# Transformada Rápida de Fourier implementada en ASSEMBLY NASM X86 embebido para distribuciones Linux

Alexander Alvarado Sotela, Allan Carranza Chavarría, Laura Piedra López, Jorge Rodríguez Esquivel
alex.alvarst96@gmail.com j.allanj@hotmail.com elena1996.lpl@hotmail.com jo.roes18@gmail.com
EL4314 - Arquitectura de Computadoras I
Escuela de Ingeniería en Electrónica
Instituto Tecnológico de Costa Rica

***Resumen*—El presente informe exponen los procedimientos y resultados obtenidos del primer proyecto del curso Arquitectura de computadoras I, el cual se basó en un algortimo en emsamblador NASM x86 que confexiona la transformada rápida de Fourier. Dicho algoritmo se basa en multiplcar los N coeficientes de la función discreta en el tiempo, con sus respectivos coeficientes de fase para cada muestra. Este algoritmo fue implementado bajo la plataforma de Proyecto Yocto, el cual crea una imagen para la visualización de dicho algoritmo.**

***Palabras clave*—FFT, NASM x86, Software Embedded, Linux, Yocto Project.**

## I. INTRODUCCIÓN

En el diseño y análisis de señales digitales la transformada rápida de Fourier (FFT como se suele indicar en la literatura) juega un rol muy importante en el procesamiento digital de señales. La FFT de una función permite conocer los coeficientes reales e imaginarios (también se pueden representar en función de la magnitud y fase) de la función $f(n)$, la cual es una función discreta en el tiempo, cuyos coeficientes pertenecen al dominio de la frecuencia.

A partir de una señal discreta de función $f(n)$, cuyos coeficientes en el dominio del tiempo están dados por el siguiente vector transpuesto $\underline{x}(n)$

$$\underline{x}(n) = [x(0), x(1), x(2), ..., x(n)]^T \tag{1}$$

Se desea calcular la FFT de $f(n)$, la forma general de expresar los coeficientes resultantes de dicha transformada para un N número de muestras, dicha función está dada por la siguiente expresión:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{2\pi j}{N} kn} \tag{2}$$

Para trabajar la transformación dada en la Ec.[2] se realiza un cambio en la expresión que simplifica más la visualización y representación matemática, en donde se sustituye lo siguiente:

$$W_n = e^{-\frac{2\pi j}{N}} \tag{3}$$

El $W_n$ dada en la igualdad anterior, es lo que se conoce como factor de giro o factor de fase, e indica de forma compleja en cuantos segmentos (muestras) debe dividirse el circulo unitario sobre el plano complejo. Para dar con la nueva forma de la FFT incluyendo el factor de fase, se sustituye el exponencial por $W_n$ y se obtiene lo siguiente

$$X(k) = \sum_{n=0}^{N-1} x(n) {W_n}^{kn} \tag{4}$$

De esta forma, cualquier cálculo computacional puede realizarse de una forma mucho más trivial que el problema inicial, ya que, partimos de lo siguiente

$$W_n = cos(\frac{2\pi}{N}) + jsen(\frac{2\pi}{N}) \tag{5}$$

Y de este modo eventualmente se calcularía computacionalmente cada coeficiente de manera instantánea, aplicando un diseño de funciones en NASM X86 que ejecuten funciones de coseno y seno mediante los métodos de Series de Taylor, además, existen otras formas de calcular esta misma transformada que involucran el álgebra lineal, como a su vez, métodos de optimizarla computacionalmente para lograr reducir el número de operaciones, pero todo esto lo veremos más adelante en el presente documento.

Por otra parte, Linux básicamente consiste en un sistema operativo que facilita herramientas a desarrolladores de software. Yocto por parte, es un open source el cual permite dar herramientas a los desarrolladores para diseñar software embebido en distribuciones Linux.

