Revisión	Fecha	Comentario	Autor
0	20/09/2010	Display: Ampire AM640480GTM QW-T00H Módulos: Digi CC9C S.O.: Linux kernel 2.6	Ulises Bigliati

Índice

Introducción

En esta ocasión evaluamos el kit de desarrollo Digi Embedded Linux 4.0 basado en el módulo ConnectCore 9C[1]. Para las pruebas utilizamos la placa de desarrollo que se incluye en el kit y de manera opcional agregamos un display LCD de 640x480 con panel de touch screen. Los detalles del hardware referidos a la conexión del display no serán abordados en esta nota ya que son análogos al trabajo realizado en la nota CoAN-019[2]

Las herramientas de desarrollo de software que ofrece Digi para su Embedded Linux consisten en una instalación del framework Eclipse, desde la cual se ofrece la posibilidad de desarrollar proyectos que involucren la compilación del kernel de linux, módulos cargables, el bootloader, del sistema de archivos, y por supuesto, las aplicaciones de usuario.
El entorno de desarrollo puede instalarse de diferentes formas según la necesidad del usuario:

1) Ejecutando la imagen booteable de Kubuntu contenida en el DVD, la cual ya presenta el entorno de desarrollo completamente instalado (modo de evaluación).
2) Instalando solo el entorno de desarrollo sobre una distribución preexistente de linux[3] con la que ya cuente el usuario.
3) Instalando la distribución de Kubuntu completa provista por Digi, la cual incluye la

[1] http://www.digi.com/support/productdetl.jsp?pid=2863&osvid=0&s=227&tp=1
[2] Allí se conectaba el mismo display, al mismo módulo mediante la misma placa de desarrollo, solo que para la ejecución de WinCE.
[3] En caso de seleccionar esta opción es altamente recomendable que la distribución sobre la cual se instale el entorno de desarrollo sea la misma que viene en el DVD de instalación. Esto evitará cualquier tipo de inconsistencia y/o conflicto entre versiones.

instalación automática del entorno de desarrollo (recomendado).

Para nuestras pruebas utilizamos esta última alternativa. Al momento de la realización de este trabajo la versión del entorno de desarrollo Digi Embedded Linux con la que contábamos era la 4.0 (DigiEL 4.0 del 2007). Está instalada sobre Kubuntu 6.06 LST (Ubuntu con entorno gráfico KDE y la versión del kernel de Linux es la 2.6.17.
Cabe mencionar que el Embedded Linux que correrá en los módulo de Digi no se corresponde con ninguna de las distribuciones más populares, sino que se trata de un trabajo del fabricante para portar el kernel de linux, los device drivers, módulos cargables y otras utilidades junto con las aplicaciones de usuario para ser todo ejecutado sobre sus módulos basados en ARM9.
A continuación vamos a ejemplificar el uso de estas herramientas llevando a cabo un proceso de desarrollo típico sobre Linux embebido.

Instalación

Para la instalación del entorno de desarrollo elegimos la instalación completa. Esto lo realizamos en nuestro caso sobre una máquina virtual [4] cuyos detalles de configuración exceden los alcances de esta nota.

Luego de bootear el host[5] desde el DVD al elegir la instalación completa, se instalará una distribución de Kubuntu y junto con ella tendremos listo para usar el entorno de desarrollo Digi ESP basado en Eclipse para Linux embebido.

Por otra parte también se instalarán y configurarán automáticamente algunas herramientas tales como un servidor TFTP y NFS junto con sus directorios de operación y los permisos apropiados.

Finalmente, para completar la instalación debemos contar con una consola serial[6] para interactuar con el módulo a través de su puerto serial A (placa de prototipos mediante) con esta conexión configurada en *38400,N,8,1* se podrá interactuar con el intérprete de comandos de U-Boot lo cual es indispensable para la puesta en marcha del sistema.

La instalación de Digi Embedded Linux permitirá la realización de proyectos que involucren, según la necesidad del usuario, el desarrollo sobre el bootloader (U-Boot), el kernel 2.6 de linux, el file system (nfsroot sobre JFFS2), módulos cargables y las aplicaciones de usuario.
Habiendo completado la instalación, se presentará un ícono en el escritorio para el acceso directo al entorno de desarrollo y ya se podrá comenzar a trabajar con Linux embebido.

Los proyectos

Para trabajar con Digi Embedded Linux, se convivir con el concepto de proyecto, lo cual se materializa en un directorio que contiene todos los componentes de software requeridos para construir una solución completa para la plataforma de hardware de nuestro sistema, esto puede involucrar, el kernel, el rootfs, el boot loader y las aplicaciones de usuario.

Continuando con el concepto de proyecto, sigue la idea de *Workspace*, que es simplemente un lugar común donde depositar nuestros proyectos, es decir, que es otro directorio.

El proceso de compilación se ejecutará con simples comandos que generarán las imágenes binarias del kernel, y del file system conteniendo las aplicaciones de usuario. El proceso de compilación tiene lugar en el directorio del proyecto con permisos normales de usuario.

---

[4] *www.**virtualbox**.org*
[5] En nuestro caso se iniciamos el guest preparado para Linux de nuestra máquina virtual.
[6] Por ejemplo, seyon o minicom (ver puntos 2.2.1 y 2.2.2 en *Digi ConnectCore User's Guide for Command Line Tools*).

El entorno de desarrollo incluye el código fuente completo del kernel de Linux, y las herramientas para personalizar el rootfs, el boot loader y para construir las aplicaciones de usuario que permitan diseñar el sistema embebido de acuerdo a las necesidades de la aplicación.

La plataforma de hardware

Aprovechamos el mismo hardware utilizado en la nota CoAN-018, esto es, el display VGA y el panel de touch screen. Los detalles de configuración del hardware no los repetiremos acá, ya que están descriptos en la nota CoAN-018 y solo a modo de recordatorio diremos que la conexión del display se realiza sobre el conector P18 (LCD_HEADER) de la placa de prototipos, y que mediante dicho conector también se establece la interfaz con el controlador ADS7846 del touch screen que ocupa el puerto B del módulo en modo SPI. Adicionalmente debemos configurar el switch SW2.2 de la placa de prototipos en posición OFF para habilitar el funcionamiento del puerto en modo SPI.

