USO DE TÉCNICAS DE CODISEÑO HARDWARE-SOFTWARE PARA EL DISEÑO DE UN DECODIFICADOR DE MP3.

Federico Salguero, Jorge Osio, Sergio Baron, José Rapallini

Universidad Nacional de La Plata
Centro de Técnicas Analógico Digitales (CeTAD)
Calle 48 y 116, La Plata 1900, Argentina
fsalgue@barcala.ing.unlp.edu.ar, josio@gioia.ing.unlp.edu.ar, josrap@ing.unlp.edu.ar,
s.d.baron@ieee.org

RESUMEN.

Para la eficiente implementación de un decodificador MP3, como la que contempla nuestro proyecto, la elección de una herramienta adecuada para la especificación y generación de código es esencial. El algoritmo de decodificación MP3, requiere necesariamente de actualización periódica de parámetros y comportamiento dinámico de sus bloques funcionales. Por esta razón, el Entorno de Programación PeaCE, basado en el Núcleo de Ptolemy, se presenta como la opción mas apropiada. A través del Dominio SPDF, PeaCE introduce una satisfactoria modificación de la semántica SDF original de Ptolemy. Brindando soporte para el manejo de estados globales, sin que esto se traduzca en la aparición de efectos adversos. En este trabajo analizamos y comprobamos, mediante un ejemplo, la utilidad de esta herramienta para la descripción adecuada de sistemas de procesamiento de señales que requieran la utilización de estados globales y generación de código eficiente.

USING HARDWARE-SOFTWARE CODESIGN TECHNIQUES TO DESIGN AN MP3 DECODER

ABSTRACT.

In order to efficiently implement an MP3 decoder, as presented in our project, choosing an adequate specification and code generation tool, is of great importance. The MP3 decoding algorithm requires periodic updates of parameters and dynamic functional block behavior. Given this, the Ptolemy-based PeaCE programming environment seems to be a good choice. Using the SPDF domain, PeaCE modifies the original Ptolemy SDF semantics, adding support for global states, without side-effects. In this work, and through the use of an example, we analyze and demonstrate the usefulness of PeaCE in describing signal processing systems that require periodic updates of states, and the generation of efficient code.

USO DE TÉCNICAS DE CODISEÑO HARDWARE-SOFTWARE PARA EL DISEÑO DE UN DECODIFICADOR DE MP3.

Federico Salguero, Jorge Osio, Sergio Baron, José Rapallini

Universidad Nacional de La Plata
Centro de Técnicas Analógico Digitales (CeTAD)
Calle 48 y 116, La Plata 1900, Argentina
fsalguero@barcala.ing.unlp.edu.ar josio@gioia.ing.unlp.edu.ar, josrap@ing.unlp.edu.ar, s.d.baron@ieee.org

RESUMEN.

Para la eficiente implementación en Software de un decodificador MP3 (MPEG-1 Layer III), como la que contempla nuestro proyecto, la elección de una herramienta adecuada para la especificación y generación de código es esencial. El algoritmo de decodificación MP3, requiere necesariamente de actualización periódica de parámetros y comportamiento dinámico de sus bloques funcionales. Por esta razón, el Entorno de Programación PeaCE, basado en el Núcleo de Ptolemy, se presenta como la opción más apropiada. A través del Dominio SPDF, PeaCE introduce una muy satisfactoria modificación de la semántica SDF original de Ptolemy. Añade así, soporte para el manejo de estados globales, sin que esto se traduzca en la aparición de efectos adversos. En este trabajo analizamos y comprobamos, mediante un ejemplo, la utilidad de esta herramienta para la descripción adecuada de sistemas de procesamiento de señales que requieran actualización periódica de estados globales y generación de código eficiente.

1. INTRODUCCIÓN.

El objetivo principal del Codiseño es evitar el aislamiento entre los diseños Hardware y Software permitiendo que estos se desarrollen en paralelo y así poder explotar las ventajas de cada alternativa de manera eficiente [3]. La implementación de sistemas de procesamiento de señales, generalmente, mezcla tanto Hardware como Software. Esto conlleva a la complicación del proceso de diseño, forzando a una solución heterogénea [1].

El proceso de diseño de sistemas puede ser visto como una serie de pasos que transforman una especificación al nivel de abstracción en una más detallada hasta llegar a la implementación final. Luego, la manera en que se especifica el comportamiento del sistema afecta el resultado final del diseño [2].

El procedimiento de Codiseño debe contemplar tres pasos fundamentales, la co-especificación, la co-simulación y la co-síntesis de módulos con distintas características [3].

