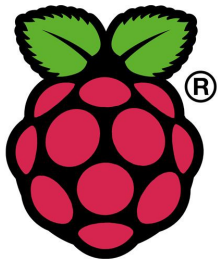
OpenDomo en Raspberry Pi



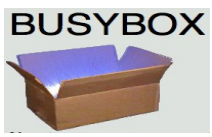
Administración de Redes y Sistemas Operativos en Entornos de Software Libre

Autor: David Sánchez Herrero

Consultor: Jordi Massaguer Pla

Tutor Externo: Oriol Palenzuela i Rosés

Langreo, 20 de junio de 2013



Copyright

Autor

David Sánchez Herrero

Para cualquier comentario o sugerencia, contactar en david.sanchez.herrero en gmail.com

Publicación

Contenido actualizado el 30/06/13.

Documento creado y basado en LibreOffice 3.

Licencia

Resumen

Este proyecto tiene como propósito portar el sistema OpenDomo, que corre actualmente sobre un dispositivo empotrado con arquitectura x86 llamado ODNetwork, para su ejecución sobre el dispositivo Raspberry Pi, con arquitectura ARM.

OpenDomo es un sistema embebido libre desarrollado por OpenDomo Services S.L., basado en GNU/Linux, y diseñado principalmente para el control de instalaciones domóticas.

El desarrollo del proyecto abarca desde la generación de un sistema embebido GNU/Linux básico, construido mediante la compilación y unión de todos los componentes utilizando una herramienta llamada Buildroot, hasta la transformación de este sistema base en OpenDomo, gracias a un conjunto de shell scripts denominados Opendomo SDK, junto con diversos paquetes software desarrollados a medida.

Índice de contenido

1 Introducción

En la actualidad existen numerosos dispositivos empotrados, destinados a diversas tareas, que ejecutan pequeños sistemas operativos para llevar a cabo sus funciones. Ejemplos de estos dispositivos son: discos duros multimedia, reproductores de DVD, receptores GPS, routers, switches, etc.

En muchas ocasiones, los fabricantes de estos dispositivos diseñan desde cero los sistemas operativos y programas que se ejecutarán en su interior, sin embargo, en otros casos optan por la utilización de sistemas libres (o mixtos), permitiéndoles simplificar el desarrollo del sistema, y en definitiva, ahorrar tiempo y abaratar costes. Esto es posible gracias a la versatilidad de estos sistemas, que pueden ejecutarse sobre una gran cantidad de arquitecturas y soportar una gran variedad de hardware. Sumado a sus innumerables opciones de configuración, y a la multitud de herramientas libres desarrolladas alrededor de los mismos, hace posible la construcción de este tipo de sistemas para casi cualquier dispositivo, sea cual sea su hardware y finalidad.

Como ejemplos de estos sistemas libres, tenemos los núcleos Linux y FreeBSD entre otros, junto con la gran cantidad de herramientas disponibles para los mismos, desarrolladas muchas de ellas dentro del proyecto GNU.

En este marco encontramos a la empresa OpenDomo Services S.L., cuyo ámbito de negocio se basa en el diseño, instalación y mantenimiento de instalaciones domóticas, utilizando para ello una serie de dispositivos empotrados que ejecutan sistemas libres. El principal sistema libre utilizado recibe el nombre de OpenDomo, y se basa en un sistema Linux embebido (núcleo, configuraciones y paquetes), sobre el que se han realizado modificaciones y desarrollos propios hasta obtener un sistema capaz, entre otras cosas, de controlar toda la instalación domótica de una vivienda. Este sistema, actualmente se ejecuta sobre un dispositivo empotrado que recibe el nombre de ODNetwork.

Por otro lado, encontramos a la Fundación Raspberry Pi, que es una asociación caritativa registrada en la Comisión de Caridad de Inglaterra y Gales, cuyo objetivo es promover el estudio de las ciencias de la computación y temas relacionados, sobre todo a nivel escolar. Esta fundación ha desarrollado una placa computadora de bajo coste, del tamaño de una tarjeta de crédito, que contiene todos los dispositivos y conexiones típicas de un PC convencional. Alrededor de este dispositivo, se han desarrollado distribuciones GNU/Linux que permiten utilizarlo para un sinfín de propósitos.

El presente proyecto final de máster pretende fusionar el sistema Opendomo con el dispositivo Raspberry Pi, permitiendo a este último, entre otras cosas, actuar como controlador central en una instalación domótica, sustituyendo al actual controlador ODNetwork.

1.1 Objetivos

La empresa Opendomo Services S.L., con la que colaboraré durante la realización de este proyecto, ha desarrollado un sistema operativo embebido llamado OpenDomo, basado en GNU/Linux, para controlar instalaciones domóticas. Se genera compilando desde cero todas las herramientas, obteniendo finalmente un sistema adaptado a un dispositivo llamado ODNetwork con arquitectura

x86. Para llevar a cabo el proceso, se utiliza una herramienta libre llamada Buildroot, en combinación con un conjunto de shell scripts desarrollados a medida que reciben el nombre de OpenDomo SDK.

El objetivo principal del proyecto, es lograr la portabilidad del sistema OpenDomo para su ejecución sobre el dispositivo Raspberry Pi, con arquitectura ARM. Esta portabilidad incluye la modificación de OpenDomo SDK para que, mediante simple parametrización de los scripts, sea capaz de generar el sistema para cualquiera de las dos arquitecturas antes mencionadas.

En primer lugar, será necesario establecer la configuración de compilación de todas las herramientas software necesarias (Buildroot, uClibc, Busybox y el kernel Linux), para generar desde cero un sistema embebido completo que funcione sobre Raspberry Pi. Durante este proceso, se obtendrá un toolchain capaz de realizar la compilación cruzada de todas las herramientas, generando código binario para ARM desde un PC con arquitectura x86 o x86_64.

Posteriormente, se modificará OpenDomo SDK para construir, a partir del sistema embebido anterior, un sistema OpenDomo totalmente funcional sobre este dispositivo hardware. Por último, se fusionará la versión de OpenDomo SDK modificada durante el proyecto con la versión oficial del mismo, que genera el sistema para el dispositivo ODNetwork, obteniendo así una versión de OpenDomo SDK que, mediante simple parametrización, sea capaz de generar el sistema OpenDomo tanto para ODNetwork como para Raspberry Pi.

A nivel personal, los objetivos que espero cumplir son:

- Poner en práctica los conocimientos adquiridos a lo largo de estos estudios.
- Adquirir conocimientos técnicos sobre sistemas y dispositivos empotrados, que hasta ahora se encontraban fuera de mi ámbito académico y profesional.
- Mejorar mis habilidades en el desarrollo de la documentación asociada a un proyecto compuesto íntegramente por Software Libre, especialmente en el ámbito de la redes y los sistemas operativos.
- Familiarizarme y perfeccionar los métodos de trabajo utilizados en los proyectos libres, basados no sólo en la lectura de documentación, sino en la experimentación y el contacto directo con los desarrolladores y las comunidades de usuarios, generando retroalimentación cuando sea posible (reporte de bugs, sugerencias de funcionalidades, etc.).

2 Análisis de requisitos y de viabilidad

Dadas las especiales características de este proyecto, donde ya se parte de un sistema completamente diseñado y funcional, que debe ser portado a otra plataforma hardware, también ya diseñada, el estudio de requisitos y de viabilidad no requiere una gran extensión.

2.1 Estudio de viabilidad

2.1.1 Necesidades y requisitos del cliente

Opendomo Services S.L., que dispone de un sistema operativo libre para el control domótico adaptado a un dispositivo hardware específico, requiere diversificar el número de dispositivos sobre los que su sistema es capaz de ejecutarse, ampliando así el abanico de opciones a ofrecer a sus clientes.

Esta diversidad de dispositivos hardware permitirá ofrecer soluciones de diferente coste, así como facilitar a los usuarios de la comunidad que se aglutina en torno este sistema libre, la posibilidad de realizar pruebas y desarrollos propios. También brindará cierta independencia de la plataforma hardware actual.