El proyecto de Linux embebido

Como se ha dicho anteriormente, un proyecto de linux embebido puede involucrar diferentes componentes: el boot loader, el sistema de archivos, el kernel, las aplicaciones de usuario.
De esta manera, de acuerdo a los requerimientos de la aplicación, el desarrollador seleccionará los componentes necesarios para su proyecto.
Probablemente, y sobre todo al comienzo, el desarrollador creará un proyecto completo, quizá exceptuando la inclusión del boot loader, ya que por lo general, ese componente no es objeto de grandes modificaciones.
Así se obtendrá un imagen del kernel adecuada a la configuración del hardware, y una imagen del sistema de archivos, junto con las aplicaciones y módulos que fueran necesarios, conforme lo requiera la aplicación a desarrollar.
Luego de lograr una imagen del kernel operativa que satisfaga los requisitos, ya no será necesario trabajar sobre ella, de tal forma solo estará en juego el desarrollo de la aplicación de usuario.
La imagen del kernel continuará siendo la misma, y el rootfs (sistema de archivos) probablemente también permanezca inmutable hasta tanto la aplicación de usuario haya madurado lo suficiente como para ser incluida definitivamente en aquel, es decir, se encuentre lista para pasar a producción. Hasta tanto esto no se produzca, la aplicación de usuario compilada será testeada en RAM, no siendo necesario su incorporación al rootfs cada vez que se quiera ejecutar durante el proceso de desarrollo.
Así, algunas de las opciones serán debugging del programa de usuario paso a paso desde el IDE, o envío del ejecutable mediante FTP (normalmente al directorio */tmp*) y operar luego vía Telnet.

Creación del proyecto

Podemos ejecutar el entorno de desarrollo mediante el ícono “Digi ESP” que se habrá creado en el escritorio (fig. a).
A continuación estaremos en condiciones de crear un nuevo proyecto, tal como puede verse en la figura b.
`File->New->Digi EL Kernel/Rootfs/U-Boot`

Fig. a

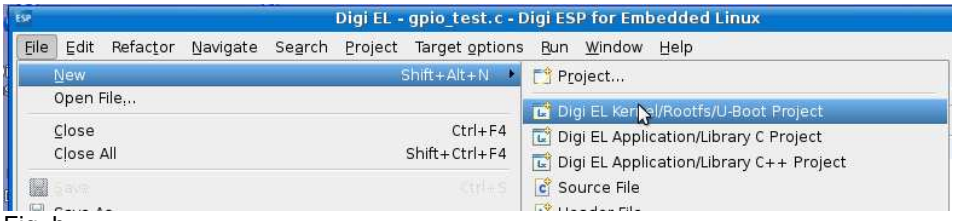


Fig. b

Seguidamente se habrá abierto el siguiente diálogo, desde el cual tenemos la oportunidad de indicar el nombre del proyecto, la plataforma de hardware sobre la cual estamos trabajando y los componentes del S.O. sobre los cuales vamos a trabajar (fig. c)

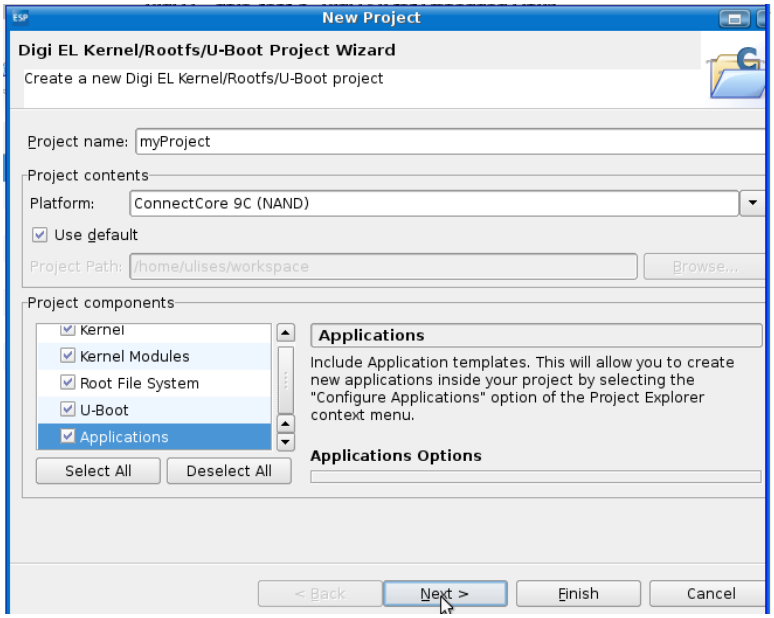


Fig. c

A continuación tenemos la oportunidad de indicar los directorios de salida de nuestras compilaciones, es decir: el directorio NFS, que por defecto es "*/exports*" y el directorio TFTP, por defecto "*/tftpboot*". Estos directorios son definidos y configurados durante la instalación.

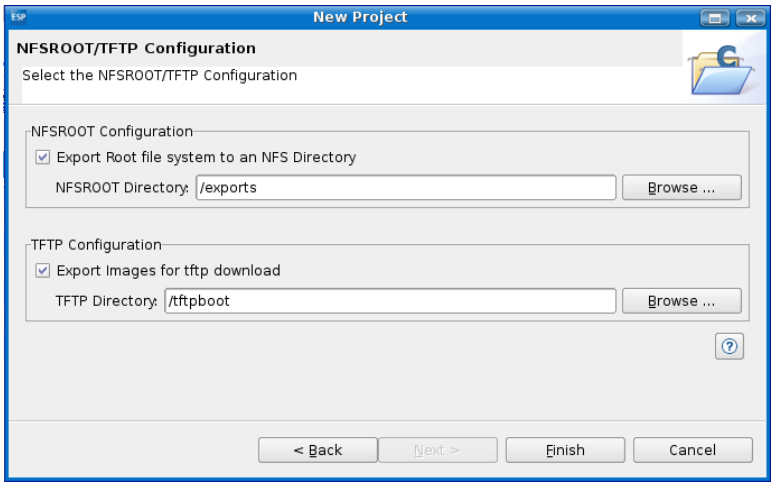


Fig. d

Una vez completados estos sencillos pasos y tras unos pocos minutos, el entorno de desarrollo habrá generado el proyecto "myProyect" (así lo nombramos) dentro del *workspace* que hayamos definido en la instalación (por default es /home/[user]/workspace). Veremos entonces en el explorador de proyectos del entorno de desarrollo el árbol con los objetos correspondientes al proyecto que acabamos de definir y ya estamos en condiciones de trabajar con el. Una de las primeras acciones que tomaremos, al menos para este caso de ejemplo será la configuración del kernel para adecuar su funcionamiento al hardware que tendrá que controlar. Esto es, fundamentalmente el display y el touch screen y con relación a este último, los puertos seriales.

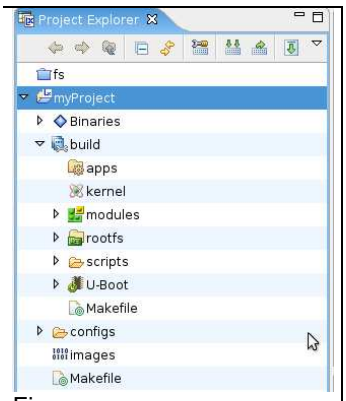


Fig. e

Configuración del sistema[7]

Para ajustar el kernel de Linux a nuestros requerimientos, podremos activar y configurar o eliminar controladores de dispositivo y/o servicios que no necesitemos consiguiendo una imagen pequeña y más eficiente.
Además, podremos definir que partes del S.O. estarán embebidas en el kernel, y cuales funcionaran como módulo cargables, estos módulos pueden cargarse y descargarse dinámicamente.

