CONCEPCION DE UN CIRCUITO INTEGRADO DEDICADO A LA CONVERSION DE FRECUENCIA DE MUESTREO DE UNA SEÑAL DE AUDIO

Ing. Jaime Rivadeneira. Escuela Politécnica Nacion

RESUMEN

El disco compacto es el sistema audio-digital más difundido y su señal está muestreada a 44,1 kHz, en tanto que las consolas audio-digitales profesionales tratan las señales a 48 kHz. Se ve que hay un problema de incompatibilidad entre estos dos sistemas a causa de la diferencia de frecuencias de las señales muestreadas. El tema de este trabajo es la concepción de un circuito integrado que establezca la comunicación entre dichos sistemas.

Se ha realizado la síntesis y la simulación del mismo circuito usando la tecnología VSC 370, 1μ del paquete COMPASS, obteniéndose un resultado particularmente atractivo en cuanto a la velocidad de cálculos del circuito, así como su talla, consumo de potencia e insignificante precio.

ABSTRACT

Compact discs (CDs) are nowadays the most popular audio-digital systems. Their signal is sampled at 44.1 kHz, while professional audio-digital equipment treats signals at 48 kHz. This incompatibility problem, due to different sampling frequencies, is the subject of this paper. An integrated circuit (IC) is developed to establish communication between these two audio-systems.

The synthesis and simulation of the proposed IC are done using VSC 370, 1μ technology, from the COMPASS program. This yields interesting results from the point of view of IC computational speed, size, power consumption and price.

INTRODUCCION

Siendo la conversión de frecuencia de muestreo un problema frecuentemente *encontrado en técnicas de audio, nuestra preocupación es la concepción de un circuito* integrado que realice la conversión de *frecuencia de muestreo de una señal de* audio. Específicamente: A la entrada del circuito se dispone de una señal muestreada a 44,1 kHz proveniente de un disc compacto, y a la salida se tendrá la mism señal pero muestreada a 48 kHz.

Esto permitiría la copia fiel del contenido un disco compacto a un cassette digital, cos que no ha sido posible hasta ahora. En efec la normalización de 1986 preveía que codificado digital sobre la consola grabador debía hacerse sobre 48 kHz, mientras qu sobre el disco compacto es de 44,1 kHz. Siendo 44,1 y 48 dos números primos entre sí (esto se lo había hecho a propósito...), la copia fiel era imposible.

Otras soluciones han sido ya presentadas a respecto, específicamente el trabajo *"Etude de la conversion de fréquence d'échantillonnage sur les consoles numériques"*, junio de 1991 [1], el cual es la base del presente estudio, ya que nuestra tarea consiste en concebir un circuito integrado que realice el algoritmo indicado en dicho trabajo.

Para hacerlo, habrá que optimizar el mencionado algoritmo para disminuir su tiempo de ejecución, eliminando todos los cálculos "inútiles" (multiplicaciones por cero y otras operaciones semejantes...), y después sintetizar el circuito minimizando la superficie de silicio necesaria al circuito, bajo las imposiciones de cálculos y los elementos de la biblioteca a utilizarse.

I. LA CONVERSIÓN DE FRECUENCIA. ALGORITMO DE BASE

1.1 La Interpolación

El proceso de conversión de frecuencia de una señal digital, consiste en modificar su frecuencia de muestreo, conservando constante su contenido espectral.

Cuando se pasa de una señal a otra, cuya frecuencia de muestreo es más elevada, se habla de INTERPOLACION.

El proceso de Interpolación de una señal in(n) *por un factor L, simbolizado por una flecha* apuntando hacia arriba *[1], impone la* inserción de L-1 muestras de valor nulo entre *cada par de muestras de la señal in(n).*

Sea e(m) el conjunto de muestras obtenidas

és de la Interpolación, lo que implica

={ in(m/L),
para m = 0,L,2L,...(señal causal)
0, para cualquier otra m;

otra manera:

)={ in(k), si k = 0,1,2,...
0, para cualquier otra k.

a bien, este proceso no modifica la señal oral, puesto que entre cada par de tras *in(n) no había ninguna información.*

mbargo esto no es válido con respecto spectro de frecuencia, puesto que la ción de ceros "de relleno" es equivalente mentar la frecuencia de muestreo y ecerán espectros imágenes que se rán en la banda pasante, y que deben ser nados. Esto podrá hacerse únicamente a ayuda de un filtro pasa-bajos que debe características muy precisas [2].

figura 1.1.1 muestra un ejemplo de polación por 4 de una señal in(n), esto es , y los símbolos utilizados.

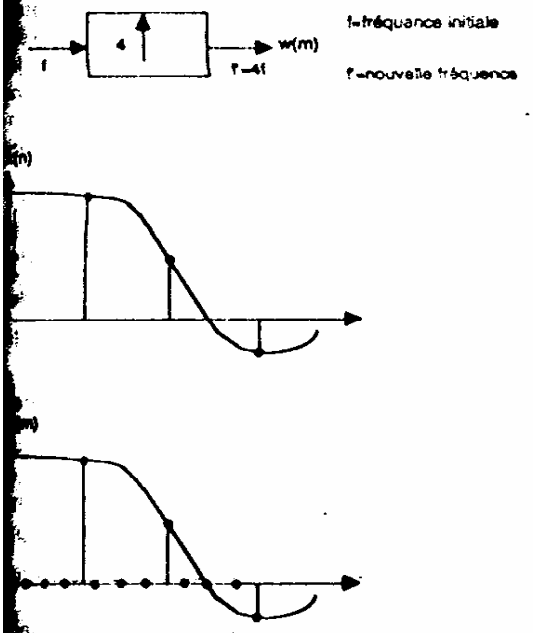

Figura 1.1.1

El Filtraje

gura 1.2.1 muestra el esquema de filtraje de una señal e(l).

