Implementación de sistemas operativos sobre Orange PI zero

Maestría en Sistemas Embebidos, segunda cohorte

Año 2018

Autor

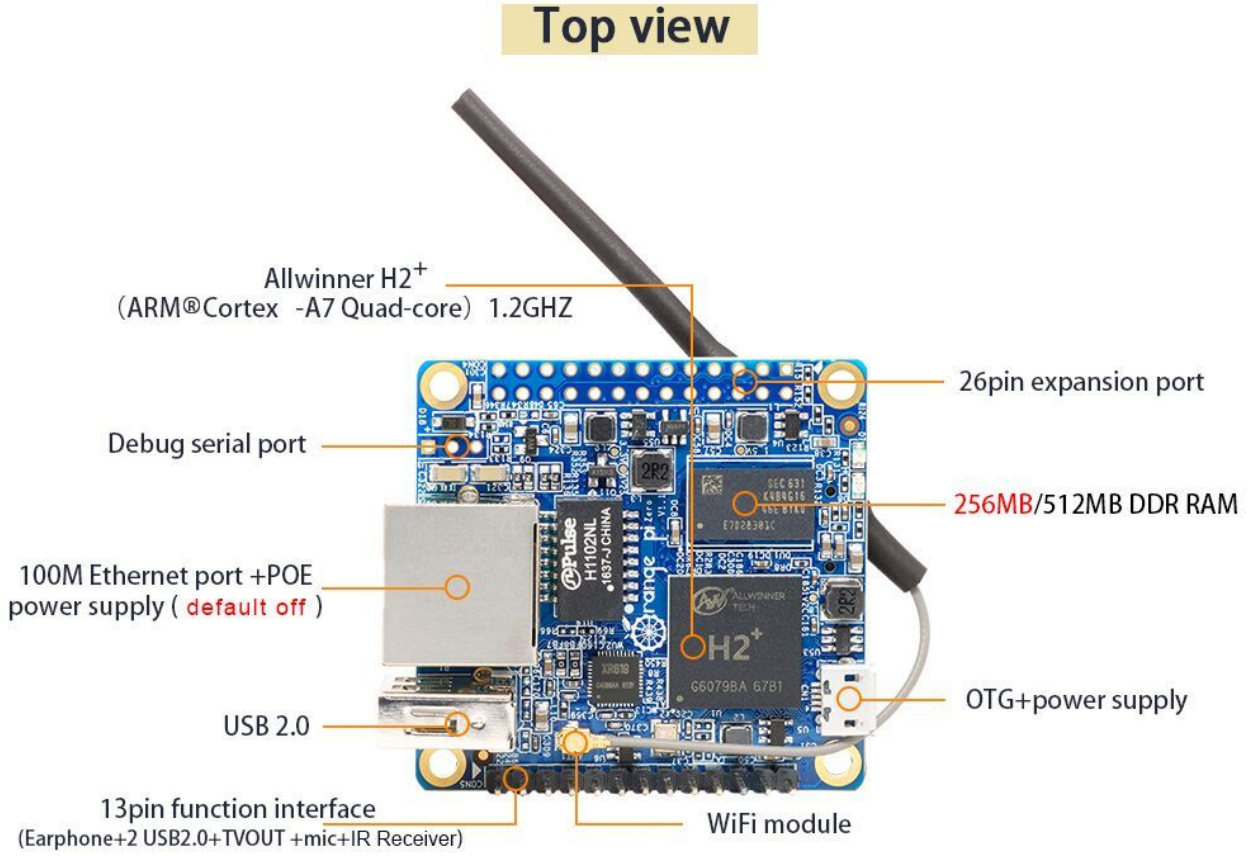
Esp. Ing. Gonzalo Sanchez

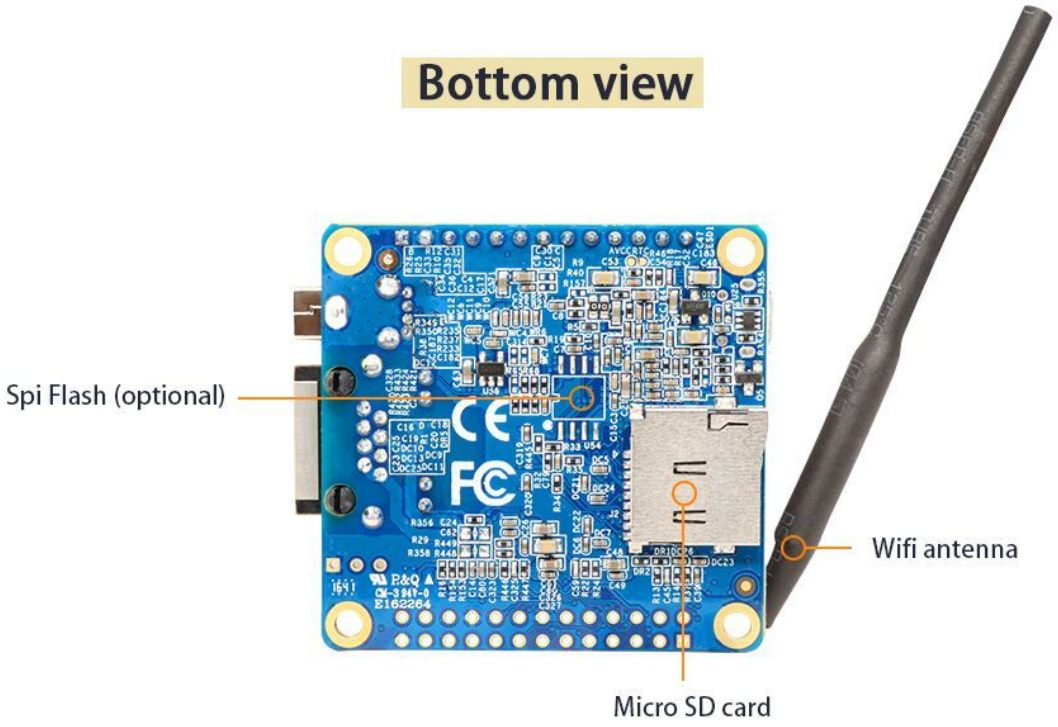
This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Tabla de contenido

Introducción: Orange PI Zero

La orange pi es una Single Board Computer (SCB) open source. Puede correr android 4.4, Ubuntu, Debian, etc. Esta placa utiliza el SoC Allwinner H2, y viene una versión de 256MB y 512MB DDR3 SDRAM(256MB es la versión Standard)





Las especificaciones de hardware pueden obtenerse de su pagina web http://www.orangepi.org/orangepizero/

Descargando el código fuente del Kernel de linux

Para el caso de la familia de SBC a la que pertenece la placa que vamos a utilizar (Orange Pi), no existe soporte mainline del kernel de linux, por lo que dependemos del kernel que nos provee el fabricante. En los últimos años la comunidad de desarrolladores que comenzaron a utilizar esta plataforma fabricada por Sunxi ha crecido significativamente, dado el bajo costo del hardware, por lo que existe mucho mejor soporte al dia de hoy que cuando se lanzó la primer SBC de esta empresa.

