

Filtro FIR

Arquitecturas Avanzadas y de Propósito Específico

Cheuk Kelly Ng Pante (alu0101364544@ull.edu.es)

13 de junio de 2024

Índice general

1. Introducción	1
2. Características del proyecto	1
3. Implementaciones	3
3.1. Versión base	4
3.2. Versión 1	4
3.3. Versión 2	4
4. Bibliografía	5

1. Introducción

FIR es un acrónimo en inglés para *Finite Impulse Response* o Respuesta Finita al Impulso. Se trata de un tipo de filtro digital que si su entrada es un impulso (una delta de Kronecker), su salida es un número limitado de términos no nulos. Este tipo de filtro se utiliza comúnmente en aplicaciones de procesamiento de señales, como en la industria de las telecomunicaciones, el procesamiento de audio y la ingeniería de control.

Su expresión en el dominio n es la siguiente:

$$y_n = \sum_{k=0}^{N-1} b_k x_{n-k}$$

donde x_n es la entrada, y_n es la salida, b_k son los coeficientes del filtro y N es el orden del filtro.

La estructura básica de un filtro FIR es la siguiente:

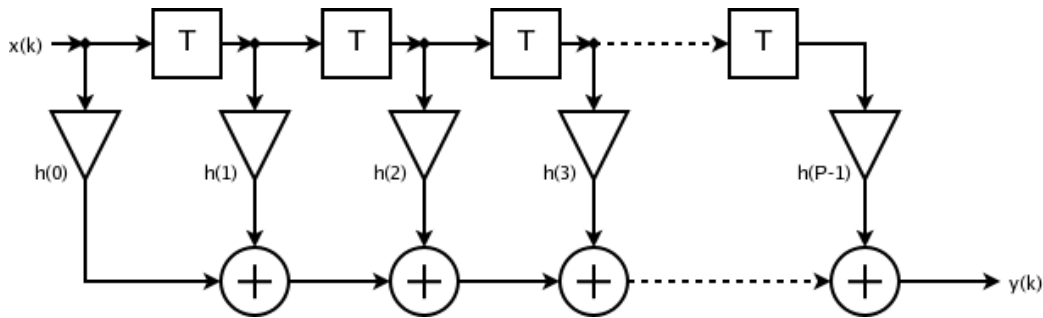


Figura 1.1: Estructura básica de un filtro FIR

2. Características del proyecto

El proyecto se divide en 5 versiones de filtros FIR:

- Versión 0: Filtro FIR base
- Versión 1: Filtro FIR mejorado
- Versión 2: Incluye el uso de keywords, como **const** y **restrict**
- Versión 3: Desenrollado manual y optimización de bucles, condicionales, etc.
- Versión 4: Utilización de **pragmas**
- Versión 5: Uso de **intrínsecos**

En cada versión incluye la implementación de un filtro FIR, dependiendo de la versión, y la lectura de los valores de los coeficientes (Coeficientes.csv) y los valores de los datos de entrada (musica4.csv).

En cuanto al profiling se optó por usar *clock()*, que es una función que mide el tiempo de ejecución de un programa. En principio se pensó en usar *gprof*, que es una herramienta de análisis de rendimiento para

aplicaciones UNIX, pero no se pudo ya que los tiempos de ejecución eran muy pequeños y no se podía medir correctamente.

Además, se ha calculado los ciclos de reloj de cada versión del filtro FIR. Para ello, se ha creado una función utilizando la instrucción *rdtsc* (Real Time Stamp Counter) que mide el número de ciclos de reloj desde el último reinicio. La función es la siguiente:

```
1 uint64_t rdtsc(){
2     unsigned int lo, hi;
3     __asm__ __volatile__("rdtsc" : "=a"(lo), "=d"(hi));
4     return ((uint64_t)hi << 32) | lo;
5 }
```

Lo que hace la función es utiliza la instrucción de ensamblador *rdtsc* para leer el contador de tiempo de la CPU, y devuelve este valor como un número de 64 bits pero dividido en dos partes de 32 bits, *lo* es la parte baja y *hi* es la parte alta. Entonces, se ejecuta la instrucción *__asm__ __volatile__* que es una instrucción de ensamblador en línea que se utiliza para incrustar código ensamblador en el código C.

