INSTITUTO TECNOLGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

PROGRAMA DE GRADUADOS EN MECATRONICA Y TECNOLOGIAS DE INFORMACION

PLATAFORMA PARA VALIDAR EL DESEMPEÑO DE LA COMUNICACIÓN Y SINCRONIZACIÓN DE UN SISTEMA CON DOBLE NÚCLEO

TESIS

PRESENTADA COMO REQUISITO PARCIAL
PARA EL GRADO DE

MAESTRO EN CIENCIAS CON ESPECIALIDAD EN INGENIERIA ELECTRONICA
(SISTEMAS ELECTRONICOS)

POR

OMAR ALEJANDRO CAVAZOS RODRIGUEZ

MONTERREY, N.L., MEXICO. DICIEMBRE, 2009

INSTITUTO TECNOLGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

PROGRAMA DE GRADUADOS EN MECATRONICA Y TECNOLOGIAS DE INFORMACION

Los miembros del comité de tesis aprobamos la presente tesis de Omar Alejandro Cavazos Rodríguez que es requisito parcial para el grado de Maestro en Ciencias con especialidad en

Ingenieria Electrónica
(Sistemas Electrónicos)

Comité de Tesis

Dr. Alfonso Ávila Ortega
Asesor de Tesis

Dr. Graciano Dieck Assad
Sinodal de Tesis

Dr. Sergio Ramírez Chapa
Sinodal de Tesis

Dr. Joaquín Acevedo Mascuarúa
Director del programa de graduados en Mecatrónica y Tecnologías de Información
Diciembre, 2009

PLATAFORMA PARA VALIDAR EL DESEMPEÑO DE LA COMUNICACIÓN Y SINCRONIZACIÓN DE UN SISTEMA CON DOBLE NÚCLEO

POR

OMAR ALEJANDRO CAVAZOS RODRIGUEZ

TESIS

PRESENTADA AL PROGRAMA DE GRADUADOS EN MECATRONICA Y TECNOLOGIAS DE INFORMACION

ESTA TESIS ES UN REQUISITO PARCIAL PARA EL GRADO DE MAESTRO EN CIENCIAS CON ESPECIALIDAD EN

INGENIERIA ELECTRONICA
(SITEMAS ELECTRONICOS)

INSTITUTO TECNOLGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

DICIEMBRE, 2009

A mis padres, ya que sin ellos no sería la persona que soy ahora.

Reconocimientos

Gracias a mis profesores por toda la ayuda que me brindaron, a mis amigos y compañeros por su amistad y a mi familia por su apoyo y su paciencia.

Omar Alejandro Cavazos Rodríguez

Instituto Tecnolgico y de Estudios Superiores de Monterrey
Diciembre, 2009

Plataforma para validar el desempeño de la comunicación y sincronización de un sistema con doble núcleo

Omar Alejandro Cavazos Rodríguez
Instituto Tecnolgico y de Estudios Superiores de Monterrey, 2009

Asesor de Tesis: Dr. Alfonso Ávila Ortega

Resumen

Con el incremento en la complejidad de los dispositivos móviles, los procesadores necesitan de más poder de procesamiento para poder seguir a la par con la demanda de las aplicaciones multimedia. Gracias a las ventajas de utilizar un esquema de procesamiento en paralelo, al utilizar un dispositivo multiprocesamiento es posible aumentar el poder de procesamiento y así obtener un mejor desempeño. En este trabajo se desarrollaron esquemas y procedimientos para determinar el desempeño de dichos dispositivos, así como también se logró la implementación de un sistema multinúcleo utilizando dos núcleos ARM y un esquema de intercomunicación basado en el uso de buzones, implementado en una memoria compartida. En las pruebas de desempeño realizadas en este trabajo, se observa un desempeño similar entre el núcleo de propósito general usado en la OMAP (ARM926) y la implementación en software de dicho núcleo, lo cual nos permite validar los resultados obtenidos en Seamless para su comparación con la tarjeta de desarrollo.

Con el esquema multinúcleo propuesto se logró una mejora de un 21 % con respecto a una implementación que utiliza un solo núcleo. Es posible mejorar el desempeño de esta implementación al optimizar la intercomunicación entre los núcleos y su uso por medio de un BIOS controlado por un sistema operativo.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Justificación

Debido a las ventajas que otorga el procesamiento en paralelo, se eligió implementar un sistema multiprocesador, utilizando procesadores ARM con alto poder de procesamiento, por su eficacia al usar lenguajes de alto nivel para su programación.

Para realizar la intercomunicación entre los dos procesadores se hará uso de un BIOS, el cual se encarga de administrar la comunicación entre los procesadores además de las señales de control necesarias, lo cual facilita el desarrollo de software al permitir a un procesador utilizar funciones o subrutinas localizadas en el procesador remoto como si se encontraran localmente.

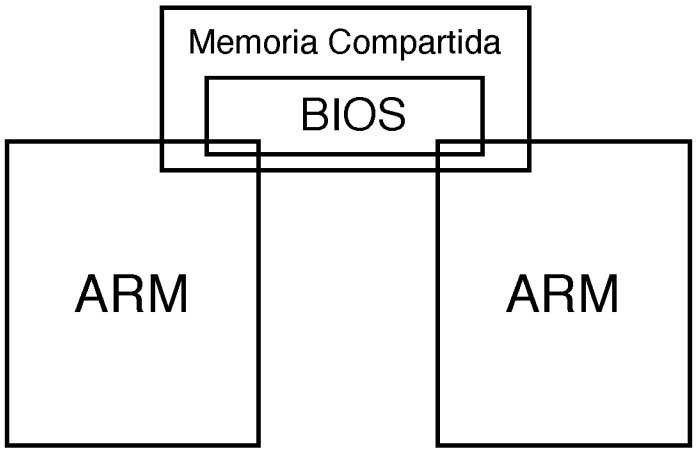


Figura 1.1: Intercomunicación de los procesadores usando un BIOS.

El diseño del BIOS, utiliza un esquema de comunicación entre procesos ("inter-

process communications" IPC), para intercambio de mensajes usando buzones ("mailboxes") implementados en una memoria compartida. Se tomará como referencia una implementación de un BIOS para la intercomunicación de un procesador de propósito general y un DSP llamado "DSP-BIOS Bridge" utilizado en los procesadores OMAP de Texas Instruments.

1.2. Descripción del problema

Con el incremento en la complejidad de los dispositivos móviles, los procesadores necesitan de más poder de procesamiento para poder seguir a la par con la demanda de las aplicaciones multimedia, sin embargo, al aumentar la velocidad de los procesadores aumenta el costo del dispositivo, además de aumentar el consumo de energía así como también aumenta la complejidad del diseño debido a factores como el calentamiento, ruido o interferencia debido a la alta frecuencia en el procesador.

En la actualidad los dispositivos móviles multimedia(principalmente celulares) utilizan un procesador que realiza todas las funciones del teléfono. Sin embargo, conforme las demandas del usuario crecen, cada vez es más difícil mantener la funcionalidad de los dispositivos debido a los costos (muchas veces prohibitivos en los productos de uso común) de los procesadores más veloces.

Si reemplazamos el esquema de procesamiento único por un esquema de procesamiento en paralelo usando multiprocesamiento, sería posible obtener el desempeño deseado al aumentar el poder de procesamiento del sistema sin aumentar la velocidad del mismo; sin embargo la implementación pudiera no ser tan trivial debido a la dificultad que representa la intercomunicación entre los procesadores.

El problema a resolver sería entonces el reemplazar el esquema de procesamiento usando un solo procesador por un esquema de procesamiento paralelo usando dos procesadores que utilicen un sistema de intercomunicación entre procesos para mejorar el desempeño en el caso particular de los dispositivos móviles multimedia.

1.3. Trabajo Relacionado

Entre los trabajos relacionados, el más notorio es el desarrollo de una serie de librerías y drivers para el funcionamiento del BIOS de los procesadores OMAP para el sistema operativo Linux desarrollado por Nokia Corporation [6].

Un trabajo de investigación realizado por la Universidad de Bucarest demuestra como implementar una aplicación existente que utiliza un solo procesador en un esquema paralelo sin reescribir el código existente usando un procesador OMAP [1].

Otro trabajo realizado por la Universidad Nacional de Seúl es una metodología para el co-diseño de hardware y software para sistemas embebidos multimedia utilizando Seamless como herramienta de co-verificación [7].

1.4. Objetivos

Este trabajo consta de diferentes objetivos. El primero es determinar el desempeño del procesador de propósito general (ARM926) en la tarjeta de desarrollo (OMAP) y en el software de co-diseño Seamless. Otro de los objetivos es el implementar un esquema de intercomunicación entre dos procesadores usando una memoria compartida usando Seamless y obtener su desempeño. Como una continuación a futuro de este trabajo se puede agregar el soporte para el manejo de la memoria compartida desde un sistema operativo por medio de librerías y drivers.

1.5. Contribución

En este trabajo se busca establecer esquemas y procedimientos para la medición de desempeño en procesadores OMAP usando Linux, esquemas y procedimientos para la co-verificación de hardware y software de un sistema multiprocesador, la implementación de una memoria compartida, la implementación de un sistema de comunicación entre procesos mediante mailboxes y la plataforma de verificación de Seamless.

1.6. Organización de la tesis

A continuación se presenta la forma en que fue organizada esta tesis.

Capítulo 1 - Introducción. En este capítulo se explica el trabajo propuesto, el problema que confronta, así como también los objetivos propuestos para solucionar dicho problema.

Capítulo 2 - Marco Teórico. Este capítulo introduce los elementos más relevantes utilizados en este trabajo necesarios para la comprensión del desarrollo de la tesis.

Capítulo 3 - Plataforma de trabajo OMAP y su configuración. Este capítulo demuestra cómo crear el entorno necesario para el funcionamiento de la plataforma de trabajo OMAP, así como su uso para realizar las pruebas comparativas.

Capítulo 4 - Plataforma de co-verificación Seamless. En este capítulo se explica el funcionamiento de la plataforma de co-verificación Seamless, así como también se explica la implementación del sistema multiprocesador propuesto en los objetivos de esta tesis.

Capítulo 5 - Metodología y resultados. En este capítulo se explica la metodología seguida en el transcurso de este trabajo, además de que se demuestran los resultados obtenidos.

Capítulo 6 - Conclusiones y trabajo futuro. Este capítulo explica las conclusiones obtenidas en este trabajo, además de proponer trabajos derivados de esta tesis.

Apéndice A - Códigos utilizados. Aquí se encuentran todos los códigos utilizados en esta tesis.

Apéndice B - Problemas encontrados. Aquí se explican algunos de los problemas encontrados a lo largo del desarrollo de esta tesis.

Capítulo 2

Marco Teórico

2.1. ARM

El procesador ARM tiene una arquitectura RISC de 32 bits desarrollada por ARM Limited y es ampliamente usada en diseños de sistemas embebidos. Debido a sus características de ahorro de energía, los procesadores ARM dominan el mercado de dispositivos electrónicos móviles. Hoy en día, se estima que aproximadamente 90 % de los procesadores embebidos RISC de 32 bits son parte de la familia de procesadores ARM [8]

2.1.1. Historia

El diseño del ARM comenzó en 1983 como un proyecto de desarrollo en la empresa Acorn Computers Ltd para construir un procesador RISC compacto. Roger Wilson y Steve Furber lideraban el equipo, cuya meta de diseño un manejo de interrupciones de baja latencia de entrada/salida similar al procesador MOS 6502. La arquitectura de acceso a memoria del MOS 6502 le permitía a los desarrolladores producir máquinas rápidas sin necesidad de utilizar un costoso hardware de acceso directo a memoria (DMA). El equipo terminó el diseño preliminar y los primeros prototipos del procesador en el año 1985, al que llamaron ARM1 [9]. La primera versión utilizada comercialmente fue conocida como ARM2 en el año 1986.

2.1.2. Desarrollo

La arquitectura del ARM2 posee un bus de datos de 32 bits y ofrece un espacio de direcciones de 26 bits, junto con 16 registros de 32 bits. Su sucesor, el ARM3, incluye una pequeña memoria caché de 4 KB, lo que mejora los accesos a memoria repetitivos. A finales de los años 80, Apple Computer comenzó a trabajar con Acorn en nuevas versiones del núcleo ARM. Este trabajo derivó en el ARM6, presentado en 1991.

2.1.3. Características Principales

Dentro de las características principales de la arquitectura ARM se encuentran:

- Arquitectura Load/Store.
- Ancho de instrucción fijo de 32 bits para facilitar la decodificación y segmentación (pipelining).
- Ejecución condicional en la mayoría de las instrucciones, reduciendo el "branch overhead".
- No soporta accesos de memoria desalineados.
- Ejecución de instrucciones mayormente en un solo ciclo.

2.2. Qué es OMAP?

La plataforma OMAP está basada en una combinación de un DSP TMS320C55xTM y un CPU de alto rendimiento ARM926EJ-S, ambos de Texas Instruments. Una arquitectura RISC como la del ARM926EJ-S está más orientada al código de control (Sistema operativo, Interfaz de usuario, aplicaciones, etc.). Un DSP está orientado al procesamiento de señales, como la decodificación de video (MPEG2, MPEG4, h.264), reconocimiento de voz, reproducción de audio, etc. La arquitectura OMAP combina ambos procesadores para obtener los máximos beneficios de ambos.

La arquitectura del OMAP está diseñada para maximizar el desempeño general del sistema de un dispositivo 3G mientras mantiene el consumo de potencia al mínimo. Ambos procesadores utilizan un cache de instrucciones para reducir el tiempo de acceso promedio a la memoria de instrucciones y eliminar accesos externos que consuman mucha potencia. Además, ambos núcleos tienen una unidad de mantenimiento de memoria (MMU por sus siglas en ingles) para la translación de la memoria virtual a la física, y para la protección de la memoria de tareas.

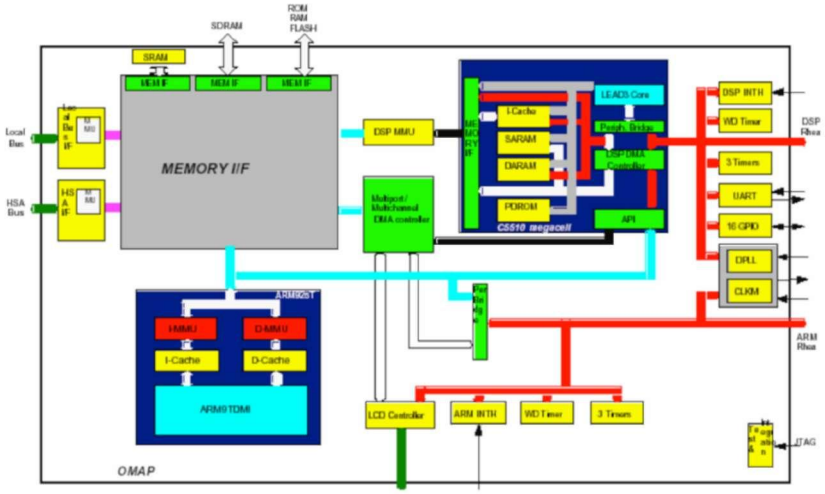


Figura 2.1: Arquitectura OMAP [1]

La arquitectura OMAP incluye una infraestructura de software abierto que soporta el desarrollo de aplicaciones y posee la capacidad de actualizado dinámico para el diseño de sistemas heterogéneos y de multiprocesamiento. Esta infraestructura incluye una plataforma para el desarrollo de software que se enfoca en el diseño del sistema y APIs para ejecutar software en el sistema de destino.

Los sistemas inalámbricos 2.5G y 3G combinan el modelo clásico centrado a la voz con la funcionalidad de datos de un asistente personal (PDA). Las aplicaciones multimedia (video, MP3, etc.) también serán descargadas a futuras plataformas telefónicas. Estos sistemas también tendrán que alojar una variedad de sistemas operativos populares como WinCETM, EPOCTM, entre otros. Las aplicaciones dinámicas multitarea requerirán también el uso de sistemas operativos en el DSP.

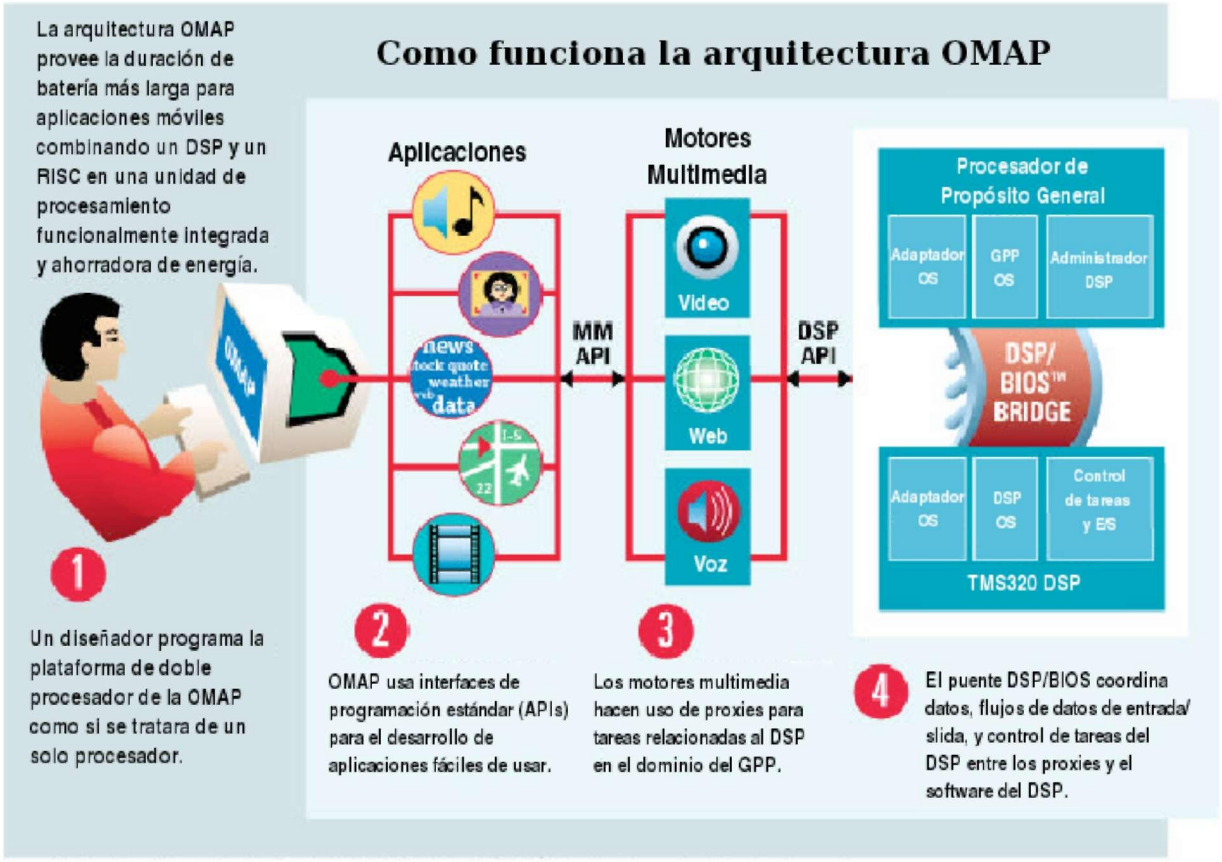


Figura 2.2: Funcionamiento del OMAP [1]

2.2.1. DSP/BIOS Bridge

El Puente DSP/BIOS (DSP/BIOS Bridge) es la clave para la funcionalidad y la facilidad de uso de la plataforma OMAP. Provee al desarrollador del software de aplicación una interfaz al DSP sencilla y fácil de usar. También permite al desarrollador accesar y controlar desde el procesador RISC el entorno del DSP usando una interfaz estandarizada de aplicación-programación (API).

La función más importante del puente DSP/BIOS es la de proveer comunicación entre las aplicaciones del GPP (Procesador de propósito general) y las tareas del DSP.

La API del puente DSP/BIOS proviene de un conjunto de DLLs y controladores que son proporcionados en el kit de desarrollo de la plataforma. Esto le permite al los desarrolladores de aplicación desarrollar en la plataforma OMAP de la misma manera que si estuvieran desarrollando para un solo procesador RISC.

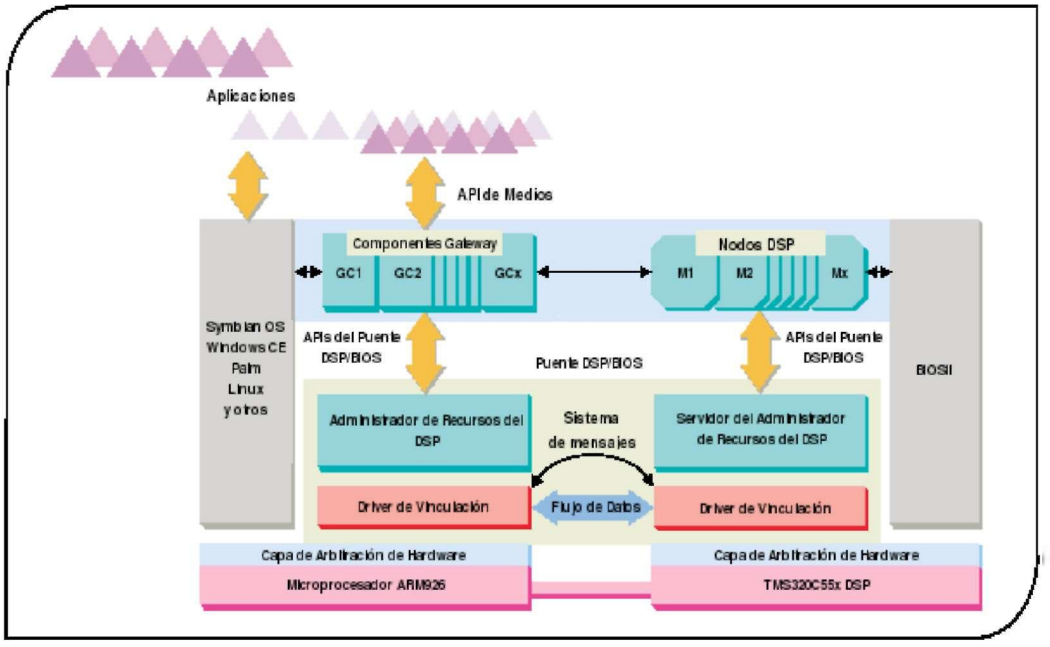


Figura 2.3: OMAP Bios [2]

Texas Instruments realizó una prueba comparativa basada en datos publicados en la cual se muestra que una tarea de procesamiento de señal típica ejecutada en los procesadores RISC más modernos requiere tres veces más ciclos que los que son requeridos en el DSP C55xTM utilizado en el OMAP.

	[illegible]	StrongARM 1100	TMS320C5510	Units
Echo cancellation 16 bits (32 ms - 8 kHz)	24	39	4	Mcycles/s
Echo cancellation 32 bits (32 ms - 8 kHz)	37	41	15	Mcycles/s
MPEG4/H263 decoding QCIF @ 15 fps	33	34	17	Mcycles/s
MPEG4/H263 encoding QCIF @ 15 fps	179	153	41	Mcycles/s
JPEG decoding (QCIF)	2.1	2.06	1.2	Mcycles/s
MP3 decoding	19	20	17	Mcycles/s
Average cycle ratio with C5510™	3.1	3	1	

Figura 2.4: Comparativa de desempeo [3]

2.3. Linux

2.3.1. Historia

En 1991, en Helsinki, Linus Torvalds comenzó un proyecto que más tarde se llegó a ser el núcleo Linux. Esto fue al principio un emulador terminal, al cual Torvalds solía tener acceso en los grandes servidores UNIX de la universidad. El escribió el programa expresamente para el hardware que usaba, e independiente de un sistema operativo, porque quiso usar las funciones de su nueva computadora personal con un procesador 80386. Este es aún el estándar de hoy. El sistema operativo que él usó durante el desarrollo fue Minix, y el compilador inicial fue el GNU C compilador, que aún es la opción principal para compilar Linux hoy.

Torvalds primero publicó el núcleo Linux bajo su propia licencia, la cual fue casi una licencia de código fuente compartida y que tenía una restricción contra la actividad comercial. En 1992, él sugirió cambiar a la GNU GPL (General Public License) o Licencia Pública General la cual es una licencia que está orientada principalmente a proteger la libre distribución, modificación y uso de software. [11]

2.3.2. Desarollo

Linus nunca anunció la versión 0.01 de Linux (agosto 1991), ya que esta versión no era ejecutable, solamente incluía los principios del núcleo del sistema, estaba escrita en lenguaje ensamblador y asumía que uno tenía acceso a un sistema Minix para su compilación.

El 5 de octubre de 1991, Linus anuncio la primera versión"Oficial"de Linux (versión 0.02). Con esta versión Linus pudo ejecutar Bash (GNU Bourne Again Shell) y gcc (Compilador GNU de C) pero no mucho mas funcionaba. Después de la versión 0.03, Linus salto en la numeración hasta la 0.10. Después de esto, más programadores empezaron a trabajar en el proyecto y después de varias revisiones, Linus incremento el numero de versión hasta la 0.95 (marzo 1992). En Diciembre de 1993 el núcleo del sistema estaba en la versión 0.99 y la versión 1.0, llego el 14 de marzo de 1994. [12]

2.3.3. Características Principales

- Multitarea: La palabra multitarea describe la habilidad de ejecutar varios programas al mismo tiempo. LINUX utiliza la llamada multitarea preventiva, la cual asegura que todos los programas que se están utilizando en un momento dado serán ejecutados, siendo el sistema operativo el encargado de ceder tiempo de microprocesador a cada programa.

- Multiusuario: Se refiere a la capacidad de poder utilizar una misma maquina por más de un usuario al mismo tiempo.

- Multiplataforma: Es posible utilizar Linux en diversas plataformas como por ejemplo x86, Alpha, ARM, MIPS, PowerPC y SPARC.

- Multiprocesador: Soporte para sistemas con más de un procesador.

- Carga de ejecutables por demanda: Linux sólo lee del disco aquellas partes de un programa que están siendo usadas actualmente.

- Protección de la memoria entre procesos: Un proceso no puede accesar la memoria de otro evitando la corrupción de datos de procesos contiguos.

Capítulo 3

Plataforma de trabajo OMAP y su configuración

3.1. Preparación del sistema de desarrollo

3.1.1. Herramientas necesarias

Para poder empezar el desarrollo de este trabajo, es necesario contar con diversas herramientas tanto de de hardware como de software para poder implementar el sistema operativo en la tarjeta de desarrollo:

3.1.1.1. Hardware

A continuación se listan los componentes requeridos para el desarrollo del proyecto.

- Una computadora con arquitectura x86 utilizando un sistema operativo Linux y Windows.
- Un cable JTAG XDS510-PP de Spectrum Digital.
- Cable ethernet cruzado.
- Cable serie hembra-hembra (nulo).
- Tarjeta de desarrollo OMAP1610H2.
- Tarjeta de desarrollo Beagleboard basada en un procesador OMAP3.

3.1.1.2. Software

En cuanto al software, los siguientes elementos y paquetes son requeridos para proceder y facilitar el uso del hardware.

- Herramientas de compilación cruzada (cross-toolchain).
- Código fuente del sistema de arranque (bootloader) U-boot [13].
- Código fuente del kernel de Linux.
- Un sistema de archivos (filesystem) específicamente diseñado para la tarjeta de desarrollo.
- IBoot de Productivity Systems Inc.
- Code Composer Studio de Texas Instruments.

La preparación del sistema de desarrollo consiste de dos partes: una parte se debe realizar en Linux, mientras que la otra parte debe realizarse en Windows.

3.1.2. Linux

3.1.2.1. Toolchain

Lo primero que se debe realizar es la creación de un "cross-toolchain" o cadena de herramientas cruzada, necesaria para compilar los códigos fuente en archivos binarios ejecutables. Para este trabajo se optó por utilizar un cross-toolchain ya compilado y listo para usar de CodeSourcery [14] para la compilación cruzada (cross-compiling) de códigos para una plataforma ARM, debido a que es un cross-toolchain muy completo y fácil de usar.

3.1.2.2. U-boot

Una vez que se tiene el cross-toolchain debidamente instalado y funcionado es necesario compilar el "bootloader" o cargador de arranque, el cual se encarga de inicializar el procesador y de pasar los parámetros de arranque al sistema operativo. Los pasos necesarios son los siguientes:

1. Primero es necesario modificar el Makefile[1] del proyecto; para esto buscamos las siguientes lineas:

```
ifeq (\$(ARCH),arm)
CROSS_COMPILE = arm-linux-
endif
```

y reemplazamos el valor de la variable *CROSS_COMPILE* de la siguiente manera:

```
CROSS_COMPILE=arm-none-linux-gnueabi-
```

donde *arm-none-linux-gnueabi-* es el prefijo que utiliza el toolchain.

2. En una terminal del sistema, navegamos a la carpeta principal donde se encuentra U-boot y tecleamos:

```
make omap1610h2
```

Esto configura y compila el código para una tarjeta desarrollo OMAP1610H2, que es la tarjeta que se usó para esta prueba.

3. Una vez compilado el código fuente, obtenemos tres archivos compilados:

```
u-boot.bin
u-boot
u-boot.srec
```

los cuales contienen el mismo código de arranque pero difieren en la forma en que fueron compilados para poder ser cargados en nuestra tarjeta de desarrollo de diferentes formas.

[1]Archivo de texto que contiene los parámetros necesarios para compilar código fuente de forma automatizada

3.1.2.3. Kernel de Linux

El siguiente paso es configurar y compilar el kernel de linux de acuerdo a las necesidades de la tarjeta de desarrollo.