Setup

Lo primero que debemos hacer es preparar un ambiente de trabajo para la posterior compilación del kernel. Este punto es sencillo, solo debemos crear un directorio en el cual trabajar. Se recomienda crear un directorio raíz donde se iran poniendo carpetas para cada herramienta a utilizar (en este caso, cada punto de la guia).
Creamos un directorio de trabajo y lo asociamos a una variable en el bash para poder trabajar más cómodamente:

```
cd $HOME
mkdir orange-pi
cd orange-pi
export ORANGEPI_ROOT=$(pwd)
```

Clonando el repositorio

El paso siguiente es conseguir los fuentes del kernel que utilizaremos. Como se mencionó antes, se utilizara el provisto por el fabricante. Para este paso es necesario que el paquete git esté instalado (**`sudo apt-get install git`**).

```
cd $ORANGEPI_ROOT
git clone --depth 1 -b sunxi-next --single-branch
https://github.com/linux-sunxi/linux-sunxi
```

Cabe destacar que el comando git debe estar en una sola línea. Esto creará una carpeta llamada *linux-sunxi* dentro de nuestro directorio raiz de trabajo, alli estaran todos los archivos fuente para la compilación de nuestro kernel. En aras de simplificar los path, haremos un nuevo export:

```
cd $ORANGEPI_ROOT
cd linux-sunxi
export KERNEL_SRC=$(pwd)
```

Para obtener nuestro kernel todavía es necesario algunos pasos extra, pero es necesaria la herramienta de compilación para generar código que la Orange Pi pueda interpretar, así que se retomara la compilación de kernel en un apartado posterior.

Generando toolchain necesario

En este apartado se abarca como generar un toolchain para la compilación de código que pueda ser interpretado por la SBC que estamos utilizando. Con este juego de herramientas podremos compilar el kernel de linux y luego otras herramientas para que nuestro sistema funcione correctamente. Antes de comenzar con la configuración, es necesario saber a que se refieren los términos *BUILD*, *HOST* y *TARGET.*
Cuando se refiere a la arquitectura de BUILD, se hace referencia al hardware que se utiliza para compilar el toolchain que se quiere generar. Normalmente se compila un toolchain en una PC poderosa para que la compilación del toolchain no requiera un tiempo excesivo.
HOST es el hardware que correrá este toolchain generado. En nuestro caso BUILD = HOST, dado que se compilara el toolchain y también se correrá en la misma PC. En algunos casos esto no es así, y se utiliza un hardware potente para generar el toolchain, pero este se corre en un hardware más limitado.
TARGET se refiere al hardware que correrá el código que sea compilado por el toolchain generado. El caso de esta guía es *BUILD = HOST != TARGET*, donde TARGET será una arquitectura ARM.

Instalación los paquetes necesarios

Es probable que sea necesario instalar algunas dependencias antes de comenzar. Se recomienda instalar los siguientes paquetes en la PC que se está utilizando como BUILD.

```
sudo apt-get install autoconf automake libtool libexpat1-dev libncurses5-dev
bison flex patch curl cvs texinfo git bc build-essential subversion gawk
python-dev gperf unzip pkg-config wget
```

Si bien parecen muchos paquetes, la realidad es que la mayoria ya estan instalados en la mayoria de las distribuciones linux, por lo que solamente se instalarán las que no estén presentes en el sistema.

Obteniendo Crosstool-ng

Una vez que hemos resuelto las dependencias necesarias, pasamos a obtener los sources de la herramienta *crosstool-ng* la cual será la que genere el toolchain para compilar nuestro kernel y las sucesivas herramientas necesarias.

```
cd $ORANGEPI_ROOT
mkdir toolchain
cd toolchain
export TOOLCHAIN_ROOT=$(pwd)
git clone https://github.com/crosstool-ng/crosstool-ng.git
```

Esto volcara todo los archivos fuentes de la herramienta crosstool-ng en una carpeta homónima, por lo que para claridad de los paths, volvemos a hacer un export:

```
cd $TOOLCHAIN_ROOT/crosstool-ng
export CROSSTOOL_SRC=$(pwd)
```

Se recomienda utilizar la rama master que por defecto está activa al clonar el repositorio, dado que tomara los fuentes más actualizados y puede ser fácilmente adaptado a la necesidad cuando el hardware sufre modificaciones al pasar el tiempo. Al momento de generar esta guia esta rama corresponde a la versión *crosstool-ng-1.23.0.*

Instalando Crosstool-ng

Para que todo nuestro ambiente de desarrollo quede encapsulado, es recomendable configurar crosstools-ng para que se instale de manera local, por lo que lo configuramos mediante los siguientes comandos:

```
cd $CROSSTOOL_SRC
./bootstrap
./configure --enable-local
make
make install
```

Estos pasos no deberían generar ninguna clase de inconvenientes, y nuestra herramienta generadora de toolchains está ahora lista para ser configurada.

Configuración del juego de herramientas a producir

Una sola instalación de crosstool-ng permite producir tantos juegos de herramientas como se quiera, para diferentes arquitecturas, con diferentes bibliotecas C y diferentes versiones de varios componentes.
Crosstool-ng contiene un juego de archivos de configuración listos para ser utilizados, de configuraciones típicas a las cuales se les da el nombre de *samples*. Estas pueden ser listadas utilizando el siguiente comando:

```
$CROSSTOOL_SRC/ct-ng listsamples
```

En esta guía se utiliza la muestra *arm-unknown-linux-gnueabi*. La misma puede ser cargada ejecutando el comando:

```
$CROSSTOOL_SRC/ct-ng arm-unknown-linux-gnueabi
```

Antes de configurar la herramienta, hace falta crear un directorio donde serán descargados los fuentes para la creación de nuestro toolchain:

```
mkdir $TOOLCHAIN_ROOT/tar_src
```

Ahora, es necesario que se hagan algunos retoques en la configuración, por lo que ejecutamos el siguiente comando, el cual nos dará acceso al menú de configuración:

```
$CROSSTOOL_SRC/ct-ng menuconfig
```

A continuación se mencionan algunas configuraciones que deben ser configuradas, según el orden de aparición en el menú que menuconfig despliega. Recuerde que para seleccionar una configuración nueva se debe presionar la tecla ***y***, y para que sea deseleccionada debe presionarse la tecla ***n***.