Es por esto que cobra suma importancia el auxilio de herramientas de diseño concebidas de forma tal de proveer al ingeniero de buena expresividad, extensibilidad, capacidad de simulación concurrente y la posibilidad de describir sus sistemas de forma heterogénea (modularidad). En definitiva, lo que se precisa es un entorno de diseño capaz de integrar eficientemente distintas semánticas de descripción y que a su vez, además de la especificación y simulación del prototipo, nos provea de una herramienta de generación de código para dispositivos programables.

Aquí es donde Ptolemy, un entorno de software de libre difusión desarrollado por la Universidad de Berkeley en EE.UU [4], juega un papel determinante en la concreción de nuestro proyecto.

2. ELECCIÓN DE LA HERRAMIENTA DE DISEÑO.

2.1. Características del Entorno Ptolemy.

Ptolemy es un entorno de programación visual, donde un sistema creado por el

usuario, se define en forma de diagramas de bloques interconectados. Para lo cual la distribución ofrece una gama de librerías conformada por bloques de variada funcionalidad. Además, y como una de sus características más importantes, permite la construcción de estructuras jerárquicas [5]. De esta manera se pueden representar sistemas muy complejos mediante subsistemas anidados. En la Figura 1, se aprecia una *Galaxia* (subsistema) que modela una Red de comunicación simple.

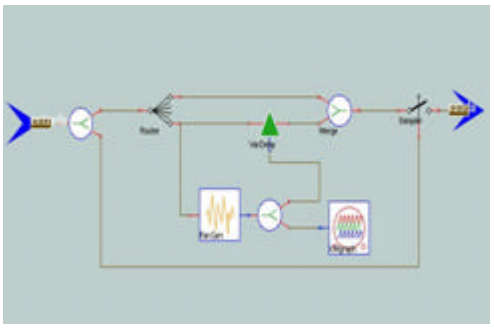

Figura 1. Modelo descripto en el Dominio DE (Eventos Discretos) de una Red de comunicación simple.

Puede pensarse en Ptolemy como un lenguaje de coordinación, aunque no sería preciso definirlo sólo como un lenguaje [1]. Esto se debe a que Ptolemy permite especificar la computación de los módulos involucrados en el diseño en un lenguaje distinto del encargado de gobernar la interacción entre ellos [1]. En varios de los entornos existentes, el código de cada módulo, sólo puede ser especificado en el lenguaje anfitrión (lenguaje madre) con semántica imperativa, como C y C++. En cambio, en Ptolemy, puede consistir en un cuanto de computación especificado con cualquiera de los varios modelos de computación (MoC´s) con que se cuenta.

El Núcleo (Kernel) de Ptolemy está constituido por una familia de definiciones de clases C++ de manera de poder brindar este tipo de soporte [5]. Esta es la clave de su versatilidad.

Ptolemy interpreta las semánticas de ejecución e interacción de sus componentes a través de los llamados *Dominios* (ambientes de diseño específicos). Luego, se asocia a cada Dominio un Modelo de Computación (MoC) que se ocupa de definir formalmente la semántica del comportamiento de sus componentes. Ptolemy posee también una característica muy importante y es que los Dominios pueden entrelazarse jerárquicamente para trabajar juntos, es decir, es posible anidar bloques pertenecientes a distintas semánticas, este mecanismo se denomina Wormhole. [5]

2.2. Descripción del objetivo del proyecto y detalles del algoritmo de Decodificación MP3.

Nuestro proyecto contempla el modelado e implementación de una etapa decodificadora de MPEG-1 Layer-III. La estructura algorítmica que plantea la Norma ISO/IEC 11172-3 para un decodificador MP3 puede representarse esquemáticamente por medio de la Figura 2:

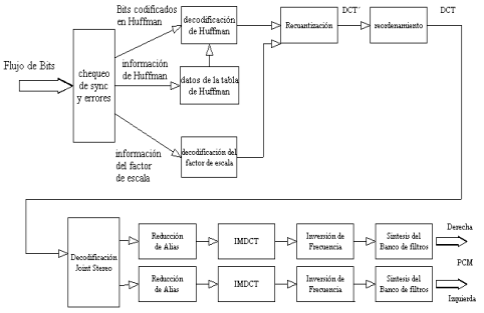

Figura 2. Diagrama en bloques del decodificador de MP3 [10].

Para el Formato MPEG-1 Layer III el Frame (trama) esta formado por: 1152 muestras de audio más la *información de la trama*. Estas muestras de audio se dividen en dos gránulos de 576 muestras cada uno.

En MP3 los Frames no son totalmente independientes unos de otros por causa del posible uso del "bit reservoir". Este es un método de codificación que usa una especie de buffer para completar con datos de audio de otros Frames los que poseen espacio libre. [6]

Figura 3. Estructura de un Frame [7].

