Especificación de una plataforma de codiseño basada en dsp/fgpa para la implementación de aplicaciones especificadas en simulink

Definition of a dsp/fpga co-design platform to implement applications described in simulink

Jorge Iván Marín*, Alexander López*, Jhon James Quintero*

Recibido: Febrero 2 de 2009
Aceptado: Junio 1 de 2009
Correspondencia: Programa de Ingenierìa Electrónica, Universidad del Quindío, Avenida Bolivar Calle 12 norte Armenia Quindío. Email: gdsproc@uniquindio.edu.co

Resumen

Se presenta la especificación del diseño de una arquitectura para el codiseño de aplicaciones de tratamiento digital de señales que parte de la especificación del sistema modelado en Simulink de Matlab. La arquitectura está compuesta por un DSP de punto fijo y una FPGA. Se construyó un traductor o parser que traduce un modelo de Simulink a sus funciones equivalentes en lenguaje C y entidades en VHDL, que son posteriormente compiladas y descargadas al DSP y FPGA, respectivamente. Adicionalmente, fue necesario construir un sistema o kernel de planificación de la ejecución de las tareas y la comunicación entre entidades hardware/software. Se detalla el modelo del sistema, y los diseños del kernel, hardware y parser.
Palabras claves: *Codiseño, Procesador Digital de Señales, DSP, Arreglos Programables en Campo, FPGA*

Abstract

In this paper, architecture for co-design of signal processing applications from a model based on Simulink-Matlab is presented. This architecture is composed by a fixed-point Digital Signal Processor (DSP) and a Field Programmable Gate Array (FPGA). A parser translates the model specified on Simulink into C language routines or VHDL entities, which are compiled and downloaded to the DSP and the FPGA, respectively. Also, a kernel inside the DSP is responsible for task scheduling and communication of the hardware/software entities. The model of this system and the design of its kernel, hardware and parser are described.
Keywords: *Co-design, Digital Signal Processor, DSP, Field Programmable Gate Array, FPGA*

INTRODUCCIÓN

El desarrollo de sistemas empleando tecnologías que combinan procesadores convencionales o DSP con hardware configurable como una FPGA ha adquirido, en los últimos años, una relativa importancia para el desarrollo de aplicaciones de tratamiento digital de la señal (1), principalmente en aquellos sistemas que implican un alto costo computacional como son los sistemas de procesamiento de video o la integración de señales de voz e imágenes con telecomunicaciones (2). La metodología de más amplia difusión para el diseño de estos sistemas es el codiseño. Codiseño se refiere al diseño simultáneo del hardware y el software, y es una estrategia que le permite a los desarrolladores evaluar diferentes alternativas y estructuras antes de obtener el prototipo final. Esta estrategia demanda herramientas que soporten la representación unificada del proceso hardware-software, la simulación heterogenea a diferentes niveles y la síntesis del hardware-software (3). Adicionalmente, dos tareas claves para la automatización del procedimiento de codiseño es la partición de los bloques funcionales y la planificación de la ejecución de dichos módulos (2).

Por otra parte, es indudable que Matlab es la herramienta de más amplia difusión para la exploración de alternativas de

* Programa de Ingeniería Electrónica, Grupo GDSPROC, Universidad del Quindío. e-mail: gdsproc@uniquindio.edu.co.

desarrollo de una aplicación de procesamiento digital de señales, y su entorno de simulación Simulink, se constituye en una de las formas más apropiadas de representar y simular un sistema complejo. Por consiguiente, para utilizar Matlab como una herramienta de codiseño completa, sería necesario adicionarle componentes que permitan el particionamiento de los bloques funcionales en implementaciones hardware o software, y una arquitectura de implementación, en la cual se lleve a cabo el proceso de planificación de la ejecución.

Este trabajo está orientado hacia el diseño de una herramienta de codiseño que permita partir de la especificación de un modelo de sistema dado en Simulink. Para conseguir este propósito este proyecto está divido en dos fases. La primera consiste en la especificación y diseño de la arquitectura de implementación basada en tecnología DSP y FPGA. Se seleccionaron estas dos tecnologías porque favorecen la construcción de sistemas embebidos. La segunda fase se trata del estudio y definición de algoritmos de partionamiento automático.

En este documento solamente se detalla la primera fase del proyecto, pues el particionamiento requiere del conocimiento a priori del comportamiento de la implementación de cada uno de los diferentes bloques funcionales y la posterior aplicación de algún algoritmo (3), (4) que seleccione la mejor implementación posible.