2.1.2 Análisis de la situación actual

Actualmente, se utiliza un dispositivo hardware llamado ODNetwork que dispone de las siguientes características:

Ilustración 2.1: Dispositivo ODNetwork

1. Especificaciones:

- Procesador: 1 Ghz (x86)
- Memoria RAM: 512 MByte DDR2
- Tarjeta gráfica: 32MB, resolución máxima 1920 x 1200
- Tarjeta de sonido: compatible AC97 v2.2
- Red: 10/100 Mbps
- Puertos USB 2.0: 3
- Dimensiones: 90 x 90 x 20 mm
- Almacenaje: SD 2GB

2. Precio: 300 Euros aproximadamente.

2.1.3 Definición de los requisitos del sistema

El nuevo dispositivo hardware debe disponer de unas especificaciones, firmware y controladores lo más libres posible, ser de pequeño tamaño, bajo coste, silencioso, y con unos consumos de energía reducidos. Además de esto, debe tener la capacidad de procesamiento necesaria para ejecutar el sistema de control domótico y los diferentes módulos y software que lo dotan de diferentes funcionalidades.

2.1.4 Estudio de alternativas de solución

Actualmente, existen en el mercado diferentes dispositivos que cumplen total o parcialmente los requisitos expuestos anteriormente. Algunos de estos dispositivos son:

b) Raspberry Pi

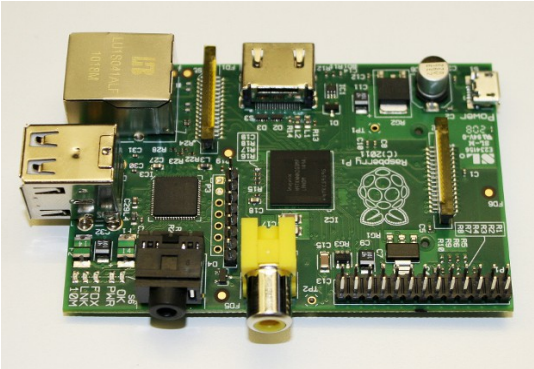


Ilustración 2.2: Dispositivo Raspberry Pi Modemo B

1. Especificaciones:

- CPU: ARM1176JZF-S (ARM11) a 700 Mhz.
- GPU: Broadcom VideoCore IV, OpenGL ES 2.0, -2 y VC-1 (con licencia), 1080p30 H.264/MPEG-4 AVC.
- Memoria: 512 MB SDRAM compartidos con la GPU.
- Puertos USB: 2 puertos.
- Salidas vídeo: Conector RCA (PAL y NTSC), HDMI (rev1.3 y 1.4), Interfaz DSI para panel LCD.
- Salidas Audio: 3.5 mm jack, HDMI.
- Almacenamiento: SD / MMC / ranura para SDIO.
- Red: 10/100 Ethernet (RJ45) via hub USB.
- Consumo energético: 700 mA, (3.5 W).
- Alimentación: 5V vía Micro USB o GPIO header.
- Dimensiones: 85.60mm × 53.98mm (3.370 × 2.125 inch).

2. Precio: 35 Dólares aproximadamente.

c) Carambola

Ilustración 2.3: Dispositivo Carambola

1. Especificaciones:

- CPU: RT3050 a 2.4 Ghz.
- Memoria Flash: 8 MB.
- Memoria RAM: 32 MB.
- Red: Estándar Wireless 802.11 bgn, puerto para antena en el chip.
- Alimentación: 3.3v.
- Consumo energético: 1.5 W.
- Dimensiones: 35 x 45 mm.

2. Precio: 22 Euros aproximadamente.

d) Cubieboard

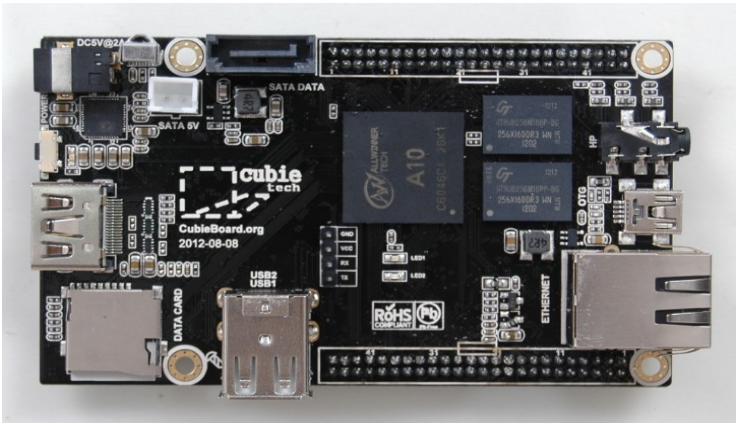


Ilustración 2.4: Dispositivo Cubieboard

1. Especificaciones:

- CPU: 1G ARM cortex-A8, NEON, VFPv3, 256KB caché L2.
- GPU: Mali400, OpenGL ES.
- Memoria: 512M/1GB DDR3 480MHz.
- Salidas vídeo: HDMI 1080p.
- Red: 10/100 Ethernet.
- Memoria Flash: 4Gb.
- Puertos USB: 2 puertos.
- Almacenamiento: 1 ranura micro SD, 1 puerto SATA, y 1 puerto IR.

2. Precio: 49 Dólares aproximadamente.

e) A13-OLinuXino WiFi DEV

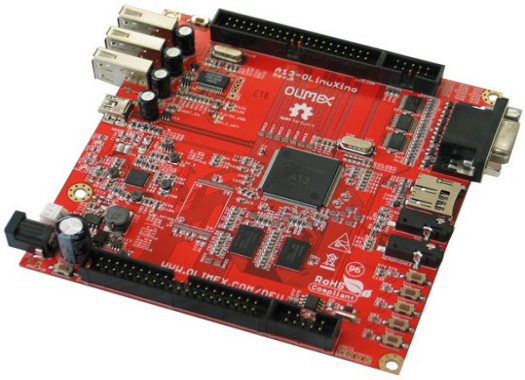


Ilustración 2.5: Dispositivo A13-OLinuXino WiFi

1. Especificaciones:

- CPU: A13 Cortex A8 a 1GHz.
- GPU: 3D Mali400.
- Memoria: 512 MB RAM.
- Memoria Flash: 4GB.
- Puertos USB: 3 puertos.
- Red: WIFI RTL8188CU 802.11n 150Mbit.
- Alimentación: 1 USB OTG
- Almacenamiento: SD-card.
- Salida de vídeo: VGA.
- Salida de audio.
- Entrada de audio para micrófono.

2. Precio: 55 Euros aproximadamente.

f) Hackberry A10

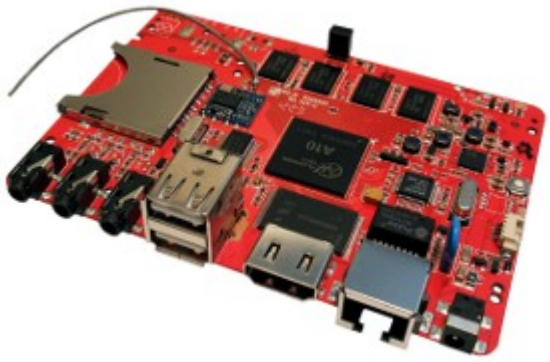


Ilustración 2.6: Dispositivo Hackberry A10

1. Especificaciones:

- CPU: 1.2 Ghz Allwinner A10 ARM Cortex A8
- GPU: Mali400 con aceleración 3D y decodificación de video por hardware.
- Memoria: 512 MB o 1 GB, reservando 100 MB aproximadamente para la GPU.
- Salida de video: HDMI 1080p, 3.5 mm AV compuesta, 3.5 mm por componentes Y/Pb/Pr
- Salida de audio: HDMI.
- Entrada de audio: jack 3.5 mm para micrófono.
- Puertos USB: 2 puertos.
- Almacenamiento interno: 4 GB NAND.
- Almacenamiento externo: Tarjetas SDHC de hasta 32 GB.
- Red: 10/100 Ethernet y Realtek 802.11n WiFi.
- Alimentación: Adaptador NEMA de 2 pines. Entrada AC100-240V-0.4A 50/60 Hz. Salida DC5v.

2. Precio: 60 – 65 Dólares aproximadamente según memoria RAM.

g) Open Exynos4 Quad

Ilustración 2.7: Dispositivo Open Exynos4 Quad

1. Especificaciones:

- CPU: Samsung Exynos4412 Cortex-A9 Quad Core 1.4Ghz con de caché 1MB L2.
- GPU: Mali400 de 4 núcleos.
- Memoria: 1 GB DDR2 de RAM.
- Salida de vídeo: micro HDMI 1080p (H.264+AAC sobre contenedor MP4).
- Salida de audio: Jack de 3.5mm, HDMI y SPDIF.
- Entrada de audio: Jack de 3.5 mm para micrófono.
- Red: 10/100 Ethernet.
- Puertos USB: 6 puertos.
- Almacenamiento: Tarjetas SDHC, puerto eMMC.
- Alimentación: 5V 2A.
- Dimensiones: 90 x 94 mm.