1. Ingrese en Path and misc options:
    a. Seleccione *Debug crosstool-NG* y dentro de esa opción seleccione también *Save intermediate steps*. Esto nos va a permitir retomar cualquiera de los pasos de compilación en caso de una falla.
    b. Cambie *Local tarballs directory* a **${TOOLCHAIN_ROOT}/tar_src**. Este directorio es donde se van a descargar los distintos componentes a compilar.
    c. Cambie *Prefix directory* a **${TOOLCHAIN_ROOT}/xtools/${CT_TARGET}**. Este es el directorio en donde el juego de herramientas se va a instalar.
    d. Asigne *Number of parallel jobs* la cantidad de núcleos que posee la estación de trabajo. Es posible consultar dicha cantidad ejecutando en una terminal:
    `cat /proc/cpuinfo | grep processor | wc -l`

e. La opción *Maximum log level to see* debe ser seteada a EXTRA, para que la salida contenga información útil, pero que no sea una lista ilegible de comandos ejecutados.

2. Ingrese en Target Options:
   a. En *Toolchain ID string* escriba **MSE**. Esta es la cadena identificatoria del toolchain generado.
   b. En *Tuple's vendor string* escribimos **FIUBA**.
   c. Escriba en *Tuple's alias* la cadena **arm-linux**. De esta forma, se podrá utilizar el toolchain como **arm-linux-gcc** en lugar de arm-FIUBA-linux-gnueabihf-gcc, que es mucho más largo de escribir. El alias solamente genera un link simbólico, por lo que puede poner el nombre que usted desee, inclusive su apodo.
3. Ingrese en C-library:
   a. Asegúrese que la opción *C library* muestre **glibc**. Para la versión *crosstool-ng-1.23.0* se deja la versión por defecto de *glibc*, que es 2.27.
4. Ingrese en C compiler:
   a. Asegúrese que la opción Version of gcc muestre **7.3.0**. Si se utiliza una versión más nueva del compilador, se generarán problemas en apartados posteriores.
5. Ingrese a Debug facilities:
   a. Asegúrese que *gdb* está habilitado. Dentro de esta opción:
      i. Asegúrese que las opciones *Cross-gdb* y *Build a static gdbserver* están habilitadas.
   b. Elimine las opciones *duma* y *ltrace.*

Produciendo el juego de herramientas

Para comenzar la compilación, el único paso faltante es ejecutar el script correspondiente:

```
cd $CROSSTOOL_SRC
./ct-ng build
```

Problemas conocidos

Es frecuente que Crosstool-ng falle debido a que no puede encontrar algunos archivos fuente en Internet, cuando dicho archivo fue movido o reemplazado por una versión más reciente. Es esta la razón por la que se sugiere utilizar siempre la última versión estable del repositorio, y adaptar las opciones para generar un toolchain funcional.

Probando el juego de herramientas

Para probar el juego de herramientas generado, se debe agregar a la variable $PATH la dirección del mismo, de la siguiente manera:

```
export PATH=$PATH:$CROSSTOOL_ROOT/xtools/arm-FIUBA-linux-gnueabihf/bin
```

Esto hará un *append* en la variable de entorno mientras esté abierto el bash donde se ejecuta, y nuestro cross-compiler sera asi visible. Para verificar esto podemos hacer

```
echo $PATH
```

Asi podemos asegurar que el path correspondiente se agregó al final de la cadena. Entonces podemos ver que nuestro compilador generado funciona mediante:

```
arm-linux-gcc --version
```

La salida de consola debería mostrar algo similar a esto:

```
arm-linux-gcc (crosstool-NG 1.23.0.418-d590-dirty - MSE 2da Cohorte) 7.3.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Lo mostrado debería coincidir con nuestras configuraciones. Si se compila cualquier archivo .c siguiendo las reglas de los argumentos como para gcc, y ejecutamos el comando **file** sobre el archivo resultante, la salida de la consola debería mostrar que esta compilado para arquitectura ARM. En este caso nuestro archivo ejecutable de salida se llama *hello*.

```
file hello
```

La salida de consola debería mostrar algo similar a esto:

```
hello: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically
linked, interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 2.6.32, not
stripped
```

Se puede ver que el formato del ejecutable de salida es el deseado.

Compilando el Kernel

Ahora que tenemos disponible nuestro toolchain listo para compilar, el próximo paso es compilar nuestro kernel de linux. Como se mencionó antes, la Orange Pi Zero no es soportada por el kernel mainline de linux, por lo tanto es necesario descargar un kernel que tiene los patches necesarios para que nuestra SBC funcione. Habiendo descargado el código fuente en un apartado anterior, solo nos queda comenzar a configurar para compilar.

Configuración del kernel

Como ya todas las variables de PATH están definidas, solamente nos resta exportar algunas variables utilizadas por el makefile del kernel:

```
cd $KERNEL_SRC
export ARCH=arm
export CROSS_COMPILE=arm-linux-
```

Aquí estamos indicando la arquitectura para la cual debe ser compilado nuestro kernel, y cual es el compilador a utilizar. Cabe destacar que si la variable PATH no es actualizada como en los pasos anteriores, se generará un error dado que el compilador no puede ser encontrado.
Ahora generamos el archivo .config segun la configuracion por defecto para la mayoria de las SCB de la empresa sunxi.