## II. FFT IMPLEMTENTADA EN ASSEMBLY NASM X86

Para el diseño de la FFT se realizaron 2 propuestas de diseño, la cual dejaba por fuera el tema de la optimización (reducción) de cálculos computacionales, esta sección de optimización se explicará antes de finalizar esta sección.

### *II-A. Propuesta I: Cálculo Inmediato de Coeficientes*

La primera propuesta consiste en el diseño de la FFT capaz de trabar para un N particular limitado únicamente por el tamaño de los cálculos a realizar no mayor a la capacidad

de los registros de 64 bits, es decir, puede trabajar números de muestras N muy grandes siempre y cuando sus cálculos no superen el valor de almacenamiento máximo por registro.
Una de las ventajas de esta propuesta es el poder calcular transformadas con número de muestras muy grandes y de una forma muy precisa y en tiempo real, sin embargo, también presenta varias desventajas, tales como gran número de cálculos, lo que se traduce a mucho tiempo de ejecución, además, utilizar un número de muestras muy grandes llegamos a lo que se conoce como *overfitting* lo cual quiere decir que se eleva la exactitud y precisión de los cálculos lo cual hace que el sistema pierda todo el sentido de llamarse transformada rápida, convirtiendola en un algoritmo ineficiente, además de que para grandes muestras, la transformada, por el mismo efecto, pierde todo su valor de información deseado.
Este método consiste en realidar la separación de $W_n$ en su parte real y su parte imaginaria, diseñar funciones que calculen las funciones de senos y cosenos mediante series de Taylor para un N de muestras de entrada, y así, únicamente tener que realizar el siguiente procedimiento que se muestras a continuación

$$W_n = cos(\frac{2\pi}{N}) + jsen(\frac{2\pi}{N})$$

$$W_n^{kn} = (cos(\frac{2\pi}{N}) + jsen(\frac{2\pi}{N}))^{kn} \quad (6)$$

La idea de trabajar el exponencial complejo en forma de coseno y seno es para lograr independizar sus partes reales e imaginarias, almacenándolas en diferentes espacios de memorias asignados dentro del programa, esto parecería solucionar el problema de los números complejos en ensamblador, sin embargo aún falta un problema por resolver que dentro de un momento detallaremos sobre eso.
Las funciones de las series de Taylor que calculan los senos y los cosenos están dadas por

$$cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + ... \quad (7)$$

$$sen(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + ... \quad (8)$$

Un detalle importante observar es que la cantidad de términos mostrados anteriormente son suficientes para realizar un cálculo bastante aproximado para lo requerido de las funciones cos(x) y sen(x).

De manera intuitiva podemos obtener a partir del diagrama de flujo, cual es el algoritmo para la función de seno, por lo cual no se mostrará en el presente trabajo.

El código de la función de coseno implementado encargado de imprimir el resultado de la función para verificar que la función está funcionando correctamente se muestra a continuación junto con su diagrama de flujo correspondiente