2. Precio: 129 Dólares aproximadamente.

2.1.5 Valoración y elección de las posibles soluciones

De entre todas las opciones disponibles, Opendomo Services S.L. ha optado por escoger el dispositivo Raspberry Pi para el proceso de portabilidad de su sistema embebido. Las razones por las cuales se ha seleccionado este dispositivo son fundamentalmente las siguientes:

- Petición masiva por parte de los usuarios de la comunidad libre en torno a sus productos.
- Éxito del dispositivo, lo que conlleva una gran comunidad de usuarios y desarrolladores alrededor del mismo a nivel mundial.
- Aunque usa firmware y algunos controladores de código cerrado, las especificaciones y la

mayor parte de los drivers son libres.

- Bajo coste del dispositivo.
- Sus prestaciones, que lo capacitan para desempeñar la función de controlador domótico.
- Reducido consumo y dimensiones.

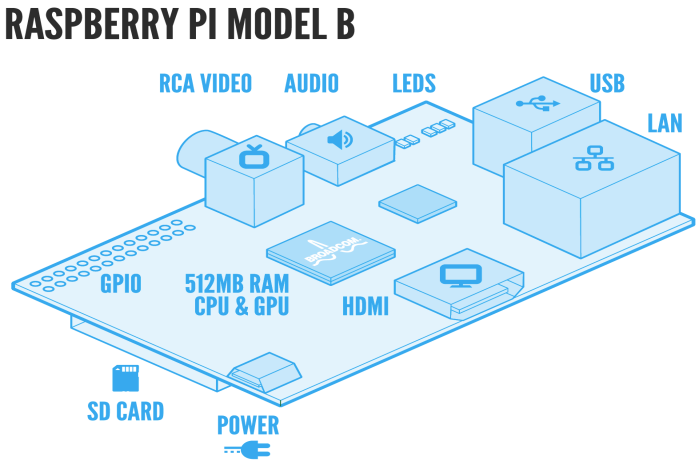


Ilustración 2.8: Esquema técnico Raspberry Pi modelo B

El coste de portar su sistema a este dispositivo será bajo, pues al tratarse de software de código abierto (tanto el kernel Linux, como las herramientas de GNU, así como los desarrollos propios de Opendomo, junto con el software Buildroot usado para construir el sistema completo), solamente será necesario llevar a cabo una reconfiguración del mismo que lo capacite para su funcionamiento en este nuevo dispositivo hardware con arquitectura ARM.

Teniendo en cuenta todo lo anterior, se concluye que el proyecto en su conjunto es viable.

2.2 Análisis del sistema

2.2.1 Requisitos exactos del proyecto

- Requisitos de propiedad intelectual y licencias

En proyecto debe basarse en software libre, o con licencias lo menos restrictivas posible. El motivo de esto no es sólo económico, sino que es necesario disponer del código de todo el software con el fin de poder portarlo (como es el caso del proyecto que nos ocupa) a diferentes plataformas, lo que implica un proceso de configuración y compilación del mismo. Además, existe una comunidad que realiza pruebas, corrección de errores, mejoras y desarrollos propios en torno al software de la empresa, por lo que existe un proceso de colaboración mutua que se ve facilitado por este tipo de licencias.

- Requisitos de prestaciones

El funcionamiento del sistema sobre la nueva plataforma debe ser estable y fluido. Se espera que una vez arrancado el sistema en el dispositivo, pase largos periodos sin apagarse o reiniciarse, por lo

que debe ofrecer durante este tiempo un rendimiento aceptable, sin cuelgues, bloqueos ni pérdidas de rendimiento. El dispositivo debe soportar estos largos periodos de funcionamiento sin averías motivadas por el recalentamiento de los componentes u otras causas.

Para lograr el rendimiento esperado, el dispositivo admite diferentes configuraciones que permiten balancear la cantidad de memoria RAM asignada a la CPU (sistema operativo y aplicaciones) y a la GPU, así como parámetros para modificar la frecuencia de reloj de los diferentes componentes (overclocking) con seguridad, sin perder la garantía ni afectar a la vida del dispositivo. Una vez migrado el sistema, habrá una fase dedicada a encontrar la configuración óptima del hardware.

- Requisitos de implantación

Todo el sistema se compila desde cero a partir de una herramienta libre llamada Buildroot, y un conjunto de shell scripts conocidos como OpenDomo SDK que parametrizan la ejecución de la herramienta anterior y procesan la salida generada por la misma, modificándola y agregando los componentes necesarios para generar el sistema OpenDomo funcional sobre ODNetwork. Una vez determinadas las configuraciones, los pasos a seguir, y las modificaciones a aplicar sobre OpenDomo SDK para obtener un sistema OpenDomo funcional sobre Raspberry Pi, se debe obtener una nueva versión de OpenDomo SDK capaz de generar el sistema OpenDomo tanto para ODNetwork como para Raspberry Pi, mediante simple parametrización de su ejecución. Por tanto, el sistema desarrollado para generar Opendomo sobre Raspberry Pi debe integrarse con el sistema actual, agregando funcionalidad, pero sin alterar la funcionalidad actual.

- Requisitos de distribución

Una vez finalizado el proyecto, los resultados del mismo deben estar disponibles públicamente para su descarga tanto en formato binario como en código fuente. Los recursos necesarios para su publicación correrán por parte de Opendomo Services S.L., que utilizará su página web así como su repositorio Subversion para este fin.

2.2.2 Especificación del plan de pruebas

El plan de pruebas a ejecutar en el proyecto que nos ocupa tiene una serie de dificultades añadidas, sujetas a las siguientes cuestiones:

- No ha sido posible comprobar el funcionamiento del sistema en el dispositivo original, ODNetwork. Por tanto, no se tiene un marco real de referencia.
- Las únicas pruebas llevadas a cabo con el sistema en producción han sido ejecutadas sobre una máquina virtual, con las limitaciones que esto implica.
- No se dispone de ningún dispositivo externo, aparte de teclado, tarjeta SD y monitor, al que conectar la Raspberry Pi con el sistema portado. Esto implica que la interacción del sistema con otros dispositivos tanto de la instalación domótica como de otro tipo (por ejemplo, Arduino), no pueden ser probados.
- El encargado de ejecutar el proyecto, no tiene experiencias previas con el sistema a portar, ni con los dispositivos utilizados por el cliente (ODNetwork, ODControl, Arduino y otros dispositivos empleados por Opendomo Services S.L.), ni con instalaciones domóticas en general.

Por todos estos motivos, ha sido necesario acordar con el cliente un conjunto de pruebas mínimas que, independientemente de estos problemas, puedan ser ejecutadas por el desarrollador del proyecto de forma autónoma. El enfoque inicial pretendía involucrar a la comunidad de usuarios en el proceso de pruebas, pero dada la necesidad de seguir una planificación, y la existencia de una fecha límite para la entrega, este enfoque ha tenido que ser dejado de lado. Esto no quiere decir que la comunidad no pueda hacer pruebas, ni que sus resultados vayan a ser ignorados, pero no implicarán compromiso de modificaciones al software ni a la planificación establecida por parte del desarrollador.

A lo largo del proyecto, se realizarán diversos intentos para encontrar la forma más adecuada de llevar a cabo la portabilidad. En diversos momentos de la vida del proyecto, se alcanzarán versiones parcial o totalmente funcionales del sistema que serán sometidas a una pequeña batería de pruebas para evaluar la evolución y estado del mismo (arranque, carga de módulos y configuración, arranque de servicios básicos, etc.), y que pasarán a disposición del cliente y de la comunidad de usuarios. Estos, apoyándose en su conocimiento del sistema y de instalaciones domóticas, y gracias a tener en su poder dispositivos externos con los que interaccionar, podrán realizar pruebas a la versión generada. Para este fin, Opendomo Services S.L. dispone de una batería de pruebas estándar (5.1) a la que se someten los sistemas desplegados sobre el dispositivo ODNetwork, y que aplicarán al port para Raspberry remitiendo posteriormente los resultados.

