulador nos entrega como posible máximo valor de entrega a la salida de 64.

Este producto de los nuevos "a" y "b", dejan como resultado un Q_{nm} (4,14) o $Q_{4.14}$, lo que es un número de hasta 18 bits, sumando los 4 bits de acarreo de las 16 sumas se obtiene un número Q_{nm} a la salida de 22 bits, permitiendo un rango más amplio de representación o una mejor resolución, como por ejemplo un $Q_{8.14}$:

$$\boldsymbol{rango}=-2^{n-1}\ a\ 2^{n-1}-\frac{1}{2^m}=-2^{8-1}\ a\ 2^{8-1}-\frac{1}{2^{14}}==-128\,a\,127.999939$$

Siendo suficiente para representar el máximo resultado que podría entregar está MAC que es 64, de forma que se podrá utilizar para otras aplicaciones donde se requieran valores más amplios de representación.

Por lo que, para que se tome en cuenta suponiendo que ahora se cambia la representación de “a” y “b” a mayor cantidad de bits, entonces se lleva a cabo el análisis de algunas de las dos maneras antes explicadas ya sea aumentando o reduciendo el valor de “N” haciendo referencia a la cantidad de sumas o también cambiando el número de bits correspondientes para la representación de cada vector y solamente se debe cuidar que el resultado este entre el límite de bits que se planteó para la realización de está MAC.

Por último, las plataformas o dispositivos en los que se puede implementar esta0 propuesta de diseño es en cualquier dispositivo que tenga la capacidad de hacer, sumas, multiplicaciones y que contenga variables que puedan cubrir la cantidad de bits necesaria para llevar a cabo esta MAC, siendo un buen ejemplo los MSP430G2553 o algunas de las familias de PIC´s.

El programa correspondiente para esta tesis se hizo sobre el mismo MSP430G2553, pero la diferencia entre el diseño de MAC hecho en el punto 3.4 que en el punto 3.2 del presente capítulo, es que en el punto 3.2 se tienen las funciones que pueden ser utilizadas por los programas para la representación de los números con fracciones o puntos decimales y en el punto 3.4 se considera que estás funciones no se tienen, por lo que, no se hace una multiplicación en flotantes, si no que se hace una multiplicación en números enteros, pero esos enteros tienen una representación en punto fijo que es el manejo de Q_{nm}.

CAPÍTULO IV: VALIDACIÓN DE LA MAC: PRUEBAS Y RESULTADOS.

En este capítulo se presentan las diversas pruebas que se realizaron para poder validar el diseño de la MAC, así como los resultados obtenidos de algunas comparaciones y pruebas importantes, en donde básicamente se realizan dos comparaciones, la primer comparación es entre Qnm MSP y Qnm MATLAB para cada prueba, con esto se realiza la validación de la MAC y luego es entre formato flotante y la representación fraccionaria de Qnm para MSP y MATLAB, donde con ello se hacen diversas pruebas y comprobaciones. Cabe destacar que para algunas pruebas es indispensable utilizar el algoritmo CORDIC para poder realizar la MAC, ya que también se presenta el escenario en donde para algunas pruebas se utilizan funciones trigonométricas y para otras el algoritmo CORDIC así como también se habla del alcance del proyecto y las aplicaciones futuras que se le pueden dar al mismo.

4.1.- ESCENARIO DE PRUEBAS

En este apartado se describe cómo se van a realizar las diversas pruebas para la validación y comprobación de la MAC a lo largo de este capítulo, así que se empezará por explicar de manera general como se van a realizar estas pruebas, así que lo primero que se va a hacer es proponer una MAC en formato flotante y posteriormente esa MAC en formato flotante se va a realizar con ayuda del software matemático llamado MATLAB. Esta MAC no es compleja, ya que básicamente es un producto punto entre los dos vectores, donde se proponen dos vectores de 16 elementos y se hace el producto entre ellos y con ello se obtiene un resultado en formato flotante, entonces lo que se va a hacer es:

Paso 1: se proponen dos vectores de 16 elementos flotantes y se realiza la MAC o el producto punto y se obtiene el resultado en formato flotante. Esto se puede observar de una manera más clara en la figura 4-1.

Figura 4-1: Diagrama a Bloques para realizar el paso 1.

Fuente: autoría propria

Paso 2: En este siguiente paso se convierten los números flotantes de cada vector a formato Qnm y se realiza la MAC en Qnm entre ellos, primero se realizará en MATLAB y luego en el MSP, después de ello se van a obtener dos resultados, se van a comparar y estos resultados deben de ser los mismos, esto se puede observar en la figura 4-2.

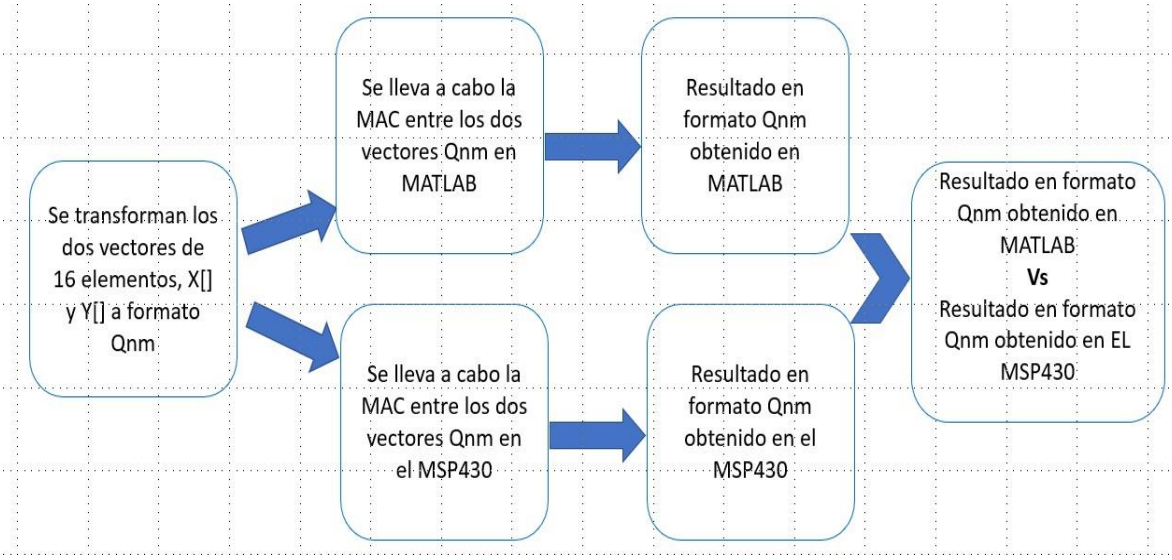


Figura 4-2: Diagrama a Bloques para realizar el paso 2 (validación de la MAC real).

Fuente: autoría propria

Algo importante que se debe de tomar en consideración es que los resultados que se obtienen en Qnm tanto en el MSP como en MATLAB son números enteros, pero la pregunta es: ¿cómo se interpretan estos resultados? , dando respuesta a este cuestionamiento es que primero se hace la representación Qnm correspondiente a cada número como ya se explicó en el capítulo 1, es decir, se transforma este número entero a su número equivalente en formato fraccionario y este se va a comparar con el resultado en formato flotante obtenido en el paso 1 (ver figura 4-3) , no se esperan resultados iguales al 100% pero si se esperan resultados que sean muy similares.

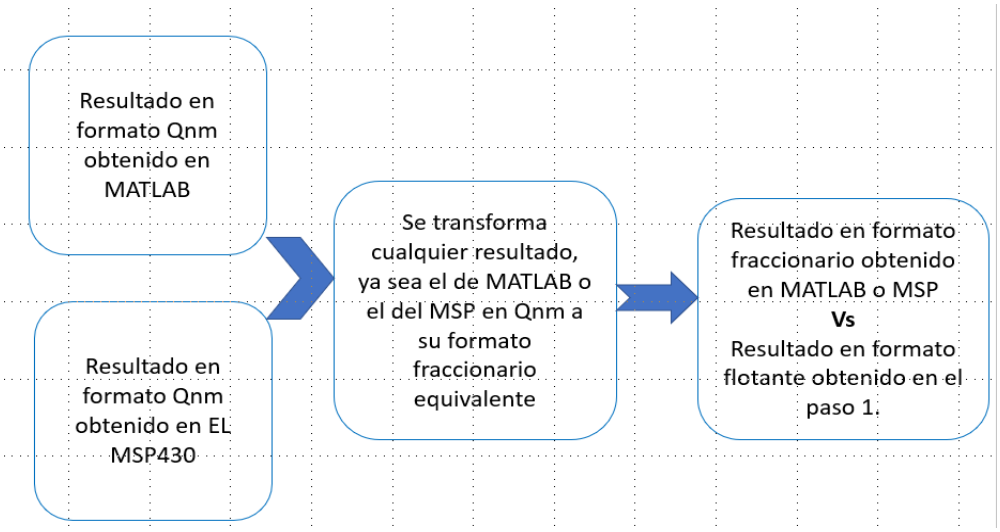


Figura 4-3: Diagrama a Bloques para realizar las pruebas de la MAC real.

Fuente: autoría propria

Después de eso se propondrá el mismo escenario, pero ahora se trabajarán con números complejos (MAC compleja).

Paso 1-complejo: se proponen dos vectores de 16 elementos complejos, solo que aquí cada vector de 16 elementos se va a dividir en 2 vectores de 8 elementos cada uno, ya que 8 elementos serán para la parte real y 8 elementos serán para la parte imaginaria del número complejo, después de ello se realiza la MAC o el producto punto entre los vectores complejos y se obtiene el resultado en formato complejo flotante. Esto se puede observar de una manera más clara en la figura 4-4.

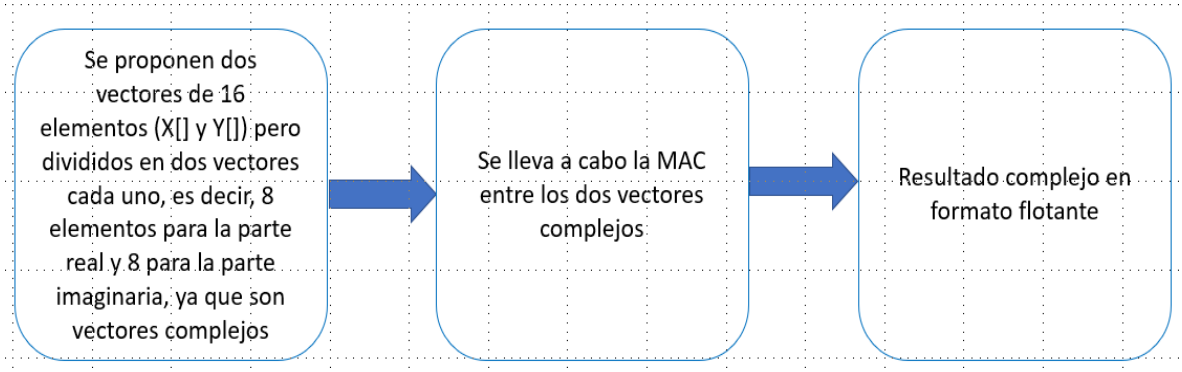


Figura 4-4: Diagrama a Bloques para realizar el paso 1-complejo.

Fuente: autoría propria

Paso 2-complejo: En este siguiente paso se convierten los números complejos de cada vector a formato Qnm y se realiza la MAC en Qnm entre ellos, primero se realizará en MATLAB y luego en el MSP, después de ello se van a obtener dos resultados en formato Qnm, se van a comparar, y estos dos resultados deben de ser los mismos, esto se puede observar en la figura 4-5.

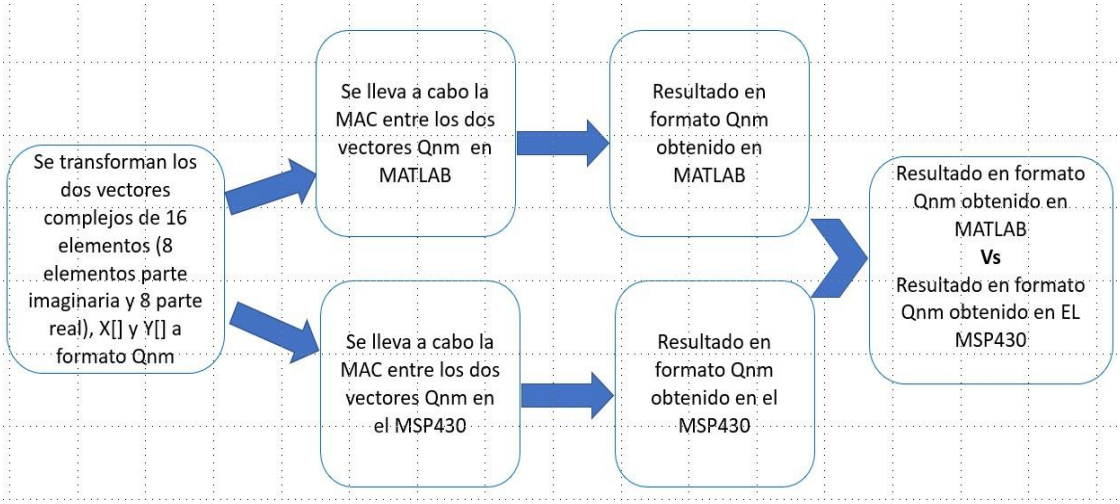


Figura 4-5: Diagrama a Bloques para realizar el paso 2-complejo (validación de la MAC compleja).

Algo importante que se debe de tomar en consideración es que los resultados que se obtienen en Qnm tanto en el MSP como en MATLAB son números enteros, pero la pregunta es: ¿cómo se interpretan estos resultados? , dando respuesta a este cuestionamiento es que primero se hace la representación Qnm correspondiente a cada número como ya se explicó en el capítulo 1, es decir, se transforma este número entero a su número equivalente en formato fraccionario y este se va a comparar con el resultado en formato flotante obtenido en el paso 1-complejo (ver figura 4-6) , no se esperan resultados iguales al 100% pero si se esperan resultados que sean muy similares.

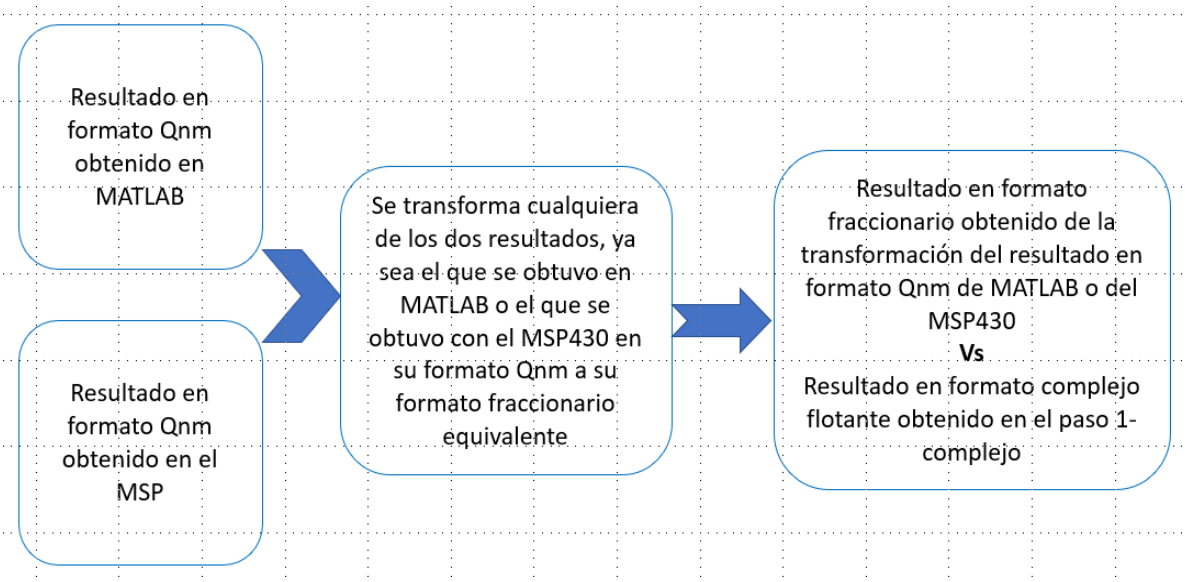


Figura 4-6: Diagrama a Bloques para realizar las pruebas de la MAC compleja.

Fuente: autoría propria

Después de realizar las pruebas citadas anteriormente ahora se procederá a realizar otra serie de pruebas donde también se tengan 2 vectores, solo que ahora uno será de valores complejos y otro de ángulos. Estas pruebas se describen a continuación:

Prueba 1: se proponen 2 vectores , uno de valores complejos y otro de ángulos, se realiza la MAC en formato flotante haciendo uso de las funciones trigonométricas para calcular el vector de ángulos, ya que se debe recordar que para cada valor de teta hay un número complejo denotado como $\cos(\theta)+j\text{sen}(\theta)$,entonces para cada valor de teta se tiene un número complejo y como ya se tiene el primer vector con los valores complejos se puede aplicar la MAC entre estos vectores sin ningún problema, esto se puede observar en la figura 4-7.

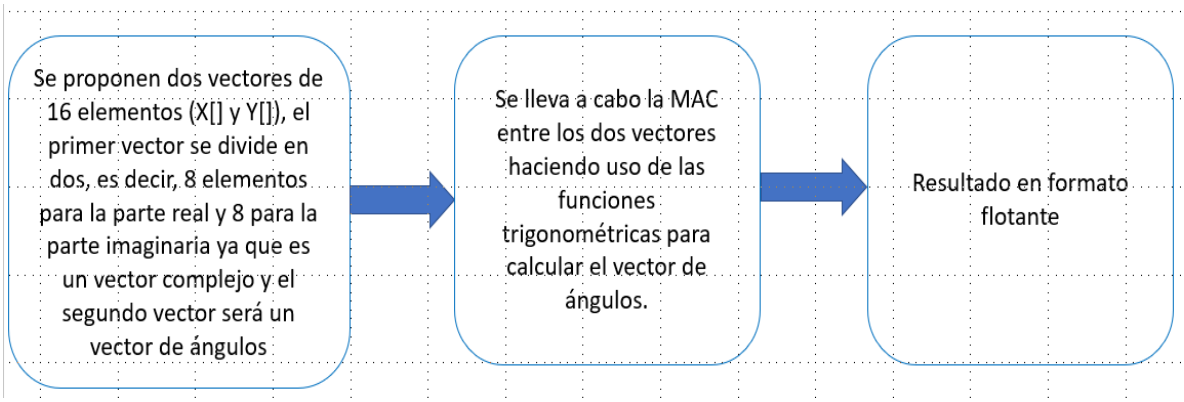


Figura 4-7: Diagrama a Bloques para realizar la prueba 1.

Fuente: autoría propria

Prueba 2: Para esta prueba se hace lo mismo que se hizo en la prueba 1, se propone un vector complejo X[] y un vector Y[] de ángulos, se realiza la MAC en formato flotante haciendo uso del algoritmo CORDIC para calcular el vector de ángulos, ya que se debe recordar que para cada valor de teta hay un número complejo denotado como $\cos(\theta)+j\text{sen}(\theta)$,entonces para cada valor de teta se tiene un número complejo y como ya se tiene el primer vector con los valores complejos se puede aplicar la MAC entre estos vectores sin ningún problema, esto se puede observar en la figura 4-8.

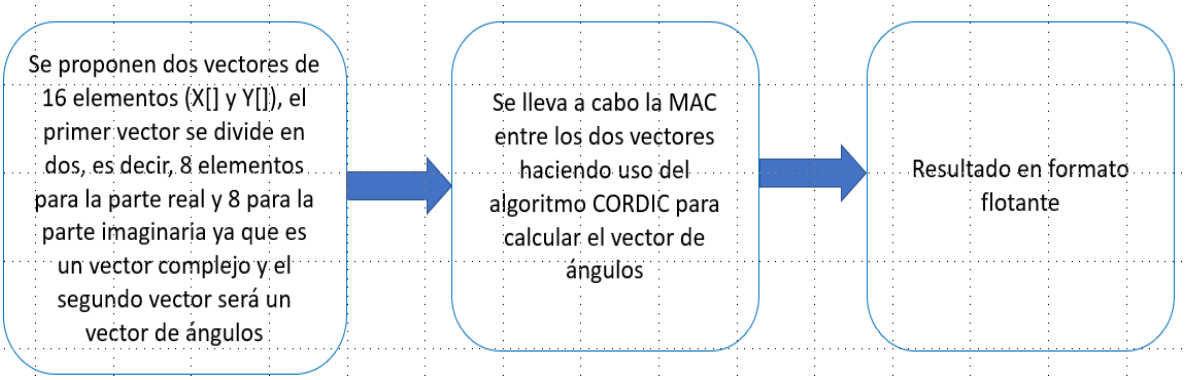


Figura 4-8: Diagrama a Bloques para realizar la prueba 2.

Fuente: autoría propria

Prueba 3: Ahora se transforman los dos vectores que se tenían anteriormente, el de valores complejos X[] y el de ángulos Y[] a formato Qnm y aplicando el algoritmo CORDIC, entonces cada valor de teta me va a dar un número complejo con el algoritmo CORDIC y como ya en los dos vectores se tienen números complejos ya

se puede realizar la MAC entre ellos y esta nos va a dar un resultado en formato Qnm, esta prueba se va a realizar en MATLAB, en la figura 4-9 se puede observar mejor este proceso con ayuda de un diagrama a bloques.

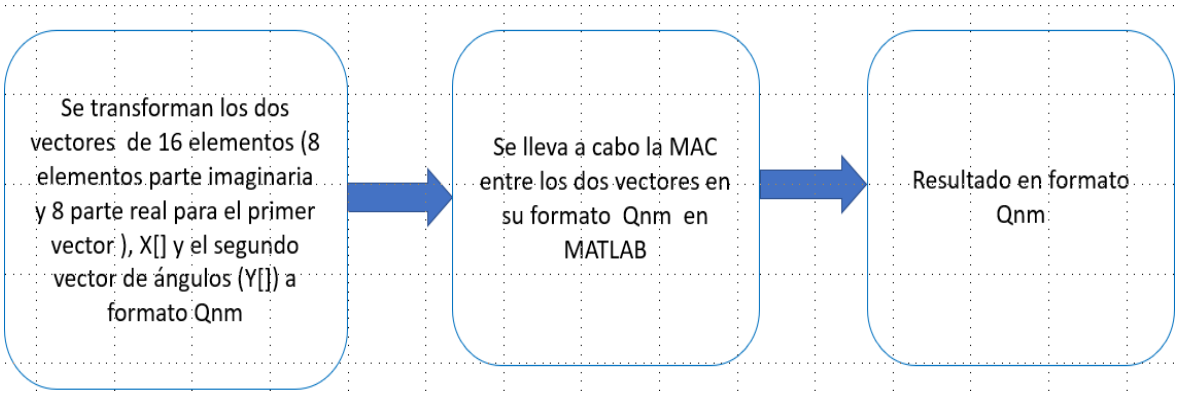


Figura 4-9: Diagrama a Bloques para realizar la prueba 3.

Fuente: autoría propria

Prueba 4: Para esta prueba se hace exactamente lo mismo que se hizo en la prueba 3, solo que ahora esta se realizará en el MSP430, después de obtener el resultado en el MSP430 se deben de comparar los dos resultados, es decir, el resultado de la prueba 3 con el resultado de la prueba 4 , aquí los resultados deben de ser iguales, esto se puede ver con mayor claridad en el diagrama a bloques para la prueba 4 (ver figura 4-10).