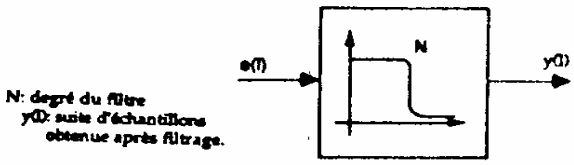

Figura 1.2.1

La figura 1.2.2 muestra los dos procesos de Interpolación y Filtraje en general, donde:

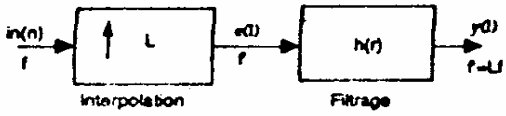

Figura 1.2.2

in(n): conjunto de muestras de entrada
e(l): conjunto de muestras después de la Interpolación
L: Factor de Interpolación
f: frecuencia original
f': nueva frecuencia alcanzada después de la Interpolación
y(l): conjunto de muestras de salida después de la Interpolación-filtraje
h(r): conjunto de coeficientes del filtro.

Utilizamos los filtros FIR (Finite Impulse Response), puesto que ellos tienen ventajas muy interesantes en lo que nos concierne: Posibilidad de fase lineal, ausencia de ciclos límites (no hay bucles de reacción), sensibilidada reducida y buenas condiciones en cuanto a ruido. En general la implementación de un filtro FIR no presenta problemas importantes ligados a la sensibilidad, cualquiera que sea su grado.

El conjunto de muestras obtenido después del filtraje, utilizando los filtros FIR está dado por:

$$y(n) = \sum_{k=0}^{N-1} h(k) * e(n-k) ,$$

donde h(k) tiene talla N, y vale cero para $k<0$ y $K>N-1$.
El cálculo de una muestra de salida y(n) se efectúa a partir de la muestra de entrada presente e(n) y de las N-1 muestras de entrada precedentes.
Para calcular un solo valor de y(n), son

necesarias N muestras de e( ).

1.3 La Decimación

El proceso que consiste en disminuir la frecuencia de muestreo de una señal numérica se llama DECIMACION. Así, la Decimación de una señal numérica x(n) por un factor M se realiza conservando solo una muestra cada M muestras. En este caso, una señal x(n) es transformada en otra señal y(m) tal que:

y(m) = x(mM), donde "m" es un entero.

Podemos simbolizar el proceso de la Decimación por una flecha apuntando hacia abajo [1].
Esta disminución de frecuencia de muestreo implica una reducción de la banda pasante disponible. Por lo tanto se vuelve *imprescindible filtrar la señal original antes de* hacer la Decimación, a una frecuencia superior al doble de la más alta frecuencia de la señal (teorema de *Shannon*).
Esto debe realizarse con la ayuda de un filtro *digital* pasa-bajos para evitar el fenómeno de recubrimiento que impide la reconstrucción ulterior de la señal. Este filtro debe tener, al igual que en el caso de la Interpolación, características muy precisas [2].

La figura 1.3.1 muestra un ejemplo de Decimación por 4 de una señal x(n), entonces M = 4, y los símbolos utilizados.

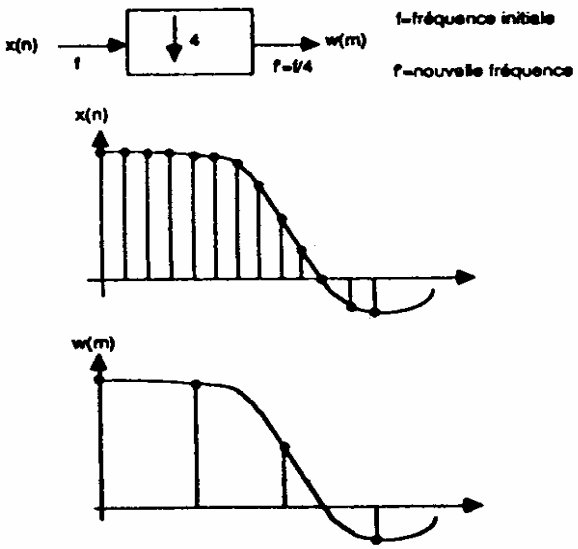

Figura 1.3.1

La figura 1.3.2 muestra los dos pr
Filtraje y Decimación.

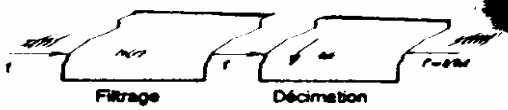

Figura 1.3.2

Podemos imaginar pasar de una frecuencia cualquiera de muestreo a otra frecuencia, interpolando la primera hasta el múltiplo común entre las dos frecuencias y decimando luego esta última para alcanzar la frecuencia deseada, a condición de encontrar este *múltiplo común* a las dos frecuencias, lo que teóricamente siempre es posible.

Así, si deseáramos realizar una conversión de frecuencia de muestreo por un factor L/M, bastará con poner en cascada los dos *procesos de Interpolación-decimación*, es decir, habrá que *primero* [2] incrementar esta frecuencia de muestreo por L para enseguida disminuirla por M.

Además, ya que es necesario filtrar la señal *después* de la Interpolación y *antes* de la Decimación, es natural el fusionar estas dos etapas de filtraje (de misma frecuencia Lf), para no tener sino un solo filtro pasa-bajos, que debe evidentemente responder al más exigente de los dos esquemas.
La figura 1.3.3 muestra en símbolos los dos procesos indicados.

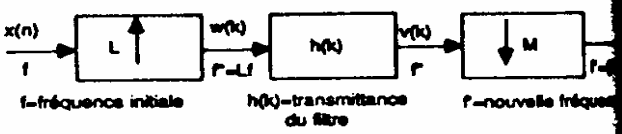

Figura 1.3.3

y(m) es la señal obtenida después del proceso de Interpolación-filtraje y Decimación.

II. PRIMERA SOLUCION PROPUESTA

Basándose en el estudio anterior, P. Colin ha propuesto un interface como el de la figura 2.1.1 para realizar la conversión de frecuencia, obteniéndose a la salida una señal muestreada a 48 kHz, teniendo como entrada una señal muestreada a 44,1 kHz.

Se ha demostrado que una estructura multi-etapas es la más adecuada a implementar (el volumen de cálculos y el espacio requerido

para almacenar los datos disminuye, los filtros son más fáciles de implementar, etc), siendo 4 el número óptimo de etapas.