El *profiling* se ha hecho de la siguiente forma:

```
1 start_cycle = rdtsc();
2 start = clock();
3 // --- Código a medir ---
4 end = clock();
5 end_cycle = rdtsc();
```

Por otra parte, para calcular el tiempo de ejecución y los ciclos de reloj se ha repetido *N* veces, para obtener un promedio de los valores. En este caso, se ha decidido hacer 1000 repeticiones, pero se puede cambiar el valor de *N* ya que se define como una constante en cada versión del filtro FIR.

```
1 #define REPETICIONES 1000
2
3 // Aplicacion del filtro FIR
4 for (i = 0; i < REPETICIONES; i++) {
5     start_cycle = rdtsc();
6     start = clock();
7     firfilter(vector_coef, vector_in, result);
8     end = clock();
9     end_cycle = rdtsc();
10
11     elapsed = (double)(end - start) * 1000.0 / CLOCKS_PER_SEC;
12     mean_time[i] = elapsed;
13     mean_cycles[i] = end_cycle - start_cycle;
14 }
```

3. Implementaciones

Lectura de los coeficientes

Para leer los coeficientes del filtro FIR se ha creado la siguiente función:

```
1 float* inicializacion_coeficientes() {
2     float* vector_coeficientes = (float*)malloc(COEF * sizeof(float));
3     int i = 0;
4     FILE* fich_coef = fopen("../data/Coeficientes.csv", "r");
5     if (fich_coef == NULL) {
6         printf("Error al abrir el archivo\n");
7         exit(1);
8     }
9
10    while (fscanf(fich_coef, "%f", &vector_coeficientes[i]) != EOF && i < COEF) {
11        i++;
12    }
13    fclose(fich_coef);
14
15    return vector_coeficientes;
16 }
```

Lo que se hace es abrir el archivo *Coeficientes.csv* que contiene los valores de los coeficientes del filtro FIR, y se lee cada valor hasta que se llegue al final del archivo o se haya leído todos los valores. Luego, se cierra el archivo y se devuelve el vector de coeficientes.

Lectura de los datos de entrada

Para leer los datos de entrada del filtro FIR se ha creado la siguiente función:

```
1 float* inicializacion_vector_in() {
2     float* array_data = (float*)malloc(N * sizeof(float));
3
4     int i = 0;
5     FILE* file_data = fopen("../data/musica4.csv", "r");
6     if (file_data == NULL) {
7         printf("Error al abrir el archivo\n");
8         exit(1);
9     }
10
11    while (fscanf(file_data, "%f", &array_data[i]) != EOF && i < N) {
12        i++;
13    }
14
15    fclose(file_data);
16
17    return array_data;
18 }
```

Para la lectura de los datos de entrada se hace lo mismo que en la lectura de los coeficientes, se abre el archivo *musica4.csv* que contiene los valores de los datos de entrada, se leen los valores hasta que se llegue al final del archivo o se haya leído todos los valores y se cierra el archivo.

3.1. Versión base

Para la implementación base se ha hecho la función *firfilter()* que recibe los coeficientes del filtro FIR y los datos de entrada como punteros a float, y devuelve un puntero a float con los valores de la salida del filtro FIR. La función es la siguiente:

```
1 float* firfilter(float* vector_coef, float* vector_data) {
2     float* result = (float*)malloc((N + COEF - 1) * sizeof(float));
3     int i, j;
4     for (i = 0; i < N + COEF - 1; i++) {
5         result[i] = 0;
6         for (j = 0; j < COEF; j++) {
7             if (i - j >= 0 && i - j < N) {
8                 result[i] += vector_coef[j] * vector_data[i - j];
9             }
10        }
11    }
12
13    return result;
14 }
```

Lo que hace la función es inicializar un vector de float con el tamaño de la suma de los tamaños de los vectores de coeficientes y de los datos de entrada. Luego, se inicializa un bucle que recorre el vector de salida, y dentro de este bucle se inicializa otro bucle que recorre el vector de coeficientes. Dentro de este segundo bucle se comprueba si el índice de la salida menos el índice del bucle interno es mayor o igual a 0 y menor que el tamaño del vector de datos de entrada. Si se cumple la condición, se multiplica el valor del coeficiente por el valor del dato de entrada y se suma al valor de la salida.

3.2. Versión 1

Para la versión 1 se ha mejorado la implementación base del filtro FIR. Lo que se ha hecho es trabajar directamente con los punteros de los vectores de coeficientes, de los datos de entrada y de la salida. Usar punteros directamente y no devolver un puntero a float con los valores de la salida del filtro FIR es más eficiente. Además, se ha eliminado la condición del bucle interno, ya que se ha comprobado que no es necesario.

```
1 void firfilter(float* vector_coef, float* vector_data, float* result) {
2     int i, j;
3     for (i = 0; i < N + COEF - 1; i++) {
4         result[i] = 0;
5         for (j = 0; j < COEF; j++) {
6             result[i] += vector_coef[j] * vector_data[i - j];
7         }
8     }
9 }
```

3.3. Versión 2

4. Bibliografía

Bibliografía

- [1] Ng Pante, C. (2001). Titulo. Nombre pagina web. Recuperado de <http://url.com>
- [2] [https://es.wikipedia.org/wiki/FIR_\(Finite_Impulse_Response\)](https://es.wikipedia.org/wiki/FIR_(Finite_Impulse_Response))
- [3] https://es.wikipedia.org/wiki/Diseo_de_Filtros_de_Respuesta_Finita_al_Impulso
- [4] <https://en.wikipedia.org/wiki/Gprof>