Para esto, primero es necesario contar con el código fuente del kernel [15]. Además es necesario contar con un parche que contiene la información necesaria para poder ser usado por la tarjeta de desarrollo. Este parche [16] debe de corresponder a la versión del kernel que se obtuvo previamente. Para aplicarlo, debemos entrar a la carpeta principal donde se encuentra el kernel y teclear el siguiente comando:

```
cat {ruta al parche} | patch -p1
```

Con el kernel parchado, se puede empezar la configuración del kernel:

1. Primero debemos modificar el Makefile para especificar que vamos a hacer una compilación cruzada. Buscamos la siguiente linea:

   ```
   CROSS_COMPILE ?=
   ```

   y agregamos la ruta de nuestro toolchain:

   ```
   CROSS_COMPILE ?=arm-none-linux-gnueabi-
   ```

2. Con el Makefile modificado, tecleamos el siguiente comando:

   ```
   make omap_1610_h2_defconfig
   ```

   Esto nos crea un archivo de configuración con las opciones por defecto para la tarjeta.

3. Después se debe de personalizar la configuración para agregar opciones extra al kernel, como por ejemplo, soporte para NFS, soporte para JFFS2, entre otras opciones. Para esto debemos teclear:

   ```
   make menuconfig
   ```

4. Posteriormente debemos compilar el kernel tecleando:

   ```
   make
   ```

5. Ahora que tenemos el kernel compilado, es necesario convertirlo a un formato que U-boot pueda reconocer. Para esto debemos teclear lo siguiente:

```
make uImage
```

3.1.2.4. Sistema de Archivos

Para crear el sistema de archivos, es necesario utilizar Busybox [17] y udev [18] . Busybox es un paquete de utilidades que contiene versiones ligeras de las aplicaciones más comunes de linux y es ideal para sistemas embebidos. Udev permite tener una lista dinámica de dispositivos detectando automáticamente los dispositivos encontrados en el sistema sin necesidad de agregar los dispositivos manualmente.

Los pasos a seguir para la creación del sistema de archivos son los siguientes:

1. Primero debemos modificar el Makefile para especificar el uso de nuestro toolchain, de la misma manera que en los pasos anteriores; quedando de la siguiente forma:

   ```
   CROSS_COMPILE ?= arm-none-linux-gnueabi-
   ```

2. Ahora configuramos el programa para usar los programas por defecto:

   ```
   make defconfig
   ```

3. Finalmente tecleamos los siguientes comandos:

   ```
   make
   make install
   ```

   Esto compilará e instalará el software en la carpeta *_install*, la cuál es creada automáticamente dentro de la carpeta principal donde se encuentra Busybox. Esta carpeta contiene la estructura básica del sistema de archivos.

4. Después de esto, debemos agregar las librerías necesarias a nuestro sistema de archivos. Para esto, debemos copiar las librerías usadas por nuestro toolchain a la carpeta *lib* de nuestro sistema de archivos. Esto se puede hacer de la siguiente forma:

```
mkdir _install/lib
cp -rvf {ruta a toolchain}/arm-none-linux-gnueabi/libc/lib/* _install/lib/
cp -rvf {ruta a toolchain}/arm-none-linux-gnueabi/lib/* _install/lib/
```

5. Con los programas y librerías instalados ahora debemos de configurar y compilar udev. Para esto, nos vamos a la carpeta de udev y modificamos el Makefile como sigue:

   ```
   DESTDIR = {ruta de la carpeta _install creada por Busybox}
   ```

   En esta línea agregamos la ruta de la carpeta *_install* creada en los pasos anteriores para que los programas creados por udev, se instalen directamente en nuestro sistema de archivos.

   ```
   CROSS_COMPILE ?= arm-none-linux-gnueabi-
   ```

   En esta línea agregamos el prefijo de nuestro toolchain.

6. Con el Makefile listo, compilamos udev:

   ```
   make install
   ```

   Esto creará y colocará en el sistema de archivos los programas necesarios para el funcionamiento de udev, además de otros archivos y carpetas necesarios para su funcionamiento.

7. A continuación debemos crear el resto de la estructura del sistema de archivos. Esto se logra como sigue:

   a) Dentro de la carpeta *_install* creamos las carpetas restantes. Para esto, abrimos una terminal y tecleamos lo siguiente (en una sola línea):

   ```
   mkdir -p dev sys proc lib tmp var/lib/misc var/lock var/log var/run
   var/tmp etc/rc.d/rc.local var/lock/subsys/hotplug dev/pts /root
   ```

   b) Vinculamos los scripts de arranque a la carpeta correspondiente

   ```
   ln -s rc.d/init.d/ init.d
   ```

   c) Creamos los dispositivos terminales

```
sudo cp -avp /dev/console /dev/ttyS* dev
```

d) Cambiamos el propietario del programa principal de BusyBox a root

```
sudo chown root:root bin/busybox
```

e) Cambiamos los permisos a la carpeta de archivos temporales

```
chmod 1777 tmp var/tmp
```

f) Creamos los siguientes archivos (Referirse al Apéndice A para el contenido de los mismos)

```
etc/busybox.conf
etc/fstab
etc/group
etc/hosts
etc/inittab
etc/login.defs
etc/nsswitch.conf
etc/passwd
etc/securetty
etc/shadow
etc/udhcpd.conf
etc/rc.d/rcE
etc/rc.d/rcS
etc/rc.d/init.d/udev
```

g) Cambiamos permisos al archivo de passwords

```
chmod 400 etc/shadow
```

h) Damos permisos de ejecución a los scripts de arranque

```
chmod +x etc/rc.d/rcS etc/rc.d/rcE
```

i) Vinculamos los scripts de udev a la carpeta de arranque

```
cd /etc/rc.d/rc.local
ln -s ../init.d/hotplug S01-hotplug
```

```
ln -s ../init.d/hotplug E01-hotplug
ln -s ../init.d/udev S01-udev
ln -s ../init.d/udev E01-udev
```

3.1.3. Windows

3.1.3.1. XDS510PP

Ahora debemos instalar los drivers para poder utilizar y configurar el dispositivo JTAG, necesario para cargar el bootloader en la tarjeta de desarrollo.

Para esto, se deben seguir los siguientes pasos:

1. Primero debemos descargar e instalar los drivers de la página de Spectrum Digital [19].

2. Después de instalar el software, es necesario reiniciar la computadora y entrar al BIOS y asegurarnos que el puerto paralelo esté configurado como "SPP".

3. Ahora conectamos un extremo del JTAG al puerto paralelo de la computadora y el otro a la tarjeta de desarrollo.

4. A continuación, debemos conectar el cable de alimentación al JTAG, así como también, el cable de alimentación de la tarjeta de desarrollo.

   Posteriormente se procede a encender la tarjeta de desarrollo.

5. Una vez hecho lo anterior, procedemos a ejecutar el software de configuración del JTAG; un link a este software se debe encontrar en el escritorio de Windows bajo el nombre de "SDconfig".

6. Una vez dentro de este programa, buscamos en la ventana de lado izquierdo el apartado nombrado "378" y damos doble click en el.

7. La forma de configurar el JTAG se puede apreciar en la Figura 3.1.

8. Una vez hecho esto, seleccionamos el menú **File** y luego **Save**.

9. Salimos del programa y lo volvemos a abrir, para asegurarnos que se hayan salvado los cambios correctamente.

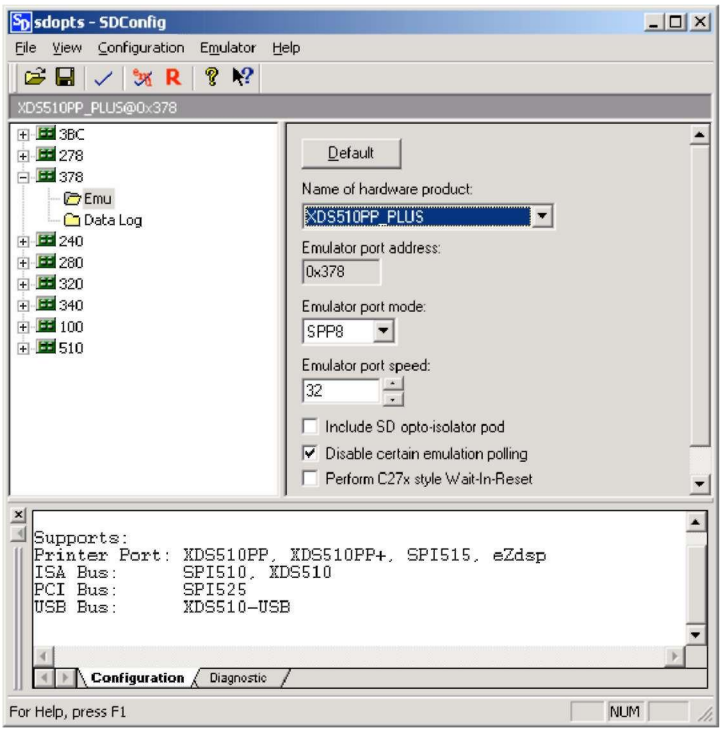


Figura 3.1: Configuración del JTAG [4].

10. Ahora seleccionamos el menú **Emulator** y luego **Emulator Test**. La siguiente información debe aparecer en la ventana inferior (Figura 3.2)

```
** Emulator Scan Test
  -- Found JTAG IR length of 50
  -- Found 3 JTAG device(s) in the scan chain
```

con esto, el JTAG ha quedado correctamente instalado.

3.1.3.2. Code Composer Studio

Code Composer Studio es un ambiente de desarrollo integrado (IDE por sus siglas en inglés) que incluye todas las herramientas necesarias para el desarrollo de aplicaciones para sistemas embebidos basados den procesadores tales como el OMAP.

Para poder utilizar este ambiente de desarrollo con la tarjeta de desarrollo, es necesario realizar una serie de pasos descritos a continuación:

1. Primero es necesario contar con una copia de este software, la cual puede ser

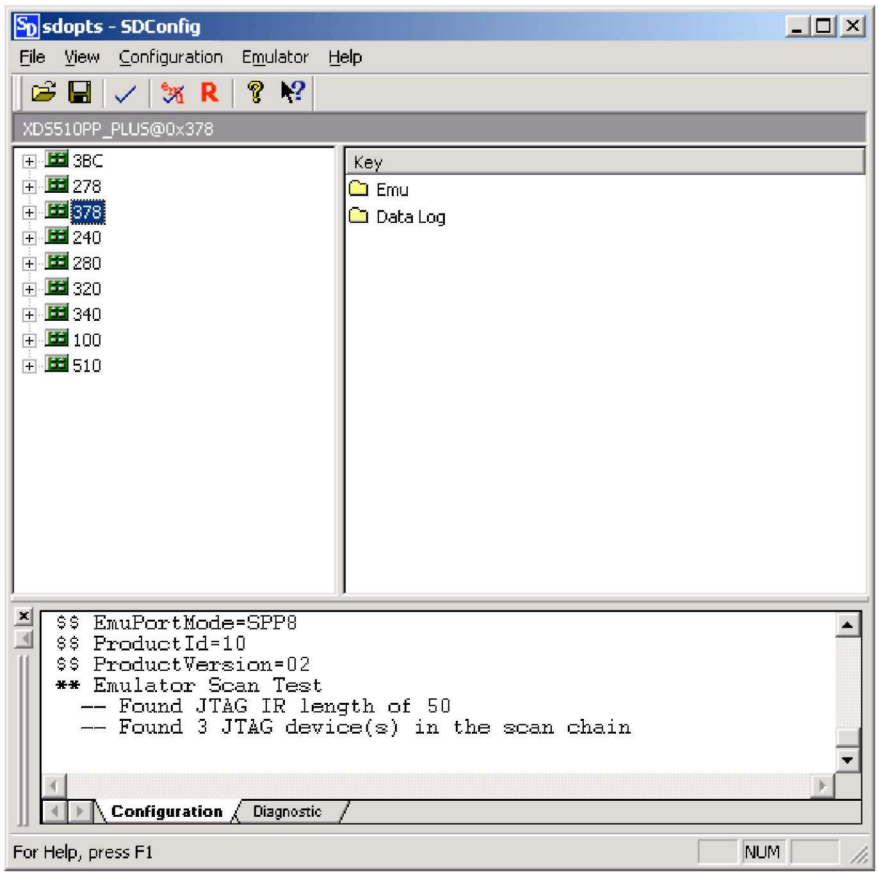


Figura 3.2: Prueba del JTAG [4].

adquirida directamente con Texas Instruments. También es posible descargar una evaluación de dicho software por 90 días desde su sitio web [20].

2. Ahora se procede a instalar el software. Una vez terminado el proceso, debemos reiniciar la computadora.

3. Después, debemos copiar los archivos *Innov_1610.gel* and *Slowclk.cfg*, los cuales se encuentran en el disco de documentación de la tarjeta de desarrollo, el cual fue proporcionado junto con el equipo, en la carpeta

   ```
   C:\ti\cc\gel
   ```

   que es donde se instala CCS por defecto.

4. Ahora procedemos a configurar CCS abriendo el link llamado "Setup CCS", el cual se encuentra en el escritorio.

5. En la ventana derecha seleccionamos **Install a Device Driver** (Figura 3.3).

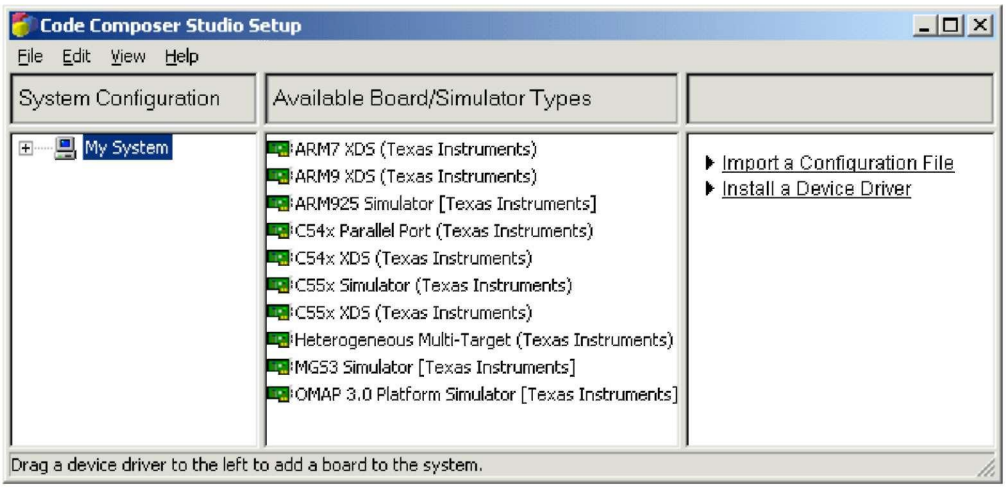


Figura 3.3: Ventana de configuración de CCS [4].

6. A continuación, debemos navegar a la carpeta **drivers** la cual se encuentra en la carpeta de instalación de CCS y seleccionamos el archivo *sdgo5xx.drv*.

7. En la ventana que aparece, solamente presionamos **OK**.

8. Ahora debemos repetir este procedimiento para todos los archivos con nombre *sdgoxxxx.drv* se encuentren instalados.

9. Una vez que todos los drivers se encuentren instalados, seleccionamos el menú **File** y luego **Save** y luego cerramos el programa.

10. Después, Reiniciamos el programa y en la ventana central seleccionamos **sdgoarm9** y lo arrastramos a la ventana de la izquierda, lo cual hace aparecer una ventana de configuración (Figura 3.4).

11. De la lista que aparece, seleccionamos **Auto-generate board data file with extra configuration file** (Figura 3.5).

12. En la ventana **Configuration File** navegamos al archivo *slowclk.cfg*.

13. Presionamos **Next**.

14. En la siguiente ventana, cambiamos el valor de **I/O Port** por "0x378" y presionamos **Next** (Figura **??**).

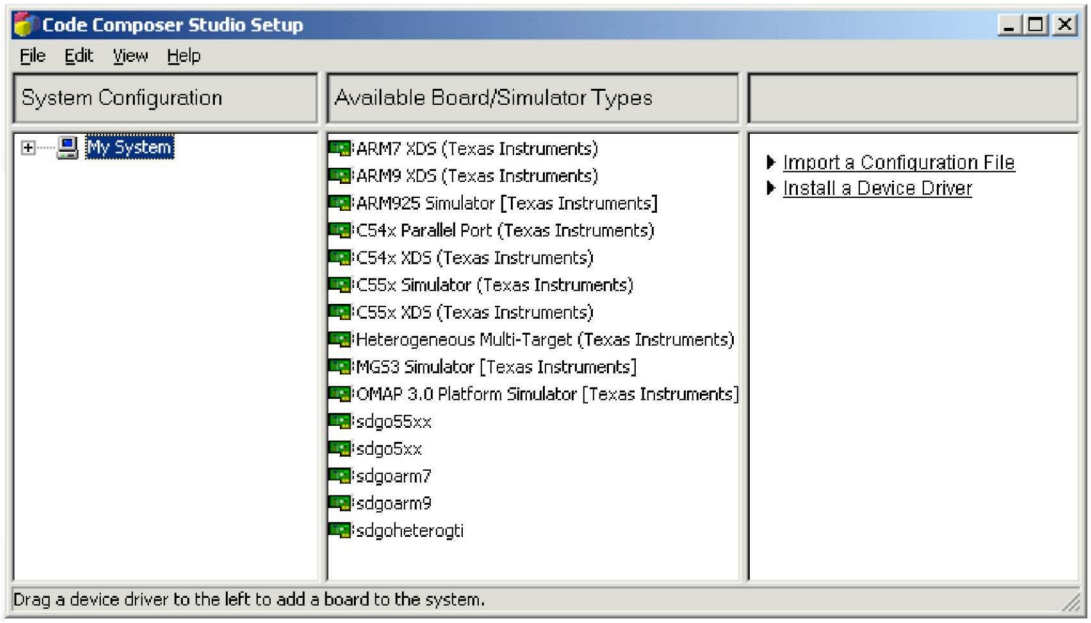


Figura 3.4: Ventana de configuración de CCS con los drivers cargados [4].

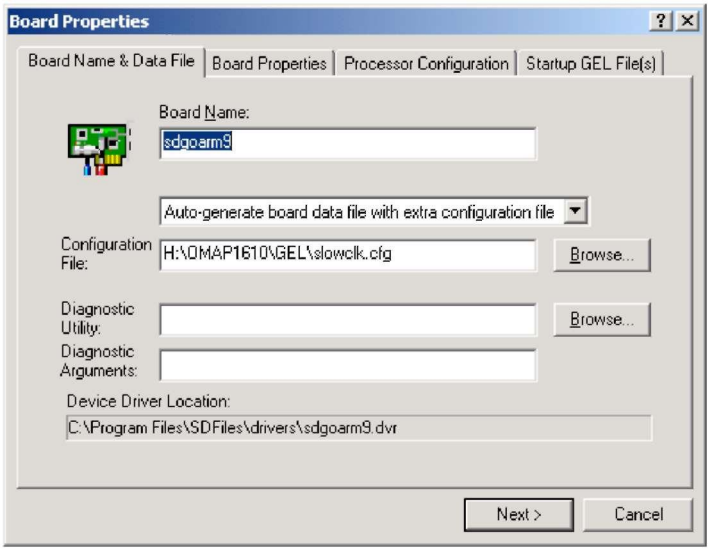


Figura 3.5: Ventana de propiedades de tarjeta [4].

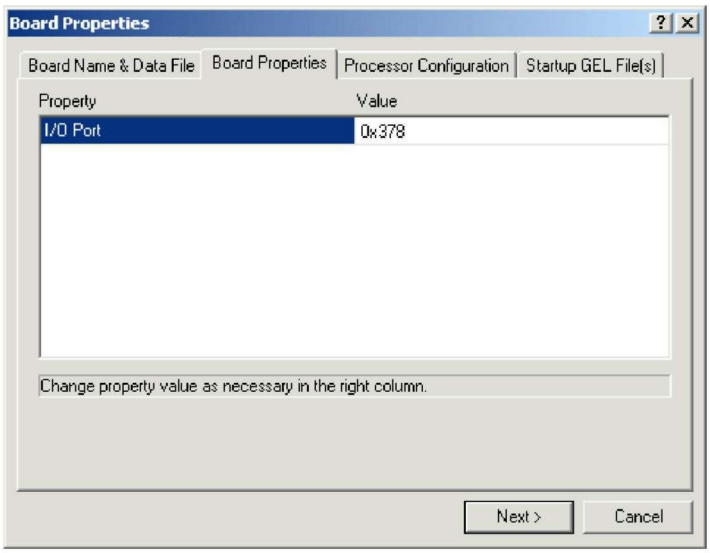


Figura 3.6: Ventana de propiedades de puerto I/O [4].
labelccs4

15. En la ventana de configuración del procesador (Figura 3.7) presionamos el botón **Add Single** y agregamos un "BYPASS" y aceptamos los valores por defecto.

16. Realizamos la misma operación, pero ahora seleccionando "TMS470R2x " de la lista. Antes de agregar el elemento, lo renombramos "ARM" en la ventana **Processor Name**.

17. Finalmente, repetimos la operación agregando otro "BYPASS" cambiando su nombre a "BYPASS_DSP".

18. En la ventana de configuración, cambiamos el valor de **Number of bits in the instruction register** a "38" y seleccionamos **OK** .

19. En el panel **Init Order** cambiamos el orden de inicialización a "3,1,2" haciendo click en el número y cambiando su valor. Luego presionamos **Next**.

20. En la siguiente ventana hacemos click en "ARM" y en el recuadro seleccionamos el archivo *Innov_1610.gel* (Figura 3.8).

21. Hacemos click en **Finish**.

22. Seleccionamos el menú **File** y luego **Save** y finalmente salimos del programa.

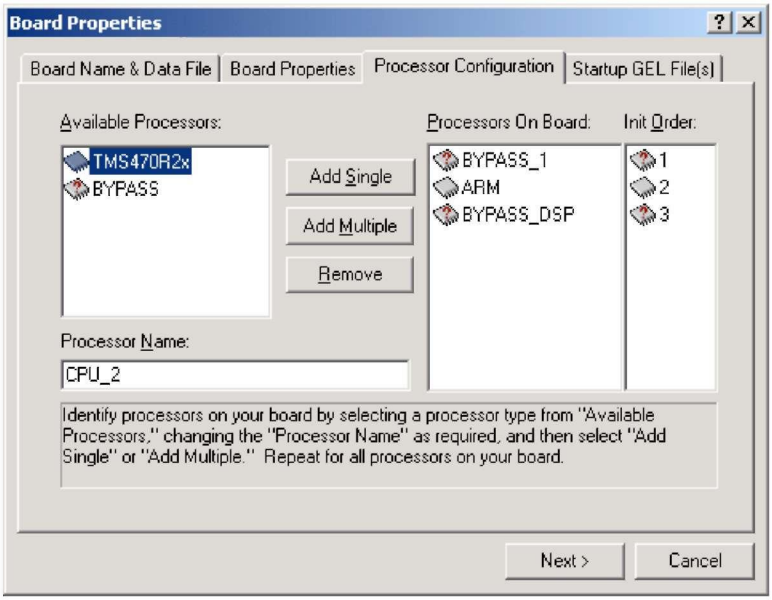


Figura 3.7: Ventana de configuración del procesador [4].

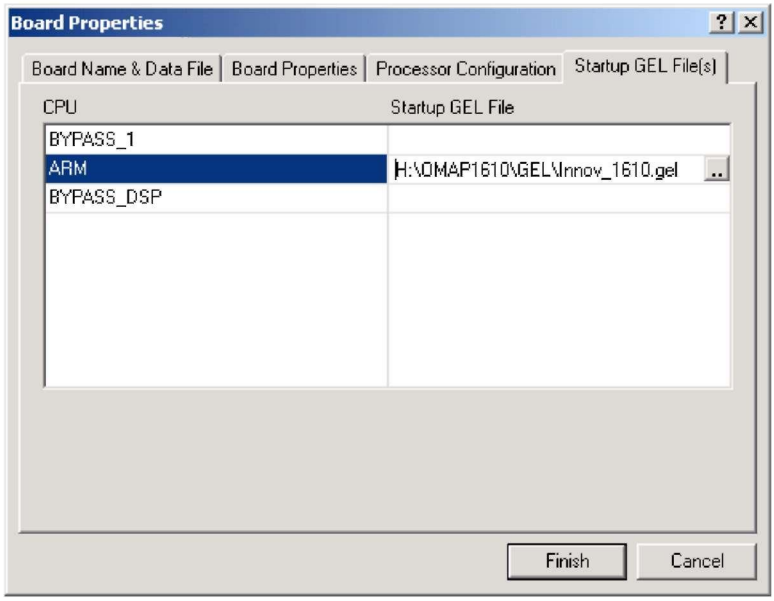


Figura 3.8: Ventana de selección de archivos de inicialización GEL [4].

23. Al cerrar el programa de configuración se nos pregunta si queremos iniciar Code Composer Studio. Seleccionamos la opción **Yes**.

24. Si todo está funcionando correctamente saldrá una ventana similar a la de la Figura 3.9.

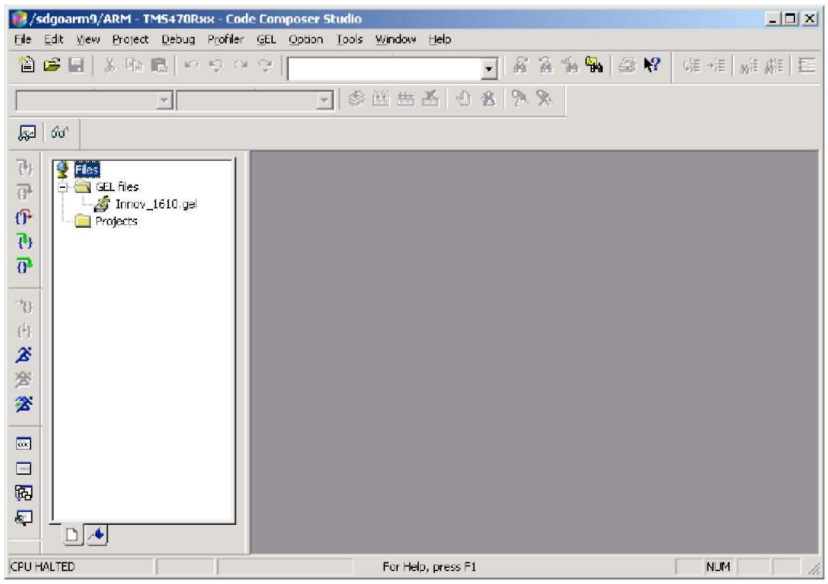


Figura 3.9: Ventana principal de CCS [4].

3.1.3.3. IBoot

3.1.3.3.1. IBoot Host IBoot Host provee una interfaz de usuario gráfica que permite cargar imágenes de sistema operativo (OS), archivos datos, etc. a través de una conexión USB o serial a un dispositivo que corra la aplicación IBoot.

Para instalar IBoot Host en la computadora, solamente es necesario encontrar e instalar con el instalador de IBoot Host. Este se encuentra en el disco que fue proporcionado junto con la tarjeta de desarrollo.

3.1.3.3.2. IBoot USB Driver Para instalar el driver USB es necesario que exista comunicación directa entre la computadora y la tarjeta de desarrollo; esto significa que la tarjeta de desarrollo debe de estar corriendo IBoot en memoria (ver Capítulo 4).

Los pasos para instalar el driver USB son los siguientes:

1. Primero nos aseguramos que IBoot ya se encuentra en memoria.

2. Después debemos abrir hyperterminal[2] y configurar la sesión con los parámetros por defecto a 57600 baudios (57600-8-N-1).

3. En la consola, tecleamos "usb" para activar la interfaz USB (Figura 3.10).

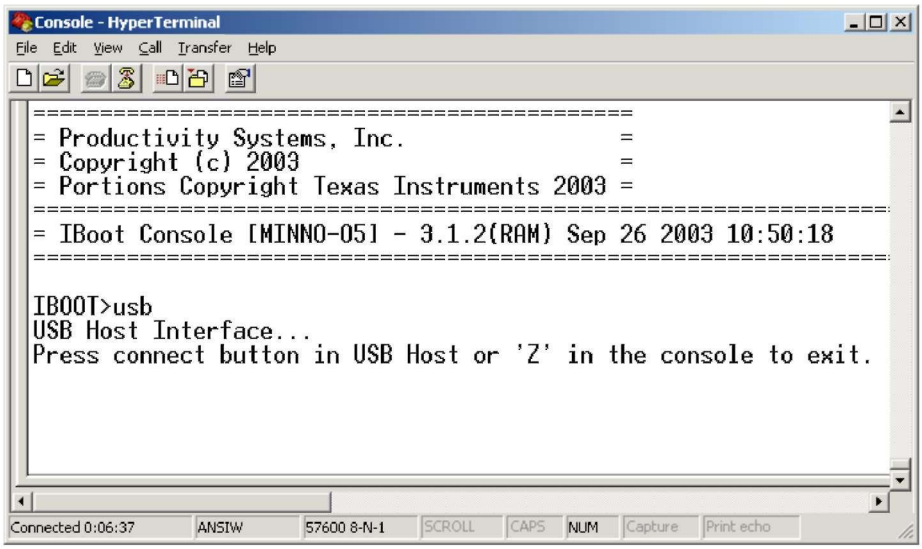


Figura 3.10: Consola de activación de USB [5].

4. Conectamos el cable usb a la computadora y a la tarjeta de desarrollo. Una ventana aparecerá indicando que se ha detectado nuevo hardware.

5. Instalamos el driver utilizando el archivo *InnovatorUSB.inf* que se encuentra en el disco de la tarjeta de desarrollo.

6. Por último nos aseguramos que el driver se haya instalado correctamente (Figura 3.11).

3.1.3.4. Preparando el Hardware

La tarjeta de desarrollo utilizada es una OMAP1610H2, la cuál consta de un módulo principal, en el cuál se encuentra el procesador; un módulo adaptador para conectar el cable JTAG y un módulo de expansión que es el módulo donde se encuentran todos los periféricos.