Este documento está organizado como sigue: en la sección 2 se describe el modelo general de la arquitectura propuesta, haciendo énfasis en su concepción y aspectos considerados para su diseño, además se detalla el mecanismo propuesto para la planificación de la ejecución de los bloques funcionales; en la sección 3 se indica la forma como se diseñó el traductor o parser encargado de pasar del esquema especificado en Simulink a una versión implementable en la arquitectura; a continuación, en la sección 4 se describen los aspectos de implementación referentes a la comunicación entre el DSP y la FPGA; y finalmente, en la sección 5 se presentan las conclusiones del trabajo.

MODELO DEL SISTEMA

La arquitectura de implementación para el codiseño de aplicaciones que se propone en este trabajo, está basada en un procesador digital de señales (DSP) de punto fijo de la Texas Instruments TMS320C5416, y una FPGA de Xilinx, Spartan III. La FPGA se conecta al DSP a través de los buses de expansión de memoria y periféricos que vienen embebidos en la tarjeta de desarrollo DSK para el DSP. Bajo esta forma de conexión, el DSP ve a la FPGA como un dispositivo mapeado en su memoria. Dicho mapa de memoria se explica en la sección 4.

Dado que la aplicación se representa en el lenguaje de descripción del Simulink, se hace necesario realizar una traducción de los bloques funcionales de Simulink a sus funciones y entidades homólogas, que se ejecutarán en software y/o hardware.

Para facilitar la traducción y el intercambio de información entre los bloques software y hardware, y permitir la planificación de los procesos, se propone en este trabajo modelar cada uno de los bloques funcionales de Simulink como una tarea. En el contexto de los sistemas operativos, una tarea puede tener diferentes estados: lista, en ejecución, bloqueada, etc.; en un sistema con un único procesador se puede realizar la conmutación entre tareas (multiprocesamiento); y éstas se pueden comunicar entre sí por diferentes métodos, siendo el más común, los semáforos (5), (6). Este tipo de abstracción permite modelar cualquier bloque funcional de Simulink independiente de sí es una implementación que se ejecutará en software o en hardware. Por otra parte, la selección de la tarea a ejecutar en un sistema operativo depende de la entidad conocida como planificador, lo que indica que el diseño del mecanismo de planificación de la ejecución de los bloques funcionales se limita al diseño de esta entidad.

Por las particularidades de la arquitectura a desarrollar, el modelo de las tareas que se proponen se diferencian de las tareas de un sistema operativo ordinario. Las características de dichas tareas son:

1) Una tarea puede tener los siguientes estados: dormida, en espera, lista para procesar, procesando y bloqueada, los cuales serán descritos más adelante.

2) Mientras una tarea de software se encuentre en ejecución (estado procesando) no es posible quitarle el control hasta tanto ésta no haya terminado. Es decir, el planificador no puede ser apropiativo. Esto le brinda estabilidad al sistema, facilita el proceso de planificación y permite emplear para el procesamiento formas altamente optimizadas.

3) Todas las tareas tienen la misma estructura funcional: inicialmente esperan la presencia de un conjunto de señales (evento wait), realizan el procesamiento, y finalmente producen una señal de respuesta (evento signal) hacia otras tareas.

4) Solamente se consideran dos tipos de prioridades para las tareas, donde las tareas de más alta prioridad son aquellas

ligadas a la adquisición y reproducción de datos, es decir, los equivalentes a los bloques Simulink del tipo source y sink.

Las características anteriores simplifican el proceso de traducción de un bloque Simulink. Por ejemplo, suponga que el bloque a sintetizar es un filtro IIR. En software, el código generado por la herramienta de síntesis debe ser: la espera de un conjunto de señales software, el llamado a una función predefinida de una lista de posibles implementaciones para el bloque, y finalmente debe generar las señales software que indican el fin de ejecución de dicho bloque. Para una síntesis en hardware, la implementación del filtro IIR sería una máquina de estados, cuyo estado inicial es la espera de una señal hardware proveniente del planificador alojado en el DSP –esta señal se genera a partir de alguna señal software-, y el estado final sería la generación de una señal hardware de finalización, la cual es capturada por alguna rutina de servicio de interrupción al interior del DSP, quien finalmente se encarga de generar las señales software produciría el bloque si fuera implementado en software.