El modelo SDF (Synchronous DataFlow) es muy popular en el área de Procesamiento Digital de señales, pero tiene serias dificultades en aplicaciones multimedia como en el caso de un decodificador de MP3 [8]. En estas aplicaciones, los datos se agrupan en flujos de bits y se comprimen para minimizar el costo en transmisión y espacio de almacenamiento.

El formato del Frame (trama) (Figura 3) se compone de *datos de audio* (Información Principal) y *parámetros de codificación* (Encabezado, Chequeo de Errores e Información Secundaria) [7] [9].

El *Encabezado* del Frame posee 32 bits (cuatro bytes) de longitud. Este contiene el "Frame Sync" (formado por los primeros 12 bits, que indican el inicio del Frame y se usan para su sincronización); información sobre qué tipo de codificación se usó (en este caso MPEG –1 Layer III); frecuencia de muestreo; modo de canal y otros.

El *Chequeo de Errores* puede o no estar presente en el Frame. Su longitud es de 16 bits, y si existe, se ubica después del *Encabezado*.

La *Información Secundaria* (Side Information) que indica cómo se realizó la codificación, y por lo tanto, cómo debe realizarse la decodificación, está formada por:

- El *Puntero a la Información Principal* (este puntero indica dónde se localiza el primer byte de los datos de audio)
- La variable "scfsi" (provee información para la selección del factor de escala) .

Por último la *Información Principal* (Main Data) corresponde a los datos de audio.

2.3. Algoritmo de Decodificación.

En el primer bloque del decodificador MP3 (Figura2) se identifican los contenidos del flujo de bits y se pasa la información a los bloques siguientes mediante la decodificación del *Encabezado.* En este bloque se realiza la demultiplexación de muestras de datos y de los parámetros de codificación (chequeo de "Frame sync" y de errores). Los parámetros de codificación contienen los valores que serán utilizados en los bloques siguientes del decodificador. Para cada Frame estos parámetros deben ser repartidos a los distintos bloques sincronizados con las muestras de datos y proveer la información necesaria para realizar la decodificación Huffman, la Recuantización y la IMDCT (Transformada Coseno Discreta Modificada Inversa). La necesidad de distribuir los parámetros a los distintos bloques nos obliga a implementarlos usando en parte *estados globales*, pues éstos deben repartirse a varios bloques y son fijos para cada Frame. Como el Dominio SDF en Ptolemy no permite la utilización de estados globales, ya que traen aparejados efectos secundarios que perjudican el buen funcionamiento del modelo, tendremos que recurrir a otras alternativas para poder utilizar estados globales eliminando dichos efectos [8].

Luego de la demultiplexación de los parámetros del Frame se realiza la decodificación Huffman, este proceso es muy rápido y se hace a través de una tabla de correspondencias. Los datos de Huffman contienen muestras codificadas con longitud variable. Las diferentes tablas de código Huffman se usan dependiendo del *máximo valor cuantizado* y de las *estadísticas locales* de la señal. [9] [10]

El bloque de Recuantización usa los factores de escala para convertir los valores de la decodificación Huffman a la forma de sus valores espectrales. Las muestras recuantizadas deben ser reordenadas según las bandas del factor de escala.

En el bloque de la "decodificación Stereo" el flujo de bits comprimido puede soportar uno o dos canales de audio en uno de cuatro modos posibles (stereo, joint stereo [10], canal doble o canal simple).

La *Reducción de Alias* debe anular los efectos de "aliasing" que aparecen en el *Banco de Filtros Polifásicos* del codificador. La IMDCT aplica la Transformada Inversa Coseno a las muestras de cada sub-banda del filtro polifásico [11]. La salida de la IMDCT esta formada por 18 muestras en el dominio del tiempo por cada uno de los 32 bloques sub-banda [10]. Para compensar inversiones de frecuencia en la *Síntesis del Banco de Filtros Polifásico*, todas las muestras de tiempo impares de todas las subbandas impares son multiplicadas por –1.

La *Síntesis del Banco de Filtros Polifásicos* transforma los 32 bloques subbanda de 18 muestras en el dominio del tiempo en cada gránulo, a 18 bloques de 32 muestras de audio PCM [6] [7].

3. GENERALIDADES DEL DOMINIO SDF.

Las características del Entorno de desarrollo Ptolemy se brindan de forma natural a los requerimientos de nuestro proyecto. Nos referimos a las bondades del Dominio de diseño SDF y al uso de los Dominios de Generación de Código C (CGC) para implementar nuestro modelo en software [5].