La configuración del kernel se realiza ejecutando la herramienta de configuración del proyecto, que está basada en una utilidad gráfica estándar para la compilación del kernel de Linux[8], la cual puede verse en la siguiente figura:

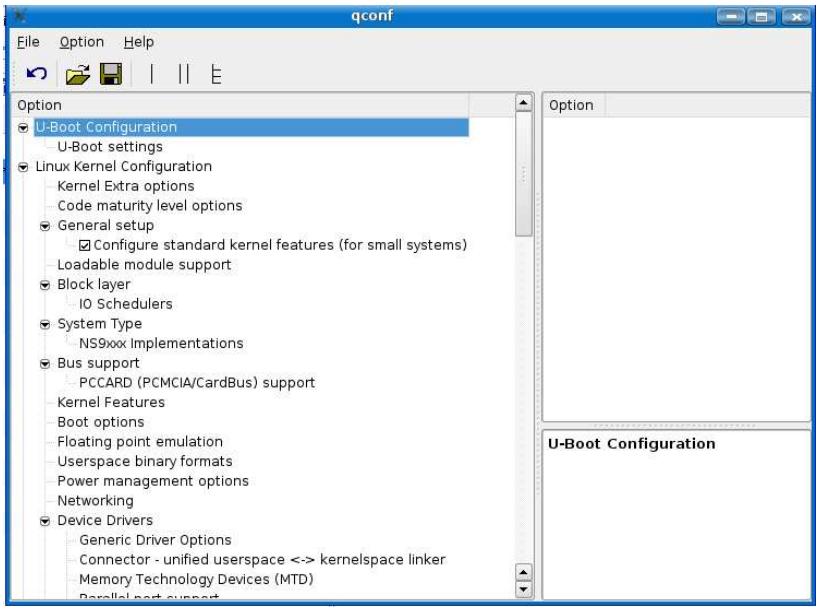


Fig. f

Se verá que la mayoría de las opciones son tristate: el objeto puede excluirse de la compilación (N), puede incluirse directamente dentro del kernel(Y), o puede compilarse como un módulo(M). En la pantalla de configuración del kernel estos tres estados se representan gráficamente por una casilla de verificación que puede estar en blanco, checkeada, con un circulo negro en el centro respectivamente.
Los módulos compilados junto al kernel como servicios o directamente como parte de el, estarán disponibles inmediatamente cuando el kernel arranque. En cambio, los módulos son piezas de código que se cargan y descargan bajo demanda. En la práctica, son archivos normales que, al cargarse, extienden la funcionalidad del kernel sin necesidad de rebootear el sistema.
Cabe mencionar que si bien los módulos son atractivos mientras dura el desarrollo o para hardware que puede cambiar con el tiempo, lo recomendable es que una vez que las características del hardware y los servicios que deban ser proporcionados por el kernel sean compilados como parte de este y no como módulos cargables.

El driver de video

Volviendo a nuestro ejemplo, ahora debemos incluir en el kernel el soporte gráfico, para esto seleccionamos el driver LQ064V3DG01 que corresponde a una resolución de 640x480. Y eso es todo.

[7] El material de referencia para estas operaciones consiste en el *Digi ConnectCore User's Guide for Command Line Tools* .
[8] Esta utilidad es xconfig, que a su vez se basa en las libraries gráficas QT.

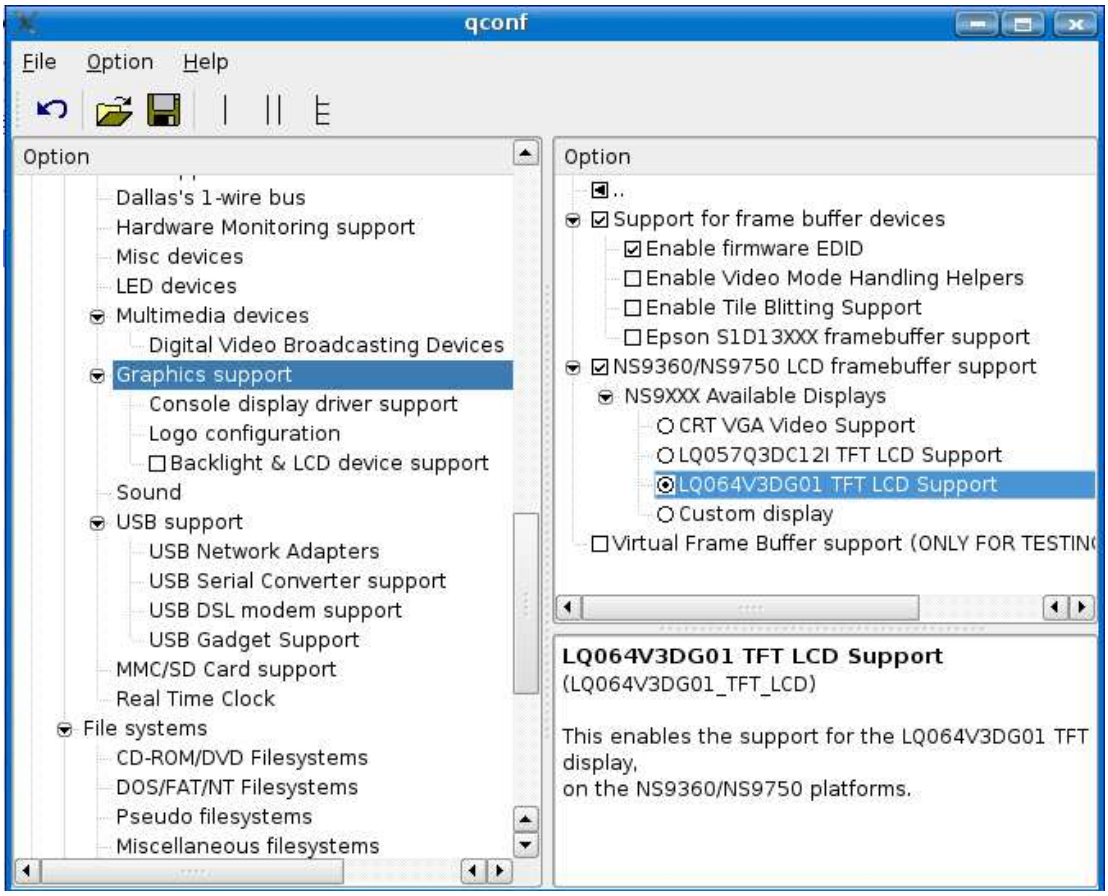


Fig. g

El driver para el touch screen

Seguimos con nuestro ejemplo, y ahora vamos a agregar el soporte para el touch screen. Primero, debemos deshabilitar el puerto serial B en modo UART, ya que lo necesitaremos en modo SPI.
El resto, (C y D) quedan afectados por la utilización del display, así que no pueden activarse, y el puerto A queda como consola de comunicación serial[9] (Fig. h).