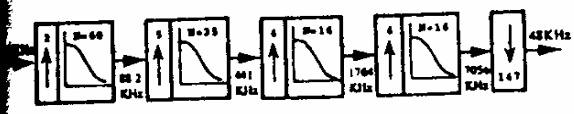

Figura 2.1.1

En cada etapa implementada se realiza el mismo proceso (Interpolación-filtraje) ya que hay que *subir* hasta 7056 kHz (= 44,1 x 160), es decir hay que interpolar la señal de entrada por 160 (= 2 x 5 x 4 x 4). Nótese que 7056 es el mínimo común múltiplo entre 44,1 y 48.
En la misma figura pueden verse también el número de coeficientes de cada filtro.

Enseguida se tiene una última etapa que realiza la Decimación por 147 para alcanzar la frecuencia deseada de 48 kHz (=7056/147) a la salida.

Este algoritmo completo presentado por P. Colin implica 136 millones de cálculos (multiplicaciones-adiciones) por segundo, y 564 millones de accesos en memoria por segundo, pero un estudio preciso demuestra que solamente son necesarios 5,1 millones de cálculos por segundo, y 11,7 millones de accesos en memoria por segundo (!).
Los cálculos "inútiles" resultan de las multiplicaciones por cero, debidas a la introducción de ceros para la Interpolación.

El interés está justamente en la optimización de este algoritmo para la implementación de otro que realice únicamente los cálculos "necesarios".

III. RECURRENCIA MATEMATICA DEL ALGORITMO

Para distinguir cada etapa de Interpolación-filtraje se usan subíndices; así, $e_1(m)$, $y_1(m)$, $h_1(m)$, N_1 para lo referente a la primera etapa; $e_2(m)$, $y_2(m)$, $h_2(m)$, N_2 para la segunda, etc.

Veamos un ejemplo de simplificación de las fórmulas utilizadas por P. Colin, y que ha servido para la implementación de nuestro algoritmo.

En la tercera etapa se tiene que la fórmula general es:

$$y_3(m) = \sum_{k=0}^{13} h_3(k) \cdot e_3(m-k) ,$$

donde

$e_3(m) = y_2(m/4)$, si m = 0,4,8,...
0, para cualquier otra m

puesto que e_3 es la serie de muestras obtenida a partir de la Interpolación por 4, esto es, se han introducido 3 muestras de valor cero entre cada par de muestras consecutivas de y_2, y hay que eliminarlas para disminuir el número de cálculos. Para lograrlo vamos a dividir la serie e_3 en 4 sub-series:

$$y_3(4i) = \sum_{k=0}^{12} h_3(k) \cdot y_2\left(\frac{4i-k}{4}\right)$$

para 4i-k = 0,4,8,...;

esto es, k = 4r (r es un entero),es decir,

$$y_3(4i) = \sum_{r=0}^{3} h_3(4r) \cdot y_2(i-r) , \quad ... (3.1)$$

La sub-serie siguiente dará:

$$y_3(4i+1) = \sum_{k=1}^{13} h_3(k) \cdot y_2\left(\frac{4i+1-k}{4}\right)$$

para 4i + 1-k = 0,4,8..., esto es, k = 4r + 1, es decir,

$$y_3(4i+1) = \sum_{r=0}^{3} h_3(4r+1) \cdot y_2(i-r) , \quad ... (3.2)$$

De la misma manera obtendremos para la siguiente sub-serie de las y_3:

$$y_3(4i+2) = \sum_{k=2}^{14} h_3(k) \cdot y_2\left(\frac{4i+2-k}{4}\right),$$

para 4i + 2-k = 0,4,8..., esto es, k = 4r + 2, es decir,

$$y_3(4i+2)=\sum_{r=0}^{3} h_3(4r+2).y_2(i-r),....(3.3)$$

Finalmente para la serie de las $y_3(4i+3)$:

$$y_3(4i+3)=\sum_{k=3}^{15} h_3(k).y_2(\frac{4i+3-k}{4}),$$

lo que da (para $k=4r+3$),

$$y_3(4i+3)=\sum_{r=0}^{3} h_3(4r+3).y_2(i-r),.....(3.4)$$

Nótese que se han añadido $h3(14)=h3(15)=0$, puesto que si el número de coeficientes del filtro es un múltiplo del factor de Interpolación (4 en este caso), las fórmulas a implementar son más simples. Habían 14 coeficientes al principio, y hemos añadido dos valores nulos con el propósito indicado.

Las fórmulas precedentes 3.1, 3.2, 3.3 y 3.4 para esta etapa pueden fusionarse como:

$$y_3(m)=\sum_{r=0}^{3} h_3(4r+m\%4).y_2(m/4-r),$$

donde usamos los símbolos "%" para *módulo* (Ej: 15%4 = 3; 27%3 = 0,...), y "/" para la *división entera* (Ej: 15/4 = 3; 13/3 = 4,...).

Esta última fórmula sólo toma los valores "útiles" de la serie y_2, habiéndose eliminado los valores nulos para los cálculos. Esto implica que $y_3(\ )$ tiene solamente los valores necesarios para la etapa siguiente.

Procediendo exactamente de la misma manera con todas las etapas se encontraron las siguientes fórmulas:

$$y_1(m)=\sum_{r=0}^{29} h_1(2r+m\%2).\epsilon(m/2-r),$$

$$y_2(m)=\sum_{r=0}^{8} h_2(5r+m\%5).y_1(m/5-r),$$

$$y_3(m)=\sum_{r=0}^{3} h_3(4r+m\%4).y_2(m/4-r),$$

$$out(m)=\sum_{i=0}^{3} h_4(4i+(147m)\%4).y_3((147m)/4-i).$$

Nótese que se ha incluido a la salida el factor 147 debido a la Decimación, y que es equivalente a $out(m)=y_4(147m)$, reemplazando luego y_4 en función de las y_3.