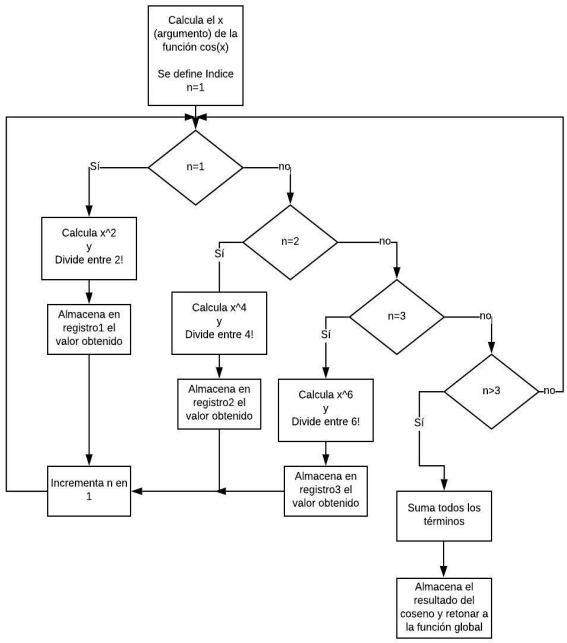


Figura 1. Diagrama de flujo para la función de coseno mediante la implementación del algoritmo de series de Taylor.

```
 GNU nano 2.9.3
section .bss
        digitSpace resb 100
        digitSpacePos resb 8
        ;;var resb 8


section .text
        global _start

_start:

        call _Parametros

        call _Pot               ;Llama función de calculo de potencias
        xor rax, rax

        mov rax, r10            ;antes r9
        call _printRAX

        mov rax, 60
        mov rdi, 0
        syscall


_Parametros:
        mov r13, 4              ;Numero de muestras
        mov rax, 360            ;Grados Total del Circulo

        idiv r13                ;Divide los grados del circulo entre numero de muestras
        mov r14, r13            ;Numero de muestras se mueve a r14
        xor r13, r13
        mov r13, rax            ;Se copia a si mismo para multiplicarse y calcular potencia

        ret

_Pot:                           ;Calcula la potencia n de un numero
        mov r7, 1               ;Indice de potencia (iteraciones del exponente)

        call _Pot2
        xor rax, rax
        mov rax, r13

        ;call _Pot4

        ;call _Pot6

        ret

_Pot2:                          ;Calcula la potencia 2
```

Figura 2. Código en NASM X86 primera parte de la función de coseno.

```
_Pot2:                          ;Calcula la potencia 2

        imult r13


        cmp r7,1
        jne _Pot2

        mov r10, rax
        xor rax, rax
        call _Pot4

_Pot4:
        inc r7                  ;Calcula la potencia 4
        imult r13

        cmp r7,4
        jne _Pot4

        mov r10,rax
        xor rax,rax

        call _Pot6

_Pot6:                          ;Calcula la potencia 6
        inc r7
        imult r13

        cmp r7,6
        jne _Pot6

        mov r10,rax
        xor rax,rax

        ret

_printRAX:
        mov rcx, digitSpace
        mov rbx, 10
        mov [rcx], rbx
        inc rcx
        mov [digitSpacePos], rcx

_printRAXLoop:
        mov rdx, 0
        mov rbx, 10
        div rbx
        push rax
        add rdx, 48
```

Figura 3. Código en NASM X86 segunda parte de la función de coseno.

```
_printRAX:
        mov rcx, digitSpace
        mov rbx, 10
        mov [rcx], rbx
        inc rcx
        mov [digitSpacePos], rcx

_printRAXLoop:
        mov rdx, 0
        mov rbx, 10
        div rbx
        push rax
        add rdx, 48

        mov rcx, [digitSpacePos]
        mov [rcx], dl
        inc rcx
        mov [digitSpacePos], rcx

        pop rax
        cmp rax, 0
        jne _printRAXLoop

_printRAXLoop2:
        mov rcx, [digitSpacePos]

        mov rax, 1
        mov rdi, 1
        mov rsi, rcx
        mov rdx, 1
        syscall

        mov rcx, [digitSpacePos]
        dec rcx
        mov [digitSpacePos], rcx

        cmp rcx, digitSpace
        jge _printRAXLoop2

        ret
```

Figura 4. Código en NASM X86 tercera parte de la función de coseno.

El detalle que quedamos de mencionar anteriormente, y es en donde radica el mayor problema sobre esta propuesta, está en el momento de evaluar los exponentes kn en $W_n$, donde no solo tenemos una formula notable que va a variar dependiendo de las dimensiones del total de muestras que forman la matriz cuadrada de los factores de fase, sino, el verdadero problema existe en trabajar los números reales e imaginarios dentro del entorno de NASM, el problema se redimensiona hacia un punto mucho más complejo de lo que ya era para resolver, simplemente NASM complicaría los cálculos a un nivel donde no es para nada factible realizar algoritmos que ejecuten dicha operación compleja, y eso nos lleva a la propuesta 2, la cual es mucho más sencilla, sin embargo, no deja de ser para nada trivial de implementar.