El dominio SDF ha sido ampliamente utilizado como base de programación en bloques, para el modelado de sistemas de procesamiento digital de señales. En este Dominio, como en otros modelos Dataflow [12] [13], la especificación del programa se lleva a cabo mediante un "grafo orientado" [1] [14]. En el grafo, los bloques funcionales representan computaciones y son llamados "actors". Las interconexiones entre bloques (arcos) representan colas FIFO que almacenan los valores de los datos ó "tokens". Llamaremos buffer, a la cola FIFO asociada con cada arco.

Una de las restricciones que impone el dominio SDF es que el número de partículas (tokens) producidos y consumidos es fijo y conocido al tiempo de compilación [5] [15].

También, al tiempo de compilación, es posible analizar estáticamente al grafo determinando el orden de ejecución de los nodos y los requerimientos de memoria de los buffers de datos. Estas ventajas que SDF tiene frente a otros Dominios de especificación, se transformarán en factores determinantes a la hora lograr implementaciones en software eficientes para dispositivos programables.

3.1. Métodos de ejecución.

La semántica de un dominio es definida por clases que manejan la ejecución de una especificación. Estas clases pueden invocar a un simulador, generar código, o pueden invocar a un compilador sofisticado.

Un Target (Objetivo) es la clase base de más alto nivel en la ejecución [5]. Un Target realizará su función vía un Scheduler (Planificador). El Scheduler define la semántica operacional de un Dominio controlando el *orden de ejecución* de un modelo funcional. Puesto más sencillo, la función del Planificador es manejar el orden de invocación de cada bloque en una aplicación.

En los dominios de simulación un Scheduler invoca los métodos de ejecución de los bloques del sistema que llevan a cabo la función asociada con el diseño. En los dominios de generación de código, el Scheduler invoca también métodos de ejecución de bloques, pero estos, a diferencia, sintetizan código en algún lenguaje. Esto es, generan código para realizar alguna función en vez de realizar la función en sí misma.

Por otro lado, el Target es responsable de generar el código de conexión entre bloques (sólo si es necesario). Este mecanismo es muy simple e independiente del lenguaje [5]. Para un Dominio de trabajo específico, el usuario puede adjudicar distintos Targets a los disitintos niveles de la jerarquía y así definir mediante qué mecanismos se ejecuta el sistema.

4.GENERACIÓN DE CÓDIGO EFICIENTE PARA APLICACIONES DSP.

A partir de especificar nuestro sistema en SDF y gracias a sus estrictas propiedades formales, el proceso de generación de código ganará en eficiencia y en performance. Esto se debe a que, a diferencia de un lenguaje imperativo, el modelo SDF de base impone restricciones al flujo de control de la especificación, pudiendo esta ventaja ser bien explotada por el compilador [15].

Dado que un grafo SDF impone sólo restricciones de ordenamiento parcial [1] entre nodos, se puede aprovechar las ventajas de ejecución concurrente y determinar el orden de disparos de varias maneras hasta que se cumpla con los objetivos de diseño.

Un grafo SDF apropiadamente construido puede compilarse una vez que se construye un Scheduler S finito [5] [14] que, dispara cada actor al menos una vez, no produce problemas de "deadlock" [5] y no se aprecian cambios en el número de tokens encolados en cada arco (buffer asociado) [16].

Cuando un Scheduler de estas características se repite indefinidamente, se llama a la secuencia de disparos de actores infinita resultante, un Scheduler "válido".

La estrategia de generación de código se denomina "threading". En este proceso, se inserta un bloque de código por cada actor del

cuerpo S del Scheduler, y el resultado de la secuencia de los códigos de bloque es encapsulado dentro de un "loop" infinito generando la implementación en software de la especificación.

El código que producirá cada bloque a través del Target, se obtiene a partir de una librería predefinida optimizada manualmente. Para la descripción de estas librerías de bloques es común la utilización de lenguajes C y Assembler para síntesis de Software, y VHDL o Verilog para síntesis de Hardware.

Estas librerías están definidas de forma tal de poder contar con bloques reusables para poder ser una herramienta útil en lo que respecta al diseño de sistemas de procesamiento de señales.

Un tema que debe tenerse en cuenta en la generación de código es la optimización de la utilización de los recursos de memoria.

El entorno PeaCE desarrollado en la Universidad de Seúl en Corea, introduce novedades con respecto al tratamiento de este tema, además de brindar soporte para el manejo de estados globales.

5. INTRODUCCIÓN AL ENTORNO PEACE EN EL AREA DE CODISEÑO HW/SW .

En el apartado 2.2 mencionamos la existencia de algunas limitaciones que afectan al Dominio SDF en Ptolemy.