A pesar de su apariencia simple, estas recurrencias matemáticas son difíciles de manejar, y por eso se tuvo que utilizar además el juego de fórmulas que se indica a continuación, obtenido después de un análisis minucioso de la evolución de los índices de cada una de las series y1, y2, y3 e y4. Se partió de los índices a obtener en la serie y4 para luego regresivamente establecer los índices necesarios de las series de las etapas precedentes.

Así se obtuvo:

si p = 147m,
y3_max = p/4, y3_min = p/4-3,
y2_max = p/16, y2_min = y3_min/4-3,
y1_max = p/80, y1_min = (y2_min + 5)/5-7.

Así por ejemplo, si m = 5:

p = 147m = 735 (índice de y4),
y3_max = 735/4 = 183,
y3_min = y3_max-3 = 180, es decir los índices de y_3 variarán de 180 a 183,
y2_max = 735/16 = 45, y2_min = 180/4-3 = 42 (los índices de y_2 variarán de 42 a 45), y los índices de y_1 variarán de 2 a 9.

Así se pudo elaborar una tabla que contiene solamente los índices *útiles* de las series de muestras, en base a la cual se logró obtener el último juego de fórmulas.

IV. VELOCIDAD DE CALCULOS ALCANZADA

Las fórmulas establecidas en la sección precedente fueron utilizadas para la implementación del algoritmo, escrito en lenguaje C puesto que éste tiene, entre otras ventajas, la de permitir una descripción suficientemente cercana del *hardware* del

circuito.
La señal de entrada es procesada por bloques sucesivos compuestos de 147 muestras que, tratadas una a una permiten obtener 160 muestras a la salida.
La entrada y la salida son ficheros codificados sobre 16 bits.

Para los ensayos se utilizaron dos ficheros: "madonna.in" (un extracto de una canción de la conocida estrella Madonna, de una talla de 1,28 Mbytes, de una duración de 13 segundos aproximadamente), y "classic.in" (un extracto de música clásica) de talla y duración semejantes.

Los resultados obtenidos fueron notables: Según el programa de base [1] el tiempo empleado para tratar el fichero "madonna.in" fue de 2 horas 15 minutos aproximadamente, mientras que el algoritmo, objeto de este estudio, sólo tomó ¡una decena de minutos! para obtener un mismo fichero de salida.

Con esto, la primera etapa del trabajo (una disminución drástica del tiempo de ejecución del algoritmo y de los accesos en memoria mediante eliminación de los cálculos "inútiles") estaba lograda.
Ahora la preocupación es la implementación y optimización del circuito que realice el algoritmo en C encontrado.

V. METODOLOGIA DE LA CONCEPCION DEL CIRCUITO

5.1 Procedimiento básico

Implica el establecimiento de una descripción funcional del circuito, bajo la forma de un algoritmo escrito en lenguaje algorítmico, C en nuestro caso. Este es luego refinado hasta la obtención de un programa C bajo la forma ASM, a la cual corresponde una arquitectura material precisa.

Además es necesaria la elaboración de un *Camino de Datos* (Data Path) y de su correspondiente *Unidad de Control* específicos al circuito. Esta última se representa a nivel de *Máquina de Estados* [3] bajo la forma de una Tabla de verdad.

La Unidad de Control asegura el secuenciamiento definido por el programa de especificación, y el Camino de Datos se ocupa de los valores numéricos de las variables.

El Camino de Datos envía variables de test a la Unidad de Control, quien a su vez influye en las operaciones del Camino de Datos mediante variables de comando.

Finalmente deberíamos tener acceso a un paquete de CAO (COMPASS en nuestro caso) para el diseño gráfico del circuito: herramientas de síntesis lógica y captación esquemática, compiladores de células (RAM y ROM), compiladores de Caminos de Datos, herramientas de verificación, etc.

5.2 Descripción funcional del circuito

Cualquier programa de especificación del circuito, escrito en lenguaje de alto nivel (lenguaje C por ejemplo), puede ser reescrito bajo otro formato llamado ASM (Algorithmic State Machine) y comprende instrucciones del tipo general:

N_i: if (t_j) { σ_{i1}; goto N_k; }
else { σ_{i2}; goto Nl; };

ó también: N_i: σ_i; goto N_m; donde:

$N_{i,k,l,m}$ son etiquetas (*labels*) de instrucciones,
t_j es el test a efectuar,
$\sigma_{i1,i2,i}$ son instrucciones a ejecutar.

En un momento dado del cálculo, el estado se caracteriza a la vez por la instrucción en ejecución y por los valores numéricos de los datos.

Así por ejemplo, el grupo de instrucciones:

```
while (j< =29)
      {
      acc=acc+X*Y; X=ROM[romi]; Y=RAM[rami];
      j++; romi++; rami=(rami+1)%30;
      }
RAM[base1]=(int) (acc>>14);
```

se transforma en:

```
N5: if (j< =29) goto N6;
    else        goto N8;
N6: acc=acc+X*Y; X=ROM[romi]; Y=RAM[rami]; goto N7;
N7: j++; romi++; rami=(rami+1)%30; goto N5;
N8: RAM[base1]=(int) (acc>>14); goto Ni;

Ni: ......
```

5.3 Transformaciones del programa

Necesitamos adaptar el programa para una implantación material, por lo que debemos:

* Suprimir las operaciones difíciles de realizar

en hardware, vg. una división por 80, la extracción de un módulo 35, etc. Se llega a este resultado añadiendo variables de test, es decir reorganizando los cálculos.

• Transformar las operaciones difíciles de realizar en hardware en una secuencia de operaciones más simples: Así, la realización de una multiplicación mediante simples operaciones de adición y desfase (algoritmo de Boot por ejemplo). Esta transformación se llama *Secuencialización*. Resulta así una simplificación de *recursos* (reemplazamos vg. un multiplicador por un sumador y defasador) contrarrestada por una disminución de la velocidad de ejecución.

Un *recurso* es un circuito combinatorio particular adaptado a los cálculos numéricos o lógicos. Así, un multiplicador, un ALU, un sumador-sustractor, las compuertas lógicas, etc. De hecho un *recurso* es implementado a partir de células elementales, que representan los cálculos sobre un bit, las cuales se repiten tantas veces cuantos lo indica el número de bits requerido.