```
make sunxi_defconfig
```

Esto generará el archivo .config necesario para poder compilar nuestro kernel. Con las configuraciones por defecto es posible trabajar sin inconvenientes, pero se recomienda que se explore las distintas opciones que el kernel tiene para ser configuradas mediante ***make menuconfig.***

Una vez configurado el kernel, lo único que resta es compilar:

```
make -j4
```

Esta compilación es larga, y dependiendo la PC puede llevar más de media hora. Una vez que la compilación ha finalizado, tendremos dos archivos importantes que nos harán falta en próximas secciones. El primero es la imagen de kernel ***zImage*** la cual se puede encontrar en la ruta ***$KERNEL_SRC/arch/arm/boot***.
Otro archivo importante es ***sun8i-h2-plus-orangepi-zero.dtb***, que es el device tree generado para nuestra SCB, el cual se encuentra en la ruta ***$KERNEL_SRC/arch/arm/boot/dts.***
Es necesaria una copia de ambos archivos para una sección posterior.

Compilando U-boot

Lo que se conoce como bootloader es la primer porción de software que se carga al arranque de todo ordenador; por lo tanto nuestra SCB no es la excepción y necesitamos un bootloader para iniciar el sistema. Se utiliza en este caso el famoso U-boot, el cual mantiene en su mainline a la Orange Pi Zero, simplificando bastante la operatoria.

Obtener código fuente

Para obtener el código fuente de U-boot, solo hace falta descargar el repositorio correspondiente:

```
cd $ORANGEPI_ROOT
git clone git://git.denx.de/u-boot.git
cd u-boot
export U_BOOT_SRC=$(pwd)
```

Ya tenemos nuestro código fuente a partir del cual compilar U-boot, y una variable para apuntar al path correspondiente. Se recomienda utilizar una versión actual de U-boot, que en este caso sera la version estable de mayo del 2018.

```
cd $U_BOOT_SRC
git checkout v2018.05
```

Configuración

Al disponer ya del código fuente, y estar soportada nuestra SCB en el mainline, la configuración es muy sencilla, es necesario tener exportadas las mismas variables que para la compilación del kernel. Repetiremos aquí para claridad:

```
cd $U_BOOT_SRC
export ARCH=arm
export CROSS_COMPILE=arm-linux-
```

Al ser soportado en mainline, tenemos una configuración por defecto para nuestra SCB. Creamos el archivo .config y luego compilamos:

```
make orangepi_zero_defconfig
make -j4
```

Si la compilación falla, es posible que sea necesaria la instalación de alguna dependencia. Esto es detectado fácilmente leyendo los mensajes de error que surgen en la compilación fallida. Un ejemplo es la ausencia del paquete ***swig*** en algunos sistemas.
Es mandatorio que el sistema tenga configurado python2.7 por defecto. En el caso de tener python3.0, es necesario crear un ambiente virtual mediante ***virtualenv*** y utilizar python2.7.

Preparando la memoria SD

Habiendo compilado U-boot, disponemos de un archivo llamado ***u-boot-sunxi-with-spl.bin*** el cual se encuentra dentro de ***$U_BOOT_SRC***. La preparación de la memoria SD es muy sencilla, solamente debemos insertarla en la PC que utilizamos para compilar U-boot, y ejecutar los siguientes comandos:

```
cd $U_BOOT_SRC
sudo dd if=u-boot-sunxi-with-spl.bin of=/dev/sdb bs=1024 seek=8 conv=notrunc
```

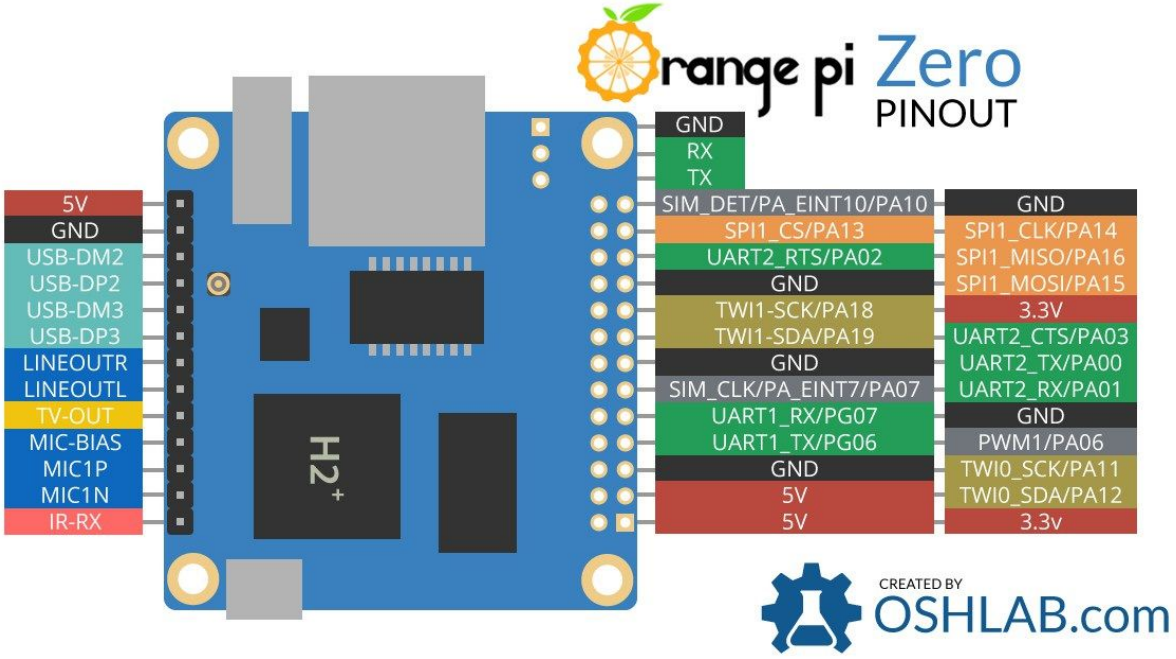
Nótese que la letra que identifica la memoria SD insertada está en otro color. Esta letra debe ser cambiada según como se enumera en el sistema utilizado. Para saber qué letra se le ha asignado basta con hacer un ***ls /dev | grep sd*** antes y despues de insertar la memoria.

Es necesario que la memoria no esté montada al momento de ejecutar la escritura. Si se ha montado automáticamente, utilice ***umount*** para desmontarla y proceda con el comando de escritura de datos.

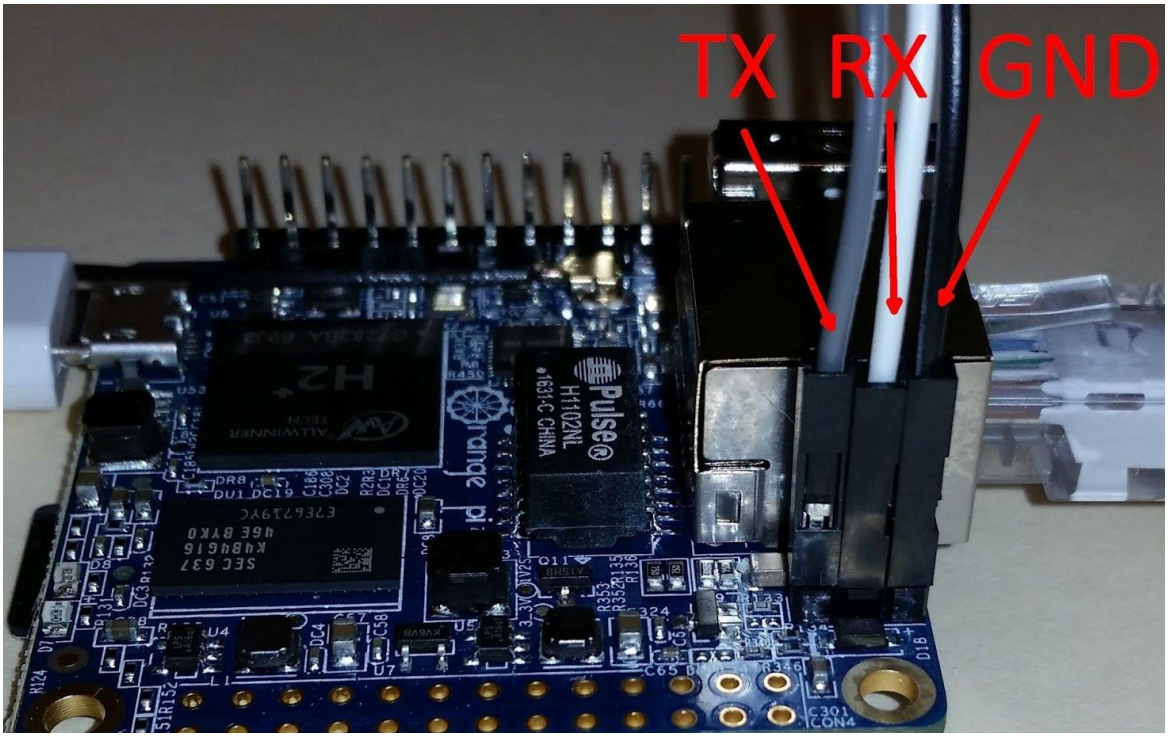
Al insertar la memoria SD en la SCB, se está en condiciones de comenzar a interactuar con la misma por medio de una conexión serie.

Preparando la SBC

Lo primero que debemos resolver en este momento es como conectar nuestra SCB para obtener una salida de consola con la que podamos interactuar. La Orange Pi Zero tiene pines dedicados para el puerto serie que se utiliza para conectarse e interactuar con la consola.
La siguiente imagen nos da una vision esquematica de nuestra SCB y a que corresponde cada pin en ella.



Con la finalidad de aclarar cuales pines han de ser conectados, se muestra también una foto de una Orange Pi Zero conectada.



Habiendo conectado la SCB a un adaptador USB-serie a la PC que utilizaremos para enviar y recibir los comandos por consola, se debe asegurar de tener instalado una aplicación de terminal, como puede ser ***gtkterm***.
Es posible que el dispositivo que se genera en el directorio ***/dev*** sea accesible sólo por usuarios del grupo ***dialout*** o por el usuario ***root***. Esto puede cambiarse agregando al usuario actual al grupo ***dialout***.

```
sudo adduser $USER dialout
```

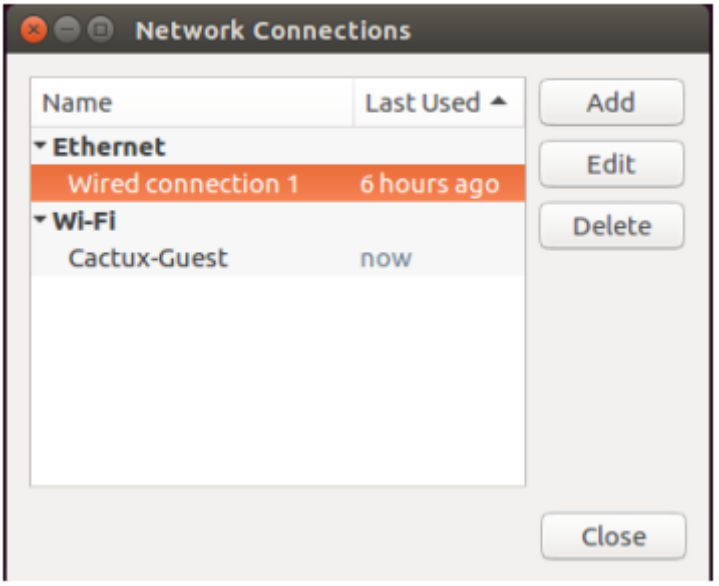
Luego para que los cambios sean reflejados, es necesario hacer un logout y volver a iniciar la sesión.
Cabe destacar que la conexión para la consola de la Orange Pi Zero es de 115200 bps 8N1.

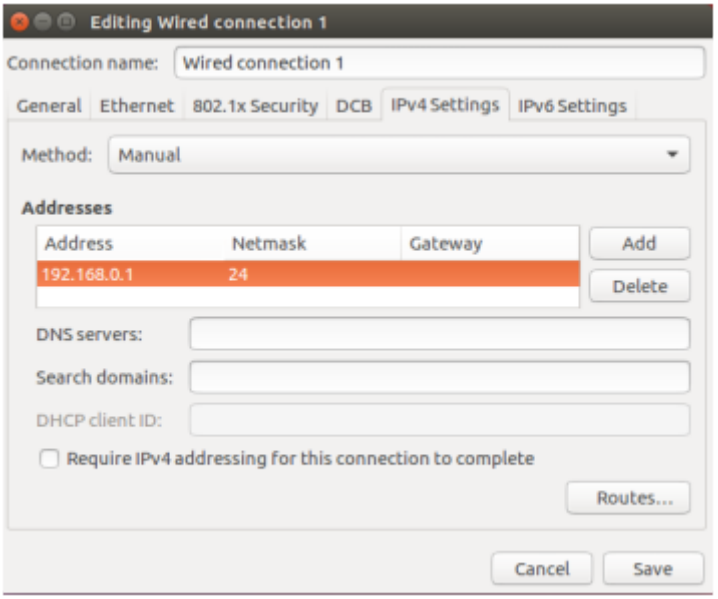
El paso siguiente es instalar un servidor TFTP en la PC de trabajo:

```
sudo apt install tftpd-hpa
```

Una vez instalado, conecte el cable de ethernet desde la SCB hacia su PC de trabajo. Ahora se deben editar las conexiones para hacerla visible.

Editando las conexiones, seleccionaremos nueva red cableada:





En las configuraciones IPv4, seleccione el método manual para hacer que la interfaz utilice una IP estática, como por ejemplo 192.168.0.1. Puede dejar el valor de la máscara de subred en 24.

Es hora de conectar la SCB a la PC mediante el adaptador USB-serie. Se debe configurar la conexión, abrirla y luego energizar la SCB con la memoria SD conectada, donde se cargó previamente el U-boot.
Cuando la SCB inicie, enviará por consola datos del hardware y luego comenzará una cuenta regresiva para hacer un booteo desde la ubicación por defecto. Presionando cualquier tecla se aborta este conteo regresivo.
Es necesario decirle a U-boot la dirección desde donde queremos cargar el kernel y el File Sistem, por lo que se deben ejecutar los siguientes comandos:

```
setenv ipaddr 192.168.0.100
setenv serverip 192.168.0.1
saveenv
```

El comando saveenv guarda la configuración en la memoria SD para futuros booteos.

Lo siguiente es configurar el servidor NFS. Si el paquete ***nfs-kernel-server*** no está instalado en su sistema, deberá instalarlo:

```
sudo apt install nfs-kernel-server
```

Una vez instalado, se debe modificar el archivo ***/etc/exports*** agregando cual es el path que se utilizara para montaje del file system por NFS. Crearemos un directorio solamente a modo de prueba:

```
cd $ORANGEPI_ROOT
mkdir nfsroot_test
```

Modificamos ahora el archivo ***/etc/exports*** de la siguiente forma

```
<path/a/su/directorio/nfs> 192.168.0.100(rw,no_root_squash,no_subtree_check)
```

Donde el path que está coloreado debe ser reemplazado por lo el resultado de este comando, sin los brackets:

```
echo $ORANGEPI_ROOT/nfsroot_test
```

Ejemplo /home/usr/nfsroot_test 192.168.0.100(rw,no_root_squash,no_subtree_check)

Ahora es necesario reiniciar el servidor NFS para que esta ruta sea incluida.

```
sudo /etc/init.d/nfs-kernel-server restart
```

Teniendo el servidor NFS corriendo y configurado como corresponde, podemos ahora hacer que nuestra SCB inicie el kernel. Conectándonos nuevamente por consola a la Orange Pi Zero, evitando que U-boot inicie en el dispositivo por defecto, ingresamos el siguiente comando:

```
setenv bootargs root=/dev/nfs rw ip=192.168.10.100 console=ttyS0,115200
nfsroot=192.168.10.1:<path/a/su/directorio/nfs>
```

Este comando debe estar en una sola linea, aquí se muestra en dos líneas dado el ancho de la hoja. Nuevamente el path al directorio nfs será el mismo que se indico en el archivo ***/etc/exports*** de la PC de trabajo.
Para que este comando quede grabado en la memoria SD de la SCB, se ejecuta nuevamente:

```
saveenv
```

Volviendo a la PC de trabajo, es necesario poner una copia de los archivos ***zImage*** y ***sun8i-h2-plus-orangepi-zero.dtb*** en la carpeta ***/var/lib/tftpboot*** para que puedan ser transferidos a la memoria ram de la SCB.
Cuando tenga una copia de estos archivos que fueran generados cuando se compilo el kernel de linux en el directorio ***/var/lib/tftpboot*** se debe volver nuevamente a la consola de la SCB y en U-boot ejecutar los siguientes comandos:

```
tftp 0x42000000 zImage
tftp 0x43000000 sun8i-h2-plus-orangepi-zero.dtb
bootz 0x42000000 - 0x43000000
```

Esto hará que se copien los archivos mencionados a las posiciones de memoria RAM indicadas, las cuales son específicas de la Orange Pi Zero, y por último se hará un booteo tomando estas direcciones de base para la carga del Kernel y del DTB.

Si todos los pasos anteriores se cumplimentaron, la salida de consola de la SCB deberia iniciar el Kernel de linux y luego de cargar, mostrar un error de kernel panic. Esto se debe a que el directorio ***/nfsroot_test*** que se genero esta vacio, en la sección siguiente se abarca como poblar este directorio.

Compilando BusyBox

Para tener una distribución mínima de linux corriendo en la SCB, el paso siguiente es descargar los fuentes de BusyBox, el cual genera un File System que puede ser copiado a la carpeta ***/nfsroot_test*** generada anteriormente, para poder levantar el sistema de esa manera.
Es necesario descargar los fuentes de busybox, por lo que ejecutamos los siguientes comandos.

```
cd $ORANGEPI_ROOT
git clone git://busybox.net/busybox.git
cd busybox
export BUSYBOX_SRC=$(pwd)
```

Configuración de BusyBox

Una vez en el directorio de busybox, se debe generar el archivo ***.config*** necesario para la compilación. Por lo que se debe ejecutar el siguiente comando (las variables **ARCH** y **CROSS_COMPILE** deben estar definidas, así como el path al binario del croscompilador):

```
make menuconfig
```

Las configuraciones se dejan libres al lector, solamente se recomienda para mayor facilidad que se busque y habilite la opción “settings->Build Options: Build static binary (no shared libs)” y se ponga la ruta absoluta del directorio nfsroot_test que anteriormente se creó en la opcion “settings->Instalation options: Destination path for ‘make install’ ”.
Creado el archivo .config, lo que queda es ejecutar los comandos:

```
make
make install
```

Esto poblara la carpeta **/nfsroot_test** de manera tal que si se repiten los pasos para bootear la SCB, no se verá un error Kernel panic, sino que se visualizará un prompt de login root.

Build-Root

Hasta este apartado, se consiguió mediante distintas herramientas que la SCB en uso pueda cargar un kernel de linux y tener un file system mínimo para poder trabajar. Habiendo adquirido esta experiencia, se procede a utilizar la herramienta build root, la cual automatiza este proceso.

Obteniendo el codigo fuente de BuildRoot

Para obtener el código fuente de build root, debemos ejecutar los siguientes comandos:

```
cd $ORANGEPI_ROOT
git clone git://git.buildroot.net/buildroot
cd buildroot
export BUILDROOT_SRC=$(pwd)
git checkout 2018.05
```

Así, tendremos nuestro código fuente y la variable correspondiente indicando el path. Se utiliza en este caso la versión estable de mayo de 2018.
Existen varios subdirectorios y archivos, de los cuales los más importantes son:

- ***/boot*** contiene los Makefiles y los ítems de configuración relacionados con la compilación de cargadores de inicio comunes (Grub, U-Boot, Barebox, etc.)
- ***/configs*** contiene un juego predefinido de configuraciones, similar al concepto de def
- en el Kernel.
- ***/docs*** contiene la documentación de Buildroot. Puede comenzar por leer buildroot.html que es la principal documentación de Buildroot.
- ***/fs*** contiene el código usado para generar varios de los formatos de imagen del sistema de archivos raíz.
- ***/linux*** contiene el Makefile y los ítems de configuración relacionados con la compilación del Kernel de Linux.
- ***/Makefile*** es el principal Makefile que vamos a utilizar para en Buildroot: todo funciona a traves de Makefiles en Buildroot.
- ***/package*** es un directorio que contiene todos los Makefiles, parches e ítems de configuración para compilar las aplicaciones en el espacio de usuario y bibliotecas de su sistema embebido Linux. Se recomienda explorar varios de sus subdirectorios y vea que es lo que contienen.
- ***/system*** contiene el esqueleto del sistema de archivos raíz y las tablas de dispositivos utilizadas cuando se utiliza un ***/dev*** estático.
- ***/toolchain*** contiene los Makefiles, parches e ítems de configuración para generar el juego de herramientas de compilación cruzada.

Configurar Buildroot

En el caso de la SCB utilizada, nos va a interesar:

- Generar un sistema embebido en Linux para ARM.
- Utilizar un juego de herramientas de compilación cruzada existente, en lugar de que Buildroot lo genere por nosotros.
- Integrar Busybox en nuestro sistema embebido en Linux.
- Integrar el sistema de archivos destino en un empaquetado (tarball).

Antes de entrar en la herramienta de configuración de BuildRoot, debemos hacer los exports necesarios para definir la arquitectura y el cross compilador a ser utilizado:

```
cd $BUILDROOT_SRC
export ARCH=arm
export CROSS_COMPILE=arm-linux-
```

Ahora debemos generar el archivo .config de base para la Orange Pi Zero:

```
make orangepi_zero_defconfig
```

Solamente a título informativo, se le invita a ver las siguientes opciones y sus valores. Si alguno difiere, debe ser actualizado a lo que se muestra a continuación:

- Target options
    - Target Architecture: **ARM (little endian)**
    - Target Architecture Variant: **cortex-A7**
    - Target ABI: **EABIhf**
    - Floating point strategy: **VFPv4**
- Toolchain
    - Toolchain type: **External toolchain**
    - Toolchain: **Custom toolchain**
    - Toolchain origin: **Pre-installed toolchain**
    - Toolchain path: vamos a utilizar en juego de herramientas que construimos, debemos copiar el resultado del comando ***echo $CROSSTOOL_ROOT/xtools/arm-FIUBA-linux-gnueabihf/bin***
    - Toolchain Prefix: **arm-linux**
    - External toolchain gcc version: **7.x**
    - External toolchain kernel headers series: **4.16.x**
    - External toolchain C library: **glibc/eglibc**
    - Debemos decirle a Buildroot acerca de la configuración de nuestro juego de herramientas, por lo tanto, tilde las opciones **Toolchain has RPC support?** y **Toolchain has C++ support?**. Buildroot verificar estos parámetros de igual manera.
    - Habilitar **Copy gdb server to the Target**
- Filesystem images

- Habilite **tar the root filesystem**

Se invita al lector a explorar las distintas opciones que ofrece la herramienta de configuración para generar un sistema adecuado a sus necesidades.

Compilando BuildRoot

Habiendo generado el archivo .config correspondiente, el paso siguiente es compilar BuidRoot. Se hace ejecutando:

```
make -j4
```

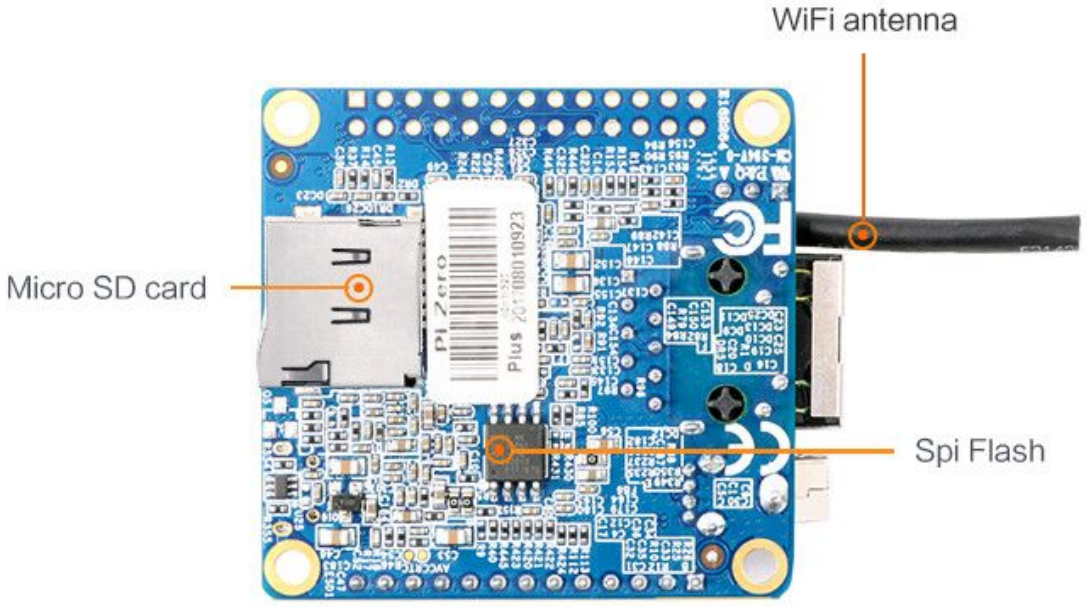
Luego de la compilación, todos los archivos de salida que debemos utilizar se encuentran en el el path ***$BUILDROOT_SRC/output/images***:

- **zImage**, que no es más que una imagen de kernel, como la generada anteriormente.
- **sun8i-h2-plus-orangepi-zero.dtb** que es el device tree.
- **u-boot-sunxi-with-spl.bin** que es el archivo que utilizamos para grabar en la memoria SD.
- **rootfs.tar** que es el file system completo comprimido.

Para utilizar todos estos archivos se deben seguir los mismos criterios y configuraciones que en los apartados anteriores. Para poblar la carpeta ***/nfsroot_test*** generada anteriormente, se debe descomprimir el archivo ***rootfs.tar*** en ella.

Grabando U-boot en la memoria SPI

Ahora que todo el sistema está funcionando y la SCB carga el kernel de linux y bootea correctamente, podemos comenzar a hacer algunos cambios y experimentar con el hardware de la Orange Pi Zero.
De fábrica, estas SCB tienen incorporada una memoria SPI como se ve en la siguiente imagen:



Esta memoria SPI es (dependiendo el modelo de Orange Pi comprada) una Macronix MX25L1606E, de 16 Mbit. Los siguientes links pueden ser de utilidad para ahondar en el tema:

http://linux-sunxi.org/Bootable_SPI_flash#SPI_driver
http://linux-sunxi.org/Xunlong_Orange_Pi_Zero#FEL_Mode

Para poder cargar el binario de U-boot generado, debemos instalar un set de herramientas que el fabricante da en forma de scripts, llamados ***sunxi-tools***. Ejecutamos los siguientes comandos