ESTADO DE UNA TAREA

Como se indicó anteriormente, cada uno de los bloques funcionales de Simulink se interpretan como tareas en el modelo del sistema. Se describen a continuación los diferentes estados que puede tener una tarea:

Dormida. Una tarea se considera dormida cuando nunca se ha ejecutado o ha acabado de generar una señal hacia otras tareas y no se conoce con certeza cual será su estado siguiente. En el ámbito de las máquinas de estado, sería el estado inicial de la tarea y/o el estado al cual llega la máquina después de terminar su ejecución.

En espera. Cuando una tarea está aguardando por una señal proveniente de otra tarea, se dice que la tarea entró en estado de espera. Esta condición es típica del inicio de la ejecución de un bloque funcional, siguiendo el modelo del sistema descrito con anterioridad.

Lista para procesar. Una tarea entra en este estado cuando ha recibido todas las señales por las cuales estaba esperando, y se encuentra a la espera de que el planificador la ponga en el estado procesando.

Procesando. Se dice que una tarea está procesando cuando ha recibido todas las señales necesarias y ésta ha sido invocada por el planificador, lo que indica que se encuentra realizando el procesamiento. En un sistema con único procesador, por las características no apropiativas de las tareas y para favorecer la ejecución en paralelo, se debe considerar para el diseño del planificador el hecho que solamente una tarea software estará en el estado procesando, por el contrario, pueden existir varias tareas hardware en dicho estado.

Bloqueada. Una tarea se penaliza con el estado bloqueado cuando debe esperar a que todas las tareas que dependen de sus señales terminen su ejecución. La planificación adecuada de este estado es crucial para el desempeño adecuado del sistema, pues es muy probable que existan tareas hardware/software, que por la concurrencia del sistema, generen señales de salida a una velocidad más rápida de lo que las pueden procesar las tareas dependientes.

En la Fig. 1 se muestra el diagrama de estados de la ejecución de una tarea. Nótese que para garantizar la planificación adecuada, una tarea entra en el estado bloqueada cuando trata de enviar una señal a alguna tarea que no está en el estado de “espera” (por ejemplo, en aquellas situaciones donde la tarea destino está en los estados “procesando” o “bloqueada”), y sólo sale de este estado cuando todas las tareas dependientes esperan por una señal de esta tarea.

Por otra parte, esta asignación de estados y su interdependencia, hace que el diseño del planificador sea muy simple, pues éste se debe limitar a hacer un llamado a aquellas tareas que se encuentren en el estado “dormida” o “lista para procesar”, y jerarquizar su ejecución según la prioridad establecida para las tareas. El planificador debe omitir entonces aquellas tareas que se encuentran en los estados “en espera”, “procesando” y bloqueada”, ya éstas salen de dichos estados por medio de los eventos wait y signal que serán descritos a continuación.

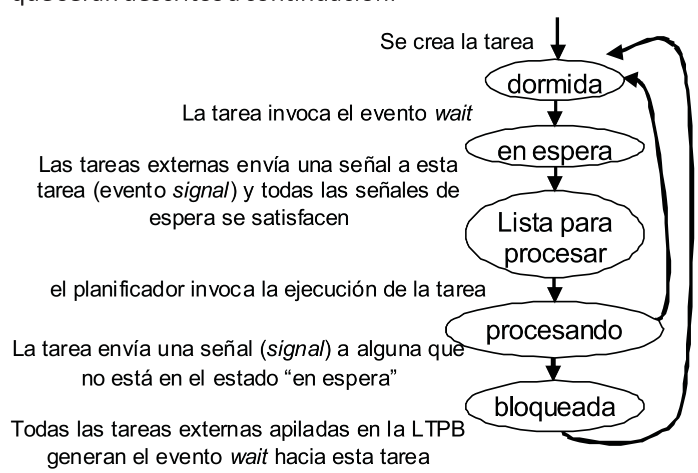

Figura 1: Estado de una tarea

COMUNICACIÓN ENTRE TAREAS

Uno de los aspectos importantes para el adecuado funcionamiento del planificador es la concepción de la comunicación entre procesos. En nuestro caso se ha mostrado que el principio básico para establecer el orden de ejecución de las tareas es su estado, el cual cambia, según se ilustra en la Fig. 1, por medio de la generación de los eventos wait y signal. El nombre de estos eventos se deriva del concepto de semáforo empleado en los sistemas operativos (5). Al igual que los semáforos para controlar el tráfico, los semáforos en software son banderas que le indican a los procesos el momento en que pueden reanudar su ejecución. En nuestra arquitectura, el concepto de semáforo debe incluir características adicionales que los adecuen a los requerimientos del diagrama de estados de la ejecución de una tarea (Fig. 1) y a las características propias de las tareas.

