



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Adaptación de GNU Radio a procesadores KeyStone II
66AK2xxx

AUTOR: Ernesto Alonso Gutiérrez

TITULACIÓN: Grado en Ingeniería Electrónica de Comunicaciones

TUTOR (o Director en su caso): Pedro José Lobo Perea

DEPARTAMENTO: Sistemas electrónicos y de control

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Eduardo Nogueira Díaz

VOCAL: Pedro José Lobo Perea

SECRETARIO: Eduardo Juárez Martínez

Fecha de lectura:

Calificación:

El Secretario,

Resumen

En el presente proyecto se ha procedido a implantar la herramienta de procesamiento software GNU Radio en la tarjeta EVMK2H, que es un módulo de evaluación fabricado por Texas Instruments que incorpora un *System on Chip* (SoC) 66AK2H14 de la familia Keystone II, el cual dispone de 4 núcleos ARM y 8 núcleos DSP.

Previamente a la instalación de GNU Radio, hubo que configurar la tarjeta, así como instalar el software necesario. De igual manera, se realizó una primera aproximación para comprender el funcionamiento de los sistemas de comunicación entre núcleos de que hace uso la tarjeta, y de los que se hizo uso posteriormente en el proyecto.

Tras el portado de GNU Radio se ha comprobado el correcto funcionamiento del mecanismo de comunicación entre núcleos ARM y DSP con un par de aplicaciones de prueba.

Abstract

In the present project it was performed the implementation of the software processing toolkit GNU Radio into the EVMK2H board, which is an evaluation module from Texas Instruments that includes a 66AK2H14 *System on Chip* (SoC) from the Keystone II family, that provides 4 ARM cores and 8 DSP cores.

Before installing GNU Radio, it was necessary to configure the board, and as well installing other needed software. Also, a first approach was performed to understand the way the communication system between cores included in the board works, which was used later in the project.

After porting GNU Radio, some test applications have been written to test the correct operation of the communication mechanism between ARM and DSP cores.

A Pedro, por su paciencia y buena
disposición a la hora de resolver las muchas
dudas que me han surgido durante la
realización de este proyecto.

A mis padres, por su preocupación y por
escucharme.

A los electrónicos (y algún que otro
infiltrado) por hacer de esta etapa de mi
vida algo inolvidable cargado de buenos
recuerdos.

Contenido

Indice de figuras.....	- 7 -
Listado de abreviaturas.....	- 8 -
1. Introducción.....	- 9 -
1.1 Antecedentes.....	- 9 -
1.2 Objetivos.....	- 9 -
1.3. Metodología.....	- 10 -
1.4 Organización del documento.....	- 11 -
2. Herramientas y Entorno de Trabajo.....	- 13 -
2.1 Módulo de evaluación EVMK2H.....	- 13 -
2.1.1 Sistemas Operativos.....	- 15 -
2.1.2 Arranque de la tarjeta.....	- 15 -
2.1.3 Comunicación entre núcleos.....	- 16 -
2.2 Emulador XDS200.....	- 17 -
2.3 Entorno de desarrollo.....	- 18 -
2.3.1 Software.....	- 18 -
3. Conexión e instalación.....	- 21 -
3.1 Conexión y arranque.....	- 21 -
3.2 Testeo del módulo de evaluación.....	- 27 -
3.2.1 Multi processor manager (MPM).....	- 28 -
3.2.2 MSGCOM.....	- 29 -
3.3 Instalación de GNU Radio.....	- 36 -
3.3.1 Instalación de paquetes externos.....	- 36 -
4. Trabajo realizado.....	- 41 -
4.1 Descripción del código existente.....	- 41 -
4.1.1 Código para ARM.....	- 41 -
4.1.2 Código para DSP.....	- 44 -
4.2 Adaptación del código a la tarjeta EVMK2H.....	- 45 -
4.2.1 Código para ARM.....	- 45 -
4.2.2 Código para los DSP.....	- 49 -

4.3 Modificaciones realizadas	- 50 -
4.3.1 Modificaciones en el código	- 50 -
4.3.2 Modificaciones en los bloques	- 53 -
4.3.3 Modificaciones en la compilación.....	- 54 -
4.4 Pruebas de rendimiento	- 56 -
4.4.1 Grafos de prueba	- 56 -
4.4.2 Resultados obtenidos.....	- 57 -
5. Conclusiones y trabajo futuro	- 59 -
5.1 Conclusiones.....	- 59 -
5.2 Trabajo futuro.....	- 60 -
Bibliografía.	- 61 -

Indice de figuras

Figura 1. Plataforma de desarrollo EVMK2H.....	- 13 -
Figura 2. Comparativa entre núcleos de OMAP3530 (IGEPv2) y 66AK2H14 (EVMK2H), extraída de la información proporcionada por TI.....	- 14 -
Figura 3. Modos de arranque.....	- 16 -
Figura 4. Emulador XDS200 una vez acoplado a la tarjeta EVMK2H.....	- 18 -
Figura 5. Ejemplo de aplicación en GNU Radio	- 20 -
Figura 6. Esquema de la conexión.....	- 21 -
Figura 7. Configuración de gtkterm para la conexión por USB.....	- 22 -
Figura 8. Detalle de los switches, extraído del manual proporcionado por Advantech [4]	- 24 -
Figura 9. Parámetros de arranque de u-boot	- 25 -
Figura 10. Captura de la nueva versión de u-boot (La anterior databa del 16/08/13).....	- 26 -
Figura 11. Login una vez finalizado el proceso de arranque.....	- 27 -
Figura 12. Fragmento de la lista de paquetes instalados	- 28 -
Figura 13. Proyectos para los DSP, una vez instalados	- 30 -
Figura 14. Inclusión de paquetes requeridos	- 31 -
Figura 15. Inclusión de las rutas donde se encuentran las librerías.....	- 32 -
Figura 16. Creación del target	- 33 -
Figura 17. Inclusión de la plataforma creada en el proyecto	- 34 -
Figura 18. Creación de la sesión de debug	- 35 -
Figura 19. Grafo de GNU Radio	- 42 -
Figura 21. Resultado de la compilación y el enlazado	- 55 -
Figura 22. Grafo ARM	- 56 -
Figura 23. Grafo DSP	- 57 -

Listado de abreviaturas

ARM	Familia de procesadores de la marca del mismo nombre
CCS	Code Composer Studio
DHCP	Dynamic Host Configuration Protocol
DSP	Digital Signal Processor
MCSDK	MultiCore Software Development Kit
NFS	Network File System
OPKG	Open Package Management
SO	Sistema Operativo
SoC	System on Chip
TFTP	Trivial File Transfer Protocol
TI	Texas Instruments

1

Introducción

1.1 Antecedentes

El presente Proyecto Fin de Grado es la continuación de otros trabajos realizados en el Grupo de Diseño Electrónico y Microelectrónico (GDEM), dentro del departamento de Sistemas Electrónicos y de Control (SEC), cuyo objetivo es investigar en arquitecturas multiprocesador para sistemas de comunicaciones basados en radio software.

Dentro de esta línea de investigación se han desarrollado ya varios Proyectos Fin de Carrera [1-3] en los que se ha implementado un receptor DVB-H utilizando la librería GNU Radio, se ha portado GNU Radio al procesador ARM que contienen los sistemas DaVinci DM6446 y OMAP 3530 de Texas Instruments, y se ha extendido GNU Radio para que emplee el procesador digital de señal (DSP) que incorporan estos últimos sistemas trabajando conjuntamente con el ARM.

1.2 Objetivos

El objetivo principal de este proyecto, que es continuación a los trabajos mencionados, es portar la herramienta GNU Radio a la familia de sistemas KeyStone II 66AK2xxx, que integran entre 2 y 12 núcleos ARM Cortex A15 y DSP C66x. El desarrollo se ha realizado en una tarjeta basada en el sistema 66AK2H14, compuesto por 4 núcleos ARM y 8 núcleos DSP.

Una vez portada la herramienta, se realizó una adaptación de la aplicación de prueba creada con anterioridad para comprobar el correcto funcionamiento de GNU Radio.

Para realizar esta adaptación será necesario actualizar el código actualmente disponible, y esto se debe a dos motivos principalmente:

- Por un lado, los DSPs de los sistemas KeyStone II utilizan una nueva versión del sistema operativo de tiempo real DSP/BIOS denominada SYS/BIOS, y también se han actualizado los sistemas de comunicación entre los núcleos DSP y los núcleos ARM, respecto a los disponibles en el OMAP 3530, por lo que habrá que modificar el código disponible para hacerlo compatible con las nuevas librerías que se emplean para establecer la comunicación entre los núcleos.
- Por otro lado, los DSPs C66x empleados en los nuevos sistemas son capaces de operar con datos en punto flotante, mientras que el C64x+ que empleaba el OMAP 3530 era de punto fijo. Esta característica facilita enormemente el traslado de tareas entre los núcleos ARM y DSP, ya que elimina la necesidad de realizar una conversión de formatos.

1.3. Metodología

La metodología de trabajo que se ha seguido para llevar a cabo el proyecto se puede dividir en las siguientes fases:

- Estudio del trabajo que se ha llevado a cabo hasta el momento.
- Aprendizaje de las herramientas necesarias, así como de otros conceptos:
 - Lenguajes de programación C++ y Python.
 - Entorno de desarrollo para DSP Code Composer Studio.
 - Librerías y módulos nuevos para la comunicación entre núcleos
 - Entorno de desarrollo de radio software GNU Radio.
- Configuración inicial de la tarjeta, y realización de pruebas para verificar su funcionamiento.
- Portado de GNU Radio al nuevo sistema multiprocesador 66AK2H14.
 - Adaptación de DSP/BIOS a las nuevas librerías y módulos
 - Implementación de algunos bloques de procesamiento de señal.
- Redacción de la memoria del proyecto, detallando los resultados obtenidos.

1.4 Organización del documento

En los posteriores capítulos se proporcionará información sobre cómo se ha desarrollado el presente proyecto.

En el capítulo 2 se detallan las características de los sistemas Keystone II y de la tarjeta empleada, así como el resto de herramientas necesarias para hacer uso de la tarjeta y realizar la adaptación del código.

En el capítulo 3 se han documentado los pasos necesarios para realizar la puesta a punto y configuración de la tarjeta, y las posteriores comprobaciones para corroborar su correcto funcionamiento.

En el capítulo 4 se describe cómo se ha procedido para realizar la adaptación del código disponible a las nuevas librerías de que hace uso la tarjeta, indicando las modificaciones realizadas.

En el capítulo 5, por último, se detallan las conclusiones obtenidas tras la realización de este proyecto, aquellos inconvenientes encontrados durante el desarrollo del mismo y posibles líneas de trabajo para el futuro.

2

Herramientas y Entorno de Trabajo

Una vez situado el contexto en el que se enmarca el proyecto, se van a describir las herramientas empleadas, así como el entorno de desarrollo por el que se ha optado.

2.1 Módulo de evaluación EVMK2H

En el anterior proyecto realizado en la presente línea de trabajo [3], se optó por utilizar la tarjeta IGEPv2 de ISEE como plataforma de desarrollo. Para este proyecto se ha elegido la tarjeta EVMK2H [4], fabricada por Advantech. Se trata de un modulo de evaluación que contiene el SoC 66AK2H14 [5], de la familia Keystone II, de Texas Instruments (TI).



Figura 1. Plataforma de desarrollo EVMK2H

Las principales características de este módulo de evaluación son:

- CPU:
 - 4 núcleos ARM – Cortex A15 (1400 MHz)
 - 8 núcleos DSP TMS320C66x (1200 MHz)
- Memoria:
 - 2 GB ECC DDR3 integrada
 - 2 GB ECC DDR3 1333 SO-DIMM
 - 512 MB NAND Flash
 - 128 Mb NOR Flash
 - 1Mb I2C EEPROM
- 2 puertos Ethernet
- Puerto USB integrado
- Conector para uSIM

En comparación con las características de la tarjeta IGEPv2, cuyo SoC incorporaba un único núcleo ARM-CortexA8, el procesador 66AK2H14 que incorpora la tarjeta EVMK2H, al disponer de cuatro núcleos ARM del tipo A15, puede procesar a una velocidad sensiblemente mayor, permitiendo que pueda correr aplicaciones más complejas. [6]

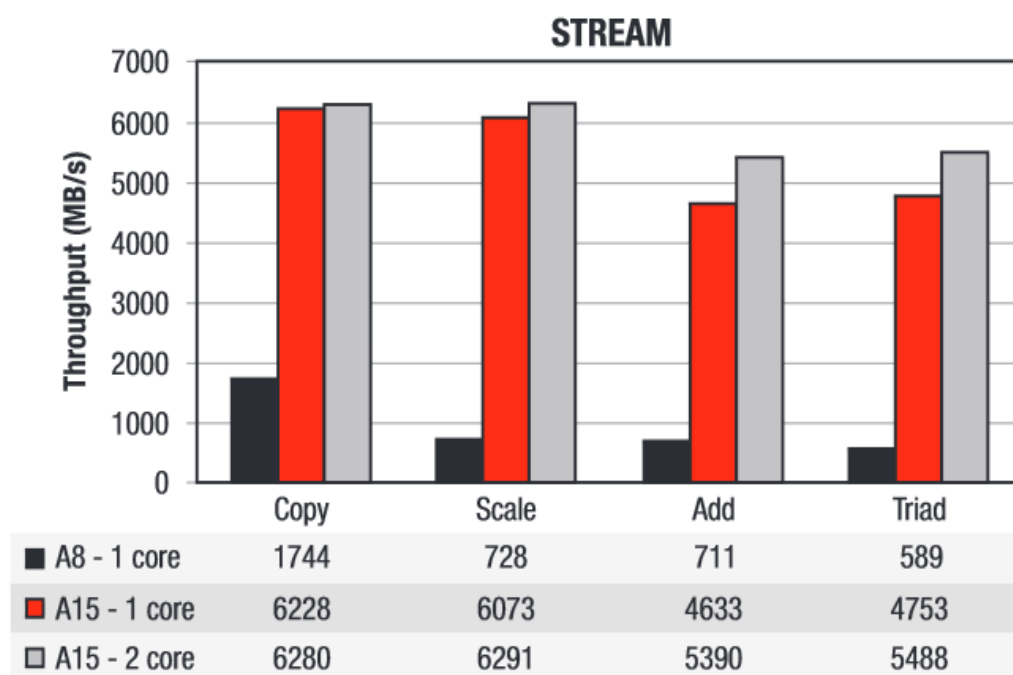


Figura 2. Comparativa entre núcleos de OMAP3530 (IGEPv2) y 66AK2H14 (EVMK2H), extraída de la información proporcionada por TI

Así mismo, el trabajo realizado anteriormente ya se desarrolló pensando en que se dispusiera de varios DSP trabajando en paralelo, pese a que la tarjeta IGEPv2 empleada solo disponía únicamente de uno de estos núcleos. De ahí que sea de interés emplear en este proyecto la tarjeta EVMK2H, ya que dispone de 8 DSP.

Esto hace que este modulo de evaluación sea una opción muy interesante, pues el fin último de la línea de trabajo en la que se enmarca este proyecto es la de crear una plataforma capaz de correr el receptor de DVB-H desarrollado con anterioridad [1], y esta tarjeta puede reunir, a priori, los requisitos necesarios.

2.1.1 Sistemas Operativos

En cuanto a sistemas operativos (SO), en la parte de ARM se utiliza una distribución de Linux adaptada a las características y limitaciones de la tarjeta. Concretamente se ha usado la distribución ‘Arago-Project’ proporcionada en el paquete ‘Multicore Software Development Kit (MCSDK)’, del cual trataremos en detalle más adelante.

