



Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Electrónica

Arquitectura de Computadores

Proyecto 1

Transformada rápida de Fourier con Yocto Project

Profesor:

Ing. Johan Carvajal Godínez

Elaborado por:

Angie Cerdas Morales 2016115784

Diego Mora Valverde 2015093722

Ayrton Muñoz Valerio 2014004759

II Semestre 2019

Transformada rápida de fourier

Algoritmo utilizado:

FFT de diezmado en el tiempo con algoritmo base 2

La transformada rápida de fourier o FFT, es un algoritmo eficiente para el cálculo de la transformada discreta de fourier DFT (Ecuación 1).

$$H_n \equiv \sum_{k=0}^{N-1} h_k e^{2\pi i k n / N}$$

Ecuación 1. DFT para N puntos

Si se toma el segundo término que pertenece a la suma $e^{2\pi i k n / N}$ y se sustituye por $W^{2\pi i / N}$, también conocido como “twiddle factor”, obtenemos la ecuación 2.

$$H_n = \sum_{k=0}^{N-1} W^{nk} h_k$$

Ecuación 2. DFT con “twiddle factor”

Al realizar la DFT con el método anterior se necesitan N^2 operaciones, por lo que se utiliza el algoritmo de la transformada rápida de fourier, el cual se beneficia de la periodicidad del “twiddle factor” para realizar este cálculo. Con esto se disminuye el número operaciones a $N \log_2 N$.

La DFT de largo N se puede reescribir como la suma de dos DFT, cada una de largo N/2. Una contiene los puntos pares y la otra los impares.

$$\begin{aligned} F_k &= \sum_{j=0}^{N/2-1} e^{2\pi i k j / (N/2)} f_{2j} + W^k \sum_{j=0}^{N/2-1} e^{2\pi i k j / (N/2)} f_{2j+1} \\ &= F_k^e + W^k F_k^o \end{aligned}$$

Ecuación 3. DFT escrita en parte par y parte impar

Las expresiones obtenidas en la ecuación 3 se pueden dividir en la suma de sus partes pares e impares, cada una de largo N/4. Para el caso de F^e se reduciría a la

suma de F^{ee} y F^{eo} . Este proceso se puede seguir aplicando hasta obtener expresiones de largo 1. Para cada entrada f_n existe un patrón de pares e impares.

$$F_k^{eoeoeoe\cdots oee} = f_n \quad \text{for some } n$$

Ecuación 4. Patrones para un punto f_n

Para saber a cuál valor n corresponde una secuencia correspondiente se invierte el patrón de pares e impares, se establecen los valores de par=0 e impar=1 y se obtiene el valor en binario del punto n .

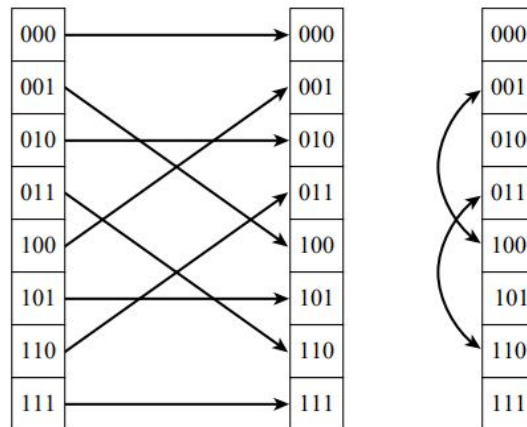


Figura 1. Bit reversed order

Luego de realizar el proceso de inversión de bits se aplica el diezmado en el tiempo utilizando mariposas simplificadas. Un ejemplo de esto se muestra en la figura 2, en el cual se utilizan 8 puntos. Se puede observar que se necesitan $M = \log_2 N$ etapas, cada una con $N/2$ mariposas.

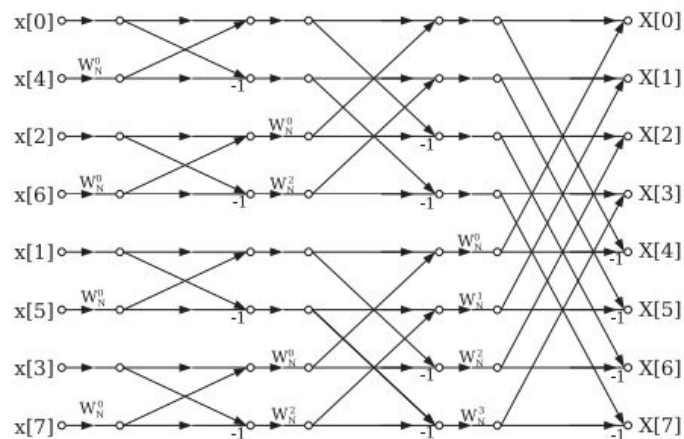


Figura 2. Ordenamiento mariposa

Por lo tanto podemos decir que la estructura del algoritmo para la FFT se divide en dos secciones, donde la primera se encarga de acomodar los datos en orden de inversión de bit (Figura 1), este paso no requiere de memoria extra ya que solo debe cambiar un par de elementos entre ellos. La segunda sección consiste en un ciclo que se ejecuta $N \log_2 N$ veces y calcula la transformada de longitud 2, 4, 8, ..., N. Para cada etapa de este proceso dos ciclos anidados se extienden sobre las sub-transformadas previamente calculadas y los elementos de cada transformada, implementando así el Danielson-Lanczos Lemma.