Este ciclo se repetirá en sucesivas ocasiones hasta alcanzar la versión definitiva, momento en el que comenzará el proceso de desarrollo de la versión de OpenDomo SDK modificada que finalmente se fusionará con la rama oficial, generando la versión final. Este último apartado tendrá otro ciclo de desarrollo y pruebas análogo al anterior, cuyo objetivo será alcanzar la versión definitiva de OpenDomo SDK.

Esquemáticamente, los ciclos mencionados pueden verse en el siguiente gráfico:

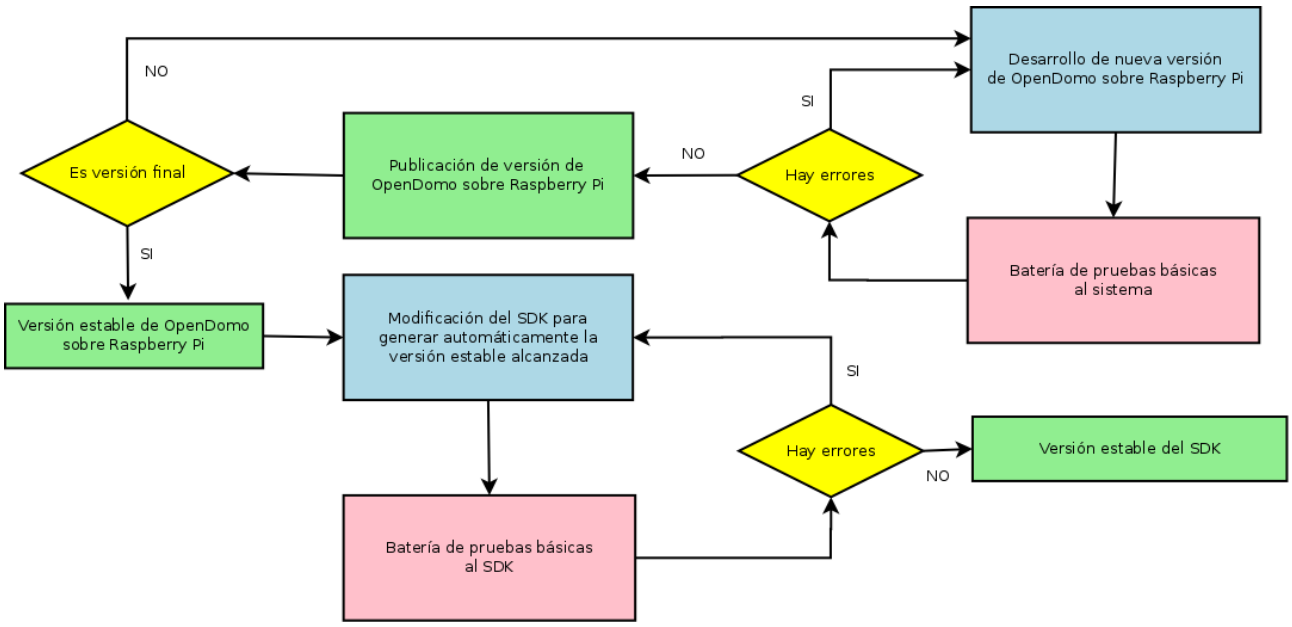


Ilustración 2.9: Ciclos de desarrollo y pruebas del proyecto

Dentro de lo que se ha denominado “Batería de pruebas básicas al sistema” se incluyen:

a) Pruebas unitarias:
   1. El sistema arranca sin errores.
   2. El sistema se conecta a la red por DHCP.
   3. El sistema es capaz de encontrar los paquetes y la configuración.
   4. El sistema es capaz de instalar los paquetes y poner en funcionamiento los servicios.
   5. El sistema reconoce el teclado.
   6. El sistema muestra el prompt, y es posible hacer login e interaccionar con normalidad.
b) Pruebas de sistema:
   1. El sistema está accesible a través de SSH y del navegador web en el puerto 80.
   2. La interfaz web funciona con normalidad.
   3. El sistema ofrece un rendimiento adecuado.

Dentro de lo que se ha denominado “Batería de pruebas básicas al SDK” se incluyen:

a) Pruebas unitarias:
   1. Las funcionalidades añadidas se ejecutan sin mostrar errores.
   2. Las funcionalidades añadidas se integran en interaccionan con el resto del SDK.
b) Pruebas de sistema:
   1. El initramfs para Raspberry Pi generado por el nuevo SDK contiene todos los componentes necesarios.
   2. El initramfs para Raspberry Pi generado está correctamente construido y es cargado correctamente por el dispositivo.
   3. El sistema completo para Raspberry Pi, generado a través nuevo SDK, supera satisfactoriamente las pruebas denominadas “Batería de pruebas básicas al sistema”.

3 Diseño de la solución e implementación

Los siguientes apartados mostrarán la arquitectura global de los sistemas domóticos propuestos por OpenDomo Services S.L., y en qué punto de los mismos se integrará el dispositivo Raspberry Pi con el sistema portado.

Posteriormente, se detallarán los pasos seguidos para la obtención del sistema portado para el nuevo dispositivo.

3.1 Diseño

3.1.1 Arquitectura

Globalmente, se puede dividir la arquitectura del proyecto en dos niveles, aportando cada uno de ellos diferentes puntos de vista:

- **Arquitectura funcional**, que refleja a grandes rasgos el sistema domótico en su conjunto, y en qué parte del mismo se ubicará el dispositivo Raspberry Pi con el sistema OpenDomo portado.
- **Arquitectura lógica**, que muestra en detalle el funcionamiento interno del sistema, con las diferentes funciones y servicios que ofrece.

3.1.1.1 Arquitectura Funcional

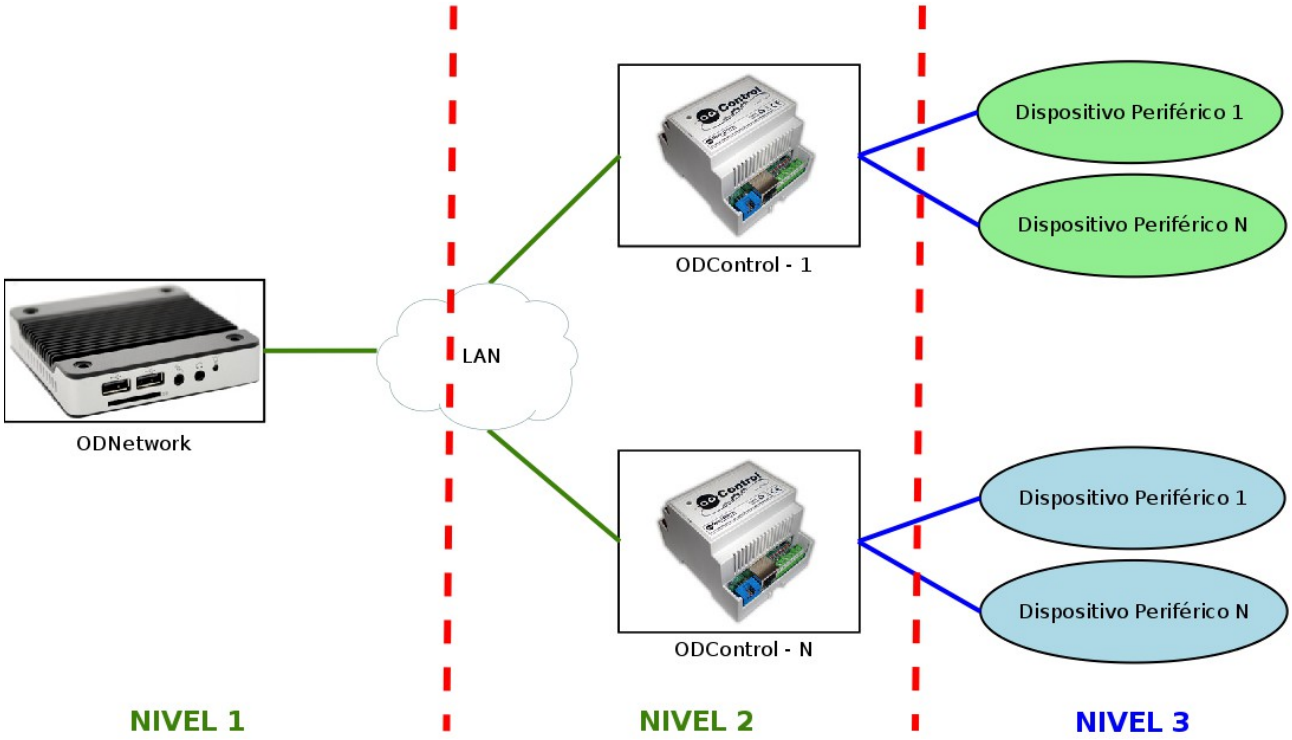


Ilustración 3.1: Arquitectura funcional de los sistemas domóticos con OpenDomo

Como puede verse en el gráfico anterior, la arquitectura general del sistema domótico se basa en una infraestructura de red estándar, compuesta por dos niveles. Dentro del primer nivel se encuentra conectado el agente principal de OpenDomo (ODNetwork), que constituye el núcleo de la instalación. La Raspberry Pi con el sistema OpenDomo se sitúa en este primer anillo sustituyendo a ODNetwork, y se conecta a la red a través de su puerto RJ45 incorporado, o mediante una interfaz de red inalámbrica suministrada a través de un puerto USB.