Arago-project es un SO gratuito creado por Texas Instruments, para incorporarlo a los kits de desarrollo de los diferentes procesadores ARM y SoC, que utiliza un sistema de paquetes. Éste SO está construido sobre otro SO llamado Yocto-project, que a su vez se basa en OpenEmbedded, que es un entorno Linux para sistemas embebidos. [7]

En los núcleos DSP, se emplea SYS/BIOS, un SO en tiempo real, multitarea y escalable diseñado específicamente por TI para los núcleos de la familia TMS320C6000, para dar soporte a las aplicaciones que corran en estos núcleos.

2.1.2 Arranque de la tarjeta

La tarjeta dispone de cinco modos diferentes de iniciar el arranque, listados en la figura 3, pudiendo elegir el usuario el más conveniente para su propósito a través de unos switches incluidos en la tarjeta.

DIP Switch (p1, p2, p3, p4)	Bootmode
0000	ARM NAND
0001	DSP no-boot
0010	ARM SPI
0011	ARM I2C
0100	ARM UART

Figura 3. Modos de arranque

En el caso de que se opte por arrancar la tarjeta cargando u-boot (modos 0, 2, 3 y 4), posteriormente se puede cargar el SO Linux en el ARM de tres maneras diferentes:

- Cargando el kernel a través de TFTP y el sistema de archivos por RAMFS
- Cargando el kernel a través de TFTP y el sistema de archivos por NFS
- A través de UBIFS

2.1.3 Comunicación entre núcleos

Otra de las ventajas que aportan los SoC de la familia Keystone II es que permite una comunicación más efectiva entre los diferentes elementos integrados en él, conseguida gracias a la optimización de recursos que aporta el ‘Multicore Navigator’. Se trata de un novedoso método de comunicación, que reemplaza a DSPLink, que es la herramienta que se empleaba anteriormente para realizar la comunicación entre núcleos, como es el caso de la tarjeta IGEPv2 que se empleó en el proyecto anterior. Este cambio es bastante radical pues ambas herramientas no comparten nada, por lo que los módulos de los que hacen uso son también del todo diferentes.

Multicore Navigator se basa en la comunicación por paquetes y la gestión de las colas a las que son enviados estos paquetes. Gracias a esto el tiempo de ejecución de la tarea de comunicación queda reducido sensiblemente, en comparación a como ocurría en sistemas que no disponían de Multicore Navigator.

Otra de sus características, es su versatilidad, pues permite establecer la comunicación de diversas maneras, atendiendo a varios aspectos, como el canal utilizado, cómo realiza las interrupciones, o si la comunicación es bloqueante o no.

- Tipos de canal:
 - Canal de cola simple. Los mensajes se envían directamente al destino.
 - Canal virtual. Varios canales virtuales asociados a una única cola.
 - Canal de cola DMA. Los mensajes se copian utilizando la infraestructura PKTDMA.
 - Canal de cola proxy. Permite la comunicación entre dos núcleos que no comparten el mismo Multicore Navigator, es decir, de tarjetas separadas.
- Tipos de interrupción:
 - Sin interrupción. El lector comprueba constantemente (polling).
 - Interrupción directa. Tiene una latencia más baja.
 - Interrupciones acumuladas. El lector recibe una interrupción cuando tiene un número determinado de mensajes en cola.

2.2 Emulador XDS200

Para cargar archivos directamente en los núcleos DSP de la tarjeta, Advantech proporciona este emulador, fabricado por TI, que irá acoplado a la misma (emulador on-board), y que es el requerido por defecto por las herramientas de desarrollo.



Figura 4. Emulador XDS200 una vez acoplado a la tarjeta EVMK2H

2.3 Entorno de desarrollo

Para llevar a cabo el presente proyecto, aparte de la tarjeta y el emulador descritos anteriormente, se ha utilizado un ordenador con el sistema operativo Ubuntu 12.04, en el que se ha descargado e instalado el software necesario para configurar la tarjeta y para el posterior desarrollo de las aplicaciones.

2.3.1 Software

Para el presente proyecto ha sido necesario disponer de varias herramientas, de entre las que cabría destacar:

- Code Composer Studio (CCS). Es un entorno de desarrollo de aplicaciones para sistemas empujados, basado en Eclipse, creado por Texas Instruments.

Se trata de un software de enorme complejidad, pues da cobertura a numerosos sistemas embebidos, lo que hace que haya numerosas variables a la hora de realizar la configuración de los proyectos.

Este software se ha empleado para compilar, depurar y cargar las diferentes aplicaciones para los DSP, tanto las de prueba proporcionadas por TI como las desarrolladas expresamente para el proyecto, como más adelante detallaremos. En particular, se ha usado la versión 5.5.

- MCSDK. Se trata de un kit de desarrollo formado por numerosos paquetes. Está muy relacionado con CCS, pues en los mencionados paquetes se incluyen aplicaciones de prueba, listas para ser compiladas, que permiten comprobar el correcto funcionamiento de algunas de las características de la tarjeta, así como librerías que serán necesarias a la hora de crear las aplicaciones que serán utilizadas en la tarjeta. También incluye todos los archivos requeridos para poder arrancar el módulo de evaluación.

En particular, se empleó la versión 3.00.03.15 del kit BIOSLINUXMCSDK-K2, que da soporte a los SoC de la familia Keystone II. Se puede descargar de manera gratuita desde la web de TI.

Dispone de una guía de usuario [8], que contiene información sobre la instalación de paquetes adicionales que puedan ser necesarios, manuales de instalación e información detallada del funcionamiento de las diferentes librerías que incluye o guías para la ejecución de aplicaciones de prueba.

- GNU Debugger (GDB). Es un depurador, una herramienta que se emplea para ver qué ocurre cuando una aplicación se está ejecutando. Entre sus funciones, permite colocar puntos de ruptura, ejecutar la aplicación paso a paso, u observar en que línea del código se encuentra. Todo esto resulta de gran utilidad a la hora de encontrar errores, al poder observar en detalle que sucede al ejecutar una aplicación, como en el caso de este proyecto, las que se crearon para el ARM.
- GNU Radio. Es un software de desarrollo de código abierto, que nos proporciona las herramientas para implementar aplicaciones de radio software, es decir que realiza el procesamiento de las señales mediante código en lugar de emplear componentes físicos. [9]

Para implementar las aplicaciones, se dispone de un conjunto de bloques, escritos en C++, cada uno encargado de realizar una función. Para realizar la conexión de estos bloques entre sí, se crea lo que se conoce como grafo, escrito habitualmente en lenguaje Python, en el que por un lado se indicará los bloques que se van a emplear y por otro de qué manera están conectadas las entradas y salidas de estos bloques entre ellas.

A continuación, en la figura 5, podemos observar un sencillo ejemplo de aplicación, con dos fuentes o bloques iniciales con solo salidas y un sumidero o bloque final con solo entradas.

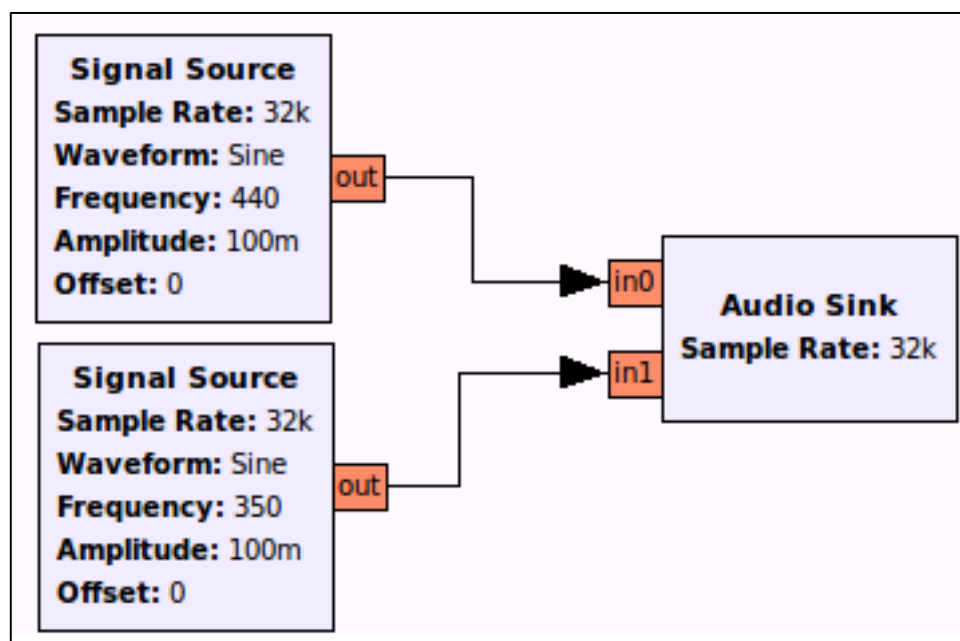


Figura 5. Ejemplo de aplicación en GNU Radio

Dentro de la librería de bloques proporcionados, existe una jerarquía. Los bloques de procesamiento derivan de una clase llamada `gr_basic_block`, o en su defecto de una clase derivada de ésta. Cada bloque reimplementa un método llamado *work* en el que se codifica el algoritmo de procesamiento de señal.

El otro elemento fundamental de GNU Radio, el planificador, es el encargado de ir llamando al *método work* de cada bloque según corresponda, controlando la ejecución del programa.

En el modo de funcionamiento más habitual se crea un hilo (*thread*) en el que se ejecuta cada uno de los bloques.

3

Conexión e instalación

En el presente capítulo se detallará como ha sido la puesta a punto del módulo de evaluación, para poder hacer uso de las herramientas y probar las aplicaciones creadas.

3.1 Conexión y arranque

La conexión, para poder establecer la comunicación entre el módulo de evaluación y el ordenador se realizó tal y como se observa en la figura 6.

Al contrario que en el proyecto anterior [3], no se requirió de la instalación de una máquina virtual, instalando todo el software necesario y realizando el desarrollo desde el SO Ubuntu, simplificando así el esquema de conexión.

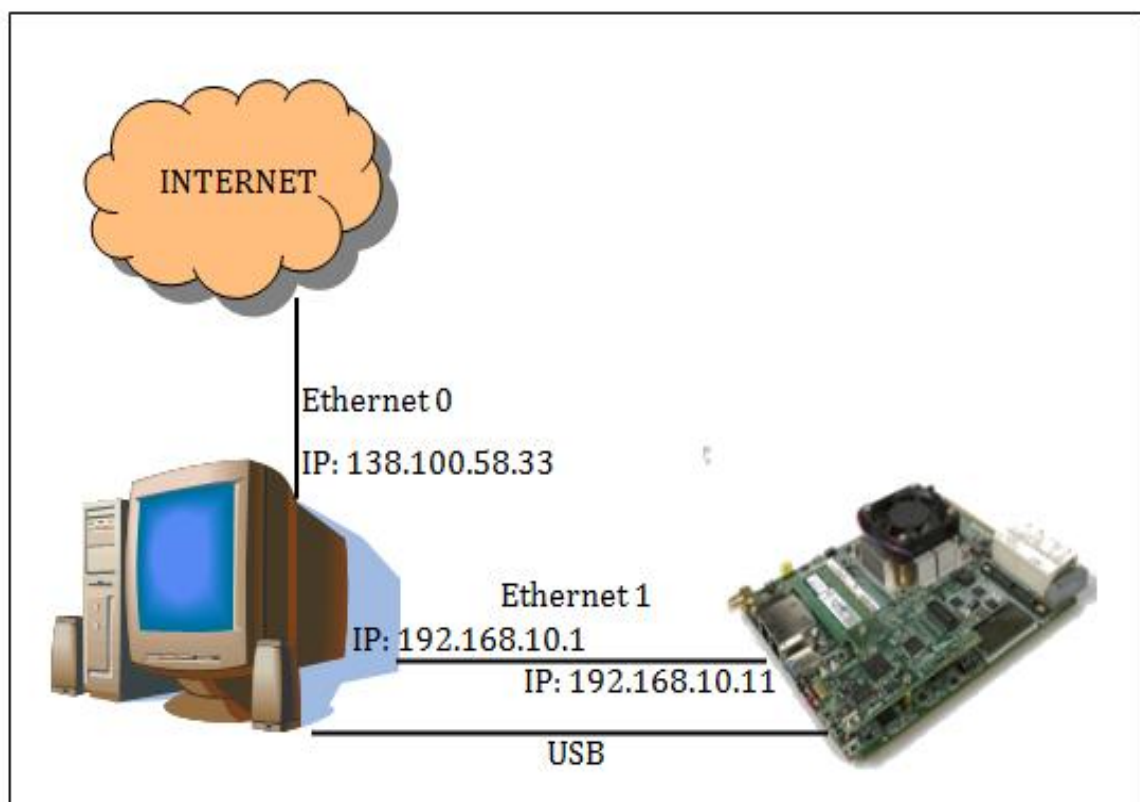


Figura 6. Esquema de la conexión

La elección de las direcciones en la Ethernet 1, tanto para el ordenador como para la tarjeta, han sido otorgadas expresamente para este proyecto, puesto que ningún otro dispositivo hace uso de esa interfaz.

Por un lado, se utilizó la conexión de la tarjeta al ordenador mediante un puerto serie (USB), para acceder a la consola. En el ordenador se utilizó el emulador de terminal 'gtkterm' para Ubuntu, configurado según las recomendaciones del fabricante. En la figura 7 se incluye la configuración de esta conexión.

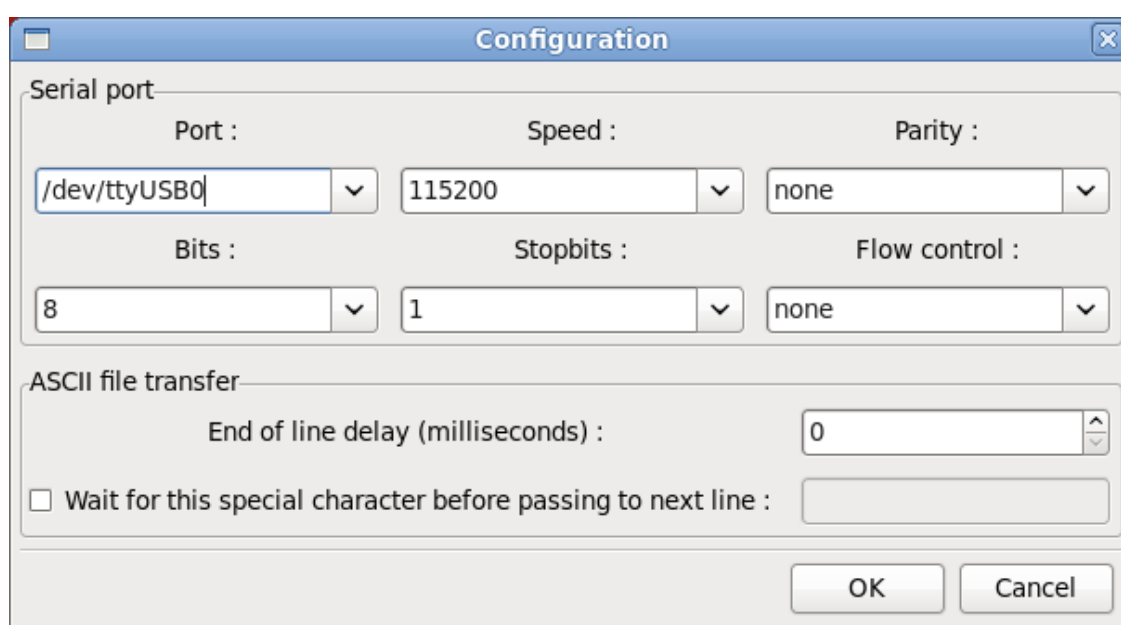


Figura 7. Configuración de gtkterm para la conexión por USB

Por otro lado, se ha utilizado la conexión a través de Ethernet para montar un sistema de ficheros a través de NFS, lo cual permite pasar archivos entre el ordenador y la tarjeta de manera automática, facilitando enormemente el trabajo.