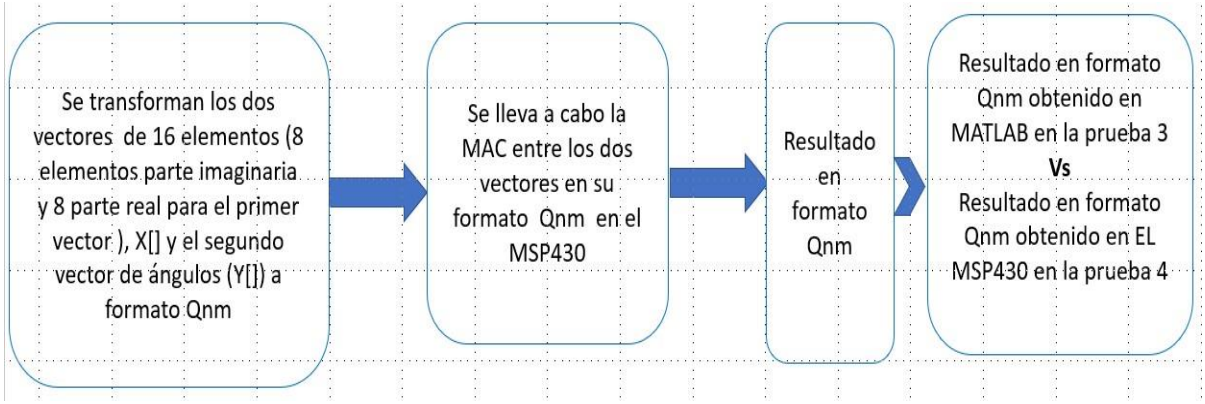


Figura 4-10: Diagrama a Bloques para realizar la prueba 4.

Fuente: autoría propria

Prueba 5: En esta última prueba hay que representar el resultado de la prueba3 y el resultado de la prueba 4 en su representación fraccionaria y compararlo con la versión flotante (prueba 1) y la versión cordico flotante (prueba 2). Estos no serán iguales en un 100% pero deben de ser similares, esto se puede observar mejor en el diagrama a bloques para esta prueba (figura 4-11).

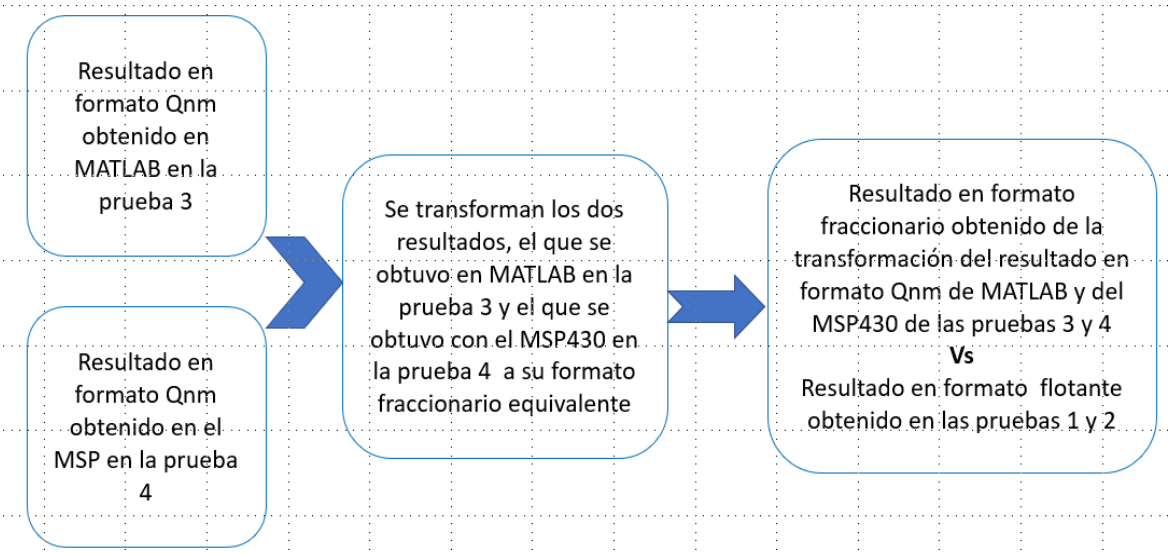


Figura 4-11: Diagrama a Bloques para realizar la prueba 5.

Fuente: autoría propria

4.2.- MAC REAL

Se proponen dos vectores de 16 elementos que sean reales en formato flotante, es decir, son dos vectores que no tienen parte imaginaria, los vectores que se propusieron para esta prueba se muestran a continuación, estos fueron utilizados tanto para flotante como para la representación Qnm:

x1=[0.54,0.67,0.34,-0.64,-0.76,-0.73,0.16,0.95,0.12,-0.05,0.67,0.6,0.34,0.6,-0.77,0.56**]**;

x2=[0.1,0.65,-0.78,0.35,0.89,-0.37,0.64,0.57,-0.69,-0.26,-0.57,0.43,0.12,0.58,-0.68,0.45**]**;

ya teniendo los dos vectores listos, entonces se procede a realizar la MAC o el producto punto y con ello se obtiene un resultado en formato flotante el cual se

puede observar en la figura 4-12, este resultado se obtuvo con la ayuda de MATLAB.

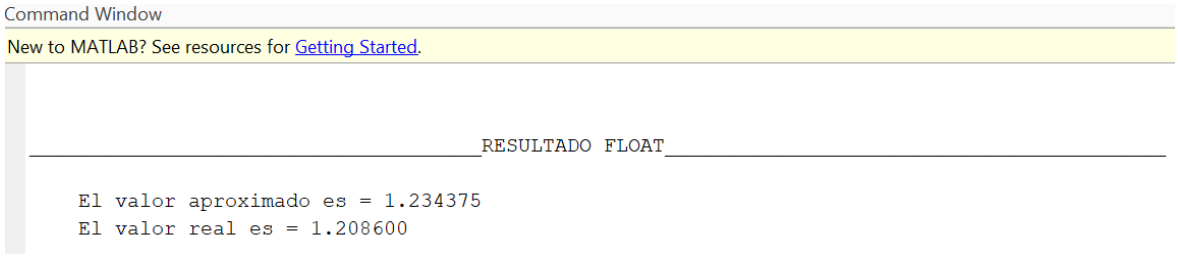


Figura 4-12: Resultado en formato flotante para la MAC real obtenido con MATLAB.

Fuente: autoría propia

Es importante aclarar que en la figura anterior se despliegan dos resultados, los cuales son el valor aproximado y el valor real, esto se debe a que se realiza la comparativa de los resultados en formato Qnm y decimal, donde el valor aproximado decimal corresponde a la decodificación Q6.10 de la MAC calculada en Qnm, lo cual se explicó a detalle en el capítulo 3, cabe mencionar que este resultado debe de ser idéntico en MATLAB y MSP, ya que se siguió el mismo procedimiento para calcularse, es decir, este resultado es el **valido** y por otro lado el valor real decimal se refiere a que se realizaron todas las operaciones en decimal solo para tener una comparativa en cuanto a eficiencia de métodos, es por ello que estos valores no son iguales en magnitud.

Nota 9: el código de este programa se puede consultar de manera más detallada en el anexo 7 en la primera parte donde se muestra el programa para la MAC real.

El siguiente paso es convertir los dos vectores propuestos con anterioridad a formatoQnm, esto lo podemos observar en la figura 4-13.

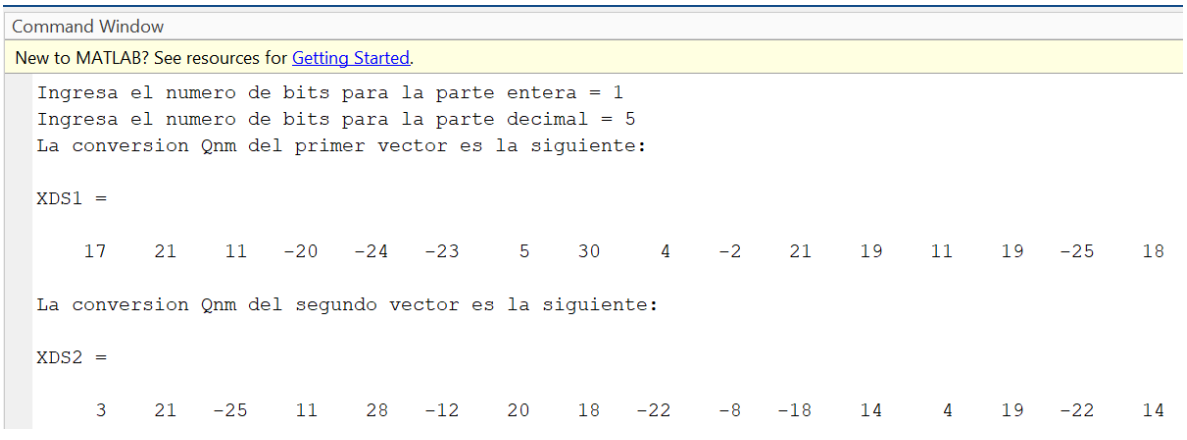


Figura 4-13: Transformación de los vectores iniciales a formato Qnm.

Fuente: autoría propria

Después de tener los dos vectores en representación Qnm se realiza la MAC en MATLAB y en el MSP con los dos vectores en Qnm y se comparan los resultados que se tengan de MATLAB y en el MSP tras aplicar la MAC, pero en formato Qnm, estos resultados se pueden ver en la figura 4-14 (resultado MAC real Qnm en MATLAB) y figura 4-15 (resultado MAC real Qnm en el MSP430).

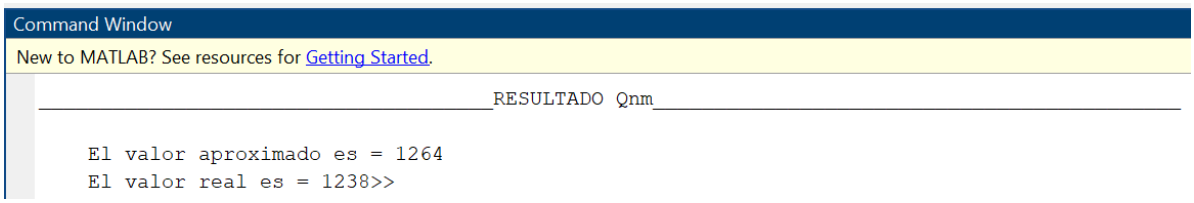


Figura 4-14: Resultado de la MAC real Qnm en MATLAB.

Fuente: autoría propria

Es importante aclarar que en la figura anterior se despliegan dos resultados, los cuales son el valor aproximado y el valor real, esto se debe a que para el despliegue del resultado Qnm el valor aproximado corresponde a realizar la codificación inicialmente de los vectores y realizar las operaciones con números enteros, es decir, el valor aproximado es el valor que nosotros vamos a tomar como **válido** mientras que el valor real es la codificación de la MAC realizada en decimal con un Q6.10 para una comparativa.

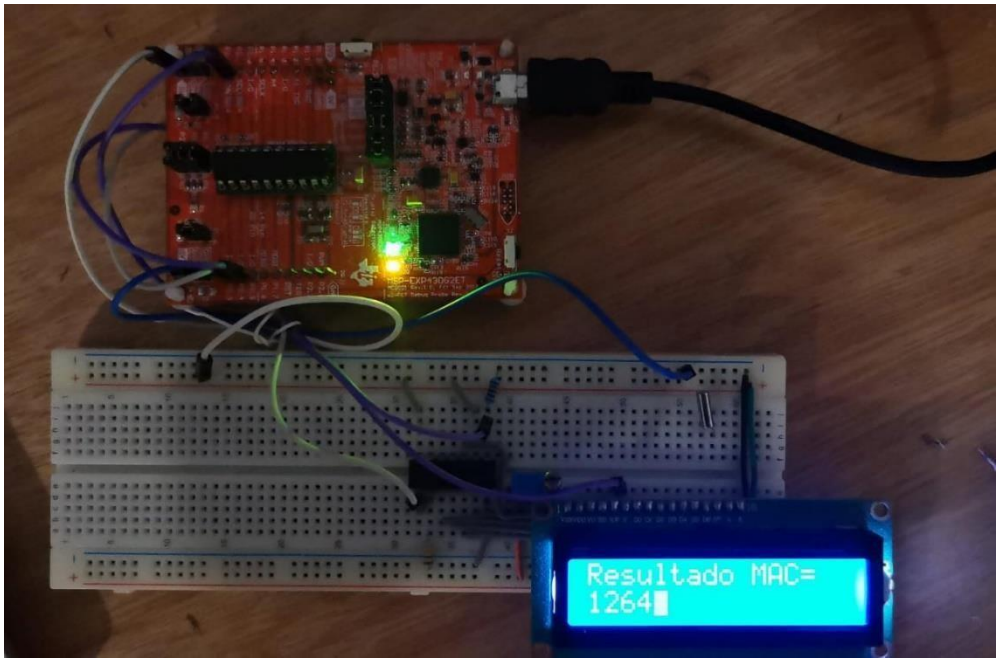


Figura 4-15: Resultado de la MAC real Qnm en el MSP430.

Fuente: autoría propia

Nota 10: el código de este programa se puede consultar de manera más detallada en el anexo 7 en la primera parte para la MAC en Qnm en MATLAB y en la segunda parte para la MAC real en Qnm en el MSP430.

Como se puede observar los resultados obtenidos en Qnm para la MAC en MATLAB y en el MSP430 son exactamente iguales. Con ello se realiza la validación de la MAC.

Por último la prueba que resta por hacer es que como se sabe los resultados obtenidos por MATLAB y el MSP430 son en formato Qnm, es decir, su representación es en números enteros, entonces se deben de interpretar estos datos y eso se hace representando el resultado ya sea del MSP430 o de MATLAB en su formato fraccionario posteriormente este se compara con el resultado en flotante que se obtuvo en la figura 4-12, no se espera que estos resultados sean iguales al 100%, pero sí que sean muy similares, para ello se realizó la conversión del resultado de la MAC Qnm obtenida del MSP430 a su formato fraccionario y este resultado se puede observar en la figura 4-16, en la figura 4-17 se colocara el resultado obtenido en formato flotante por la MAC real en MATLAB.

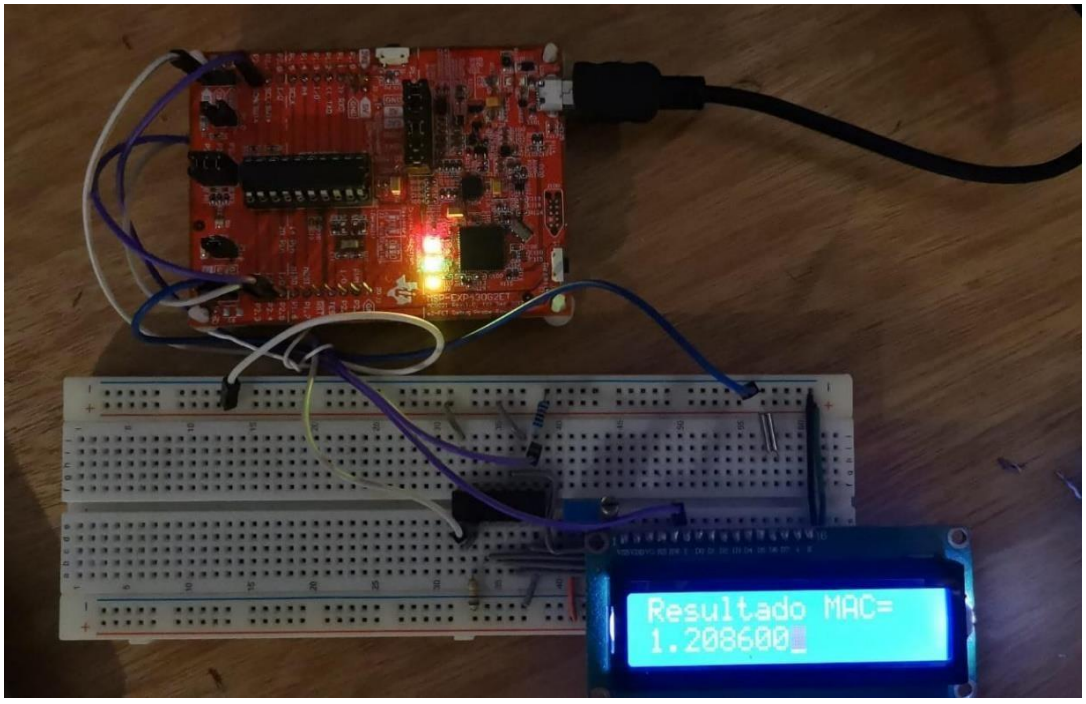


Figura 4-16: Resultado de la conversión del resultado en formato Qnm a formato fraccionario en el MSP430.

Fuente: autoría propria

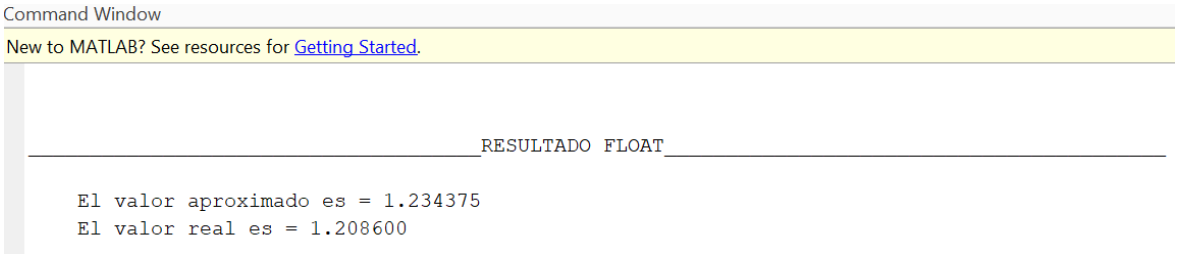


Figura 4-17: Resultado de la MAC real en formato flotante en MATLAB.

Fuente: autoría propria

Con esto se puede observar que se cumple lo que se mencionaba anteriormente, al representar al resultado obtenido para la MAC en Qnm ya sea de MATLAB o del MSP en formato fraccionario y compararlo con el resultado obtenido en flotante al aplicar la MAC real en MATLAB fueron resultados similares, pero no iguales.

Entonces se puede observar que la validación se está llevando a cabo en el Qnm cuando se hace la comparación entre los resultados del Qnm obtenidos en MATLAB y en el MSP430, ya que lo que sale en el MSP430 debe ser lo mismo que lo que sale en MATLAB y con eso estamos validando que se está implementando correctamente la MAC en el MSP430 porque ya nos dan los mismos resultados, eso sería la validación. Y las pruebas serian comparar el resultado que salió en Qnm y se transformó a número fraccionario con el resultado que se obtuvo en flotante que es lo que se hizo como prueba final en este subcapítulo.

4.3.- MAC COMPLEJA

Aquí se hace lo mismo que en el apartado 4.2, solo que se trabaja con números complejos, es decir, se proponen dos vectores de 16 elementos que sean complejos en formato flotante, vector complejo X[] y un vector complejo Y[], solo que aquí cada vector de 16 elementos se va a dividir en 2 vectores de 8 elementos cada uno, ya que 8 elementos serán para la parte real y 8 elementos serán para la parte imaginaria del número complejo, los vectores que se propusieron para esta prueba se muestran a continuación, estos fueron utilizados tanto para representación flotantecomo para la representación Qnm:

Números complejos utilizados en los vectores	
X	Y
$0.54 + 0.1j$	$0.12 - 0.69j$
$0.67 + 0.65j$	$-0.05 - 0.26j$
$0.34 - 0.78j$	$0.67 - 0.57j$
$-0.64 + 0.35j$	$0.6 + 0.43j$
$-0.76 + 0.89j$	$0.34 + 0.12j$
$-0.73 - 0.37j$	$0.6 + 0.58j$
$0.16 + 0.64j$	$-0.77 - 0.68j$
$0.95 + 0.57j$	$0.56 + 0.45j$

Para trabajarlos en MATLAB se definieron de la siguiente manera:

XR=[0.54 0.67 0.34 -0.64 -0.76 -0.73 0.16 0.95**];** →Se define un primer vector que almacena la parte real del vector X para realizar la MAC compleja en formato Qnm y float

XI=[0.1 0.65 -0.78 0.35 0.89 -0.37 0.64 0.57**];**→Se define un segundo vector que almacena la parte imaginaria del vector X para realizar la MAC compleja en formato Qnm y float

YR=[0.12 -0.05 0.67 0.6 0.34 0.6 -0.77 0.56**];**→Se define un tercer vector que almacena la parte real del vector Y para realizar la MAC compleja en formato Qnm y float

YI=[-0.69 -0.26 -0.57 0.43 0.12 0.58 -0.68 0.45]; →Se define un cuarto vector que almacena la parte imaginaria del vector Y para realiza r la MAC compleja en formato Qnm y float.

Ya teniendo los dos vectores listos, que en este caso para MATLAB se definieron 4 por lo explicado anteriormente, entonces se procede a realizar la MAC compleja o el producto punto y con ello se obtiene un resultado en formato flotante el cual se puede observar en la figura 4-18, este resultado se obtuvo con la ayuda de MATLAB.

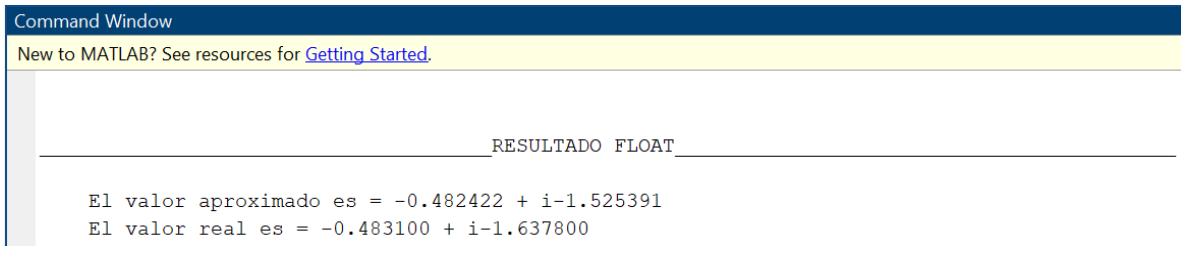


Figura 4-18: Resultado en formato flotante para la MAC compleja obtenido con MATLAB.

Fuente: autoría propria

Es importante aclarar que en la figura anterior se despliegan dos resultados, los cuales son el valor aproximado y el valor real, esto se debe a que se realiza la comparativa de los resultados en formato Qnm y decimal, donde el valor aproximado decimal corresponde a la decodificación Q6.10 de la MAC compleja calculada en Qnm, lo cual se explicó a detalle en el capítulo 3, cabe mencionar que este resultado debe de ser idéntico en MATLAB y MSP, ya que se siguió el mismo procedimiento para calcularse, es decir, este resultado es el **valido** y por otro lado el valor real decimal se refiere a que se realizaron todas las operaciones en decimal solo para tener una comparativa en cuanto a eficiencia de métodos, es por ello que estos valores no son iguales en magnitud.

Nota 11: el código de este programa se puede consultar de manera más detallada en el anexo 9 donde se muestra la MAC compleja en MATLAB.

El siguiente paso es convertir los dos vectores complejos propuestos con anterioridad a formato Qnm, esto lo podemos observar en la figura 4-19.