[2]Software emulador de terminal incluido en Windows

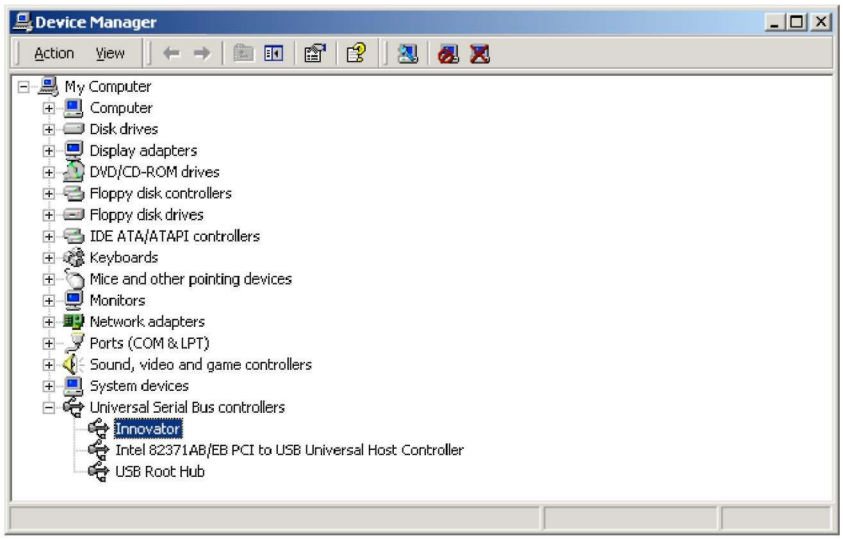


Figura 3.11: Administrador de dispositivos [5].

Antes de empezar a utilizar la tarjeta de desarrollo, debemos asegurarnos que el hardware esté conectado correctamente. La forma en que se realizan estos pasos no sigue un orden particular.

- Conectar la batería en su conector en la tarjeta de expansión.
- Conectar el cable serial a la computadora y a la tarjeta de desarrollo.
- Conectar el JTAG a la computadora usando el puerto paralelo y hacia la tarjeta de desarrollo usando el conector de 14 pines.
- Conectar el cable de red a los puertos de ethernet de la computadora y de la tarjeta.
- Conectar el cable miniusb-usb de la tarjeta de desarrollo a la computadora.
- Conectar el cable de alimentación del módulo principal al módulo de expansión.
- Conectar la fuente de poder de la tarjeta al módulo de expansión.
- Colocar el switch SW9 de la tarjeta principal en la posición “ON”.

Estas conexiones se pueden apreciar mejor en la Figura 3.12. Con esto, tenemos todo lo necesario para empezar a utilizar la tarjeta de desarrollo.

3.2. Uso de OMAP

Una vez que contamos con el hardware y software necesario para trabajar, se debe de instalar en la tarjeta de desarrollo.

3.2.1. Arrancando IBoot

Para poder inicializar la tarjeta de desarrollo, primero debemos cargar un bootloader en la memoria. Antes de cargar u-boot en la memoria flash de la tarjeta de desarrollo, es necesario cargar IBoot en la memoria ram. Para esto es necesario realizar lo siguiente:

1. Primero debemos arrancar hyperterminal usando los parámetros especificados en el capítulo anterior.

2. Después debemos arrancar Code Composer Studio y abrir el archivo *IBootLoader_H16.out* el cual se encuentra en el disco incluido en la tarjeta de desarrollo.

3. Una vez abierto este archivo, lo corremos presionando la tecla "F5". Con esto, ya tenemos IBoot corriendo en memoria.

4. En la ventada de hyperterminal tecleamos "usb" para inicializar la conexión por usb a la computadora.

5. Corremos el programa "IBoot Host".

6. En la barra de herramientas, seleccionamos "USB" de la lista y presionamos el ícono "Connect".

7. Renombramos el archivo *u-boot.bin* (ver capítulo 3) por *u-boot.raw*.

8. Llenamos todos los campos de la ventana principal como se muestra en la Figura 3.13.

9. Presionamos el ícono "Load" de la barra de herramientas. Con esto u-boot se ha cargado en la memoria flash de la tarjeta.

10. Cerramos todos los programas y apagamos la tarjeta de desarrollo.

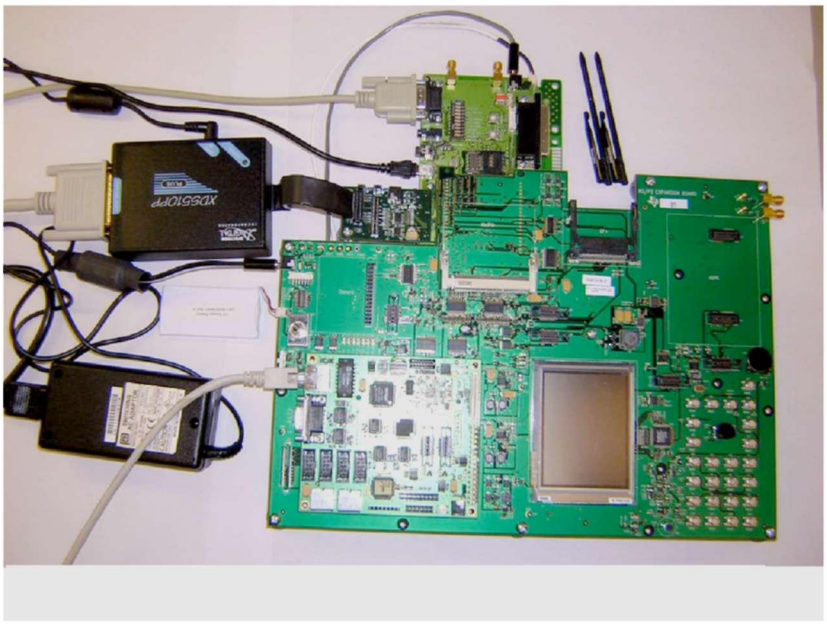


Figura 3.12: Tarjeta de desarrollo ensamblada.

Figura 3.13: Configuración en IBoot para la instalación de u-boot.

11. Volvemos a abrir hyperterminal con la configuración por defecto, pero con una velocidad de 115200 baudios (115200-8-N-1).

12. Cambiamos el switch SW9 de la tarjeta a la posición "OFF"

13. Encendemos la tarjeta de desarrollo y nos aseguramos que haya arrancado en u-boot correctamente. Deberá aparecer algo similar a lo siguiente:

```
U-Boot 1.0.0 (Feb 16 2004 - 11:51:41)
U-Boot code: 11000000 -> 110143AC BSS: -> 110180AC
DRAM Configuration:
Bank #0: 10000000 32 MB
Flash: 32 MB
In:  serial
Out: serial
Err: serial
OMAP1610 H2 #
```

Con u-boot instalado en la memoria flash de la tarjeta, ya no es necesario utilizar Windows o Code Composer Studio, por lo tanto debemos reiniciar la computadora en Linux.

3.2.2. Configurando U-boot

Una vez iniciado linux, debemos de asegurarnos de tener configurados los servicios de tftp y nfs (ver apéndiceA) para compartir los archivos con la tarjeta de desarrollo.

Los pasos a seguir son los siguientes:

1. Primero copiamos el archivo *uImage* a la carpeta compartida por tftp.

2. En una terminal de linux debemos teclear

   ```
   ifconfig eth0 192.168.0.1
   ```

   para cambiar la ip de la computadora a una ip fija 192.168.0.1.

3. Después abrimos gtkterm[3] y utilizamos los parámetros por defecto, cambiando la velocidad a 115200 baudios (115200-8-N-1).

[3]Software emulador de terminal para linux.

4. Ahora encendemos la tarjeta y esperamos que cargue u-boot.

5. A continuación, debemos teclear en una sola línea lo siguiente:

```
setenv bootargs 'console=ttyS0,115200n8 noinitrd rw
ip=192.168.198.10::255.255.255.0 root=/dev/nfs
nfsroot=192.168.198.1:/opt/omap/filesystem,nolock,
rsize=1024,wsize=1024,rootdelay=2'
```

   Esto configura u-boot para arrancar por nfs desde el filesystem que se creó, el cual se encuentra en este caso en la carpeta */opt/omap/filesystem*; además se le indica a u-boot que asigne una dirección ip 192.168.0.10 a la tarjeta, mientras que la dirección ip de la computadora en este caso es 192.168.0.1.

6. También debemos teclear:

```
setenv bootcmd 'tftp 10000000 uImage;bootm 10000000'
```

   En este comando estamos configurando la tarjeta para que descargue por tftp el kernel de linux a la dirección de memoria 0x10000000, que es donde empieza la memoria DRAM en la tarjeta; también estamos diciendo que arranque en la misma dirección.

7. Finalmente debemos teclear

```
saveenv
```

   para salvar la configuración en u-boot.

8. Ahora cuando reiniciemos la tarjeta, deberá arrancar automáticamente y cuando termine de cargar el sistema operativo veremos algo similar a lo siguiente:

```
VFS: Mounted root (nfs filesystem).
VFS: Mounted root (nfs filesystem).
<6>Freeing init memory: 112K
Freeing init memory: 112K
starting up local systems
```

```
Mounting a tmpfs over /dev...done.
Creating initial devicetiny nodes...done.
done
192.168.0.10 login: root
login[529]: root login on 'tts/0'
BusyBox v1.3.1 (2007-01-12 00:29:58 CST) Built-in shell (ash)
Enter 'help' for a list of built-in commands.
#
```

En este punto, ya tenemos el sistema operativo cargado y funcionando en la tarjeta de desarrollo, con lo ya es posible empezar a realizar las pruebas necesarias en la tarjeta de desarrollo.

Capítulo 4

Plataforma de Co-verificación Seamless

4.1. Uso de Seamless

Seamless [21] es un software de simulación y co-verificación que permite depurar hardware y software al mismo tiempo, gracias a que es posible ejecutar software embebido en un modelo de simulación del hardware de dicho sistema embebido.

Para la simulación del sistema embebido se optó por usar un procesador ARM926ej-s por ser el mismo utilizado en la OMAP1610. Para ésto se utilizó como sistema base una implementación completa de un sistema básico que incluye el procesador, un controlador de bus AMBA, una memoria SRAM y un emulador de terminal para desplegar datos.

4.1.1. Preparación

Antes de empezar a usar Seamless, es necesario realizar una serie de pasos para poder simular el sistema embebido:

1. Primero es necesario modificar un script de arranque para poder utilizar el simulador de software Xray para el ARM926ej-s. Este archivo se encuentra en la siguiente ruta:

   ```
   CVE_HOME/isms/bin/xray_49fpz_setup
   ```

2. Una vez localizado este archivo, es necesario comentar las líneas 88 y 89 como se muestra:

```
#     LD_ASSUME_KERNEL="2.4.1"
#     export LD_ASSUME_KERNEL
```

3. Ahora creamos una carpeta de trabajo, donde pondremos todos los archivos necesarios para trabajar. En este caso, los archivos corresponden al procesador ARM926ej-s, los cuales se encuentran en la carpeta:

   `CVE_HOME/example/arm926ejs_rev0`

   Con esto podremos modificar los scripts de arranque y configuraciones deseadas sin perder la configuración por defecto que trae Seamless.

4. Después debemos copiar los archivos que se deseen simular a la carpeta *sw*, que se encuentra dentro de nuestra carpeta de trabajo. Es importante copiar todos los archivos utilizados en la creación del binario ejecutable (archivos de arranque, archivos fuente en ensamblador y C, archivos de cabecera, etc.) para poder realizar el depuramiento del programa de manera adecuada.

5. A continuación debemos modificar los archivos *example.cve* y *example.inc* que se encuentran en la carpeta *bin* para que las rutas de los archivos a ejecutar apunten a nuestros archivos en la carpeta de trabajo, en lugar de apuntar a los archivos que se encuentran en la carpeta de Seamless.

Con esto, ya es posible empezar a trabajar en Seamless. Para poder arrancar Seamless, tecleamos lo siguiente en una terminal:

```
./runme
```

4.1.2. Configurando Seamless

Cuando arrancamos Seamless, nos aparece la ventana principal, la cuál se muestra en la Figura 4.1. Desde esta ventana podemos configurar todos los aspectos de Seamless, como la configuración de el simulador lógico, el simulador de instrucciones, los mapeos de la memoria, la configuración del profiler, etc. además de controlar la interacción entre el simulador lógico y el simulador de instrucciones.

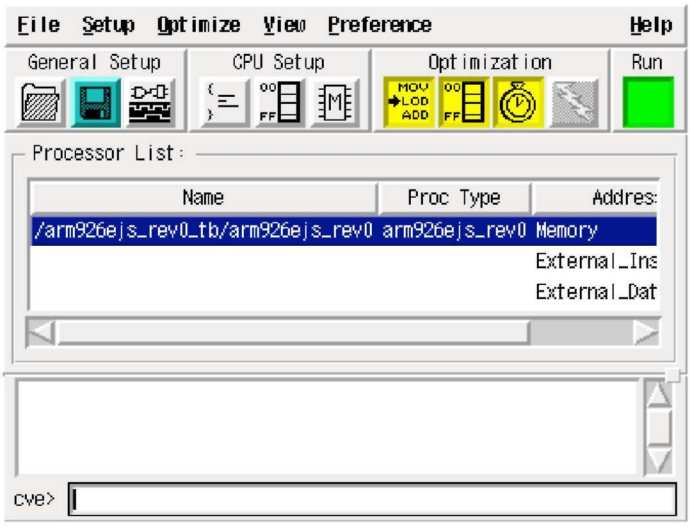


Figura 4.1: Ventana principal de Seamless.

4.1.2.1. Memoria

Para configurar los aspectos de la memoria tenemos el menú **Memory Map...** y **Memory Ranges...**, que se encuentran en el menú **setup**.

En el menú "Memory Map" (Figura 4.2) podemos seleccionar las memorias que tengamos disponibles en nuestro proyecto y "mapearlas" o asignarlas a los espacios de memoria que queramos, en dependencia de las necesidades del proyecto.

En el menú "Memory Range " (Figura 4.3) podemos cambiar parámetros de la memoria, como el tipo de acceso (hardware o software), los ciclos que toma accesar la memoria, el nombre de los espacios de memoria, entre otros.

4.1.2.2. Optimización

En Seamless, la co-verificación puede verificar la interacción entre hardware y software con un alto grado de detalle, pero esto puede resultar en un proceso muy lento. Seamless permite optimizar la co-verificación intercambiando detalle de simulación por velocidad.

Entre las optimizaciones posibles que se pueden realizar se encuentran:

- Optimización de acceso de datos.
- Optimización de instrucciones.
- Optimización de tiempo.

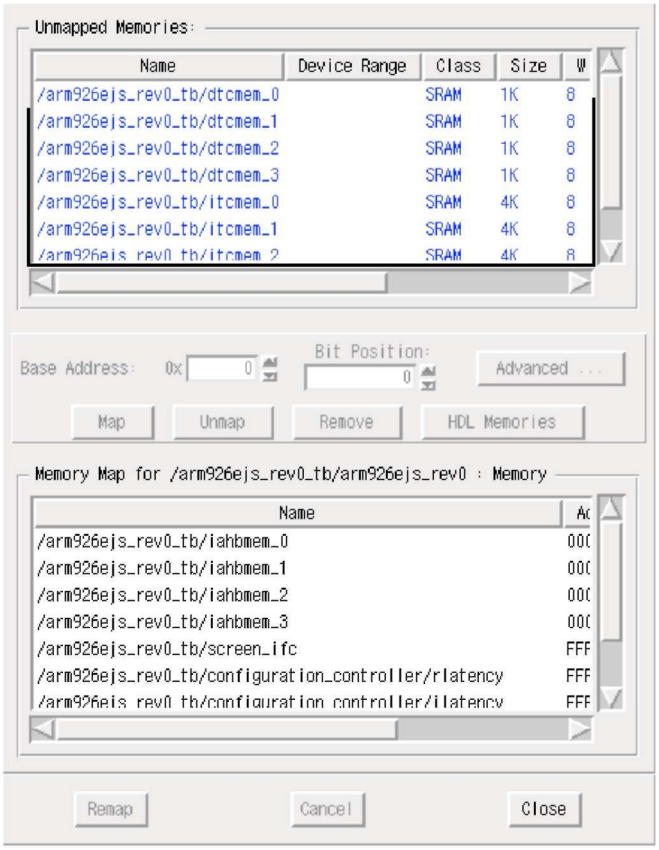


Figura 4.2: Ventana del mapa de memoria.

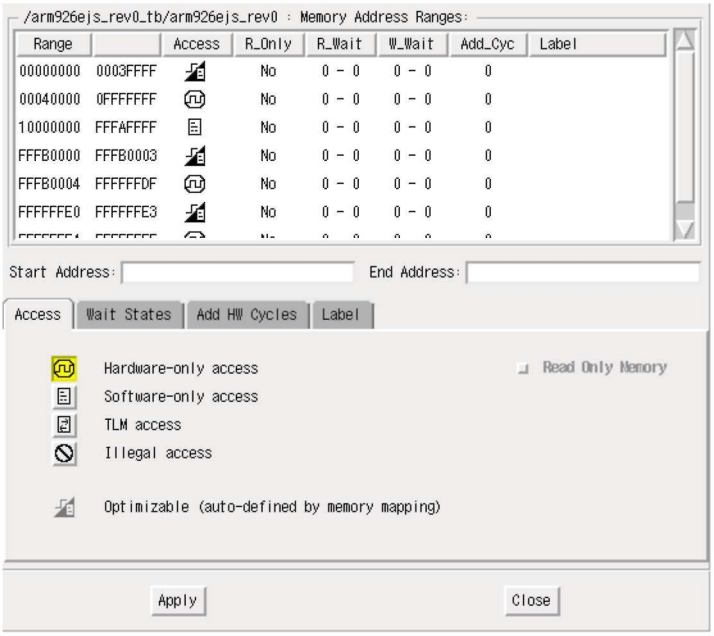


Figura 4.3: Ventana del rango de memoria.

La optimización de acceso de datos permite desabilitar la simulación de los ciclos del bus del hardware cuando se accesan direcciones de memoria optimizada. En este caso, la simulación lógica sigue avanzando, pero lo hace sin simular los accesos al bus cuando se esté accesando la memoria optimizada.

Para activar esta optimización, debemos de seleccionarla del menú **Optimize**. Una vez seleccionada, nos aparecerá una ventana similar a la de la Figura 4.4. En esta ventana debemos seleccionar los rangos de memoria que queramos optimizar, siempre y cuando ese rango de memoria sea optimizable.

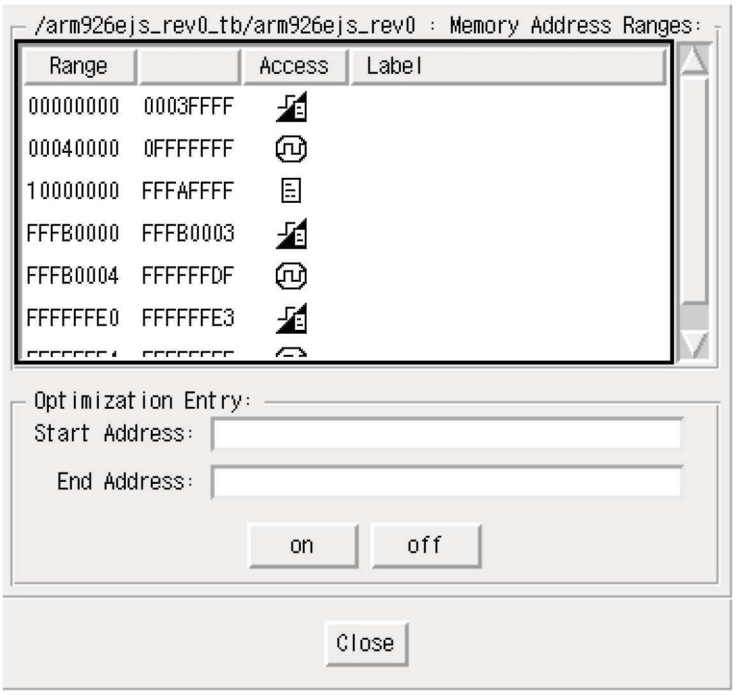


Figura 4.4: Ventana de optimización de memoria.

La optimización de instrucciones elimina la simulación de los accesos al bus para todas las instrucciones leídas (fetches) que se encuentren en memoria optimizada. esta optimización se encuentra en el menú **Optimize**.

La optimización por tiempo permite desacoplar la sincronización entre la simulación de hardware y software, permitiendo correr la simulación de software a toda velocidad. La simulación lógica avanza solamente ocasionalmente cuando haya un acceso a un área de memoria no optimizada, como por ejemplo, un dispositivo de E/S. En el menú **Optimize** seleccionamos **Time Optimization** y aparecerá una ventana como se muestra en la Figura 4.5. En esta ventana seleccionamos la segunda opción: "Only for non-optimized memory accesses." para activar la optimización de tiempo completo.

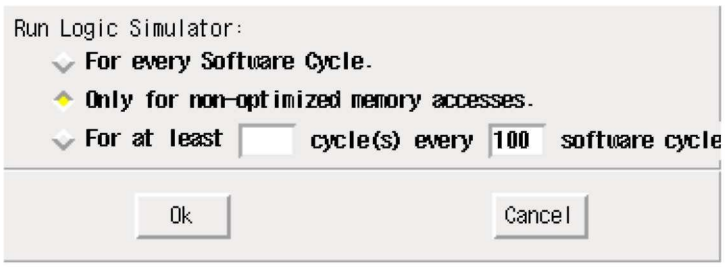


Figura 4.5: Ventana de optimización de tiempo.

4.1.2.3. Profiler

El profiler es una herramienta incluida en Seamless que nos permite determinar el desempeño de las aplicaciones que se simulan en la sesión de co-diseño. Esta herramienta nos proporciona información como el porcentaje de ejecución de cada función contenida en el programa ejecutado, el tiempo de ejecución de dicho programa, los espacios de memoria utilizados, el consumo de energía del procesador, entre otros valores.

Podemos activar el profiler desde el menú **Performance Logfile Directory**, el cuál se encuentra en el menú **Setup** en la ventana principal de Seamless. En la ventana que aparece (Figura 4.6), debemos seleccionar el directorio donde se desea almacenar la información del profiler, así como también seleccionar los tipos de datos que se deseen almacenar.

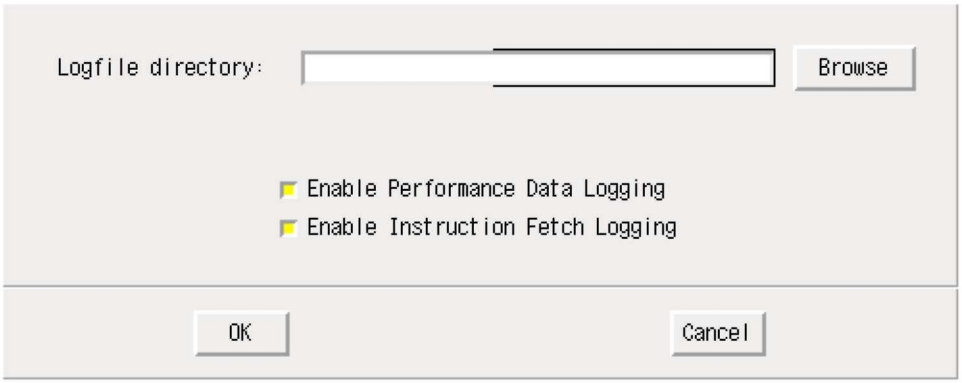


Figura 4.6: Ventana configuración del profiler.

4.1.3. Corriendo Seamless

Es importante notar que todas estas acciones de configuración son posibles de automatizar utilizando un script de arranque, el cuál nos permite inicializar todo el

sistema en un solo paso. En este proyecto ya contamos con dicho script el cuál está incluido en nuestra carpeta de trabajo, gracias a que copiamos un proyecto ya existente, como se mostró en la sección anterior.

Para correr dicho script tecleamos en una terminal lo siguiente:

```
./runme
```

Una vez que hayamos arrancado Seamless utilizando el script de arranque, nos debe aparecer un ambiente de trabajo similar al de la Figura 4.7.

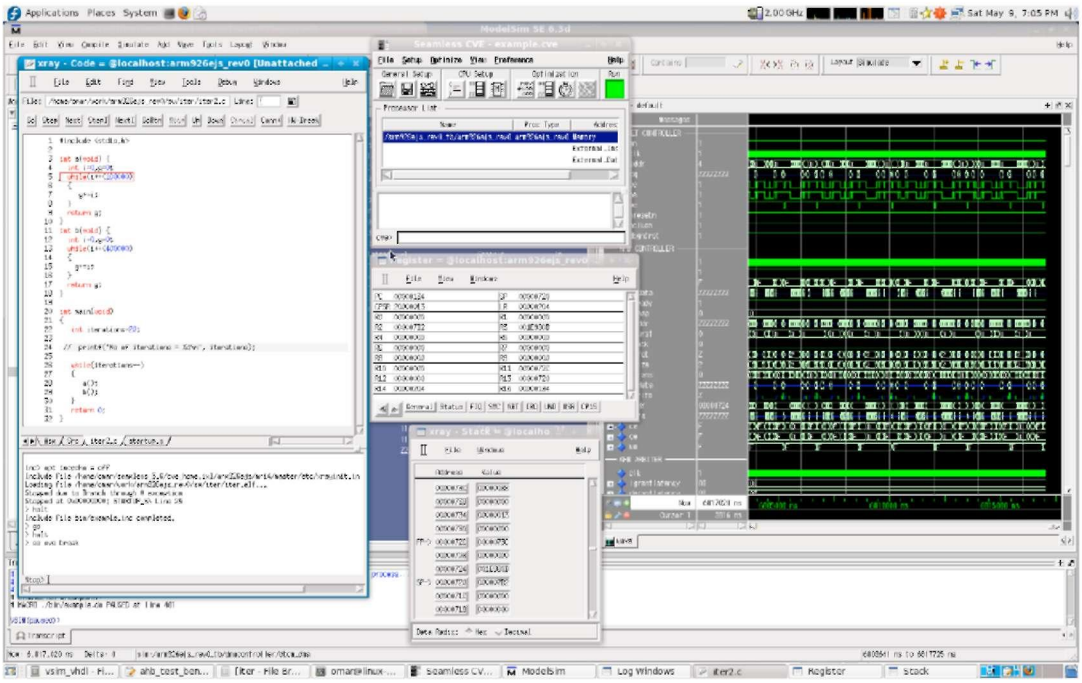


Figura 4.7: Espacio de trabajo de Seamless.

En esta Figura se pueden apreciar las ventanas del simulador de instrucciones llamado “Xray” y el simulador lógico “Modelsim”.

Cuando las herramientas terminan de cargar, el debugger arranca automáticamente nuestro programa. El debugger nos permite controlar la ejecución del código, de manera que podemos correr el programa paso a paso, ver los registros del procesador y modificar sus valores, ver la memoria del sistema, etc. además de poder interactuar con el simulador lógico directamente.

Una vez terminada la ejecución del programa, es posible ver las estadísticas generadas por este, como por ejemplo, el número de instrucciones ejecutadas, el número de

ciclos de reloj transcurridos, el número de accesos a cache, etc. tecleando el comando *statistics* en la ventana principal de Xray (Figura 4.8).

Figura 4.8: Ventana principal del simulador de instrucciones Xray.

En el caso del simulador lógico, no es necesario configurar o modificar los parámetros por defecto, ya que este funciona de manera automática, en conjunto con el simulador de instrucciones.

4.2. Implementación multiprocesador

Además de implementar hardware similar al usado en la OMAP, se realizó también una implementación en software de dos procesadores funcionando concurrentemente, utilizando una memoria de doble puerto para la intercomunicación entre ambos (Figura 4.9).

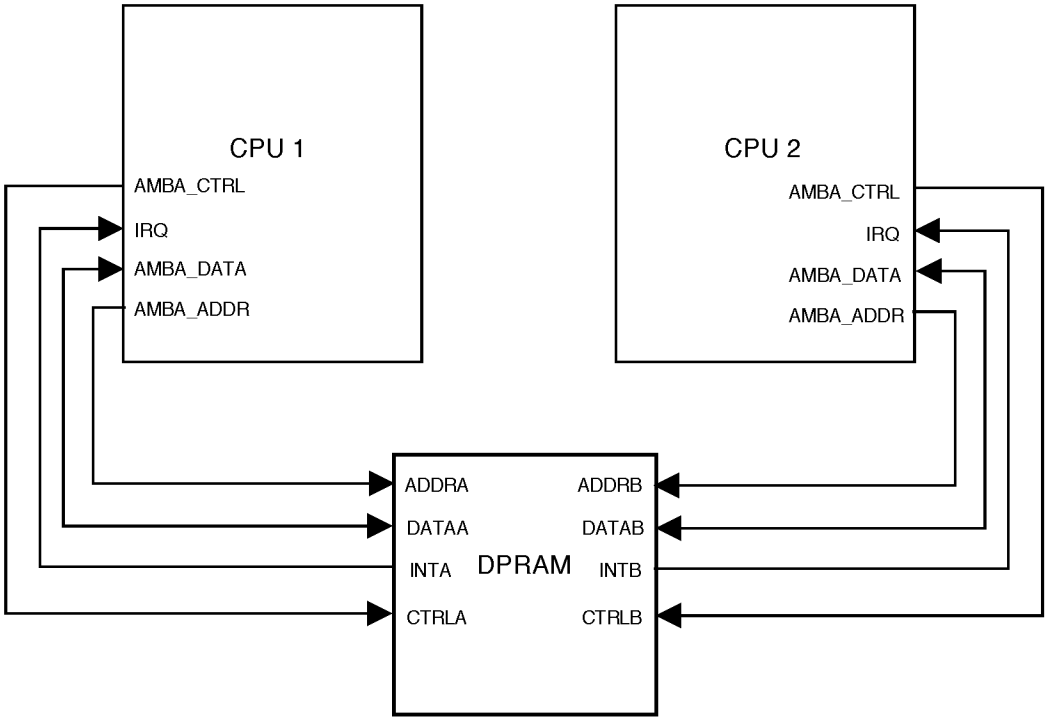


Figura 4.9: Interconección de los procesadores a través de la memoria de doble puerto .