### *II-B. Propuesta II: FFT por multiplicación matricial*

Esta segunda propuesta se basa en la forma de ver la FFT como producto matricial, donde se multiplica el vector que almacena los n coeficientes tomados de la muestra de la función, por la matriz de factores de fase, esta es una matriz que se define de la siguiente forma:

$$\mathrm{W}_N^{kn} = \begin{pmatrix} 1 & 1 & \cdots 1 & \\ 1 & W_{2,2} & \cdots & W_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & W_{k,2} & \cdots & W_{k,n} \end{pmatrix}$$

Donde el producto matricial de ambos da como resultado la solución de la FFT, sin embargo, en este caso, tenemos nuevas ventajas y desventajas. Como ventajas podemos mencionar el veloz cálculo de la FFT, ya que solo debe realizar la multiplicación entre el vector de coeficientes y la matriz de factores de fase, para finalmente solo sumar los que correspondan a cada muestra.

## III. IMAGEN IMPLEMENTADA EN YOCTO PROYECT

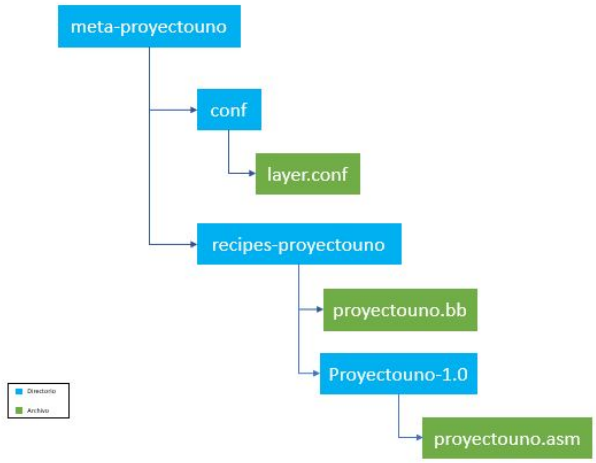


Figura 5. Diagrama del distribución de las carpetas de la implementación de la imagen en Yocto

En base a la figura 5 se puede destacar lo siguiente:

1. Se inicia creando un nuevo meta, el cual atraves de mkdir se creó meta-proyectouno.
   *a*) Dentro de este meta se crearon dos subsecciones más las cuales fueron:
      1) conf
         - Se crea (mkdir) layer.conf
      2) recipes-proyectouno
         - Se crea (mkdir) proyectouno.bb; cabe resaltar que este archivo es que va contener las recetas para el ejecutable del algoritmo a implementar.En este operamos todo que necesitamos esto quiere decir que el archivo .bb que creemos nos dará el espacio para que la programación sea ejecutada por el bitbake. En este espacio creamos el link entre los dos compiladores que se necesitaban para el proyecto entre nasm y el gcc de linux.
         - Se crea (mkdir) proyectouno-1.0, el cual el 1.0 se refiere a la versión de dicho algoritmo. En esta carpeta se encontrara la programación.Y es donde haremos referencia en el .bb
           *a′* Se crea o se agrega el archivo que contiene el algoritmo, se le agrega .asm dado a que es su formato de compilación (ensamblador).
   *b*) Una vez creada estas carpetas, nos devolvemos a meta-proyectouno; acá realizamos lo siguiente:
      1) Se ingresa a la carpeta conf, acá se modifica (nano) layer.conf. A este archivo se le asigna todas las configuraciones que se necesiten para inicializar la imagen , en otras palabras va a buscar todos los documentos que se encuentren en nuestra receta.

```
# We have a conf and classes directory, add to BBPATH
BBPATH.=":${LAYERDIR}"

# We have recipes-* directories, add to BBFILES
BBFILES += "${LAYERDIR}/recipes-*/*.bb \
            ${LAYERDIR}/recipes-*/*/*.bbappend"

BBFILE_COLLECTIONS +="proyectouno"
BBFILE_PATTERN_proyectouno ="^${LAYERDIR}/"
BBFILE_PRIORITY_proyectouno ="5"
LAYERVERSION_proyectouno ="4"
LAYERSERIES_COMPAT_proyectouno = "warrior"
```