Para poder montar el sistema de ficheros, en primer lugar hubo que instalar un servidor de DHCP. Esto era necesario para configurar en la tarjeta la dirección IP que se le ha asignado anteriormente, quedando así establecida la conexión entre el modulo de evaluación y el ordenador. Para el correcto funcionamiento del servidor DHCP, hubo que configurarlo para que únicamente escuchase en la interfaz Ethernet 1, dedicada a la tarjeta.

Para ello hubo que modificar los archivos:

- `/etc/dhcp/dhcpd.conf`

```
host evmk2h {  
    hardware ethernet 00:24:2B:65:54:84;  
    fixed-address 192.168.10.11;  
    next-server 192.168.10.1  
}
```
- `/etc/default/isc-dhcp-server`

```
INTERFACES="eth1"
```

Posteriormente se procedió a instalar en el ordenador un servidor TFTP, a través del cual se cargarán los archivos imagen del kernel del SO que se utilizará en la tarjeta. Para ello se siguieron los pasos indicados en el manual [10] que proporciona TI, para la correcta instalación y configuración del mismo.

Tras esto, se instaló el servidor NFS, que servirá para cargar el sistema de ficheros, empleando para su instalación y configuración las instrucciones [11] que se utilizaron para la tarjeta usada en el proyecto anterior.

Se crearon para los servidores anteriores los respectivos directorios para dar cabida a los archivos y/o directorios que han de contener cada uno. En el caso de TFTP en la ruta `/srv/tftp` donde se alojaron los archivos necesarios para el arranque de la tarjeta, detallados más adelante y en el de NFS `/srv/nfs/evmk2h_ti` se alojó aquí el sistema de ficheros. En particular, en el MCSDK se proporcionan dos sistemas de ficheros en formato comprimido:

- `tisdk-rootfs.tar.gz`
- `arago-console-image.tar.gz`

En nuestro caso, se ha descomprimido y copiado en la ruta de NFS el primero de ellos. Se ha elegido este sistema de ficheros porque es más completo que el segundo, ya que incorpora algunos paquetes que nos son imprescindibles.

Una vez instalados los servidores, se procedió al arranque de la tarjeta. En primer lugar, se seleccionó en los switches el modo 0010 (ver apartado 3.1.2) para que se cargue u-boot desde la memoria SPI. Esta configuración de los switches se observa en la figura 8.



Figura 8. Detalle de los switches, extraído del manual proporcionado por Advantech [4]

Una vez cargado u-boot, es necesario configurar los parámetros de arranque de este, para lo cual detenemos el arranque automático, para realizar los siguientes cambios en los parámetros:

Indicamos cual es la dirección IP del servidor:

```
#setenv serverip 192.168.1.10
```

Indicamos la ruta donde están los archivos a cargar por TFTP

```
#setenv tftp_root /srv/tftp
```

Indicamos la ruta donde está el sistema de ficheros

```
#setenv nfs_root /srv/nfs/evmk2h_ti
```

Configuramos para que el arranque de la tarjeta se haga a través de TFTP+ NFS

```
#setenv boot net
```

Por último, ejecutamos saveenv para guardar los cambios realizados en la configuración, y printenv para visualizar todos los parámetros de arranque y comprobar que están correctos, tal y como se puede observar en la figura 9 en la página siguiente.

```

GtkTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Control signals View Help
TCI6638 EVM # printenv
addr_fdt=0x87000000
addr_fs=0x82000000
addr_kern=0x88000000
addr_mon=0x0c5f0000
addr_ubi=0x82000000
addr_uboot=0x87000000
args_all=setenv bootargs console=ttyS0,115200n8 rootwait=1
args_net=setenv bootargs ${bootargs} rootfstype=nfs root=/dev/nfs rw nfsroot=${serv
erip}:${nfs_root},${nfs_options} ip=dhcp
args_ramfs=setenv bootargs ${bootargs} earlyprintk rdinit=/sbin/init rw root=/dev/r
am0 initrd=0x802000000,9M
args_ubi=setenv bootargs ${bootargs} rootfstype=ubifs root=ubi0:rootfs rootflags=sy
nc rw ubi.mtd=2,2048
args_uinitrd=setenv bootargs ${bootargs} earlyprintk rdinit=/sbin/init rw root=/dev
/ram0
baudrate=115200
boot=net
bootcmd=run init_${boot} get_fdt_${boot} get_mon_${boot} get_kern_${boot} run_mon r
un_kern
bootdelay=3
bootfile=uImage
burn_ubi=nand erase.part ubifs; nand write ${addr_ubi} ubifs ${filesize}
burn_uboot=sf probe; sf erase 0 0x100000; sf write ${addr_uboot} 0 ${filesize}
ethact=TCI6638_EMAC
ethaddr=c4:ed:ba:a8:ba:41
fdt_high=0xffffffff
get_fdt_net=dhcp ${addr_fdt} ${tftp_root}/${name_fdt}
get_fdt_ramfs=dhcp ${addr_fdt} ${tftp_root}/${name_fdt}
get_fdt_ubi=ubifsload ${addr_fdt} ${name_fdt}
get_fdt_uinitrd=dhcp ${addr_fdt} ${tftp_root}/${name_fdt}
get_fs_ramfs=dhcp ${addr_fs} ${tftp_root}/${name_fs}
get_fs_uinitrd=dhcp ${addr_fs} ${tftp_root}/${name_uinitrd}
get_kern_net=dhcp ${addr_kern} ${tftp_root}/${name_kern}
get_kern_ramfs=dhcp ${addr_kern} ${tftp_root}/${name_kern}
get_kern_ubi=ubifsload ${addr_kern} ${name_kern}
get_kern_uinitrd=dhcp ${addr_kern} ${tftp_root}/${name_kern}
get_mon_net=dhcp ${addr_mon} ${tftp_root}/${name_mon}
get_mon_ramfs=dhcp ${addr_mon} ${tftp_root}/${name_mon}
get_mon_ubi=ubifsload ${addr_mon} ${name_mon}
get_mon_uinitrd=dhcp ${addr_mon} ${tftp_root}/${name_mon}
get_ubi_net=dhcp ${addr_ubi} ${tftp_root}/${name_ubi}
get_uboot_net=dhcp ${addr_uboot} ${tftp_root}/${name_uboot}
has_mdio=0
init_net=run set_fs_none args_all args_net
init_ramfs=run set_fs_none args_all args_ramfs get_fs_ramfs
init_ubi=run set_fs_none args_all args_ubi; ubi part ubifs; ubi mount boot
init_uinitrd=run set_fs_uinitrd args_all args_uinitrd get_fs_uinitrd
initrd_high=0xffffffff
mem_lpa=1
mem_reserve=512M
mtdparts=mtdparts=davinci_nand.0:1024k(bootloader)ro,512k(params)ro,129536k(ubifs)
name_fdt=uImage-rt-k2hk-evm.dtb
name_fs=arago-console-image.cpio.gz
name_kern=uImage-rt-keystone-evm.bin
name_mon=skern-keystone-evm.bin
name_ubi=keystone-evm-ubifs.ubi
name_uboot=u-boot-spi-keystone-evm.gph
name_uinitrd=uinitrd.bin
nfs_options=v3,tcp,rsize=4096,wsz=4096
nfs_root=/srv/nfs/evmk2h_ti
no_post=1
run_kern=bootm ${addr_kern} ${addr_uinitrd} ${addr_fdt}
run_mon=mon_install ${addr_mon}
serverip=192.168.10.1
set_fs_none=setenv addr_uinitrd -
set_fs_uinitrd=setenv addr_uinitrd ${addr_fs}
stderr=serial
stdin=serial
stdout=serial
tftp_root=/srv/tftp
ver=U-Boot 2013.01 (Nov 24 2013 - 16:43:18)

Environment size: 2983/262140 bytes
TCI6638 EVM #

```

Figura 9. Parámetros de arranque de u-boot

Para evitar problemas de compatibilidad con los servidores anteriores, se actualizó la versión de u-boot que venía por defecto con la tarjeta por una más actual. Así pues, haciendo uso del servidor TFTP previamente instalado, se volcó en la tarjeta el archivo “u-boot-spi-keystone-evm.gph”, que se obtuvo del paquete MCSDK, para después desde u-boot realizar la actualización:

```
#dhcp 0xc300000 u-boot-spi-keystone-evm.gph
#sf probe
#sf erase 0 <size of u-boot-spi-keystone-evm.gph in hex
rounded to sector>
#boundary of 0x10000>
#sf write 0xc300000 0 <size of u-boot-spi-keystone-
evm.gph image in hex>
```

Al reiniciar la tarjeta, se comprueba que efectivamente se ha actualizado correctamente la versión de u-boot.

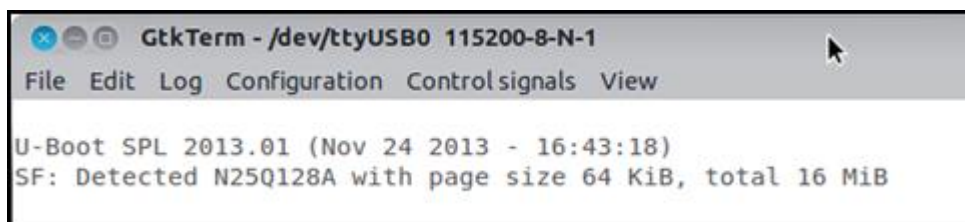
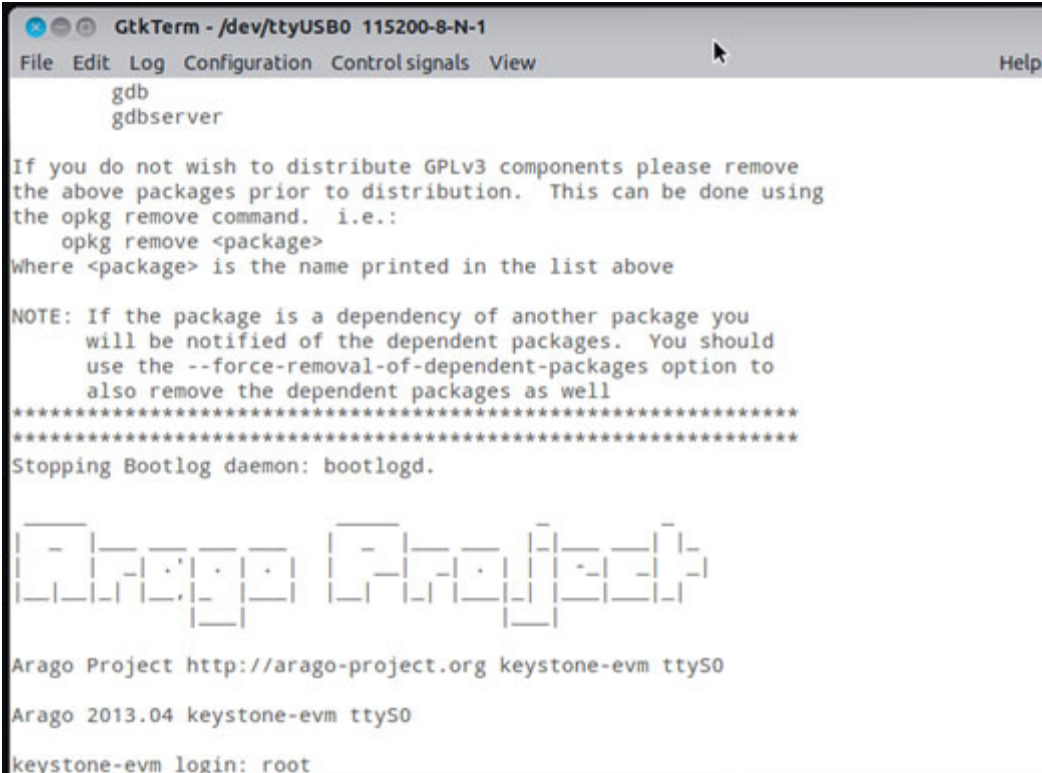


Figura 10. Captura de la nueva versión de u-boot (La anterior databa del 16/08/13)

Al arrancar por primera vez la tarjeta a través de TFTP+NFS, irá solicitando los archivos imagen necesarios, que se proporcionan en el paquete MCSDK y que se encontraban en la ruta /opt/ti/mcsdk_linux_3_00_03_15, y habrán de ser copiados al directorio de TFTP creado anteriormente, desde donde serán cargados:

- La imagen del monitor de arranque: *skern-keystone-evm.bin*
- La imagen del device tree, que contiene información como la configuración del mapa de memoria: *uImage-rt-k2hk-evm.dtb*
- La imagen precompilada del kernel: *uImage-rt-keystone-evm.bin*

Si no se producen errores, al finalizar el proceso de arranque, se pedirá un login y la tarjeta estará disponible, como se observa en la figura 11.



```
GtkTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Controlsignals View Help

gdb
gdbserver

If you do not wish to distribute GPLv3 components please remove
the above packages prior to distribution. This can be done using
the opkg remove command. i.e.:
  opkg remove <package>
Where <package> is the name printed in the list above

NOTE: If the package is a dependency of another package you
      will be notified of the dependent packages. You should
      use the --force-removal-of-dependent-packages option to
      also remove the dependent packages as well
*****
Stopping Bootlog daemon: bootlogd.

[ASCII art logo]

Arago Project http://arago-project.org keystone-evm ttyS0
Arago 2013.04 keystone-evm ttyS0
keystone-evm login: root
```

Figura 11. Login una vez finalizado el proceso de arranque

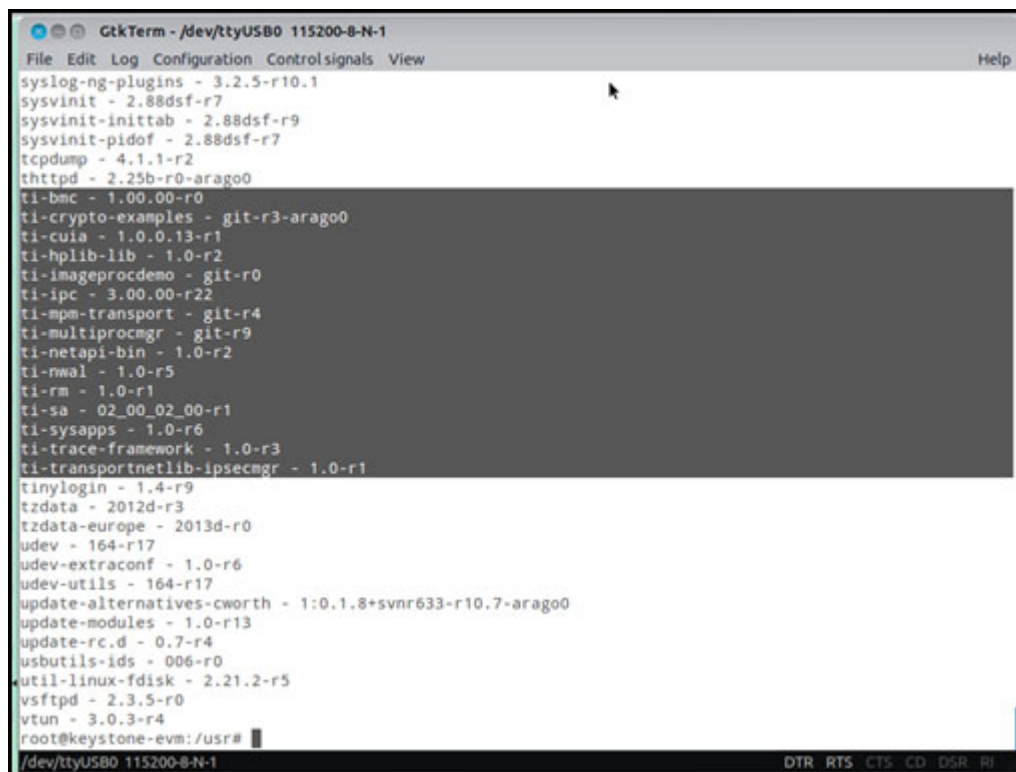
Una vez configurada y arrancada la tarjeta, el siguiente paso fue testear las diferentes librerías empleadas para comunicar ARM y DSP.