Dicha memoria cuenta con un sistema de mensajes (Cuadro 4.1) que le permite a los procesadores saber cuando hay un dato entrante por parte del otro procesador sin necesidad de tener que leer la memoria en intervalos regulares de tiempo (polling).

Este sistema de mensajes funciona de la siguiente manera:

1. Primero se especifican las direcciones de memoria que serán usadas para el sistema de mensajes (mailboxes). Dichas direcciones deben estar dentro del rango de memoria instanciado y deben encontrarse en espacios contiguos (Figura 4.10).

2. Cuando un dato es recibido en la dirección destinada a un "mailbox", se genera una señal de interrupción en el procesador correspondiente a dicho mailbox (Figura 4.11).

3. Después de la interrupción, es necesario que el procesador que fue interrumpido lea su "mailbox" para poder deshabilitar la bandera de interrupción y con esto poder regresar al programa que se estaba ejecutando anteriormente (Figura 4.12).

   En la Figura 4.11 vemos que se declara una variable tipo apuntador que contiene la dirección donde se encuentra el mailbox del procesador que queremos interrumpir

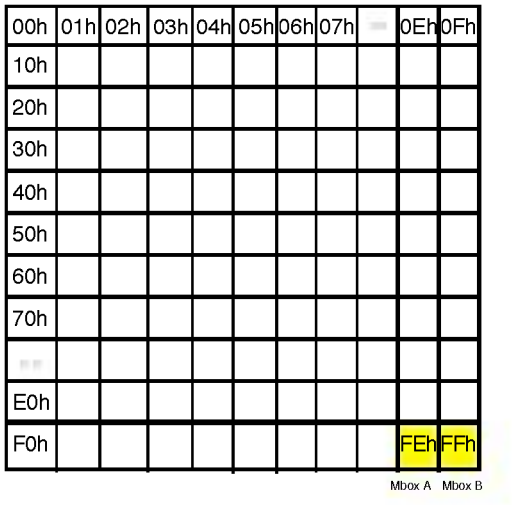


Figura 4.10: Organización de la memoria y mailboxes.

(mailbox A). Después, al escribir un valor en dicha variable, se genera una señal de interrupción en el pin IntA del procesador A.

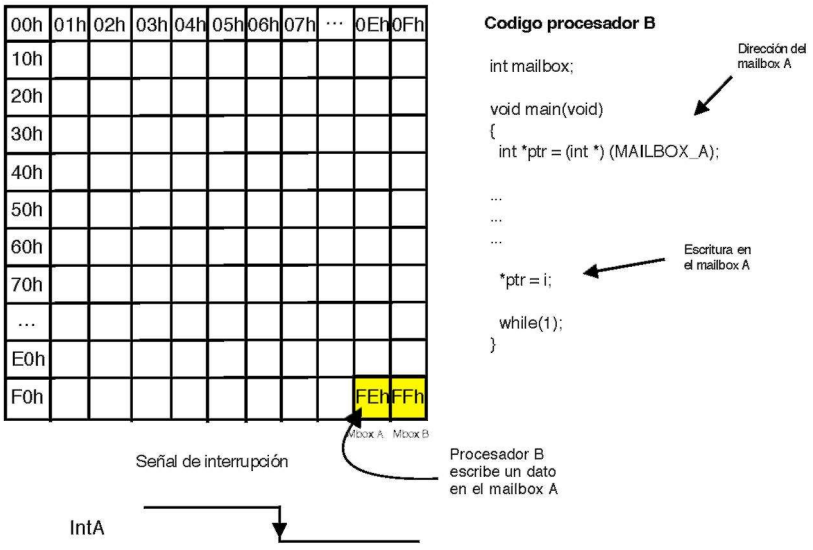


Figura 4.11: Generación de la señal de interrupción.

En la Figura 4.12 vemos que cuando el procesador entra en su rutina de interrupción se realiza la lectura del mailbox a una variable. Con esto, la señal de interrupción regresa a su estado original. Si no se realizara una operación de lectura en el mailbox, el estado de la señal de interrupción no regresaría a su estado original.

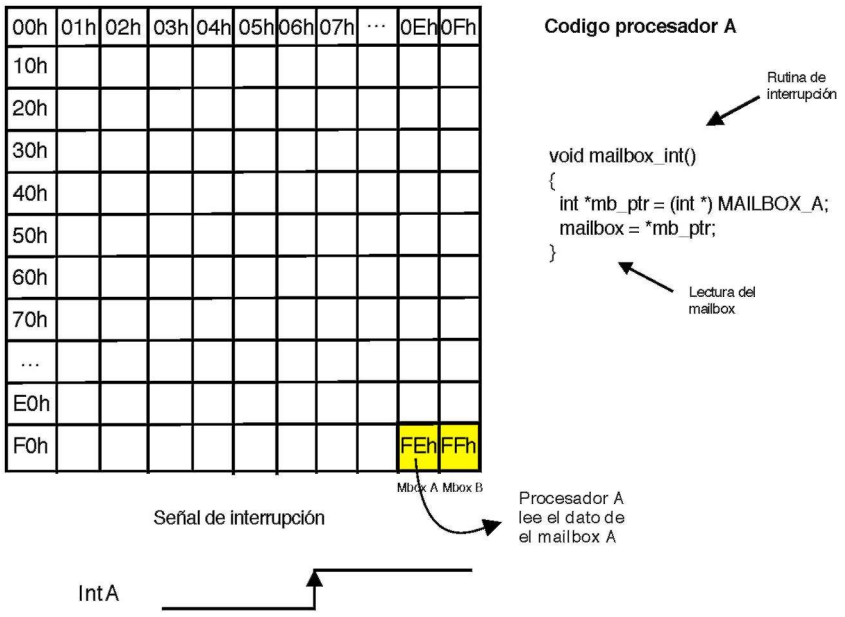


Figura 4.12: Ejecución de la interrupción.

Además del sistema de mensajes, este modelo incluye lógica para evitar que ambos procesadores accedan a la misma dirección de memoria al mismo tiempo. Esto se logra comparando los buses de datos de ambos puertos para determinar si se están realizando accesos de memoria en la misma localidad al mismo tiempo. En caso de que esto suceda, el primer procesador en acceder a la memoria es el que tiene prioridad sobre el acceso, pudiendo éste escribir o leer según se requiera, mientras que la petición de acceso del otro procesador es ignorada (Cuadros 4.2 y 4.3). Es responsabilidad del programador verificar si un acceso a memoria fue exitoso. Para esto se pueden utilizar diferentes métodos, como por ejemplo semáforos, mutexes, etc.

El Cuadro 4.1 muestra la interacción de los procesadores al utilizar el sistema de mensajes. Dicho cuadro está dividido en tres columnas las cuales representan el procesador A, la memoria de doble puerto y el procesador B, respectivamente. La numeración a la izquierda de las columnas representa los pasos requeridos para el funcionamiento normal del sistema de mensajes.

Tabla 4.1: Funcionamiento del sistema de mensajes

	ARM A	**DPRAM**	**ARM B**
1	Se hace una operación de escritura en el mailbox del procesador B. *(0xFF)*		
2		Se escribe en la dirección de memoria solicitada por el procesador A (mailbox B.) *(0xFF)* Se genera una señal de interrupción en el procesador B.	
3			El procesador B entra en su rutina de interrupción.
4			El procesador B realiza una operación de lectura de su mailbox. *(0xFF)*
5		Se lee la dirección de memoria solicitada por el procesador A (mailbox B). *(0xFF)* Se restaura la señal de interrupción del procesador B.	
6			El procesador B regresa de la rutina de interrupción al programa principal.

En el Cuadro 4.2 se muestra una operación normal en una memoria de doble puerto. Como se puede ver, a pesar de que los dos procesadores acceden a la memoria al mismo tiempo, no ocurre problema alguno siempre y cuando las direcciones de memoria

accesadas sean diferentes entre sí.

Tabla 4.2: Funcionamiento normal de la memoria compartida.

	ARM A	**DPRAM**	**ARM B**
1	Se hace un acceso a la memoria compartida. *(0x00)*		Se hace un acceso a la memoria compartida. *(0x01)*
2		Se accede a la dirección de memoria solicitada por el procesador A. *(0x00)* Se accede a la dirección de memoria solicitada por el procesador B. *(0x01)*	
3	Termina la transacción sin problemas.		Termina la transacción sin problemas.

En caso de que ambos procesadores realicen un acceso a la misma dirección de memoria al mismo tiempo, como se muestra en el Cuadro 4.3, solamente un procesador puede completar la operación normalmente, mientras que la operación del otro procesador es ignorada.

Tabla 4.3: Conflicto de acceso a la memoria compartida.

	ARM A	**DPRAM**	**ARM B**
1	Se hace un acceso a la memoria compartida. *(0x00)*		Se hace un acceso a la memoria compartida. *(0x00)*
2		Se accede a la dirección de memoria solicitada por el procesador A. *(0x00)* Se ignora el acceso a la dirección de memoria solicitada por el procesador B. *(0x00)*	
3	Termina la transacción sin problemas.		Hay problemas debido a que no se accedió a la dirección de memoria solicitada por el procesador B.

Capítulo 5

Metodología y resultados

5.1. Metodología

La Figura 5.1 muestra la metodología seguida en este trabajo y se explica a continuación. El trabajo de esta tesis esta dividido en varios objetivos. Uno de los objetivos era obtener mediciones del desempeño del procesador OMAP para usar como referencia para validar los resultados obtenidos con Seamless. Otro de los objetivos era implementar un sistema multiprocesador y encontrar una forma de intercomunicarlos. Para esto se decidió utilizar un esquema de mensajes a través de mailboxes, utilizando para su implementación una memoria compartida de doble puerto, la cual puede ser accesada por ambos procesadores al mismo tiempo. Después se debían realizar pruebas de desempeño de este sistema para su comparación con un sistema de un solo procesador. Como continuación a futuro de este trabajo se podría desarrollar un BIOS para facilitar la intercomunicación entre procesadores por medio de drivers y librerías que podrían ser utilizados en conjunto con un sistema operativo.

5.1.1. Plataforma de trabajo OMAP

La plataforma OMAP fue utilizada como referencia para las pruebas comparativas realizadas en esta tesis.

5.1.1.1. Implementación de un sistema operativo Linux

Para poder utilizar ésta plataforma de trabajo, fue necesario usar un sistema operativo Linux, el cual consta de un kernel Linux, un sistema de archivos y librerías

y herramientas de uso libre.

5.1.1.2. Mediciones

Las mediciones realizadas fueron medidas con una herramienta de perfilación llamada "Oprof", la cual sirve para determinar los porcentajes de ejecución de las diferentes funciones utilizadas por los programas de prueba. Dichos programas fueron realizados de manera que pudieran ser utilizados en la plataforma Seamless sin problemas. En la Figura 5.2 vemos el procedimiento que se siguió para realizar las pruebas de desempeño.

5.1.2. Comparaciones

Una vez obtenidos los resultados, se realizaron comparaciones de desempeño entre la plataforma OMAP y la plataforma Seamless para validar que ambas plataformas son similares y para determinar el "overhead" que incurre la ejecución de un programa haciendo uso de un sistema operativo (OMAP) con respecto a la ejecución de dicho programa sin él (Seamless).

5.1.3. Plataforma de co-verificación Seamless

Para poder realizar pruebas comparativas contra el procesador utilizado en la OMAP, se utilizó la plataforma de co-verificación Seamless. Además se utilizó dicha plataforma para la implementación de un sistema multiprocesador.

5.1.3.1. Implementación de un sistema con un procesador

La plataforma de desarrollo simulada en Seamless consiste en una implementación en VHDL de un procesador ARM926ej-s, así como también la implementación de 4 memorias, un controlador de interrupciones, un generador de reloj, un controlador de reset y un controlador de bus AMBA.

5.1.3.2. Implementación de un sistema multiprocesador

Para implementar el esquema de comunicación entre procesos entre los dos procesadores se eligió utilizar un sistema de mensajes. Para esto, se optó por el uso de una memoria compartida multipuerto, la cual puede ser accesada por ambos procesadores al

mismo tiempo. Esta memoria cuenta con dos puertos de entrada-salida independientes además de contar con un sistema de mailboxes que generan una señal de interrupción al ser accesados por los procesadores en direcciones específicas.

Para la implementación de los dos procesadores se utilizaron dos instancias iguales basadas en la implementación del sistema con un solo procesador. Para comunicarse con la memoria compartida se conectaron los buses de alta velocidad de ambos procesadores a cada uno de los puertos de la memoria compartida, además de conectar las señales de interrupción de un procesador a otro. Utilizando la memoria multipuerto, fue posible intercomunicar los dos procesadores para propósitos de sincronización y de ejecución de código de manera concurrente.

5.1.3.3. Mediciones

Se realizaron mediciones para un procesador y para dos procesadores. En la Figura 5.3 vemos el procedimiento seguido para realizar las mediciones. En el caso de la medición para un procesador, se ejecutaron varios programas sencillos, de manera que también pudieran ser corridos en la tarjeta de desarrollo OMAP. Para la medición de los dos procesadores se utilizó un benchmark que es fácilmente paralelizable, y que lejos de ser una paralelización ideal demuestra la mejora en tiempo de ejecución utilizando un sistema con múltiples procesadores con respecto a un sistema con un solo procesador.

5.1.4. Conclusiones

En base a los resultados obtenidos se determinó que la implementación de un sistema de desarrollo utilizando la plataforma Seamless es similar a la implementación en un sistema real (plataforma OMAP), por tanto es posible utilizar esta plataforma para desarrollar un sistema multiprocesador con la expectación de obtener resultados similares a los que se pudieran obtener en un sistema real con dichas características.

También fue posible implementar la intercomunicación entre dos procesadores utilizando un esquema de comunicación entre procesos usando mailboxes implementado en una memoria compartida.

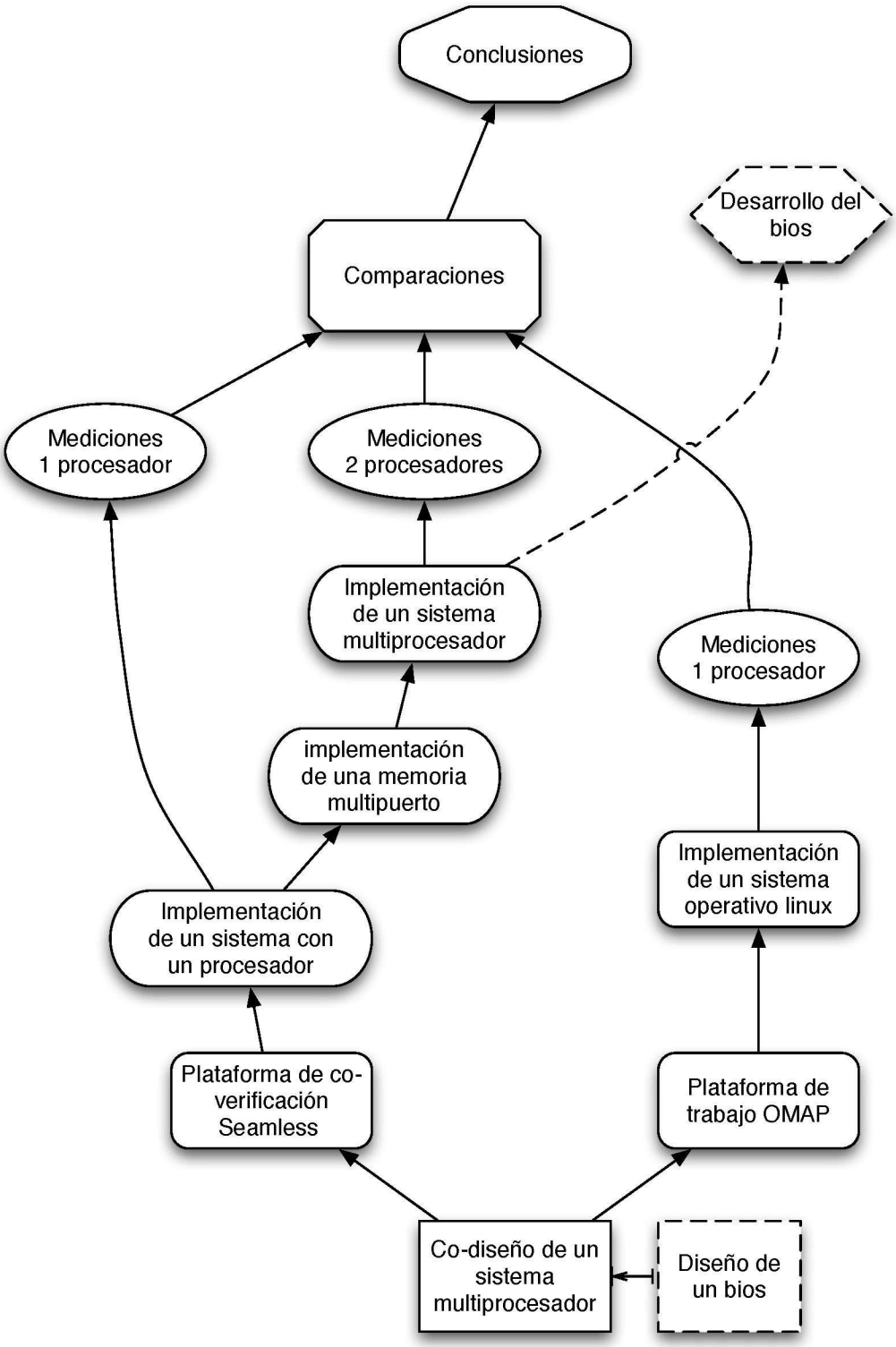


Figura 5.1: Metodología seguida.

Para la sesión de mediciones se debe iniciar la comunicación entre la tarjeta de desarrollo y la computadora y arrancar la herramienta de perfilación, como se muestra en la Figura 5.2. Una vez hecho esto, solamente es necesario correr el programa que se desee perfilar, detener la herramienta de perfilación y observar los reportes que fueron generados.

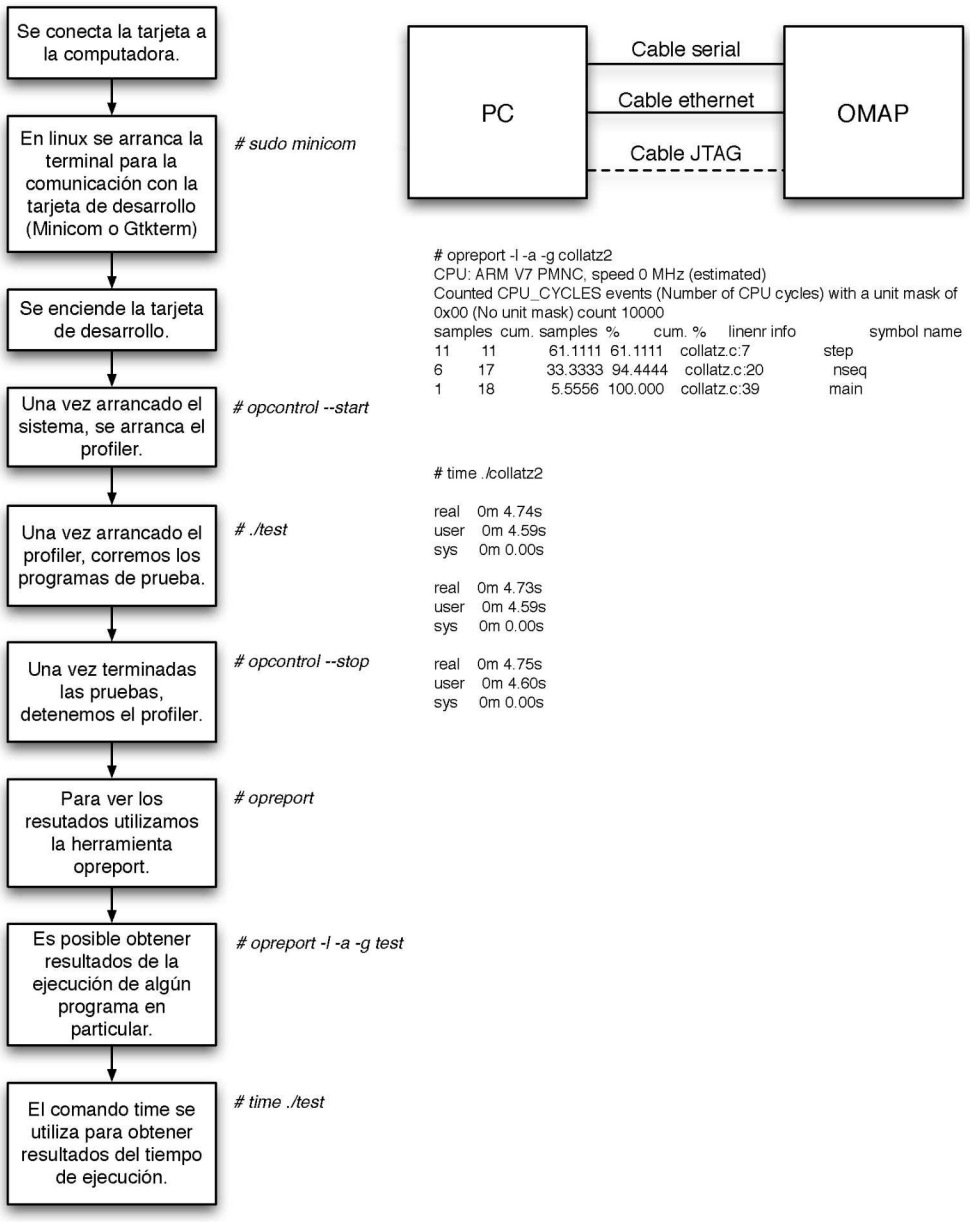


Figura 5.2: Sesión de mediciones de la OMAP.

Para la sesión de mediciones en Seamless preferentemente corremos un script que contenga todos los parámetros necesarios para la configuración y ejecución del código de manera automática, esto con el fin de agilizar y automatizar el proceso de perfilación. Después de haber terminado la ejecución, podemos observar los resultados corriendo la herramienta de perfilado (profiler).

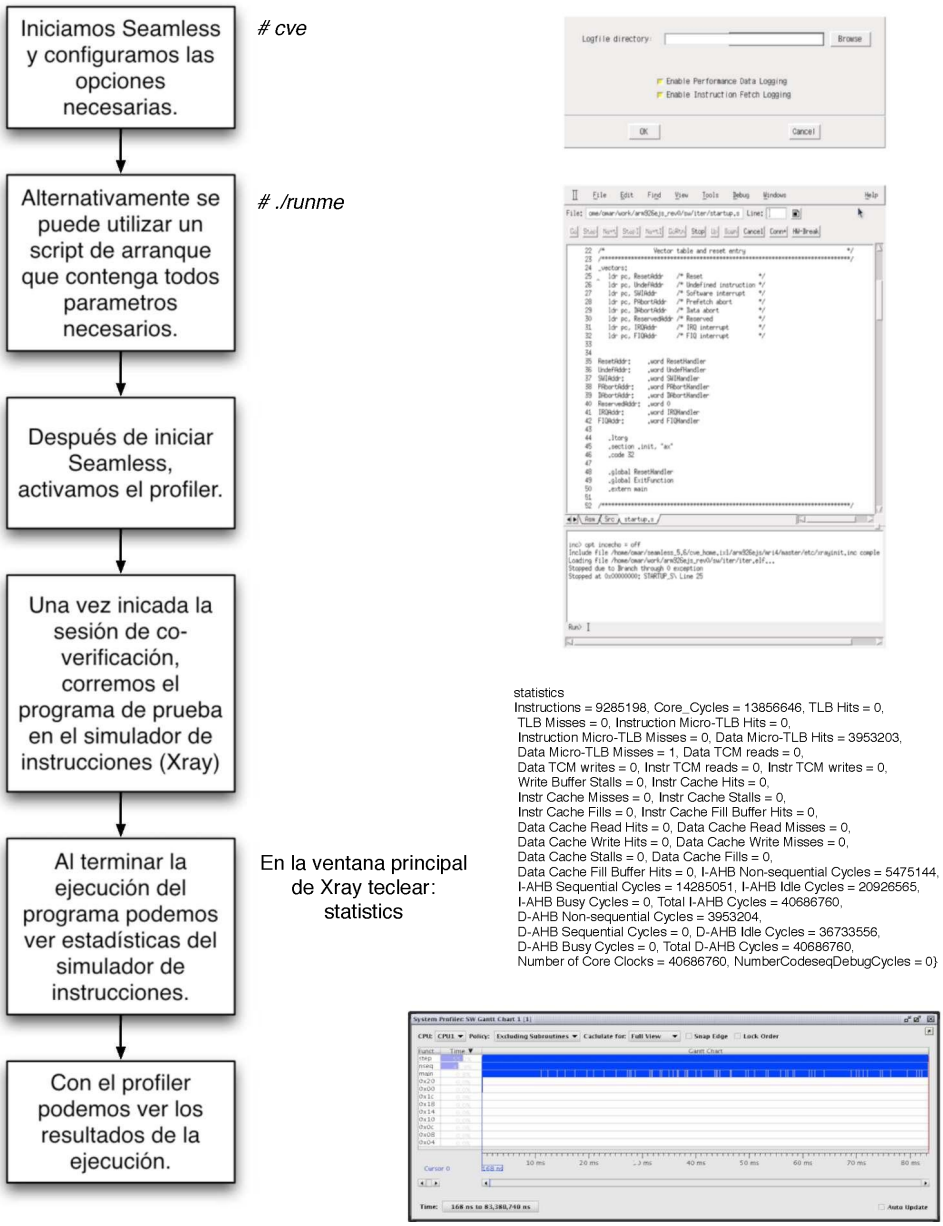


Figura 5.3: Sesión de mediciones en Seamless.

5.2. Herramientas para perfilamiento y medición de tiempos de ejecución

El perfilador OProfile es una herramienta de monitoreo de desempeño de todo el sistema. Usa el hardware de monitoreo de desempeño del propio procesador para determinar la información necesaria del kernel y de los ejecutables en el sistema, tal como cuando la memoria es referenciada, el número de solicitudes a caché, el número de interrupciones, entre otros. Muchos procesadores incluyen hardware de monitoreo dedicado. Este hardware hace posible determinar cuando suceden ciertos eventos. Normalmente este hardware consta de contadores que son incrementados cada vez que ocurre un evento. Cuando el contador se desborda, una interrupción es generada, haciendo posible controlar la cantidad de detalle producido por los monitores de desempeño.

Debido a que OProfile es una herramienta de perfilación estadística, solamente es posible determinar el porcentaje de ejecución de cada función, con respecto al número de muestras tomadas. Esto quiere decir que esta herramienta no puede determinar el tiempo de ejecución de las funciones ó el tiempo total de ejecución del programa, solamente puede determinar de manera porcentual la ejecución de las funciones con respecto a la ejecución total del programa, independientemente del tiempo que tarde dicha ejecución.

Debido a problemas con el software de perfilación, no fue posible determinar los porcentajes de ejecución en la OMAP1610, por tanto se utilizó una tarjeta de desarrollo más nueva que cuenta con un procesador OMAP3530 el cuál cuenta con un GPP ARM Cortex A8, que es compatible a nivel instrucción con el ARM926ej-s. Las mediciones de los tiempos de ejecución sin embargo. sí fueron obtenidas de la OMAP1610.

Gracias a la naturaleza de OProfile, es posible utilizar las estadísticas obtenidas del Cortex A8 y compararlas con el ARM926ej-s simulado en Seamless. Para poder estimar el tiempo total de ejecución de un programa, se utilizó una función del sistema operativo llamada *time*. Esta función permite estimar el tiempo total de ejecución de un programa, así como también estimar cuanto de este tiempo ocurrió en espacio de usuario y de sistema.

Para las pruebas de desempeño en el ambiente de simulación, se utilizó un programa llamado "Profiler" incluido en Seamless. Este programa permite determinar el tiempo de ejecución de cada función, así como también el tiempo total de ejecución de todo el ejecutable.

5.3. Benchmarks

Los programas de prueba utilizados para las mediciones de desempeño para un solo procesador son programas sencillos cuyo propósito es solamente el de poner a trabajar al procesador para determinar los tiempos y porcentajes de ejecución de procesadores similares para observar las diferencias entre ejecutar un programa usando un sistema operativo y ejecutarlo sin el.

El primer programa de prueba se llama "Collatz", el cual calcula la longitud de secuencias de Collatz para n elementos. Este programa consta de tres funciones: La función *step*, la cual se encarga del cálculo de la conjetura de Collatz. La función *nstep* se encarga de decidir si el valor de entrada a esta función debe de ser enviado a la función *step* o no. Finalmente, la función *main* se encarga de repetir este proceso por n veces de forma ascendente, empezando desde 1.

El segundo programa de prueba llamado "Iter" consta de dos funciones llamadas "a" y "b", las cuales solamente realizan una suma de manera iterativa por 100000 y 400000 veces respectivamente. Debido a la sencillez de este código, es muy fácil determinar el porcentaje de ejecución de cada función, de manera que es posible así, comprobar si el profiler funciona adecuadamente. Debido a que las dos funciones son iguales, es posible determinar que la función "a" tomará aproximadamente 20 % del tiempo, mientras que "b" tomará 80 %.