Costo computacional de una FFT con diezmado en el tiempo en base 2

$\frac{N}{2} \log_2 N$ multiplicaciones complejas.

$N \log_2 N$ sumas complejas

Diagrama de flujo

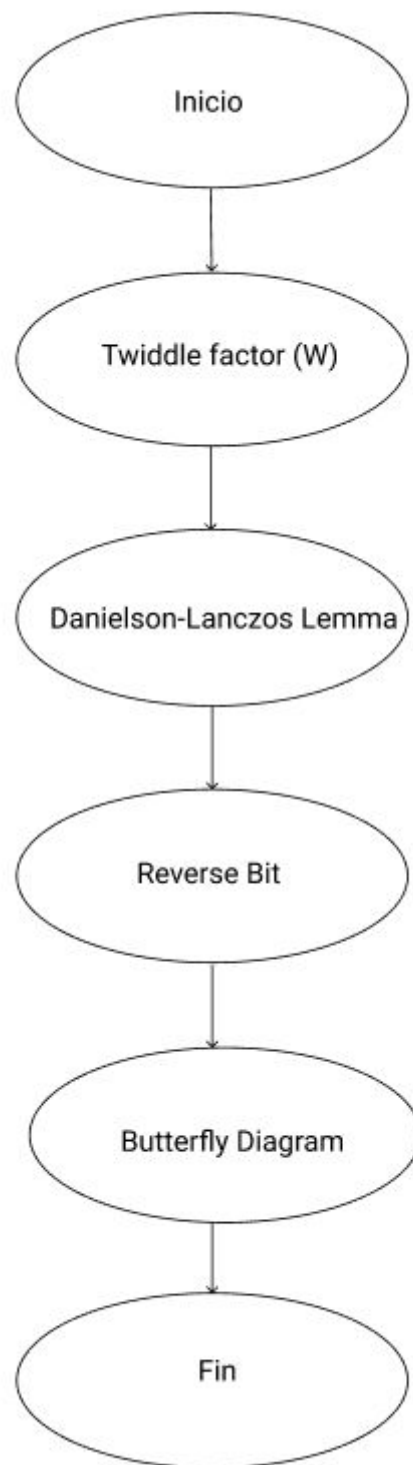


Figura 3. Diagrama de flujo del algoritmo de la FFT

GCC

Compilar en c: `gcc FFTinc.c -o FFTinc`

Correr el ejecutable: `./FFTinc`

Objconv

Objconv se utiliza en el desarrollo multiplataforma de bibliotecas de funciones, para convertir y modificar archivos objeto, además de desensamblar archivos objetos y archivos ejecutables para múltiples plataformas x86 y x86-64. Esto se utilizó como herramienta para cambiar un archivo de lenguaje c a ensamblador x86-64.

Secuencia de pasos:

1. Escribir el comando: `gcc -fno-asynchronous-unwind-tables -s -c -o filec.o file.c`
FFT: `gcc -fno-asynchronous-unwind-tables -s -c -o FFTinc.o FFTinc.c`
2. Para generar el archivo.asm, escribir el comando: `./objconv -fnasm FFTinc.o`
FFT: `./objconv -fnasm FFTinc.o`
3. Modificaciones

Ensamblador NASM

NASM (Netwide assembler) es un ensamblador de 80x86 diseñado para portabilidad y modularidad, soporta varios tipos de formatos de archivos de objetos, como linux, ELF, COFF entre otros.

Secuencia de pasos:

1. Ensamblar: `nasm -f elf64 -o file.o file.asm`
FFT: `nasm -f elf64 -o FFTinc.o FFTinc.asm`
2. Compilar: `gcc -no-pie file.o -o file -lm`
FFT: `gcc -no-pie FFTinc.o -o FFTInc -lm`
3. Ejecutar: `./file`
FFT: `./FFTinc`

Ejemplos

Para demostrar el funcionamiento de la aplicación se probaron 3 ejemplos, realizando la transformada rápida de fourier para diferente cantidad de puntos.