El segundo nivel está compuesto principalmente por dispositivos ODControl, cada uno con una

configuración diferente dependiendo de la función a desempeñar en la instalación (control de iluminación, control de motorización para puertas y persianas, …).

Interconectando los dispositivos del primer y segundo nivel, habrá uno o varios dispositivos de red estándar (routers, switches, puntos de acceso, etc.), que podrán comunicar también la instalación domótica con la LAN doméstica, e incluso ofrecer salida a Internet si fuera necesario.

Existe un tercer nivel, que ya no forma parte de la infraestructura de red, donde se sitúan los diferentes módulos de salidas y entradas, sensores de movimiento, crepusculares, termostáticos, etc., que se comunican con los dispositivos conectados a la red desde el segundo nivel, pero utilizando conexiones eléctricas y/o electrónicas.

Un único ODControl es capaz de gestionar diversos dispositivos del tercer nivel destinados a una mismo fin, y a su vez, un único dispositivo con OpenDomo (ODNetwork o Raspberry Pi) es capaz de gestionar múltiples dispositivos ODControl, que desempeñarán diferentes funciones como se ha indicado anteriormente. En instalaciones domóticas complejas, pueden existir otros dispositivos (videovigilancia, medición de consumos, etc.) que, dependiendo de sus características, irán conectados en el segundo o tercer nivel.

Seguidamente, se muestra un esquema de las diferentes partes que componen el dispositivo ODControl.

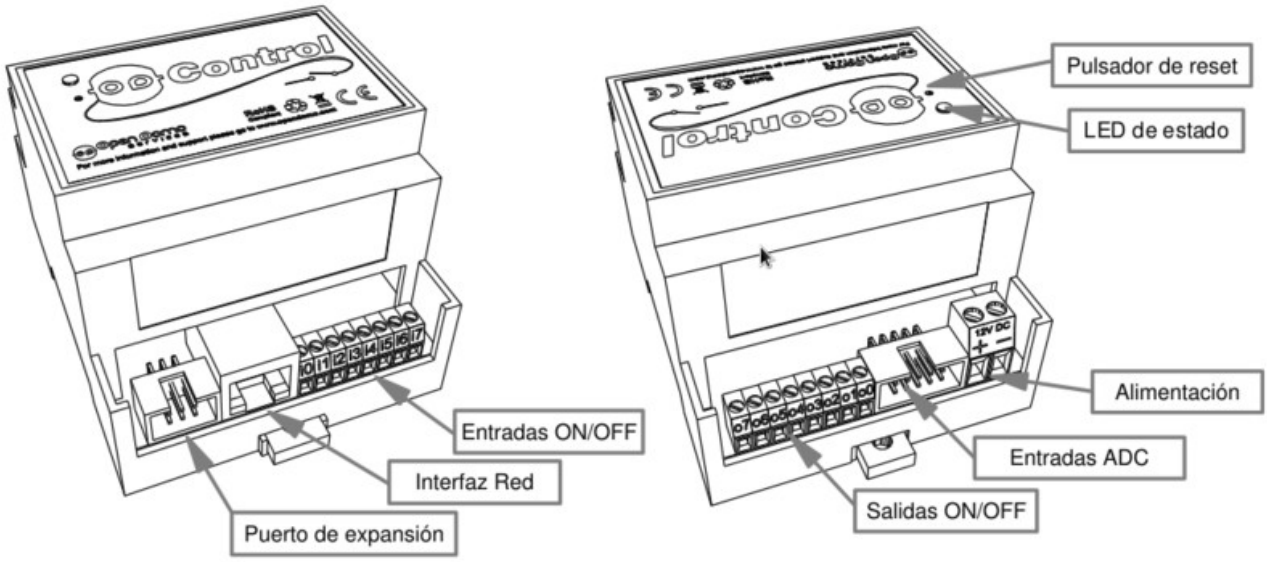


Ilustración 3.2: Esquema técnico del dispositivo ODControl

3.1.1.2 Arquitectura Lógica

- **Sistema operativo**

El sistema OpenDomo está basado en el núcleo Linux, y su arquitectura puede resumirse en las siguientes capas:

– Núcleo Linux, que gestiona la E/S, los dispositivos, la memoria y los procesos del sistema.

Es la parte más próxima al hardware, y proporciona un nivel de abstracción que permite desarrollar programas capaces de ejecutarse sobre máquinas muy diferentes.

- Herramientas básicas Busybox, que funcionan sobre el núcleo, y consisten principalmente en un único ejecutable de pequeño tamaño, que empaqueta versiones reducidas de multitud de herramientas para sistemas UNIX (y derivados), muchas de las cuales han sido desarrolladas por el proyecto GNU (comandos básicos del sistema, editores, la shell...). En los directorios donde normalmente se encuentran los binarios de estas herramientas, se encuentran enlaces simbólicos que apuntan directamente a este ejecutable.
- Paquetes OpenDomo, que forman parte de la capa de aplicación, proporcionando cada uno de ellos una serie de funcionalidades adicionales al sistema. Actualmente, existen los siguientes paquetes:
  a) opendomo-ai: contenedor de rutinas de Inteligencia Artificial, módulo experimental para desarrollo.
  b) opendomo-alerts: permite que el sistema OpenDomo pueda enviar notificaciones al usuario, ya sea a través de correo electrónico o mediante el servicio de alertas en el Cloud (opcional).
  c) opendomo-cgi: interfaz web del sistema OpenDomo, que actúa como la interfaz principal.
  d) opendomo-common: archivos comunes empleados por todo el sistema, como las funciones de secuencias y escenas, condiciones, librerías, etc.
  e) opendomo-discovery: servicio de sincronización entre agentes de la red. Detecta y configura otros ODNetworks para que puedan compartir servicios y distribuir la carga.
  f) opendomo-events: módulo de gestión de eventos, dedicado a enlazar cada evento con una acción respuesta determinada, configurable por el usuario (opcional).
  g) opendomo-hal: capa de abstracción de hardware, donde encontramos los drivers para distintas tecnologías de controladores como ODControl, X10, Arduino, etc.
  h) opendomo-music: conjunto de servicios para convertir el agente OpenDomo en un reproductor de música para el hilo musical (opcional).
  i) opendomo-pkg: gestor de paquetes y actualizaciones automáticas.
  j) opendomo-speech: sintetizador de voz, para poder generar notificaciones verbales (opcional).
  k) opendomo-vision: gestor básico de cámaras IP (opcional, no estable).
  l) kernel_sound: controladores y binarios para los servicios de audio (opcional).
  m) kernel_video: controladores para cámaras USB (obsoleto, eliminado).
  n) kernel_wireless: controladores para adaptadores wifi, usado para experimentación (opcional).

Como puede verse, OpenDomo podría considerarse una distribución GNU/Linux, diseñada para un fin muy específico y adaptada para funcionar con recursos muy limitados.

- **Secuencia de arranque**

Todo el sistema base se encuentra empaquetado dentro de un fichero CPIO en formato “initramfs”, que es cargado en memoria, a continuación del kernel, por el cargador de arranque de la Raspberry Pi. Al finalizar el arranque inicial del kernel sobre el “initramfs”, no se descarga de la memoria y se pivota a un sistema de ficheros ubicado en un dispositivo de almacenamiento externo, sino que este sistema de ficheros raíz cargado en RAM es el sistema de ficheros raíz final. Esto permite, entre otras cosas, que el sistema base se mantenga intacto en cada arranque, ya que las modificaciones realizadas al mismo se pierden en cada apagado al encontrase almacenadas únicamente en la RAM. No sucede lo mismo con las configuraciones no relacionadas con el sistema base, como se verá más adelante.