Una de ellas es su incapacidad de describir sistemas en los que las ejecuciones de bloques funcionales dependan del valor de los datos presentes a la entrada, como las populares construcciones tipo "for" ó "if then else" disponibles en todos los lenguajes imperativos [3] [16] [17]. Esto último se debe a que SDF está concebido como un lenguaje de especificación totalmente determinado [1], en el que el flujo de partículas es continuo de entrada a salida y todos los bloques se ejecutan sin chequear el valor de los datos entrantes. De acuerdo con la estructura de un decodificador MP3, la utilización de estados globales se hace indispensable en el momento de la especificación. [3] [7] [9] [10] [16] [17].

Por otro lado es una realidad y una necesidad poder especificar un sistema de manera ágil, visual, independiente de la implementación y natural en cuanto a la identificación de bloques y su jerarquía dentro del esquema general, pero por supuesto no a cualquier costo. El problema reside en que el código que se genera a partir de una especificación SDF no es lo suficientemente eficiente en cuanto a los requerimientos de memoria en comparación con una implementación en software hecha manualmente y optimizada.

PeaCE (Ptolemy extension as Codesign Environment) es un entorno de Codiseño Hardware-Software [18] [19] desarrollado por la Universidad de Seúl en Corea basado en el Kernel de Ptolemy, dedicado exclusivamente a resolver la problemática de diseño de sistemas multimedia. Con esto los investigadores de la Universidad de Seúl han conseguido extender de manera muy satisfactoria la semántica de los tres Dominios por excelencia utilizados en el diseño de sistemas digitales: SDF, DE (Eventos Discretos [1]) y FSM (Máquinas de estado finitas). A pesar de que PeaCE es un entorno todavía en desarrollo, creemos que las prácticas innovaciones que ofrece la hacen especial para la concreción de nuestro proyecto.

PeaCE extiende la semántica SDF y FSM para especificar tareas de procesamiento de señales y tareas de control respectivamente, y una extensión del modelo DE ayuda a definir una estructura de transfondo (Codesign Backplane) que maneja toda la interacción entre tareas[backplane].

Si bien PeaCE aborda con éxito la introducción del manejo de estados globales y la mejora en los requerimientos de memoria de datos y tamaño de código con el novedoso dominio SPDF, también introduce otra extensión del dominio SDF en Ptolemy, llamado FRDF (Fractional Rate Dataflow). Este es todavía más agresivo en el ahorro de memoria de datos aplicando una técnica de optimización por compartición de buffers [16]. Particularmente estamos interesados en las novedades del dominio SPDF (Synchronous Piggybacked Dataflow) de la distribución, de acuerdo con los alcances de la primera etapa de nuestro proyecto, y se dejará para posterior análisis la aplicación del dominio FRDF una vez concluida la misma.

6. EL DOMINIO SPDF (SYNCHRONOUS PIGGYBACKED DATAFLOW MODEL).

El dominio SPDF logra embeber en el Dominio SDF el manejo de estados globales sin que se hagan presentes efectos indeseados o adversos. Así, el equipo de trabajo de la Universidad de Seúl en Corea logra incluir dentro de la semántica Dataflow el manejo de estructuras de dependencia de control entre bloques funcionales. Sin embargo, la modificación introducida permite dependencia de control limitada. Esta limitación viene dada de manera de evitar dañar el poder de modelización y el paralelismo. Por lo que el único punto negativo es el incremento en la complejidad del Scheduler [17] [8].

La idea principal de la solución propuesta dentro del entorno PeaCE, es generar una tabla global (GST, Global state table) de acceso limitado. Con las palabras "acceso limitado" quiere decirse que existirá sólo un "Escritor" de estados globales aunque no hay restricciones para el número de "Lectores", que pueden ser numerosos. Una entrada de una GST es una n-upla {nombre_estado; un arreglo de valores de estado}. El primer componente de esta n-upla es el nombre del estado global al que hacen referencia los bloques. Mientras que el arreglo de valores de estado, es un buffer circular que mantiene los valores de estado encolados. Como se verá, el tamaño del arreglo viene determinado por la organización de disparos del grafo.

En resumen, para manejar un parámetro de un bloque como un estado global, se define una GST que mantiene los valores de los parámetros que puedan ser cambiados desde el exterior a través de un pedido SU

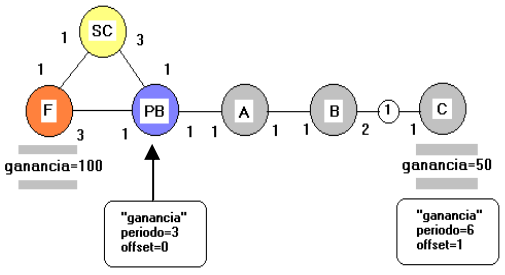

Figura 4. Esquema de un grafo SPDF.