Ejemplo 1: FFT para 4 puntos con parte real e imaginaria

Entrada: $x[n] = \{1.5 + 4.5j, 2 + 8j, 3.9 + 2j, 5 + 0j\}$

Salida: $X[k] = \{12.400000 + 14.500000j$
 $5.600000 + 5.500000j$
 $-1.600000 + -1.500000j$
 $-10.400000 + -0.500000j\}$

```
FFT: Indique la cantidad de puntos 4
FFT:Indique la parte real de los puntos
(unos a la vez, presione enter)
1.5
2
3.9
5
FFT:Indique la parte imaginaria de los puntos
(unos a la vez, presione enter)
4.5
8
2
0
FFT para: 4 puntos
X[k]={12.400000 + 14.500000j
5.600000 + 5.500000j
-1.600000 + -1.500000j
-10.400000 + -0.500000j}
```

Figura 3. Prueba ejemplo 1

Ejemplo 2: FFT para 8 puntos con solo parte real

Entrada: $x[n] = \{1+0j, 2+0j, 3+0j, 4+0j, 4+0j, 3+0j, 2+0j, 1+0j\}$

Salida: $X[k] = \{20.000000 + 0.000000j$
-5.828427 + -2.414214j
0.000000 + 0.000000j
-0.171573 + -0.414214j
0.000000 + 0.000000j
-0.171573 + 0.414214j
0.000000 + 0.000000j
-5.828427 + 2.414214j}

```
FFT: Indique la cantidad de puntos 8
FFT:Indique la parte real de los puntos
(unos a la vez, presione enter)
1
2
3
4
4
3
2
1
FFT:Indique la parte imaginaria de los puntos
(unos a la vez, presione enter)
0
0
0
0
0
0
0
0
FFT para: 8 puntos
X[k]={20.000000 + 0.000000j
-5.828427 + -2.414214j
0.000000 + 0.000000j
-0.171573 + -0.414214j
0.000000 + 0.000000j
-0.171573 + 0.414214j
0.000000 + 0.000000j
-5.828427 + 2.414214j}
```

Figura 4. Prueba ejemplo 2

Ejemplo 3: FFT para 32 puntos

Entrada: $x[n]$:

$\{1+6j, 2+5j, 3+4j, 4+3j, 5+2j, 6+1j, 7+9j, 8+8j, 9+7j, 1+6j, 2+5j, 3+4j, 4+3j, 5+2j, 6+1j, 7+9j, 8+8j, 9+7j, 1+6j, 2+5j, 3+4j, 4+3j, 5+2j, 6+1j, 7+9j, 8+8j, 9+7j, 1+6j, 2+5j, 3+4j, 4+3j, 5+2j\}$

Salida:

```
FFT para: 32 puntos
X[k]={150.000000 + 155.000000j
-12.035648 + -7.035648j
-17.054679 + -12.054679j
-40.318636 + -35.318636j
40.627417 + 45.627417j
9.095481 + 14.095481j
2.734711 + 7.734711j
-8.077868 + -3.077868j
0.000000 + 5.000000j
-2.746289 + 2.253711j
-8.336357 + -3.336357j
13.690309 + 18.690309j
1.171573 + 6.171573j
-2.477194 + 2.522806j
-20.125747 + -15.125747j
5.586387 + 10.586387j
2.000000 + 7.000000j
1.444424 + 6.444424j
-6.602175 + -1.602175j
-3.185920 + 1.814080j
-4.627417 + 0.372583j
-16.026666 + -11.026666j
7.064279 + 12.064279j
3.023748 + 8.023748j
4.000000 + 9.000000j
-33.976195 + -28.976195j
-4.006788 + 0.993212j
-1.499461 + 3.500539j
6.828427 + 11.828427j
-15.277913 + -10.277913j
-9.673243 + -4.673243j
-9.218559 + -4.218559j
```

Figura 5. Prueba del ejemplo 3

Yocto

Para la sección de Yocto Project se en los videos de tutorial compartidos por el asistente del curso, Carlos Emilio Soto Porras, así como tomando como referencia el siguiente diagrama.