3.2 Testeo del módulo de evaluación

Una vez realizada la puesta a punto del entorno, se procedió a comprobar el funcionamiento de la comunicación entre ARM y DSP.

En primer lugar, comprobamos que efectivamente, tal y como se comentó en el apartado anterior, el sistema de ficheros elegido dispone de los paquetes necesarios.

Para ello se empleó la herramienta Open Package Management (opkg), que permite listar los paquetes ya disponibles así como instalar paquetes adicionales previamente descargados.



```
GtkTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Controlsignals View Help
syslog-ng-plugins - 3.2.5-r10.1
sysvinit - 2.88dsf-r7
sysvinit-inittab - 2.88dsf-r9
sysvinit-pidof - 2.88dsf-r7
tcpdump - 4.1.1-r2
thttpd - 2.25b-r0-arago0
ti-bmc - 1.00.00-r0
ti-crypto-examples - git-r3-arago0
ti-cuia - 1.0.0.13-r1
ti-hplib-lib - 1.0-r2
ti-imageprocdemo - git-r0
ti-ipc - 3.00.00-r22
ti-mpm-transport - git-r4
ti-multiprocgr - git-r9
ti-netapi-bin - 1.0-r2
ti-nwal - 1.0-r5
ti-rm - 1.0-r1
ti-sa - 02_00_02_00-r1
ti-sysapps - 1.0-r6
ti-trace-framework - 1.0-r3
ti-transportnetlib-ipsecmgr - 1.0-r1
tinylogin - 1.4-r9
tzdata - 2012d-r3
tzdata-europe - 2013d-r0
udev - 164-r17
udev-extraconf - 1.0-r6
udev-utils - 164-r17
update-alternatives-cworth - 1:0.1.8+svn633-r10.7-arago0
update-modules - 1.0-r13
update-rc.d - 0.7-r4
usbutils-ids - 006-r0
util-linux-fdisk - 2.21.2-r5
vsftpd - 2.3.5-r0
vtun - 3.0.3-r4
root@keystone-evm: /usr#
```

Figura 12. Fragmento de la lista de paquetes instalados

Tal y como se observa en la figura 12, los paquetes específicos de la tarjeta comienzan por “TI”, entre ellos aquellos necesarios para la comunicación.

Llegados a este punto, se puede proceder con el testeo de la comunicación entre núcleos, que hace uso de dos módulos principalmente:

3.2.1 Multi processor manager (MPM).

Se trata de un módulo incluido en el MCSDK que sirve para cargar archivos imagen del DSP desde el ARM. Consta de dos partes principales, un servidor que se incluye en el sistema de ficheros empleado para la tarjeta y una utilidad, llamada mpmcl, para acceder al servidor. Se puede hacer uso de esta utilidad desde la propia línea de comandos del SO Linux de la tarjeta.

En el MCSDK se dispone de un proyecto de CCS, así como de un archivo ejecutable listo para usarse con extensión .out, para testear el funcionamiento de este módulo. Así mismo TI proporciona información en la guía de usuario de MCSDK [8] en la que se explica cómo hacer uso de mpmcl.

Al ejecutar el archivo, como respuesta por parte del DSP, se crea un fichero de texto (trace), que queda guardado en el ARM en la ruta */debug/remoteproc/remoteprocX* (siendo X el número del DSP), con el resultado de la operación. En este caso, el ejecutable proporcionado devolvía como respuesta en trace una línea de texto con el número del DSP utilizado.

3.2.2 MSGCOM

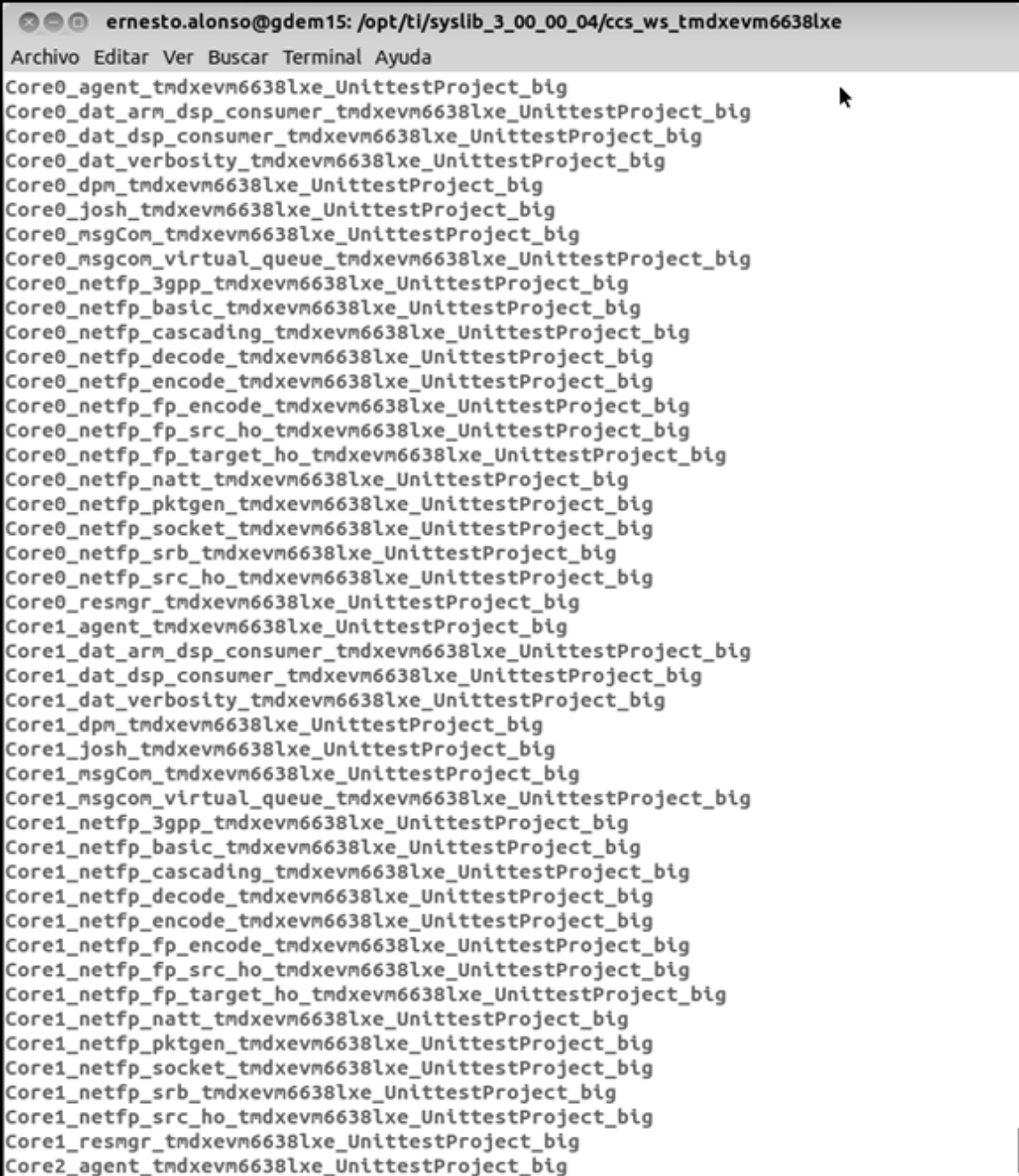
Es una librería creada para simplificar el intercambio de mensajes entre núcleos a través del Multicore Navigator. Ésta a su vez depende otros componentes para poder llevar a cabo la comunicación:

- Resource Manager (RM). Garantiza que los recursos del sistema pueden ser accedidos sin conflicto, para lo cual dispone de una base de datos, que se actualiza cuando el ARM o los DSP hacen uso de alguno de estos recursos.
- Message Router. Es un proceso secundario (daemon) que crea canales de control.
- Job Scheduler (JOSH). Es un mecanismo que permite que una función llamada en un núcleo, se ejecute en otro distinto.
- AGENT. Componente empleado para realizar la sincronización de los recursos empleados para la gestión de los canales. Envía los mensajes a través de los canales de control creados por el Message Router, para crear, gestionar o borrar canales.
- PKT. Es una librería que permite crear canales para enviar y recibir buffers, y asignar paquetes a esos buffers, desde el lado de DSP. Estos buffers estarán en zonas de memoria compartida.
- UDMA. Es la equivalente a PKT, pero en el lado de ARM.

Para comprobar su funcionamiento, se procedió siguiendo las indicaciones de la guía que se proporciona para ello en el MCSDK, que se encuentra en la ruta */opt/ti/syslib_3_00_00_04/docs*.

En primer lugar, se crearon los proyectos para CCS que permitirán generar los ejecutables que serán cargados en los DSP. El inconveniente encontrado en este paso es que los archivos de instalación proporcionados son para ordenadores con SO Windows (extensión .bat), hubo por tanto que realizar una adaptación de los mismos para que pudieran ser utilizados en Ubuntu (extensión .sh).

En la figura 13, se muestra una captura del directorio con todos los directorios que contienen estos proyectos.



```
ernesto.alonso@gdem15: /opt/ti/syslib_3_00_00_04/ccs_ws_tmdxevm6638lxe
Archivo  Editor  Ver  Buscar  Terminal  Ayuda
Core0_agent_tmdxevm6638lxe_UnittestProject_big
Core0_dat_arm_dsp_consumer_tmdxevm6638lxe_UnittestProject_big
Core0_dat_dsp_consumer_tmdxevm6638lxe_UnittestProject_big
Core0_dat_verbosity_tmdxevm6638lxe_UnittestProject_big
Core0_dpm_tmdxevm6638lxe_UnittestProject_big
Core0_josh_tmdxevm6638lxe_UnittestProject_big
Core0_msgCom_tmdxevm6638lxe_UnittestProject_big
Core0_msgcom_virtual_queue_tmdxevm6638lxe_UnittestProject_big
Core0_netfp_3gpp_tmdxevm6638lxe_UnittestProject_big
Core0_netfp_basic_tmdxevm6638lxe_UnittestProject_big
Core0_netfp_cascading_tmdxevm6638lxe_UnittestProject_big
Core0_netfp_decode_tmdxevm6638lxe_UnittestProject_big
Core0_netfp_encode_tmdxevm6638lxe_UnittestProject_big
Core0_netfp_fp_encode_tmdxevm6638lxe_UnittestProject_big
Core0_netfp_fp_src_ho_tmdxevm6638lxe_UnittestProject_big
Core0_netfp_fp_target_ho_tmdxevm6638lxe_UnittestProject_big
Core0_netfp_natt_tmdxevm6638lxe_UnittestProject_big
Core0_netfp_pktgen_tmdxevm6638lxe_UnittestProject_big
Core0_netfp_socket_tmdxevm6638lxe_UnittestProject_big
Core0_netfp_srb_tmdxevm6638lxe_UnittestProject_big
Core0_netfp_src_ho_tmdxevm6638lxe_UnittestProject_big
Core0_resmgr_tmdxevm6638lxe_UnittestProject_big
Core1_agent_tmdxevm6638lxe_UnittestProject_big
Core1_dat_arm_dsp_consumer_tmdxevm6638lxe_UnittestProject_big
Core1_dat_dsp_consumer_tmdxevm6638lxe_UnittestProject_big
Core1_dat_verbosity_tmdxevm6638lxe_UnittestProject_big
Core1_dpm_tmdxevm6638lxe_UnittestProject_big
Core1_josh_tmdxevm6638lxe_UnittestProject_big
Core1_msgCom_tmdxevm6638lxe_UnittestProject_big
Core1_msgcom_virtual_queue_tmdxevm6638lxe_UnittestProject_big
Core1_netfp_3gpp_tmdxevm6638lxe_UnittestProject_big
Core1_netfp_basic_tmdxevm6638lxe_UnittestProject_big
Core1_netfp_cascading_tmdxevm6638lxe_UnittestProject_big
Core1_netfp_decode_tmdxevm6638lxe_UnittestProject_big
Core1_netfp_encode_tmdxevm6638lxe_UnittestProject_big
Core1_netfp_fp_encode_tmdxevm6638lxe_UnittestProject_big
Core1_netfp_fp_src_ho_tmdxevm6638lxe_UnittestProject_big
Core1_netfp_fp_target_ho_tmdxevm6638lxe_UnittestProject_big
Core1_netfp_natt_tmdxevm6638lxe_UnittestProject_big
Core1_netfp_pktgen_tmdxevm6638lxe_UnittestProject_big
Core1_netfp_socket_tmdxevm6638lxe_UnittestProject_big
Core1_netfp_srb_tmdxevm6638lxe_UnittestProject_big
Core1_netfp_src_ho_tmdxevm6638lxe_UnittestProject_big
Core1_resmgr_tmdxevm6638lxe_UnittestProject_big
Core2_agent_tmdxevm6638lxe_UnittestProject_big
```

Figura 13. Proyectos para los DSP, una vez instalados

En concreto, los proyectos empleados son:

- Core0_msgCom_tdmxevm6638lxe_UnittestProject_big
- Core1_msgCom_tdmxevm6638lxe_UnittestProject_big

Ya en CCS, hubo que indicar la correcta ubicación de las librerías necesarias para poder compilar los proyectos. Para ello, en la pestaña de propiedades de cada uno de los proyectos, se realizaron las modificaciones que se aprecian en las figuras 14 y 15.