A continuación debemos activar el puerto B en modo SPI, como puede verse en la fig. i

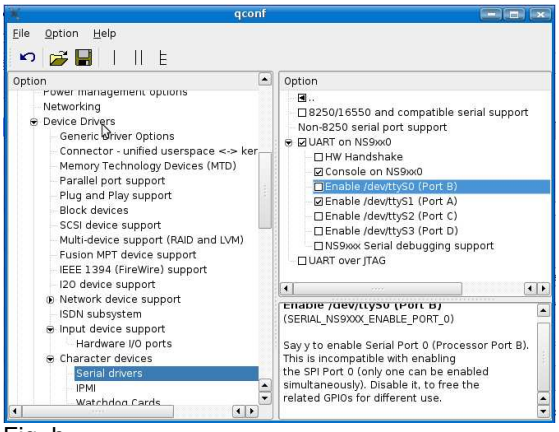


Fig. h

[9] Ver la documentación “Hardware Reference” correspondiente al módulo Digi ConnectCore 9C para obtener detalles sobre la disponibilidad de periféricos.

Finalmente, en la categoría “Input Devices” seleccionamos el driver para el touch screen ADS7846.
Nótese que hay cuatro parámetros configurables correspondientes con valores de coordenadas mínimos y máximos que pueden ser establecidos por el usuario para adaptarlos a las diferentes resoluciones de pantalla que

Habiendo llegado hasta acá habremos terminado de configurado el driver de video y el driver del panel de touch screen.

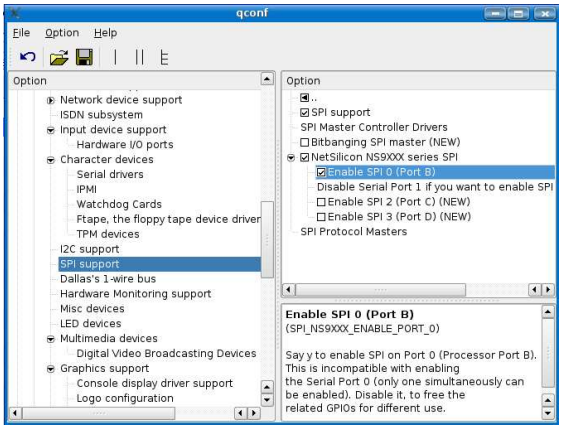


Fig. i

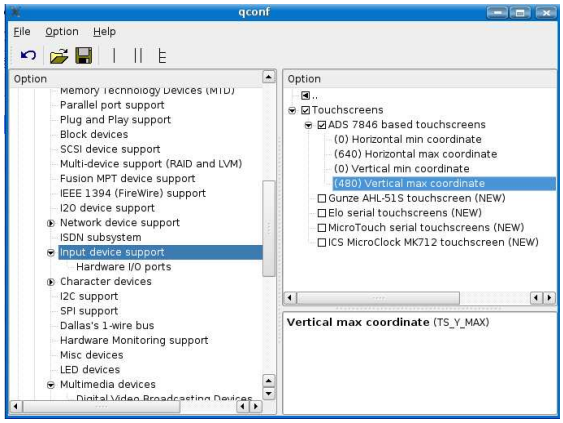


Fig. j

Aplicaciones gráficas de ejemplo (configuración de rootfs)

Embedded Linux incluye como ejemplo las librerías gráficas Qtopia y un par de aplicaciones de demostración que pueden ser agregadas al sistema de archivos, siempre que se haya incluido el componente “Root File System” al proyecto.
Estas librerías proporcionan un potente soporte gráfico para aplicaciones embebidas ofreciendo, soporte para mouse (el touch screen funcionará también como un mouse), manejo de ventanas, cajas de texto, manejo de eventos, etc.

Para incluir en la imagen del file system las aplicaciones gráficas de demostración que utilizan Qtopia, debemos abrir la interfaz de configuración y en Rootfs configuration > Pre-built applications seleccionar el elemento “Qtopia core example applications”. Esta selección incluye dos aplicaciones y las libraries *Qtopia core* en la imagen rootfs. Se debe considerar que la inclusión de estos elementos agrega a la imagen rootfs unos 9MB.

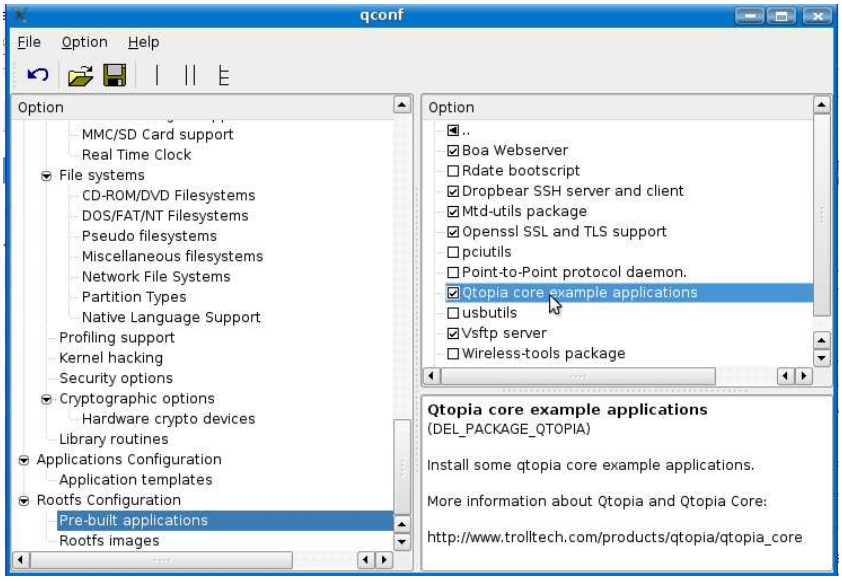


Fig. k

Los ejecutables de las aplicaciones quedarán almacenados en el directorio `/usr/bin/qtopia` del rootfs.
Para ejecutarlas:

```
./application -qws
./spreadsheet -qws
./ftp -qws
```

Otras configuraciones sobre rootfs

Continuando con la interfaz de configuraciones del sistema, y en relación con el file system, tenemos la posibilidad de seleccionar el tipo de file system que vamos a utilizar en bajo nivel como rootfs:

- Network File System (NFS) [10]
- Journaling Flash File System 2 (JFFS2) (http://en.wikipedia.org/wiki/JFFS2)
- Compressed ROM filesystem (CRAMFS) (http://en.wikipedia.org/wiki/Cramfs)
- ROM filesystem (ROMFS) (http://romfs.sourceforge.net/)
- Initial ramdisk filesystem (INITRD) (http://en.wikipedia.org/wiki/Initrd)

[10] El servidor NFS se configura automáticamente durante la instalación de DEL, en caso de requerir una instalación manual, ver el punto “2.3.2. NFS server” del manual “Digi ConnectCore User's Guide for Command Line Tools”