```
cd $ORANGEPI_ROOT
git clone https://github.com/linux-sunxi/sunxi-tools
cd sunxi-tools
make
```

Dentro de este directorio, una vez compilado, tendremos algunos scripts de utilidad, entre ellos uno llamado ***sunxi-fel***.
El FEL de la Orange Pi es una subrutina de bajo nivel contenida en el BootROM de los dispositivos Allwinner. Es utilizado para programación inicial y recuperación de dispositivos utilizando USB. El FEL es en realidad un stack usb pequeño, el cual implementa un protocolo USB especial.
El script ***sunxi-fel*** es utilizado para cargar información en la SCB, por lo que lo utilizaremos para cargar el binario de u-boot en la memoria SPI.
Para lograr entrar en modo FEL, se debe conectar a la PC de trabajo mediante USB al puerto mini-USB, que también es utilizado para la alimentación de la SCB. No debe haber ninguna memoria SD conectada, ni tampoco dispositivos USB.
Cumplimentados estos requisitos, hacemos una copia del archivo ***u-boot-sunxi-with-spl.bin*** en la carpeta ***/sunxi-tools*** donde se compiló las herramientas, y se ejecuta el siguiente comando:

```
sunxi-fel -v -p spiflash-write 0 u-boot-sunxi-with-spl.bin
```

Luego de algunos segundos, la copia de U-boot a la memoria SPI ya está lista y podremos iniciar el sistema desde la memoria SPI.

Hay un detalle y no es menor, el cual es necesario para que esto funcione: la configuración de U-boot para poder lograr este comportamiento debe tener los siguientes flags seteados:

- CONFIG_SPL_SPI_FLASH_SUPPORT
- CONFIG_SPI_BOOT
- CONFIG_SPL_SPI_SUNXI

Por suerte, estos flags estan seteados por defecto en el archivo .config creado en los apartados anteriores, por lo que no hace falta re-compilar el binario.
Ahora se insta al lector a probar conectar por consola serie la SCB sin insertar ninguna memoria SD y ver que el dispositivo inicia y carga U-boot desde la memoria SPI.

Grabando una memoria SD con el sistema de archivos

Avanzando un paso más lejos, podremos iniciar todo nuestro sistema desde una memoria SD, teniendo U-boot en la memoria SPI, cargando el device tree y el kernel desde la misma memoria SD.

Preparación de la memoria SD

Siga los siguientes pasos:

- Se recomienda tomar una memoria SD y formatearla en formato ext4, con solamente una partición.
- Luego extraiga el archivo ***rootfs.tar*** generado por BuildRoot dentro de la memoria SD.
- Cree un directorio llamado ***boot*** dentro de la memoria SD, el cual esté en la raíz de la misma.
- Copie los archivos ***zImage*** y ***sun8i-h2-plus-orangepi-zero.dtb*** dentro del directorio ***/boot***

Su memoria SD está lista para iniciar el sistema.

Iniciando el sistema con U-boot

Una vez que tenga la memoria SD lista, insertela en la SCB, conecte la consola serie a una estación de trabajo y energice la SCB.
El prompt de U-boot debería desplegarse en la consola; presione cualquier tecla para detener el booteo automático. En este punto ejecute los siguientes comandos para U-boot:

```
setenv bootargs root=/dev/mmcblk0p1 rootwait rw console=ttyS0,115200
ext4load mmc 0:1 0x42000000 /boot/zImage
ext4load mmc 0:1 0x43000000 /boot/sun8i-h2-plus-orangepi-zero.dtb
bootz 0x42000000 - 0x43000000
```

Estos comandos harán que:

- Se utilice como punto de montaje principal la memoria SD, bloque 0 particion 1 (unica en esta memoria).
- Se carge el archivo zImage desde su ubicación en el bloque 0 partición 1 a la posición de memoria 0x42000000.
- Se carge el archivo sun8i-h2-plus-orangepi-zero.dtb desde su ubicación en el bloque 0 partición 1 a la posición de memoria 0x43000000.
- Se inicie la secuencia de booteo.

Luego de algunos segundos, debería ser bienvenido con el mensaje de login para usuario root.

Conclusiones

Se ha dado un pantallazo general de como hacer un sistema mínimo en una Single Board Computer de la empresa Sunxi, conocida como Orange Pi Zero. Siendo una variante que puede llegar a costar hasta un tercio del valor de una BeagleBone Black, es una muy buena opción desde el punto de vista didáctico y para adquirir conocimientos en los tópicos abordados en esta guía.
Se insta al lector a que experimente más allá de los límites de esta guía, la cual sirve solamente de punta pie inicial, y no de punto final para desarrollar conocimientos en linux para sistemas embebidos.