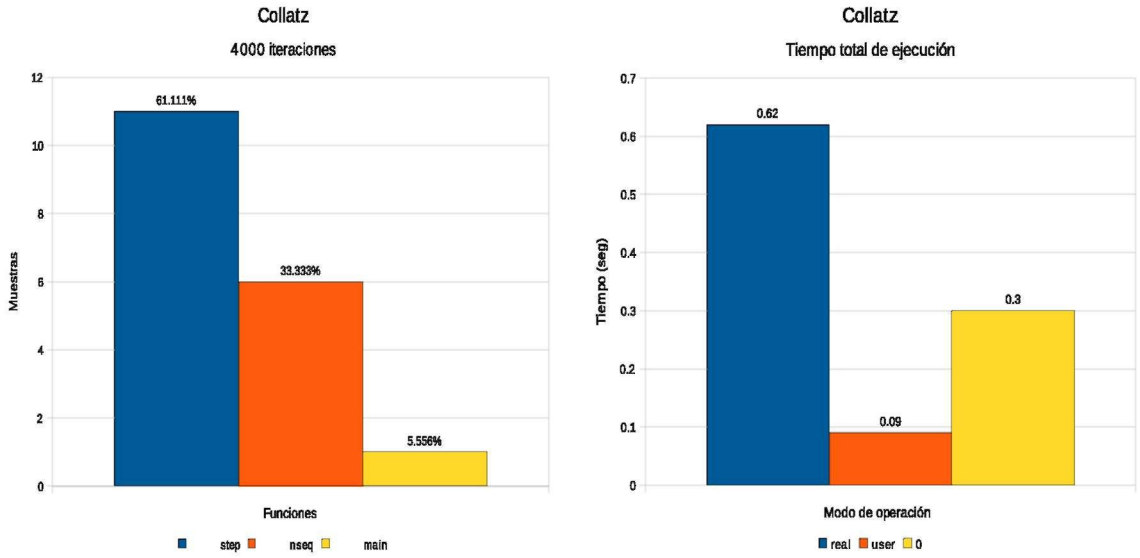
El tercer programa de prueba llamado "Prime" incrementa un contador y verifica si el número de dicho contador es un número primo. Para esto, realizamos la operación $n \ mod \ m$, donde n es el valor de la iteración actual y m es el valor de un contador que empieza de 2 y se incrementa hasta llegar al valor de n. Cuando dicha operación resulta en 0, el número calculado es un número primo.

Para el benchmark usado en la implementación de los dos procesadores se uso un benchmark comúnmente utilizado para sistemas paralelos llamado "Livermore loops" el cual consiste en una serie de 24 subrutinas llamadas "kernels" algunas de las cuales están diseñadas para ser vectorizadas y algunas no.

5.4. Resultados usando un solo procesador

5.4.1. Linux

El primer programa de prueba es "Collatz". En el primer caso, la cantidad de elementos es de 4000. En la figura 5.4a vemos que la mayor parte del tiempo el programa se encuentra ejecutando la función *step*. En la figura 5.4b vemos que el tiempo real de ejecución es mayor a la suma del tiempo de ejecución en modo de usuario y en modo de sistema (kernel), debido al overhead del sistema operativo.

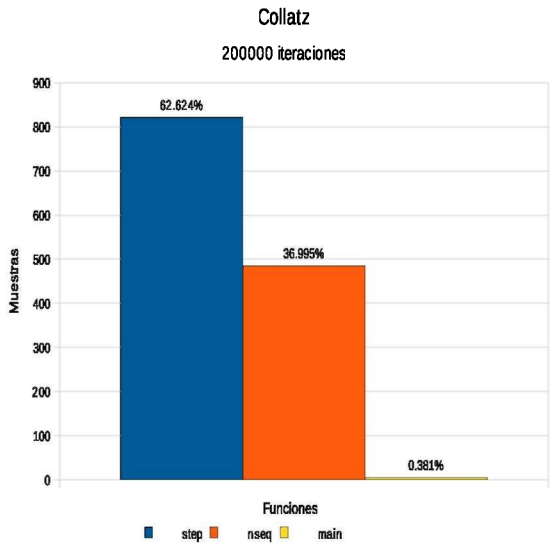


(a) Porcentaje del tiempo de ejecución.

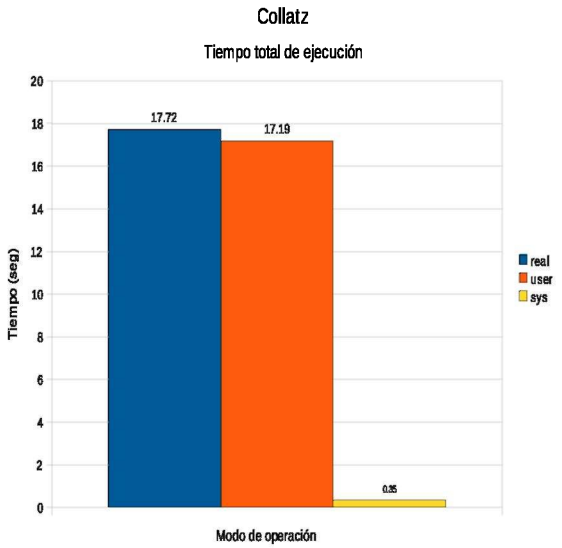
(b) Tiempo total de ejecución del programa

Figura 5.4: Programa Collatz para 4000 iteraciones

Ahora tenemos un caso con 200000 iteraciones. Como se puede apreciar en la figura 5.5a, los porcentajes de ejecución han cambiado con respecto al caso anterior. Sobre todo es notorio ver que el porcentaje de ejecución de la función *main* ha disminuido a un valor muy pequeño, debido a que las otras funciones pasan un tiempo cada vez mayor en ejecución, conforme aumentan las iteraciones. Ahora tenemos el tiempo de ejecución de este caso, el cual podemos ver en la figura 5.5b.



(a) Porcentaje del tiempo de ejecución.

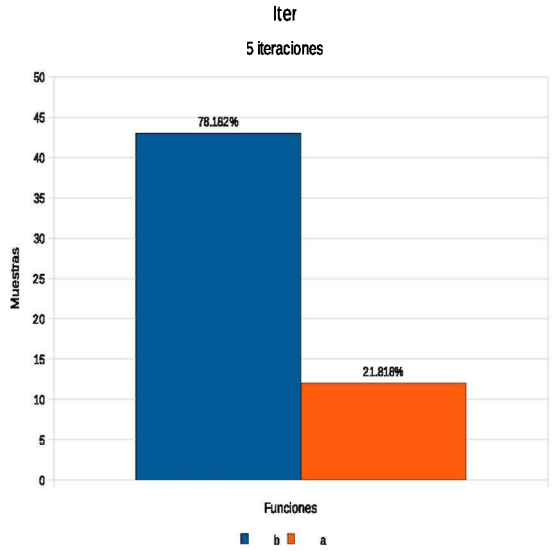


(b) Tiempo total de ejecución del programa

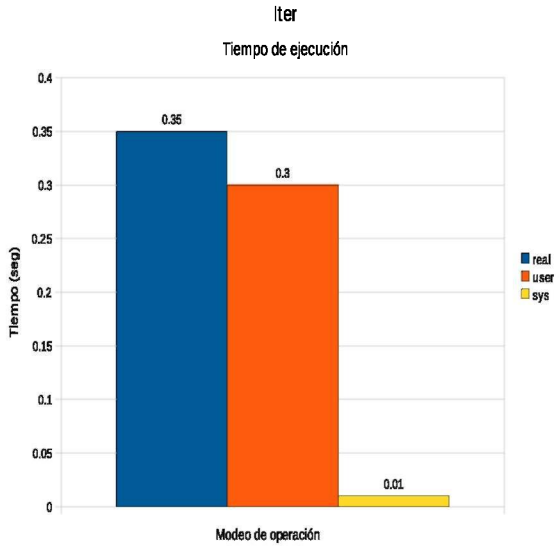
Figura 5.5: Programa Collatz para 200000 iteraciones

El siguiente programa de prueba es "Iter". En el primer caso, las funciones en *main* se repiten por 5 veces. Como se aprecia en la figura 5.6a, vemos que en efecto, los porcentajes de ejecución de las funciones son similares a los esperados. El tiempo de ejecución se muestra en la figura 5.6b. Al igual que en el programa anterior, vemos que el tiempo total de ejecución es un poco mayor que la suma del tiempo de ejecución en modo de usuario y en modo de sistema.

Ahora tenemos el caso de 20 iteraciones. De nuevo, en la figura 5.7a apreciamos que el resultado la función b es aproximadamente 80 % y la función a es de aproximadamente 20 %. El tiempo total de ejecución mostrado en la figura 5.7b es muy similar al tiempo de ejecución en modo de usuario.

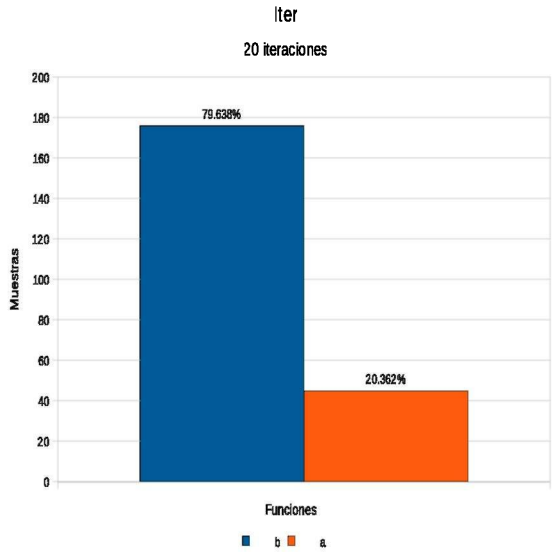


(a) Porcentaje del tiempo de ejecución.

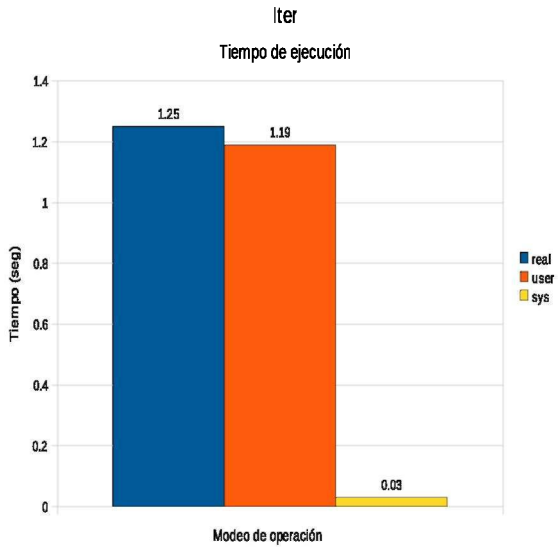


(b) Tiempo total de ejecución del programa

Figura 5.6: Programa Iter para 5 iteraciones



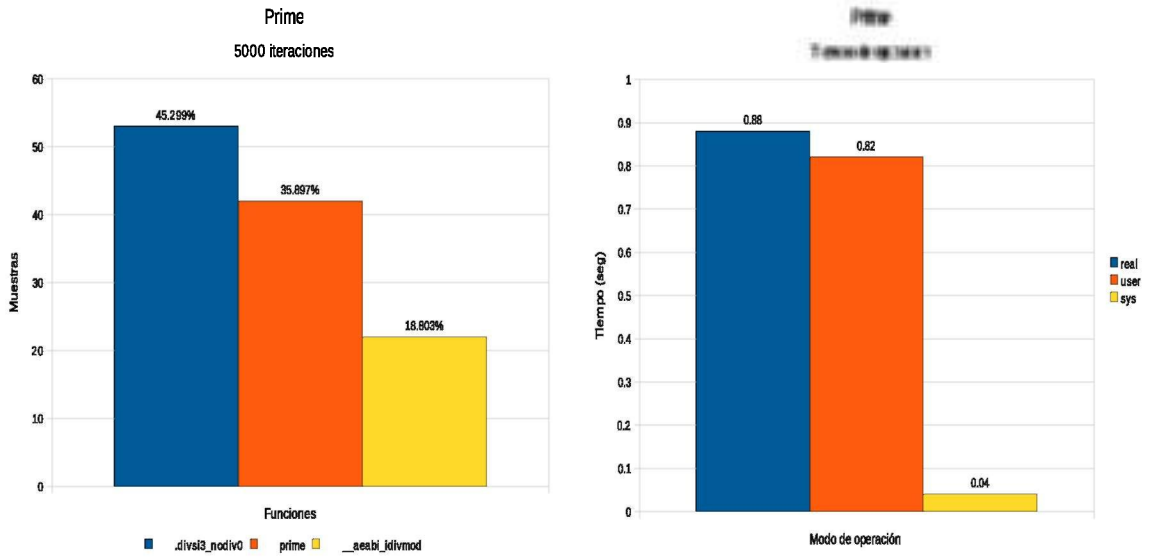
(a) Porcentaje del tiempo de ejecución.



(b) Tiempo total de ejecución del programa

Figura 5.7: Programa Iter para 20 iteraciones

El tercer programa de prueba es “Prime”. En este único caso, el contador llegará hasta las 5000 iteraciones. Como se muestra en la figura 5.8a, vemos que se están ejecutando funciones que no aparecen en el programa original. Estas funciones son las encargadas de calcular los módulos, y son funciones internas de las librerías utilizadas por el programa. En la figura 5.8b vemos el tiempo total de ejecución.



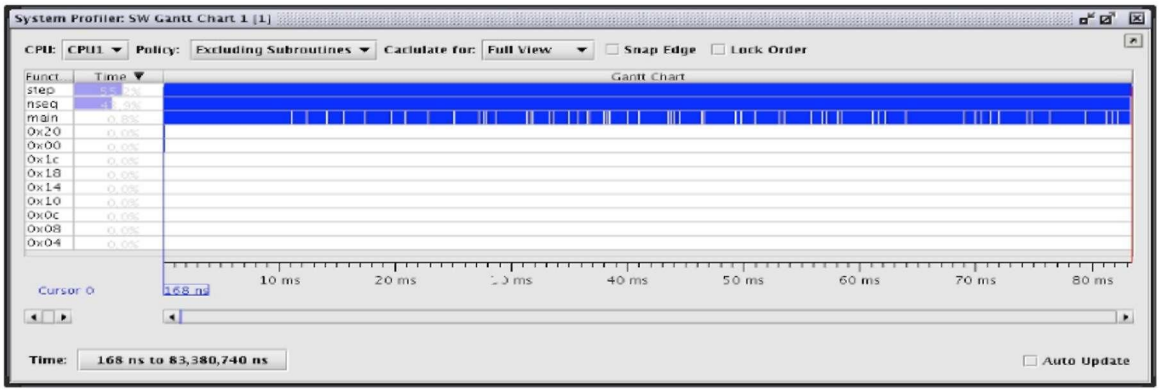
(a) Porcentaje del tiempo de ejecución.

(b) Tiempo total de ejecución del programa

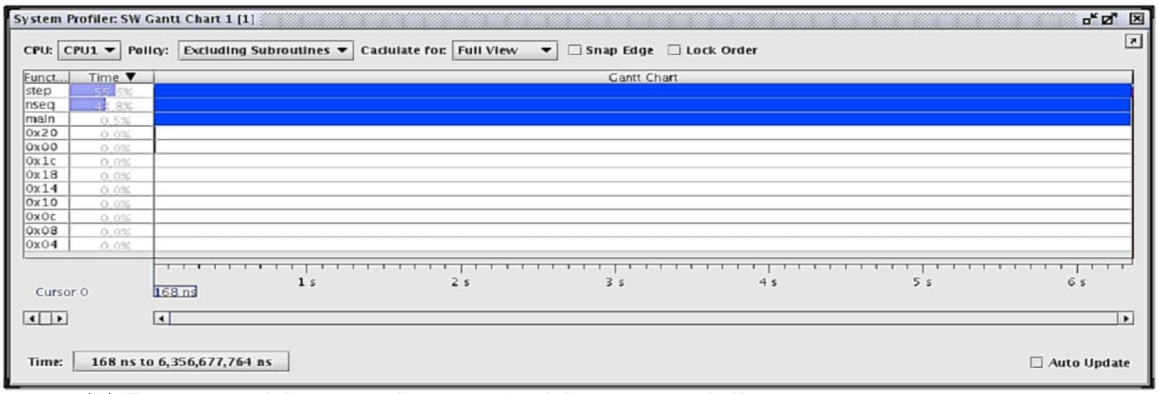
Figura 5.8: Programa Prime para 5000 iteraciones

5.4.2. Seamless

Ahora analizaremos el desempeño de los mismos programas utilizando Seamless. Primero empezaremos con el programa "Collatz" utilizando 4000 iteraciones. El resultado se muestra en la Figura 5.9a. Para el caso de 200000 iteraciones vemos el resultado en la Figura 5.9b. En estos resultados vemos que para 4000 iteraciones, la función *step* se ejecuta en un 55.6 %, la función *nseq* 43.8 %, la función *main* 0.8 % y el 0.1 % restante le corresponde a las funciones de inicialización del procesador. El tiempo de ejecución es de 83.38 ms. Para el caso de 200000 iteraciones, vemos que la función *step* se ejecuta en un 55.2 %, la función *nseq* 43.9 %, la función *main* 0.5 % y el 0.1 % restante le corresponde a las funciones de inicialización del procesador. El tiempo de ejecución es de 6.35 s.



(a) Porcentaje del tiempo de ejecución del programa Collatz para 4000 iteraciones.



(b) Porcentaje del tiempo de ejecución del programa Collatz para 200000 iteraciones.

Figura 5.9: Programa Collatz

Una ventaja del profiler de Seamless es que nos permite ver el patrón de ejecución del programa, así como también, el porcentaje de ejecución del programa en cualquier

segmento de tiempo, como se muestra en la figura 5.10. Arriba tenemos las primeras 15 iteraciones del programa, mientras que abajo tenemos las últimas 15. En este programa podemos notar que conforme pasa el tiempo, el programa tarda más en ejecutar las funciones *step* y *nseq*, mientras que *main* sigue tardando el mismo tiempo.

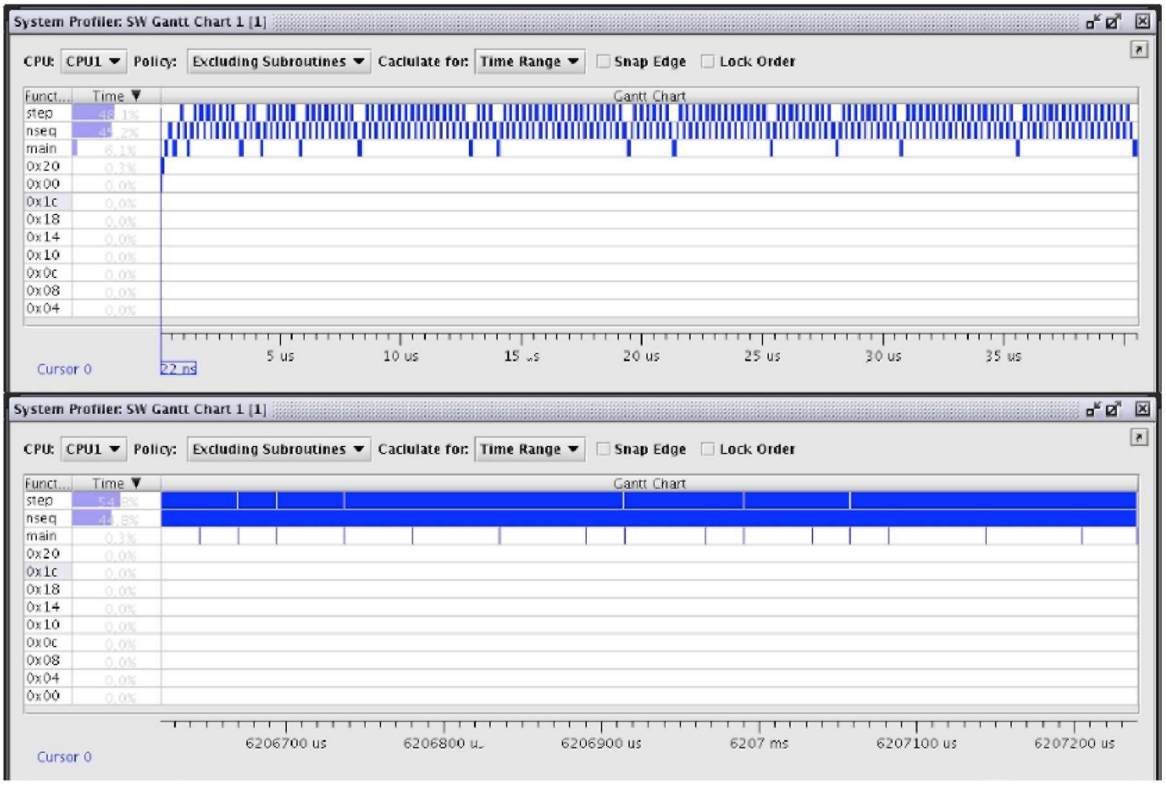
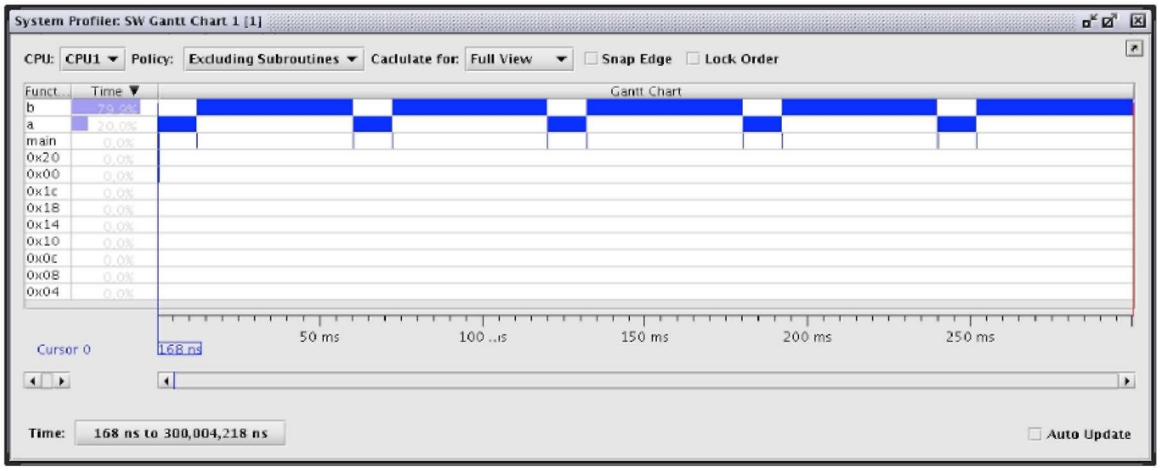
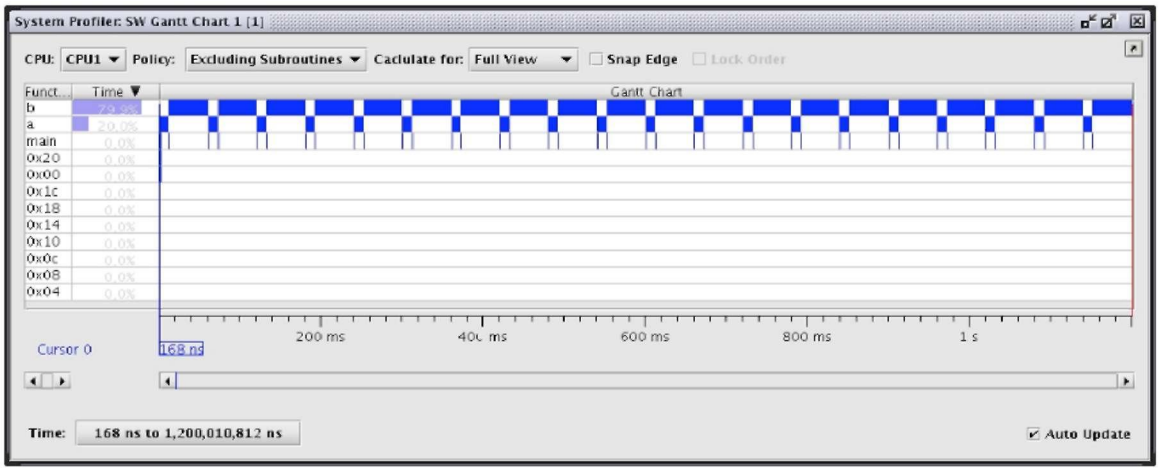


Figura 5.10: Porcentaje del tiempo de ejecución del programa Collatz en diferentes segmentos de tiempo.

En la Figura 5.11a tenemos los resultados del programa "Iter" utilizando 5 iteraciones. Después tenemos el caso para 20 iteraciones en la Figura 5.11b. En este programa se puede notar que el porcentaje de ejecución de cada función no varia conforme pasa el tiempo, a diferencia del programa anterior debido a que los tiempos de ejecución de las funciones a y b son independientes de la cantidad de iteraciones que se ejecuten. Para 5 iteraciones el tiempo de ejecución del programa es de 300 ms, mientras que para 20 iteraciones el tiempo es de 1.2 s.



(a) Porcentaje del tiempo de ejecución del programa Iter para 5 iteraciones.



(b) Porcentaje del tiempo de ejecución del programa Iter para 20 iteraciones.

Figura 5.11: Programa Iter

Finalmente, tenemos el programa "Prime" mostrado en la Figura 5.12 para el caso de 5000 iteraciones. Podemos ver que la mayor parte del tiempo el programa se pasa ejecutando una función interna de las librerías usadas en el código, llamada *aeabi_idiv*, con un 52 % de tiempo de ejecución. También vemos la función *prime* con un porcentaje de ejecución de 27.3 %, la función interna *aeabi_idivmod* con 20.4 % y podemos ver que el tiempo de ejecución de *main* no es lo suficientemente significativo para ser considerado en la medición. El tiempo total de ejecución es de 961 ms.

El patrón de ejecución de este programa para las primeras 15 muestras y las últimas 15 se muestra en la Figura 5.13. Arriba tenemos las primeras 15 iteraciones del programa, mientras que abajo tenemos las últimas 15.

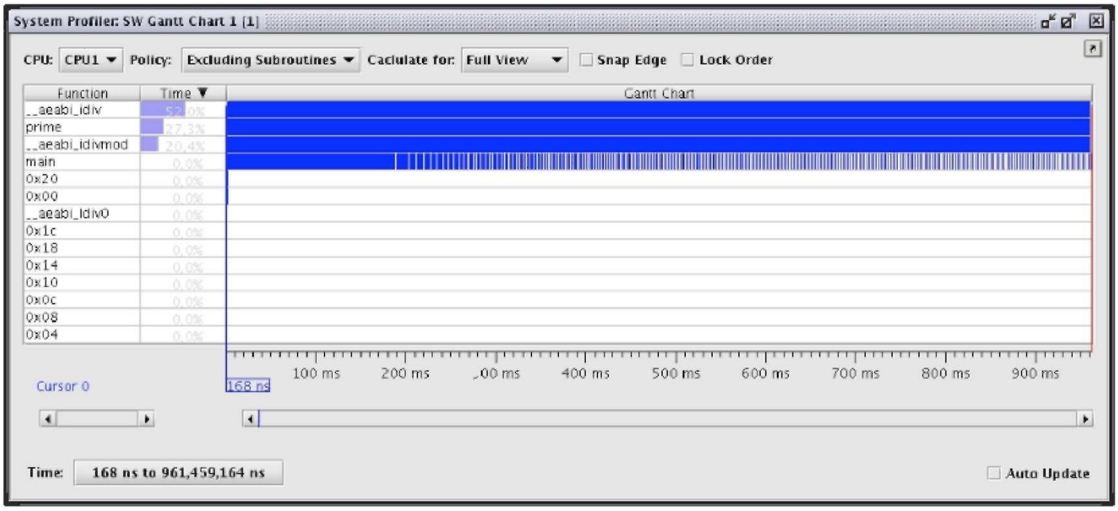


Figura 5.12: Porcentaje del tiempo de ejecución del programa Prime para 5000 iteraciones.

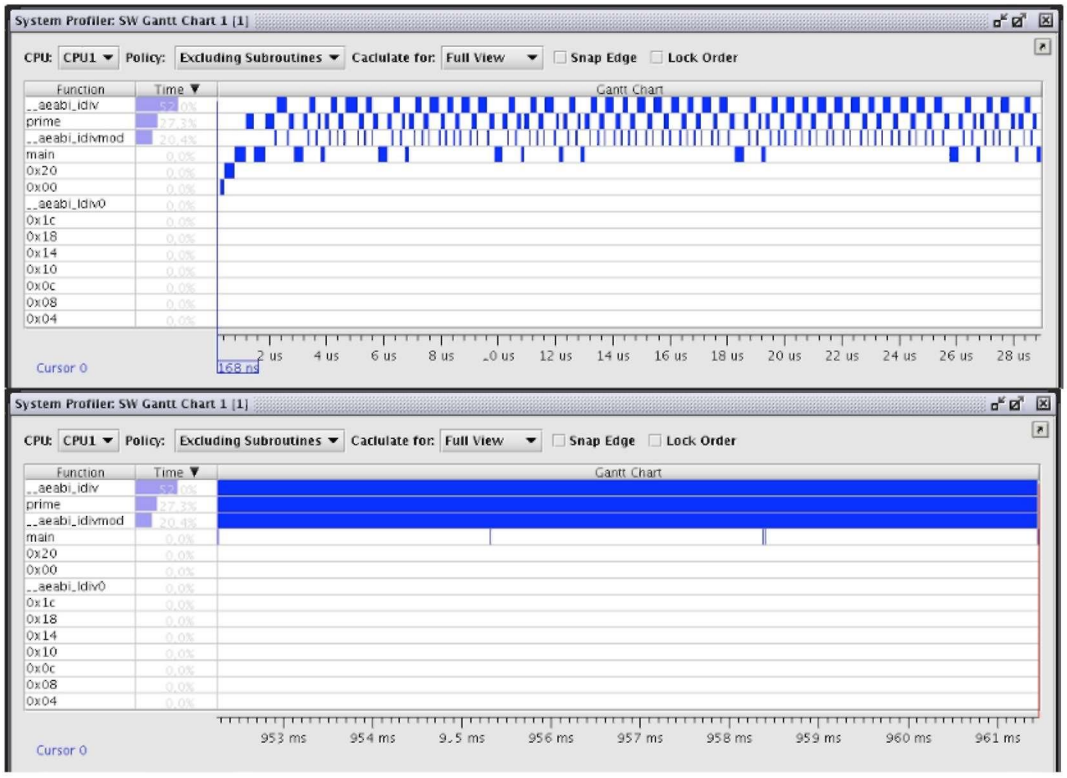


Figura 5.13: Porcentaje del tiempo de ejecución del programa en diferentes segmentos de tiempo.