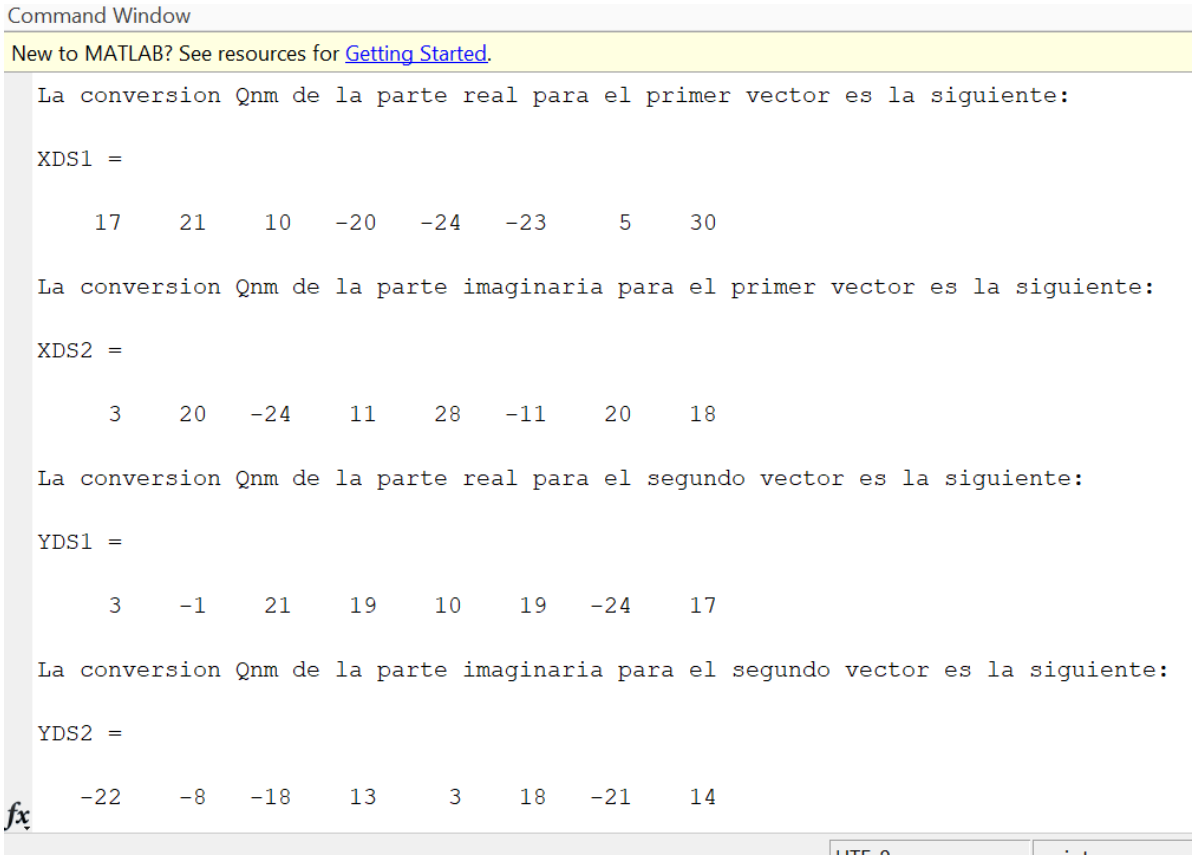


Figura 4-19: Transformación de los vectores iniciales a formato Qnm.

Fuente: autoría propria

Después de tener los dos vectores en representación Qnm se realiza la MAC compleja en MATLAB y en el MSP430 con los dos vectores en formato Qnm y se comparan los resultados que se tengan de MATLAB y en el MSP tras aplicar la MAC compleja, pero en formato Qnm, estos resultados se pueden ver en la figura 4-20 (resultado MAC compleja Qnm en MATLAB) y figura 4-21 (resultado MAC compleja Qnm en el MSP430).

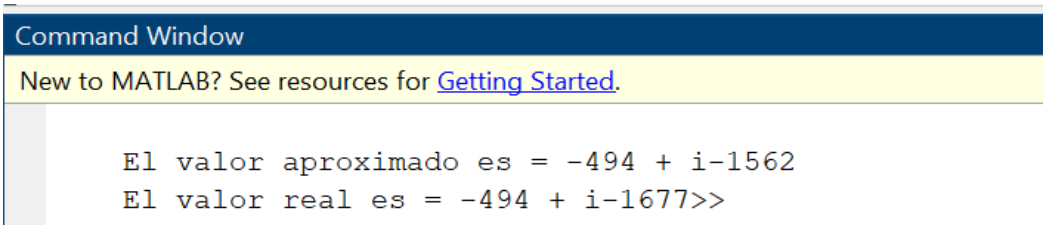


Figura 4-20: Resultado de la MAC compleja Qnm en MATLAB.

Fuente: autoría propria

Es importante aclarar que en la figura anterior se despliegan dos resultados, los cuales son el valor aproximado y el valor real, esto se debe a que para el despliegue del resultado Qnm el valor aproximado corresponde a realizar la codificación inicialmente de los vectores y realizar las operaciones con números enteros, es decir, el valor aproximado es el valor que nosotros vamos a tomar como **válido** mientras que el valor real es la codificación de la MAC compleja realizada en decimal con un Q6.10 para una comparativa.

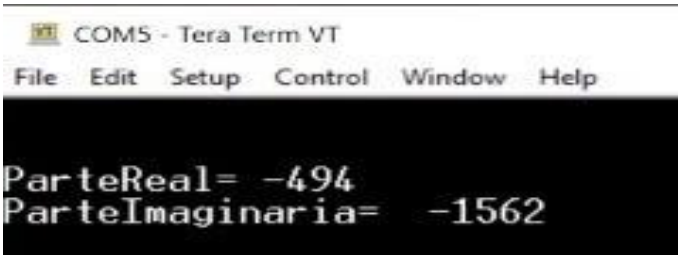


Figura 4-21: Resultado MAC compleja Qnm en el MSP430.

Fuente: autoría propria

Nota 12: El código de este programa se puede consultar de manera más detallada en el anexo 10 donde está el código para la MAC compleja en formato Qnm en el MSP430.

Como se puede observar los resultados obtenidos en Qnm para la MAC compleja en MATLAB y en el MSP430 son exactamente iguales. Con ello se realiza la validación de la MAC compleja.

Por último la prueba que resta por hacer es que como se sabe los resultados obtenidos por MATLAB y el MSP430 son en formato Qnm, es decir, su representación es en números enteros, entonces se deben de interpretar estos datos y eso se hace representando el resultado ya sea del MSP430 o de MATLAB en su formato fraccionario posteriormente este se compara con el resultado en flotante que se obtuvo en la figura 4-18, no se espera que estos resultados sean iguales al 100%, pero sí que sean muy similares, para ello se realizó la conversión del resultado de la MAC compleja Qnm obtenida de MATLAB a su formato fraccionario y este resultado se puede observar en la figura 4-22, en la figura 4-23 se colocara el resultado obtenido en formato flotante por la MAC compleja en MATLAB.

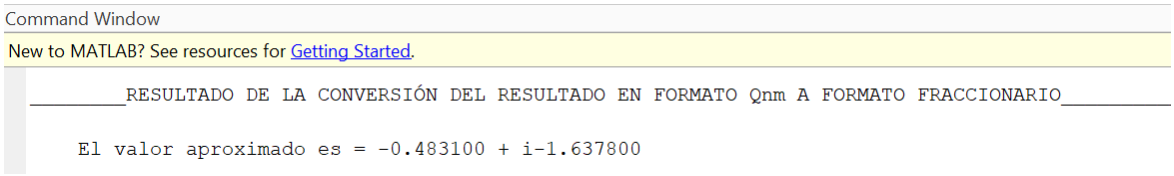


Figura 4-22: Resultado de la conversión del resultado en formato Qnm a formato fraccionario en MATLAB.

Fuente: autoría propria

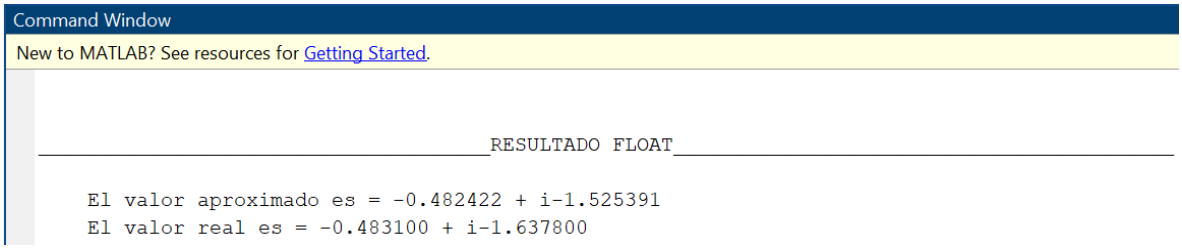


Figura 4-23: Resultado de la MAC compleja en formato flotante.

Fuente: autoría propria

Con esto se puede observar que se cumple lo que se mencionó anteriormente, al representar al resultado obtenido para la MAC compleja en Qnm ya sea de MATLAB o del MSP en formato fraccionario y compararlo con el resultado obtenido en flotante al aplicar la MAC compleja en MATLAB fueron resultados similares, pero no iguales.

Entonces se puede observar que la validación se está llevando a cabo en el Qnm cuando se hace la comparación entre los resultados del Qnm obtenidos en MATLAB y en el MSP430 para la MAC compleja, ya que lo que sale en el MSP430 debe ser lo mismo que lo que sale en MATLAB y con eso estamos validando que

se está implementando correctamente la MAC compleja en el MSP430 porque ya nos dan los mismos resultados, eso sería la validación. Y las pruebas serian comparar el resultado que salió en Qnm y se transformó a número fraccionario con el resultado que se obtuvo en flotante que es lo que se hizo como prueba final en este subcapítulo.

4.4.- MAC CON ALGORITMO CORDIC

Ahora se propone un vector complejo junto con un vector de ángulos, porque cada ángulo va ingresado en Cordico lo que hace que cada ángulo arroje un valor de Cordico, por lo que en Matlab se propone para cada teta **(θ)** un número complejo el cual está compuesto por **cos(θ)+jsen(θ)**:

$$\sum x_{Q_{nm}} y_{Q_{nm}}$$

$x_{Q_{nm}}$

$y_{Q_{nm}}$

$x = [x_0\ x_1\ x_2\ x_3]$
$y = [\theta_0\ \theta_1\ \theta_2\ \theta_3]$

$\sum xy = 12$

Una vez obtenido esto se realizaron diferentes pruebas que son las que se muestran a continuación:

Prueba No. 1:

Proponiendo los vectores “x” y “y”, para lo que los vectores “x” son números complejos y “y” son ángulos compuestos de **cos(θ)+jsen(θ)**, calculado en flotante de los cuales los resultados son mostrados en la figura 4-24:

$x = [x_0\ x_1\ x_2\ ]$
$y = [\theta_0\ \theta_1\ \theta_2\ ]$

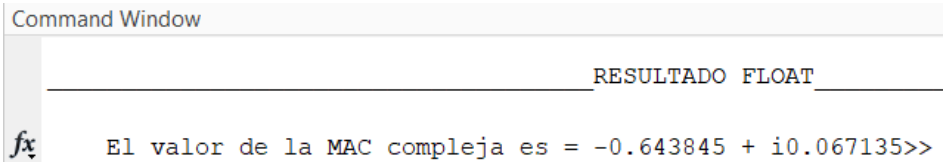


Figura 4-24: Resultado de la MAC compleja con funciones trigonométricas cos(θ)+jsen(θ) en formato flotante.

Fuente: autoría propria

Nota 13: El código de este programa se puede consultar de manera más detallada en el anexo 11 donde está el código para la MAC compleja con funciones trigonométricas en formato flotante.

Prueba No. 2:

Proponiendo los vectores "x" y "y", para lo que los vectores "x" son números complejos y "y" son ángulos, pero ahora el vector "y" ya no se calcula con las funciones trigonométricas **cos(θ)+jsen(θ)**, sino que ahora se calcula con Cordico en Matlab en flotante para los cuales los resultados son mostrados en la figura 4-25:

$x = [x_0\ x_1\ x_2\ ]$
$y = [\theta_0\ \theta_1\ \theta_2\ ]$

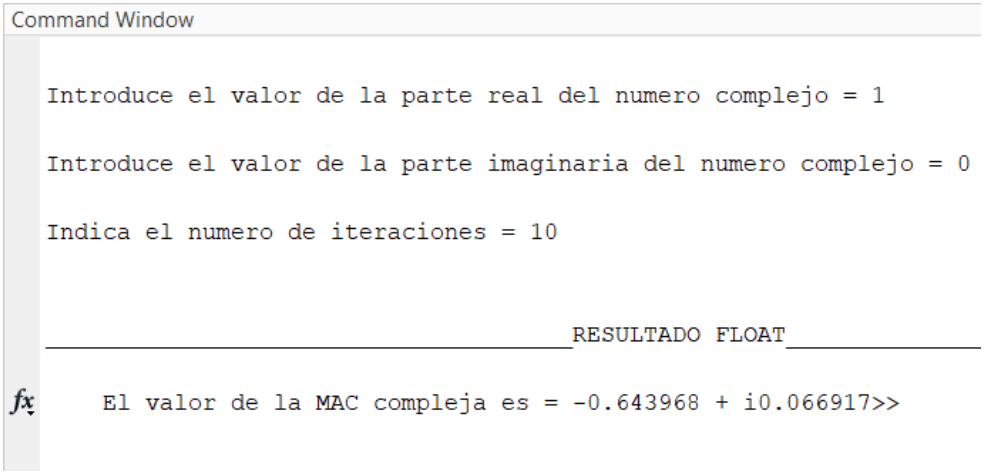


Figura 4-25: Resultado de la MAC compleja con algoritmo Cordic en formato flotante.

Fuente: autoría propria

Nota 14: El código de este programa se puede consultar de manera más detallada en el anexo 12 donde está el código para la MAC compleja con el algoritmo Cordic en formato flotante.

Prueba No. 3:

Ahora es probar los vectores "x" que sigue siendo de números complejos solo que ahora será en representación Q_{nm} y además el vector "y" siendo ángulos aplicando Cordico y en representación Q_{nm}, cada valor de θ dará como resultado un número

complejo con el algoritmo Cordico, al tener ya todos los números complejos ya se puede realizar la MAC lo cual da un resultado siendo realizado en Matlab que es mostrado en la figura 4-26:

$x = [x_0\ x_1\ x_2\ ]$ Q_{nm}
$y = [\theta_0\ \theta_1\ \theta_2\ ]$ Q_{nm} Cordic en Matlab

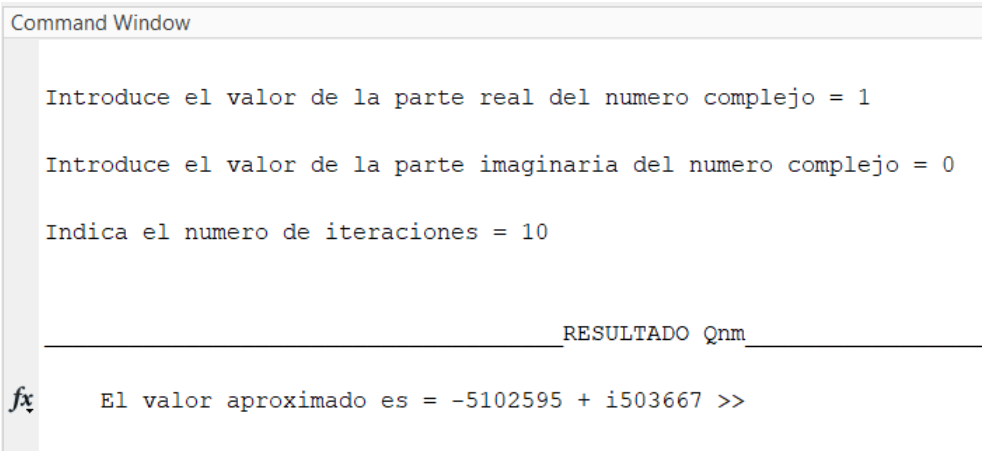


Figura 4-26: Resultado de la MAC compleja con algoritmo Cordic con representación Q_{nm} en Matlab.

Fuente: autoría propria

Nota 15: El código de este programa se puede consultar de manera más detallada en el anexo 13 donde está el código para la MAC compleja con el algoritmo Cordic en representación Q_{nm} en Matlab.

Prueba No. 4:

Ahora se deben probar los vectores "x" que sigue siendo de números complejos solo que ahora será en representación Q_{nm} y además el vector "y" siendo ángulos aplicando Cordico y en representación Q_{nm}, cada valor de θ dará como resultado un número complejo con el algoritmo Cordico, al tener ya todos los números complejos ya se puede realizar la MAC lo cual da un resultado siendo realizado en el MSP430G2553 que está siendo mostrado en la figura 4-27:

$x = [x_0\ x_1\ x_2\ ]$ Q_{nm}
$y = [\theta_0\ \theta_1\ \theta_2\ ]$ Q_{nm} Cordic en el MSP430G2553

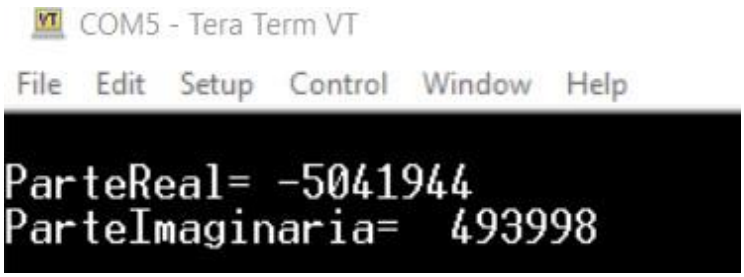


Figura 4-27: Resultado de la MAC compleja con algoritmo Cordic con representación Q_{nm} en el MSP430G2553.

Fuente: autoría propria

Nota 16: El código de este programa se puede consultar de manera más detallada en el anexo 14 donde está el código para la MAC compleja con el algoritmo Cordic en representación Q_{nm} en el MSP430G2553.

Se debe comparar la prueba 3 con la 4 para verificar que los resultados sean iguales

Prueba No. 5:

En la figura 4-28 puede ser apreciada la representación de los resultados de las pruebas 3 y 4 que prácticamente ambas tienen el mismo resultado en su representación fraccionaria y son comparadas con la versión flotante y la respectiva versión con Cordico flotante:

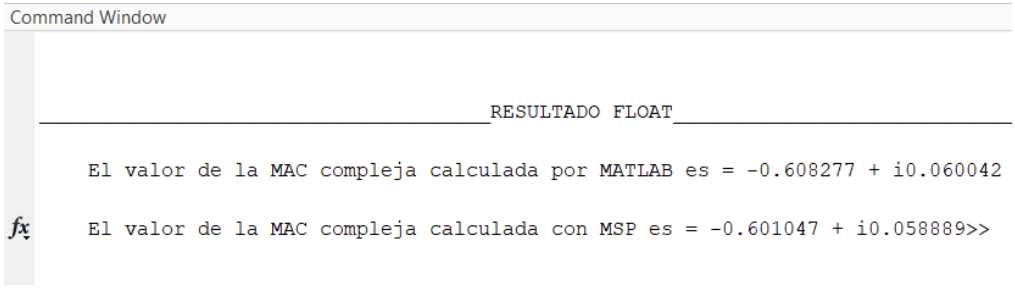


Figura 4-28: Comparación de los resultados de la MAC compleja con algoritmo Cordic en formato flotante en el MSP430G2553 y en Matlab.

Nota 17: El código de este programa se puede consultar de manera más detallada en el anexo 15 donde está el código para la MAC compleja con el algoritmo Cordic para la comparación de los resultados de la MAC compleja con algoritmo Cordic en formato flotante en el MSP430G2553 y en Matlab.

CONCLUSIONES

En esta tesis se desarrolló la implementación de una Unidad Acumulador Multiplicador Compleja (MAC), la cual es el núcleo de todo procesamiento digital de señales e indispensable para que este pueda llevarse a cabo.

Se utilizó la herramienta de programación llamada MATLAB y el lenguaje de programación llamado C++ para desarrollar la MAC y se implementó en una tarjeta de desarrollo de uso comercial de fácil acceso para los estudiantes, que en este caso fue la tarjeta de desarrollo MSP430G2553.

El primer inconveniente que se tuvo fue que muchos dispositivos de gama media baja no son capaces de manejar y procesar funciones trigonométricas, ya que como son de precio bajo su capacidad de memoria y hardware no le permiten procesar estas funciones y eso complicaba el objetivo, entonces lo que se tuvo que hacer fue implementar el algoritmo CORDIC y la representación de números en formato Qnm para hacer uso de las funciones trigonométricas usuales en microcontroladoresque no tuvieran la capacidad de procesar este tipo de operaciones.

Otro reto al que nos enfrentamos fue el despliegue de datos, ya que la pantalla LCD solo se pueden desplegar 32 caracteres, pero para imprimir muchos resultados resultaba algo difícil y no se veía bien, así que por eso se hizo uso de la comunicación serial computadora-tarjeta desarrollo para poder desplegar más valores.

Otra limitación a la que se tuvo que hacer frente fue en la declaración de las variables tipo int, ya que inicialmente estas eran las indicadas para las operaciones de toda la MAC, sin embargo al momento de hacer las operaciones de multiplicación y acumulación estos tipo de datos se quedaron cortos en el rango de valores, por lo tanto se tuvieron que declarar todas las variables tipo Long Int para que el rango de valores se incrementara, este tipo de valores ya son de 32 bits lo cual nos permitió poder representar mejor los resultados aunque cabe aclarar que se necesitan variables con mayor capacidad para poder realizar una MAC de mayor magnitud.

La memoria utilizada para declarar cada una de las variables nos redujo el espacio que estaba destinado para todo el código y el procesamiento de la MAC compleja y CORDIC, por lo tanto, el hardware “nos quedó corto” y por ende solo se pudo aplicar la MAC compleja con CORDIC para un vector de 3 valores cada uno, pero en este caso se mencionaron posteriormente las posibles soluciones para que algún otro dispositivo pueda trabajarlo sin problema.

Se implementó la representación de números en formato Qnm, ya que con esta representación podemos representar cualquier número en flotante en un número entero y con ello aplicar procesos de PDS y así poder ocupar ADC´s y DAC´s en un alcance a futuro para este proyecto.

Una característica de la representación Q_{nm} es que, si se aumenta el valor de bits para la representación de la parte decimal del dato, el valor se aproximará más al valor real esperado, es por esto que al mismo tiempo se requerirá de un mayor uso de memoria, lo que quiere decir que a menor uso de bits para la representación de la parte decimal es más probable tener menor redundancia o exactitud con los resultados, el código que se encarga de realizar este redondeo es parte del código de codificación Q_{nm}.

Al obtener resultados de la implementación de la MAC real y compleja tanto en MATLAB como en el MSP430 los resultados a veces no fueron tan precisos como se hubiera esperado. Ante esto se puede concluir que para ser mucho más precisos en los resultados que se presentan se debe de incrementar el número de bits en la parte fraccionaria, con ello se lograr una mayor precisión en los resultados. Pero también se debe de tomar en cuenta que eso implica más capacidad de procesamiento, más memoria, tener un hardware con mayores prestaciones, entre otras cosas, entonces no es tan sencillo.

Algo importante que se debe de mencionar es que los resultados obtenidos en la implementación de la MAC real y compleja se alejan entre sí, es decir, no son completamente iguales, esto se dio porque se va perdiendo resolución en algunos procesos, por ejemplo, a la hora de transformar los números en formato Qnm perdemos resolución, a la hora de implementar el algoritmo CORDIC perdemos precisión, es por ello por lo que algunos resultados no son completamente iguales.

Otro inconveniente al que se tuvo que enfrentar este proyecto de investigación fue no poder asistir al laboratorio a realizar pruebas, esto freno y limito mucho el alcance de este proyecto y al desarrollo de este.

Otro punto muy importante a mencionar es que si se propone otro método para visualizar los resultados y para ingresar datos sin necesidad de un ADC y de un DAC, se podría tener una aplicación que no requiriera de un equipo de medición y por lo tanto que se puedan hacer en casa, así que se puede decir que este trabajo es pionero para este objetivo, ya que con este proyecto se desarrolló una pequeña herramienta con la cual los alumnos puedan empezar a realizar pequeñas practicas a nivel hardware desde casa.