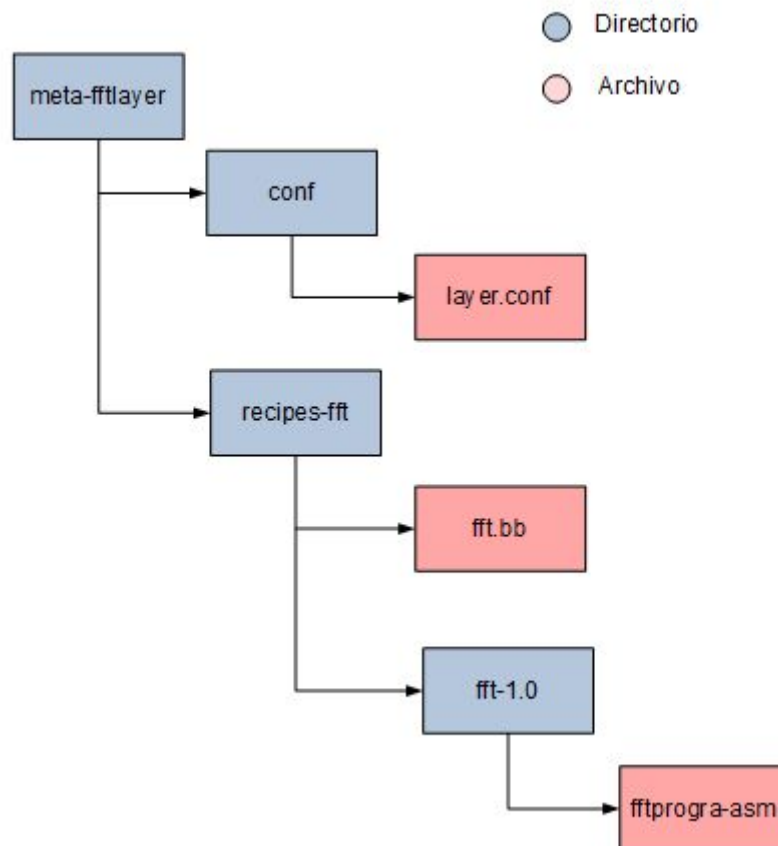


Figura 6. Diagrama estructura de Yocto.

La implementación de Yocto se basó en el aporte que Carlos comparte en sus videos. De estos vídeos se logró implementar la sección de Yocto, presentando un Error el cual se arregló luego de cambiar el directorio “nombre de recetas”_1.0 por el directorio “nombre de recetas”-1.0. Una vez implementado el cambio se logró correr un “Hello World!” en .c

Una vez listo se procedió a ejecutar un “Hello World!” en .asm, para eso se realizaron algunos cambios en el archivo bitbake.conf del directorio de /poky-warrior/meta/conf/. (Ver Figura 6.)

```

# Tools needed to run builds with OE-Core
# python is special cased to point at python2
HOSTTOOLS += " \
    [ ar as awk basename bash bzip2 cat chgrp chmod chown chroot cmp comm cp cpio \
    cpp cut date dd diff diffstat dirname du echo egrep env expand expr false \
    fgrep file find flock g++ gawk gcc getconf getopt git grep gunzip gzip \
    head hostname id install ld ldd ln ls make makeinfo md5sum mkdir mknod \
    mktemp mv nasm nm objcopy objdump od patch perl pod2man pr printf pwd python2 \
    python2.7 python3 ranlib readelf readlink realpath rm rmdir rpcgen sed seq sh sha256sum \
    sleep sort split stat strings strip tail tar tee test touch tr true uname \
    uniq wc wget which xargs \

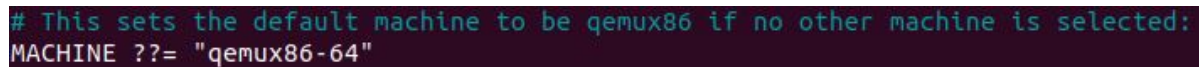
```

Figura 7. Cambios para bitbake nasm

Luego de esto se logró correr archivos de formato .asm.

Otro cambio realizado fue cambiar el *qemux86* por *qemux86-64* para poder correr archivos de 64 bits. Este cambio se realizó basado en la información recuperada .

Realizando los cambios correspondientes en el archivo “nombre de la recipe”.bb del directorio *recipes-“nombre de la recipe”*.

A screenshot of a code editor with a dark background. It shows two lines of text: a comment line starting with a hash symbol and a configuration line for the MACHINE variable.

```
# This sets the default machine to be qemux86 if no other machine is selected:  
MACHINE ??= "qemux86-64"
```

Figura 8.Reemplazo de qemux86 por qemux86-64

Se logró compilar archivos de formato .c de 64 bits, así como archivos de .asm de 64 bits. Con el problema que la ejecución de la FFT en formato ensamblador no se ejecutó correctamente generando un error *Not Found*

Bibliografía

[1]"Yocto Project Quick Start", *Yoctoproject.org*, 2019. [Online]. Recuperado de:
<https://www.yoctoproject.org/docs/2.1/yocto-project-qs/yocto-project-qs.html>.

[2] Fog, A. (2018). *Instructions for objconv*. 2nd ed. Agner Fog, pp.10-14. Recuperado de:
<http://www.agner.org/optimize/objconv-instructions.pdf>

[3]Jones, D. and Selesnick, I. (2010). *The DFT, FFT, and Practical Spectral Analysis*.
Houston, Texas: CONNEXIONS.