Figura 6. Modificación realizada a la carpeta layer.conf

      2) Se modifica en la carpeta proyectouno.bb el SRC URI, el do compile y do install; tal y como se muestra a continuación:
      3) Una vez actualizado dicho archivo, nos dirigimos a la carpeta proyectouno-1.0 para ya sea actualizar o cargar el algoritmo del FFT y le damos guardar para que se guarden dichos cambios.
2. Nos devolvemos a la raíz poky-warrior:

```
# Recipe created by recipetool
# This is the basis of a recipe and may need further editing in order to be ful$
# (Feel free to remove these comments when editing.)

# Unable to find any files that looked like license statements. Check the accom$
# documentation and source headers and set LICENSE and LIC_FILES_CHKSUM accordi$
#
# NOTE: LICENSE is being set to "CLOSED" to allow you to at least start buildin$
# this is not accurate with respect to the licensing of the software being buil$
# will not be in most cases) you must specify the correct value before using th$
# recipe for anything other than initial testing/development!
LICENSE = "CLOSED"
LIC_FILES_CHKSUM = ""
COMPATIBLE_HOST = '(x86_64|i.86).*-(linux|freebsd.*)'

# No information for SRC_URI yet (only an external source tree was specified)
SRC_URI = "file://miprogra.asm"

# NOTE: no Makefile found, unable to determine what needs to be done

S="${WORKDIR}"

TARGET_CC_ARCH += "${LDFLAGS}"

do_configure () {
        # Specify any needed configure commands here
        :
}

do_compile () {
     nasm -f elf64 -o miprogra.o miprogra.asm
     ${CC} miprogra.o -o miprogra
}

do_install () {
        install -d ${D}${bindir}
        install -m 0755 miprogra ${D}${bindir}
}
```

Figura 7. Modificación realizada a la carpeta proyectouno.bb

   *a*) Entramos a la carpeta build
      1) Dentro de la carpeta build nos dirigimos a la carpeta conf, para realizarle modificaciones a las dos siguientes carpetas:
         - bblayers.conf
           *a′* Se le agrega (nano) a esta carpeta, lo siguiente:

```
# POKY_BBLAYERS_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
POKY_BBLAYERS_CONF_VERSION = "2"

BBPATH = "${TOPDIR}"
BBFILES ?= ""

BBLAYERS ?= " \
  /home/jorge/Yocto/poky-warrior/meta \
  /home/jorge/Yocto/poky-warrior/meta-poky \
  /home/jorge/Yocto/poky-warrior/meta-yocto-bsp \
  /home/jorge/Yocto/poky-warrior/meta-newlayer \
  /home/jorge/Yocto/poky-warrior/meta-prueba \
  /home/jorge/Yocto/poky-warrior/meta-proyectouno \
  "
```

Figura 8. Modificación realizada a la carpeta bblayers.conf

         - local.conf
           *a′* Se le agrega (nano) a esta carpeta, lo siguiente:

```
#
# Qemu configuration
#
# By default qemu will build with a builtin VNC server where graphical output can be
# seen. The two lines below enable the SDL backend too. By default libsdl2-native will
# be built, if you want to use your host's libSDL instead of the minimal libsdl built
# by libsdl2-native then uncomment the ASSUME_PROVIDED line below.
PACKAGECONFIG_append_pn-qemu-system-native = " sdl"
PACKAGECONFIG_append_pn-nativesdk-qemu = " sdl"
#ASSUME_PROVIDED += "libsdl2-native"

# CONF_VERSION is increased each time build/conf/ changes incompatibly and is used to
# track the version of this file when it was generated. This can safely be ignored if
# this doesn't mean anything to you.
CONF_VERSION = "1"

IMAGE_INSTALL_append = "\
        helloworldinc \
        prueba \
        proyectouno \
"
```