Para su arranque, OpenDomo se basa en un proceso llamado “init” que se encarga de arrancar el resto del sistema, siguiendo la filosofía de todos los sistemas UNIX y derivados. En este caso particular, el sistema organiza el ejecutable “init”, los scripts de arranque y ficheros de configuración de manera un tanto particular al tratarse de un sistema embebido basado en Busybox, no siguiendo la estructura convencional de los estilos SysV y BSD. Al igual que el resto de herramientas del sistema, el proceso “init” forma parte de Busybox.

El mecanismo utilizado para el arranque consiste en una serie de scripts ubicados en el directorio /etc/init.d/, cuya misión es la de controlar la ejecución de los demonios del sistema durante el inicio. Mediante enlaces simbólicos situados en el directorio /etc/init.d/boot, se determina qué demonios deben ser arrancados y cuáles no. Estos enlaces simbólicos están compuestos por una “S” o “K”, para indicar los demonios a arrancar o parar respectivamente, seguidos de un número de dos dígitos para indicar el orden de invocación (permitiendo establecer dependencias), y finalmente el nombre del script al que enlazan. Los scripts, por su parte, reciben parámetros (“start”, “stop”, “status”...) que determinan la acción a llevar a cabo sobre el demonio que gestionan. Una vez el kernel ha sido cargado en memoria, se ejecuta el script maestro llamado “rcS”, que dispara todo el mecanismo anteriormente indicado.

Los servicios/demonios iniciados en el arranque, así como su orden, pueden verse en el siguiente gráfico:

#	Servicio	Descripción
05	udev	Construye el sistema de archivos en /dev/
10	syslog	Inicia el servicio de registro de eventos
20	urandom	Carga el generador de números aleatorios
25	modstore	Carga los controladores de acceso a disco
30	mount	Monta las unidades locales y carga la configuración local, si existe
35	modnet	Carga los controladores de red necesarios
40	network	Inicia los servicios de red
45	netconf	Carga la configuración de red (si mount ha fallado)
50	netdrives	Monta las unidades de red, si las hay configuradas
55	inetd	Inicia los servicios TCP/IP (servidor web, etc.)
60	inspkg	Descomprime los paquetes instalados
65	ssl	Arranca el servicio SSH
70	modextra	Carga los controladores extra configurados
71	uid	Inicia el servicio de identificación universal
80	oddiscovery	Inicia el servicio de replicación
91	odcontrol	Controladores de dispositivos
95	crond	Inicia el programador de eventos
99	sound	Inicia el sistema de sonido

Ilustración 3.3: Servicios al inicio y orden de arranque en OpenDomo

- **Almacenamiento**

Como se ha comentado anteriormente, el sistema arranca sobre un sistema de ficheros en RAM que compondrá el sistema de ficheros raíz. Este sistema de ficheros, en formato "initramfs", se carga en la memoria desde la tarjeta de SD conectada a la Raspberry Pi, que constituye el principal dispositivo de almacenamiento. Para que esto sea posible, la tarjeta de memoria debe tener unas características en particular que la hagan utilizable por la Raspberry Pi, y por el sistema OpenDomo:

– Su formato debe ser SD, aunque tarjetas en formato miniSD y microSD pueden ser utilizadas mediante un adaptador. Se recomienda utilizar en particular tarjetas SDHC. Existe una lista de tarjetas cuyo funcionamiento con el dispositivo ha sido verificado, y puede consultarse en http://elinux.org/RPi_VerifiedPeripherals#SD_cards. Para el proyecto, se han utilizado dos tarjetas SD: una SanDisk de 2 GB, y una Kingston SDHC de 16 GB.

– Es imprescindible que la tarjeta contenga una partición de arranque con un sistema de ficheros FAT32, que será donde la Raspberry Pi busque los ficheros necesarios para su arranque y funcionamiento (firmware, kernel Linux, ficheros de configuración e initramfs

con OpenDomo).

Los ficheros de configuración que es necesario incluir en esta partición han ido variando a lo largo de la vida del proyecto, por lo que es posible que en el momento de presentar esta documentación la información que se detallará a continuación no esté actualizada. Pueden consultarse en http://elinux.org/RPi_Advanced_Setup.

Los ficheros a incluir son:

a) start.elf → Imagen binaria del firmware para la GPU, suministrada por la fundación.

b) start_cd.elf → Imagen binaria del firmware para la GPU, cuando se asignan 16 MB de RAM a la misma.

c) fixup.dat →Fichero de firmware encargado de la división de la memoria entre la CPU y la GPU, y que complementa al fichero "start.elf".

a) fixup_cd.dat → Fichero de firmware encargado de la división de la memoria entre la CPU y la GPU, y que complementa al fichero "start_cd.elf". Se usa cuando se asignan 16 MB de RAM a la GPU.

d) bootcode.bin → Cargador de arranque de segunda (previa a la activación de la SDRAM) y tercera (posterior a la activación de la SDRAM) etapa.

e) kernel.img → Imagen binaria del kernel Linux compilada para esta arquitectura y dispositivo.

f) init.gz → Initramfs con el sistema de ficheros raíz de OpenDomo. Constituye el sistema OpenDomo en sí mismo, exceptuando los paquetes adicionales y las configuraciones.

g) config.txt → Fichero de configuración que indica a la Raspberry Pi parámetros necesarios para la carga del initramfs, y el reparto de la SDRAM entre la CPU y la GPU. El dispositivo será responsable de cargar en memoria el kernel, a continuación el initramfs, y por último de indicarle la presencia y ubicación del mismo al kernel cuando le transfiera el control. Los parámetros que se indican en este fichero son:

   boot_delay=2 (fuerza esperar 2 segundos a "start.elf" antes de cargar el kernel. Evita algunos problemas cuando se utiliza initramfs)

   initramfs init.gz (indica el nombre del initramfs a cargar. El fichero puede tener como máximo 8 caracteres en su nombre)

   gpu_mem=16 (indica los MB de SDRAM que se asignarán para el funcionamiento de la GPU, en bloques de 16 MB)

   La lista completa de parámetros disponibles puede consultarse en http://elinux.org/RPi_config.txt.

   Tanto los ficheros del firmware, como el kernel, pueden descargarse en https://github.com/raspberrypi/firmware. Es posible clonar el repositorio completo al disco duro mediante el comando "git clone git://github.com/raspberrypi/firmware.git", para lo que es necesario disponer previamente del software "Git".

   Cabe mencionar, que **la imagen binaria del kernel descargada desde la URL anterior no incluye soporte para la carga del initramfs ni el funcionamiento del sonido**, por

tanto, es necesario descargarse las fuentes del kernel, y tras modificar el fichero de configuración para incluir dichas funcionalidades (consultar 3.2.1), compilarlo. Las fuentes del kernel pueden encontrarse en https://github.com/raspberrypi/linux.git (git clone https://github.com/raspberrypi/linux.git), y las instrucciones de compilación en http://elinux.org/RPi_Kernel_Compilation.

Por último, existe una tercera URL donde se pueden obtener algunas herramientas adicionales, si bien no han sido utilizadas a lo largo de este proyecto:

https://github.com/raspberrypi/tools.git

– El sistema OpenDomo necesita una partición sobre la que trabajar, donde almacenará los paquetes a instalar en el sistema durante el arranque, las configuraciones que deben ser persistentes entre reinicios, etc. Esta partición puede ser la misma utilizada por la Raspberry Pi para los ficheros de arranque u otra diferente, y hasta el momento se soportan FAT32, EXT2 y EXT3 como sistemas de ficheros para la misma.

  Para que el sistema tenga en cuenta la partición de configuración y la utilice, se debe incluir en la misma un fichero de texto llamado “opendomo.cfg”. Este fichero incluirá información necesaria para que el sistema determine el uso de dicha partición y su contenido, ya que una partición puede tener diversos usos. Cuando el sistema de arranque detecta una partición, éste intentará montarla y, si lo consigue, buscará en su interior el archivo de configuración “opendomo.cfg”. Si el archivo no existe, se procederá a desmontar la unidad inmediatamente por razones de seguridad, pero si el archivo de configuración existe, aunque esté vacío, la unidad permanecerá montada en “/mnt/<*nombre_de_unidad*>”. Si además, el archivo indica un nombre descriptivo mediante el parámetro “LABEL”, se creará un enlace simbólico en “/media/<*nombre_label*>”, y su contenido será visible desde la interfaz.