• Disminuir el número de ciclos paralelizado la ejecución de ciertas operaciones. Esta transformación es llamada *Paralelización* y tiene por meta la de utilizar mejor el hardware y acelerar los cálculos, así como el de hacer que éstos sean terminados en el lapso de tiempo impartido.

Se duplica el hardware del circuito para que éste pueda trabajar en tiempo real.

A todo lo largo de la síntesis del circuito se trata de agrupar varios *comportamientos* en un solo *recurso*, proveyendo a este último de uno o varios bits de control, lo que permite una disminución de la superficie de silicio.

Esto implica que los *comportamientos* atribuidos a un mismo *recurso* no deben realizarse al mismo tiempo.

5.4 Síntesis del Camino de Datos y de la Unidad de control

Para la concepción del Camino de Datos se tiene que tomar en cuenta el siguiente proceso:

1. Determinar las memorias requeridas: las variables de un programa no son utilizadas constantemente durante el curso de la ejecución. Una variable es llamada *viva* cuando su contenido es utilizado en el desarrollo del programa, y es llamada *muerta* en el caso contrario. El análisis del tiempo de vida permite determinar el número mínimo de registros o elementos de RAM de que se debe disponer para almacenar las variables que están simultáneamente vivas.

2. Asignación de memorias: que consiste en la determinación exacta de cuáles registros contendrán cuáles variables en las diferentes partes del cálculo.

3. Determinación de los diferentes comportamientos requeridos: es necesario establecer la lista completa de todas las funciones aritméticas (adiciones, sustracciones, productos, etc.), así como de las funciones especiales (módulos, defasajes, etc.) utilizadas en el programa. Es necesario conocer todas las transformaciones aplicadas a las variables, por cada línea del programa.

4. Determinación de las interconexiones entre los elementos memorias y los recursos.

5. Realización de una microprogramación horizontal. Esta etapa tiene el propósito de disminuir el *Ancho de Control* (es decir el número de bits necesarios para manejar el Camino de Datos) de la Unidad de Control propiamente dicha. Así, la célula "barrel shift" de COMPASS posee 7 bits de control, pero en nuestro caso para este *recurso* sólo necesitamos utilizar 4 comportamientos. Una etapa de lógica puesta antes del "defasador" permite reducir el número de bits a 2, sin aumentar el tiempo de respuesta del cálculo del *recurso*.

La Unidad de Control se encarga del secuenciamiento de las operaciones que el Camino de Datos puede realizar, esto es, asegura el secuenciamiento definido por el programa.

Para su síntesis hay que:

1. Escoger un tipo de programa: para evitar memorizar por cada *label* dos comandos (lo que aumenta al doble la talla del control) hay que limitarse a cierto tipo de instrucciones. Se obtienen así un programa *condicional, condicional restringido, incremental*, etc. presentando cada uno sus ventajas e inconvenientes [4]. Se puede igualmente crear un conjunto personal de

instrucciones.

2. Hacer una lista de los controles del Camino de Datos: la selección de un comportamiento de un recurso se lo hace por intermedio de bits de control. En cuanto a los registros, cada uno de ellos posee un bit de control llamado *load* del registro que puede estar a 1 (load activo), o a 0 (load inactivo).
Para las conexiones, cada multiplexor, cada circuito emisor o receptor apunta a los datos según los valores impuestos sobre los controles.

3. Realizar una tabla de control la que está compuesta por palabras de control (σ_{i1}, σ_{i2}) y estados siguientes posibles (N_i, N_k) del programa. Por cada línea de programa y por cada rama de test se establecen los bits de control necesarios para realizar la instrucción, y se codifica el estado siguiente.

Diversas ténicas permiten la concepción de las unidades de control, como la lógica cableada (llamada también lógica multi-niveles), la lógica que utiliza las redes lógicas programables (PLA o ROM). Dichas técnicas tienen sus ventajas e inconvenientes [5] según la aplicación deseada. La selección se realiza simplemente compilando la tabla de control hacia las diferentes soluciones, y comparándolas desde el punto de vista de superficie, consumo de potencia y velocidad.

VI. EL PROBLEMA DE LA SINTESIS DEL CIRCUITO

6.1 Alcance de la síntesis

El objetivo de la síntesis es la implantación de un circuito que tenga la mínima superficie de silicio bajo la imposición de realizar un cálculo dado en un tiempo fijado, y bajo la imposición de los elementos de biblioteca a utilizarse. Para alcanzar dicho propósito es necesario:

- Seleccionar el algoritmo de partida.
- Seleccionar la secuencialización o la paralelización de los cálculos.
- Seleccionar los diferentes recursos (vg. la elección entre un multiplicador y un algoritmo que permita la realización de la multiplicación a partir de un sumador, la selección de un almacenamiento en una RAM antes que en un registro).
- Seleccionar el tipo de implementación de la Unida de Control (tipo de instrucción, tipo de implantación).
- Selección entre las diferentes implantaciones de los recursos existentes en bibliotecas o de una implantación a nivel de *layout*.

6.2 Imposiciones para la síntesis

Estas imposiciones reducen el campo de síntesis a explorar. Tenemos así:

1. Imposiciones de Biblioteca: La concepción de un circuito se hace con una tecnología dada la cual va a condicionar grandemente la síntesis. Para la realización de las operaciones se llama a operadores de la tecnología escogida. Hemos utilizado la tecnología VLSI 1μ, la cual es capaz de realizar típicamente:

- Una adición de 2 números de 16 bits en 20 ns con un sumador simple.
- Una adición de 2 números de 32 bits en 35 ns con un sumador simple.
- Una adición de 2 números de 32 bits en 20 ns con un sumador rápido.
- Una multiplicación de 2 números de 16 bits en 60 ns.
- Una multiplicación de 2 números de 16 bits en 10 ns con un multiplicador de 10 niveles de *pipeline* (2000 compuertas).
- Una multiplicación-adición de 2 números de 32 bits en 80 ns.
- Un acceso a la RAM o ROM 128x16 en 40 ns, etc.