El MSP430G2553 es capaz de procesar datos en representación flotante y las funciones trigonométricas gracias a las librerías de C++, pero los PIC logran conseguir lo mismo, existen librerías que permiten realizar el procesamiento de los datos de tipo flotante en los PIC's, el problema es que se recomienda no hacerlo ya que el microcontrolador se puede confundir en algunos valores, llegando a reportar resultados equivocados lo que representa un problema a la hora de ocuparlo.

La validación de la MAC real se está llevando a cabo cuando se hace la comparación entre los resultados del Qnm obtenidos en MATLAB, el cual fue 1264 y en el MSP430, el cual también nos dio 1264. Entonces como lo que sale en el

MSP430 es lo mismo que lo que sale en MATLAB con eso se está validando que se está implementando correctamente la MAC real.

La validación de la MAC compleja se está llevando a cabo cuando se hace la comparación entre los resultados del Qnm obtenidos en MATLAB, el cual fue -494 + (-j1562) y en el MSP430, el cual también nos dio -494 + (-j1562). Entonces como lo que sale en el MSP430 es lo mismo que lo que sale en MATLAB con eso se está validando que se está implementando correctamente la MAC compleja.

Por último, se puede concluir que se cumplió con el objetivo general que se planteó para este trabajo al inicio de este, el cual fue diseñar una unidad MAC compleja en el microcontrolador MSP430G2553 utilizando el algoritmo CORDIC para procesos y aplicaciones de PDS. Esto se cumplió porque al final de este trabajo se desarrolló el diseño de una MAC real y compleja las cuales se pueden aplicar a procesos de PDS. Cabe aclarar que, para aplicar la MAC a una transformada de Fourier, una correlación, una convolución entre otras se requiere de algunos elementos más que el alumno tendría que desarrollar por cuenta propia, pero le será de vital ayuda la MAC diseñada en este proyecto.

TRABAJO FUTURO

Si alguien decide continuar con este trabajo, de entrada se puede implementar un ADC (Analogue to Digital Converter) en español un conversor de señal analógica a digital o conversor analógico digital y también un DAC (Digital to Analogue Converter) en español un conversor de señal digital a analógica o conversor digital analógico, para que pueda procesar señales para poder lograr esto hay que manejar la representación Q_{nm} tanto en el DAC como en el ADC y luego que los resultados arrojados en Q_{nm} sean acoplados a los DAC y ADC, una vez teniendo este trabajo se puede implementar alguna aplicación como lo que es un filtro digital, una transformada de Fourier o una correlación.

Otro trabajo futuro es la implementación del algoritmo en un dispositivo como puede ser un FPGA (Field Programmable Gate Arrays) en español es una matriz de puertas lógicas programable en campo o un CPLD (Complex Programmable Logic Device) en español es un dispositivo lógico programable complejo, que tienen mayores prestaciones que un microcontrolador y es difícil que se encuentren con las funciones trigonométricas.

Referencias:

[1] Huerta, R. (14 de abril, 2003), "Representación de Números en punto fijo y flotante." Recuperado de http://www2.elo.utfsm.cl/~elo385/docs/Biblio/Lab4/Números_Punto_Fijo_y_Flotante.pdf

[2] Gómez, J. (27 de abril, 2021), "Representación de números en formato Qnm." Clase de PDS.

[3] "Operaciones con números binarios y conversión a otros sistemas." Recuperado de https://www.ingmecafenix.com/electronica/operaciones-con-números-binarios/

[4] Martínez, A. (septiembre, 2005), "Análisis de Arquitectura de multiplicadores digitales y su implementación en FPGA." Recuperado de

https://tesis.ipn.mx/bitstream/handle/123456789/2641/448_2005_ESIME-CUL_MAESTRIA_alfredo_martinez.pdf?sequence=1&isAllowed=y

[5] "Muestreo Digital." Recuperado de

http://serbal.pntic.mec.es/srug0007/archivos/radiocomúnicaciones/3%20SE%D1ALES%20DIGITALES/Muestreo%20digital.pdf

[6] Proakis, J. y Manolakis, D. (2007). "*Tratamiento Digital de Señales*." PEARSON EDUCACIÓN (4ta Edición). pp. 17-19, 23-28, 31-32, 361-366.

[7] "Muestreo y recuperación de las señales." recuperado de http://agamenon.tsc.uah.es/Asignaturas/it/tds/apuntes/tds_tema_2_teoría.pdf

[8] Laakso, T. L., (January 1996), et al. "Splitting the Unit Delay," HEET Signal Processing Magazine, pp. 46.

[9] Sajad Shawl, M., Singh, A., Gaur, N., Bathla, S., and Mehra, A., (2018), "Implementation of Area and Power Efficient Components of a MAC Unit for DSP Processors," 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), pp. 1155-1159.

[10] Rakesh S. and Grace K. S. V., (2017), "A survey on the design and performance of various MAC unit architectures," 2017 IEEE International Conference on Circuits and Systems (ICCS), pp. 312-315.

[11] Samimian N., Mousazadeh M. and Khoie A., (2019), "A Time-based All-Digital Analog to Digital converter for IOT Applications," 2019 27th Iranian Conference on Electrical Engineering (ICEE), pp. 249-252

[12] Kim S. and Kwon K., (2016), "A hybrid ADC combining capacitive DAC-based multi-bit/cycle SAR ADC with flash ADC," 2016 International Conference on Electronics, Information, and Communications (ICEIC), pp. 1-4

[13] https://www.arrow.com/es-mx/categories/data-acquisition/data-converters/adcs.

[14] "Problemás de Tratamiento Digital de Señales" tomado de http://agamenon.tsc.uah.es/Asignaturas/it/tds/apuntes/tds_tema_2_teoría.pdf

[15] **"**Introducción a los Filtros Digitales" tomado de

https://www.eumus.edu.uy/eme/ensenanza/electivas/dsp/presentaciones/clase10.pdf

[16] Olvera J. **"**Convolución: Un proceso natural en los sistemas lineales e invariantes en el tiempo" tomado de https://www.mty.itesm.mx/etie/deptos/m/Paginas/MateParaTodos/e07/archivos/convolución.pdf

[17] (diciembre 2013), "Convolución, procesamiento de señales." Tomado de

https://ramaucsa.wordpress.com/2013/12/17/convolución-procesamiento-de-senales/

[18] Jimmy Alexander Cortés O, Francisco Alejandro Medina A Y José Andrés Chaves O, Scientia et Technica Año XIII, No 34, Mayo de 2007. Universidad Tecnológica de Pereira "DEL ANÁLISIS DE FOURIER A LAS WAVELETS ANÁLISIS DE FOURIER" file:///C:/Users/soyfe/Downloads/5563-Texto%20del%20art%C3%ADculo-3787-1-10-20120425%20(1).pdf

[19] Jesús Bernal, Pedro Gómez y Jesús Bobadilla, Departamento de Informática Aplicada, Universidad Politécnica de Madrid "UNA VISIÓN PRÁCTICA EN EL USO DE LA TRANSFORMADA DE FOURIER COMO HERRAMIENTA PARA EL ANÁLISIS ESPECTRAL DE LA VOZ" file:///C:/Users/soyfe/Downloads/144489-Text%20de%20l'article-256979-1-10-20100614%20(1).pdf

[20] Franco V. (Marzo 2013), Universidad Nacional del Sur, Avda. Alem 1253, B8000CPB Bahía Blanca, Argentina, Transformada de Fourier en procesamiento digital de Imágenes Funciones de Variable Compleja http://lcr.uns.edu.ar/fvc/NotasDeAplicacion/FVC-Vela%20Franco.pdf

[21] Gouri Wazurkar, D. S. (2016). Power Efficient GALS Pipelined MAC Unit for FFT with Complex Numbers. International conference on Signal Processing,

Communication, Power and Embedded System (SCOPES)-2016 (págs. 1361-1364). Nagpur, India : IEEE.

[22] H. R. Spoorthi, C. P. (2019). Low Power Datapath Architecture for Multiply - Accumulate (MAC) Unit. 2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT) (págs. 391-395). IEEE.

[23] Mohamed Asan, N. M. (2014). An Efficient Hardware Based MAC Design in Digital Filters with Complex Numbers. International Conference on Signal Processing and Integrated Networks (SPIN) (págs. 475-480). IEEE.

[24] Okan K, E. (1994). A comparative Review of Real and Complex Fourier-Related Transforms. (págs. 429-447). IEEE.

[25] Salim, T. (2008). A Serial MAC Architecture for FPGA Implementation of a Complex Adaptive Beamformer. (págs. 1-6). Australia : IEEE.

[26] Yuan Luo, Z. Z. (2014). Architecture and Implementation of a Vector MAC Unit for Complex Number. 2014 9th International Conference on Communications and Networking in China (CHINACOM), (págs. 589-594).

[27] Duoandikoetxea, UNAM-Managua, (2003), "Lecciones sobre las series y transformadas y transformadas de Fourier" tomado de https://www.ugr.es/~acanada/docencia/matematicas/analisisdefourier/Duoandikoetxeafourier.pdf

[28] González G. Departamento de Matemática y Computación, Facultad Experimental de Ciencias Universidad del Zulia, “Series de Fourier, Transformadas de Fourier y Aplicaciones” tomado de https://www.emis.de/journals/DM/v5/art6.pdf

[29] Andraka, R. (1996). A survey of CORDIC algorithms for FPGA based computers . (págs. 191-200). North Kingstown, RI: Andraka Consulting Group.

[30] Bronshtein, I. y Semendiaev, K. (1973). “Manual de matemáticas para ingenieros y estudiantes”. Editorial MIR (2da Edición). Pp 61,99,104 y 566.

[31] Alaminos , J. (2012). “Apuntes de Cálculo avanzado”. Departamento de Análisis Matemático de la Universidad de Granada. Tomado de https://www.ugr.es/~dpto_am/docencia/Apuntes/Cálculo_avanzado_Caminos.pdf

[32] (2013). “MSP430x2xx Family, User’s Guide”. Texas Instruments. Pág 424. Tomado de https://www.ti.com/lit/ug/slau144j/slau144j.pdf

ANEXOS

Anexo 1:

Realizando función la función tangente inversa en el MSP430G2553

Código en c++ programado en el entorno IAR.

```
#include "io430.h"

#include "LCD16x2c.h" //librería para poder utilizar la pantalla LCD

#include "stdio.h"     //Librería para hacer uso de la función sprintf para desplegar los datos en el LCD

#include "math.h"      //Librería para hacer uso de la función arco tangente

char Dato[12]={"Ángulo  atan"},res[9],valang[3];  //Cadenas de caracteres para hacer el despliegue de datos

float ángulo=0;            //Variable que almacena el valor del ángulo

double resultado=0;  //Variable tipo double que almacena el valor de la operación


int main( void )

{

  // Stop watchdog timer to prevent time out reset

WDTCTL = WDTPW + WDTHOLD;

BCSCTL1 = CALBC1_1MHZ; //DCO configurado a 1MHz

DCOCTL = CALDCO_1MHZ;

 Ini_Lcd();  //Función para inicializar la pantalla LCD

__bis_SR_register (GIE); //permiso a nivel global

Cmd_Lcd(0x01);  //Se limpia la pantalla LCD y se coloca en la primera línea

 for(char i=0; i<=11 ;i++)  //Se realiza el despliegue del mensaje "ángulo atan"

{

Dato_Lcd(Dato[i]);

}

  while(1)
```

```
{
 ángulo=1;
resultado=180*atan(ángulo)/3.1416; //Se realiza en cálculo del arco tangente del ángulo indicado
sprintf(res,"%f",resultado); //El valor se convierte en una cadena de caracteres para hacer uso de la función Dato_Lcd()
sprintf(valang,"%f",ángulo); //El valor se convierte en una cadena de caracteres para hacer uso de la función Dato_Lcd()
 Cmd_Lcd(0xC1); //El cursor se coloca en la segunda línea del LCD
for(char i=0; i<=2 ;i++) //Se hace el despliegue del resultado en la segunda línea de la pantalla LCD con ayuda de la función Dato_Lcd
{
 Dato_Lcd(valang[i]);
}
 Cmd_Lcd(0xC9);
 for(char i=0; i<=8 ;i++)
{
 Dato_Lcd(res[i]);
}
__delay_cycles(1000000); //Se realiza una pausa de 1 segundo para visualizar el valor
}
}
```

En la figura siguiente se puede observar el resultado que se obtiene del programa anterior, en donde se evalúa a la función tangente inversa (ver figura 1-A1).

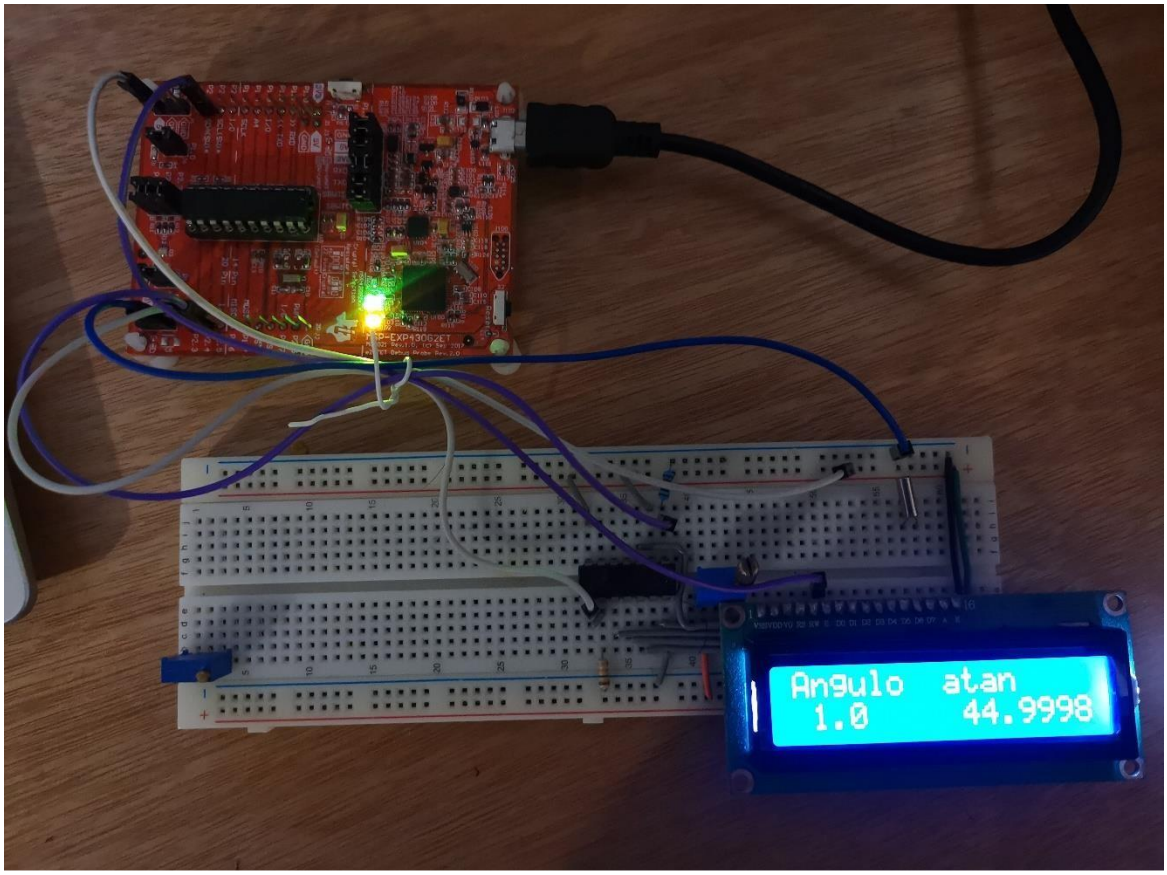


Figura 1-A1: Mostrando el resultado de evaluar la función tangente inversa en el MSP430G2553.

Fuente: autoría propia.

Nota importante: Los resultados obtenidos con el MSP430G2553, los cuales se muestran en la pantalla LCD están dados en radianes, así que si necesitamos saber su equivalente en grados únicamente debemos de multiplicar el valor en radianes por $\frac{180}{\pi}$ y así obtendremos el equivalente en grados. Esto aplica también para las pruebas que se muestran a continuación.

Realizando función la función seno en el MSP430G2553

Código en c++ programado en el entorno IAR.

```
#include "io430.h"

#include "LCD16x2c.h" //Librería para poder utilizar la pantalla LCD

#include "stdio.h"     //Librería para hacer uso de la función sprintf para desplegar los datos en el LCD

#include "math.h"     //Librería para hacer uso de la función seno
```

```
char Dato[14]={"Ángulo  Seno"},seno[9],valang[3];  //Cadenas de caracteres para hacer el despliegue de datos

int ángulo=0; //Variable que almacena el valor del ángulo al que se desea calcular el seno

double resultado=0; //Variable tipo double que almacena el valor del seno del ángulo


int main( void )

{

// Stop watchdog timer to prevent time out reset

 WDTCTL = WDTPW + WDTHOLD;

BCSCTL1 = CALBC1_1MHZ; //DCO configurado a 1MHz

DCOCTL = CALDCO_1MHZ;

Ini_Lcd();  //Función para inicializar la pantalla LCD

__bis_SR_register (GIE); //permiso a nivel global

Cmd_Lcd(0x01);  //Se limpia la pantalla LCD y se coloca en la primer línea

for(char i=0; i<=11 ;i++)

{

Dato_Lcd(Dato[i]);

}


 while(1)

{

ángulo=90;  //Al ángulo se le asignan un valor a calcular (90 grados)

resultado=sin(3.1416*ángulo/180);   //Se hace el cálculo de la función seno del ángulo   especificado

sprintf(seno,"%f",resultado);  //El valor se convierte en una cadena de caracteres para hacer uso de la función Dato_Lcd()

sprintf(valang,"%u",ángulo);  //El valor se convierte en una cadena de caracteres para hacer uso de la función Dato_Lcd()
```

```
Cmd_Lcd(0xC1);  //El cursor del LCD se coloca en el primer espacio de la segunda línea de la pantalla LCD

for(char i=0; i<=2 ;i++)  //Se hace el despliegue de los valores del ángulo y el seno en la segunda línea de la pantalla LCD con ayuda de la función Dato_Lcd

{

 if(valang[i]!=0){Dato_Lcd(valang[i]);}

 }

Cmd_Lcd(0xC9);

for(char i=0; i<=5 ;i++)

 {

 Dato_Lcd(seno[i]);

}

__delay_cycles(1000000);  //Se realiza una pausa de 1 segundo para visualizar el valor

}

}
```

En la figura siguiente se puede observar el resultado que se obtiene del programa anterior, en donde se evalúa a la función seno (ver figura 2-A1).

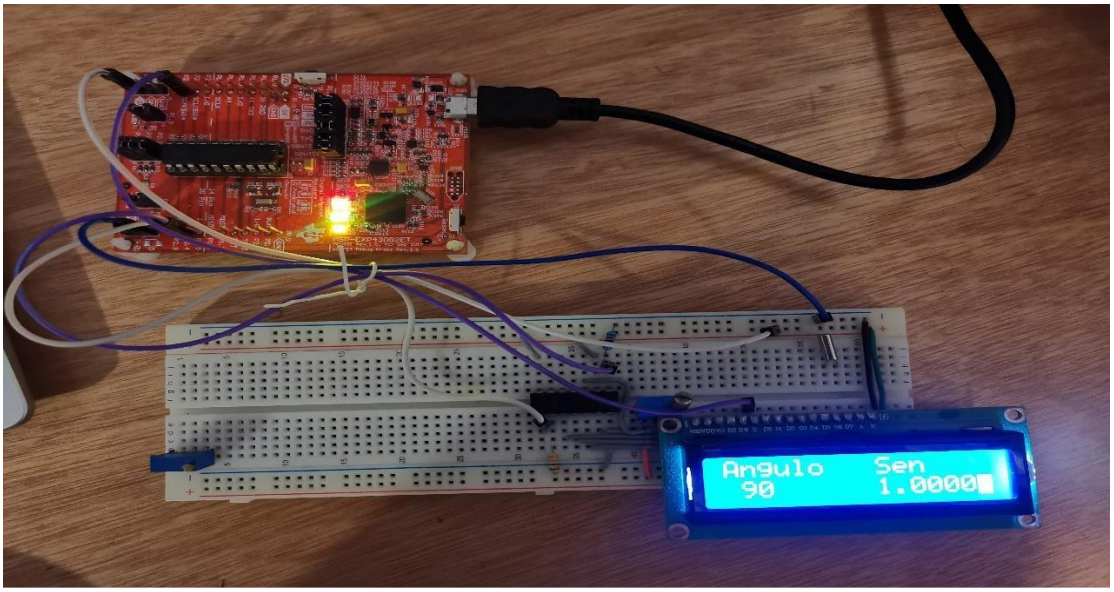


Figura 2-A1: Mostrando el resultado de evaluar la función seno en el MSP430G2553.

Fuente: autoría propia.

En la figura siguiente se puede observar el resultado que se obtiene del programa anterior, en donde se evalúa a la función seno (ver figura 2-A1).

Realizando función la función seno+coseno en el MSP430G2553

Código en c++ programado en el entorno IAR.

```
#include "io430.h"

#include "LCD16x2c.h" //Librería para poder utilizar la pantalla LCD

#include "stdio.h"     //Librería para hacer uso de la función sprintf para desplegar los datos en el LCD

#include "math.h"     //Librería para hacer uso de la función seno

char  Dato[10]={"Resultado="},res[9];    //Cadenas de caracteres para hacer el despliegue de datos

double resultado=0;  //Variable tipo double que almacena el valor de la operación


int main( void )

{

// Stop watchdog timer to prevent time out reset

  WDTCTL = WDTPW + WDTHOLD;


 BCSCTL1 = CALBC1_1MHZ; //DCO configurado a 1MHz

DCOCTL = CALDCO_1MHZ;

Ini_Lcd();  //función para inicializar la pantalla LCD

   __bis_SR_register (GIE); //permiso a nivel global

Cmd_Lcd(0x01);  //Se limpia la pantalla LCD y se coloca en la primer línea

for(char  i=0;  i<=9  ;i++)     //Se realiza el despliegue del mensaje "-0.035cos25+0.01sin25"

{

Dato_Lcd(Dato[i]);

 }

while(1)
```

```
{
resultado=-0.035*cos(3.1416*25/180)+0.01*sin(3.1416*25/180); //Se realiza en cálculo de la suma con los ángulos y amplitudes indicados
sprintf(res,"%f",resultado);  //El valor se convierte en una cadena de caracteres para hacer uso de la función Dato_Lcd()
Cmd_Lcd(0xC1);  //El cursor se coloca en la segunda línea del LCD
 for(char i=0; i<=8 ;i++) //Se hace el despliegue del resultado en la segunda línea de la pantalla LCD con ayuda de la función Dato_Lcd
{
 Dato_Lcd(res[i]);
}
__delay_cycles(1000000);  //Se realiza una pausa de 1 segundo para visualizar el valor
 }
}
```

En la figura siguiente se puede observar el resultado que se obtiene del programa anterior, en donde se evalúa a la función seno x + cos y (ver figura 3-A1).