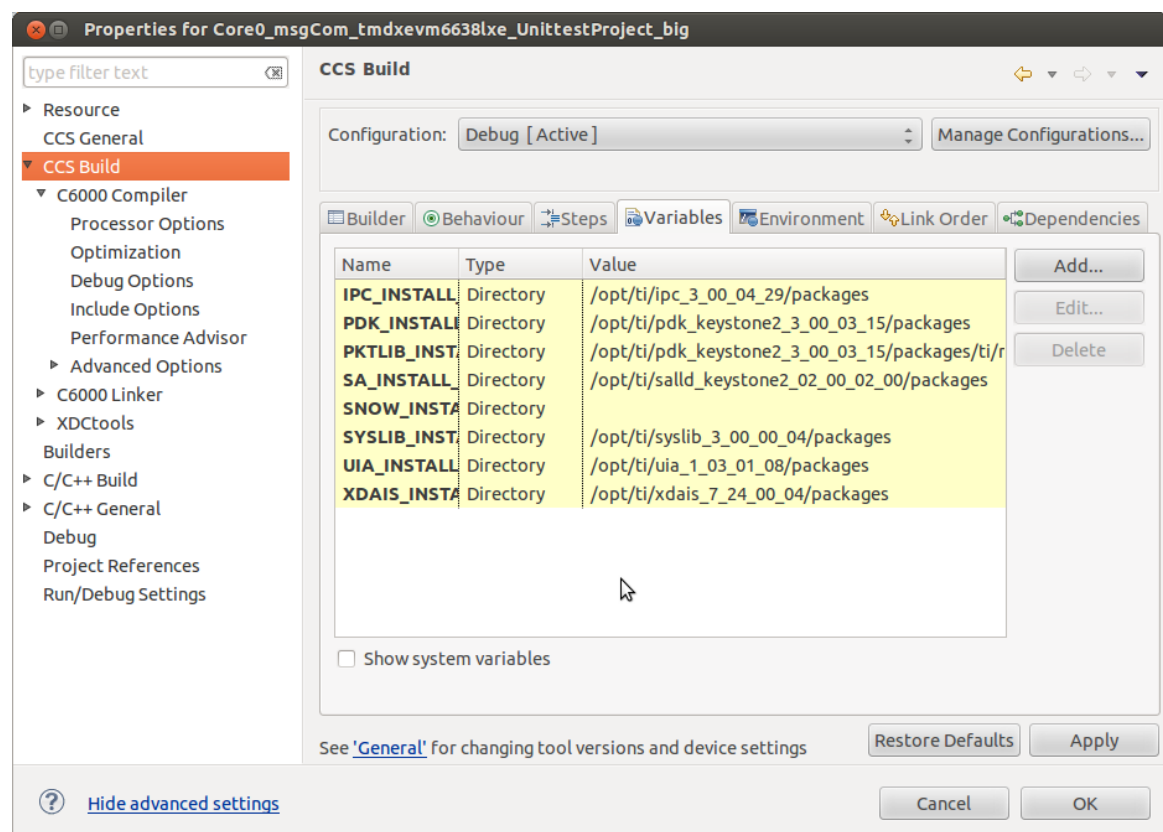


Figura 14. Inclusión de paquetes requeridos

Todas las librerías incluidas, vienen en el MCSDK, y se encuentran por tanto en la misma ruta en que se instaló éste.

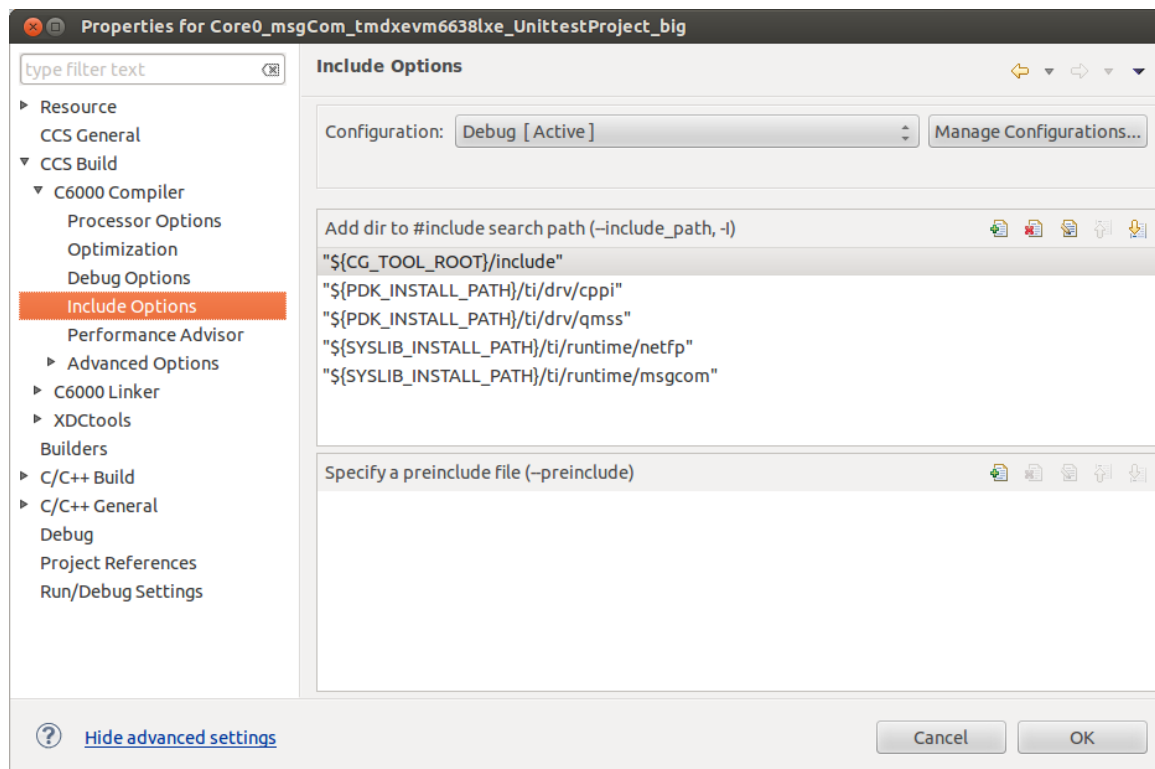


Figura 15. Inclusión de las rutas donde se encuentran las librerías

Así mismo, hubo que crear una configuración específica para la tarjeta, con las características de ésta, para que así los ejecutables creados tras compilar sean cargados en la zona de memoria correspondiente a los DSP. Para ello, en el menú Tools-->RTSC Tools-->Platform, se creó una nueva plataforma, a la que se nombró tmdxevm6638lxe, con las características que se aprecian en la figura 16.

Especial mención a la memoria externa, pues es compartida y habrá que indicar que tamaño y a partir de qué dirección corresponde a los DSP.

En el caso de la tarjeta EVMK2H, se ha situado a partir de la dirección 0xA0000000.

Edit Platform

Page 2 of 2 - Device Page

Enter Details for device

Device Details

Device Name: TMS320TCI6638

Device Family: c6000

Clock Speed (MHz): 1220 Import...

Device Memory

Name	Base	Length	Space	Access
MSMCSRAM	0x0c000000	0x00600000	code/data	RWX
L1DSRAM	0x00f00000	0x00000000	data	RW
L1DSRAM	0x00f00000	0x00000000	code	DIWV

L2 Cache: 0k L1D Cache: 32k L1P Cache: 32k

☐ Customize Memory

External Memory

Name	Base	Length	Space	Access
DDR3	0xA0000000	0x80000000	code/data	RWX

Memory Sections

Code Memory: L2SRAM Data Memory: L2SRAM Stack Memory: L2SRAM

? < Back Next > Cancel Finish

Figura 16. Creación del target

Esta nueva configuración, tiene que incluirse en cada proyecto. Para ello, en la pestaña de propiedades de cada uno, se indicó que la plataforma a utilizar será la creada, tmdxevm6638lxe, tal y como se observa en la figura 17.

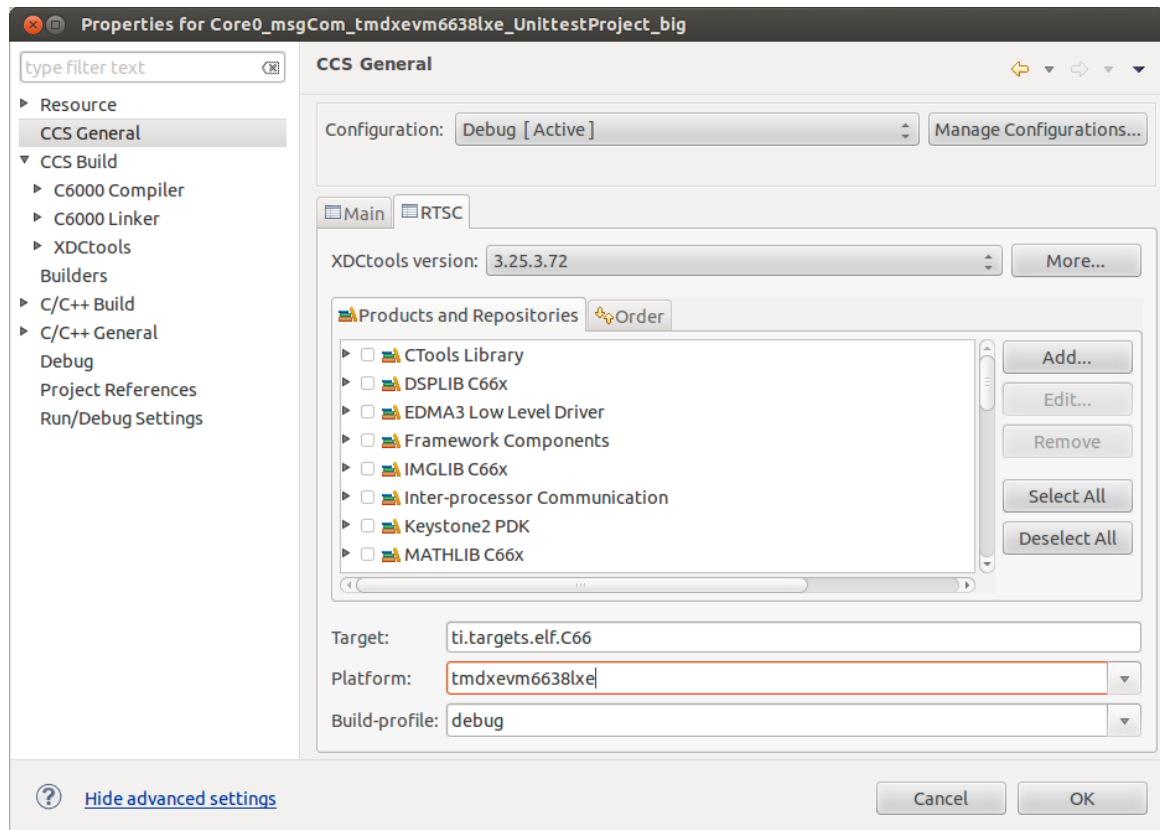


Figura 17. Inclusión de la plataforma creada en el proyecto

Por último, se comprobó en el archivo de configuración, con extensión .cfg, que los núcleos a utilizar están habilitados, comprobando para ello:

- Aquí se listan los núcleos a utilizar, siendo X el número de identificación del DSP (0 - 7)
`MultiProc.setConfig(null, ["COREX"]);`
- Aquí se informa del número de DSP a utilizar, siendo X el número total (1 - 8)
`MultiProc.numProcessors = X;`

Una vez finalizadas las comprobaciones, se procedió a compilar para obtener los ejecutables para los DSP.

Posteriormente, para obtener los ejecutables para el lado de ARM, hubo que descargar el paquete 'sysapps', desde el repositorio de arago-project [12]. En concreto, se descargó la versión 03.00.00.06 que es aquella adaptada a los ARM que emplea el módulo de evaluación EVMK2H. Para poder compilar los archivos de este paquete se instalaron también la cadena de herramientas (toolchain) linaro y el kit de desarrollo (devkit), incluido este último en el MCSDK.

Dentro del paquete sysapps, el ejecutable para el ARM, al finalizar la compilación, quedará alojado en la ruta `../sysapps/msgcom/test`.

Una vez compilados ambos proyectos y obtenidos los correspondientes ejecutables para ARM y DSP, se procedió con la ejecución siguiendo los pasos descritos en la guía.

En primer lugar, en la consola de la tarjeta, se ejecuta la aplicación msgrouter, necesaria para gestionar los canales de comunicación, escribiendo el comando:

```
msgrouter.out -n 4 -d 10 &
```

Después, se lanzará por un lado, desde CCS la ejecución de los ejecutables para DSP, haciendo uso del emulador, conectado a través de USB al ordenador.

Con ese fin, en CCS, hubo que crear una sesión de debug. Para configurarla y que CCS únicamente se conecte con los DSP a utilizar y cargue en ellos directamente los ejecutables, correspondientes a cada uno, en el menú Run-->Debug Configurations, se creó una sesión para cada proyecto, como se observa en la figura 18.

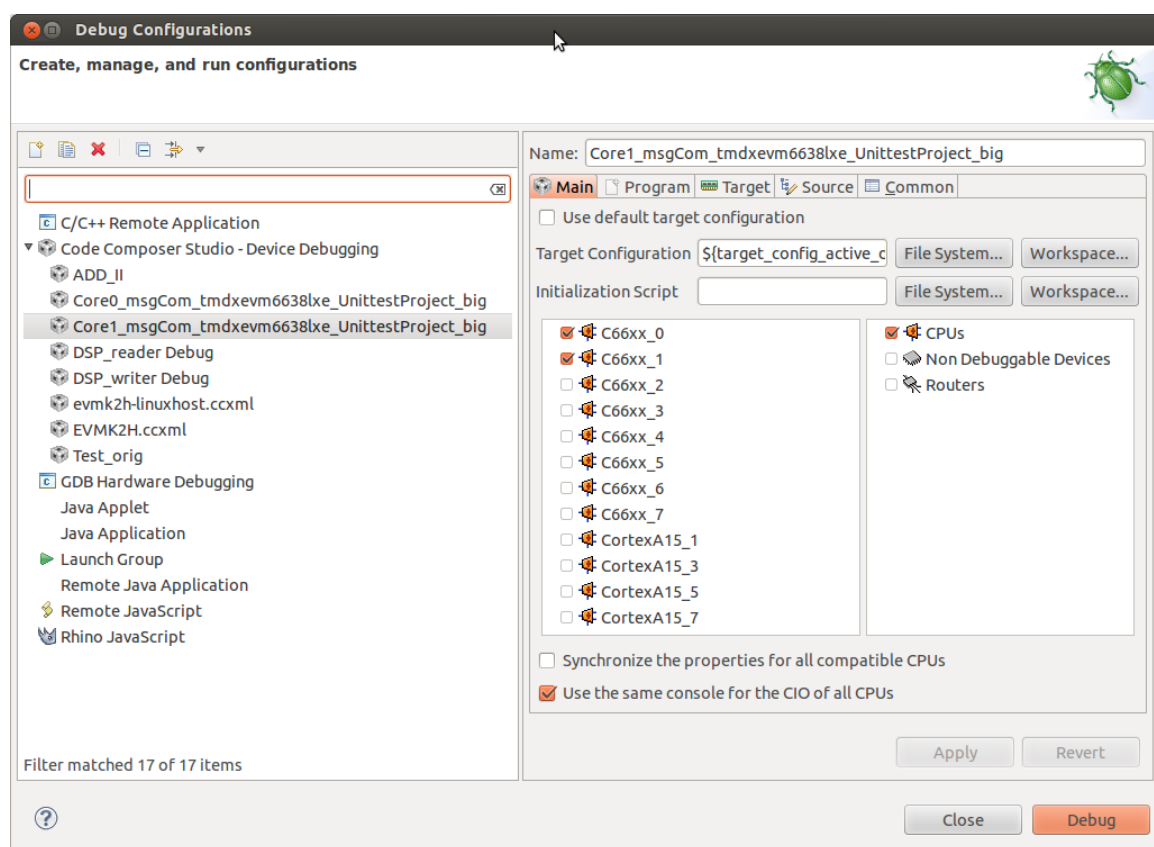


Figura 18. Creación de la sesión de debug

En la pestaña *main* se indica en que núcleo DSP se ha de cargar, y en la pestaña *program*, cual es el ejecutable a cargar.

Por el otro lado, desde la línea de comandos de la tarjeta, se cargan los ejecutables para el lado de ARM, en el orden establecido en la guía. Al finalizar la ejecución aparecen mensajes tanto en la consola de CCS como en el terminal indicando que la comprobación de la librería se ha llevado a cabo.