5.4.3. Comparativa

Ahora es tiempo de comparar los resultados obtenidos con la tarjeta de desarrollo, contra los obtenidos en el simulador. En la Figura 5.14 tenemos la comparativa para el programa "Collatz" para 4000 y 200000 iteraciones. El tiempo de ejecución se muestra en la Figura 5.15.

Como se puede ver, las variaciones en los porcentajes de ejecución en cada una de las plataformas son pequeñas, sin embargo, permiten apreciar que al aumentar el numero de iteraciones aumenta la ejecución de las funciones *step* y *nseq* de manera exponencial, mientras que la función *main* incrementa su ejecución de manera lineal, por lo tanto su porcentaje de ejecución se ve disminuido conforme aumentan las iteraciones. debido a los tiempos de ejecución tan pequeños en la ejecución de 4000 iteraciones, se puede atribuir la diferencia en porcentajes de ejecución entre la OMAP y Seamless a overhead del sistema operativo, así como a cambios de contexto, situación que no sucede en Seamless al no utilizar un sistema operativo para correr los programas.

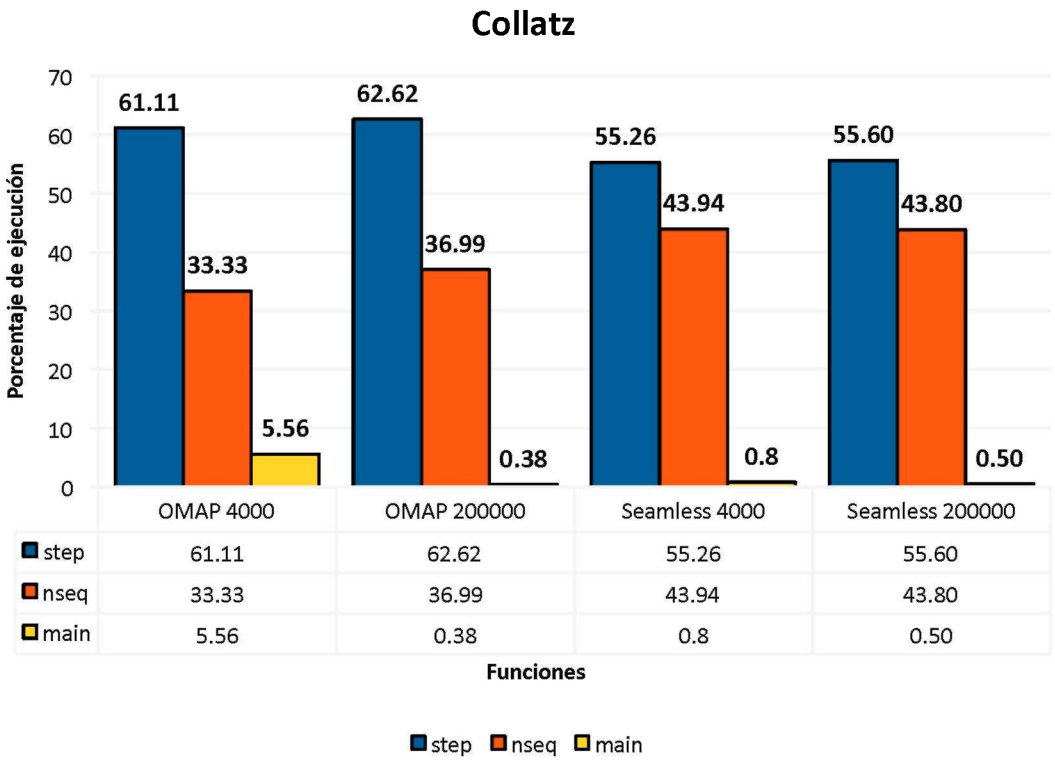


Figura 5.14: Porcentaje del tiempo de ejecución del programa para 4000 y 200000 iteraciones.

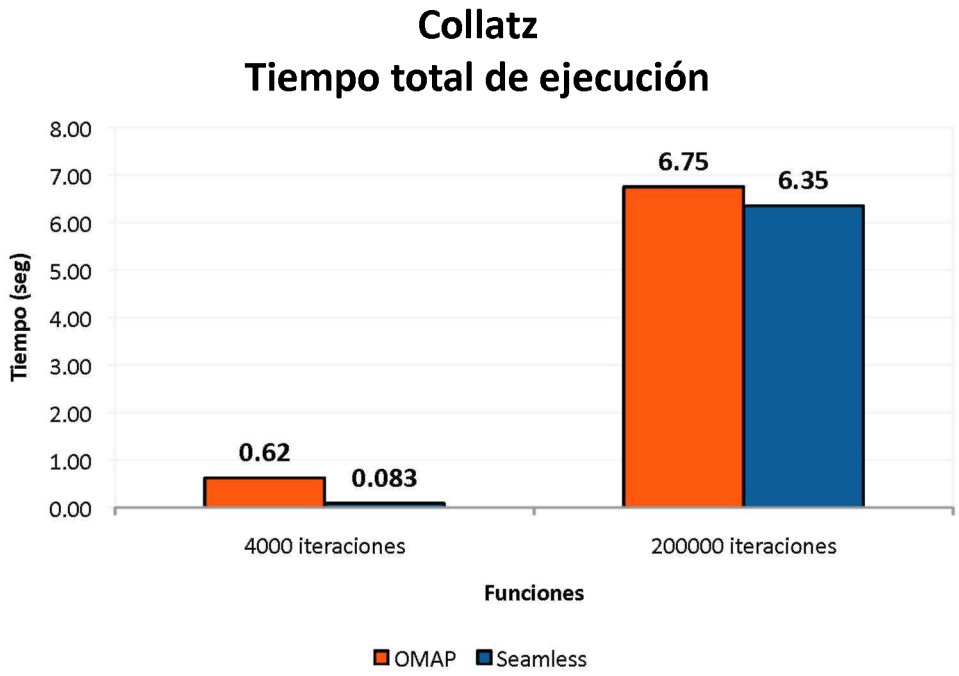


Figura 5.15: Tiempo total de ejecución del programa para 4000 y 200000 iteraciones.

Para el programa "Iter", podemos ver los resultados comparativos en la Figura 5.16, para 5 y 20 iteraciones. El tiempo de ejecución se muestra en la Figura 5.17. Como se puede ver, la variación en los porcentajes de ejecución es muy pequeña en la OMAP y en Seamless no hay variación alguna. Esto se debe a que Seamless corre los programas siempre de la misma manera debido a la falta de un sistema operativo, a diferencia de la plataforma OMAP que depende del sistema operativo para correr los programas y por tanto ninguna ejecución de programas es idéntica. En los tiempos de ejecución vemos de nuevo que la diferencia en los tiempos esta dada por el overhead que incurre el uso de un sistema operativo.

Finalmente, en la Figura 5.18 tenemos los resultados del programa "Prime" para 5000 iteraciones. El tiempo de ejecución se muestra en la Figura 5.19. En el caso de este programa, la variación principal en los porcentajes de ejecución esta dada por las librerías usadas para realizar las operaciones requeridas por las funciones del programa. Estas librerías son específicas del toolchain ya que dependen de otras librerías que pudieran no existir, como es el caso de un toolchain orientado a una plataforma que no utilice un sistema operativo, por lo tanto puede no ser posible optimizar su funcionamiento.

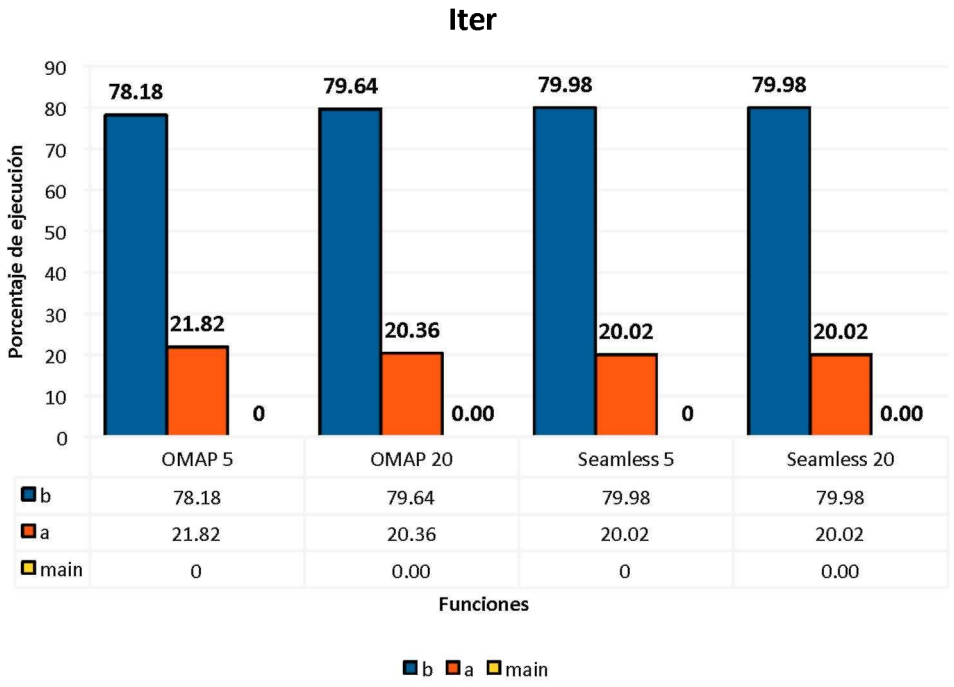


Figura 5.16: Porcentaje del tiempo de ejecución del programa para 5 y 20 iteraciones.

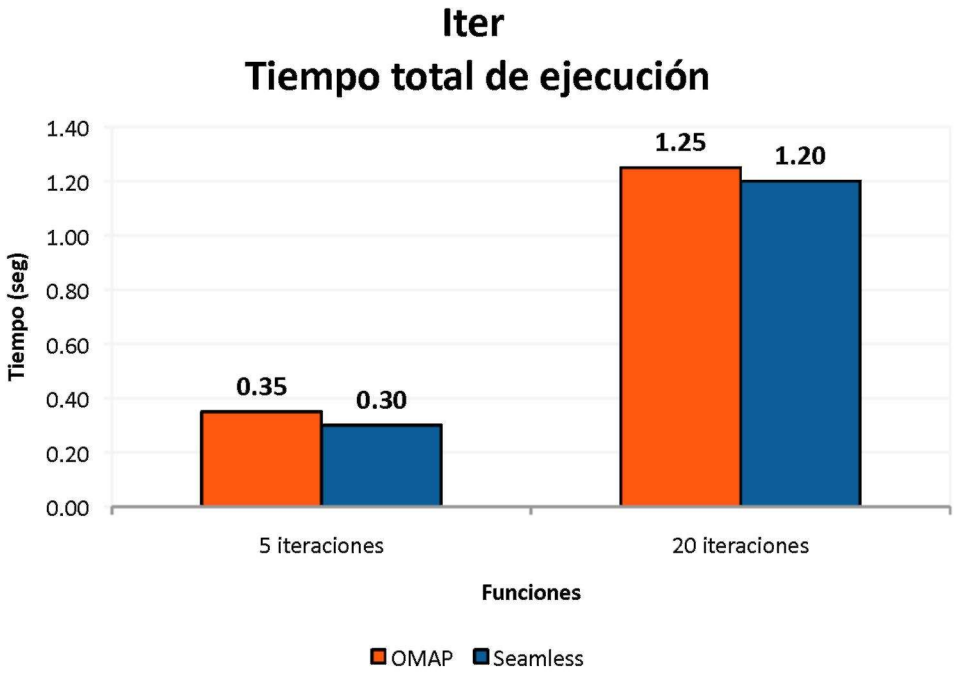


Figura 5.17: Tiempo total de ejecución del programa para 5 y 20 iteraciones.

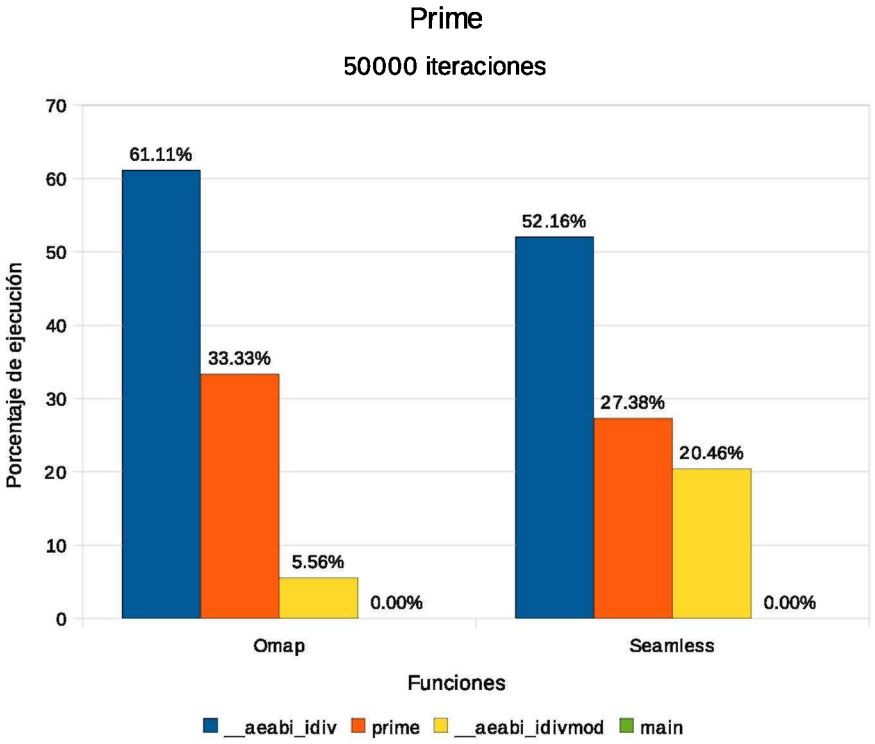


Figura 5.18: Porcentaje del tiempo de ejecución del programa para 5000 iteraciones.

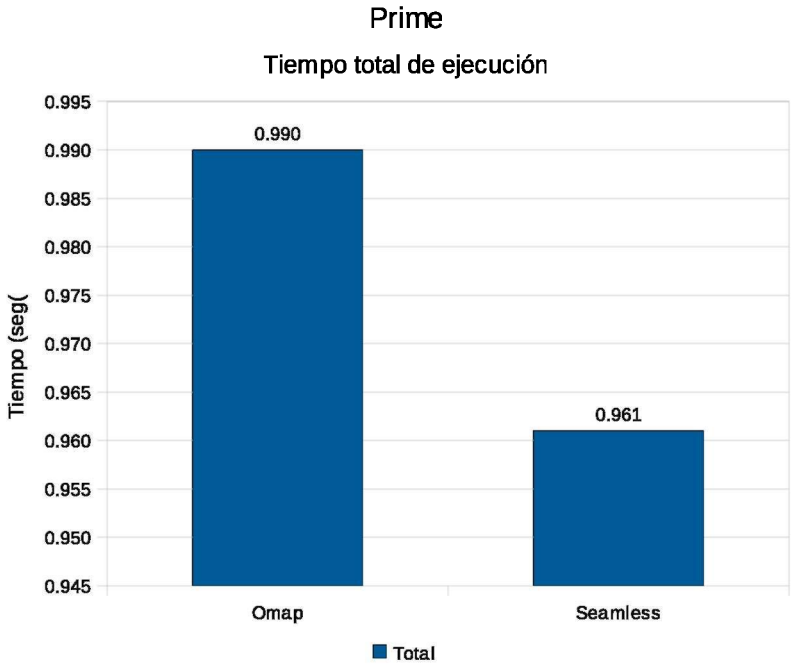


Figura 5.19: Tiempo total de ejecución del programa para 5000 iteraciones.

5.5. Resultados usando dos procesadores

Para poder determinar la diferencia en rendimiento de una implementación de dos procesadores contra una implementación de un solo procesador, se utilizó un benchmark conocido como "Livermoore Loops".

Este benchmark ejecuta 24 pruebas diferentes (llamadas "kernels") para determinar el rendimiento de un procesador. Para nuestra comparación, se ejecutó este código primero en la implementación de un solo procesador sin realizar modificaciones al código original. Para la implementación de los dos procesadores, se dividió el código en dos partes de tal manera que se pueda aprovechar el uso de dos procesadores lo más posible.

En la Figura 5.20 tenemos la ejecución del código para un solo procesador. El tiempo total de ejecución fue de 105.43 ms

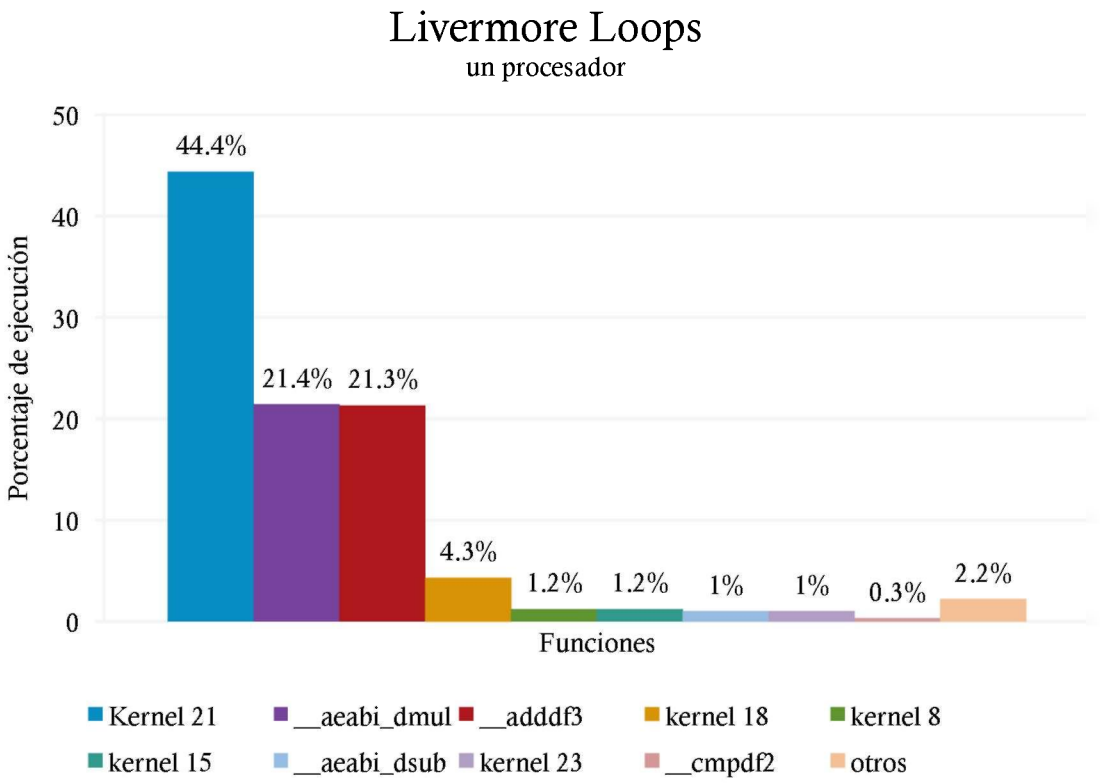


Figura 5.20: Porcentaje de ejecución para un solo procesador.

Ahora tenemos los resultados de la ejecución del benchmark utilizando los dos procesadores. (Figuras 5.21 y 5.22).

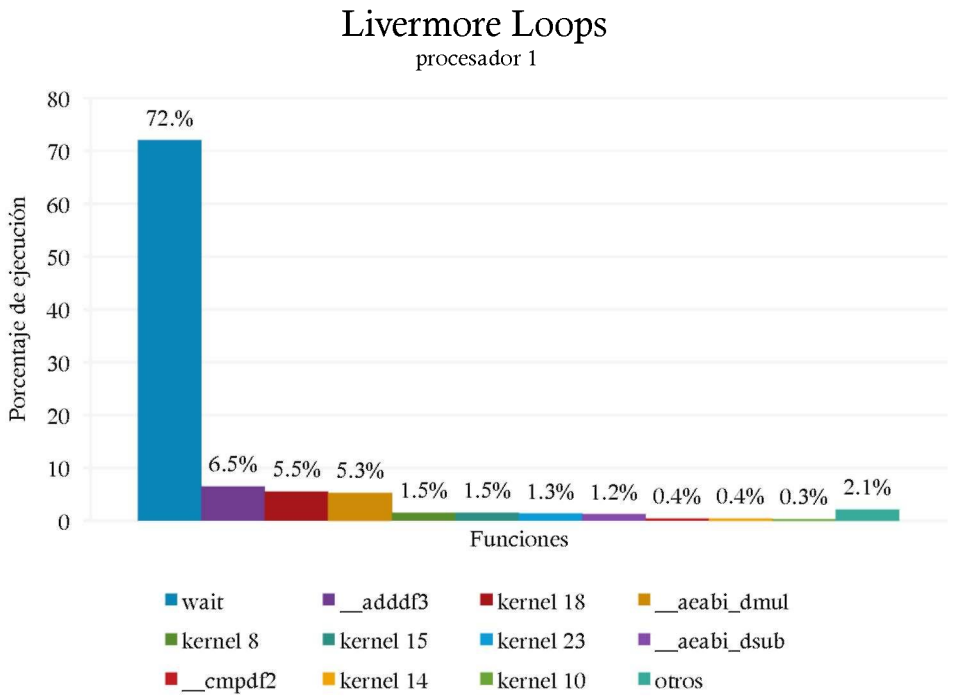


Figura 5.21: Porcentaje de ejecución para el procesador 1.

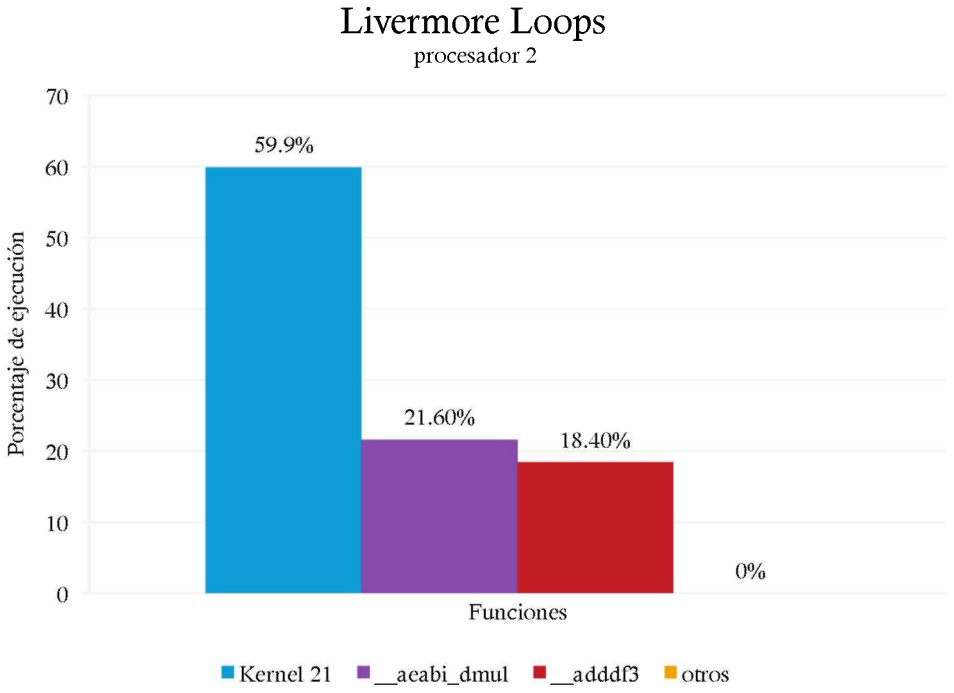


Figura 5.22: Porcentaje de ejecución para el procesador 2.

En este caso, el tiempo total de ejecución fue de 83.31 ms. Esto representa una mejora de un 21 % con respecto a la ejecución del código en un solo procesador. Si bien la mejora en el tiempo total de ejecución no es muy significativa, si observamos la Figura 5.24 veremos que en realidad, las pruebas ejecutadas por este procesador se realizaron en solo 23.36 ms. El resto del tiempo, el procesador se quedó esperando a que el segundo procesador terminara la ejecución de sus pruebas, lo cuál representa el 72 % del tiempo total de ejecución.

Si sumamos el tiempo que le tomó a ambos procesadores ejecutar las pruebas, veremos que el resultado de dicha suma es de 106.67 ms. lo cuál es aproximado al tiempo que tardó la ejecución en un solo procesador si consideramos que en la prueba con los dos procesadores hay un pequeño overhead dado por la sincronización de los procesadores, así como también por la declaración de todas las variables en ambos procesadores (Figura 5.23).

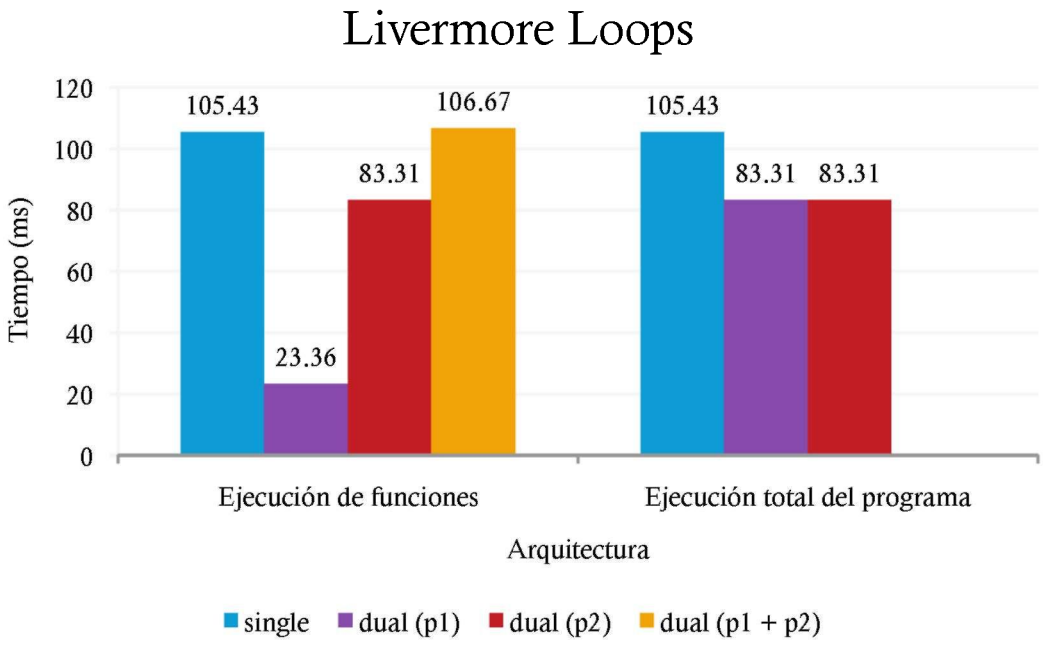


Figura 5.23: Comparativa del tiempo total de ejecución.

Figura 5.24: Tiempo total de ejecución para el procesador 1.

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

En este trabajo se logro cumplir los objetivos de determinar el desempeño del procesador de propósito general (ARM926) en la tarjeta de desarrollo (OMAP) y en Seamless, así como también la implementación de un esquema de intercomunicación entre dos procesadores usando una memoria compartida usando Seamless. Para lograr esto se establecieron esquemas y procedimientos para la medición de desempeño en procesadores OMAP usando Linux, además de establecer esquemas y procedimientos para la co-verificación de hardware y software de un sistema multiprocesador. Con dichos procedimientos fue posible simular y obtener mediciones de la implementación de una memoria compartida y la implementación de un sistema de comunicación entre procesos mediante mailboxes usando Seamless.

6.2. Trabajo Futuro

Gracias a los objetivos logrados en este trabajo, es posible el desarrollo de un BIOS para facilitar la intercomunicación de los dos procesadores al crear librerías y drivers que permitieran mandar y recibir datos de un procesador a otro de una manera sencilla y que puedan ser utilizados por un sistema operativo. Además, gracias a los esquemas y procedimientos establecidos aquí, es posible realizar pruebas de desempeño para aplicaciones multiprocesador, como por ejemplo la intercomunicación con un DSP, la cual puede ser posteriormente implementada y perfilada en software usando Seamless.