En la Figura 4, se aprecia la introducción de dos nuevos protagonistas en escena que serán los responsables de la extensión de la semántica. Uno de ellos es el bloque PB (Piggybacking block [8] [17] [18]) el cual posee dos tipos de puertos distintos en su entrada. Uno es el puerto común de datos (el que utiliza el bloque Fuente "F"), y el otro llamado "State Port" es un nuevo tipo introducido. El otro bloque llamado SC (State Convert block) es el segundo protagonista cuya tarea es la conversión de un puerto de datos en un puerto de estado. En la figura se muestra como 3 muestras de datos comparte un pedido SU.

El bloque SC consume una muestra con el valor de estado "ganancia" y produce 3 pedidos SU al puerto de estado del bloque PB. Cuando el bloque PB acepta un valor de estado y una muestra de datos de los bloques precedentes, actualiza la GST con el valor de estado recibido y produce un par {dato;puntero}.

Se logra así entregar implícitamente un pedido de "actualización de estados" (SU, System Update) al nodo que debe hacer uso de un estado global. El nodo actualizará el valor de su estado local con el valor apuntado a la entrada de la GST antes de procesar los datos que arriban a su entrada [8] [17]. El bloque C es el bloque destino que examina la información de estado añadida a cada muestra de datos cada vez que se ejecuta. Cuando este bloque reconoce la necesidad de una actualización, copia el valor del estado global "ganancia" en su estado local.

La utilización de punteros, en vez de trabajar directamente con los valores de estado, no resuelve el problema de la copia redundante de pedidos de SU [17] (estos deben atravesar todos los bloques intermedios, aunque no hagan uso de estados globales, hasta arribar al nodo destino).

Entonces, se aprovecha una ventaja de la sintaxis de un grafo SDF [8] [17]. En realidad lo que se hace es definir dos valores: {período, offset} como propiedades del bloque PB. El período es el período de repetición de actualización de estados (SU) computado en términos de ejecuciones de nodos. El offset es la posición inicial de las muestras de datos a los cuales se les acoplan los pedidos de SU.

Estos parámetros son decididos al tiempo de compilación y permanecen constantes durante la ejecución. Entonces, a través del

análisis del camino de datos (cambios en las frecuencias de ejecución y introducción de retrasos entre bloques) entre el bloque PB y el bloque C (se hace en tiempo de compilación), pueden calcularse los valores de período y offset del bloque destino. Propagando este vector {período;offset} hasta el bloque C se consigue conocer el "momento" justo de la actualización del estado local. Pero además recordando que el Scheduler se construye en tiempo de compilación y, por tanto, se conoce el número de iteraciones activas: puede determinarse óptimamente el tamaño del arreglo de una entrada a la GST. Esto conlleva a la posibilidad de construir la estructura de datos para los estados globales al tiempo de compilación [8] [17].

La tarea del bloque PB es simplemente primero propagar su par, manualmente determinado por el usuario, {período, offset} y calcular (se hace automáticamente) los mismos valores para el bloque destino. Segundo, construir la entrada respectiva a la GST y una vez que recibe el valor de estado por el puerto especial actualizar el valor del estado global correspondiente. Para esto, el bloque PB, transfiere datos desde su entrada hasta su salida y va incrementando un contador interno. Cuando la cuenta interna es igual al valor de offset especificado recibe el valor de estado y procede a la actualización de la GST.

Finalmente, como tercer y más importante paso, incorpora un segmento de código en la parte superior (código de SU) previa al código propio del bloque C. El mismo, podrá actualizar sus estados locales si ahora su contador interno se iguala al valor de offset que le corresponde. Esto permite síntesis eficiente de código en términos de requerimientos de memoria, eliminando la copia redundante de pedidos SU en que incurrían otras técnicas que intentaban, sin éxito, incorporar el uso de estados globales en grafos Dataflow[16].

7. DETECTOR DE CAMPOS DE BITS UTILIZANDO ESTADOS GLOBALES.

El sistema del ejemplo (Figura 5) consiste en un grafo perteneciente al Dominio CGC (Generación de Código C a partir de un grafo SPDF). Mediante la actualización sincrónica del estado global de nombre "campo" (Period:2 Offset:0), se logra subdividir periódicamente una trama de bits en 4 campos de 2, 4, 6 y 4 bits de longitud respectivamente para su posterior análisis y procesamiento.