3.3 Instalación de GNU Radio

Una vez realizada la inicialización de la tarjeta y comprobado su correcto funcionamiento, se procedió a instalar GNU Radio en esta.

En primer lugar, se descargó el código fuente de GNU Radio en formato comprimido (.tar.gz) desde la página oficial. Se ofrecen de manera preferente otros métodos de instalación directa, pero en el caso de este proyecto no son de interés, puesto que pretendemos portarlo desde el ordenador a la tarjeta, no instalar la herramienta directamente en el ordenador. En particular se descargó la última versión disponible, *gnuradio-3.7.3*.

El fichero descargado se descomprimió y el directorio obtenido se alojó, a través de NFS, en la ruta */usr/src/* perteneciente al sistema de ficheros de la tarjeta.

Dentro del directorio obtenido, se incluye un archivo README con las instrucciones necesarias para proceder a la instalación de GNU Radio. Tal y como se indica en este archivo, se requieren paquetes externos adicionales para poder realizar la instalación.

3.3.1 Instalación de paquetes externos

El SO Arago instalado en la tarjeta, incluye algunos paquetes y librerías necesarios para poder hacer uso de herramientas software en la tarjeta, pero puesto que ésta no dispone de recursos ilimitados, es imposible que se incluyan todos y cada uno de los innumerables paquetes existentes.

En primer lugar se comprobó cuales eran los paquetes ya incluidos en el SO Arago, y por tanto instalados en la tarjeta.

Comparando la lista obtenida con `opkg` con la lista de requisitos de GNU Radio, se comprobó qué paquetes es necesario instalar en la tarjeta.

Estos paquetes dependen de la arquitectura en la que vayan a ser instalados, por lo que en nuestro caso deberemos obtener aquellos compatibles con núcleos ARM-Cortex A-15 (Ver punto 2.1).

Anteriormente los SO, disponían de un repositorio con todos los paquetes disponibles, ocurriendo que en muchas ocasiones se encontraba el mismo paquete para varias arquitecturas.

En Arago, debido a que el número de arquitecturas disponible cada vez es mayor, y en previsión de que mantener un repositorio con todas las posibles variantes sería imposible, han creado un entorno de construcción de paquetes basado en una estructura de capas, que permite crear paquetes personalizados, eligiendo por un lado el paquete deseado y por otro la arquitectura.

Por tanto, se procedió a instalar este entorno de construcción en el ordenador, siguiendo las instrucciones que proporciona Arago para ello [7].

```
#sudo git clone git://arago-project.org/git/projects/oe-  
layerssetup.git tisdk  
  
#cd tisdk  
  
#sudo ./oe-layertool-setup.sh -f configs/arago-dora-  
config.txt  
  
#cd build  
  
#. conf/setenv  
  
#export PATH="/opt/linaro/gcc-linaro-arm-linux-gnueabi-  
4.7-2013.03-20130313_linux/bin:$PATH"  
  
#MACHINE=keystone-evm bitbake core-image-minimal
```

Durante la instalación, se ha elegido como arquitectura KEYSTONE-EVM, y se comprobó en los mensajes que devuelve el instalador, que la configuración es correcta, siendo en nuestro caso:

Build Configuration:

```
BB_VERSION      = "1.20.0"
BUILD_SYS       = "x86_64-linux"
NATIVE_LSBSTRING = "Ubuntu-12.04"
TARGET_SYS      = "arm-oe-linux-gnueabi"
MACHINE         = "keystone-evm"
DISTRO          = "arago"
DISTRO_VERSION  = "2014.05"
TUNE_FEATURES   = "armv7a vfp thumb neon callconvention-
hard cortexal5"
TARGET_FPU      = "vfp-neon"
```

Una vez finalizada la instalación, se pueden construir los paquetes necesarios, que atendiendo al fichero README de GNU Radio, son:

- Boost versión 1.53 o superior
- Cppunit versión 1.9.14 o superior
- FFTW3F versión 3.0 o superior
- Python versión 2.5 o superior
- SWIG versión 1.3.31 o superior
- Numpy versión 1.1.0 o superior
- GSL versión 1.10 o superior

Para ello, se utiliza el siguiente comando:

```
#MACHINE=keystone-evm bitbake <nombre del paquete>
```

Quedando los paquetes instalados, que tendrán extensión .ipk, dentro del directorio del entorno de construcción, en la ruta /deploy/ipk. Si observamos el contenido de esta ruta tras finalizar la construcción de los paquetes, observamos que estos a su vez aparecen fragmentados en diversos subpaquetes. El objetivo de esto es que se instalen en la tarjeta únicamente aquellos fragmentos estrictamente necesarios, puesto que el espacio disponible en la tarjeta es limitado y no conviene malgastarlo con subpaquetes innecesarios.

Para instalar los paquetes, se emplea `opkg`. Esta herramienta, dispone de un archivo de configuración llamado `opkg.conf`, para que pueda reconocer que paquetes son compatibles con la tarjeta. Para que pueda reconocer los paquetes construidos hubo que modificar el archivo, que se encuentra en la ubicación `/etc/opkg` y añadirle la línea:

```
arch cortexal5hf-vfp-neon-3.10 91
```

Además de los paquetes mencionados, la compilación de GNU Radio, al contrario que en versiones anteriores, en las últimas versiones se realiza con el sistema de compilación CMake, por tanto será necesario instalar en la tarjeta los paquetes para este sistema, que se pueden obtener también del entorno de construcción.

Así mismo hubo que instalar en la tarjeta los paquetes GCC y G++, que son compiladores de ficheros con código en C y C++

Los paquetes pueden tener a su vez dependencia de otros paquetes secundarios, de modo que la metodología seguida a la hora de instalar GNU Radio es la de proceder a instalar sin haber incluido ningún paquete en la tarjeta. Así, prestando atención a los errores que devuelve el instalador, se van incluyendo únicamente aquellos subpaquetes que requiera, asegurando de esta manera que los paquetes incluidos en la tarjeta son los estrictamente necesarios.

De entre los numerosos paquetes secundarios instalados, cabe destacar el paquete “`libstdc++-dev`”, que incluye numerosas librerías requeridas por el compilador GCC. Por defecto este paquete se instala en la ruta `/include` al contrario que el resto de librerías que se hayan ubicadas en la ruta `/usr/include`.

Por tanto para que el compilador GCC pueda localizar las librerías de este paquete hubo que crear dos links:

- `/usr/include/c++` → `/include/c++/4.7.3`
- `/include/c++/4.7.3/arm-oe-linux-gnueabi` → `arm-linux-gnueabi`
`gnueabi`

Una vez que CMake finaliza la compilación con éxito de GNU Radio, se procede a la instalación, eligiendo como ubicación del programa la ruta `/opt/gnuradio/3.7.3`.

Para su correcto funcionamiento, hay que crear algunas variables de entorno que contienen los paths necesarios para que GNU Radio pueda localizar la ubicación de todos sus archivos:

- `PATH=/opt/gnuradio/3.7.3/bin`
- `LD_LIBRARY_PATH=/opt/gnuradio/3.7.3/lib`
- `PKG_CONFIG_PATH=/opt/gnuradio/3.7.3/lib/pkgconfig`
- `PYTHONPATH=/opt/gnuradio/3.7.3/lib/python2.7/site-packages`

4

Trabajo realizado

Como se ha mencionado anteriormente, el presente proyecto es una continuación a otros trabajos realizados con anterioridad. Por eso, el trabajo realizado consiste en adaptar un código ya existente, creado por Carlos Sánchez Martín [3] en lugar de codificar una aplicación desde cero.

Como se ha descrito en el capítulo anterior, el método de comunicación entre los núcleos ARM y los DSP que se emplea en el módulo EVMK2H, ha cambiado completamente respecto al modo en que se hacía en la tarjeta IGEPv2 empleada en el proyecto anterior. Por ello, las funciones que se emplean en el código difieren totalmente, siendo por tanto la principal tarea para realizar la adaptación encontrar las equivalencias y similitudes entre ambas para poder realizar una sustitución.

4.1 Descripción del código existente

La idea principal que surgió en el anterior proyecto [3] para extender el uso de GNU Radio, es que la tarea de procesado se ejecuta en un núcleo DSP, el código en el ARM únicamente se encarga de enviar la tarea y recoger los resultados.

4.1.1 Código para ARM

El diseño del programa que correrá en el lado de ARM se realizó empleando GNU Radio, es decir tendrá una estructura compuesta por varios módulos. De entre ellos, cabe destacar dos: un módulo, que contendrá los bloques de procesado con las diferentes tareas que se puedan realizar en los DSP, al que se llamó 'bloque de procesado' y otro módulo encargado de gestionar el envío de tareas entre el ARM y los DSP, llamado 'gestor de trabajos'.

Este último módulo contiene una tabla que dispone de información acerca de las tareas que puede realizar cada DSP, para así poder saber a cuál de ellos asignar cada tarea. Esta tabla se inicializa con la información que envía cada DSP en su primer mensaje tras ser inicializado.

Atendiendo a esto, el grafo de GNU Radio tendría una estructura como la que se puede observar en la figura 19.

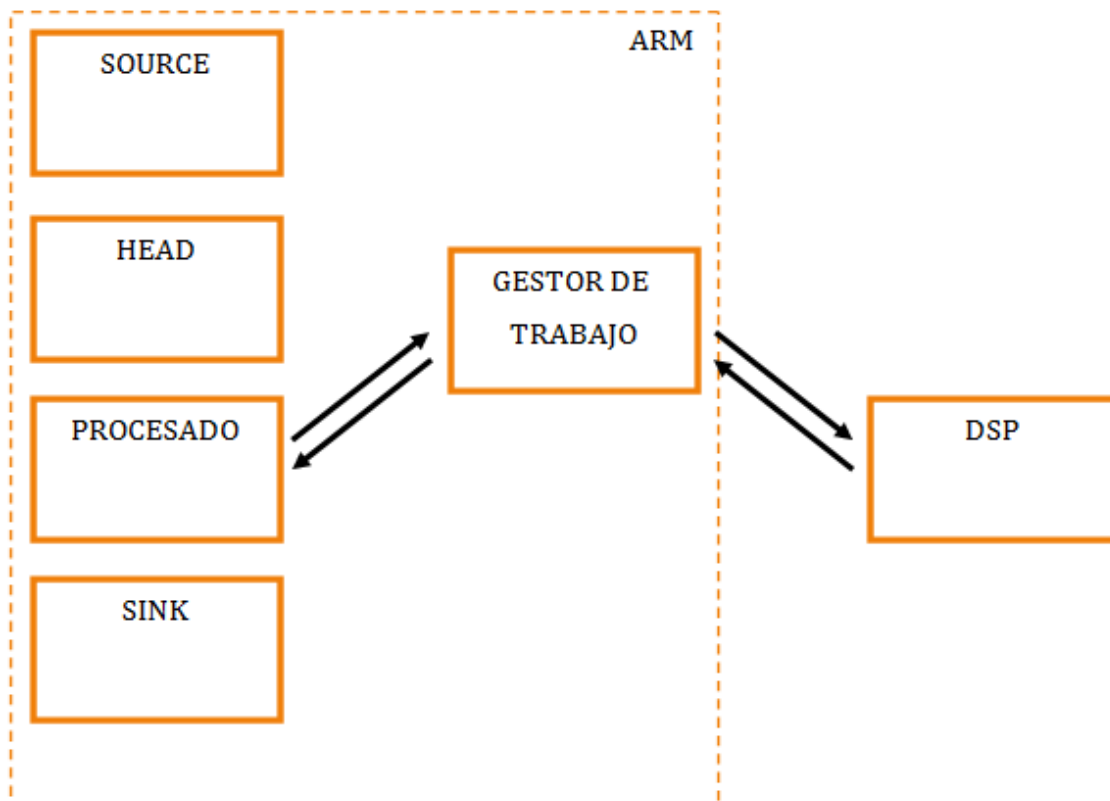


Figura 19. Grafo de GNU Radio

Se crearon por tanto en GNU Radio dos nuevos módulos

- Módulo GR_GDSP para bloques de procesamiento de señal:

Este módulo contiene los distintos bloques de procesamiento de señal, para cada una de las tareas que se pretendan realizar en los DSP. En particular, se incluyeron dos bloques, uno, llamado `gdsp_add_ii` que calcula la suma de vectores enteros y otro llamado `gdsp_moving_avg`, que interpreta la media móvil.

- Módulo GDSP para el gestor de trabajos:

En este módulo se reúnen todas aquellas tareas relacionadas con la comunicación entre ARM y DSP. En primer lugar se realiza la inicialización de los DSP, y después se gestiona el envío y recepción de mensajes.

Sus principales archivos son `gd_job_manager.cc` y `gd_job_manager_impl.cc` que contienen el código para implementar el gestor de trabajos.

En el código, aparte de las funciones para gestionar la comunicación, cabe mencionar también que se creó una estructura que será el mensaje que se envíen los núcleos entre sí:

```
typedef struct My_msg_tag{
    MSGQ_MsgHeader    msgHeader;
    gd_proc_def_t      tabla_id_fun;
    int                n_func;
    gd_job_desc_t      job;
}My_msg;
```

Contenía una cabecera, la información necesaria para que el ARM complete la tabla al inicializar, y el descriptor del trabajo para cuando el ARM asigne una tarea a un DSP.

En el archivo `gd_job_manager_impl.cc`, que es donde se encuentra el código que realiza la gestión de la comunicación, se hallan los siguientes métodos:

- `Alloc_job_desc`. Función que asigna un contenedor para la tarea. Está protegida por un semáforo, para mantener el control de cuantas tareas activas hay en cada momento.
- `Free_job_desc`. Una vez concluida la tarea, esta función libera el contenedor que tenía asignado.
- `Submit_job`. Comprueba el estado en que se encuentran los DSP. Si hay alguno disponible le envía la tarea, si están todos ocupados, almacena la tarea en una cola de tareas pendientes.
- `Send_job`. Función encargada de enviar el mensaje al DSP que corresponda, informando de la tarea a realizar.

- `Wait_jobs`. Espera a recibir confirmación de que las tareas se han completado para informar al bloque de procesado.
- `Job_completer_loop`. Bucle en el que se lee el mensaje enviado por el DSP como respuesta, y se marca la tarea como completada.
- `Lookup_proc`. Función para buscar una tarea en la tabla creada al inicializar.
- `INICIALIZAR_DSP`. En esta función se crea la cola para el ARM, se cargan los ejecutables en los DSP, se arrancan estos últimos, y se busca la cola que han creado para engancharse a ella, estableciendo así la comunicación entre núcleos.

4.1.2 Código para DSP

Por otro lado, para implementar el ejecutable del que harán uso los DSP se empleó CCS. En primer lugar se establece la comunicación.

- **Inicialización.** El DSP crea su cola, y se engancha a la cola del ARM, quedando así establecida la comunicación entre ambos núcleos. Una vez hecho esto, el DSP envía la tabla de funciones con las tareas que puede realizar. En el código del programa, se corresponde con la función denominada *TSKMESSAGE_create*.

A partir de ahí, cada vez que se reciba un mensaje desde el ARM indicando que se ejecute, se realizará la tarea siguiendo siempre los mismos pasos.

- **Ejecución.** El DSP leerá en su cola el mensaje con la tarea a realizar y una vez concluida envía un mensaje al ARM como respuesta con los resultados obtenidos. Este paso se corresponde en el código con la función llamada *TSKMESSAGE_execute*.

El DSP permanecerá a la espera de nuevos mensajes de ejecución, pero también cabe la posibilidad de cerrar la comunicación.