Apéndice A

Códigos Utilizados

A.1. Busybox

```
[etc/busybox.conf]
[SUID]
su = ssx root.0 # applet su can be run by anyone and runs with euid=0/egid=0
su = ssx        # exactly the same
[etc/fstab]
# /etc/fstab: static file system information.
#
# <file system> <mount point>   <type> <options>  <dump> <pass>
#/dev/ram0 /      ext2    defaults     0      0
proc /proc       proc    defaults     0      0
sysfs /sys       sysfs   defaults     0      0
none /dev/pts    devpts gid=5,mode=620 0     0
[etc/group]
root:x:0:root
sshd:x:74:
[etc/hosts]
127.0.0.1       localhost.localdomain   localhost
# Additional names can be added
#192.168.1.106 squeezel


[etc/inittab]
# This is run first except when booting in single-user mode.
#
::sysinit:/etc/rc.d/rcS
#
#
#con::respawn:/sbin/getty console
# Put a getty on the serial port
ttyS0::respawn:/sbin/getty -L ttyS0 115200 vt100
#
```

```
# /sbin/getty invocations for selected ttys
#
#tty1::respawn:/sbin/getty 115200 tty1
#tty2::respawn:/sbin/getty 115200 tty2
#tty3::respawn:/sbin/getty 115200 tty3
#tty4::respawn:/sbin/getty 115200 tty4
#tty5::respawn:/sbin/getty 115200 tty5
#tty6::respawn:/sbin/getty 115200 tty6
#tty7::respawn:/sbin/getty 115200 tty7
#tty8::respawn:/sbin/getty 115200 tty8
#tty9::respawn:/sbin/getty 115200 tty9
#
#
# Example of how to put a getty on a serial line (for a terminal)
#
#::respawn:/sbin/getty -L ttyS0 9600 vt100
#::respawn:/sbin/getty -L ttyS1 9600 vt100
#
# Example how to put a getty on a modem line.
#::respawn:/sbin/getty 57600 ttyS2
#
# Stuff to do when restarting the init process
::restart:/sbin/init
#
# Stuff to do before rebooting
::ctrlaltdel:/sbin/reboot
::shutdown:/etc/rc.d/rcE
::shutdown:/bin/umount -a -r
::shutdown:/sbin/swapoff -a
[etc/login.defs]
# *REQUIRED*
#   Directory where mailboxes reside, _or_ name of file, relative to the
#   home directory. If you _do_ define both, MAIL_DIR takes precedence.
#   QMAIL_DIR is for Qmail
#
#QMAIL_DIR     Maildir
MAIL_DIR        /var/spool/mail
#MAIL_FILE     .mail
# Password aging controls:
#
#       PASS_MAX_DAYS Maximum number of days a password may be used.
#       PASS_MIN_DAYS Minimum number of days allowed between password changes.
#       PASS_MIN_LEN   Minimum acceptable password length.
#       PASS_WARN_AGE Number of days warning given before a password expires.
#
PASS_MAX_DAYS 99999
PASS_MIN_DAYS 0
PASS_MIN_LEN    5
PASS_WARN_AGE 7
#
# Min/max values for automatic uid selection in useradd
```

```
#
UID_MIN                  500
UID_MAX                60000
#
# Min/max values for automatic gid selection in groupadd
#
GID_MIN                  500
GID_MAX                60000
#
# If defined, this command is run when removing a user.
# It should remove any at/cron/print jobs etc. owned by
# the user to be removed (passed as the first argument).
#
#USERDEL_CMD    /usr/sbin/userdel_local
#
# If useradd should create home directories for users by default
# On RH systems, we do. This option is ORed with the -m flag on
# useradd command line.
#
CREATE_HOME     yes
[etc/nsswitch.conf]
# /etc/nsswitch.conf
#
# Example configuration of GNU Name Service Switch functionality.
# If you have the 'glibc-doc' and 'info' packages installed, try:
# 'info libc "Name Service Switch"' for information about this file.
passwd:         files
group:          files
shadow:         files
hosts:          files dns
networks:       files
protocols:      files
services:       files
ethers:         files
rpc:            files
netgroup:       files


[etc/passwd]
root:x:0:0:Linux User,,,:/root:/bin/sh
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/bin/false


[etc/securetty]
# /etc/securetty: list of terminals on which root is allowed to login.
# See securetty(5) and login(1).
console
# for people with serial port consoles
ttyS0
# for devfs
tts/0
```

```
# Standard consoles
tty1
tty2
tty3
tty4
tty5
tty6
tty7
tty8
tty9
tty10
tty11
tty12
tty13
tty14
tty15
tty16
tty17
tty18
tty19
tty20
tty21
tty22
tty23
tty24
tty25
tty26
tty27
tty28
# Same as above, but these only occur with devfs devices
vc/1
vc/2
vc/3
vc/4
vc/5
vc/6
vc/7
vc/8
vc/9
vc/10
vc/11
vc/12
vc/13
vc/14
vc/15
vc/16
vc/17
vc/18
vc/19
vc/20
vc/21
```

```
vc/22
vc/23
vc/24
vc/25
vc/26
vc/27
vc/28
vc/29
vc/30
vc/31
vc/32
vc/33
vc/34
vc/35
vc/36
# psudo terminals used by telnet
pts/0
pts/1
pts/2
pts/3
pts/4
pts/5
pts/6
pts/7
```

```
[etc/shadow]
root::12439:0:99999:7:::
sshd:*:11880:0:99999:7:-1:-1:0
```

```
[etc/udhcpd.conf]
# Sample udhcpd configuration file (/etc/udhcpd.conf)
# The start and end of the IP lease block
# The interface that udhcpd will use
interface       eth0            #default: eth0
# The maximim number of leases (includes addressesd reserved
# by OFFER's, DECLINE's, and ARP conficts
#max_leases     254             #default: 254
# If remaining is true (default), udhcpd will store the time
# remaining for each lease in the udhcpd leases file. This is
# for embedded systems that cannot keep time between reboots.
# If you set remaining to no, the absolute time that the lease
# expires at will be stored in the dhcpd.leases file.
#remaining      yes             #default: yes
# The time period at which udhcpd will write out a dhcpd.leases
# file. If this is 0, udhcpd will never automatically write a
# lease file. (specified in seconds)
#auto_time      7200            #default: 7200 (2 hours)
# The amount of time that an IP will be reserved (leased) for if a
# DHCP decline message is received (seconds).
```

```
                              #default: 3600 (1 hour)
#decline_time   3600
# The amount of time that an IP will be reserved (leased) for if an
# ARP conflct occurs. (seconds
#conflict_time 3600             #default: 3600 (1 hour)
# How long an offered address is reserved (leased) in seconds
#offer_time     60               #default: 60 (1 minute)
# If a lease to be given is below this value, the full lease time is
# instead used (seconds).
#min_lease      60               #defult: 60
# The location of the leases file
#lease_file     /var/lib/misc/udhcpd.leases   #defualt: /var/lib/misc/udhcpd.leases
# The location of the pid file
#pidfile        /var/run/udhcpd.pid     #default: /var/run/udhcpd.pid
# Everytime udhcpd writes a leases file, the below script will be called.
# Useful for writing the lease file to flash every few hours.
#notify_file                            #default: (no script)
#notify_file    dumpleases      # <--- usefull for debugging
# The following are bootp specific options, setable by udhcpd.
#siaddr         192.168.0.22            #default: 0.0.0.0
#sname          zorak                   #default: (none)
#boot_file      /var/nfs_root           #default: (none)
# The remainer of options are DHCP options and can be specifed with the
# keyword 'opt' or 'option'. If an option can take multiple items, such
# as the dns option, they can be listed on the same line, or multiple
# lines. The only option with a default is 'lease'.


[etc/rc.d/rcE]
#!/bin/sh
#End the local scripts
echo "Shutting Down Local Systems"
for i in /etc/rc.d/rc.local/E*
do
   $i stop
done
#umount all before dying
/bin/umount -a
echo "DOne"


[etc/rc.d/rcS]
#!/bin/sh
echo "starting up local systems"
/bin/mount -a
# below getting rid of ram being mounted ro
/bin/mount -o remount /
#
# The following is for dhcp
#
#ifconfig eth0 0.0.0.0
```

```
#/sbin/udhcpc
#
# Instead, if you want static IP address
#
#ifconfig eth0 192.168.1.13 netmask 255.255.252.0
#route add default gw 192.168.1.1
#
# Run ssh daemon
#/sbin/sshd
for i in /etc/rc.d/rc.local/S*
do
   $i start
done
echo "done"


[etc/rc.d/init.d/udev]
#!/bin/sh -e
# chkconfig: S 4 99
#set -x
PATH="/sbin:/bin:/usr/bin:/usr/sbin"
# defaults
tmpfs_size="10M"
udev_root="/dev"
################################################################################
# we need to unmount /dev/pts/ and remount it later over the tmpfs
unmount_devpts() {
  if mountpoint -q /dev/pts/; then
    umount -l /dev/pts/
  fi
  if mountpoint -q /dev/shm/; then
    umount -l /dev/shm/
  fi
}
# mount a tmpfs over /dev, if somebody did not already do it
mount_tmpfs() {
  if grep -E -q "^[^[:space:]]+ /dev tmpfs" /proc/mounts; then
    return 0
  fi
  # /dev/.static/dev/ is used by MAKEDEV to access the real /dev/ directory.
  # /etc/udev/ is recycled as a temporary mount point because it's the only
  # directory which is guaranteed to be available.
  mount -n --bind /dev /etc/udev
  echo -n "Mounting a tmpfs over /dev..."
  if ! mount -n -o size=$tmpfs_size,mode=0755 -t tmpfs none /dev; then
    echo " FAILED!"
    echo "FATAL: udev requires tmpfs support, not started."
    umount /etc/udev
    exit 1
  fi
  # using ln to test if /dev works, because touch is in /usr/bin/
```

```
  if ln -s test /dev/test-file; then
    rm /dev/test-file
    echo "done."
  else
    echo " FAILED!"
    echo "FATAL: udev requires tmpfs support, not started."
    umount /etc/udev
    umount /dev
    exit 1
  fi
  mkdir -p /dev/.static/dev
  chmod 700 /dev/.static/
  mount -n --move /etc/udev /dev/.static/dev
}
# I hate this hack. -- Md
make_extra_nodes() {
  [ -e etc/udev/links.conf ] || return
  grep '^[^#]' /etc/udev/links.conf | \
  while read type name arg1; do
    [ "$type" -a "$name" -a ! -e "/dev/$name" -a ! -L "/dev/$name" ] ||continue
    case "$type" in
      L) ln -s $arg1 /dev/$name ;;
      D) mkdir -p /dev/$name ;;
      M) mknod -m 600 /dev/$name $arg1 ;;
      *) echo "links.conf: unparseable line ($type $name $arg1)" ;;
    esac
  done
}
# this function is duplicated in preinst, postinst and d-i
supported_kernel() {
  case "$(uname -r)" in
    2.[012345].*|2.6.[0-7]|2.6.[0-7][!0-9]*) return 1 ;;
  esac
  return 0
}
# Kernels < 2.6.10 break some drivers when udevsend is used as the hotplug
# multiplexer. See #297481 for details.
events_not_ordered() {
  case "$(uname -r)" in
    2.6.[0-9]|2.6.[0-9][!0-9]*) return 0 ;;
  esac
  return 1
}
# shell version of /usr/bin/tty
my_tty() {
  [ -x /usr/bin/readlink ] || return
  [ -e /proc/self/fd/0 ] || return
  readlink /proc/self/fd/0 || true
}
warn_if_interactive() {
  if [ "$RUNLEVEL" = "S" -a "$PREVLEVEL" = "N" ]; then
```

```
    return
  fi
  TTY=$(my_tty)
  if [ -z "$TTY" -o "$TTY" = "/dev/console" -o "$TTY" = "/dev/ttyS0" ]; then
    return
  fi
  printf "\n\n\nIt has been detected that the command\n\n\t$0 $*\n\n"
  printf "has been run from an interactive shell $TTY\n"
  printf "It will probably not do what you expect, so this script will wait\n"
  printf "1 seconds before continuing. Press ^C to stop it.\n"
  printf "RUNNING THIS COMMAND IS HIGHLY DISCOURAGED!\n\n\n\n"
  sleep 1
}
##############################################################################
[ -x /sbin/udevstart ] || exit 0
. /etc/udev/udev.conf
if [ "$UDEV_DISABLED" = "yes" ]; then
  echo "udev disabled on the kernel command line, not started"
  exit 0
fi
if ! supported_kernel; then
  echo "udev requires a kernel >= 2.6.8, not started."
  exit 0
fi
if ! grep -q '[[:space:]]tmpfs$' /proc/filesystems; then
  echo "udev requires tmpfs support, not started."
  exit 0
fi
if [ ! -e /proc/sys/kernel/hotplug ]; then
  echo "udev requires hotplug support, not started."
  exit 0
fi
##############################################################################
udev_root=${udev_root%/}
if [ "$udev_root" != "/dev" ]; then
  echo "WARNING: udev_root != /dev/"
case "$1" in
  start)
    if [ -e "$udev_root/.udevdb" ]; then
      if mountpoint -q /dev/; then
        echo "FATAL: udev is already active on $udev_root."
        exit 1
      else
        echo "WARNING: .udevdb already exists on the old $udev_root!"
      fi
    fi
    mount -n -o size=$tmpfs_size,mode=0755 -t tmpfs none $udev_root
    echo -n "Creating initial device nodes..."
    udevstart
    echo "done."
    ;;
```

```
  stop)
    start-stop-daemon --stop --exec /sbin/udevd --oknodo --quiet
    echo -n "Unmounting $udev_root..."
    # unmounting with -l should never fail
    if umount -l $udev_root; then
      echo "done."
    else
      echo "failed."
    fi
    ;;
  restart|force-reload)
    $0 stop
    $0 start
    ;;
  *)
    echo "Usage: /etc/init.d/udev {start|stop|restart|force-reload}"
    exit 1
    ;;
esac
  exit 0
fi # udev_root != /dev
################################################################################
# When modifying this script, do not forget that between the time that
# the new /dev has been mounted and udevstart has been run there will be
# no /dev/null. This also means that you cannot use the "&" shell command.
case "$1" in
  start)
    if [ -e "$udev_root/.udevdb" ]; then
      if mountpoint -q /dev/; then
        echo "FATAL: udev is already active on $udev_root."
        exit 1
      else
        echo "WARNING: .udevdb already exists on the old $udev_root!"
      fi
    fi
    warn_if_interactive
    if ! events_not_ordered; then
      echo /sbin/udevsend > /proc/sys/kernel/hotplug
    fi
    unmount_devpts
    mount_tmpfs
    [ -d /proc/1 ] || mount -n /proc
    echo -n "Creating initial device nodes..."
    udevstart
    echo "done."
    make_extra_nodes
    ;;
  stop)
    warn_if_interactive
    start-stop-daemon --stop --exec /sbin/udevd --oknodo --quiet
    unmount_devpts
```

```
    if [ -d /dev/.static/dev/ ]; then
      umount -l /dev/.static/dev/ || true
    fi
    echo -n "Unmounting /dev..."
    # unmounting with -l should never fail
    if umount -l /dev; then
      echo "done."
      if [-x "/etc/init.d/mountvirtfs" ]; then
      /etc/init.d/mountvirtfs start
      else
      echo "No mountvirtfs"
      fi
    else
      echo "failed."
    fi
    ;;
  restart|force-reload)
    start-stop-daemon --stop --exec /sbin/udevd --oknodo --quiet
    echo -n "Recreating device nodes..."
    udevstart
    make_extra_nodes
    echo "done."
    ;;
  *)
    echo "Usage: /etc/init.d/udev {start|stop|restart|force-reload}"
    exit 1
    ;;
esac
exit 0
```

A.2. Servicios de Linux

```
[etc/tftp]
# description: The tftp server serves files using the trivial file transfer \
#       protocol. The tftp protocol is often used to boot diskless \
#         workstations, download configuration files to network-aware printers, \
#         and to start the installation process for some operating systems.
service tftp
{
  socket_type             = dgram
  protocol                = udp
  wait                    = yes
  user                    = root
  server                  = /usr/sbin/in.tftpd
  server_args             = -s /tftpboot
  disable                 = no
  per_source              = 11
```

```
  cps                     = 100 2
}

[etc/exports]
# /etc/exports: the access control list for filesystems which may be exported
# to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync) hostname2(ro,sync)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt)
# /srv/nfs4/homes  gss/krb5i(rw,sync)
/opt/omap/filesystem    192.168.0.0/24 (rw,nosync,no_root_squash)
```

A.3. Códigos de prueba

```
[collatz.c]

#include <stdio.h>

/* Computes the length of Collatz sequences */

unsigned int
step (unsigned int x)
{
  if (x % 2 == 0)
    {
      return (x / 2);
    }
  else
    {
      return (3 * x + 1);
    }
}

unsigned int
nseq (unsigned int x0)
{
  unsigned int i = 1, x;

  if (x0 == 1 || x0 == 0)
    return i;

  x = step (x0);

  while (x != 1 && x != 0)
```

```
    {
      x = step (x);
      i++;
    }

  return i;
}

int
main (void)
{
  unsigned int i, m = 0, im = 0;

  for (i = 1; i < 4000; i++)
    {
      unsigned int k = nseq (i);

      if (k > m)
        {
          m = k;
          im = i;
          printf ("sequence length = %u for %u\n", m, im);
        }
    }

  return 0;
}
```

[iter.c]

```
#include <stdio.h>

int a(void) {
  int i=0,g=0;
  while(i++<100000)
  {
     g+=i;
  }
  return g;
}
int b(void) {
  int i=0,g=0;
  while(i++<400000)
  {
    g+=i;
  }
  return g;
}

int main(void)
```

```
{
   int iterations=5;

   printf("No of iterations = %d\n", iterations);

   while(iterations--)
   {
      a();
      b();
   }
   return 0;
}
```

[prime.c]

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

 int prime (int num);

 int main()
 {
  int i;
  int colcnt = 0;
 for (i=2; i <= 5000; i++)
  if (prime(i)) {
   colcnt++;
   if (colcnt%9 == 0) {
     printf("%5d\n",i);
     colcnt = 0;
   }
  else
    printf("%5d ", i);
  }
//  putchar(?\n?);
  return 0;
 }

 int prime (int num) {
   /* check to see if the number is a prime? */
   int i;
   for (i=2; i < num; i++)
   if (num %i == 0)
    return 0;
   return 1;
}
```

[Livermore.c]