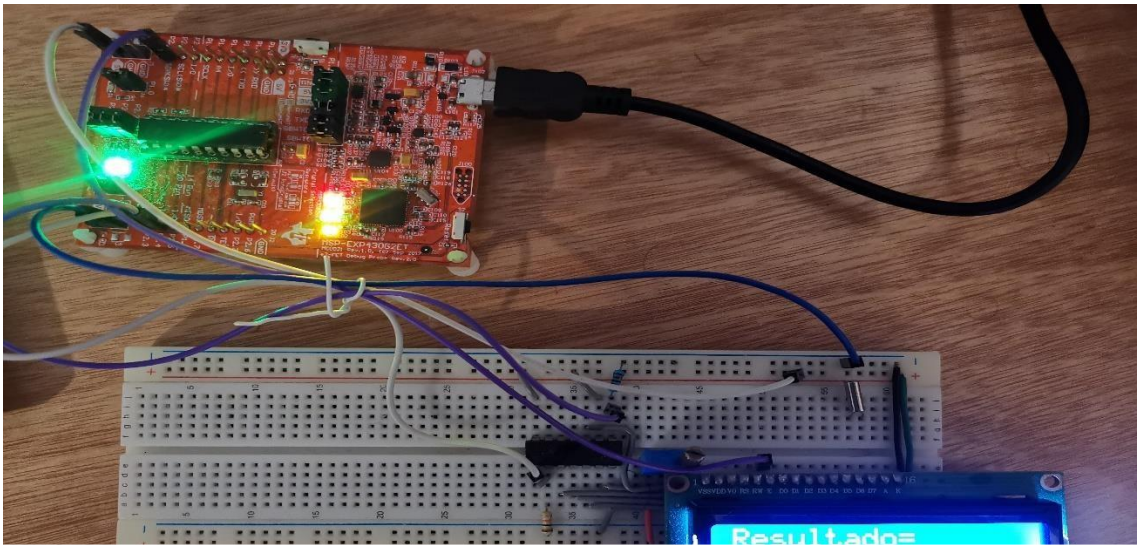


Figura 3-A1: Mostrando el resultado de evaluar la función "-0.035cos25+0.01sin25" en el MSP430G2553.

Fuente: autoría propia.

Nota importante: el resultado mostrado en la figura 3-A1 fue obtenido al realizar la siguiente suma: "-0.035cos25+0.01sin25", pero en la pantalla LCD por motivos de espacio no se pudo mostrar completa la operación, es por ello que solo se muestra el resultado.

Anexo 2:

Realizando la programación de la MAC en el MSP430G2553

Código en c++ programado en el entorno IAR.

```
#include "io430.h"

#include "LCD16x2c.h" //Librería para poder utilizar la pantalla LCD
```

Nota importante: el resultado mostrado en la figura 3-A1 fue obtenido al realizar la siguiente suma: "-0.035cos25+0.01sin25", pero en la pantalla LCD por motivos de espacio no se pudo mostrar completa la operación, es por ello que solo se muestra el resultado.

```
#include "stdio.h"  //Librería para hacer uso de la función sprintf para desplegar los datos en el LCD

char Dato[14]={"Resultado MAC="},resultado[7];

float x1[16]={0.54,0.67,0.34,-0.64,-0.76,-0.73,0.16,0.95,0.12,-0.05,0.67,0.6,0.34,0.6,-0.77,0.56}; //Se declara un vector para realizar la MAC en formato float

float x2[16]={0.1,0.65,-0.78,0.35,0.89,-0.37,0.64,0.57,-0.69,-0.26,-0.57,0.43,0.12,0.58,-0.68,0.45}; //Se declara un segundo vector para realizar la MAC en formato float

float Xres[16]; //Se declara un vector para almacenar las multiplicaciones de elemento a elemento de los dos vectores

float acum=0;  //Se declara un acumulador tipo float que se encargara de realizarla suma de todos los productos


int main( void )

{

//Stop watchdog timer to prevent time out reset

WDTCTL = WDTPW + WDTHOLD;

BCSCTL1 = CALBC1_1MHZ;        //DCO configurado a 1MHz
```

```
DCOCTL  = CALDCO_1MHZ;


 Ini_Lcd();                  //Función para inicializar la pantalla LCD
Cmd_Lcd(0x01);                //Se limpia la pantalla LCD y se coloca en la primer línea
for(char i=0; i<=13 ;i++)     //Se despliega el mensaje "Resultado MAC" con ayuda de la función Dato_Lcd
 {
 Dato_Lcd(Dato[i]);
}
 while(1){  //While infinito


 acum=0;                     //Se inicializa el acumulador con valor 0
  for(char n=0; n<16; n++)        //Se realiza la operación de multiplicación de elemento a elemento de los dos vectores
 {
Xres[n]=x1[n]*x2[n];
 }
for(char k=0; k<16; k++)          //Se realiza la suma de todos los productos previamente calculados y almacenados
  {
  acum+=Xres[k];
 }
sprintf(resultado,"%f",acum); //Compone una cadena con el valor de la MAC, el contenido se almacena como cadena en resultado
Cmd_Lcd(0xC0);            //El cursor de la pantalla LCD se posiciona en la segunda línea de la pantalla LCD
for(char i=0; i<=7 ;i++)        //Se hace el despliegue del valor de la suma en la segunda línea de la pantalla LCD con ayuda de la función Dato_Lcd
{
```

```
Dato_Lcd(resultado[i]);
}
__delay_cycles(1000000);       //Se realiza una pausa de 1 segundo para visualizar el valor
}
}
```

Anexo 3:

Realizando la programación de la MAC compleja en el MATLAB

Código programado en el entorno de MATLAB

```
%%SIMULACION MAC%%
clc       %%borra todo el texto de la ventana de comandos, lo que da como resultado una pantalla clara
clear all  %%elimina todas las variables del espacio de trabajo actual y las libera de la memoria del sistema.
close all  %%Formas de cerrar o salir
x1=[0.54,0.67,0.34,-0.64,-0.76,-0.73,0.16,0.95,0.12,-0.05,0.67,0.6,0.34,0.6,-0.77,0.56];  %%Se define un vector para realizar la MAC
for(K=1:1:16)                    %%bucle externo para repetir un número especificado de veces para calcular la MAC compleja
acum=0;                          %%Se reinicia el acumulador cada que se realiza el bucle interno
for(n=1:1:16)                    %%bucle interno para repetir un número especificado de veces para calcular la MAC compleja
acum=acum+x1(n)*exp(-1i*2*pi*(K-1)*(n-1)/16); %%Se realiza la multiplicación de cada elemento del vector por la exponencial compleja y se acumula en una variable
end
Xk(K)=abs(acum);                 %%Se almacena el valor absoluto de cada valor de la MAC en otro vector
```

```
End
Xk                          %%Se despliega el vector correspondiente a la MAC compleja
```

Nota importante: el resultado mostrado en la figura 3-A1 fue obtenido al realizar la siguiente suma: "-0.035cos25+0.01sin25", pero en la pantalla LCD por motivos de espacio no se pudo mostrar completa la operación, es por ello que solo se muestra el resultado.

Anexo 4:

Realizando la programación de la MAC compleja en el MSP430G2553

Código en c++ programado en el entorno de IAR.

```
#include "io430.h"

#include "math.h"       //Librería para hacer uso de las funciones sqrt(raíz cuadrada) y pow(potencia)

#include "stdio.h"      //Librería para hacer uso de la función sprintf para desplegar los datos en el LCD

#include "LCD16x2c.h"  //Librería para poder utilizar la pantalla LCD

#include "complex.h"    //Librería que nos permite hacer uso de números complejos

float                   x1[16]={0.54,0.67,0.34,-0.64,-0.76,-0.73,0.16,0.95,0.12,-0.05,0.67,0.6,0.34,0.6,-0.77,0.56};  //Se declara un vector para realizar la MAC en formato float

double complex acum=0+0*I,res[15],acum1=0+0*I; //Se decalaran variables tipo double complex para realizar la acumulacion y un vector para los productos

char resultado[9],msj[5];  //Cadenas de caracteres para hacer el despliegue de datos


int main( void )

{

 // Stop watchdog timer to prevent time out reset

 WDTCTL = WDTPW + WDTHOLD;
```

```
 BCSCTL1 = CALBC1_1MHZ; //DCO configurado a 1MHz
  DCOCTL = CALDCO_1MHZ;
Ini_Lcd();  //Función para inicializar la pantalla LCD
__bis_SR_register (GIE); //permiso a nivel global


while(1)
{
 for(char k=0;k<=15;k++)  //Bucle externo para recorrer todo el vector inicial
{
acum=0+0*I;              //Se inicializan los acumuladores en cada iteración para los distintos valores del vector inicial
acum1=0+0*I;
for(char n=0;n<=15;n++) //Bucle interno para realizar la acumulación y los productos correspondientes
{
acum+=x1[n]*cexp(-I*2*3.14159265*(k)*(n)/16); //Se realiza la MAC compleja con la exponencial compleja
 }
acum1=sqrt(pow(creal(acum),2)+pow(cimag(acum),2)); //Se realiza el cálculo del modulo de cada acumulacion de productos
res[k]=acum1;       //Estos productos se almacenan en un vector para su posterior despliegue
sprintf(msj,"X[%d]=",k); //Se realiza el despliegue del mensaje "X[n]="
Cmd_Lcd(0x01);     //Se limpia la pantalla LCD y se coloca en la primer línea
for(char i=0; i<=4 ;i++)
 {
Dato_Lcd(msj[i]);
 }
sprintf(resultado,"%f",res[k]); //El valor se convierte en una cadena de caracteres para hacer uso de la función Dato_Lcd()
```

```
Cmd_Lcd(0xC0);     //El cursor de la pantalla LCD se posiciona en la segunda línea de la pantalla LCD

for(char i=0; i<=8 ;i++) //Se hace el despliegue de los valores de la MAC en la segunda línea de la pantalla LCD con ayuda de la función Dato_Lcd

 {

 Dato_Lcd(resultado[i]);

 }

__delay_cycles(1000000); //Se realiza una pausa de 1 segundo para visualizar el valor

}

}

  }
```

Anexo 5:

Realizando la programación del algoritmo CORDIC en MATLAB

Código programado en el entorno de MATLAB

Primero se mostrará un diagrama de flujo de este algoritmo (ver figura 4-A5)

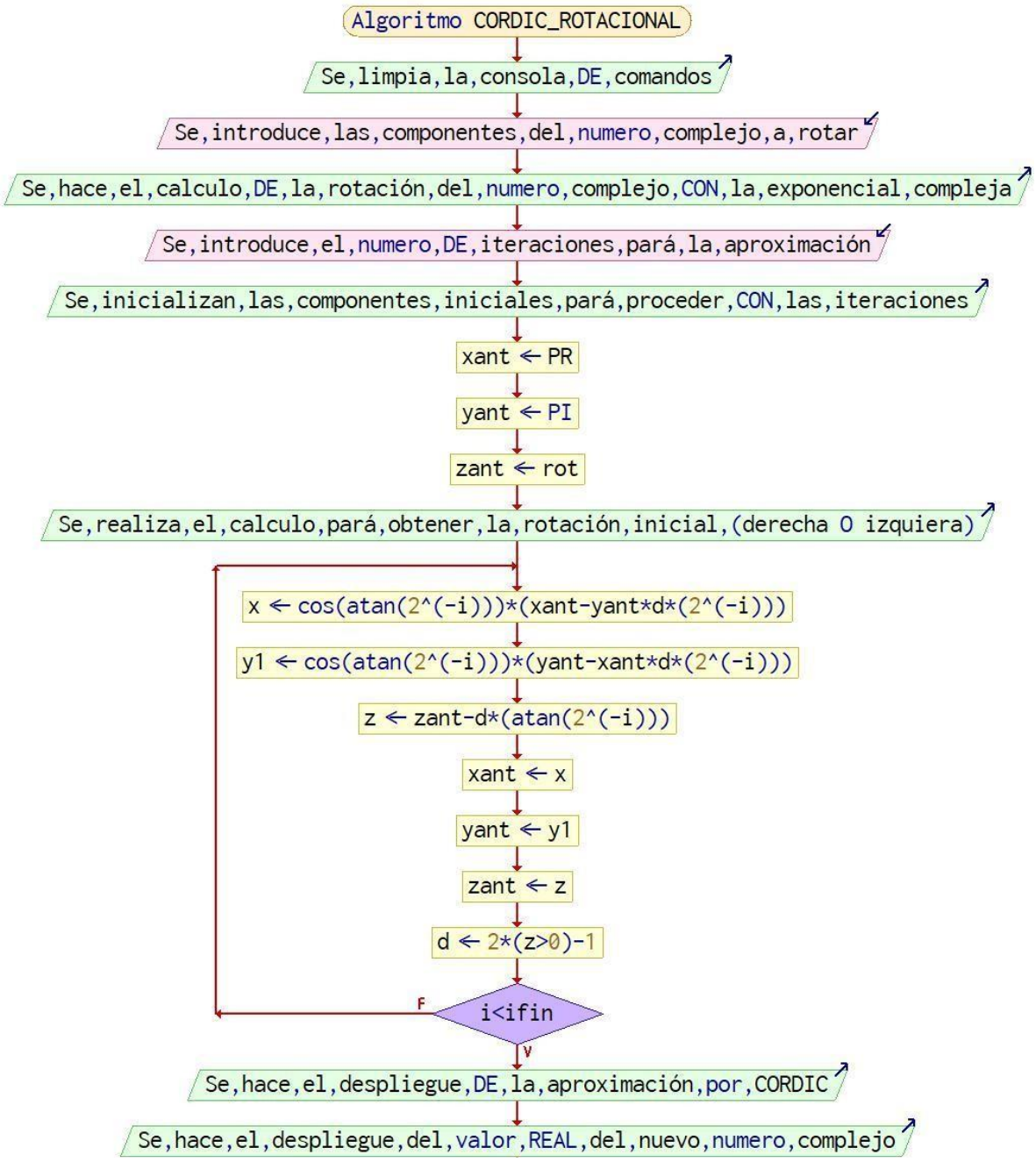


Figura 4-A5: Diagrama de flujo del algoritmo CORDIC.

Fuente: autoría propia.

Programa realizado en MATLAB

```
%%Simulación de CORDIC rotaciónal

clc       %%borra todo el texto de la ventana de comandos, lo que da como resultado una pantalla clara

clear all %%elimina todas las variables del espacio de trabajo actual y las libera de la memoria del sistema.

close all  %%Formas de cerrar o salir

PR=input('\nIntroduce el valor de la parte real del número complejo = '); %%En la variable PR se almacena

 %%el valor de la parte real del número complejo que se desea aproximar por CORDIC

PI=input('\nIntroduce el valor de la parte imaginaria del número complejo = '); %%En la variable PI se

 %%almacena el valor de la parte imaginaria del número complejo que se desea aproximar por CORDIC

rot=input('\nIntroduce el valor del ángulo de rotación(?) = ');  %%En la variable rot se almacena el valor %%del ángulo de rotación al que se desea aproximar el número complejo

RES=(PR+i*PI)*exp(i*rot*pi/180); %%Se calcula el Valor real de la aproximación
                                 multiplicando el número %%complejo por la exponencial compleja del ángulo de rotación

ifin=input('\nIndica el número de iteraciones = '); %%En la variable ifin se almacena el número de %%iteraciones que indique el usuario

xant=PR; %%Se inicializa la variable correspondiente a la componente x-1 con la parte real del número  %%complejo inicial

yant=PI; %%Se inicializa la variable correspondiente a la componente y-1 con la parte imaginaria del %%número complejo inicial

zant=rot; %%Se inicializa la variable correspondiente al ángulo de rotación con el ángulo al que se desea %%rotar el número complejo inicial

d=(2*(rot>0)-1);    %%Se hace el cálculo del signo inicial para realizar la rotación del número complejo %%hacia el ángulo deseado

for(i=1:ifin)
```

Nota importante: el resultado mostrado en la figura 3-A1 fue obtenido al realizar la siguiente suma: "-0.035cos25+0.01sin25", pero en la pantalla LCD por motivos de espacio no se pudo mostrar completa la operación, es por ello que solo se muestra el resultado.

```
 x=cos(atan(2^-i))*(xant-yant*d*(2^-i)); %%Se calcula la nueva componente x con la formula indicada

y=cos(atan(2^-i))*(yant+xant*d*(2^-i)); %%Se calcula la nueva componente y con la formula indicada

z=zant-d*(atand(2^-i));          %%Se calcula el nuevo ángulo de rotación con la formula indicada

  xant=x; %%Se sobrescribe la componente x-1 con la componente %%calculada para ser utilizada en la siguiente iteración

 yant=y;   %%Se sobrescribe la componente y-1 con la componente %%calculada para ser utilizada en la siguiente iteración

 zant=z;          %%Se sobrescribe el ángulo de rotación z-1 con la componente %%calculada para ser utilizada en la siguiente iteración

 d=(2*(z>0)-1);                %%Se hace el cálculo del signo para saber a dónde hay que %rotar el vector nuevamente (izquierda o derecha)

end

fprintf('\n\n\tEl valor aproximado es = %f + i%f , con un ángulo de = %f',x,y,z); %%Se hace el despligue %%de la aproximación por CORDIC

fprintf('\n\tEl valor real es = %f + %fi',real(RES),imag(RES));     %%Se hace el despliegue %del número complejo real para la comparativa
```

Anexo 6:

El código del programa que lleva a cabo la gráfica de la magnitud del error producido debido al número de bits utilizados para la redundancia de la parte fraccionaria de los valores:

```
clear all

close all

clc
```

```
m=[5 6 7 8 9 10 11 12 13 14 15];  %%Se hará la aproximación con m=5,6,7,8,9,10 y 11

x1=[0.54 0.67 0.34 -0.64 -0.76 -0.73 0.16 0.95 0.12 -0.05 0.67 0.6 0.34 0.6 -0.77 0.56];  %%Se define un primer vector para realizar la MAC en formato Qnm y float

x2=[0.1 0.65 -0.78 0.35 0.89 -0.37 0.64 0.57 -0.69 -0.26 -0.57 0.43 0.12 0.58 -0.68 0.45];%%Se define un segundo vector para realizar la MAC en formato Qnm y float

flota=sum(x1.*x2); %%Se realiza la MAC en formato float para hacer el cálculo del error

for(i=1:11)  %%Bucle para calcular las diferentes aproximaciones

 [X XQnm1 XD1 XDS1]=Cod_Qnm(x1,1,(i+4));      % Cambiar de decimal a Qnm
[X XQnm2 XD2 XDS2]=Cod_Qnm(x2,1,(i+4));      % Cambiar de decimal a Qnm

XDS=sum(XDS1.*XDS2);  %%Se realiza la MAC en formato Qnm

  [XQnm  XD]=Decod_Qnm(XDS,6,(i+4)*2,'S');   %%Se hace la conversión de Qnm a float para la comparación

if(flota<XQnm) %%Se realiza el cálculo del error para cada valor de m

error(i)=XQnm-flota;

else
error(i)=flota-XQnm;
end
end
 [XQnm XD]=Decod_Qnm(XDS,6,(i+4)*2,'S');  %%Se hace la conversión de Qnm a float para la comparación

semilogy(m,error); %%Se grafican los valores en una grafica tipo semilogarítmica

title('m vs error')  %%Se coloca el titulo a la grafica

ylabel('error') %%Se coloca un nombre al eje y

xlabel('m [bits]') %%Se coloca un nombre al eje x

hold on

function [X XQnm XD XDS]=Cod_Qnm(xnTs,n,m)

Entero_positivo=2^(n-1)-1/(2^m);

Entero_negativo=-2^(n-1);

resolucion=1/(2^m);
```

```
Muestras=xnTs;
indMm=find(Muestras==max(Muestras));
Muestra_mayor=Muestras(indMm(1));
indMm=find(Muestras==min(Muestras));
Muestra_menor=Muestras(indMm(1));
if Muestra_mayor>Entero_positivo || Muestra_menor<Entero_negativo
fprintf('\n\t¡Error! no hay suficientes bits para codificar\n')
 fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)
fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
if Muestra_M==1
Muestra_M=1.1;
End
ns=ceil(log2(Muestra_M))+1;
Entero_positivo=2^(ns-1)-1/(2^m);
Entero_negativo=-2^(ns-1);
fprintf('\n\tAumente el valor de n a %d\n',ns)
fprintf('\n\tRango de codificación sería: [%f a %f]\n\n',Entero_negativo,Entero_positivo)
X=Muestras;
XQnm=zeros(1,numel(X));
XD=XQnm;
XDS=XD;
Else
%     ind=find(abs(xnTs)<1/(2^m));     % busca los elementos que no alcanzan la minima resolución
%     xnTs(ind)=0;
```

```
%       ind=find((xnTs)==(2^(n-1)));       % Busca los elementos con la maxima resolución
%     xnTs(ind)=(2^(n-1))-(1/(2^m));
 signo=(xnTs>=0);
signo=2*signo-1;
xnTsA=xnTs;
xnTs=abs(xnTs);
x1=xnTs/resolucion;
 xQnmES=floor(x1).*signo; %% Aquí se cambia el criterio truncamiento floor y redondeo round
  xQnm=xQnmES*(resolucion);
 ind=find(xQnmES<0);
 xQnmEU=xQnmES;
 xQnmEU(ind)=xQnmEU(ind)+2^(n+m);
 X=Muestras;
 XQnm=xQnm;
XD=xQnmEU;
XDS=xQnmES;
End
end  %%Función que codifica un valor o un vector a Qnm, recibe los datos así como los valores de n y m
function [XQnm XD]=Decod_Qnm(X,n,m,S)
 if S=='S' || S=='s'
 Entero_positivo=2^(n+m-1)-1;
Entero_negativo=-2^(n+m-1);
Muestras=X;
indMm=find(Muestras==max(Muestras));
Muestra_mayor=Muestras(indMm(1));
```

```
 indMm=find(Muestras==min(Muestras));

Muestra_menor=Muestras(indMm(1));

Max=max(abs(X));

 N=log2(Max(1));

if N>(n+m-1)

 fprintf('\n\t¡Error! no hay suficientes bits para decodificar\n')

 fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)

fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
ns=floor(log2(Muestra_M))+1;
Entero_positivo=2^(ns-1);
Entero_negativo=-2^(ns-1);
fprintf('\n\tAumente el valor de n+m a %d\n',ns+1)
fprintf('\n\tRango de codificación sería: [%f a
%f]\n\n',Entero_negativo,Entero_positivo)

X=Muestras;
XQnm=zeros(1,numel(X));
XD=XQnm;
  else
indc=find(X<0);
Xaux=X;
X(indc)=X(indc)+2^(n+m);
XC=de2bi(X,n+m);
% XC=[PD PE];
PDc=1./(2.^(m:-1:1));
PEc=(2.^(0:n-1));
PEc(end)=-PEc(end);
Rc=kron([PDc PEc],ones(numel(X),1));
xp=sum(XC.*Rc,2)';
XQnm=xp;
XD=Xaux;
end
 elseif S=='U' || S=='u'
Entero_positivo=2^(n+m)-1;
Entero_negativo=0;
Muestras=X;
indMm=find(Muestras==max(Muestras));
Muestra_mayor=Muestras(indMm(1));
indMm=find(Muestras==min(Muestras));
Muestra_menor=Muestras(indMm(1));
```

```
Max=max(abs(X));
N=log2(Max(1));
if N>=(n+m)
fprintf('\n\t¡Error! no hay suficientes bits para decodificar\n')
fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)

fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
ns=floor(log2(Muestra_M))+1;
Entero_positivo=2^(ns)-1;
Entero_negativo=0;
fprintf('\n\tAumente el valor de n+m a %d\n',ns)
fprintf('\n\tRango de codificación sería: [%f a %f]\n\n',Entero_negativo,Entero_positivo)

X=Muestras;
XQnm=zeros(1,numel(X));
XD=XQnm;
else
XC=de2bi(X,n+m);
% XC=[PD PE];
PDc=1./(2.^(m:-1:1));
PEc=(2.^(0:n-1));
PEc(end)=-PEc(end);
Rc=kron([PDc PEc],ones(numel(X),1));
xp=sum(XC.*Rc,2)';
XQnm=xp;
XD=X;
end
else
fprintf('\n\n\tError, Especifique Signado (S) o no signado (U)\n\n')
end
end      %%Función que decodifica un valor o un vector a tipo float
```

Anexo 7:

Código del programa que lleva a cabo la MAC con valores en representación Qnm tanto en MATLAB como en IAR para ocupar el MSP430G2553.