Una síntesis que exige una más grande superficie permite a menudo un cálculo más rápido que un circuito más pequeño. Una buena arquitectura resulta de este compromiso superficie/velocidad. Los operadores existen ya sintetizados en las librerías de VLSI. Por ejemplo, un multiplicador de 2 números de 16 bits, ocupa una superficie de 1,17 mm^2.

2. Imposiciones de cálculo: Nuestro algoritmo necesita el siguiente volumen de cálculo por segundo:

5,1 millones de multiplicaciones-adiciones, 11,7 millones de accesos en memoria y 20,4 millones de cálculos sobre las direcciones.

El análisis del algoritmo muestra que una gran parte (no todo...) de las operaciones de cálculos sobre las direcciones pueden ser ejecutados en paralelo con las multiplicaciones-adiciones.

6.3 Solución propuesta

A la vista de las cifras anteriores, está claro que el uso de un multiplicador es necesario, ya que la utilización del algoritmo de Booth conduciría a un total de 50 millones de operaciones, esto es a un ciclo mínimo de 20 ns.

Igualmente para realizar las adiciones es necesaria la utilización de un sumador a 32 bits.

La frecuencia mínima es de 5 MHz ya que tenemos que realizar aproximadamente 5 millones de multiplicaciones por segundo.

Por otro lado, la frecuencia máxima del circuito está condicionada por el tiempo de reacción del multiplicador (70 ns), es decir 16 MHz.

Un análisis más realista indica que no es posible realizar 1 operación útil por ciclo sino más bien una operación por cada 1,5 o 2 ciclos, lo que da una frecuencia mínima entre 7,5 MHz y 10 MHz, esto es, podemos imponernos un tiempo de ciclo máximo de 120 ns.

Así pues los valores extremos del tiempo de ciclo son 70 ns <tiempo de ciclo < 120 ns.

Paralelamente a este razonamiento precedente se puede evaluar la longitud máxima de ciclo requerida para ejecutar una instrucción de programa. Así, la operación más larga del programa es:

```
acc = acc + ROM[romi + j]*RAM[y10 + (rami + j) % 8];
```

la cual, asumiendo los retardos mínimos para la realización de cada cálculo, necesitaría 200 ns, por lo que se ve la necesidad de introducir un nivel de *pipeline* para la implementación de dicha operación, y poder mantener el tiempo de ciclo impuesto.

Igualmente la introducción de registros para las variables, nos permitirá disminuir el tiempo de cálculo por ciclo. Así para la realización de la misma operación anterior, introduciendo registros nos da:

```
X = ROM[romi + j] ; Y = RAM[y10 + (rami + j) % 8] ;
acc = acc + X*Y ;
```

y se sabe que el último cálculo sólo requiere 80 ns aproximadamente, lo que es inferior a nuestro ciclo máximo impuesto de 120 ns, y por consiguiente realizable.

Se puede también calcular el número exacto de ciclos para la realización de la conversión de frecuencia. Con la ayuda de Turbo Profiler se obtienen 8700000 ciclos. Entonces la frecuencia mínima de funcionamiento es 8,7 MHz y la máxima es de 12 MHz (que corresponde a los 80 ns para la multiplicación-adición).

6.4 Separación de las operaciones sobre las direcciones y los datos. Repartición de las variables de direcciones

Se puede pensar en la utilización del mismo *recurso* (adición) para efectuar las sumas de los datos y los cálculos de las direcciones, lo que disminuiría la superficie total del circuito. Sin embargo esto no es muy práctico debido al número de bits de las direcciones. Así los datos a la entrada/salida son codificados sobre 16 bits y una multiplicación-adición es ejecutada sobre 32 bits mientras que 7 bits son suficientes para codificar las direcciones.

El análisis del presente algoritmo muestra que las operaciones efectuadas sobre las direcciones son más complejas que las realizadas sobre los datos. Entonces, tratar de hacer compartir el sumador entre las direcciones y los datos aumentaría las conexiones y controles, y por lo tanto la talla del circuito. Así pues, nos conviene separar las operaciones sobre los datos y sobre las direcciones.

La frecuencia de funcionamiento (aprox. 9 MHz) impone que los cálculos de dirección sobre las ram y rom deben realizarse en paralelo. Las variables de direccionamiento no pueden entonces utilizar ni los mismos recursos ni las mismas redes de conexión. Resulta entonces normal sintetizar separadamente los Caminos de Datos correspondientes.
Se tiene entonces la repartición siguiente:

Camino de Datos 1: acc, X, Y, IN, RAM[ ], ROM[ ], OUT
Camino de Datos 2: rami, base0, base1
Camino de Datos 3: k, beta, deltay1, lim1, p
Camino de Datos 4: j, romi
Camino de Datos 5: [ROM], [RAM].

6.5 Otras optimizaciones

Se ha hecho una optimización de *recursos* agrupando todos los *comportamientos* que pueden ser realizados sobre un mismo recurso. Se logra así una disminución sustancial de la superficie de circuito.

Igualmente se ha realizado una optimización de memorias y registros al reutilizar los emplazamientos memoria dejados por las variables *muertas* luego de la terminación de los cálculos de un sub-algoritmo.

No se realizó una optimización de buses de forma global, sino solamente a nivel de Camino de Datos.

VII. LA SINTESIS DEL CIRCUITO PROPIAMENTE DICHA

7.1 Descripción del circuito

La figura 7.1.1 es el esquema general del circuito propuesto de conversión de frecuencia, donde se muestran las interconexiones de los diversos Caminos de

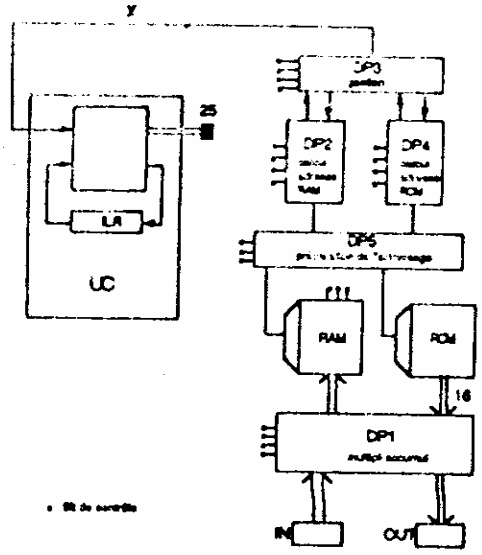

Figura 7.1.1

Datos (DP1, DP2, ... DP5), las memorias RAM y ROM, la Unidad de Control (U.C.), la entrada (IN) y la salida (OUT).