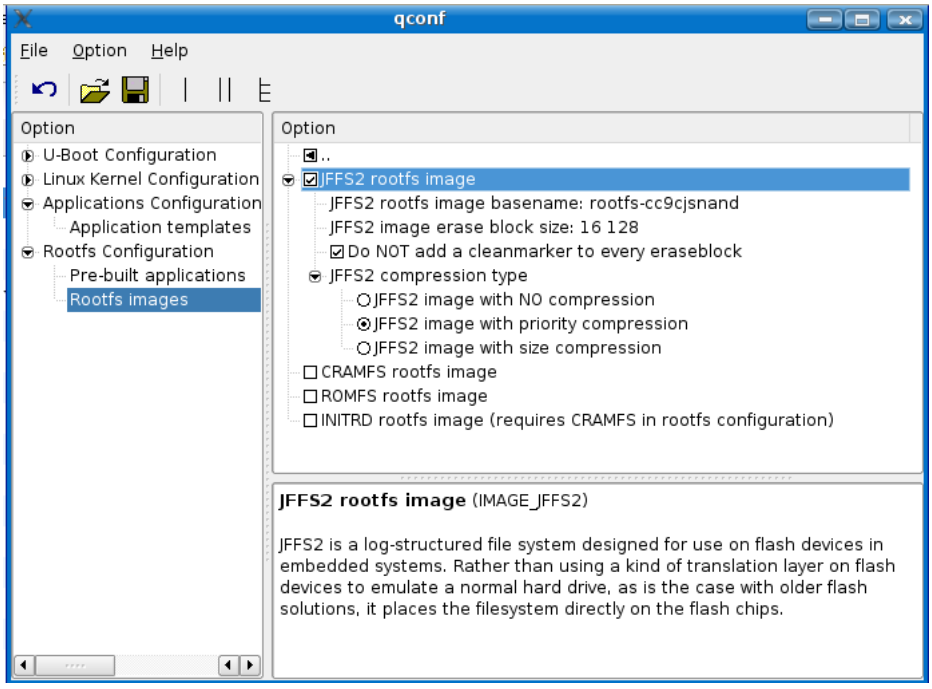


Fig. l

De acuerdo con las selecciones que realicemos aquí, se crearán las correspondientes imagenes del rootfs. En nuestro caso y en general, esta opción quedará establecida por defecto.

Cómo indicarle al kernel que tipo de rootfs tiene que usar?

La indicación de que tipo de rootfs se va a usar debe pasarse como argumento al kernel en el booteo. Esto se hace editando las variables de entorno de U-Boot. La variable que indica el tipo de file system es `rootfstype`, que a su vez corresponde a la variable de entorno `bootargs`.

Aplicaciones de usuario

Desde la interfaz de configuración del sistema tenemos la posibilidad de agregar diversas aplicaciones de ejemplo que nos serán útiles para emplear como plantillas para nuestras propias aplicaciones, para la ocasión agregaremos todas las disponibles, a fines de que su código fuente esté disponible desde el explorador de proyectos.

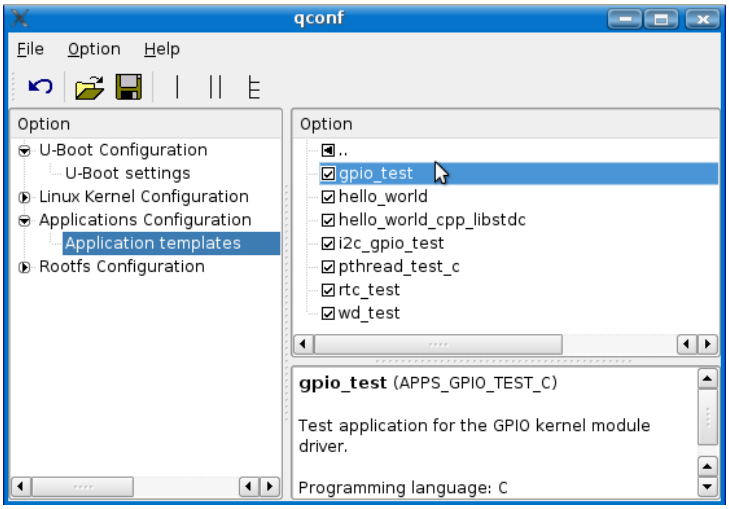
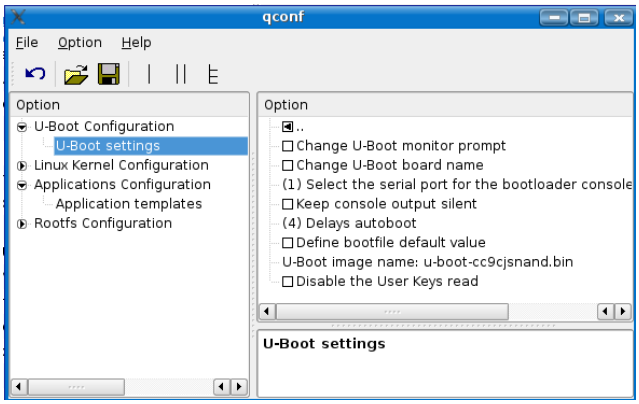


Fig. m

Configuraciones de U-Boot

Para finalizar esta sección de configuraciones mediante la interfaz gráfica debemos mencionar que si agregamos al boot loader como uno de los componentes del proyecto tendremos la oportunidad de modificar algunos parámetros de sus funcionamiento desde la ventana de configuraciones:



Algunas opciones, son de presentación, otras de configuración del canal de comunicación, y otra más interesante quizá como la que permite la posibilidad de utilizar un archivo conteniendo las variables de entorno (“Define bootfile default value”).

Fig. n

Cómo modificar el código fuente del kernel de Linux

El código fuente del Kernel está almacenado en un directorio común de la instalación de Digi Embedded Linux: `/usr/local/DigiEL-4.0/kernel/linux`. Cuando los proyectos incluyen componentes del kernel, solo los archivos de código objeto de dichos componentes y la imagen final del kernel es almacenada en la carpeta de proyecto del usuario. Es decir, el código fuente del kernel no está copiado a la carpeta de proyecto con el fin de salvar cientos de MB’s de espacio de disco.
Ahora supongamos el caso en que fuera necesario modificar el código fuente del kernel para agregar alguna funcionalidad o personalizar algún driver. Si el código fuente fuera editado directamente, los cambios realizados serían aplicados a todos los proyectos que incluyan el componente afectado, sin embargo, a menos que se trate de algún patch globalmente aceptado, esto no es lo que normalmente necesitamos.

Entonces, para editar el código fuente del kernel en forma local a un proyecto, se deben copiar los archivos que deben ser modificados dentro de la siguiente carpeta de nuestro proyecto: `/build/kernel`, teniendo cuidado de reproducir la misma ruta relativa que el archivo tenía en su ubicación dentro del directorio original del kernel.
Por ejemplo: para modificar el proceso de inicio del kernel, en particular el archivo `/usr/local/DigiEL-4.0/kernel/linux/init/`**`main.c`**
Se debe crear la misma ruta en el directorio `/build/kernel/` del proyecto y entonces se debe copiar allí el archivo a modificar:

```
mkdir -p build/kernel/init
cp /usr/local/DigiEL-4.0/kernel/linux/init/main.c build/kernel/init/
```

Y ahora si, va a ser posible editar el archivo local sin afectar el kernel original, obteniendo los cambios deseados solo en nuestro proyecto actual, porque durante la compilación, el entorno comprobará primero la existencia de copias locales de los archivos de código fuente y solo si estas no existen utiliza las del kernel original. En otras palabras, los archivos locales al proyecto ocultan a los globales.