En la arquitectura que se propone, cada tarea tiene asociado un semáforo para indicarle cuando pasar del estado “en espera” al estado “lista para procesar”, y una lista de las tareas que han provocado llevarla al estado “bloqueada” (LTPB).

Cada semáforo es un arreglo de señales pendientes. Para representar las señales pendientes se hace uso de una estructura, denominada ESP, que tiene como miembros: el identificador de la tarea de la cual proviene la señal –tarea padre-, el número de la salida de la tarea padre y un contador del número de datos.

Como en el proceso de síntesis, es posible seleccionar para un mismo bloque funcional de Simulink diferentes implementaciones en hardware o software, que hacen uso de procesamiento orientado a muestras o a bloque, el envío de señales entre tareas debe tener en cuenta la cantidad de datos que entrega la salida de un bloque funcional a otro. Esto justifica el porque del contador del número de datos. Para ilustrar mejor este problema y mostrar la necesidad del contador, considere que se va a sintetizar una aplicación que tiene conectados en serie un filtro IIR y una FFT. La FFT se calcula una vez han ingresado N datos que le son entregados por el filtro, pero el filtro IIR está implementado con una estructura orientada a muestras, de tal forma, que la ejecución de la tarea asociada al filtro IIR debe ser invocada N veces por el planificador antes de poder ejecutar la tarea asociada a la FFT. Lo anterior significa que la tarea FFT debe permanecer en el estado “en espera”, hasta tanto la tarea IIR se ejecute N veces. Como se verá a continuación, estos problemas se solucionan con una modificación de los eventos signal y wait:

wait. Al ser invocado por una tarea A lleva a cabo la siguiente secuencia de acciones: a) por cada señal pendiente incrementa el contador de datos en la cantidad requerida por la tarea A. Sí este contador es mayor o igual cero, y la tarea B, de la cual se espera la señal, está en el estado “bloqueada”, elimina la referencia A de la LTPB de la tarea B. Cuando la LTPB de la tarea B está vacía, su estado pasa a “dormida” (Fig. 1); b) la tarea A, a la cual pertenece el semáforo, pasa al estado “en espera” sí al menos un contador de datos es mayor a cero, en caso contrario pasa al estado “Lista para procesar”.

signal. Al ser invocado este evento por la tarea A se genera la siguiente secuencia de acciones: a) por cada señal que se envía se decrementa el contador de datos de la tarea destino (tarea B) en la cantidad proporcionada por la tarea A; b) sí todos los contadores del semáforo de tarea B son menores o iguales a cero, pasa la tarea B al estado “Lista para procesar”; c) sí al menos uno de los contadores que modificó este evento se hizo menor o igual a cero o la tarea B no esta en el estado “en espera”, pone la tarea A en estado “bloqueada”, insertando en la LTPB las tareas que produjeron dicha situación; en caso contrario pasa la tarea A al estado “dormida”.

Una característica adicional de los eventos wait y signal, es que éstos se encargan internamente de los movimientos de datos, que sean necesarios, entre el buffer de salida de una tarea y el buffer de entrada de otra. Para ello, la estructura que define las señales pendientes (ESP) en el semáforo contiene también la referencia a la función encargada del movimiento de datos según el caso.

DISEÑO DEL PARSER

Como se indicó anteriormente, el procedimiento de síntesis de los bloques funcionales consiste en la traducción de diagramas Simulink a sus correspondientes implementaciones en lenguaje C o VHDL. Para lo cual se desarrolló una librería de los bloques fundamentales para procesamiento digital de señales, conformada por una serie de plantillas previamente probadas en cada uno de los lenguajes objetivo. Adicionalmente se recopiló información estadística referente al consumo de recursos y tiempos de ejecución que son necesarios para los algoritmos de particionamiento automático que se usarán en la segunda fase del proyecto.