En un momento dado del cálculo, durante 1 ciclo de reloj, la U.C. envía una *palabra de control* hacia el Camino de Datos (DP1 a DP5) y genera el *label* de la instrucción siguiente.

En DP2, DP3 y DP4 se hacen los cálculos de las direcciones de la RAM y ROM. DP5 prepara este direccionamiento y al mismo tiempo (es decir en *paralelo*) DP1 realiza los cálculos sobre los datos, esto es, la multiplicación-adición utilizando los datos de la intrucción precedente.

7.2 *Ejemplo de la síntesis de uno de los bloques: Camino de Datos 5 ([RAM], [ROM])*

El esquema general de este Camino de Datos se encuentra en la figura 7.2.1

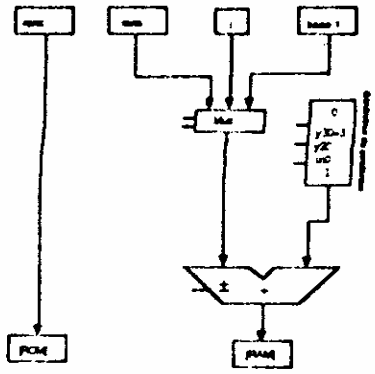

Figura 7.2.1

En este bloque se hacen los cálculos para la preparación del direccionamiento de la [RAM] y de la [ROM]. Utiliza los valores "rami" y "romi" que han sido calculados previamente en DP2 y DP4 respectivamente.

Este bloque posee los bits externos x0, x1, y x2 destinados a controlar los bits internos m0, m1, s0, h0, h1 y h2 para obtener los comportamientos indicados en la tabla 7.2.2 donde:

x2	x1	x0	m0	m1	s0	h0	h1	h2	Comportam.
0	0	0	0	0	1	1	0	0	[RAM]=y30+3-j
0	0	1	0	0	0	0	1	1	[RAM]=y20+j
0	1	1	1	0	0	0	0	0	[RAM]=base1
1	0	1	1	0	0	0	0	1	[RAM]=base1+1
0	1	0	0	1	0	0	0	0	[RAM]=rami
1	0	0	0	1	0	0	1	0	[RAM]=in0+rami

Tabla 7.2.2

m0, m1 son los bits de control del multiplexor que permiten seleccionar j (0,0), base1 (1,0) o rami (0,1);
h0, h1, h2 son los bits de control del generador de constantes que permiten seleccionar 0 (0,0,0), 1 (0,0,1), in0 (0,1,0), y20 (0,1,1) o y30+3 (1,0,0);
s0 selecciona el comportamiento del recurso sumador/sustractor, poniéndose en 0 para la

realización de una suma y en 1 para una sustracción.

Se tienen los *recursos* "gen4_1G", "ad-str7bt" cuyas síntesis se indican en las figuras 7.2.3 y 7.2.4 respectivamente.
"gen4_1G" es un generador de constantes y "ad-str7bt" realiza el comportamiento ±.

Las entradas son rami, romi, j, bas1 que trabajan a 7 bits.
Las memorias y registros son [RAM] y [ROM] igualmente a 7 bits, utilizados para almacenar los cálculos realizados de direccionamiento.

La síntesis completa de este bloque se encuentra en la figura 7.2.5 en donde se puede ver también el bloque "cod_ramjb1" que contiene los bits x0, x1 y x2 indicados anteriormente.

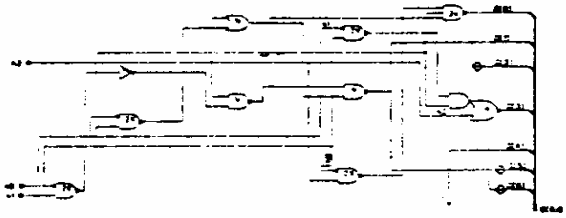

Figura 7.2.3

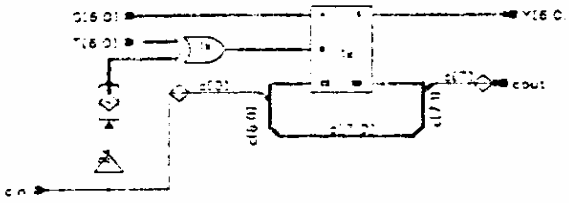

Figura 7.2.4

Se ha procedido exactamente de la misma manera para la síntesis de cada uno de los bloques DP1 a DP4 y se ha realizado luego la compilación de cada uno de ellos así como la verificación respectiva con las herramientas de COMPASS.

7.3 La Unidad de Control

La síntesis de la Unidad de Control se realiza fácilmente a partir de la versión ASM del algoritmo en C. Una vez que los Caminos de Datos están dibujados, un utilitario desarrollado en el Laboratorio de Microelectrónica de la FPMs permite generar automáticamente la Tabla de verdad de esta

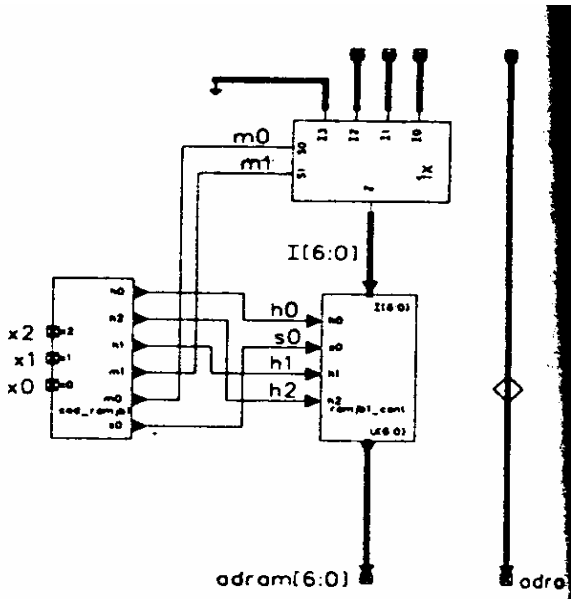

Figura 7.2.5

máquina de estados. Esta descripción es inmediatamente compilable usando la herramienta de síntesis lógica de COMPASS.