Ejemplo: modificar parámetros del driver de video

En ocasiones, de acuerdo al hardware utilizado, puede ser necesario variar ciertos parámetros de funcionamiento del display que no son accesibles desde la interfaz gráfica de configuración, por

ejemplo, el centrado de la imagen en la pantalla. Cuando esto sucede, debemos modificar el header correspondiente al driver de video, para nuestro caso, tenemos que acceder al archivo `lq064v3dg01.h` Este archivo se encuentra en la ruta:
`/usr/local/DigiEL-4.0/kernel/linux/drivers/video/display/ns9xxx/`
[11] *Entonces, de acuerdo al lo dicho anteriormente, creamos el directorio siguiente* `build/kernel/drivers/video/display/ns9xxx` *dentro de la carpeta del proyecto. Luego copiamos allí el archivo que necesitamos, cuya ruta completa es:*
`/usr/local/DigiEL-4.0/kernel/linux/drivers/video/display/ns9xxx/lq064v3dg01.h`

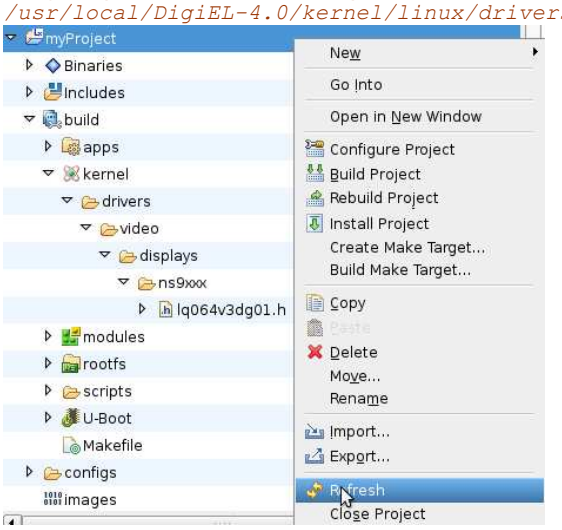


Fig. o

Si ejecutamos estos comandos desde un terminal tendremos que ejecutar el comando "refresh" sobre el proyecto para ver las cambios. Entonces allí tenemos el header del driver de video en forma local para modificarlo según nuestras necesidades y sin riesgos de dañar el código fuente original.

Para nuestro ejemplo, nos será útil disponer del header para regular como decíamos el centrado de la imagen en la pantalla. Nos interesan particularmente las macros:

```
LCD_TIMING_0_HBP(x)
LCD_TIMING_0_HFP(x)
LCD_TIMING_0_VBP(x)
LCD_TIMING_0_VFP(x)
```

que pueden observarse en la fig. p y que respectivamente representan los siguientes parámetros de operación del display:

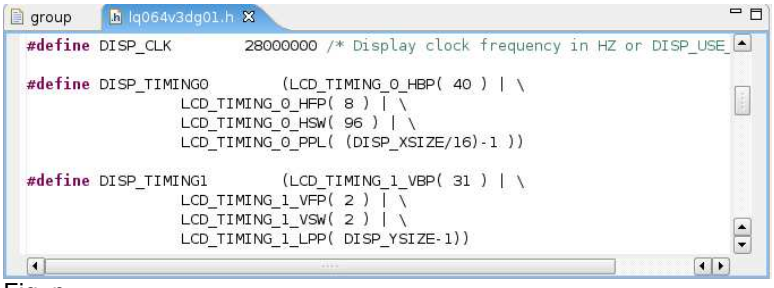


Fig. p

- Horizontal Back Porch
- Vertical Back Porch
- Horizontal Front Porch
- Vertical Front Porch

Los cuales están gráficamente esquematizados en la figura q.

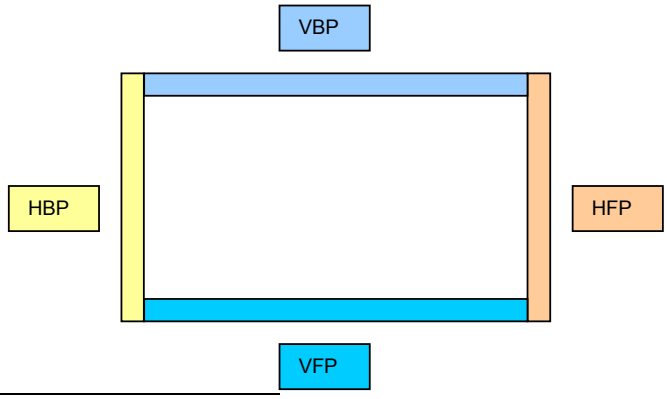


Fig. q

Aquí puede deducirse que, por ejemplo, si quisiéramos *subir* la imágen debemos aumentar VFP y disminuir VBP; y el mismo criterio se aplica a los parámetros HBP/HFP para desplazar la imágen horizontalmente. Se recomienda consultar el datasheet del display para conocer los rangos de operación.

[11] IMPORTANTE: al realizar las pruebas hemos advertido que las modificaciones hechas sobre el archivo `.h` copiado en localmente en el proyecto no tenía ningún efecto en la compilación. Es decir, que al parecer, en el caso de los headers, son tenidos en cuenta solo a nivel global por el entorno de desarrollo, no así los archivos fuente `.c`. De tal modo, para que los cambios citados tuvieran efecto fue necesario realizarlos en modo global, es decir directamente en el archivo original.

Ejemplo: modificar parámetros del driver del touchscreen

Si esto fuera necesario, debemos buscar el driver del dispositivo ADS7846, representado por el archivo `ads7846.c`. Lo encontraremos en el directorio global del kernel:
`drivers/input/touchscreen/`
Debemos reproducir la ruta correcta en el directorio local del proyecto y luego tendremos libre acceso al archivo sin riesgos de dañar el código fuente original.

Cómo incluir archivos y directorios en el rootfs

Una vez que el rootfs es configurado con las aplicaciones pre-compiladas y que el tipo de imagen (JFFS2, CRAMFS, ROM, INITRD) está definido pueden agregarse archivo y/o carpetas de dos maneras distintas:

1) Modificando el directorio rootfs
2) Usando el script `add_files.sh`

Modificación del directorio rootfs

El root file system es conformado en el directorio `build/rootfs/`. De modo que los archivos y carpetas del usuario pueden ser copiados y creados en esa ruta y estarán disponibles luego de la compilación e instalación del file system en el módulo.

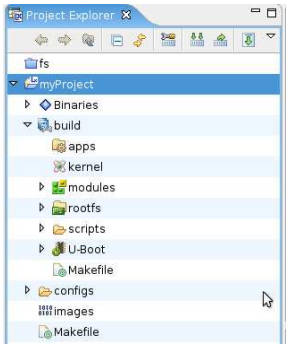
Los cambios en `build/rootfs/` requieren la ejecución de los comandos build e install para estar disponibles en el destino.
Pero este método tiene una desventaja: si se ejecuta un “make clean” o un “make rebuild” se borra completamente el rootfs y se regenera la carpeta `build/rootfs/`, entonces cualquier elemento que hubiera sido depositado allí pro el usuario se perdería.