- **Finalización.** Se libera la cola, dando por concluida la comunicación entre los núcleos, permaneciendo el DSP a la espera de una nueva llamada por parte del ARM. De igual manera, este paso tiene correspondencia en el código con la función llamada *TSKMESSAGE_delete*.

4.2 Adaptación del código a la tarjeta EVMK2H

Una vez explicado como estaba organizado el código en el proyecto anterior, tanto en el lado de ARM como en los ejecutables de los DSP, se describirá cual ha sido la organización que se ha decidido emplear en el presente proyecto.

4.2.1 Código para ARM

La herramienta GNU Radio, a partir de la versión 3.6.5 incorpora una nueva característica, que para el presente proyecto es de gran utilidad. En el anterior proyecto, la versión de GNU Radio empleada era la 3.3.0, por lo que aún no se encontraba disponible esta nueva característica.

Hasta la versión 3.6.5, si se desarrollaba un nuevo módulo para GNU Radio, con el fin de realizar una tarea específica dependiendo del proyecto que se esté realizando, este módulo de nueva creación debía incorporarse al directorio donde se encontrase ubicada GNU Radio. Por el contrario, en las versiones más actuales, es posible mantener los módulos de creación propia en otro directorio, creando así una separación, para no incluir ficheros de nueva creación en el directorio de GNU Radio.

Así mismo, incorpora una nueva herramienta, llamada `gr_modtool`, que permite crear nuevos módulos y su posterior gestión, permitiendo por ejemplo añadir bloques o actualizar los archivos `makefile` de manera sencilla, simplificando enormemente la labor del desarrollador, que puede centrarse en mayor medida en el código.

Por tanto, en la parte de ARM, se ha optado por reorganizar el código. Esta decisión se ha tomado debido a que los módulos que acabamos de introducir en el apartado anterior tienen una dependencia mutua, por lo que no tiene sentido ni aporta ninguna ventaja el hecho de que estén separados. Es por esto que, ya que se han de realizar modificaciones en los módulos, se ha tomado la decisión de modificar también la estructura del programa, unificando los dos módulos en uno solo.

Así se ha creado un único módulo, que contiene tanto los bloques de procesado con las tareas a enviar a los DSP como el bloque encargado de gestionar la comunicación entre núcleos. A este nuevo módulo se le nombró `gr_c6x`, en referencia a la familia de DSP presente en la tarjeta. Este módulo contiene en su interior dos bloques:

- C6X_add_ii. Contendrá una tarea de procesado, similar a la que se encontraba en el bloque `gdsp_add_ii` dentro del módulo `GR_GDSP`
- C6X_job_manager. En su interior se encontrará el bloque equivalente al que había en el módulo `GDSP`, encargado de gestionar y controlar el envío de mensajes entre núcleos. En este caso los archivos principales serán `c6x_job_manager.cc` y `c6x_job_manager_impl.cc`, renombrados así para mantener un orden.

Haciendo uso de la herramienta `gr_modtool` se creó por tanto el módulo `gr_c6x`. Con este fin, se siguieron las indicaciones que aporta GNU Radio

```
#gr_modtool newmod <nombre del módulo>
```

Una vez creado, se incluyeron los bloques en su interior, teniendo así un conjunto de archivos en los que ir incluyendo el código una vez modificado.

```
#cd gr-<nombre del módulo>
```

```
#gr_modtool add -t <tipo de bloque> <nombre del bloque>
```

En relación a esto, en `c6x_job_manager`, el archivo que más modificaciones sufrió con respecto a su equivalente en el código del proyecto anterior fue `c6x_job_manager_impl.cc`. A continuación se detallará como quedan las funciones (ver apartado 4.1.1) que más modificaciones han sufrido.

INICIALIZAR_DSP:

Es la primera función en ejecutarse y se encarga de crear la cola del ARM, cargar los ejecutables en los DSP, y engancharse a las colas creados por estos.

Puesto que el método de comunicación entre núcleos ha cambiado por completo, esta función tuvo que ser reescrita enteramente.

Tras inicializar `UDMA` primero, y `MSGCOM` después, se crea un hilo para ejecutar `AGENT` antes de crear las colas.

```
/* Create a thread to run the Agent Receive thread. */
if ((retVal = pthread_create (&agentThread, NULL,
    (PTHREAD_FN)agentInitTask, NULL)) < 0) //Agent failed
    exit (1);
```

Una vez inicializados los módulos necesarios para la comunicación, se cargan los ejecutables en los DSP con la función `mpm_load`, y se arrancan con la función `mpm_run`.

```
// Load the executable on to DSP's.
for(unsigned int i=0; i<8; i++)
{
    status = mpm_load (dspNames[i], executable, &error);
}

// Start execution on DSP's.
for(unsigned int i=0; i<8; i++)
{
    status = mpm_run (dspNames[i], &error);
}
```

Una vez hecho esto, se configura y crea, con la función `Msgcom_create`, la cola de mensajes del ARM.

```
/* Initialize the channel configuration. */
memset ((void *)&chConfig, 0, sizeof(Msgcom_ChannelCfg));

/* Populate the channel configuration. */
chConfig.mode = mode; //Blocking
/* Handle to the memory pool used for receive buffers */
chConfig.queueRingCfg.memHandle = memHandle;
/* Number of receive buffers */
chConfig.queueRingCfg.numPackets = 64;
/* Size of receive buffers */
chConfig.queueRingCfg.dataBufferSize = 2048;
/* Create the Message communicator channel. */
chHandle = Msgcom_create (channelName,
Msgcom_ChannelType_QUEUE_RING, (Msgcom_ChCfg*)&chConfig);
```

Para que esta cola sea visible desde el DSP, hay que notificárselo a AGENT para que realice la sincronización.

```
/* Push the created named resource to the remote processor. */
retVal = Agent_push (agentHandle, channelName,
ResMgr_NamedResourceType_INTERNAL_SYSLIB,
Agent_NamedResource_CREATE, &errCode);
```

Si la cola del ARM se crea correctamente, se pasa a buscar las colas de los DSP, para engancharse a ellas con la función `Msgcom_find`, y dejar así establecida la comunicación.


```

for(unsigned int i=0; i<ncores; i++) {
    strcpy(channelName, dspNames[i]);
    while (1)
    {
        // Check if the communicator channel has been created or not?
        chHandle = Msgcom_find (channelName, (Msgcom_ChCfg*)
&chConfig);
        if (chHandle != NULL) // Got a valid channel handle
            break;
        sleep (1);
    }
}

```

Una vez establecida la comunicación, se lee el primer mensaje enviado por los DSP, con la función `Msgcom_getmessage`, para completar la tabla.

```

if (Msgcom_getMessage (chHandle, (MsgCom_Buffer**)
&ptrQRingMsgBuffer) < 0)
    return -1;

```

Send_job:

Para enviar el mensaje, en primer lugar se reserva memoria para el mismo

```

/* Send out the messages to the reader. */
if ((ptrDataBuffer = reinterpret_cast<My_msg *>(udma_mem_block_alloc
(memHandle))) == NULL)

    return -1;

```

Una vez hecho, se completa el mensaje con la información a enviar.

```

dataLen = sizeof(msg);
msg.job = *jd;
msg.msg_id = 0;

*ptrDataBuffer = msg;

memset ((void *)&qRingPktBuffer, 0, sizeof (Msgcom_QRingMsgBuffer));

qRingPktBuffer.ptrBuffer      =  reinterpret_cast<uint8_t
*>(ptrDataBuffer);
qRingPktBuffer.bufferLen      =  dataLen;
qRingPktBuffer.memHandle      =  memHandle;
qRingPktBuffer.metaData       =  0;

```

Y una vez completado, se envía.

```
/* Send the message. */
if (Msgcom_putMessage (chHandleDSP, (MsgCom_Buffer *)&qRingPktBuffer) <
0)

    return -1;
```

Job_completer_loop

Al igual que en INICIALIZAR_DSP, se llama a la función `Msgcom_getMessage` para leer el mensaje, y una vez recibido se guarda.

```
/* Save the received message. */
msg = *(reinterpret_cast<My_msg *>(ptrQRingMsgBuffer->ptrBuffer));
```

4.2.2 Código para los DSP

En la parte relativa a los DSP, si bien se ha optado por mantener la organización que había con anterioridad, respetando inclusive el nombre de las funciones que se correspondían con cada uno de los pasos a dar, las modificaciones realizadas en el código han sido más profundas, ya que el grueso del mismo es el que se dedica a la comunicación: crear la cola, el envío y recepción de mensajes y la finalización de la comunicación.

Así, las funciones, ahora quedarían:

- *TSKMESSAGING_create*. Se encarga de la inicialización. Crea la cola con la función *Msgcom_create* y busca la cola del ARM para engancharse a ella con la función *Msgcom_find*.
- *TSKMESSAGING_execute*. Para leer el mensaje con la tarea a realizar, se emplea la función *Msgcom_getMessage*, y para enviar la respuesta con los resultados obtenidos, se utiliza la función *Msgcom_putMessage*.
- *TSKMESSAGING_delete*. Se encarga de cerrar la comunicación, utilizando para ello la función *Msgcom_delete*, una vez que se ha finalizado la ejecución y han enviado los correspondientes mensajes.

4.3 Modificaciones realizadas

En el presente apartado se describen las modificaciones más destacables que se han realizado, documentando por un lado las funciones que se empleaban en el código del proyecto anterior, y aquellas por las que se han sustituido, así como aquellas que se han suprimido por innecesarias.

Para poder conocer más a fondo y entender el funcionamiento de las funciones empleadas en la comunicación entre núcleos en la tarjeta EVMK2H, se procedió a crear un proyecto personalizado realizando modificaciones a los ya empleados para las comprobaciones de funcionamiento de los módulos (ver apartados 3.2.1 y 3.2.2). Así, se pudo identificar aquellas funciones empleadas para inicializar la comunicación, las utilizadas para el envío de mensajes y las funciones que usan para comprobar el correcto funcionamiento de la comunicación.

4.3.1 Modificaciones en el código

A continuación se detalla en una tabla las correlaciones entre las funciones más destacables de los módulos que se emplean para gestionar la comunicación en la presente tarjeta (ver apartados 3.2.1 y 3.2.2), con las funciones que se emplearon en el trabajo realizado con anterioridad que tienen una funcionalidad similar.

MSGCOM	
TARJETA IGEPv2	TARJETA EVMK2H
Función empleada para localizar una cola	
MSGQ_locate (Pstr MSGQ_Queue * queueName, MSGQ_LocateAttrs * msgqQueue, attrs);	MsgCom_ChHandle Msgcom_find (char* channelName, Msgcom_ChCfg* ptrChCfg)

Función empleada para poner un mensaje en la cola

```
MSGQ_put
(
    MSGQ_Queue    msgqQueue,
    MSGQ_Msg      msg
);
```

```
int32_t Msgcom_putMessage
(
    MsgCom_ChHandle    msgChHandle,
    MsgCom_Buffer*     msgBuffer
)
```

Función empleada para leer un mensaje de la cola

```
MSGQ_get
(
    MSGQ_Queue    msgqQueue,
    Uint32        timeout,
    MSGQ_Msg *    msg
);
```

```
int32_t Msgcom_getMessage
(
    MsgCom_ChHandle    msgChHandle,
    MsgCom_Buffer**    msgBuffer
)
```

Función empleada para crear una cola

```
MSGQ_open
(
    Pstr          queueName,
    MSGQ_Queue *  msgqQueue,
    MSGQ_Attrs *  attrs
);
```

```
MsgCom_ChHandle Msgcom_create
(
    char*          channelName,
    uint32_t       channelType,
    Msgcom_ChCfg*  ptrChCfg,
    int32_t*       errorCode
)
```

Función empleada para eliminar una cola

```
MSGQ_close
(
    MSGQ_Queue    msgqQueue
)
```

```
MsgCom_ChHandle Msgcom_delete
(
    char*          channelName,
    freePkt        freePkt
)
```

Función empleada para eliminar un paquete

```
MSGQ_free
(
    MSGQ_Msg      msg
)
```

```
void Pktlib_freePacket
(
    Ti_Pkt*       pPkt
)
```

Función empleada para asignar un mensaje

<pre>MSGQ_alloc (PoolId poolId, Uint16 size, MSGQ_Msg * msg);</pre>	<pre>Ti_Pkt* Pktlib_allocPacket (Pktlib_HeapHandle heapHandle, uint32_t size);</pre>
--	---

MPM

TARJETA IGEPv2

TARJETA EVMK2H

Función empleada para cargar un ejecutable en un DSP

<pre>PROC_load (Processor Id proclId, Char8 * imagePath, Uint32 argc, Char8 ** argv);</pre>	<pre>int mpm_load (const char * slave_name, const char * file_name, int * error_code);</pre>
---	--

Función empleada para arrancar un DSP

<pre>PROC_start (ProcessorId proclId);</pre>	<pre>int mpm_run (const char * slave_name, int * error_code);</pre>
---	--

También cabe destacar en el capítulo de modificaciones que se han suprimido algunos elementos. Tal y como se mencionó en el apartado 2.1.3, el sistema de comunicación entre núcleos ha cambiado, por lo que aquellas partes dependientes del ya extinto sistema DSPLink no pueden ser utilizadas.

- POOL. Se trata de un módulo de DSPLink, que servía para configurar el uso de zonas de memoria compartida.

- DSP_SUCCEEDED/DSP_FAILED. Eran dos macros que incluía DSPLink para comprobar si las llamadas realizadas a los DSP se habían realizado correctamente.
- MSGQ_setErrorHandler. Esta función se empleaba para crear una cola específica para recibir mensajes de error asíncronos. Actualmente, el módulo de comunicación MSGCOM no dispone de una función similar, por lo que se ha prescindido de ella.
- MSGQ_MsgHeader. Este atributo que se incorporaba en la estructura que se empleaba como mensaje estaba predefinido por DSPLink. Se ha eliminado puesto que la información que aportaba ya no es necesaria. Por tanto, la estructura que se emplea como mensaje contiene ahora únicamente la información relativa a la tabla, y el descriptor del trabajo.

```

Typedef struct My_msg_tag{
        gd_proc_def_t      tabla_id_fun;
        int                n_func;
        gd_job_desc_t      job;
}My_msg;

```

4.3.2 Modificaciones en los bloques

Respecto a los bloques, una de las diferencias reseñables, es la manera en que están codificados éstos en GNU Radio. Se ha observado que hay ligeras modificaciones entre el código generado por defecto en los bloques creados con la herramienta gr_modtool y el código de los bloques del proyecto anterior. Estas variaciones se deben a la versión de GNU Radio empleada.

Las variaciones afectan al modo en que se declaran los constructores de clases principalmente. También cabe destacar que las versiones más recientes de GNU Radio utilizan espacios de nombres, que son ámbitos en los que los nombres de las entidades declaradas son únicos, para así evitar solapamientos. Así, en los archivos de los bloques creados se incluyó:

```

namespace gr{
    namespace c6x{
        //Código...
    }
}

```

Este mínimo cambio, que a primera vista puede parecer poco relevante, no lo es tanto pues GNU Radio notifica errores al compilar y enlazar si no se mantiene el estilo que viene por defecto, de ahí que no sea posible hacer uso de las declaraciones hechas en el código del anterior proyecto directamente, sino que igualmente hubo que realizar una adaptación de las mismas, para mantener el estilo de declaración de constructores.

Otro cambio destacable que se ha realizado es el modo en que se pasan los datos de las operaciones entre núcleos.