Primero en MATLAB:

```
%Qnm

clc      %%borra todo el texto de la ventana de comandos, lo que da como resultado una pantalla clara
```

```
clear all  %%elimina todas las variables del espacio de trabajo actual y las libera de la memoria del sistema.

close all  %%Formas de cerrar o salir

n1=input('Ingresa el número de bits para la parte entera = '); %%En la variable n1 se almacena el número de bits para la parte entera de la codificación

m1=input('Ingresa el número de bits para la parte decimal = '); %%En la variable m1 se almacena el número de bits para la parte decimal de la codificación

x1=[0.54,0.67,0.34,-0.64,-0.76,-0.73,0.16,0.95,0.12,-0.05,0.67,0.6,0.34,0.6,-0.77,0.56];  %%Se define un primer vector para realizar la MAC en formato Qnm y float

x2=[0.1,0.65,-0.78,0.35,0.89,-0.37,0.64,0.57,-0.69,-0.26,-0.57,0.43,0.12,0.58,-0.68,0.45]; %%Se define un segundo vector para realizar la MAC en formato Qnm y float

fprintf('La conversión Qnm del primer vector es la siguiente: \n');

[X1 XQnm1 XD1 XDS1]=Cod_Qnm(x1,n1,m1);    %%Función que realiza la codificacion n,m del vector 1

XDS1      %%Se despliega la codificacion Qnm del vector 1

fprintf('La conversión Qnm del segundo vector es la siguiente: \n');

[X2 XQnm2 XD2 XDS2]=Cod_Qnm(x2,n1,m1);    %%Función que realiza la codificacion n,m del vector 2

XDS2      %%Se despliega la codificacion Qnm del vector 2

for(i=1:16) %%Con este bucle se realiza la multiplicación elemento a elemento de cada uno de los vectores en formato Qnm

mult(i)=XDS1(i)*XDS2(i);

end%%acarreo=n1+ceil(log2(length(x)));

fprintf('\n\n\t\tEl valor de la MAC Qnm es el siguiente: \n');
suma=sum(mult) %%Se realiza la suma de todos los elementos de la multiplicación en f ormato tipo Qnm

function [X XQnm XD XDS]=Cod_Qnm(xnTs,n,m)

Entero_positivo=2^(n-1)-1/(2^m);
Entero_negativo=-2^(n-1);
resolucion=1/(2^m);
```

```
Muestras=xnTs;
indMm=find(Muestras==max(Muestras));
Muestra_mayor=Muestras(indMm(1));
indMm=find(Muestras==min(Muestras));
Muestra_menor=Muestras(indMm(1));
if Muestra_mayor>Entero_positivo || Muestra_menor<Entero_negativo
fprintf('\n\t¡Error! no hay suficientes bits para codificar\n')
fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)

fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
If Muestra_M==1
Muestra_M=1.1;
end
ns=ceil(log2(Muestra_M))+1;
Entero_positivo=2^(ns-1)-1/(2^m);
Entero_negativo=-2^(ns-1);
fprintf('\n\tAumente el valor de n a %d\n',ns)
fprintf('\n\tRango de codificación sería: [%f a %f]\n\n',Entero_negativo,Entero_positivo)

 X=Muestras;
 XQnm=zeros(1,numel(X));
 XD=XQnm;
 XDS=XD;
else

% ind=find(abs(xnTs)<1/(2^m)); % busca los elementos que no alcanzan la minima resolución xnTs(ind)=0; ind=find((xnTs)==(2^(n-1))); % Busca los elementos con la máxima resolución xnTs(ind)=(2^(n-1))-(1/(2^m));


signo=(xnTs>=0);
signo=2*signo-1;
xnTsA=xnTs;
xnTs=abs(xnTs);
x1=xnTs/resolucion;
xQnmES=round(x1).*signo; %% Aquí se cambia el criterio truncamiento floor y
 redondeo round
xQnm=xQnmES*(resolucion);
ind=find(xQnmES<0);
xQnmEU=xQnmES;
xQnmEU(ind)=xQnmEU(ind)+2^(n+m);
X=Muestras;
XQnm=xQnm;
XD=xQnmEU;
XDS=xQnmES;
```

```
end
end  %%Función que codifica un valor o un vector a Qnm, recibe los datos así
como los valores de n y m

function [XQnm XD]=Decod_Qnm(X,n,m,S)


if S=='S' || S=='s'
Entero_positivo=2^(n+m-1)-1;
Entero_negativo=-2^(n+m-1);
Muestras=X;
indMm=find(Muestras==max(Muestras));
Muestra_mayor=Muestras(indMm(1));
IndMm=find(Muestras==min(Muestras));
Muestra_menor=Muestras(indMm(1));

Max=max(abs(X));
N=log2(Max(1));
 if N>(n+m-1)
fprintf('\n\t¡Error! no hay suficientes bits para decodificar\n')
fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)
fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
ns=floor(log2(Muestra_M))+1;
Entero_positivo=2^(ns-1);
Entero_negativo=-2^(ns-1);
fprintf('\n\tAumente el valor de n+m a %d\n',ns+1)
fprintf('\n\tRango de codificación sería: [%f a %f]\n\n',Entero_negativo,Entero_positivo)
X=Muestras;
XQnm=zeros(1,numel(X));
XD=XQnm;
else
indc=find(X<0);
Xaux=X;
X(indc)=X(indc)+2^(n+m);
XC=de2bi(X,n+m);
% XC=[PD PE];
PDc=1./(2.^(m:-1:1));
PEc=(2.^(0:n-1));
PEc(end)=-PEc(end);
Rc=kron([PDc PEc],ones(numel(X),1));
xp=sum(XC.*Rc,2)';
XQnm=xp;
XD=Xaux;
end
elseif S=='U' || S=='u'
```

```
 Entero_positivo=2^(n+m)-1;
Entero_negativo=0;
Muestras=X;
indMm=find(Muestras==max(Muestras));
Muestra_mayor=Muestras(indMm(1));
indMm=find(Muestras==min(Muestras));
Muestra_menor=Muestras(indMm(1));

 Max=max(abs(X));
N=log2(Max(1));
if N>=(n+m)
fprintf('\n\t¡Error! no hay suficientes bits para decodificar\n')
fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)
fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
ns=floor(log2(Muestra_M))+1;
Entero_positivo=2^(ns)-1;
Entero_negativo=0;
fprintf('\n\tAumente el valor de n+m a %d\n',ns)
fprintf('\n\tRango de codificación sería: [%f a %f]\n\n',Entero_negativo,Entero_positivo)
X=Muestras;
XQnm=zeros(1,numel(X));
XD=XQnm;
else
XC=de2bi(X,n+m);
% XC=[PD PE];
PDc=1./(2.^(m:-1:1));
PEc=(2.^(0:n-1));
PEc(end)=-PEc(end);
Rc=kron([PDc PEc],ones(numel(X),1));
xp=sum(XC.*Rc,2)';
XQnm=xp;
XD=X;
  end
 else
fprintf('\n\n\tError, Especifique Signado (S) o no signado (U)\n\n')
  end
end      %%Función que decodifica un valor o un vector a tipo float
```

y ahora en IAR desplegado en el LCD para ocupar el MSP430G2553:

```
#include "io430.h"

#include "LCD16x2c.h" //Librería para poder utilizar la pantalla LCD

#include "stdio.h"    //Librería para hacer uso de la función sprintf para desplegar
```

```
los datos en el LCD

char Dato[14]={"Resultado MAC="},resultado[7];
int x1[16]={17,21,11,-20,-24,-23,5,30,4,-2,21,19,11,19,-25,18};  //Se declara
un vector para realizar la MAC en formato float
int x2[16]={3,21,-25,11,28,-12,20,18,-22,-8,-18,14,4,19,-22,14}; //Se declara
un segundo vector para realizar la MAC en formato float
int Xres[16]; //Se declara un vector para almacenar las multiplicaciones de
elemento a elemento de los dos vectores
int acum=0;    //Se declara un acumulador tipo float que se encargara de realizar
la suma de todos los productos

int main( void )
{
int acum=0;    //Se declara un acumulador tipo float que se encargara de realizar
la suma de todos los productos
WDTCTL = WDTPW + WDTHOLD;      //Stop watchdog timer to prevent time out
reset
BCSCTL1 = CALBC1_1MHZ;        //DCO configurado a 1MHz
DCOCTL = CALDCO_1MHZ;

Ini_Lcd();                  //Función para inicializar la pantalla LCD
Cmd_Lcd(0x01);                //Se limpia la pantalla LCD y se coloca en la primer
línea
for(char i=0; i<=13 ;i++)       //Se despliega el mensaje "Resultado MAC" con
ayuda de la función Dato_Lcd
{
Dato_Lcd(Dato[i]);
  }
```

```
 while(1){  //While infinito


acum=0;                    //Se inicializa el acumulador con valor 0

for(char n=0; n<16; n++)    //Se realiza la operación de multiplicación de elementoa elemento de los dos vectores

 {

 Xres[n]=x1[n]*x2[n];

}

for(char  k=0;  k<16;  k++)          //Se realiza la suma de todos los productos previamente calculados y almacenados

 {

acum+=Xres[k];

}

sprintf(resultado,"%u",acum); //Compone una cadena con el valor de la MAC, el contenido se almacena como cadena en resultado

 Cmd_Lcd(0xC0);                  //El cursor de la pantalla LCD se posiciona en la segunda línea de la pantalla LCD

for(char i=0; i<=4 ;i++)  //Se hace el despliegue del valor de la suma en la segunda línea de la pantalla LCD con ayuda de la función Dato_Lcd

  {

 if(resultado[i]!=0){Dato_Lcd(resultado[i]);}

 }

__delay_cycles(1000000);       //Se realiza una pausa de 1 segundo para visualizar el valor

}

}
```

Anexo 8:

Código del programa que lleva a cabo la MAC con valores en representación Qnm junto con el código utilizado para que las entradas fueran introducidas mediante Cordic, se realizaron en MATLAB y en IAR con lenguaje C.

Primero el realizado en MATLAB:

```
%%Simulacion de CORDIC rotaciónal

clc      %%borra todo el texto de la ventana de comandos, lo que da como resultado una pantalla clara

clear all  %%elimina todas las variables del espacio de trabajo actual y las libera de la memoria del sistema.

close all  %%Formas de cerrar o salir

fact2=[0.894427 0.867723 0.861023 0.859346 0.8592415 0.85921531 0.8592087 0.85920711 0.8592067 0.8592066 0.8592065 0.85920656]; %%Diferentes valores para el valor de la ganancia K en diferentes iteraciones

angles=[atand(2^-1) atand(2^-2) atand(2^-3) atand(2^-4) atand(2^-5) atand(2^-6) atand(2^-7) atand(2^-8) atand(2^-9) atand(2^-10) atand(2^-11) atand(2^-12)]; %%Diferentes valores para el valor del ángulo de rotación en diferentes iteraciones

ds=[2^-1 2^-2 2^-3 2^-4 2^-5 2^-6 2^-7 2^-8 2^-9 2^-10 2^-11 2^-12];

[X1 XQnm1 XD1 XDS1]=Cod_Qnm(fact2,1,10); %%Función que realiza la codificacion n,m del vector 1

[X2 XQnm2 XD2 XDS2]=Cod_Qnm(angles,6,8);  %%Función que realiza la codificacion n,m del vector 2

PR=input('\nIntroduce el valor de la parte real del número complejo = '); %%En la variable PR se almacena el valor de la parte real del número complejo que se desea aproximar por CORDIC

PI=input('\nIntroduce el valor de la parte imaginaria del número complejo = '); %%En la variable PI se almacena el valor de la parte imaginaria del número complejo que se desea aproximar por CORDIC

rot=input('\nIntroduce el valor del ángulo de rotación(?) = '); %%En la variable rot se almacena el valor del ángulo de rotación al que se desea aproximar el número complejo

[X4 XQnm4 XD4 XDS4]=Cod_Qnm(PR,2,8); %%Función que realiza la codificacion n,m del valor de la parte real

[X5 XQnm5 XD5 XDS5]=Cod_Qnm(PI,2,8); %%Función que realiza la codificacion n,m del valor de la parte imaginaria

[X6 XQnm6 XD6 XDS6]=Cod_Qnm(rot,6,8); %%Función que realiza la codificacion n,m del valor del ángulo inicial de rotación

RES=(PR+i*PI)*exp(i*rot*pi/180); %%Se calcula el Valor real de laaproximación multiplicando el número complejo por la exponencial compleja del ángulo de rotación
```

```
[X7 XQnm7 XD7 XDS7]=Cod_Qnm(real(RES),3,18); %%Función que realiza la codificacion n,m del valor de la parte real del número flotante

[X8 XQnm8 XD8 XDS8]=Cod_Qnm(imag(RES),3,18); %%Función que realiza la codificacion n,m del valor de la parte imaginaria del número flotante

ifin=input('\nIndica el número de iteraciones = '); %%En la variable ifin se almacena el número de iteraciones que indique el usuario

xant=XDS4; %%Se inicializa la variable correspondiente a la componente x-1 con la parte real del número complejo inicial

yant=XDS5; %%Se inicializa la variable correspondiente a la componente y-1 con la parte imaginaria del número complejo inicial

zant=XDS6; %%Se inicializa la variable correspondiente al ángulo de rotación con el ángulo al que se desea rotar el número complejo inicial

d=(2*(rot>0)-1); %%Se hace el cálculo del signo inicial para realizar la rotación del número complejo hacia el ángulo deseado

for(i=1:ifin)

x=(xant-floor(yant*d*ds(i))); %%Se calcula la nueva componente x con la formula indicada

y=(yant+floor(xant*d*ds(i))); %%Se calcula la nueva componente y con la formula indicada

 z=zant-d*(XDS2(i)); %%Se calcula el nuevo ángulo de rotación con la formula indicada

 xant=x; %%Se sobreescribe la componente x-1 con la componente calculada para ser utilizada en la siguiente iteración

 yant=y; %%Se sobreescribe la componente y-1 con la componente calculada para ser utilizada en la siguiente iteración

zant=z; %%Se sobreescribe el ángulo de rotación z-1 con la componente calculada para ser utilizada en la siguiente iteración

d=(2*(z>0)-1); %%Se hace el cálculo del signo para saber a donde hay que rotar el vector nuevamente (izquierda o derecha

end

x=XDS1(ifin)*x; %%El resultado de la componente x se multiplica por su factor de ganancia para el ajuste

y=XDS1(ifin)*y; %%El resultado de la componente x se multiplica por su factor de ganancia para el ajuste
```

```
[XQnm XDx]=Decod_Qnm(x,3,18,'S'); %%Se realiza la decodificacion de la componente x aproximado por cordic y Qnm

[YQnm XDy]=Decod_Qnm(y,3,18,'S'); %%Se realiza la decodificacion de la componente y aproximado por cordic y Qnm

[ZQnm ZDz]=Decod_Qnm(z,6,8,'S'); %%Se realiza la decodificacion del ángulo resudial

fprintf('\n\n_____RESULTADO FLOAT_____');

fprintf('\n\n\tEl valor aproximado es = %f + i%f , con un ángulo de = %f',XQnm,YQnm,ZQnm);      %%Se hace el despligue de la aproximación por CORDIC

fprintf('\n\tEl valor real es = %f + i%f',real(RES),real(RES));                %%Se hace el despligue del número complejo real para la comparativa

fprintf('\n\n_____RESULTADO Qnm_____');

fprintf('\n\n\tEl valor aproximado es = %d + i%d , con un ángulo de = %d',x,y,z); %%Se hace el despligue de la aproximación por CORDIC

fprintf('\n\tEl valor real es = %d + i%d',XDS7,XDS8);                %%Se hace el despligue del número complejo real para la comparativa

function [X XQnm XD XDS]=Cod_Qnm(xnTs,n,m)

Entero_positivo=2^(n-1)-1/(2^m);

Entero_negativo=-2^(n-1);

resolucion=1/(2^m);

Muestras=xnTs;

indMm=find(Muestras==max(Muestras));

Muestra_mayor=Muestras(indMm(1));

indMm=find(Muestras==min(Muestras));
Muestra_menor=Muestras(indMm(1));
if Muestra_mayor>Entero_positivo || Muestra_menor<Entero_negativo
 fprintf('\n\t¡Error! no hay suficientes bits para codificar\n')
fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)
fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
if Muestra_M==1
Muestra_M=1.1;
```

```
 end
ns=ceil(log2(Muestra_M))+1;
Entero_positivo=2^(ns-1)-1/(2^m);
Entero_negativo=-2^(ns-1);
fprintf('\n\tAumente el valor de n a %d\n',ns)
fprintf('\n\tRango de codificación sería: [%f a %f]\n\n',Entero_negativo,Entero_positivo)

 X=Muestras;
```

Diseñar una unidad MAC compleja (Multiply–accumulate operation) en el microcontrolador MSP430G2553 utilizando el algoritmo CORDIC para procesos y aplicaciones de PDS.

```
X        Qnm=zeros(1,numel(X));

XD=XQnm;

XDS=XD;

Else

% ind=find(abs(xnTs)<1/(2^m)); % busca los elementos que no alcanzan la minima resolución %  xnTs(ind)=0;

%  ind=find((xnTs)==(2^(n-1)));  % Busca los elementos con la maxima resolución
%  xnTs(ind)=(2^(n-1))-(1/(2^m));


signo=(xnTs>=0);
signo=2*signo-1;
xnTsA=xnTs;
xnTs=abs(xnTs);
x1=xnTs/resolucion;
xQnmES=floor(x1).*signo; %% Aquí se cambia el criterio truncamiento
 floor y redondeo round
xQnm=xQnmES*(resolucion);
ind=find(xQnmES<0);
xQnmEU=xQnmES;
xQnmEU(ind)=xQnmEU(ind)+2^(n+m);
X=Muestras;
XQnm=xQnm;
XD=xQnmEU;
XDS=xQnmES;
end
end  %%Función que codifica un valor o un vector a Qnm, recibe los datos
así como los valores de n y m

function [XQnm XD]=Decod_Qnm(X,n,m,S)
```

```
if S=='S' || S=='s'
Entero_positivo=2^(n+m-1)-1;
Entero_negativo=-2^(n+m-1);
Muestras=X;
indMm=find(Muestras==max(Muestras));
Muestra_mayor=Muestras(indMm(1));
indMm=find(Muestras==min(Muestras));
Muestra_menor=Muestras(indMm(1));

Max=max(abs(X));
N=log2(Max(1));
  if N>(n+m-1)
fprintf('\n\t¡Error! no hay suficientes bits para decodificar\n')
fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)
fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
ns=floor(log2(Muestra_M))+1;
Entero_positivo=2^(ns-1);
Entero_negativo=-2^(ns-1);
fprintf('\n\tAumente el valor de n+m a %d\n',ns+1)
fprintf('\n\tRango de codificación sería: [%f a %f]\n\n',Entero_negativo,Entero_
 positivo)
X=Muestras;
XQnm=zeros(1,numel(X));
XD=XQnm;
else
indc=find(X<0);
Xaux=X;
X(indc)=X(indc)+2^(n+m);
XC=de2bi(X,n+m);
% XC=[PD PE];
PDc=1./(2.^(m:-1:1));
PEc=(2.^(0:n-1));
PEc(end)=-PEc(end);
Rc=kron([PDc PEc],ones(numel(X),1));
xp=sum(XC.*Rc,2)';
XQnm=xp;
XD=Xaux;
end
elseif S=='U' || S=='u'
Entero_positivo=2^(n+m)-1;
Entero_negativo=0;
Muestras=X;
indMm=find(Muestras==max(Muestras));
Muestra_mayor=Muestras(indMm(1));
```

```
indMm=find(Muestras==min(Muestras));
Muestra_menor=Muestras(indMm(1));

Max=max(abs(X));
N=log2(Max(1));
if N>=(n+m)
fprintf('\n\t¡Error! no hay suficientes bits para decodificar\n')
fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)
fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
ns=floor(log2(Muestra_M))+1;
Entero_positivo=2^(ns)-1;
Entero_negativo=0;
fprintf('\n\tAumente el valor de n+m a %d\n',ns)
fprintf('\n\tRango de codificación sería: [%f a %f]\n\n',Entero_negativo,Entero
_positivo)
X=Muestras;
XQnm=zeros(1,numel(X));
XD=XQnm;
 else
XC=de2bi(X,n+m);
% XC=[PD PE];
PDc=1./(2.^(m:-1:1));
PEc=(2.^(0:n-1));
PEc(end)=-PEc(end);
Rc=kron([PDc PEc],ones(numel(X),1));
xp=sum(XC.*Rc,2)';
XQnm=xp;
XD=X;
end
else
fprintf('\n\n\tError, Especifique Signado (S) o no signado (U)\n\n')
end
end      %%Función que decodifica un valor o un vector a tipo float
```

Y ahora el realizado en IAR para poder ser desplegado en un LCD mediante el uso del MSP430G2553:

```
#include "io430.h"

#include "LCD16x2c.h" //Librería para poder utilizar la pantalla LCD

#include "stdio.h"    //Librería para hacer uso de la función sprintf para desplegar
los datos en el LCD

#include "math.h"
```

```
char msj1[]={"\r\nParteReal= "};

char msj2[]={"\r\nParteImaginaria= "};

char msj3[]={"\r\nÁnguloResidual= "};

char Dato[16]={" PR PI ANG"},resultadoreal[6],resultadoimag[6],resultadoangle[6]; //Vectores que almacenaran los valores de los vectores para el despliegue

signed long int fact[12]={915,888,881,879,879,879,879,879,879,879,879,879}; //Diferentes valores para el valor de la ganancia K en diferentes iteraciones

signed long int angles[12]={6800,3593,1824,915,458,229,114,57,28,14,7,3}; //Diferentes valores para los distintos ángulos de rotación

signed long int ds[12]={2,4,8,16,32,64,128,256,512,1024,2048,4096};

signed long int PR=256,s,xant=0,yant=0,zant=0,x=0,y=0,z=0,xres,yres; //En la variable PR se almacena el valor de la parte real del número complejo que se desea aproximar por CORDIC

signed long int PI=0; //En la variable PI se almacena el valor de la parte imaginaria del número complejo que se desea aproximar por CORDIC

signed long int rot=2560; //En la variable rot se almacena el valor del ángulo inicial que se desea aproximar por CORDIC

int ifin=10; //En la variable ifin se almacena el número de iteraciones que indique el usuario

int main( void )

{
// Stop watchdog timer to prevent time out reset

 WDTCTL = WDTPW + WDTHOLD;

 BCSCTL1 = CALBC1_1MHZ;        //DCO configurado a 1MHz

 DCOCTL = CALDCO_1MHZ;

 //CONFIGURACIÓN DEL MODULO UART

P1SEL  |= BIT1 + BIT2 ;             // Configurando las terminales P1.1 y P1.2

P1SEL2 |= BIT1 + BIT2 ;             // como parte del sistema UART

P1DIR &= ~BIT1;                    // P1.1 <-- RXD

P1DIR |= BIT2;                    // P1.2 --> TXD

UCA0CTL1 |= UCSSEL_2;               // SMCLK
```

```
//.........9600bps.....................................................
UCA0BR0 = 104;                    // 1MHz 9600
UCA0BR1 = 0;                      // 1MHz 9600
UCA0MCTL = UCBRS0;                // Modulation UCBRSx = 1
//.....................................................................
UCA0CTL1 &= ~UCSWRST;             // **Initialize USCI state machine**
Ini_Lcd();                  //Función para inicializar la pantalla LCD
 Cmd_Lcd(0x01);                 //Se limpia la pantalla LCD y se coloca en la primer línea
for(char i=0; i<=15 ;i++)     //Se despliega el mensaje "Resultado MAC" con ayuda de la función Dato_Lcd
{
Dato_Lcd(Dato[i]);  //Se imprime el mensaje "  PR    PI  ANG"
}

 while(1)
{
xant=PR;  //Se inicializa la variable correspondiente a la componente x-1 con la parte real del número complejo inicial
yant=PI;  //Se inicializa la variable correspondiente a la componente y-1 con la parte imaginaria del número complejo inicial
zant=rot; //Se inicializa la variable correspondiente al ángulo de rotación con el ángulo al que se desea rotar el número complejo inicial
if(rot>=0){s=1;}else{s=-1;}; //Se hace el cálculo del signo inicial para realizar la rotación del número complejo hacia el ángulo deseado
for(char i=0;i<ifin;i++)
{
x=(xant-(yant*s)/ds[i]);     //Se calcula la nueva componente x con la formula indicada
y=(yant+(xant*s)/ds[i]);     //Se calcula la nueva componente y con la formula indicada
z=zant-s*(angles[i]);          //Se calcula el nuevo ángulo de rotación con la formula indicada
xant=x;                       //Se sobreescribe la componente x-1 con la componente calculada para ser utilizada en la siguiente iteración
```

```
yant=y;                    //Se sobreescribe la componente y-1 con la componente calculada para ser utilizada en la siguiente iteración

 zant=z;                   //Se sobreescribe el ángulo de rotación z-1 con la componente calculada para ser utilizada en la siguiente iteración

 if(z>=0){s=1;}else{s=-1;};    //Se hace el cálculo del signo para saber a donde hay que rotar el vector nuevamente (izquierda o derecha)

}
```