La talla de nuestra Unidad de Control, en número de bits está dada por la fórmula:

(tamaño de control Z + tamaño de palabra de estado E + tamaño de palabra de test T + tamaño de palabra de tipo de instrucción I) x número de líneas de control L,

donde:

- "tamaño de control" es el número de bits de control externos del circuito. Reuniendo todos los bits usados en los 5 Caminos de Datos hallamos 21 (luego Z = 21),
- "número de líneas" es el número de líneas del programa principal (*main*) que comienzan por una etiqueta N_i. Este programa posee 70 *labels*, por lo que harán falta 7 bits para codificarlos, luego E = 7,
- "tamaño de la palabra de tipo de test" se refiere al del tipo de test a tomarse en cuenta. En nuestro caso hay 9 tests por lo que nos harán falta 4 bits para codificarlos, esto es T = 4,
- "tamaño de la palabra de tipo de instrucción" se refiere al del tipo de instrucción. Se han implementado 4 tipos de instrucciones en el programa propuesto, por lo que 2 bits son necesarios para su codificación, y entonces I = 2.

Se deducen entonces en total $21 + 7 + 4 + 2 = 34$ bits de control para

nuestra Unidad de Control, lo que implica una talla de 34x70 = 2800 bits.

Una Unidad de Control de esta talla se implanta fácilmente bajo la forma de lógica *multi-niveles*, particularmente atractiva por su bajo consumo de potencia.

VIII. CONCLUSION

El presente trabajo consistía en la búsqueda de una concepción óptima del circuito de conversión de frecuencia 44,1 - 48 kHz.
Para obtenerla, se han realizado muchas transformaciones al algoritmo encontrado, obteniendo así una última versión *altamente* mejorada que ha sido la base para la implementación de dicho algoritmo.

La estimación de la superficie del circuito ha sido realizada usando la tecnología VSC370 1μ, del paquete COMPASS, lo que nos lleva a un cálculo de 40.000 transistores, una superficie de 22 mm^2 y una disipación típica de potencia de 200 mw. Esto incluye las superficies de las conexiones del circuito y las superficies de guarda de los plots. El esquema final del circuito a nivel de *layout* (compilado por COMPASS) se encuentra en la figura 8.1.1.

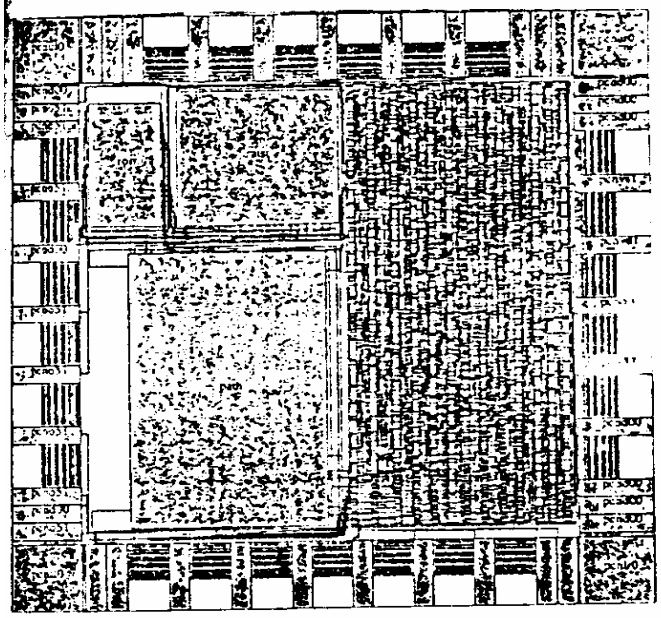

Figura 8.1.1

Estos resultados son notorios desde el punto de vista de la eficiencia del circuito (alta velocidad de cálculos), de su talla y de su precio estimado en una decena de U.S. dólares (aproximadamente 1 U.S. dólar por cada 1000 compuertas), lo que es muy económico en comparación con otras soluciones propuestas a este mismo respecto.

REFERENCIAS

[1] "Etude de la conversion de fréquence sur les consoles numériques". Pierre Colin, Travail de fin d'études, Faculté Polytechnique de Mons, junio de 1991.

[2] "Multirate Digital Signal Processing". Ronald E. Crochiere & Lawrence R. Rabiner, Sec. 2.3 Sampling Rate Conversion - A Direct Digital Approach. Ed. Prentice-Hall.

[3] "Introduction à la Théorie des Systèmes Logiques", Tome I. Hancq J., Service de Théorie des Circuits et de Traitement du signal, Faculté Polytechnique de Mons. 1988

[4] "Digital Systems". Marc Davio, cours Université Catholique de Louvain, 1989.

[5] "Méthodologie d'intégration de circuits spécifiques à partir d'une description algorithmique". N. Prévost & I. Sahinoglu, Tome I, Deuxième partie, Théorie. Travail de fin d'études, junio de 1989, FPMs.

BIOGRAFIA

[illegible] Ingeniera obtuvo el [illegible]tulo de Ingeniero en [illegible]lectrónica y [illegible]omunicaciones en la [illegible]cuela Politécnica [illegible]acional de Quito, en [illegible]re de 1984, [illegible]ión en la que [illegible]ra actualmente como [illegible]fesor principal. [illegible] la [illegible]P.N. realizó estudios de [illegible]stgrad[illegible] la [illegible]aculté Polytechnique de Mons ([illegible]Ms), Bélgica, en donde obtuvo el Grado de Maître en Sciences Appliquées, en Diciembre de 1992.