En el anterior proyecto, en el bloque `gdsp_add_ii`, al enviar la tarea se copiaban los datos de entrada a la zona de memoria compartida y se pasaba un array de punteros a los datos de entrada y otro array de punteros a la zona en la que situar los datos de salida, también visibles desde el DSP.

En el presente proyecto, por el modo en que se realiza la comunicación y como se gestiona la zona de memoria compartida sería posible, si bien resultaría más complicado. Por este motivo, con el fin de simplificar, se ha optado por pasar los datos de manera directa, en lugar de un puntero a la dirección de memoria en que se encuentran.

4.3.3 Modificaciones en la compilación

Por último, se realizó otra modificación de importancia. Al enlazar, solo se pudieron crear librerías estáticas (con extensión `.a`), debido a que las librerías proporcionadas por TI para poder hacer uso de los módulos MPM y MSGCOM son estáticas también.

Cabe la posibilidad de crear una librería compartida (con extensión `.so`) a partir de librerías estáticas, siempre y cuando éstas últimas hayan sido compiladas con la opción `-fPIC`, hecho que no ocurre en las librerías proporcionadas por TI, por lo que como se ha comentado, solo se crearon librerías estáticas.

En GNU Radio, para poder hacer uso de bloques escritos en C++ en un grafo codificado en Python, es necesaria la herramienta SWIG, que permite que elementos que han sido escritos en diferentes lenguajes de programación puedan interactuar entre ellos.

SWIG únicamente utiliza librerías compartidas, por lo que hubo que prescindir de esta herramienta y codificar los grafos de GNU Radio en C++, al contrario que en el proyecto anterior, en el que si se podía hacer uso de SWIG y por tanto los grafos fueron escritos en Python.

Por tanto, el código de los grafos tuvo que ser reescrito por completo en C++, si bien el código disponible sirvió como guía.

En el resultado de la compilación y enlazado, en la figura 20, se pueden observar las librerías creadas.

```

root@keystone-evm:~/blocks/gr-c6x/build# make
Scanning dependencies of target gnuradio-c6x-shared
[ 3%] Building CXX object lib/CMakeFiles/gnuradio-c6x-shared.dir/c6x_job_manager.cc.o
[ 7%] Building CXX object lib/CMakeFiles/gnuradio-c6x-shared.dir/c6x_job_manager_impl.cc.o
[ 11%] Building CXX object lib/CMakeFiles/gnuradio-c6x-shared.dir/c6x_add_ii_impl.cc.o
[ 15%] Building C object lib/CMakeFiles/gnuradio-c6x-shared.dir/c6x_jd_stack.c.o
[ 19%] Building C object lib/CMakeFiles/gnuradio-c6x-shared.dir/osal.c.o
Linking CXX shared library libgnuradio-c6x-shared.so
[ 19%] Built target gnuradio-c6x-shared
Scanning dependencies of target gnuradio-c6x-static
[ 23%] Building CXX object lib/CMakeFiles/gnuradio-c6x-static.dir/c6x_job_manager.cc.o
[ 26%] Building CXX object lib/CMakeFiles/gnuradio-c6x-static.dir/c6x_job_manager_impl.cc.o
[ 30%] Building CXX object lib/CMakeFiles/gnuradio-c6x-static.dir/c6x_add_ii_impl.cc.o
[ 34%] Building C object lib/CMakeFiles/gnuradio-c6x-static.dir/c6x_jd_stack.c.o
[ 38%] Building C object lib/CMakeFiles/gnuradio-c6x-static.dir/osal.c.o
Linking CXX static library libgnuradio-c6x-static.a
[ 38%] Built target gnuradio-c6x-static
Scanning dependencies of target test-c6x
[ 42%] Building CXX object lib/CMakeFiles/test-c6x.dir/test_c6x.cc.o
[ 46%] Building CXX object lib/CMakeFiles/test-c6x.dir/qa_c6x.cc.o
[ 50%] Building CXX object lib/CMakeFiles/test-c6x.dir/qa_c6x_add_ii.cc.o
[ 53%] Building CXX object lib/CMakeFiles/test-c6x.dir/qa_c6x_add_ii_t2.cc.o
[ 57%] Building CXX object lib/CMakeFiles/test-c6x.dir/qa_c6x_job_manager.cc.o
[ 61%] Building C object lib/CMakeFiles/test-c6x.dir/osal.c.o
Linking CXX executable test-c6x
[ 61%] Built target test-c6x
Scanning dependencies of target c6x_swig_swig_doc
[ 61%] Built target c6x_swig_swig_doc
Scanning dependencies of target _c6x_swig_swig_tag
[ 65%] Building CXX object swig/CMakeFiles/_c6x_swig_swig_tag.dir/_c6x_swig_swig_tag.cpp.o
Linking CXX executable _c6x_swig_swig_tag
[ 65%] Built target _c6x_swig_swig_tag
[ 69%] Generating c6x_swig.tag
[ 73%] Swig source
/home/root/blocks/gr-c6x/include/c6x/c6x_job_manager.h:148: Warning 319: No access specifier
/home/root/blocks/gr-c6x/include/c6x/c6x_job_manager.h:76: Warning 451: Setting a const char
Scanning dependencies of target _c6x_swig
[ 76%] Building CXX object swig/CMakeFiles/_c6x_swig.dir/c6x_swigPYTHON_wrap.cxx.o
Linking CXX shared module _c6x_swig.so
[ 76%] Built target _c6x_swig
Scanning dependencies of target pygen_swig_213e9
[ 80%] Generating c6x_swig.pyc
[ 84%] Generating c6x_swig.pyo
[ 92%] Built target pygen_swig_213e9
Scanning dependencies of target pygen_python_ab77a
[ 96%] Generating __init__.pyc
[100%] Generating __init__.pyo
[100%] Built target pygen_python_ab77a
Scanning dependencies of target pygen_apps_9a6dd
[100%] Built target pygen_apps_9a6dd
root@keystone-evm:~/blocks/gr-c6x/build#
root@keystone-evm:~/blocks/gr-c6x/build#

```

Figura 20. Resultado de la compilación y el enlazado

4.4 Pruebas de rendimiento

Una vez que se hubo creado la aplicación, se procedió a realizar la comprobación del rendimiento de la misma en la tarjeta EVMK2H.

Para ello, se han creado distintas aplicaciones en GNU Radio, que serán implementadas como grafos, que estarán codificados en C++.

En particular, en estos grafos se han incluido por un lado el módulo creado, que calcula una suma de vectores, y por otro lado el módulo *add_ii* ya creado de antemano por GNU Radio que realiza la misma función, pero sobre el ARM.

4.4.1 Grafos de prueba

Se han creado por tanto varios grafos, para estudiar el rendimiento, realizando la misma operación de calcular una suma de vectores, comprobando así las diferencias obtenidas al ejecutar la misma operación en los diferentes núcleos, bien sobre el ARM, bien sobre los DSP.

- ARM

En este primer grafo, se realiza todo el trabajo sobre el ARM, quedando por tanto una estructura como la que se aprecia en la figura 21.

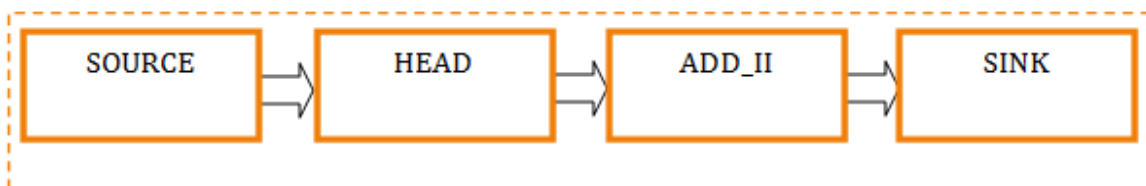


Figura 21. Grafo ARM

- DSP

Este grafo tiene una estructura igual al primero, pero en este caso sustituiremos el módulo prefabricado de GNU Radio, por el realizado en este proyecto, para así realizar los cálculos en los DSP.

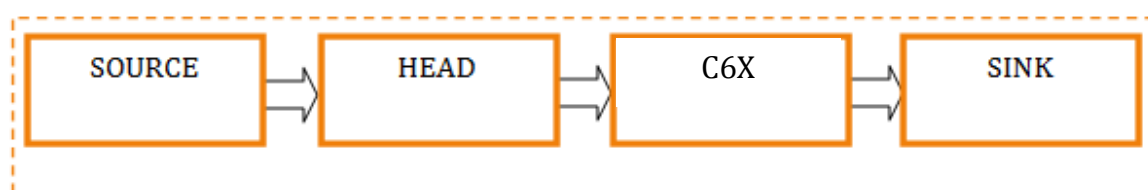


Figura 22. Grafo DSP

4.4.2 Resultados obtenidos

Al ejecutar los grafos, se comprobó que los resultados podían variar ligeramente, pero nunca de manera significativa. Por tanto, los datos proporcionados a continuación para ser interpretados son representativos.

- ARM

```

Running tests...
Test project /home/root/blocks/gr-add_arm_ii/build
  Start 1: test_add_arm_ii
1/2 Test #1: test_add_arm_ii ..... Passed    0.16 sec
  Start 2: qa_add_ii
2/2 Test #2: qa_add_ii ..... Passed    1.54 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) =  1.87 sec
  
```

Figura 23. Resultado de ejecución del grafo ARM

El resultado obtenido para este primer grafo, si lo comparamos con la medición realizada en el proyecto anterior, se aprecia una clara diferencia.

IGEPv2	EVMK2H
7,748 s	1,87 s

El tiempo de ejecución ha disminuido sensiblemente, como era de esperar, debido fundamentalmente al incremento de la frecuencia de reloj de los núcleos ARM (1400 MHz frente a 800) y al mejor rendimiento de los núcleos Cortex A15 frente al Cortex A8 del OMAP 3530, y también, aunque afecte en menor medida a la prueba realizada, al aumento del número de núcleos ARM (4 frente a 1).

- DSP

En el caso de los DSP, únicamente se ha comprobado que se obtienen los resultados de manera correcta al ejecutar la aplicación, si bien no se han podido realizar medidas del tiempo de ejecución.

Esto se debe a que los bloques del módulo gr-c6x están desarrollados partiendo de los proyectos proporcionados por TI (ver apartado 3.2.2), por lo que están configurados para que los ejecutables sean cargados en los DSP desde CCS.

Por tanto, los tiempos medidos no se correspondían con el verdadero tiempo de ejecución de la aplicación, al estar influido por otros factores, como el tiempo de impresión en pantalla de mensajes de control.

Para poder realizar las medidas de manera apropiada habría que realizar modificaciones significativas, lo cual no fue posible a fecha de finalización de redacción de la presente memoria.

Se ha evaluado por tanto únicamente el correcto funcionamiento, no siendo posible evaluar el rendimiento que se obtiene al ejecutar la tarea en un DSP.

5

Conclusiones y trabajo futuro

5.1 Conclusiones

En lo que respecta a las conclusiones tras realizar este proyecto, se puede decir que se ha conseguido alcanzar el objetivo principal. Ha sido posible portar con éxito la herramienta de procesamiento GNU Radio a la tarjeta EVMK2H.

En cuanto a la adaptación del trabajo anteriormente realizado, por un lado se ha comprobado el funcionamiento de los nuevos sistemas de comunicación entre núcleos, haciendo uso de los módulos MSGCOM y MPM, para después compararlo con los módulos anteriores de los que hacía uso DSPLink y evaluar las diferencias existentes entre ambos. Desde el punto de vista del programador, en lo que al código respecta, el cambio no ha supuesto ninguna mejora significativa pues no se ha simplificado apenas, siendo necesarias prácticamente las mismas funciones por lo que el código tiene una extensión similar.

A lo largo del desarrollo del presente proyecto se han encontrado también contratiempos, que si bien no se mencionan a lo largo de la memoria, han afectado notablemente a su ejecución. De entre estas contrariedades, la más destacable es la falta de información. Los procesadores Keystone II y la tarjeta EVMK2H llevan poco tiempo en el mercado, y Texas Instruments los ha comercializado antes de completar la documentación técnica, que es escasa, en algunos casos se halla incompleta y no es sencilla de localizar. Esto dificulta considerablemente el trabajo, teniendo en cuenta además la gran complejidad de la arquitectura de estos SoC.

Además, al contrario que en otras plataformas con más tiempo a la venta y que por tanto han sido puestas a prueba por un número de usuarios considerable, en la tarjeta EVMK2H muchos de los errores que se han encontrado están sin documentar. Por todo ello las soluciones planteadas han partido de criterios propios, tras descartar diferentes alternativas.

Hay que añadir también la escasa estabilidad del entorno de desarrollo CCS. Por ejemplo, la conexión entre el depurador y el emulador XDS200 falla intermitentemente, y el propio CCS se reinicia sin motivo aparente con cierta frecuencia.

Todos estos inconvenientes han prolongado notablemente el trabajo desarrollado, y por ese motivo no se han podido realizar pruebas exhaustivas del rendimiento del sistema.

5.2 Trabajo futuro

En lo que concierne a trabajos futuros, hay varias líneas para continuar:

La principal y más destacable opción es la de integrar el receptor DVB-H realizado con la herramienta GNU Radio por Daniel Sánchez Villalba en la tarjeta EVMK2H, haciendo uso plenamente de su capacidad de procesado.

Otro aspecto relevante, sería el de realizar una aplicación algo más compleja. En el presente proyecto se ha empleado una modificación de la aplicación desarrollada anteriormente, que únicamente hacía uso de un solo DSP. La actual tarjeta EVMK2H dispone de 8 de estos núcleos, por lo que sería interesante desarrollar una aplicación que usase varios DSP en paralelo.

De igual manera, también sería interesante la idea de desarrollar otros bloques en GNU Radio, para realizar otras tareas de procesado.

Referencias

1. Daniel Sánchez Villalba. Receptor DVB-H basado en GNU Radio. PFC de la EUIT de telecomunicación; Septiembre, 2010.
2. Alberto Vélez Felguera. Portado de GNU Radio a los procesadores OMAP 3530 y DaVinci DM6446. PFC de la EUIT de telecomunicación; Septiembre, 2010.
3. Carlos Sánchez Martín. Adaptación de GNU Radio a arquitecturas multiprocesador de punto fijo. PFC de la EUIT de telecomunicación; Septiembre, 2012.
4. Sitio web de Advantech. Información y características del módulo de evaluación EVMK2H. Accesible desde:
<http://www.advantech.com/Support/TI-EVM/EVMK2HX.aspx>
5. Sitio web de Texas Instruments. Descripción e información técnica del SoC 66AK2H14. Accesible desde:
<http://www.ti.com/product/66AK2H14>
6. Texas Instruments. SPRY223: Enhancing the Keystone II architecture with multicore RISC processing; 2012
7. Sitio web de Arago Project. Accesible desde:
<http://arago-project.org>
8. Sitio web de Texas Instruments. Guía de usuario para MCSDK de Keystone II. Accesible desde:
http://processors.wiki.ti.com/index.php/MCSDK_User_Guide_for_KeyStone_II
9. Sitio web de GNU Radio. Accesible desde:
<http://gnuradio.org>
10. Sitio web de Texas Instruments. Setting up a TFTP Server. Accesible desde:
http://processors.wiki.ti.com/index.php/Setting_Up_a_TFTP_Server
11. Sitio web de IGEP Community. Set up a Network File System between IGEP Board and IGEP Virtual Machine. Accesible desde:
http://labs.isee.biz/index.php/Set_up_a_Network_File_System_between_IGEP_Board_and_IGEP_Virtual_Machine
12. Sitio web de arago-project. Repositorio. Accesible desde:
<http://arago-project.org/git/projects>