El script add_files.sh

El script `add_files.sh` está en el directorio `configs/` dentro del directorio del proyecto e inicialmente se encuentra vacío, y puede ser completado con comandos para crear directorios y copiar archivos en el directorio `build/rootfs/ .`
El script `add_files.sh` es llamado durante el proceso de compilación, y construye el rootfs en tiempo de compilación.
Este script puede usar toda las capacidades que tenga el shell del host donde esté instalado DEL: crear directorios, cambiar permisos, copiar archivos, usar condicionales y loops, etc. También hereda variables de entorno y tiene definida una, ROOTFS_DIR, que apunta al rootfs, para facilitar el uso. A continuación, un ejemplo de un script add_files.sh básico :

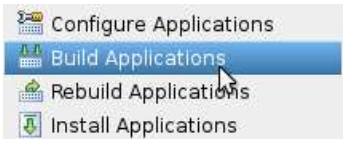
```
ROOTFS_DIR="${DEL_PROJ_DIR}/build/rootfs"

## Example: create a custom directory in rootfs etc dir.
mkdir -p "${ROOTFS_DIR}/etc/myfolder"
mkdir -p "${ROOTFS_DIR}/etc/myimgs"

## Example: copy files to a directory
cp ~/*.jpg "${ROOTFS_DIR}/etc/myimgs"
```

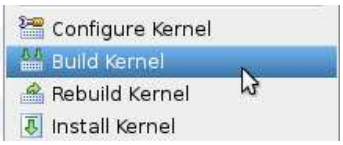
Comandos Build e Install



Hasta acá podría decirse que tenemos el terreno listo para comenzar a trabajar con nuestro sistema Linux embebido.
En lo que respecta a la instalación del entorno de desarrollo y a la preparación de un proyecto completo (bootloader, kernel, rootfs, aplicaciones) está todo preparado para comenzar con la compilación.
Asumimos que del otro lado del entorno de desarrollo nos espera un target que solo tiene cargado el U-Boot (y afortunadamente lo tiene).
Volviendo al explorador de proyectos, recordemos que en nuestros ejemplos estamos trabajando sobre tres[12] grandes componentes:
- El kernel, el File System (rootfs) y las aplicaciones. Por lo tanto, debemos **compilar** e **instalar** cada uno de esos componentes generando sus correspondientes imagenes.

Gráficamente, haciendo un click derecho sobre el componente elegido, por ejemplo, la carpeta de aplicaciones `apps`[13]. se va a desplegar una lista de acciones posible, lo mismo para el resto de los componentes (rootfs y kernel). Ejecutamos entonces la orden Build Applications y se habrán compilado y creado un archivos ejecutable para cada aplicación. A continuación ejecutamos la orden Install Applications, y los ejecutables serán copiados en el rootfs (`/usr/bin/`) para que estén disponibles en el target.
Hacemos lo propio con el rootfs, “Build Rootfs”, para copilarlo, y luego “Install Rootfs”, que en este caso, copiará la imagen del file system al directorio del servicio TFTP (`/tftpboot`) para que pueda ser accedido desde el target.[14]
Por último, lo mismo para el Kernel, con “Build Kernel” se creará la imagen, y con “Install Kernel” se copiará al directorio tftfboot para quedar accesible al target.





En este punto tenemos las imagenes binarias del kernel y del file system (conteniendo las aplicaciones) compiladas y disponibles en el directorio TFTP. Todos estaría listo para ser transferido al módulo.

Carga de las imagenes de linux en módulos Digi

Ya vimos cómo mediante las herramientas de desarrollo se crean las imagenes personalizadas de los diferentes grandes bloques que componen el sistema operativo:

- El bootloader (*U-boot*)
- El kernel (*linux*)
- El sistema de archivos (*rootfs*)
- Aplicaciones (transferidas al rootfs)

El *rootfs* será el contenedor de nuestras aplicaciones de usuario que serán incorporadas en forma definitiva al sistema de archivos cuando llegue el momento de la producción, hasta tanto eso no

[12] Si bien incluimos a U-Boot en el proyecto, en este ejemplo no le aplicamos cambios.
[13] Recordar que dentro de esta carpeta están todas las aplicaciones de ejemplo que seleccionamos durante las configuraciones.
[14] Antes problemas de acceso, intentar desde una consola:
`sudo chown -R USER:USERS_GROUP /exports -v`

suceda, no tendría sentido crear la imagen definitia del rootfs, siendo que es preferible enviar las aplicaciones vía FTP y ejecutarlas directamente desde la imagen cargada en RAM.

El entorno de desarrollo, luego de compilar, genera los archivos de imagen y les asigna los siguientes nombres por defecto que luego serán invocados por U-boot:

U-boot: `u-boot-cc9cjsnand.bin`
Kernel: `uImage-ccw9cjsnand`
Rootfs: `rootfs-ccw9cjsnand-128.jffs2`

Vimos que luego del proceso de compilación dichos archivos también pueden copiarse automáticamente[15] (comando "Install") en el directorio del servidor TFTP[16] que se esté ejecutando en el host de desarrollo. Este directorio por defecto se llama `tftpboot`.

Los módulos Digi vienen de fábrica con una imagen U-boot pre-instalada, lo cual facilita la tarea de actualización del firmware[17]. Si bien, como decíamos, el U-boot puede modificarse para adaptarse a necesidades específicas, en la gran mayoría de los casos, esto no sería necesario.

Para trabajar sobre U-boot, necesitamos una conexión serial entre el Port A y una terminal serial configurada en 38400,8,N,1. Establecida la conexión, si energizamos el módulo, se desplegará el diálogo de arranque en la terminal y luego de pocos segundos se encontrará lista para recibir comandos. Si tuvieramos instalado algún sistema operativo en el módulo, debemos presionar alguna tecla antes de que comience la carga de aquel y así poder interactuar con U-boot.