Diseñar una unidad MAC compleja (Multiply–accumulate operation) en el microcontrolador MSP430G2553 utilizando el algoritmo CORDIC para procesos y aplicaciones de PDS.

```
xres=x*fact[ifin-1]; //El resultado de la componente x se multiplica por su factor de ganancia para el ajuste

yres=y*fact[ifin-1]; //El resultado de la componente y se multiplica por su factor de ganancia para el ajuste

 sprintf(resultadoreal,"%ld",xres); //Compone una cadena con el valor de la componente x, el contenido se almacena como cadena en resultado

sprintf(resultadoimag," %ld",yres); //Compone una cadena con el valor de la componente y, el contenido se almacena como cadena en resultado

sprintf(resultadoangle," %ld\n",z); //Compone una cadena con el valor del ángulo residual, el contenido se almacena como cadena en resultado

Cmd_Lcd(0xC0);             //El cursor de la pantalla LCD se posiciona en la segunda línea de la pantalla LCD

 for(char i=0; i<sizeof resultadoreal  ;i++)     //Se hace el despliegue del valor de la componente x en la segunda línea de la pantalla LCD con ayuda de la función Dato_Lcd

{
 if(resultadoreal[i]!=0){Dato_Lcd(resultadoreal[i]);}
 }
Cmd_Lcd(0xC6);             //El cursor de la pantalla LCD se posiciona en la segunda línea de la pantalla LCD

for(char i=0; i<sizeof resultadoimag ;i++)     //Se hace el despliegue del valor de la componente y en la segunda línea de la pantalla LCD con ayuda de la función Dato_Lcd

{
if(resultadoimag[i]!=0){Dato_Lcd(resultadoimag[i]);}
}
```

```
Cmd_Lcd(0xCB);            //El cursor de la pantalla LCD se posiciona en la segunda línea de la pantalla LCD

for(char i=0; i<sizeof resultadoangle ;i++)       //Se hace el despliegue del ángulo residual en la segunda línea de la pantalla LCD con ayuda de la función Dato_Lcd

 {
if(resultadoangle[i]!=0){Dato_Lcd(resultadoangle[i]);}
}
__delay_cycles(5000000);       //Se realiza una pausa de 1 segundo para visualizar el valor

for(char i = 0; i <= sizeof msj1 -1 ; i++)

{
UCA0TXBUF = msj1[i];

while (!(IFG2&UCA0TXIFG)){}

}
for(char i = 0; i <= sizeof resultadoreal -1 ; i++)
{
UCA0TXBUF = resultadoreal[i];

while (!(IFG2&UCA0TXIFG)){}

 }
for(char i = 0; i <= sizeof msj2 -1 ; i++)
{
  UCA0TXBUF = msj2[i];
 while (!(IFG2&UCA0TXIFG)){}
 }
for(char i = 0; i <= sizeof resultadoimag -1 ; i++)
{
UCA0TXBUF = resultadoimag[i];
while (!(IFG2&UCA0TXIFG)){}
}
for(char i = 0; i <= sizeof msj3 -1 ; i++)
{
 UCA0TXBUF = msj3[i];
 while (!(IFG2&UCA0TXIFG)){}
}
for(char i = 0; i <= sizeof resultadoangle -1 ; i++)
{
UCA0TXBUF = resultadoangle[i];
while (!(IFG2&UCA0TXIFG)){}
 }
__delay_cycles(1000000); //Se realiza una pausa de 1 segundo para visualizar el valor
```

Anexo 9:

Código del programa que lleva a cabo la MAC compleja obteniendo un resultado en formato flotante y un resultado en formato Qnm utilizando el software de desarrollo llamado MATLAB:

```
%%MAC COMPLEJA

clc  %%borra todo el texto de la ventana de comandos, lo que da como resultadouna pantalla clara

clear all  %%elimina todas las variables del espacio de trabajo actual y las libera de la memoria del sistema.

close all  %%Formas de cerrar o salir

XR=[0.54 0.67 0.34 -0.64 -0.76 -0.73 0.16 0.95]; %%Se define un primer vector que almacena la parte real del vector X para realizar la MAC compleja en formato Qnm y float

XI=[0.1 0.65 -0.78 0.35 0.89 -0.37 0.64 0.57]; %%Se define un segundo vector que almacena la parte imaginaria del vector X para realizar la MAC compleja en formato Qnm y float

YR=[0.12 -0.05 0.67 0.6 0.34 0.6 -0.77 0.56];  %%Se define un tercer vector que almacena la parte real del vector Y para realizar la MAC compleja en formato Qnm y float

YI=[-0.69 -0.26 -0.57 0.43 0.12 0.58 -0.68 0.45]; %%Se define un cuarto vector que almacena la parte imaginaria del vector Y para realizar la MAC compleja en formato Qnm y float

for(i=1:8)

 Rf(i)=XR(i)*YR(i)-XI(i)*YI(i); %%Se hacen las operaciones correspondientes a la parte real en formato float

 If(i)=XR(i)*YI(i)+XI(i)*YR(i); %%Se hacen las operaciones correspondientes a la parte imaginaria en formato float

End

RRf=sum(Rf);  %%Se realiza la suma de las partes reales de cada producto para la MAC en formato float

RIf=sum(If); %%Se realiza la suma de las partes imaginarias de cada producto para la MAC en formato float

[Rf1 RfQnm1 RfD1 RfDS1]=Cod_Qnm(RRf,6,10); %%Función que realiza la codificacion 3,10 de la parte real del resultado float
```

```
[If1 IfQnm1 IfD1 IfDS1]=Cod_Qnm(RIf,6,10);    %%Función que realiza la codificacion 3,10 de la parte imaginaria del resultado float

[X1 XQnm1 XD1 XDS1]=Cod_Qnm(XR,1,5);    %%Función que realiza la codificacion 1,5 de la parte real del vector X

[X2 XQnm2 XD2 XDS2]=Cod_Qnm(XI,1,5);    %%Función que realiza la codificacion 1,5 de la parte imaginaria del vector X

[Y1 YQnm1 YD1 YDS1]=Cod_Qnm(YR,1,5);    %%Función que realiza la codificacion 1,5 de la parte real del vector Y

[Y2 YQnm2 YD2 YDS2]=Cod_Qnm(YI,1,5);    %%Función que realiza la codificacion 1,5 de la parte imaginaria del vector Y

for(i=1:8)

  R(i)=XDS1(i)*YDS1(i)-XDS2(i)*YDS2(i);      %%Se hacen las operaciones correspondientes a la parte real en Qnm

  I(i)=XDS1(i)*YDS2(i)+XDS2(i)*YDS1(i);      %%Se   hacen las operaciones correspondientes a la parte imaginaria en Qnm

End

RR=sum(R); %%Se realiza la suma de las partes reales de cada producto para la MAC en Qnm

RI=sum(I); %%Se realiza la suma de las partes imaginarias de cada producto para la MAC en Qnm

[RRQnm RRD]=Decod_Qnm(RR,6,10,'S'); %%Se realiza la decodificacion de la parte real de la MAC compleja del resultado

[RIQnm RID]=Decod_Qnm(RI,6,10,'S'); %%Se realiza la decodificacion de la parte imaginaria de la MAC compleja del resultado

fprintf('\n\n_____RESULTADO FLOAT_____');

fprintf('\n\n\tEl valor aproximado es = %f + i%f',RRQnm,RIQnm);    %%Se hace el despligue del número complejo resultante aproximado por Qnm decodificado

fprintf('\n\tEl valor real es = %f + i%f',RRf,RIf);                %%Se hace el despligue del número complejo real para la comparativa

fprintf('\n\n_____RESULTADO Qnm_____');

fprintf('\n\n\tEl valor aproximado es = %d + i%d',RR,RI);      %%Se hace el despligue del número complejo resultante aproximado por Qnm
```

```
fprintf('\n\tEl valor real es = %d + i%d',RfDS1,IfDS1);                %%Se hace el despligue del número complejo real para la comparativa
function [X XQnm XD XDS]=Cod_Qnm(xnTs,n,m)
Entero_positivo=2^(n-1)-1/(2^m);
Entero_negativo=-2^(n-1);
resolucion=1/(2^m);
Muestras=xnTs;
indMm=find(Muestras==max(Muestras));
Muestra_mayor=Muestras(indMm(1));
indMm=find(Muestras==min(Muestras));
Muestra_menor=Muestras(indMm(1));
if Muestra_mayor>Entero_positivo || Muestra_menor<Entero_negativo
 fprintf('\n\t¡Error! no hay suficientes bits para codificar\n')
fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)
 fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
 Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
 if Muestra_M==1
Muestra_M=1.1;
End
ns=ceil(log2(Muestra_M))+1;
Entero_positivo=2^(ns-1)-1/(2^m);
Entero_negativo=-2^(ns-1);
fprintf('\n\tAumente el valor de n a %d\n',ns)
fprintf('\n\tRango de codificación sería: [%f a%f]\n\n',Entero_negativo,Entero_positivo)
 X=Muestras;
XQnm=zeros(1,numel(X));
XD=XQnm;
 XDS=XD;
```

```
Else
%     ind=find(abs(xnTs)<1/(2^m));     % busca los elementos que no alcanzan
la minima resolución
%     xnTs(ind)=0;
%       ind=find((xnTs)==(2^(n-1)));       % Busca los elementos con la maxima
resolución
%     xnTs(ind)=(2^(n-1))-(1/(2^m));
signo=(xnTs>=0);
signo=2*signo-1;
xnTsA=xnTs;
xnTs=abs(xnTs);
x1=xnTs/resolucion;
xQnmES=floor(x1).*signo; %% Aquí se cambia el criterio truncamiento floor y
redondeo round
 xQnm=xQnmES*(resolucion);
ind=find(xQnmES<0);
xQnmEU=xQnmES;
xQnmEU(ind)=xQnmEU(ind)+2^(n+m);
X=Muestras;
XQnm=xQnm;
XD=xQnmEU;
XDS=xQnmES;
End
end  %%Función que codifica un valor o un vector a Qnm, recibe los datos así como
los valores de n y m
function [XQnm XD]=Decod_Qnm(X,n,m,S)
 if S=='S' || S=='s'
Entero_positivo=2^(n+m-1)-1;
Entero_negativo=-2^(n+m-1);
```

```
Muestras=X;
indMm=find(Muestras==max(Muestras));
Muestra_mayor=Muestras(indMm(1));
indMm=find(Muestras==min(Muestras));
Muestra_menor=Muestras(indMm(1));
Max=max(abs(X));
 N=log2(Max(1));
if N>(n+m-1)
 fprintf('\n\t¡Error! no hay suficientes bits para decodificar\n')
fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)
fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
ns=floor(log2(Muestra_M))+1;
Entero_positivo=2^(ns-1);
Entero_negativo=-2^(ns-1);
fprintf('\n\tAumente el valor de n+m a %d\n',ns+1)
fprintf('\n\tRango de codificación sería: [%f a %f]\n\n',Entero_negativo,Entero_positivo)
 X=Muestras;
XQnm=zeros(1,numel(X));
XD=XQnm;
Else
indc=find(X<0);
Xaux=X;
 X(indc)=X(indc)+2^(n+m);
XC=de2bi(X,n+m);
% XC=[PD PE];
PDc=1./(2.^(m:-1:1));
PEc=(2.^(0:n-1));
```

```
PEc(end)=-PEc(end);
Rc=kron([PDc PEc],ones(numel(X),1));
xp=sum(XC.*Rc,2)';
XQnm=xp;
XD=Xaux;
End
elseif S=='U' || S=='u'
Entero_positivo=2^(n+m)-1;
Entero_negativo=0;
Muestras=X;
indMm=find(Muestras==max(Muestras));
Muestra_mayor=Muestras(indMm(1));
indMm=find(Muestras==min(Muestras));
Muestra_menor=Muestras(indMm(1));
Max=max(abs(X));
N=log2(Max(1));
if N>=(n+m)
fprintf('\n\t¡Error! no hay suficientes bits para decodificar\n'
fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)
fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
ns=floor(log2(Muestra_M))+1;
Entero_positivo=2^(ns)-1;
Entero_negativo=0;
fprintf('\n\tAumente el valor de n+m a %d\n',ns)
fprintf('\n\tRango de codificación sería: [%f a %f]\n\n',Entero_negativo,Entero_positivo)
X=Muestras;
XQnm=zeros(1,numel(X));
```

```
XD=XQnm;

else

XC=de2bi(X,n+m);
% XC=[PD PE];
PDc=1./(2.^(m:-1:1));
PEc=(2.^(0:n-1));
PEc(end)=-PEc(end);
Rc=kron([PDc PEc],ones(numel(X),1));
xp=sum(XC.*Rc,2)';
XQnm=xp;
XD=X;
end
else
fprintf('\n\n\tError, Especifique Signado (S) o no signado (U)\n\n')
end
end      %%Función que decodifica un valor o un vector a tipo float
```

Anexo 10:

Código del programa que lleva a cabo la MAC compleja obteniendo un resultado en formato flotante y un resultado en formato Qnm utilizando el software de desarrollo llamado MATLAB:

```
//CODIGO DE LA MAC COMPLEJA EN EL MSP430

#include "io430.h"

#include "stdio.h"  //librería para hacer uso de la función sprintf para desplegar los datos en el LCD


char msj1[]={"\r\n\nParteReal= "};

char msj2[]={"\r\nParteImaginaria= "};

char resultadoreal[6],resultadoimag[6]; //Cadena de caracteres que almacenaran los valores de los vectores para el despliegue

signed long int XR[8]={17,21,10,-20,-24,-23,5,30}; //Se define un primer vector que almacena la parte real del vector X para realizar la MAC compleja en formato Qnm

signed long int XI[8]={3,20,-24,11,28,-11,20,18}; //Se define un primer vector que almacena la parte imaginaria del vector X para realizar la MAC compleja en formato Qnm
```

```
signed long int YR[8]={3,-1,21,19,10,19,-24,17}; //Se define un primer vector que almacena la parte real del vector Y para realizar la MAC compleja en formatoQnm

signed long int YI[8]={-22,-8,-18,13,3,18,-21,14}; //ESe define un primer vector que almacena la parte imaginaria del vector Y para realizar la MAC compleja en formato Qnm

signed long int R[8],I[8],RR=0,RI=0; //En el vector R se almacenan todas las partes reales de los productos así como en el vector I, y en RR y RI se almacenaran los resultados



int main( void )

{

 // Stop watchdog timer to prevent time out reset

 WDTCTL = WDTPW + WDTHOLD;

 BCSCTL1 = CALBC1_1MHZ;        //DCO configurado a 1MHz

 DCOCTL = CALDCO_1MHZ;


 //CONFIGURACIÓN DEL MODULO UART

 P1SEL  |= BIT1 + BIT2 ;            // Configurando las terminales P1.1 y P1.2

 P1SEL2 |= BIT1 + BIT2 ;             // como parte del sistema UART

 P1DIR &= ~BIT1;                  // P1.1 <-- RXD

 P1DIR |= BIT2;                  // P1.2 --> TXD

 UCA0CTL1 |= UCSSEL_2;              // SMCLK

 //.....9600bps.....

 UCA0BR0 = 104;                  // 1MHz 9600

 UCA0BR1 = 0;                  // 1MHz 9600

 UCA0MCTL = UCBRS0;              // Modulation UCBRSx = 1

 //.....
```

```
 UCA0CTL1 &= ~UCSWRST;                  // **Initialize USCI state machine**


  while(1)
  {
    RR=0;  //Se inicializan las variables que almacenaran el resultado
    RI=0;
    for(char i=0;i<8;i++) //Se realiza el cálculo de cada una de las partes reales e imaginarias de los productos
    {
      R[i]=XR[i]*YR[i]-XI[i]*YI[i];
      I[i]=XR[i]*YI[i]+XI[i]*YR[i];
    }
    for(char i=0;i<8;i++) //Se realiza la suma de cada una de todas las partes reales e imaginarias de los productos
    {
      RR=RR+R[i];
      RI=RI+I[i];
    }
  sprintf(resultadoreal,"%ld",RR);  //Compone una cadena con el valor real de la MAC compleja, el contenido se almacena como cadena en resultado real
 sprintf(resultadoimag," %ld",RI); //Compone una cadena con el valor imaginario de la MAC compleja y, el contenido se almacena como cadena en resultado imaginario
for(char i = 0; i <= sizeof msj1 -1 ; i++) //Se hace el despliegue de todos los datos por el puerto serial con ayuda de TERA TERM
{
        UCA0TXBUF = msj1[i];
        while (!(IFG2&UCA0TXIFG)){}
 }
```

```
      for(char i = 0; i <= sizeof resultadoreal -1 ; i++)
      {
       UCA0TXBUF = resultadoreal[i];
        while (!(IFG2&UCA0TXIFG)){}
      }
      for(char i = 0; i <= sizeof msj2 -1 ; i++)
      {
        UCA0TXBUF = msj2[i];
        while (!(IFG2&UCA0TXIFG)){}
      }
      for(char i = 0; i <= sizeof resultadoimag -1 ; i++)
      {
       UCA0TXBUF = resultadoimag[i];
       while (!(IFG2&UCA0TXIFG)){}
      }
    __delay_cycles(5000000); //Se realiza una pausa de 5 segundos para visualizar el valor
  }
}
```

Anexo 11:

Código para la MAC compleja con funciones trigonométricas en formato flotante:

```
%%MAC compleja PRUEBA 1
clc      %%borra todo el texto de la ventana de comandos, lo que da como resultado una pantalla clara
clear all  %%elimina todas las variables del espacio de trabajo actual y las libera de la memoria del sistema.
close all  %%Formas de cerrar o salir
```

```
%%_____ROTACIÓNES_____
__
rotaciones=[10 1.78 -15.7];

for(m=1:3) %%Se realiza el cálculo del número complejo con las distintas rotaciones

YR(m)=cos(rotaciones(m)*pi/180);
YI(m)=sin(rotaciones(m)*pi/180);
end
YR
YI
%%_____MACOMPLEJA_____
__

XR=[-0.14 0.07 -0.34]; %%Se define un primer vector que almacena la parte real del vector X para realizar la MAC compleja en formato Qnm y float

XI=[0.1 0.65 -0.78];     %%Se define un segundo vector que almacena la parte imaginaria del vector X para realizar la MAC compleja en formato Qnm y float

for(i=1:3)

R(i)=XR(i)*YR(i)-XI(i)*YI(i);  %%Se hacen las operaciones correspondientes a la parte real en float

I(i)=XR(i)*YI(i)+XI(i)*YR(i);  %%Se hacen las operaciones correspondientes a la parte imaginaria en float

End

RR=sum(R); %%Se realiza la suma de las partes reales de cada producto para la MAC en float

RI=sum(I); %%Se realiza la suma de las partes imaginarias de cada producto para la MAC en float

fprintf('\n\n_____RESULTADO FLOAT_____');

fprintf('\n\n\tEl valor de la MAC compleja es = %f + i%f',RR,RI);    %%Se hace el despligue de la MAC compleja en formato float
```

Anexo 12:

Código para la MAC compleja con el algoritmo Cordic en formato flotante:

```
%%MAC compleja PRUEBA 2 (CORDIC)

clc      %%borra todo el texto de la ventana de comandos, lo que da como resultado una pantalla clara

clear all  %%elimina todas las variables del espacio de trabajo actual y las libera de la memoria del sistema.

close all  %%Formas de cerrar o salir

%%_____CORDIC_____
_____

fact2=[0.894427 0.867723 0.861023 0.859346 0.8592415 0.85921531 0.8592087 0.85920711 0.8592067 0.8592066 0.8592065 0.85920656]; %%Diferentes valores para el valor de la ganancia K en diferentes iteraciones

angles=[atand(2^-1) atand(2^-2) atand(2^-3) atand(2^-4) atand(2^-5) atand(2^-6) atand(2^-7) atand(2^-8) atand(2^-9) atand(2^-10) atand(2^-11) atand(2^-12)]; %%Diferentes valores para el valor del ángulo de rotación en diferentes iteraciones

ds=[2^-1 2^-2 2^-3 2^-4 2^-5 2^-6 2^-7 2^-8 2^-9 2^-10 2^-11 2^-12]; %%Vector que almacena los corrimientos logicos hacia la derecha

rotaciones=[10 1.78 -15.7]; %%Vector que almacena las diferentes rotaciones a aproximar por CORDIC

PR=input('\nIntroduce el valor de la parte real del número complejo = '); %%En la variable PR se almacena el valor de la parte real del número complejo que se desea aproximar por CORDIC

PI=input('\nIntroduce el valor de la parte imaginaria del número complejo = '); %%En la variable PI se almacena el valor de la parte imaginaria del número complejo que se desea aproximar por CORDIC

ifin=input('\nIndica el número de iteraciones = ');  %%En la variable ifin se almacena el número de iteraciones que indique el usuario

for(k=1:3)

xant=PR; %%Se inicializa la variable correspondiente a la componente x-1 con la parte real del número complejo inicial

yant=PI; %%Se inicializa la variable correspondiente a la componente y-1 con la parte imaginaria del número complejo inicial
```

```
zant=rotaciones(k); %%Se inicializa la  variable correspondiente al ángulo de rotación con el ángulo al que se desea rotar el número complejo inicial

d=(2*(zant>0)-1);    %%Se hace el cálculo del signo inicial para realizar la rotación del número complejo hacia el ángulo deseado

 for(i=1:ifin)

x(k)=(xant-(yant*d*ds(i)));  %%Se calcula la nueva componente x con la formula indicada

y(k)=(yant+(xant*d*ds(i)));  %%Se calcula la nueva componente y con la formula indicada

z(k)=zant-d*(angles(i));                %%Se calcula el nuevo ángulo de rotación con la formula indicada

xant=x(k);                                %%Se sobreescribe la componente x-1 con la componente calculada para ser utilizada en la siguiente iteración

yant=y(k);                                %%Se sobreescribe la componente y-1 con la componente calculada para ser utilizada en la siguiente iteración

zant=z(k);                              %%Se sobreescribe el ángulo de rotación z-1 con la componente calculada para ser utilizada en la siguiente iteración

d=(2*(z(k)>0)-1);                      %%Se hace el cálculo del signo para saber a donde hay que rotar el vector nuevamente (izquierda o derecha)

end

x(k)=fact2(ifin)*x(k); %%El resultado de la componente x se multiplica por su factor de ganancia para el ajuste

y(k)=fact2(ifin)*y(k); %%El resultado de la componente y se multiplica por su factor de ganancia para el ajuste

end

%%_____MACOMPLEJA_____

XR=[-0.14 0.07 -0.34]; %%Se define un primer vector que almacena la parte real del vector X para realizar la MAC compleja en formato float

XI=[0.1 0.65 -0.78];     %%Se define un segundo vector que almacena la parte imaginaria del vector X para realizar la MAC compleja en formato float

for(i=1:3)
```

```
R(i)=XR(i)*x(i)-XI(i)*y(i); %%Se hacen las operaciones correspondientes a la parte real en float

I(i)=XR(i)*y(i)+XI(i)*x(i);  %%Se hacen las operaciones correspondientes a la parte imaginaria en float

End

RR=sum(R); %%Se realiza la suma de las partes reales de cada producto para la MAC en float

RI=sum(I); %%Se realiza la suma de las partes imaginarias de cada producto para la MAC en float

fprintf('\n\n_____RESULTADO FLOAT_____');

fprintf('\n\n\tEl valor de la MAC compleja es = %f + i%f',RR,RI);    %%Se hace el despligue de la MAC compleja en formato float
```