```
#include <stdio.h>
```

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

/* Define the structs or COMMON BLOCKS */

//    double tk[6];

 /*   long mk;
    long ik;
    long im;
    long ml;
    long il;
    long mruns;
    long nruns;
    long jr;
    long npfs[47][3][8];*/

    /*double tic;
    double times[47][3][8];
    double see[3][8][3][5];
    double terrs[47][3][8];
    double csums[47][3][8];
    double fopn[47][3][8];
    double dos[47][3][8];*/

  //  long ion;
    long j5;
    long k2;
    long k3;
 //  long multi;
 //   long laps;
    long loop=10;
    long m;
 /*   long kr;
    long it;
    long n13h;
    long ibuf;
    long npass;
    long nfail;*/
    long n=10;
  /*  long n1;
    long n2;
    long n13;
    long n213;
    long n813;
    long n14;
    long n16;
    long n416;
    long n21;
    long nt1;
    long nt2;*/
```

```
   double a11;
   double a12;
   double a13;
   double a21;
   double a22;
   double a23;
   double a31;
   double a32;
   double a33;
   double ar;
   double br;
   double c0;
   double cr;
   double di;
   double dk;
   double dm22;
   double dm23;
   double dm24;
   double dm25;
   double dm26;
   double dm27;
   double dm28;
   double dn;
   double e3;
   double e6;
   double expmax;
   double flx;
   double q;
   double qa;
   double r;
   double ri;
   double s;
   double scale;
   double sig;
   double stb5;
   double t;
   double xnc;
   double xnei;
   double xnm;

/*   double time[47];
   double csum[47];
   double ww[47];
   double wt[47];
   double ticks;
   double fr[9];
   double terr1[47];
   double sumw[7];
   double start;
   double skale[47];
```

```
   double bias[47];
   double ws[95];
   double total[47];
   double flopn[47];
   long iq[7];
   long npf;
   long npfs1[47];*/

/*   double wtp[3];
   long mult[3];
   long ispan[3][47];
   long ipass[3][47];*/

   long e[96];
   long f[96];
   long ix[1001];
   long ir[1001];
   long zone[300];

   double u[1001];
   double v[1001];
   double w[1001];
   double x[1001];
   double y[1001];
   double z[1001];
   double g[1001];
   double du1[101];
   double du2[101];
   double du3[101];
   double grd[1001];
   double dex[1001];
   double xi[1001];
   double ex[1001];
   double ex1[1001];
   double dex1[1001];
   double vx[1001];
   double xx[1001];
   double rx[1001];
   double rh[2048];
   double vsp[101];
   double vstp[101];
   double vxne[101];
   double vxnd[101];
   double ve3[101];
   double vlr[101];
   double vlin[101];
   double b5[101];
   double plan[300];
   double d[300];
   double sa[101];
   double sb[101];
```

```
    double p[512][4];
    double px[101][25];
    double cx[101][25];
    double vy[25][101];
    double vh[7][101];
    double vf[7][101];
    double vg[7][101];
    double vs[7][101];
    double za[7][101];
    double zp[7][101];
    double zq[7][101];
    double zr[7][101];
    double zm[7][101];
    double zb[7][101];
    double zu[7][101];
    double zv[7][101];
    double zz[7][101];
    double b[64][64];
    double c[64][64];
    double h[64][64];
    double u1[2][101][5];
    double u2[2][101][5];
    double u3[2][101][5];


main() {

    long argument , k , l , ipnt , ipntp , i;
    long lw , j , nl1 , nl2 , kx , ky , ip , kn;
    long i1 , j1 , i2 , j2 , nz , ink , jn , kb5i;
    long ii , lb , j4 , ng;
    double tmp , temp, sum, som;
//    char name[8];
    argument = 0;


    /*
     *****************************************************************
     *   Kernel 1 -- hydro fragment
     *****************************************************************
     *       DO 1 L = 1,Loop
     *       DO 1 k = 1,n
     *  1       X(k)= Q + Y(k)*(R*ZX(k+10) + T*ZX(k+11))
     */

    for ( l=1 ; l<=loop ; l++ ) {
        for ( k=0 ; k<n ; k++ ) {
            x[k] = q + y[k]*( r*z[k+10] + t*z[k+11] );
        }
```

```
    }
    argument = 1;


    /*
     ********************************************************************
     *   Kernel 2 -- ICCG excerpt (Incomplete Cholesky Conjugate Gradient)
     ********************************************************************
     *    DO 200  L= 1,Loop
     *        II= n
     *     IPNTP= 0
     *222   IPNT= IPNTP
     *     IPNTP= IPNTP+II
     *        II= II/2
     *         i= IPNTP
     CDIR$ IVDEP
     *    DO 2 k= IPNT+2,IPNTP,2
     *         i= i+1
     *  2   X(i)= X(k) - V(k)*X(k-1) - V(k+1)*X(k+1)
     *        IF( II.GT.1) GO TO 222
     *200 CONTINUE
     */

    for ( l=1 ; l<=loop ; l++ ) {
        ii = n;
        ipntp = 0;
        do {
            ipnt = ipntp;
            ipntp += ii;
            ii /= 2;
            i = ipntp - 1;
#pragma nohazard
            for ( k=ipnt+1 ; k<ipntp ; k=k+2 ) {
                i++;
                x[i] = x[k] - v[k  ]*x[k-1] - v[k+1]*x[k+1];
            }
        } while ( ii>0 );
    }
    argument = 2;


    /*
     ********************************************************************
     *   Kernel 3 -- inner product
     ********************************************************************
     *    DO 3 L= 1,Loop
     *         Q= 0.0
     *    DO 3 k= 1,n
     *  3      Q= Q + Z(k)*X(k)
     */
```

```
    for ( l=1 ; l<=loop ; l++ ) {
        q = 0.0;
        for ( k=0 ; k<n ; k++ ) {
            q += z[k]*x[k];
        }
    }
    argument = 3;


    /*
     ********************************************************************
     *   Kernel 4 -- banded linear equations
     ********************************************************************
     *           m= (1001-7)/2
     *    DO 444  L= 1,Loop
     *    DO 444  k= 7,1001,m
     *          lw= k-6
     *        temp= X(k-1)
     CDIR$ IVDEP
     *    DO   4  j= 5,n,5
     *       temp  = temp   - XZ(lw)*Y(j)
     *  4        lw= lw+1
     *       X(k-1)= Y(5)*temp
     *444 CONTINUE
     */

    m = ( 1001-7 )/2;
    for ( l=1 ; l<=loop ; l++ ) {
        for ( k=6 ; k<1001 ; k=k+m ) {
            lw = k - 6;
            temp = x[k-1];
#pragma nohazard
            for ( j=4 ; j<n ; j=j+5 ) {
                temp -= x[lw]*y[j];
                lw++;
            }
            x[k-1] = y[4]*temp;
        }
    }
    argument = 4;


    /*
     ********************************************************************
     *   Kernel 5 -- tri-diagonal elimination, below diagonal
     ********************************************************************
     *    DO 5 L = 1,Loop
     *    DO 5 i = 2,n
     *  5    X(i)= Z(i)*(Y(i) - X(i-1))
     */
```

```
    for ( l=1 ; l<=loop ; l++ ) {
        for ( i=1 ; i<n ; i++ ) {
            x[i] = z[i]*( y[i] - x[i-1] );
        }
    }
    argument = 5;


    /*
     ********************************************************************
     *   Kernel 6 -- general linear recurrence equations
     ********************************************************************
     *    DO  6  L= 1,Loop
     *    DO  6  i= 2,n
     *    DO  6  k= 1,i-1
     *        W(i)= W(i)  + B(i,k) * W(i-k)
     *  6 CONTINUE
     */

    for ( l=1 ; l<=loop ; l++ ) {
        for ( i=1 ; i<n ; i++ ) {
            for ( k=0 ; k<i ; k++ ) {
                w[i] += b[k][i] * w[(i-k)-1];
            }
        }
    }
    argument = 6;


    /*
     ********************************************************************
     *   Kernel 7 -- equation of state fragment
     ********************************************************************
     *    DO 7 L= 1,Loop
     *    DO 7 k= 1,n
     *      X(k)=     U(k  ) + R*( Z(k  ) + R*Y(k  )) +
     *   .         T*( U(k+3) + R*( U(k+2) + R*U(k+1)) +
     *   .         T*( U(k+6) + R*( U(k+5) + R*U(k+4))))
     *  7 CONTINUE
     */

    for ( l=1 ; l<=loop ; l++ ) {
#pragma nohazard
        for ( k=0 ; k<n ; k++ ) {
            x[k] = u[k] + r*( z[k] + r*y[k] ) +
                   t*( u[k+3] + r*( u[k+2] + r*u[k+1] ) +
                      t*( u[k+6] + r*( u[k+5] + r*u[k+4] ) ) );
        }
    }
    argument = 7;
```

```
    /*
     ******************************************************************
     *   Kernel 8 -- ADI integration
     ******************************************************************
     *    DO  8      L = 1,Loop
     *             nl1 = 1
     *             nl2 = 2
     *    DO  8     kx = 2,3
     CDIR$ IVDEP
     *    DO  8     ky = 2,n
     *          DU1(ky)=U1(kx,ky+1,nl1)  -  U1(kx,ky-1,nl1)
     *          DU2(ky)=U2(kx,ky+1,nl1)  -  U2(kx,ky-1,nl1)
     *          DU3(ky)=U3(kx,ky+1,nl1)  -  U3(kx,ky-1,nl1)
     *    U1(kx,ky,nl2)=U1(kx,ky,nl1) +A11*DU1(ky) +A12*DU2(ky) +A13*DU3(ky)
     *   .       + SIG*(U1(kx+1,ky,nl1) -2.*U1(kx,ky,nl1) +U1(kx-1,ky,nl1))
     *    U2(kx,ky,nl2)=U2(kx,ky,nl1) +A21*DU1(ky) +A22*DU2(ky) +A23*DU3(ky)
     *   .       + SIG*(U2(kx+1,ky,nl1) -2.*U2(kx,ky,nl1) +U2(kx-1,ky,nl1))
     *    U3(kx,ky,nl2)=U3(kx,ky,nl1) +A31*DU1(ky) +A32*DU2(ky) +A33*DU3(ky)
     *   .       + SIG*(U3(kx+1,ky,nl1) -2.*U3(kx,ky,nl1) +U3(kx-1,ky,nl1))
     *  8 CONTINUE
     */

    for ( l=1 ; l<=loop ; l++ ) {
        nl1 = 0;
        nl2 = 1;
        for ( kx=1 ; kx<3 ; kx++ ){
#pragma nohazard
            for ( ky=1 ; ky<n ; ky++ ) {
               du1[ky] = u1[nl1][ky+1][kx] - u1[nl1][ky-1][kx];
               du2[ky] = u2[nl1][ky+1][kx] - u2[nl1][ky-1][kx];
               du3[ky] = u3[nl1][ky+1][kx] - u3[nl1][ky-1][kx];
               u1[nl2][ky][kx]=
                  u1[nl1][ky][kx]+a11*du1[ky]+a12*du2[ky]+a13*du3[ky] + sig*
                     (u1[nl1][ky][kx+1]-2.0*u1[nl1][ky][kx]+u1[nl1][ky][kx-1]);
               u2[nl2][ky][kx]=
                  u2[nl1][ky][kx]+a21*du1[ky]+a22*du2[ky]+a23*du3[ky] + sig*
                     (u2[nl1][ky][kx+1]-2.0*u2[nl1][ky][kx]+u2[nl1][ky][kx-1]);
               u3[nl2][ky][kx]=
                  u3[nl1][ky][kx]+a31*du1[ky]+a32*du2[ky]+a33*du3[ky] + sig*
                     (u3[nl1][ky][kx+1]-2.0*u3[nl1][ky][kx]+u3[nl1][ky][kx-1]);
            }
        }
    }
    argument = 8;


    /*
     ******************************************************************
     *   Kernel 9 -- integrate predictors
     ******************************************************************
```

```
*    DO 9  L = 1,Loop
*    DO 9  i = 1,n
*    PX( 1,i)= DM28*PX(13,i) + DM27*PX(12,i) + DM26*PX(11,i) +
*   .          DM25*PX(10,i) + DM24*PX( 9,i) + DM23*PX( 8,i) +
*   .          DM22*PX( 7,i) +  C0*(PX( 5,i) +      PX( 6,i))+ PX( 3,i)
*  9 CONTINUE
*/

for ( l=1 ; l<=loop ; l++ ) {
    for ( i=0 ; i<n ; i++ ) {
        px[i][0] = dm28*px[i][12] + dm27*px[i][11] + dm26*px[i][10] +
                   dm25*px[i][ 9] + dm24*px[i][ 8] + dm23*px[i][ 7] +
                   dm22*px[i][ 6] + c0*( px[i][ 4] + px[i][ 5]) + px[i][ 2];
    }
}
argument = 9;


/*
 ********************************************************************
 *   Kernel 10 -- difference predictors
 ********************************************************************
 *    DO 10  L= 1,Loop
 *    DO 10  i= 1,n
 *    AR      =      CX(5,i)
 *    BR      = AR - PX(5,i)
 *    PX(5,i) = AR
 *    CR      = BR - PX(6,i)
 *    PX(6,i) = BR
 *    AR      = CR - PX(7,i)
 *    PX(7,i) = CR
 *    BR      = AR - PX(8,i)
 *    PX(8,i) = AR
 *    CR      = BR - PX(9,i)
 *    PX(9,i) = BR
 *    AR      = CR - PX(10,i)
 *    PX(10,i)= CR
 *    BR      = AR - PX(11,i)
 *    PX(11,i)= AR
 *    CR      = BR - PX(12,i)
 *    PX(12,i)= BR
 *    PX(14,i)= CR - PX(13,i)
 *    PX(13,i)= CR
 * 10 CONTINUE
 */

for ( l=1 ; l<=loop ; l++ ) {
    for ( i=0 ; i<n ; i++ ) {
        ar        =      cx[i][ 4];
        br        = ar - px[i][ 4];
        px[i][ 4] = ar;
```

```
        cr        = br - px[i][ 5];
        px[i][ 5] = br;
        ar        = cr - px[i][ 6];
        px[i][ 6] = cr;
        br        = ar - px[i][ 7];
        px[i][ 7] = ar;
        cr        = br - px[i][ 8];
        px[i][ 8] = br;
        ar        = cr - px[i][ 9];
        px[i][ 9] = cr;
        br        = ar - px[i][10];
        px[i][10] = ar;
        cr        = br - px[i][11];
        px[i][11] = br;
        px[i][13] = cr - px[i][12];
        px[i][12] = cr;
    }
}
argument = 10;


/*
 ********************************************************************
 *   Kernel 11 -- first sum
 ********************************************************************
 *    DO 11 L = 1,Loop
 *        X(1)= Y(1)
 *    DO 11 k = 2,n
 * 11     X(k)= X(k-1) + Y(k)
 */

for ( l=1 ; l<=loop ; l++ ) {
    x[0] = y[0];
    for ( k=1 ; k<n ; k++ ) {
        x[k] = x[k-1] + y[k];
    }
}
argument = 11;


/*
 ********************************************************************
 *   Kernel 12 -- first difference
 ********************************************************************
 *    DO 12 L = 1,Loop
 *    DO 12 k = 1,n
 * 12     X(k)= Y(k+1) - Y(k)
 */

for ( l=1 ; l<=loop ; l++ ) {
    for ( k=0 ; k<n ; k++ ) {
```

```
        x[k] = y[k+1] - y[k];
    }
}
argument = 12;


/*
 ******************************************************************
 *   Kernel 13 -- 2-D PIC (Particle In Cell)
 ******************************************************************
 *    DO  13     L= 1,Loop
 *    DO  13    ip= 1,n
 *              i1= P(1,ip)
 *              j1= P(2,ip)
 *              i1=        1 + MOD2N(i1,64)
 *              j1=        1 + MOD2N(j1,64)
 *         P(3,ip)= P(3,ip)  + B(i1,j1)
 *         P(4,ip)= P(4,ip)  + C(i1,j1)
 *         P(1,ip)= P(1,ip)  + P(3,ip)
 *         P(2,ip)= P(2,ip)  + P(4,ip)
 *              i2= P(1,ip)
 *              j2= P(2,ip)
 *              i2=            MOD2N(i2,64)
 *              j2=            MOD2N(j2,64)
 *         P(1,ip)= P(1,ip)  + Y(i2+32)
 *         P(2,ip)= P(2,ip)  + Z(j2+32)
 *              i2= i2       + E(i2+32)
 *              j2= j2       + F(j2+32)
 *        H(i2,j2)= H(i2,j2) + 1.0
 * 13 CONTINUE
 */

for ( l=1 ; l<=loop ; l++ ) {
    for ( ip=0 ; ip<n ; ip++ ) {
        i1 = p[ip][0];
        j1 = p[ip][1];
        i1 &= 64-1;
        j1 &= 64-1;
        p[ip][2] += b[j1][i1];
        p[ip][3] += c[j1][i1];
        p[ip][0] += p[ip][2];
        p[ip][1] += p[ip][3];
        i2 = p[ip][0];
        j2 = p[ip][1];
        i2 = ( i2 & 64-1 ) - 1 ;
        j2 = ( j2 & 64-1 ) - 1 ;
        p[ip][0] += y[i2+32];
        p[ip][1] += z[j2+32];
        i2 += e[i2+32];
        j2 += f[j2+32];
        h[j2][i2] += 1.0;
```

```
    }
}
argument = 13;


/*
 ********************************************************************
 *   Kernel 14 -- 1-D PIC (Particle In Cell)
 ********************************************************************
 *    DO   14   L= 1,Loop
 *    DO   141  k= 1,n
 *          VX(k)= 0.0
 *          XX(k)= 0.0
 *          IX(k)= INT(  GRD(k))
 *          XI(k)= REAL( IX(k))
 *         EX1(k)= EX   ( IX(k))
 *        DEX1(k)= DEX  ( IX(k))
 *41  CONTINUE
 *    DO   142  k= 1,n
 *          VX(k)= VX(k) + EX1(k) + (XX(k) - XI(k))*DEX1(k)
 *          XX(k)= XX(k) + VX(k)  + FLX
 *          IR(k)= XX(k)
 *          RX(k)= XX(k) - IR(k)
 *          IR(k)= MOD2N(  IR(k),2048) + 1
 *          XX(k)= RX(k) + IR(k)
 *42  CONTINUE
 *    DO  14    k= 1,n
 *    RH(IR(k)  )= RH(IR(k)  ) + 1.0 - RX(k)
 *    RH(IR(k)+1)= RH(IR(k)+1) + RX(k)
 *14  CONTINUE
 */

for ( l=1 ; l<=loop ; l++ ) {
    for ( k=0 ; k<n ; k++ ) {
        vx[k] = 0.0;
        xx[k] = 0.0;
        ix[k] = (long) grd[k];
        xi[k] = (double) ix[k];
        ex1[k] = ex[ ix[k] - 1 ];
        dex1[k] = dex[ ix[k] - 1 ];
    }
    for ( k=0 ; k<n ; k++ ) {
        vx[k] = vx[k] + ex1[k] + ( xx[k] - xi[k] )*dex1[k];
        xx[k] = xx[k] + vx[k]  + flx;
        ir[k] = xx[k];
        rx[k] = xx[k] - ir[k];
        ir[k] = ( ir[k] & 2048-1 ) + 1;
        xx[k] = rx[k] + ir[k];
    }
    for ( k=0 ; k<n ; k++ ) {
        rh[ ir[k]-1 ] += 1.0 - rx[k];
```

```
            rh[ ir[k]   ] += rx[k];
        }
    }
    argument = 14;


    /*
     ********************************************************************
     *   Kernel 15 -- Casual Fortran.  Development version
     ********************************************************************
     *      DO 45  L = 1,Loop
     *             NG= 7
     *             NZ= n
     *             AR= 0.053
     *             BR= 0.073
     * 15  DO 45  j = 2,NG
     *      DO 45  k = 2,NZ
     *             IF( j-NG) 31,30,30
     * 30     VY(k,j)= 0.0
     *                GO TO 45
     * 31          IF( VH(k,j+1) -VH(k,j)) 33,33,32
     * 32           T= AR
     *                GO TO 34
     * 33           T= BR
     * 34          IF( VF(k,j) -VF(k-1,j)) 35,36,36
     * 35           R= MAX( VH(k-1,j), VH(k-1,j+1))
     *              S= VF(k-1,j)
     *                GO TO 37
     * 36           R= MAX( VH(k,j),   VH(k,j+1))
     *              S= VF(k,j)
     * 37     VY(k,j)= SQRT( VG(k,j)**2 +R*R)*T/S
     * 38          IF( k-NZ) 40,39,39
     * 39     VS(k,j)= 0.
     *                GO TO 45
     * 40          IF( VF(k,j) -VF(k,j-1)) 41,42,42
     * 41           R= MAX( VG(k,j-1), VG(k+1,j-1))
     *              S= VF(k,j-1)
     *              T= BR
     *                GO TO 43
     * 42           R= MAX( VG(k,j),   VG(k+1,j))
     *              S= VF(k,j)
     *              T= AR
     * 43     VS(k,j)= SQRT( VH(k,j)**2 +R*R)*T/S
     * 45    CONTINUE
     */

//Link the math library (libm.a) whe compiling for the sqrt function to work
//#pragma intrinsic sqrt

    for ( l=1 ; l<=loop ; l++ ) {
        ng = 7;
```

```
      nz = n;
      ar = 0.053;
      br = 0.073;
      for ( j=1 ; j<ng ; j++ ) {
          for ( k=1 ; k<nz ; k++ ) {
              if ( (j+1) >= ng ) {
                  vy[j][k] = 0.0;
                  continue;
              }
              if ( vh[j+1][k] > vh[j][k] ) {
                  t = ar;
              }
              else {
                  t = br;
              }
              if ( vf[j][k] < vf[j][k-1] ) {
                  if ( vh[j][k-1] > vh[j+1][k-1] )
                      r = vh[j][k-1];
                  else
                      r = vh[j+1][k-1];
                  s = vf[j][k-1];
              }
              else {
                  if ( vh[j][k] > vh[j+1][k] )
                      r = vh[j][k];
                  else
                      r = vh[j+1][k];
                  s = vf[j][k];
              }
//              vy[j][k] = sqrt( vg[j][k]*vg[j][k] + r*r )* t/s;
              if ( (k+1) >= nz ) {
                  vs[j][k] = 0.0;
                  continue;
              }
              if ( vf[j][k] < vf[j-1][k] ) {
                  if ( vg[j-1][k] > vg[j-1][k+1] )
                      r = vg[j-1][k];
                  else
                      r = vg[j-1][k+1];
                  s = vf[j-1][k];
                  t = br;
              }
              else {
                  if ( vg[j][k] > vg[j][k+1] )
                      r = vg[j][k];
                  else
                      r = vg[j][k+1];
                  s = vf[j][k];
                  t = ar;
              }
//              vs[j][k] = sqrt( vh[j][k]*vh[j][k] + r*r )* t / s;
```

```
        }
    }
}
argument = 15;


/*
 ********************************************************************
 *   Kernel 16 -- Monte Carlo search loop
 ********************************************************************
 *          II= n/3
 *          LB= II+II
 *          k2= 0
 *          k3= 0
 *    DO 485 L= 1,Loop
 *           m= 1
 *405       i1= m
 *410       j2= (n+n)*(m-1)+1
 *    DO 470 k= 1,n
 *          k2= k2+1
 *          j4= j2+k+k
 *          j5= ZONE(j4)
 *          IF( j5-n      ) 420,475,450
 *415       IF( j5-n+II   ) 430,425,425
 *420       IF( j5-n+LB   ) 435,415,415
 *425       IF( PLAN(j5)-R) 445,480,440
 *430       IF( PLAN(j5)-S) 445,480,440
 *435       IF( PLAN(j5)-T) 445,480,440
 *440       IF( ZONE(j4-1)) 455,485,470
 *445       IF( ZONE(j4-1)) 470,485,455
 *450       k3= k3+1
 *          IF( D(j5)-(D(j5-1)*(T-D(j5-2))**2+(S-D(j5-3))**2
 *   .                        +(R-D(j5-4))**2)) 445,480,440
 *455        m= m+1
 *          IF( m-ZONE(1) ) 465,465,460
 *460        m= 1
 *465       IF( i1-m) 410,480,410
 *470 CONTINUE
 *475 CONTINUE
 *480 CONTINUE
 *485 CONTINUE
 */

ii = n / 3;
lb = ii + ii;
k3 = k2 = 0;
for ( l=1 ; l<=loop ; l++ ) {
    i1 = m = 1;
    label410:
    j2 = ( n + n )*( m - 1 ) + 1;
    for ( k=1 ; k<=n ; k++ ) {
```

```
        k2++;
        j4 = j2 + k + k;
        j5 = zone[j4-1];
        if ( j5 < n ) {
            if ( j5+1b < n ) {              /* 420 */
                tmp = plan[j5-1] - t;       /* 435 */
            } else {
                if ( j5+ii < n ) {          /* 415 */
                    tmp = plan[j5-1] - s;   /* 430 */
                } else {
                    tmp = plan[j5-1] - r;   /* 425 */
                }
            }
        } else if( j5 == n ) {
            break;                          /* 475 */
        } else {
            k3++;                           /* 450 */
            tmp=(d[j5-1]-(d[j5-2]*(t-d[j5-3])*(t-d[j5-3])+(s-d[j5-4])*
                        (s-d[j5-4])+(r-d[j5-5])*(r-d[j5-5])));
        }
        if ( tmp < 0.0 ) {
            if ( zone[j4-2] < 0 )           /* 445 */
                continue;                   /* 470 */
            else if ( !zone[j4-2] )
                break;                      /* 480 */
        } else if ( tmp ) {
            if ( zone[j4-2] > 0 )           /* 440 */
                continue;                   /* 470 */
            else if ( !zone[j4-2] )
                break;                      /* 480 */
        } else break;                       /* 485 */
        m++;                                /* 455 */
        if ( m > zone[0] )
            m = 1;                          /* 460 */
        if ( i1-m )                         /* 465 */
            goto label410;
        else
            break;
    }
}
argument = 16;


/*
 ******************************************************************
 *   Kernel 17 -- implicit, conditional computation
 ******************************************************************
 *          DO 62 L= 1,Loop
 *                i= n
 *                j= 1
 *              INK= -1
```

```
*            SCALE= 5./3.
*              XNM= 1./3.
*               E6= 1.03/3.07
*                   GO TO 61
*60             E6= XNM*VSP(i)+VSTP(i)
*          VXNE(i)= E6
*              XNM= E6
*           VE3(i)= E6
*                i= i+INK
*               IF( i.EQ.j) GO TO  62
*61             E3= XNM*VLR(i) +VLIN(i)
*            XNEI= VXNE(i)
*          VXND(i)= E6
*              XNC= SCALE*E3
*               IF( XNM .GT.XNC) GO TO  60
*               IF( XNEI.GT.XNC) GO TO  60
*           VE3(i)= E3
*               E6= E3+E3-XNM
*          VXNE(i)= E3+E3-XNEI
*              XNM= E6
*                i= i+INK
*               IF( i.NE.j) GO TO 61
* 62 CONTINUE
*/

   for ( l=1 ; l<=loop ; l++ ) {
      i = n-1;
      j = 0;
      ink = -1;
      scale = 5.0 / 3.0;
      xnm = 1.0 / 3.0;
      e6 = 1.03 / 3.07;
      goto l61;
l60:  e6 = xnm*vsp[i] + vstp[i];
      vxne[i] = e6;
      xnm = e6;
      ve3[i] = e6;
      i += ink;
      if ( i==j ) goto l62;
l61:  e3 = xnm*vlr[i] + vlin[i];
      xnei = vxne[i];
      vxnd[i] = e6;
      xnc = scale*e3;
      if ( xnm > xnc ) goto l60;
      if ( xnei > xnc ) goto l60;
      ve3[i] = e3;
      e6 = e3 + e3 - xnm;
      vxne[i] = e3 + e3 - xnei;
      xnm = e6;
      i += ink;
      if ( i != j ) goto l61;
```

```
162:;
    }

    argument = 17;


    /*
     ********************************************************************
     *   Kernel 18 - 2-D explicit hydrodynamics fragment
     ********************************************************************
     *       DO 75  L= 1,Loop
     *              T= 0.0037
     *              S= 0.0041
     *             KN= 6
     *              JN= n
     *       DO 70  k= 2,KN
     *       DO 70  j= 2,JN
     *        ZA(j,k)= (ZP(j-1,k+1)+ZQ(j-1,k+1)-ZP(j-1,k)-ZQ(j-1,k))
     *   .            *(ZR(j,k)+ZR(j-1,k))/(ZM(j-1,k)+ZM(j-1,k+1))
     *        ZB(j,k)= (ZP(j-1,k)+ZQ(j-1,k)-ZP(j,k)-ZQ(j,k))
     *   .            *(ZR(j,k)+ZR(j,k-1))/(ZM(j,k)+ZM(j-1,k))
     * 70    CONTINUE
     *       DO 72  k= 2,KN
     *       DO 72  j= 2,JN
     *        ZU(j,k)= ZU(j,k)+S*(ZA(j,k)*(ZZ(j,k)-ZZ(j+1,k))
     *   .                       -ZA(j-1,k) *(ZZ(j,k)-ZZ(j-1,k))
     *   .                       -ZB(j,k)   *(ZZ(j,k)-ZZ(j,k-1))
     *   .                       +ZB(j,k+1) *(ZZ(j,k)-ZZ(j,k+1)))
     *        ZV(j,k)= ZV(j,k)+S*(ZA(j,k)*(ZR(j,k)-ZR(j+1,k))
     *   .                       -ZA(j-1,k) *(ZR(j,k)-ZR(j-1,k))
     *   .                       -ZB(j,k)   *(ZR(j,k)-ZR(j,k-1))
     *   .                       +ZB(j,k+1) *(ZR(j,k)-ZR(j,k+1)))
     * 72    CONTINUE
     *       DO 75  k= 2,KN
     *       DO 75  j= 2,JN
     *        ZR(j,k)= ZR(j,k)+T*ZU(j,k)
     *        ZZ(j,k)= ZZ(j,k)+T*ZV(j,k)
     * 75    CONTINUE
     */

    for ( l=1 ; l<=loop ; l++ ) {
        t = 0.0037;
        s = 0.0041;
        kn = 6;
        jn = n;
        for ( k=1 ; k<kn ; k++ ) {
#pragma nohazard
          for ( j=1 ; j<jn ; j++ ) {
              za[k][j] = ( zp[k+1][j-1] +zq[k+1][j-1] -zp[k][j-1] -zq[k][j-1] )*
                         ( zr[k][j] +zr[k][j-1] ) / ( zm[k][j-1] +zm[k+1][j-1]);
              zb[k][j] = ( zp[k][j-1] +zq[k][j-1] -zp[k][j] -zq[k][j] ) *
```

```
                    ( zr[k][j] +zr[k-1][j] ) / ( zm[k][j] +zm[k][j-1]);
          }
        }
        for ( k=1 ; k<kn ; k++ ) {
#pragma nohazard
            for ( j=1 ; j<jn ; j++ ) {
                zu[k][j] += s*( za[k][j]   *( zz[k][j] - zz[k][j+1] ) -
                                za[k][j-1] *( zz[k][j] - zz[k][j-1] ) -
                                zb[k][j]   *( zz[k][j] - zz[k-1][j] ) +
                                zb[k+1][j] *( zz[k][j] - zz[k+1][j] ) );
                zv[k][j] += s*( za[k][j]   *( zr[k][j] - zr[k][j+1] ) -
                                za[k][j-1] *( zr[k][j] - zr[k][j-1] ) -
                                zb[k][j]   *( zr[k][j] - zr[k-1][j] ) +
                                zb[k+1][j] *( zr[k][j] - zr[k+1][j] ) );
            }
        }
        for ( k=1 ; k<kn ; k++ ) {
#pragma nohazard
            for ( j=1 ; j<jn ; j++ ) {
                zr[k][j] = zr[k][j] + t*zu[k][j];
                zz[k][j] = zz[k][j] + t*zv[k][j];
            }
        }
    }
    argument = 18;


    /*
     ******************************************************************
     *   Kernel 19 -- general linear recurrence equations
     ******************************************************************
     *              KB5I= 0
     *           DO 194 L= 1,Loop
     *           DO 191 k= 1,n
     *         B5(k+KB5I)= SA(k) +STB5*SB(k)
     *              STB5= B5(k+KB5I) -STB5
     *191        CONTINUE
     *192        DO 193 i= 1,n
     *                k= n-i+1
     *         B5(k+KB5I)= SA(k) +STB5*SB(k)
     *              STB5= B5(k+KB5I) -STB5
     *193        CONTINUE
     *194 CONTINUE
     */

    kb5i = 0;
    for ( l=1 ; l<=loop ; l++ ) {
        for ( k=0 ; k<n ; k++ ) {
            b5[k+kb5i] = sa[k] + stb5*sb[k];
            stb5 = b5[k+kb5i] - stb5;
        }
```

```
    for ( i=1 ; i<=n ; i++ ) {
        k = n - i ;
        b5[k+kb5i] = sa[k] + stb5*sb[k];
        stb5 = b5[k+kb5i] - stb5;
    }
}
argument = 19;


/*
 ********************************************************************
 *   Kernel 20 -- Discrete ordinates transport, conditional recurrence on xx
 ********************************************************************
 *    DO 20 L= 1,Loop
 *    DO 20 k= 1,n
 *         DI= Y(k)-G(k)/( XX(k)+DK)
 *         DN= 0.2
 *         IF( DI.NE.0.0) DN= MAX( S,MIN( Z(k)/DI, T))
 *       X(k)= ((W(k)+V(k)*DN)* XX(k)+U(k))/(VX(k)+V(k)*DN)
 *    XX(k+1)= (X(k)- XX(k))*DN+ XX(k)
 * 20 CONTINUE
 */

for ( l=1 ; l<=loop ; l++ ) {
    for ( k=0 ; k<n ; k++ ) {
        di = y[k] - g[k] / ( xx[k] + dk );
        dn = 0.2;
        if ( di ) {
            dn = z[k]/di ;
            if ( t < dn ) dn = t;
            if ( s > dn ) dn = s;
        }
        x[k] = ( ( w[k] + v[k]*dn )* xx[k] + u[k] ) / ( vx[k] + v[k]*dn );
        xx[k+1] = ( x[k] - xx[k] )* dn + xx[k];
    }
}
argument = 20;


/*
 ********************************************************************
 *   Kernel 21 -- matrix*matrix product
 ********************************************************************
 *    DO 21 L= 1,Loop
 *    DO 21 k= 1,25
 *    DO 21 i= 1,25
 *    DO 21 j= 1,n
 *    PX(i,j)= PX(i,j) +VY(i,k) * CX(k,j)
 * 21 CONTINUE
 */
```

```
    for ( l=1 ; l<=loop ; l++ ) {
        for ( k=0 ; k<25 ; k++ ) {
            for ( i=0 ; i<25 ; i++ ) {
#pragma nohazard
                for ( j=0 ; j<n ; j++ ) {
                    px[j][i] += vy[k][i] * cx[j][k];
                }
            }
        }
    }
    argument = 21;


    /*
     ******************************************************************
     *   Kernel 22 -- Planckian distribution
     ******************************************************************
     *     EXPMAX= 20.0
     *       U(n)= 0.99*EXPMAX*V(n)
     *    DO 22 L= 1,Loop
     *    DO 22 k= 1,n
     *                                          Y(k)= U(k)/V(k)
     *       W(k)= X(k)/( EXP( Y(k)) -1.0)
     * 22 CONTINUE
     */

//Link the math library (libm.a) whe compiling for the sqrt function to work

//#pragma intrinsic exp

    expmax = 20.0;
    u[n-1] = 0.99*expmax*v[n-1];
    for ( l=1 ; l<=loop ; l++ ) {
        for ( k=0 ; k<n ; k++ ) {
            y[k] = u[k] / v[k];
            w[k] = x[k] / ( exp( y[k] ) -1.0 );
              w[k] = x[k] / ( y[k] -1.0 );
        }
    }
    argument = 22;


    /*
     ******************************************************************
     *   Kernel 23 -- 2-D implicit hydrodynamics fragment
     ******************************************************************
     *    DO 23  L= 1,Loop
     *    DO 23  j= 2,6
     *    DO 23  k= 2,n
     *          QA= ZA(k,j+1)*ZR(k,j) +ZA(k,j-1)*ZB(k,j) +
     *   .          ZA(k+1,j)*ZU(k,j) +ZA(k-1,j)*ZV(k,j) +ZZ(k,j)
```

```
     * 23  ZA(k,j)= ZA(k,j) +.175*(QA -ZA(k,j))
     */

    for ( l=1 ; l<=loop ; l++ ) {
        for ( j=1 ; j<6 ; j++ ) {
            for ( k=1 ; k<n ; k++ ) {
                qa = za[j+1][k]*zr[j][k] + za[j-1][k]*zb[j][k] +
                     za[j][k+1]*zu[j][k] + za[j][k-1]*zv[j][k] + zz[j][k];
                za[j][k] += 0.175*( qa - za[j][k] );
            }
        }
    }
    argument = 23;


    /*
     ********************************************************************
     *   Kernel 24 -- find location of first minimum in array
     ********************************************************************
     *     X( n/2)= -1.0E+10
     *    DO 24  L= 1,Loop
     *           m= 1
     *    DO 24  k= 2,n
     *          IF( X(k).LT.X(m))  m= k
     * 24 CONTINUE
     */

    x[n/2] = -1.0e+10;
    for ( l=1 ; l<=loop ; l++ ) {
        m = 0;
        for ( k=1 ; k<n ; k++ ) {
            if ( x[k] < x[m] ) m = k;
        }
    }
    argument = 24;

}
```

Apéndice B

Problemas Encontrados

En esta sección se describirán los problemas encontrados a lo largo de esta tesis, así como también sus soluciones en caso de existir.

B.1. OMAP1610

En la primera etapa del desarrollo de esta tesis, el principal problema fue la falta de documentación de la tarjeta de desarrollo, debido a que dicha tarjeta está disponible solamente para desarrolladores de equipo inalámbrico de muy alto volumen, por lo cual la información es escasa y muy restringida. Aunado a esto, la empresa que desarrolló estas tarjetas ya no existía al momento de empezar a trabajar en esta tesis, por lo cual fue imposible solicitar documentación a esta empresa.

Otro problema que se tuvo y que es relacionado a la exclusividad de esta tarjeta fue la dificultad para conseguir las herramientas necesarias para cargar el sistema por primera vez, ya que son herramientas propietarias y de muy alto costo.

B.2. Linux

La información de referencia para crear un filesystem para la OMAP suponía el uso de un kernel de linux obsoleto (v2.4) el cual no utiliza udev, por lo cuál fue necesario investigar la implementación y funcionamiento de udev y crear la estructura de archivos necesarios para el funcionamiento de udev, además de los scripts de arranque, lo cuál requirió de mucha prueba y error.

B.2.1. Profiler

El software libre disponible para realizar pruebas de desempeño (gprof) en procesadores ARM al parecer no funciona correctamente, debido a que sin importar qué tipo de programa se ejecutara, o el tiempo que tardara en dicha ejecución, no regresaba datos de tiempos de ejecución, por lo cual no fue posible realizar alguna prueba de desempeño a la OMAP 1610.

Esto fue solucionado en parte utilizando una tarjeta de desarrollo Beagleboard, la cual está basada en un procesador OMAP3530 en conjunto de una distribución de linux especialmente diseñada para esta tarjeta, la cuál incluye un software para realizar pruebas de desempeño más reciente (oprof) cuyo funcionamiento está íntimamente relacionado a la configuración del kernel, por lo cuál su implementación en la OMAP 1610 no es algo trivial.

B.3. Seamless

Debido al problema anteriormente mencionado, antes de contar con la nueva tarjeta Beagleboard, se optó por utilizar el software de co-diseño Seamless de Mentor Graphics para determinar las pruebas de desempeño, utilizando para esto una implementación virtual de un procesador ARM 926ej-s. Sin embargo, la documentación de este software es muy pobre y no todos los módulos de hardware están documentados, por lo cuál es muy difícil saber con exactitud que función realizan. Además el software es muy complejo y complicado de usar.

Un problema que surgió en Seamless al ejecutar el simulador de software XRAY es que no quería correr y marcaba un error. Este error se solucionó al comentar una línea en el script de ejecución del simulador.

B.4. DSP/BIOS Bridge

Debido a la naturaleza de este procesador, es muy difícil realizar pruebas comparativas del desempeño del sistema de intercomunicación entre el GPP y el DSP (DSP/BIOS) de la OMAP y la intercomunicación usada en este trabajo, por tanto, las únicas métricas de desempeño presentadas son aquellas realizadas en el GPP (ARM).

Bibliografía

[1] O. Florescu, "Dsp mcu bridge -technical memorandum-," Master's thesis, "Politehnica" University of Bucharest Computer Science Engineering Department, June 2002.

[2] T. Instruments, "Developing core software technologies for ti's omap platform," *SWPY006 - White Paper*, 2002.

[3] J. Chaoui, "Omap : Enabling multimedia applications in third generation (3g) wireless terminals," *Dedicated Systems Magazine*, 2001. [Online]. Available: http://www.dedicated-systems.com

[4] *CCS Integration Guide for SD XDS510PP+*, version 1.0 ed., Productivity Systems Inc., September 2003.

[5] *IBoot Host Users Guide*, version 1.1 ed., Productivity Systems Inc., October 2003.

[6] T. Kobayashi, *Linux DSP Gateway Specification*, 3rd ed., Nokia Corporation, Nov 2004.

[7] *Hardware-software Codesign of Multimedia Embedded Systems: the PeaCE Approach*, The CAP Laboratory, Seoul National University. IEEE, 09 2006.

[8] ARM. (2009, 04) Arm products and solutions. [Online]. Available: http://www.arm.com/miscPDFs/3823.pdf

[9] R. Wilson. (1988, Nov) Some facts about the acorn risc machine. posting to comp.arch.

[10] Arm architecture. Wikipedia the free encyclopedia. [Online]. Available: http://en.wikipedia.org/wiki/ARM_architecture

[11] Historia de linux. Wikipedia la enciclopedia libre. [Online]. Available: http://es.wikipedia.org/wiki/Historia_de_Linux

[12] U. de las Palmas de Gran Canaria. ¿qué es linux? [Online]. Available: http://labsopa.dis.ulpgc.es/aso-itis/tutorial/CONTENIDOS/LINUX1/LINUX1.htm

[13] W. Denk and D. Zundel. U-boot. [Online]. Available: http://www.denx.de

[14] Sourcery g++ lite. CodeSourcery. [Online]. Available: http://www.codesourcery.com/

[15] I. Linux Kernel Organization. The linux kernel archives. Linux Kernel Organization, Inc. [Online]. Available: http://www.kernel.org/

[16] Unofficial omap linux patches. [Online]. Available: http://www.muru.com/linux/omap/

[17] D. Vlasenko. Busybox. [Online]. Available: http://www.busybox.net/

[18] G. Kroah-Hartman and K. Sievers. udev. [Online]. Available: http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html

[19] Omap development tools. Spectrum Digital. [Online]. Available: http://www.omap.spectrumdigital.com/

[20] Texas Instruments. [Online]. Available: http://www.ti.com

[21] Seamless. Mentor Graphics. [Online]. Available: http://www.mentor.com/products/fv/seamless/