Para el proceso de traducción de cada bloque Simulink, se hace necesario indicar el tipo de implementación que se desea del mismo, por ejemplo un filtro FIR puede estar implementado en estructura directa o estructura simétrica, en punto fijo o punto flotante, orientado a bloques o muestras, implementado en hardware o en software, entre otras características que dependen del bloque en particular y el particionamiento. De esta forma, el parser se encarga de escoger de la librería de funciones, la implementación más adecuada para cada bloque y genera automáticamente el código, en lenguaje C, para la construcción de los semáforos de cada una de las tareas y el llamado a los eventos wait y signal, los cuales son los encargados de representar las conexiones entre los diferentes bloques funcionales.

En el caso de la traducción de los bloques hardware, además de emplear las plantillas para representar el funcionamiento de los bloques Simulink, se genera el código C que realiza la comunicación entre las tareas hardware y software, así como el código en VHDL del controlador que se encarga de la conexión entre las tareas hardware, alojadas en la FPGA, y el bus de expansión del DSP.

IMPLEMENTACIÓN DEL SISTEMA

En la arquitectura propuesta, el kernel del sistema lo conforman el planificador y las funciones de comunicación entre tareas, signal y wait, las cuales se implementaron en software, siguiendo el protocolo y la especificación descrita en la sección II. Estas rutinas fueron escritas en lenguaje C y hacen uso de las funciones de soporte disponibles para la tarjeta de desarrollo DSK en la cual se encuentra el procesador digital de señales de punto fijo TMS320C5416.

Como se indicó anteriormente, la FPGA se encuentra conectada a los buses de expansión de memoria y periféricos de la DSK, de tal forma que desde el punto de vista de programación del DSP, la FPGA se ve como un conjunto de registros ubicados en zonas específicas de memoria. En la Tabla 1 se presenta el mapa de memoria de los registros con los que se controla las entidades implementadas en la FPGA.

Tabla 1: Mapa de Memoria para los Registros de la FPGA

Base+0	INI_DIS	Iniciar dispositivo
Base+1	FIN_DIS	Bandera de estado del dispositivo que ha finalizado.
Base+2	NUM_DIS_IN	Identificador del dispositivo de entrada que se accederá.
Base+3	DIR_DIS_IN	Dirección de la memoria en el dispositivo de entrada.
Base+4	DATA_ DIS_IN	Registro de datos entrada.
Base+5	NUM_DIS_OUT	Identificador del dispositivo de salida que se accederá.
Base+6	DIR_DIS_OUT	Dirección de la memoria en el dispositivo de salida.
Base+7	DATA_ DIS_OUT	Registro de datos salida.

Para la implementación de las tareas hardware es necesario generar código en lenguaje C y VHDL. El código en C es necesario para adaptar los bloques implementados en hardware con el protocolo establecido para la comunicación entre tareas y su respectiva ejecución por parte del planificador. Al igual que las tareas software, se usan los mismos semáforos para iniciar una tarea hardware. Para poner una tarea hardware en el estado “procesando” se hace necesario enviar inicialmente los datos de entrada al bloque hardware. Esto se hace a través de registros dedicados de la Tabla 1, NUM_DIS_IN, DIR_DIS_IN, DATA_DIS_IN. Por ejemplo, para enviar un arreglo de N datos a un dispositivo hardware, se escribe inicialmente en NUM_DIS_IN, el número o identificador de la entidad, luego en DIR_DIS_IN, se indica la dirección inicial del buffer interno de memoria donde se copiarán los datos, y posteriormente se hacen N escrituras sucesivas en el registro DATA_DIS_IN. En la FPGA, un controlador se encarga de direccionar los datos desde estos registros hacia los respectivos buffers de memoria (Fig. 2).

Finalmente, para comenzar la ejecución de un bloque, el programa en el DSP debe escribir en el registro INI_DIS el identificador asignado a la entidad. Al interior de la FPGA, el controlador genera la señal de inicialización de la máquina de estados correspondiente.

Cuando la tarea hardware termina, pone activa una bandera de finalización al interior del controlador, quien se encarga de enviar una solicitud de interrupción a la DSK. En la rutina de servicio de interrupción (ISR), se lee el registro FIN_DIS cuyo valor es el identificador de la tarea hardware que ha concluido. Repetidas lecturas a este registro entregan los identificadores de los procesos que han concluido, si dicho registro devuelve el valor FFFFh, indica que no hay más tareas pendientes. Con dicho identificador, la ISR hace uso de los registros NUM_DIS_OUT, DIR_DIS_OUT, DATA_DIS_OUT para leer los buffers de salida del dispositivo que ha concluido, y envía los eventos signal para notificarle esta situación a las restantes tareas.