  Este archivo permite indicar si se desea que la unidad se utilice para almacenar la configuración, como es el caso, mediante la variable “CONFDEVICE”, o si se desea almacenar archivos de sistema, mediante la variable “SYSDEVICE”. También puede indicarse una ruta donde guardar los archivos de log mediante la variable “LOGDIR”, de modo que permanezcan intactos después de reiniciar el sistema (normalmente los mensajes de log se muestran en una terminal, pero no se almacenan en fichero, ya que ocuparían espacio en la memoria RAM). Por otro lado, mediante el parámetro “SYSBACK”, indicamos al sistema que use esta unidad de apoyo para conservar la configuración en caso de actualizar el sistema principal. Un ejemplo de fichero “opendomo.cfg” sería el siguiente:

```
CONFDEVICE="1"
SYSDEVICE="1"
SYSBACK="1"
LABEL="sistema"
LOGDIR="logs"
```

  En el caso que nos ocupa, donde sencillamente se requiere una partición que el sistema use para almacenar su configuración, bastará con crear en la misma el fichero de texto “opendomo.cfg” con el siguiente contenido:

```
CONFDEVICE="1"
```

A continuación, se deberán crear en la misma los directorios “sysconf” (guardará el archivo de configuración, si existe) y “pkgcache” (almacenará los paquetes a instalar en el arranque).

Hecho esto, al arrancar el sistema la partición será procesada como partición de configuración, quedando finalmente montada en “/mnt/odconf/”.

Si hubiera sido configurada como partición de sistema, debería contener un directorio “libs” (guardaría librerías pesadas, no incluidas en el initramfs por su tamaño), y quedaría montada en “/mnt/system/”.

Resumiendo los dos puntos anteriores, una tarjeta SD con el sistema OpenDomo listo para ser arrancado tendrá una de las siguientes estructuras internas:

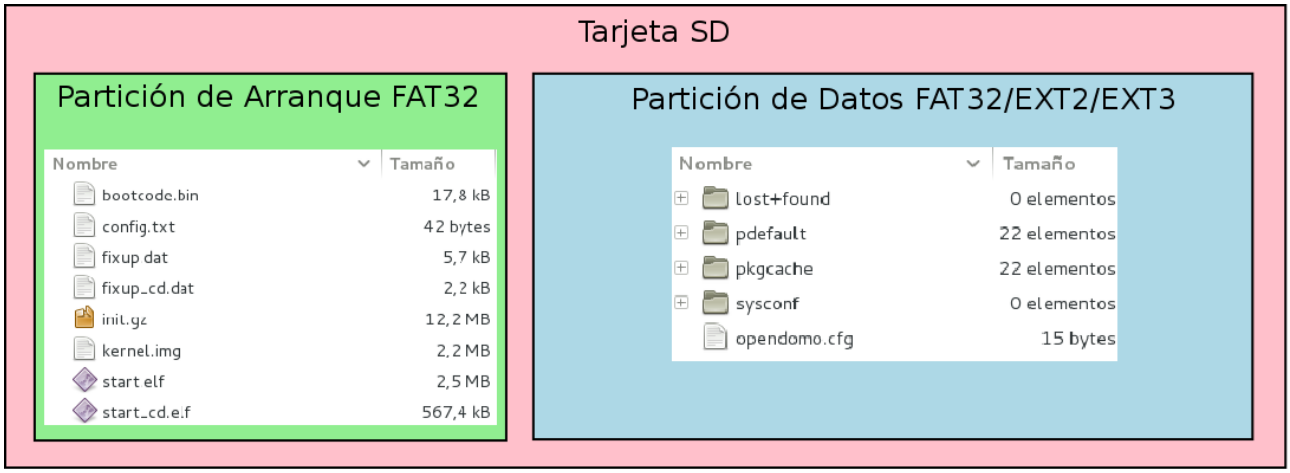


Ilustración 3.4: Estructura interna de una tarjeta SD de OpenDomo basada en dos particiones

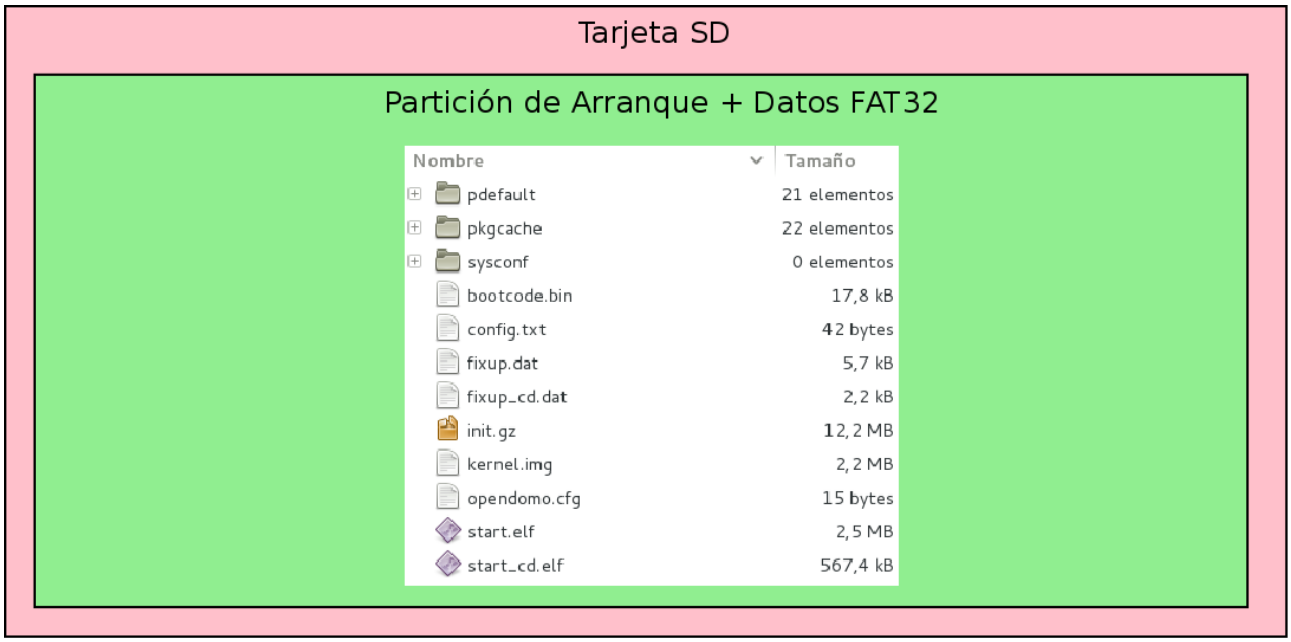


Ilustración 3.5: Estructura interna de una tarjeta SD de OpenDomo basada en una única partición

Aunque se han probado ambos casos, en el proyecto se ha utilizado principalmente, y se recomienda, la estructura de dos particiones.

– Independientemente del reparto de particiones que se haga a la tarjeta, se deben seguir unas normas para que su funcionamiento sea correcto:

a) Se debe simular una geometría de 255 cabezas y 63 sectores, como si de un disco duro se tratase.

b) El número de cilindros a simular será diferente en función de la capacidad de cada tarjeta SD. Se calculará a partir de las cabezas, los sectores, y la capacidad de la tarjeta expresada en bytes, siguiendo la siguiente fórmula:

Cilindros = Bytes_Totales_Tarjeta / 255 / 63 / 512

El resultado de la fórmula siempre debe redondearse a la baja (por ejemplo, un resultado de 299,9 se redondeará a 299). Para llevar a cabo esta tarea se recomienda utilizar la herramienta “fdisk”, y los pasos detallados a seguir pueden encontrarse en http://elinux.org/RPi_Advanced_Setup. Seguidamente se muestra como ejemplo el particionado una tajeta SD de 2 GB, utilizando el esquema recomendado basado en dos particiones:

✔ El primer paso, será determinar el dispositivo que corresponde a la tarjeta. Para ello se recomienda analizar con los comandos “mount” y “fdisk” los discos montados y conectados al sistema. Analizar la salida del comando “dmesg” o el fichero “/var/log/messages” tras conectar la tarjeta al equipo también puede ser de ayuda. En el ejemplo, la tarjeta se encuentra en “/dev/sdc”.