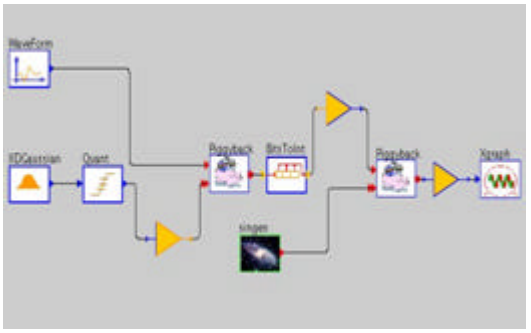

Figura 5. Diagrama en bloques del sistema.

De esta forma se controla, a través de la utilización de otro estado global de nombre "ganancia", la amplitud de la senoidal generada por la Galaxia "singen" (100 muestras por ciclo).

La Fuente de señal "Waveform" se encarga de decidir la longitud de cada campo de bits mediante la edición de su parámetro o estado local "value". En él, el usuario especifica por medio del arreglo contenido en "value", los valores del estado global "campo" que, el primer bloque "Piggyback", irá acoplando a cada muestra cada vez que se detecte un pedido SU.

Colocando en serie el bloque fuente de señal "IIDGaussian" con un cuantizador (bloque "Quant") de dos niveles (0:bajo; 1:alto), se generan los bits aleatoriamente regidos por una Distribución Gaussiana de media 0,5 y varianza 0,25.

Esta es la entrada de señal del primer bloque "Piggyback". El valor inicial del estado global "campo" toma el valor 2.

Luego en cada SU, el parámetro local "nBits" del bloque "BitsToInt" cambia de acuerdo con la señal determinada por el bloque "Waveform".

El parámetro "nBits" especifica la cantidad de bits que consumirá el bloque "BitsToInt" para producir un valor entero. Este es el parámetro que se hace corresponder con el valor del estado global "campo". El entero producido por este último bloque servirá como fuente de los valores que tomará el segundo estado global "ganancia" (Period:100; Offset:0). Luego, se hace corresponder el parámetro "gain" del bloque "Gain" (se encuentra entre el segundo bloque

"Piggyback" y el bloque graficador "XGraph") con el estado global "ganancia".

Se observa en la Figura 6, luego de 25 iteraciones, como varía la amplitud de la senoidal de acuerdo con la información extraída de los campos de bits analizados y con la frecuencia de cambio determinada por el período de pedido de actualización de estados al que debe responder el segundo bloque "Piggyback".

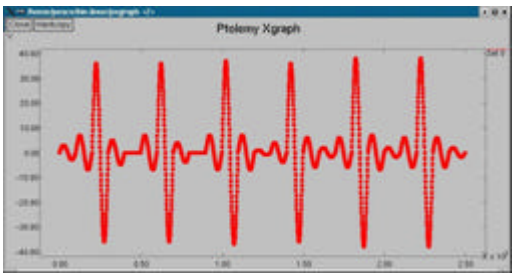

Figura 6. Resultado de la simulación del sistema.

Todos los bloques del grafo pertenecen al Dominio CGC (inclusive los que conforman la Galaxia "singen") y el Target elegido ha sido "Default-CGC". Por último en la Figura 7 se aprecia una parte del código C generado a partir de la especificación, visualizado con la herramienta "Emacs" que provee Linux.

```
/* User:     peace
   Date:     Thu Nov 27 19:25:55 2003
   Target:   default-CGC
   Universe: Detector_bits */

/* Define macro for prototyping functions on ANSI & non-ANSI compilers */
#ifndef ARGS
#if defined(__STDC__) || defined(__cplusplus)
#define ARGS(args) args
#else
#define ARGS(args) ()
#endif
#endif

#include <stdlib.h>
#include <math.h>
#include <stdio.h>

/* Define constants TRUE and FALSE for portability */
#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif

/* Define a complex data type if one has not been defined */
#if !defined(COMPLEX_DATA)
#define COMPLEX_DATA 1
typedef struct complex_data { double real; double imag; } Complex;
```

Figura 7. Código C generado a partir de la especificación del sistema en el Dominio CGC utilizando el Target "Default-CGC".

8. CONCLUSIONES.

Durante el desarrollo de este trabajo, hemos estudiado el estándar MP3 y así identificamos las necesidades computacionales que hacen falta para decodificar un archivo comprimido mediante este formato.

También estudiamos exhaustivamente el dominio SDF de Ptolemy y encontramos que por sí sólo no cumplía las condiciones necesarias para resolver nuestro problema. Por lo que suplimos la carencia de estados globales de SDF, mediante el uso del dominio SPDF que implementa estado globales.

Mediante el ejemplo del "detector de campos de bits", demostramos que se pueden describir sistemas que necesiten de ejecución dinámica de bloques y pasaje de parámetros globales entre los mismos.