Figura 9. Modificación realizada a la carpeta local.conf

*b*) Una vez realizada dichas modificaciones, nos devolvemos a la carpeta poky-warrior para realizar el siguiente instrucción en la terminal:

```
jorge@jorger:~/Yocto/poky-warrior$ source oe-init-build-env

### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common targets are:
    core-image-minimal
    core-image-sato
    meta-toolchain
    meta-ide-support

You can also run generated qemu images with a command like 'runqemu qemux86'
```

Figura 10. Comando para iniciar en la elaboración de la receta

*c*) Una vez realizado dicha instrucción se prosigue a crear la receta del proyectouno, con el comando bitbake proyectouno.

*d*) Luego se inicia la creación de la imagen con el siguiente comando: bitbake core-image-sato

## IV. CONCLUSIONES

1. Debido al lenguaje de programación para el proyecto las funciones trigonométricas se debe trabajar como aproximaciones con otros métodos de análisis como Series de Taylor.
2. La forma más práctica para manejar los cálculos del factor de giro para NASM x86 es de forma rectangular por la falta de instrucciones para desarrollarlo de otra forma más efectiva.
3. Para implementar de la FFT, NASM x86 no es la opción más viable en cuanto a facilidad de implementación, debido al excesivo proceso de diseño para operar ya sea números complejos, eso incluye matrices, multiplicaciones, entre otras operaciones matemáticas requeridas, se puede realizar este mismo algoritmo mucho más eficiente en otros lenguajes de programación de igual bajo nivel, tales como TASM y x86.
4. Yocto es una herramienta muy completa pero con un orden muy estricto si se hace el correcto diagrama para crear nuestras recetas no habrá problemas al crear nuestra imagen y correrla en el emulador
5. Se debe tener un panorama detallado de como funciona Yocto , ya que a la hora de trabajar en el se hará mas fácil a la hora de correcion o implentacion del sistema que estemos creando
6. No se logró darle la imagen final por problemas de bibliotecas o directorios que se debía de enlazar en la subcarpeta de proyectouno.bb la cual está contenida en la carpeta recipes-proyectouno

## V. RECOMENDACIONES

- Considerar otros entornos de ensamblador, tales como TASM, x86, e incluso, en caso de que se permita, en cualquier otro lenguaje de alto nivel.
- Perfilar muy bien la aplicación, definir muy bien si lo que se necesita es tiempo, velocidad de ejecución y optimización del computador, o si simplemente se desea obtener el resultado preciso de los coeficientes reales e imaginarios.
- Si se desea calcular los coeficientes instantaneamente, sin importar el tiempo de ejecución del programa, es altamente importante considerar utilizar otro lenguaje de programación, ya que el cálculo de números en ensamblador es para nada trivial, es sumamente complejo y no es para nada factible a nivel de trabajo-resultados.
- A la hora de instalar Yocto asegurarse que el computador tenga la capacidad necesaria para utilizar o instalar la plataforma específicamente cuando se utiliza una maquina virtual.Ya que esta dependiendo del hardware puede llegar a ser un obstáculo para el flujo de trabajo

## REFERENCIAS

[1] ”Yocto Project Reference Manual”, Yoctoproject.org, 2019. Disponible: https://www.yoctoproject.org/docs/2.6/ref-manual/ref-manual.html.

[2] ”Yocto Project Quick Build”, Yoctoproject.org, 2019. Disponible : https://www.yoctoproject.org/docs/2.6/brief-yoctoprojectqs/brief-yoctoprojectqs.html.

[3] ”Yocto Project Development Tasks Manual”, Yoctoproject.org, 2019. Disponible: https://www.yoctoproject.org/docs/2.6/dev-manual/dev-manual.html.

[4] ”BitBake User Manual”, Yoctoproject.org, 2019. Disponible : https://www.yoctoproject.org/docs/2.6/bitbake-user-manual/bitbake-user-manual.html