Si el módulo no tiene definida la tabla de particiones debemos crearla una única vez ejecutando `flpart` , luego `"r"` (reset) a continuación `"l"` (linux) y después `"q"` para salir y finalmente `Enter` para salvar los cambios. Hecho esto, se crearán las particiones para el sistema Linux conforme a la siguiente tabla, en la cual puede apreciarse el espacio físico que ocupa cada imagen en memoria:

Partition number	Name	Flash start address	Flash end address	Length	Description
0	U-Boot	0x00000000	0x000c0000	768 KB	Imagen del boot loader U-Boot
1	NVRAM	0x000c0000	0x00100000	256 KB	Guarda parámetros de configuración como MAC address, número de serie del módulo, variables de entorno de U-Boot
2	FPGA	0x00100000	0x00200000	1 MB	Firmware FPGA
3	Kernel	0x00200000	0x00500000	3 MB	Linux kernel
4	RootFS-JFFS2	0x00500000	0x01500000	16 MB	Linux root file system
5	User-JFFS2	0x01500000	end of flash	rest	Espacio libre para usar ídemmente

U-boot es configurable en varios aspectos gracias a una serie de variables de entorno[18] que pueden ser manipuladas con los siguientes comandos:

```
printenv <var_name>
printenv_dynamic <var_name>
setenv <var_name> <var_value>
saveenv
envreset
```

[15] Eventualmente los archivos de imagen se pueden copiar manualmente al directorio TFTP.
[16] Si se optó por la instalación completa de Kubuntu desde el "Digi Embedded Linux DVD", todo aspecto relacionado con el servidor TFTP será configurado durante la instalación y no habrá que preocuparse por ello. De lo contrario, se debe instalar y configurar el servidor TFTP manualmente.
[17] Si así no fuera, deberíamos utilizar el JTAG para grabar al menos una primer imagen del U-boot y comenzar a trabajar luego a partir de este.
[18] Para conocer los detalles de U-boot, consultar *U-boot_reference_manual.pdf*

Por ejemplo, la variable de entorno `bootcmd` almacena el script de booteo, en este caso, nosotros necesitamos bootear linux desde la flash del módulo, entonces ejecutamos:

```
setenv bootcmd dboot linux flash
saveenv
```

Por otra parte, U-boot utiliza las variables de entorno para configurar su acceso a la red IP. Estas variables son básicamente las siguientes:

`serverip` :la dirección IP del host donde se ejecute el servidor TFTP <host_ip>
`ipaddr` :la dirección IP del módulo <module_ip>

Entonces, como debemos asegurarnos que tanto el módulo como el host sean mutuamente accesibles mediante Ethernet debemos establecer:

```
setenv serverip <host_ip>
setenv ipaddr <module_ip>
```

No está de más verificar también variables tales como `netmask, dhcp,` etc. Por ejemplo, `setenv dhcp off` para evitar que el módulo adquiera una IP por DHCP.

Una vez configurada la red, pordemos comprobar que el módulo tenga acceso al host:

```
ping $(serverip)[19]
```

Finalmente, guardar los cambios con `saveenv`

El último requerimiento antes de descargar el firmware en el módulo, es que el servidor TFTP que administra el directorio que contiene las imagenes se esté ejecutando en el host de desarrollo[20] y que los archivos de imágen que querramos cargar estén allí ubicados.

Habiendo llegado hasta acá, si nuestro sistema quedó debidamente configurado y si mantuvimos los parámetros de forma estándar, podemos despreocuparnos de todo aspecto de configuración.

Cabe destacar que los nombres por defecto de los archivos de imagen ya están previamente asignados a sus respectivas variables de entorno: `uimg`, `kimg`, `rimg` y gracias a ello no tenemos la obligación de recordar sus desagradables nombres al momento de realizar el `update`, claro está, siempre que no cambiemos los nombres por defecto de los archivos de imagen.

Finalmente el comandos para descargar la imagen deseada vía TFTP, verificarla y luego la grabarla en la partición correspondiente es el comando `update <part> <source>:`

```
update uboot tftp
update linux tftp
update rootfs tftp
```

Con esto, habremos grabado la flash del módulo con la imagen recientemente compilada y prodremos bootear para testear nuestro trabajo reiniciando el módulo, o ejecutando el comando `boot`, o el `reset`.

Es importante destacar que durante el desarrollo no es necesario grabar la flash del módulo, siendo que es posible bootear desde el directorio TFTP o desde USB con los siguientes comandos:

[19] Tener en cuenta que U-Boot no responde los pings.

```
dboot linux tftp
dboot linux usb
```

Por último, desde la consola de U-Boot, tipeando `help` obtendrá la lista de todos los comandos disponibles, y con `help <command_name>`, la ayuda para un comando en particular.

Una cosa más: si deseamos redirigir la consola hacia el display necesitamos definir la variable de entorno `console` como `ttyS1,9600` , para esto:

```
setenv_dynamic console ttyS1, 9600
saveenv
```

Probando[21]

Una vez cargado el kernel y el rootfs en el módulo estamos en condiciones de hacer arrancar el módulo.

- Si le conectamos un teclado USB al módulo CC9C podremos operar sobre la terminal, cuya salida ya dirigimos al display LCD.

- Otra prueba que podemos hacer es conectarnos mediante una terminal, pero vía telnet. (`telnet IPMODULO`)

- También podemos conectar vía FTP con usuario anónimo y transferir aplicaciones para probar al directorio `/tmp`

- Si nos conectamos mediante un explorador web, tendremos acceso al servidor web del módulo donde vermos una simple página de bienvenida.

- Desde la teminal también podemos ejecutar alguna de las aplicaciones de muestra (y por supuesto, cualquiera de los comandos de Linux.), por ejemplo:

```
./usr/bin/hello_world
./usr/bin/qtopia/ftp -qws
./usr/bin/qtopia/application -qws
./usr/bin/qtopia/spreadsheet -qws
./usr/bin/pthread_test
./usr/bin/gpio_test
```

En caso de la demo "`gpio_test`", que fue diseñada para demostrar el uso de módulos cargables, para ejecutarla debemos antes que nada cargar el módulo llamado "gpio.ko" con `modprobe gpio` para el caso en que el módulo se haya integrado al rootfs al momento de la Compilación /

[21] Troubleshotings

- Al compilar el `rootfs` e intentar luego el comando `Install`, se obtenían errores de acceso al directorio `/exports`
  Se solucionó ejecutando desde una terminal :
  sudo chown –R USER:USER_GROUP /exports -v
- Al intentar bootear directamente vía TFTP con `dboot linux tftp` la carga de linux se interrumpe por no encontrar el `rootfs`, lo cual, obviamente no nos sucedió al bootear y se puede solucionar configurando las variables de entorno.
- La instalación mínima, es decir, la instalación del entorno de desarrollo únicamente, sobre una distibución linux preexistente, es altamente susceptible a conflictos entre versiones y por ello no se recomienda
- En particular, la versión que estuvimos probando DEL4.0 es algo antigua (año 2007), y no nos permitió un uso adecuado desde una máquina virtual, pero las últimas versiones, ya permiten un uso fluído.

Instalación. O con `insmod gpio.ko` si el módulo no está integrado al rootfs y se transfiere luego por algún otro medio.

Conclusiones

El sistema Linux embebido de Digi constituye una alternativa muy atractiva para reunir las ventajas del código abierto junto con la confiabilidad de un sistema de estado sólido, sin partes mecánicas, sin cooler, de dimensiones reducidas, de bajo consumo, y que además soporta rangos industriales de temperatura ambiente. Nos parece una opción muy viable para el reemplazo de servidores que corran en PC y que por razones de confiabilidad, espacio, costo, consumo u hostilidad del ambiente, se los quiera reemplazar por su equivalente embebido. Por otra parte, es también adecuado para aplicaciones altamente dedicadas al networking que no necesariamente requieran interfaz gráfica.