Anexo 13:

Código para la MAC compleja con el algoritmo Cordic en representación Qnm en Matlab:

```
%%Simulacion de MAC COMPLEJA

clc      %%borra todo el texto de la ventana de comandos, lo que da como resultado una pantalla clara

clear all  %%elimina todas las variables del espacio de trabajo actual y las libera de la memoria del sistema.

close all  %%Formas de cerrar o salir

%%_____CORDIC_____
_____

fact2=[0.894427 0.867723 0.861023 0.859346 0.8592415 0.85921531 0.8592087 0.85920711 0.8592067 0.8592066 0.8592065 0.85920656]; %%Diferentes valores para el valor de la ganancia K en diferentes iteraciones

angles=[atand(2^-1) atand(2^-2) atand(2^-3) atand(2^-4) atand(2^-5) atand(2^-6) atand(2^-7) atand(2^-8) atand(2^-9) atand(2^-10) atand(2^-11) atand(2^-12)]; %%Diferentes valores para el valor del ángulo de rotación en diferentes iteraciones

ds=[2^-1 2^-2 2^-3 2^-4 2^-5 2^-6 2^-7 2^-8 2^-9 2^-10 2^-11 2^-12]; %%Vector que almacena los corrimientos logicos hacia la derecha
```

```
rotaciones=[10 1.78 -15.7]; %%Vector que almacena las diferentes rotaciones a aproximar por CORDIC

[X1 XQnm1 XD1 XDS1]=Cod_Qnm(fact2,1,10); %%Funcion que realiza la codificacion n,m del vector 1

[X2 XQnm2 XD2 XDS2]=Cod_Qnm(angles,6,8); %%Funcion que realiza la codificacion n,m del vector 2

[X6 XQnm6 XD6 XDS6]=Cod_Qnm(rotaciónes,6,8); %%Funcion que realiza la codificacion n,m del valor del ángulo de rotación

PR=input('\nIntroduce el valor de la parte real del número complejo = '); %%En la variable PR se almacena el valor de la parte real del número complejo que se desea aproximar por CORDIC

PI=input('\nIntroduce el valor de la parte imaginaria del número complejo = '); %%En la variable PI se almacena el valor de la parte imaginaria del número complejo que se desea aproximar por CORDIC

ifin=input('\nIndica el número de iteraciones = '); %%En la variable ifin se almacena el número de iteraciones que indique el usuario

[X4 XQnm4 XD4 XDS4]=Cod_Qnm(PR,2,8); %%Funcion que realiza la codificacion n,m del valor de la parte real

[X5 XQnm5 XD5 XDS5]=Cod_Qnm(PI,2,8); %%Funcion que realiza la codificacion n,m del valor de la parte imaginaria

for(k=1:3)

xant=XDS4; %%Se inicializa la variable correspondiente a la componente x-1 con la parte real del número complejo inicial

yant=XDS5; %%Se inicializa la variable correspondiente a la componente y-1 con la parte imaginaria del número complejo inicial

zant=XDS6(k); %%Se inicializa la variable correspondiente al ángulo de rotación con el ángulo al que se desea rotar el número complejo inicial

d=(2*(zant>0)-1); %%Se hace el cálculo del signo inicial para realizar la rotación del número complejo hacia el ángulo deseado

for(i=1:ifin)

x(k)=(xant-floor(yant*d*ds(i))); %%Se calcula la nueva componente x con la formula indicada

y(k)=(yant+floor(xant*d*ds(i))); %%Se calcula la nueva componente y con la formula indicada
```

```
z(k)=zant-d*(XDS2(i));          %%Se calcula el nuevo ángulo de rotación con la formula indicada

xant=x(k);                      %%Se sobreescribe la componente x-1 con la componente calculada para ser utilizada en la siguiente iteración

yant=y(k);                      %%Se sobreescribe la componente y-1 con la componente calculada para ser utilizada en la siguiente iteración

zant=z(k);                      %%Se sobreescribe el ángulo de rotación z-1 con la componente calculada para ser utilizada en la siguiente iteración

d=(2*(z(k)>0)-1);               %%Se hace el cálculo del signo para saber a donde hay que rotar el vector nuevamente (izquierda o derecha)

end

x(k)=XDS1(ifin)*x(k); %%El resultado de la componente x se multiplica por su factor de ganancia para el ajuste

y(k)=XDS1(ifin)*y(k); %%El resultado de la componente y se multiplica por su factor de ganancia para el ajuste

end


%%_____________________________________MACOMPLEJA_____________________
_____________________________________________________

XR=[-0.14 0.07 -0.34]; %%Se define un primer vector que almacena la parte real del vector X para realizar la MAC compleja en formato Qnm y float

XI=[0.1 0.65 -0.78];    %%Se define un segundo vector que almacena la parte imaginaria del vector X para realizar la MAC compleja en formato Qnm y float

[XR1 XRQnm1 XRD1 XRDS1]=Cod_Qnm(XR,1,5); %%Funcion que realiza la codificacion 1,5 de la parte real del vector X

[XI2 XIQnm2 XID2 XIDS2]=Cod_Qnm(XI,1,5);   %%Funcion que realiza la codificacion 1,5 de la parte imaginaria del vector X

for(i=1:3)

R(i)=XRDS1(i)*x(i)-XIDS2(i)*y(i);       %%Se hacen las operaciones correspondientes a la parte real en Qnm

I(i)=XRDS1(i)*y(i)+XIDS2(i)*x(i);       %%Se hacen las operaciones correspondientes a la parte imaginaria en Qnm

End
```

```
RR=sum(R); %%Se realiza la suma de las partes reales de cada producto para la MAC en Qnm

RI=sum(I);  %%Se realiza la suma de las partes imaginarias de cada producto para la MAC en Qnm

fprintf('\n\n_____RESULTADO Qnm_____');

fprintf('\n\n\tEl valor aproximado es = %d + i%d ',RR,RI);      %%Se hace el despligue de la aproximación por CORDIC


function [X XQnm XD XDS]=Cod_Qnm(xnTs,n,m)

Entero_positivo=2^(n-1)-1/(2^m);
Entero_negativo=-2^(n-1);
resolucion=1/(2^m);
Muestras=xnTs;
indMm=find(Muestras==max(Muestras));
Muestra_mayor=Muestras(indMm(1));
indMm=find(Muestras==min(Muestras));
Muestra_menor=Muestras(indMm(1));
if Muestra_mayor>Entero_positivo || Muestra_menor<Entero_negativo
fprintf('\n\t¡Error! no hay suficientes bits para codificar\n')
fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)
fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
if Muestra_M==1
Muestra_M=1.1;
end
ns=ceil(log2(Muestra_M))+1;
Entero_positivo=2^(ns-1)-1/(2^m);
Entero_negativo=-2^(ns-1);
fprintf('\n\tAumente el valor de n a %d\n',ns)
fprintf('\n\tRango de codificación sería: [%f a %f]\n\n',Entero_negativo,Entero_positivo)

X=Muestras;
XQnm=zeros(1,numel(X));
XD=XQnm;
XDS=XD;
Else
%     ind=find(abs(xnTs)<1/(2^m));     % busca los elementos que no alcanzan la minima resolución

%     xnTs(ind)=0;
```

```
%      ind=find((xnTs)==(2^(n-1)));      % Busca los elementos con la maxima resolución
%    xnTs(ind)=(2^(n-1))-(1/(2^m));

signo=(xnTs>=0);
signo=2*signo-1;
xnTsA=xnTs;
xnTs=abs(xnTs);
x1=xnTs/resolucion;
xQnmES=floor(x1).*signo; %% Aquí se cambia el criterio truncamiento floor y redondeo round

xQnm=xQnmES*(resolucion);
ind=find(xQnmES<0);
xQnmEU=xQnmES;
xQnmEU(ind)=xQnmEU(ind)+2^(n+m);
X=Muestras;
XQnm=xQnm;
XD=xQnmEU;
XDS=xQnmES;
End
end  %%Funcion que codifica un valor o un vector a Qnm, recibe los datos así como los valores de n y m


function [XQnm XD]=Decod_Qnm(X,n,m,S)


if S=='S' || S=='s'
Entero_positivo=2^(n+m-1)-1;
Entero_negativo=-2^(n+m-1);
Muestras=X;
indMm=find(Muestras==max(Muestras));
Muestra_mayor=Muestras(indMm(1));
indMm=find(Muestras==min(Muestras));
Muestra_menor=Muestras(indMm(1));

Max=max(abs(X));
N=log2(Max(1));
if N>(n+m-1)
fprintf('\n\t¡Error! no hay suficientes bits para decodificar\n')
fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)
fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
ns=floor(log2(Muestra_M))+1;
Entero_positivo=2^(ns-1);
```

```
Entero_negativo=-2^(ns-1);
fprintf('\n\tAumente el valor de n+m a %d\n',ns+1)
 fprintf('\n\tRango de codificación sería: [%f a %f]\n\n',Entero_negativo,Entero_positivo)
X=Muestras;
XQnm=zeros(1,numel(X));
XD=XQnm;
else
indc=find(X<0);
Xaux=X;
X(indc)=X(indc)+2^(n+m);
XC=de2bi(X,n+m);
% XC=[PD PE];
PDc=1./(2.^(m:-1:1));
PEc=(2.^(0:n-1));
PEc(end)=-PEc(end);
Rc=kron([PDc PEc],ones(numel(X),1));
xp=sum(XC.*Rc,2)';
XQnm=xp;
        XD=Xaux;
   end
elseif S=='U' || S=='u'
Entero_positivo=2^(n+m)-1;
Entero_negativo=0;
Muestras=X;
indMm=find(Muestras==max(Muestras));
Muestra_mayor=Muestras(indMm(1));
indMm=find(Muestras==min(Muestras));
Muestra_menor=Muestras(indMm(1));

Max=max(abs(X));
N=log2(Max(1));
if N>=(n+m)
fprintf('\n\t¡Error! no hay suficientes bits para decodificar\n')
fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)
fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
ns=floor(log2(Muestra_M))+1;
Entero_positivo=2^(ns)-1;
Entero_negativo=0;
fprintf('\n\tAumente el valor de n+m a %d\n',ns)
fprintf('\n\tRango de codificación sería:[%fa %f]\n\n',Entero_negativo,Entero_positivo)
X=Muestras;
XQnm=zeros(1,numel(X));
XD=XQnm;
else
XC=de2bi(X,n+m);
```

```
% XC=[PD PE];
PDc=1./(2.^(m:-1:1));
PEc=(2.^(0:n-1));
PEc(end)=-PEc(end);
Rc=kron([PDc PEc],ones(numel(X),1));
xp=sum(XC.*Rc,2)';
XQnm=xp;
XD=X;
end
else
   fprintf('\n\n\tError, Especifique Signado (S) o no signado (U)\n\n')
end
end      %%Funcion que decodifica un valor o un vector a tipo float
```

Anexo 14:

Código para la MAC compleja con el algoritmo Cordic en representación Qnm en el MSP430G2553:

```
#include "io430.h"

#include "stdio.h"  //Libreria para hacer uso de la funcion sprintf para desplegar los datos en el LCD


char msj1[]={"\r\n\nParteReal= "}; //Cadenas de caracteres que almacenan los dos distintos mensajes

char msj2[]={"\r\nParteImaginaria= "};

char resultadoreal[8],resultadoimag[8]; //Cadenas de caracteres que almacenaran los valores de la MAC compleja para el despliegue

signed long fact[10]={915,888,881,879,879,879,879,879,879,879}; //Diferentes valores para el valor de la ganancia K en diferentes iteraciones

signed long angles[10]={6800,3593,1824,915,458,229,114,57,28,14}; //Diferentes valores para los distintos ángulos de rotación

signed long ds[10]={2,4,8,16,32,64,128,256,512,1024}; //Corrimientos logicos a la derecha para las disintas iteraciones

signed long PR=256,s,xant=0,yant=0,zant=0,x=0,y=0,z=0,xres[3],yres[3]; //En la variable PR se almacena el valor de la parte real del número complejo que se desea aproximar por CORDIC
```

```
signed long PI=0; //En la variable PI se almacena el valor de la parte imaginaria del número complejo que se desea aproximar por CORDIC

signed long rot[3]={2560,455,-4019}; //En el vector rot se almacenan los valores de rotación que se desea aproximar por CORDIC

int ifin=10; //En la variable ifin se almacena el número de iteraciones que indique el usuario

signed long XREAL[3]={-4,2,-10}; //Se define un primer vector que almacena la parte real del vector X para realizar la MAC compleja en formato Qnm

signed long XIMAG[3]={3,20,-24}; //Se define un primer vector que almacena la parte imaginatia del vector X para realizar la MAC compleja en formato Qnm

signed long REAL[3],IMAG[3],RESREAL=0,RESIMAG=0; //En el vector REAL se almacenan todas las partes reales de los productos así como en el vector IMAG, y en RESREAL y RESIMAG se almacenaran los resultados de la MAC compleja



int main( void )

{

 // Stop watchdog timer to prevent time out reset

 WDTCTL = WDTPW + WDTHOLD;

 BCSCTL1 = CALBC1_1MHZ;        //DCO configurado a 1MHz

 DCOCTL = CALDCO_1MHZ;


 //CONFIGURACIÓN DEL MODULO UART

 P1SEL  |= BIT1 + BIT2 ;            // Configurando las terminales P1.1 y P1.2

 P1SEL2 |= BIT1 + BIT2 ;             // como parte del sistema UART

 P1DIR &= ~BIT1;                  // P1.1 <-- RXD

 P1DIR |= BIT2;                  // P1.2 --> TXD

 UCA0CTL1 |= UCSSEL_2;               // SMCLK

 //.....9600bps.....

 UCA0BR0 = 104;                  // 1MHz 9600
```

```
 UCA0BR1 = 0;                      // 1MHz 9600
 UCA0MCTL = UCBRS0;                // Modulation UCBRSx = 1
 //------------------------------------------------------------------


 UCA0CTL1 &= ~UCSWRST;             // **Initialize USCI state machine**


 while(1)
 {
   for(char j=0;j<3;j++) //Bucle que permite realizar las aproximaciones de los números complejos por CORDIC para cada valor de rotación
   {
     xant=PR; //Se inicializa la variable correspondiente a la componente x-1 con la parte real del número complejo inicial
     yant=PI;  //Se inicializa la variable correspondiente a la componente y-1 con la parte imaginaria del número complejo inicial
     zant=rot[j]; //Se inicializa la variable correspondiente al ángulo de rotación con el ángulo al que se desea rotar el número complejo
     if(zant>=0){s=1;}else{s=-1;}; //Se hace el cálculo del signo inicial para realizar la rotación del número complejo hacia el ángulo deseado
     for(char i=0;i<ifin;i++)
     {
       x=(xant-(yant*s)/ds[i]);    //Se calcula la nueva componente x con la formula indicada
       y=(yant+(xant*s)/ds[i]);    //Se calcula la nueva componente y con la formula indicada
       z=zant-s*(angles[i]);           //Se calcula el nuevo ángulo de rotación con la formula indicada
       xant=x;                   //Se sobreescribe la componente x-1 con la componente calculada para ser utilizada en la siguiente iteración
       yant=y;                   //Se sobreescribe la componente y-1 con la componente calculada para ser utilizada en la siguiente iteración
```

```
      zant=z;                    //Se sobreescribe el ángulo de rotación z-1 con la componente calculada para ser utilizada en la siguiente iteración

      if(z>=0){s=1;}else{s=-1;};          //Se hace el cálculo del signo para saber a donde hay que rotar el vector nuevamente (izquierda o derecha)

    }

    xres[j]=x*fact[ifin-1]; //El resultado de la componente x se multiplica por su factor de ganancia para el ajuste

    yres[j]=y*fact[ifin-1]; //El resultado de la componente y se multiplica por su factor de ganancia para el ajuste

  }


  RESREAL=0;  //Se inicializan las variables que almacenaran el resultado

  RESIMAG=0;

  for(char i=0;i<3;i++) //Se realiza el cálculo de cada una de las partes reales e imaginarias de los productos

  {

    REAL[i]=XREAL[i]*xres[i]-XIMAG[i]*yres[i];

    IMAG[i]=XREAL[i]*yres[i]+XIMAG[i]*xres[i];

  }

  for(char i=0;i<3;i++) //Se realiza la suma de cada una de todas las partes reales e imaginarias de los productos

  {

    RESREAL=RESREAL+REAL[i];

    RESIMAG=RESIMAG+IMAG[i];

  }

  sprintf(resultadoreal,"%ld",RESREAL); //Compone una cadena con el valor real de la MAC compleja, el contenido se almacena como cadena en resultadoreal

  sprintf(resultadoimag," %ld",RESIMAG); //Compone una cadena con el valor imaginario de la MAC complejay, el contenido se almacena como cadena en resultadoimaginario

  for(char i = 0; i <= sizeof msj1 -1 ; i++) //Se hace el despliegue de todos los datos por el puerto serial con ayuda de TERA TERM
```

```
    {
      UCA0TXBUF = msj1[i];
      while (!(IFG2&UCA0TXIFG)){}
    }
    for(char i = 0; i <= sizeof resultadoreal -1 ; i++)
    {
      UCA0TXBUF = resultadoreal[i];
      while (!(IFG2&UCA0TXIFG)){}
    }
    for(char i = 0; i <= sizeof msj2 -1 ; i++)
    {
      UCA0TXBUF = msj2[i];
      while (!(IFG2&UCA0TXIFG)){}
    }
    for(char i = 0; i <= sizeof resultadoimag -1 ; i++)
    {
      UCA0TXBUF = resultadoimag[i];
      while (!(IFG2&UCA0TXIFG)){}
    }
   __delay_cycles(5000000); //Se realiza una pausa de 5 segundos para visualizar el valor
  }
}
```

Anexo 15:

Código para la MAC compleja con el algoritmo Cordic para la comparación de los resultados de la MAC compleja con algoritmo Cordic en formato flotante en el MSP430G2553 y en Matlab:

```
clc
clear all
close all
```

```
%%Los resultados de CORDIC dan un Q3.18
%%Los números del vector son codificados con Q1.5
%%Las sumas dieron un acarreo de 2 bits para la decodificación
%%Se realiza la decodificacion de los valores real e imaginario que resultan de la MAC compleja que nos da un Q6.23

[RESMATLABRQnm RESMATLABRD]=Decod_Qnm(-5102595,6,23,'S');
[RESMATLABIQnm RESMATLABID]=Decod_Qnm(503667,6,23,'S');
[RESMSPRQnm RESMSPRD]=Decod_Qnm(-5041944,6,23,'S');
[RESMSPIQnm RESMSPID]=Decod_Qnm(493998,6,23,'S');

fprintf('\n\n_____RESULTADO FLOAT_____');
fprintf('\n\n\tEl valor de la MAC compleja calculada por MATLAB es = %f + i%f',RESMATLABRQnm,RESMATLABIQnm);    %%Se hace el despligue de la MAC compleja en formato float calculada con MATLAB

fprintf('\n\n\tEl valor de la MAC compleja calculada con MSP es = %f + i%f',RESMSPRQnm,RESMSPIQnm);      %%Se hace el despligue de la MAC compleja en formato float calculada con MSP

function [XQnm XD]=Decod_Qnm(X,n,m,S)


  if S=='S' || S=='s'
Entero_positivo=2^(n+m-1)-1;
Entero_negativo=-2^(n+m-1);
Muestras=X;
indMm=find(Muestras==max(Muestras));
Muestra_mayor=Muestras(indMm(1));
indMm=find(Muestras==min(Muestras));
Muestra_menor=Muestras(indMm(1));

Max=max(abs(X));
N=log2(Max(1));
if N>(n+m-1)
fprintf('\n\t¡Error! no hay suficientes bits para decodificar\n')
fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)
fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
ns=floor(log2(Muestra_M))+1;
Entero_positivo=2^(ns-1);
Entero_negativo=-2^(ns-1);
fprintf('\n\tAumente el valor de n+m a %d\n',ns+1)
        fprintf('\n\tRango de codificación sería: [%f a %f]\n\n',Entero_negativo,Entero_positivo)
X=Muestras;
```

```
XQnm=zeros(1,numel(X));
XD=XQnm;
else
indc=find(X<0);
Xaux=X;
X(indc)=X(indc)+2^(n+m);
XC=de2bi(X,n+m);
% XC=[PD PE];
PDc=1./(2.^(m:-1:1));
PEc=(2.^(0:n-1));
PEc(end)=-PEc(end);
Rc=kron([PDc PEc],ones(numel(X),1));
xp=sum(XC.*Rc,2)';
XQnm=xp;
XD=Xaux;
end
elseif S=='U' || S=='u'
Entero_positivo=2^(n+m)-1;
Entero_negativo=0;
Muestras=X;
indMm=find(Muestras==max(Muestras));
Muestra_mayor=Muestras(indMm(1));
indMm=find(Muestras==min(Muestras));
Muestra_menor=Muestras(indMm(1));

Max=max(abs(X));
N=log2(Max(1));
if N>=(n+m)
fprintf('\n\t¡Error! no hay suficientes bits para decodificar\n')
fprintf('\n\tRango de codificación: [%f a %f]',Entero_negativo,Entero_positivo)
fprintf('\n\tRango a codificación: [%f a %f]\n',Muestra_menor,Muestra_mayor)
Muestra_M=ceil(max([abs(Muestra_mayor) abs(Muestra_menor)]));
ns=floor(log2(Muestra_M))+1;
Entero_positivo=2^(ns)-1;
Entero_negativo=0;
fprintf('\n\tAumente el valor de n+m a %d\n',ns)
        fprintf('\n\tRango de codificación sería: [%f a
%f]\n\n',Entero_negativo,Entero_positivo)
X=Muestras;
XQnm=zeros(1,numel(X));
XD=XQnm;
else
XC=de2bi(X,n+m);
% XC=[PD PE];
PDc=1./(2.^(m:-1:1));
PEc=(2.^(0:n-1));
```

```
PEc(end)=-PEc(end);
Rc=kron([PDc PEc],ones(numel(X),1));
xp=sum(XC.*Rc,2)';
XQnm=xp;
XD=X;
end
else
fprintf('\n\n\tError, Especifique Signado (S) o no signado (U)\n\n')
end
end      %%Funcion que decodifica un valor o un vector a tipo float
```