Aprovechando las nuevas características de la semántica SPDF, pudimos hacer uso de estados globales de forma ágil y eficiente en cuanto a la generación de código, conservando las buenas cualidades de la semántica SDF original de Ptolemy (posibilidad de análisis estático de los grafos).

Este primer uso del dominio SPDF avanza en el camino de la programación específica de los diversos bloques necesarios para decodificación de MP3, siguiendo el Estándar Internacional ISO/IEC 11172-3.

9. REFERENCIAS.

[1] W.Chang, S.Ha, and E.A.Lee, "Heterogeneous Simulation - Mixing Discrete-Event Models with Dataflow," RASSP Special Issue of VLSI Signal Processing January 1997

[2] Jose Luis Pino, Soonhoi Ha, Edward A. Lee y Joseph T. Buck, "Software Syntesis for DSP Using Ptolemy", Journal of VLSI Signal Procesing, 9, 7-21 (1995)

[3] Kim, D.;Ha. S., "System level specification for multimedia applications," 6th Conference of Asia Pacific CHip Design Languages, Fukuoka, Japan, Oct. 6-8 1999

[4] UC Berkeley, Department of EECS, "The Ptolemy Project", *http://ptolemy.eecs.berkeley.edu*

[5] User´s Manual: *Ptolemy User´s Manual*, Version 0.7, Linux, University of California at Berkeley, College of Engineering, Department of Electrical Engineering and Computer Sc iences, 1997.

[6] Hung- Chih Lai ,“Real-Time Implementation of MPEG-1 Layer III Audio Decoder on DSP Chip”,National Chiao-Tung University, Deparment of Electrical and Control Engineering, June 2001, Taiwan, China.

[7] Davis Pan. “A Tutorial on MPEG/Audio Compression”, *IEEE Multimedia Journal*, Summer 1995 issue.

[8] Chanik Park, Jaewoong Chung and Soonhoi Ha. "Extended Synchrous Dataflow for Efficient DSP System Prototyping”,*Poc. IEEE International Workshop on Rapid System Prototyping*, Florida, USA, June, 1999.

[9] ISO/IEC International Standard IS 11172-3, “Information Technology-Coding of Moving Pictures and associated Audio for digital storage at up to about 1.5 Mbit/s-Part 3:Audio”.

[10] Krister Lagerstrom, “Design and Implementation of an MPEG-1 Layer III Audio Decoder”, Chalmers University of Technology, Department of Computer Engineering Gothenburg, Sweden 2001.

[11] B. Porat, A Course in Digital Signal Processing. New York: John Wiley & Sons, Inc., 1997.

[12] Edward A. Lee, Alberto Sangiovanni-Vincentelli, “Comparing Models of Computation”, Proceedings of ICCAD, San Jose, CA, Nov. 10-14, 1996.

[13] Marleen Adì, Rudy Lanwereing , J.A.Peperstraete, “Data Memory Minimization for Sinchronous Dataflow Graphs Emulated on DSP-FPGA Targets” Katholieve Universiteit Leuven, ESAT Department, ACCA Laboratory, Kard, M

[14] Joseph Buck , Edward Lee, “ The Token Flow Model”, presented at Data Flw Workshop, Hamilton Island, Australia, Mayo 1992. ercierlaan 94, B-3001 Heverlee, Belgium.

[15] Pro veen K. Murthy, Shuvre S. Bhattacharyye y Edward A. Lee,”Joint Minimization of Code and Data for Synchronous Dataflow Programs” “Journal of Formal Methods in System Design”, vol. 11 , Nº1, Julio 1997

[16] Hyunok Oh and Soonhoi Ha, "Memory-optimized Software Synthesis from Dataflow Program Graphs with Large Size Data Samples," EURASIP Journal on Applied Signal Processing Vol. 2003 pp 514-529 May 2003

[17] Chanik Park, Jaewoong Chung and Soonhoi Ha, "Efficient Dataflow Representation of MPEG-1 Audio(Layer III)Decoder Algorithm with Controlled Global States," IEEE Workshop on Signal Processing Systems(SiPS): Design and Implementation, Taiwan, ROC October 1999

[18] User´s Manual: *PeaCE User´s Manual*, Version 1.0, Linux, CAP Laboratory of Seoul National Univ ersity and the Pringet corporation, may 28, 2003.

[19]”PeaCE:CodesignEnvironment” *http://peace.snu.ac.kr*

[20]J.Buck, S.Ha, E.A.Lee, and D.G.Messerschmitt, "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems," International Journal of Computer Simulation Vol. 4 pp 155-182 April 1994