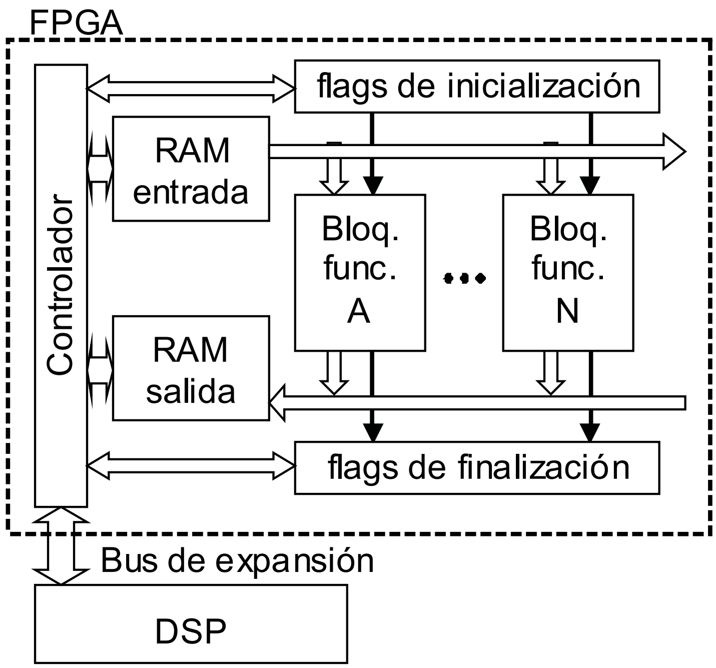

Figura 2: Diagrama de bloques de la interconexión entre el controlador y las tareas hardware

CONCLUSIONES

Se construyó una arquitectura el codiseño de aplicaciones de tratamiento digital de señales que parte de la especificación de un modelo funcional en Simulink, lo que permite simplificar y estandarizar los procesos de diseño de aplicaciones de tratamiento de señales. Para lograrlo se construyó una base de datos de diferentes implementaciones en lenguaje C y VHDL para un mismo tipo de bloque funcional, a partir de las cuales el parser selecciona la implementación según el particionamiento manual del usuario. Se cuenta también con una estadística del tiempo de cómputo de cada implementación, con las que, en trabajos posteriores se pretende hacer el particionamiento de manera automática haciendo uso de algoritmos heurísticos. Los bloques que actualmente se han construido son filtros digitales FIR e IIR.

Finalmente, se mostró que a partir de los conceptos teóricos del diseño de sistemas operativos, tales como la sincronización de tareas, es posible realizar, con ciertas modificaciones, la comunicación entre procesos software y hardware, con el ánimo de construir una arquitectura para el codiseño de aplicaciones de procesamiento de señales.

Al respecto, se incluyen como modificaciones el hecho que las tareas no sean apropiativas y se usan semáforos para sincronizar y ordenar la ejecución de los bloques funcionales. Estos semáforos son una variante de los semáforos típicos, puesto que incluyen información de las señales dependientes, el número de datos que debe esperar cada señal, y funciones especializadas para el movimiento datos de un bloque software a software y de un bloque hardware a software, y viceversa.

AGRADECIMIENTOS

Este trabajo se ha podido realizar gracias al apoyo financiero de la Universidad del Quindío a través del proyecto 306.

BIBLIOGRAFÍA

(1) Meyer-Baese U. Digital Signal Processing with Field Programmable Gate Arrays. Berlin: Springer-Verlag; 2001.
(2) Wiangtong T, Cheung P, Luk W. Hardware/Software Codesign. IEEE Signal Processing Magazine. 2005; 22:14-22.
(3) Kalavade A, Lee E. A hardware-software codesign methodology for DSP applications. IEEE Design & Test of Computers. 1993; 10:16-28.
(4) Ondghiri H, Kaminska B, Rajski J. A hardware/software partitioning technique with hierarchical design space exploration. Proc. of IEEE Custom Integrated Circuits Conference. 1997; 95-98.
(5) Burns A, Wellings A. Sistemas de tiempo real y lenguajes de programación. Pearson; 2005.
(6) Buhr R.J.A, Bailey D. L. An introduction to real-time systems from design to networking with C/C++. Prentice Hall; 1998.