Líneas escritas en “/var/log/messages” al conectar la tarjeta:

```
Jan 19 11:47:05 kernel: [ 9487.025957] sd 12:0:0:0: [sdc] 3842048 512-byte
logical blocks: (1.96 GB/1.83 GiB)
Jan 19 11:47:05 kernel: [ 9487.031490]  sdc: unknown partition table
```

✔ Seguidamente, se ejecuta “fdisk” sobre el dispositivo, y se analizan sus particiones con la opción “p”, eliminándolas con la opción “o” si fuera necesario.

```
# fdisk /dev/sdc


Orden (m para obtener ayuda): p


Disco /dev/sdc: 1967 MB, 1967128576 bytes
61 heads, 62 sectors/track, 1015 cylinders, 3842048 sectores en total
Units = sectores of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Identificador del disco: 0x33897119


Disposit. Inicio    Comienzo      Fin      Bloques  Id  Sistema
```

```
Orden (m para obtener ayuda): o
Building a new DOS disklabel with disk identifier 0xe85b38ee.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Atención: el indicador 0x0000 inválido de la tabla de particiones 4 se
corregirá mediante w(rite)

Orden (m para obtener ayuda): p

Disco /dev/sdc: 1967 MB, 1967128576 bytes
61 heads, 62 sectors/track, 1015 cylinders, 3842048 sectores en total
Units = sectores of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Identificador del disco: 0xe85b38ee

Disposit. Inicio    Comienzo      Fin      Bloques  Id  Sistema
```

- ✔ En caso de usar “o” para eliminar las particiones, se recomienda pulsar “w” para salvar cambios, y reconectar la tarjeta. Si esto no elimina completamente la información, se recomienda usar el comando “dd” para llenar el dispositivo de ceros.

```
# dd if=/dev/zero of=/dev/sdc
dd: escribiendo en «/dev/sdc»: No queda espacio en el dispositivo
3842049+0 registros leídos
3842048+0 registros escritos
1967128576 bytes (2,0 GB) copiados, 511,269 s, 3,8 MB/s
```

- ✔ Se debe anotar el número de bytes totales de la tarjeta, visibles desde “fdisk” con la opción “p”, pues será utilizado más adelante para establecer la geometría. En el ejemplo, la tarjeta tiene un total de 1967128576 bytes.
- ✔ Se selecciona “x” para entrar en el modo experto, y se establece una geometría de 255 cabezas con la opción “h” (heads), y 63 sectores con la opción “s” (sectors).

```
Orden (m para obtener ayuda): x

Orden avanzada (m para obtener ayuda): h
Número de cabezas (1-256, valor predeterminado 61): 255

Orden avanzada (m para obtener ayuda): s
Número de sectores (1-63, valor predeterminado 62): 63
```

- ✔ Se calculan los cilindros a partir de los bytes de la tarjeta (1967128576 en este caso), y la formula anteriormente explicada, y se establecen con la opción “c” (cylinders). El resultado de la fórmula debe ser redondeado siempre a la baja.

  Cilindros = 1967128576 / 255 / 63 / 512 = 239,156427015 = 239

```
Orden avanzada (m para obtener ayuda): c
Número de cilindros (1-1048576, valor predeterminado 1015): 239
```

- ✔ Establecida la geometría, se sale del modo experto con la opción “r”, y se crean dos particiones primarias con la opción “n”. La primera tendrá 500 MB de tamaño (opcional, se recomienda un tamaño mínimo de 100 MB), que se establecerá dejando por defecto el valor del primer sector y marcando “+500M” para el último, será de tipo FAT32, que se establecerá con la opción “t” y el código de partición “c”, y además se marcará como de arranque con la opción “a”. La segunda se dejará con los valores por defecto, ocupando el resto de la tarjeta.

```
Orden avanzada (m para obtener ayuda): r

Orden (m para obtener ayuda): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Número de partición (1-4, valor predeterminado 1): 1
Primer sector (2048-3842047, valor predeterminado 2048):
Se está utilizando el valor predeterminado 2048
Last sector, +sectores or +size{K,M,G} (2048-3842047, valor predeterminado
3842047): +500M

Orden (m para obtener ayuda): t
Se ha seleccionado la partición 1
Código hexadecimal (escriba L para ver los códigos): c
Se ha cambiado el tipo de sistema de la partición 1 por c (W95 FAT32 (LBA))

Orden (m para obtener ayuda): a
Número de partición (1-4): 1
Orden (m para obtener ayuda): p

Disco /dev/sdc: 1967 MB, 1967128576 bytes
255 heads, 63 sectors/track, 239 cylinders, 3842048 sectores en total
Units = sectores of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

```
Identificador del disco: 0xe85b38ee

Disposit. Inicio    Comienzo      Fin      Bloques  Id  Sistema
/dev/sdc1   *        2048     1026047      512000    c  W95 FAT32 (LBA)

Orden (m para obtener ayuda): n
Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p): p
Número de partición (1-4, valor predeterminado 2): 2
Primer sector (1026048-3842047, valor predeterminado 1026048):
Se está utilizando el valor predeterminado 1026048
Last   sector,   +sectores   or   +size{K,M,G}   (1026048-3842047,   valor
predeterminado 3842047):
Se está utilizando el valor predeterminado 3842047
```

✔ Una vez comprobado que el particionado es correcto con la opción “p”, se escribe ta tabla de particiones al dispositivo con la opción “w”.

```
Orden (m para obtener ayuda): p

Disco /dev/sdc: 1967 MB, 1967128576 bytes
255 heads, 63 sectors/track, 239 cylinders, 3842048 sectores en total
Units = sectores of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Identificador del disco: 0xe85b38ee

Disposit. Inicio    Comienzo      Fin      Bloques  Id  Sistema
/dev/sdc1   *        2048     1026047      512000    c  W95 FAT32 (LBA)
/dev/sdc2         1026048     3842047     1408000   83  Linux

Orden (m para obtener ayuda): w
¡Se ha modificado la tabla de particiones!

Llamando a ioctl() para volver a leer la tabla de particiones.

ATENCIÓN: Si ha creado o modificado alguna de las
particiones DOS 6.x, consulte la página man de fdisk
para ver información adicional.
```

```
Se están sincronizando los discos.
```

✔ En este punto, se recomienda desconectar y volver a conectar la tarjeta para que el sistema detecte los cambios, y se pasará a formatear las particiones como sigue, marcando cada una con la etiqueta correspondiente si se desea (en el ejemplo, “boot” para la de arranque y “datos” para la de almacenamiento).

```
# mkfs.msdos -F 32 /dev/sdc1 -n boot
mkfs.msdos 3.0.13 (30 Jun 2012)
# mkfs.ext3 /dev/sdc2 -L datos
mke2fs 1.42.5 (29-Jul-2012)
Etiqueta del sistema de ficheros=datos
OS type: Linux
Tamaño del bloque=4096 (bitácora=2)
Tamaño del fragmento=4096 (bitácora=2)
Stride=0 blocks, Stripe width=0 blocks
88000 inodes, 352000 blocks
17600 blocks (5.00%) reserved for the super user
Primer bloque de datos=0
Número máximo de bloques del sistema de ficheros=360710144
11 bloque de grupos
32768 bloques por grupo, 32768 fragmentos por grupo
8000 nodos-i por grupo
Respaldo del superbloque guardado en los bloques:
       32768, 98304, 163840, 229376, 294912

Allocating group tables: hecho
Escribiendo las tablas de nodos-i: hecho
Creating journal (8192 blocks): hecho
Escribiendo superbloques y la información contable del sistema de ficheros:
hecho
```

Se recomienda emplear dos particiones separadas: una FAT32 para el arranque del dispositivo, y una EXT3 con los ficheros específicos de OpenDomo. Esto permite, además de organizar los diferentes ficheros según su finalidad, gestionar los permisos al estilo UNIX sobre los ficheros que serán utilizados por OpenDomo, si bien esta partición EXT3 no será accesible desde los sistemas Windows.

En caso de seguir el esquema de particionamiento recomendado, basado en dos particiones, OpenDomo no montará automáticamente la partición de arranque FAT32, y el sistema de ficheros tras el arranque quedará de la siguiente forma:

```
# mount
rootfs on / type rootfs (rw)
devtmpfs on /dev type devtmpfs (rw,relatime,size=48040k,nr_inodes=12010,mode=755)
proc on /proc type proc (rw,relatime)
/sys on /sys type sysfs (rw,relatime)
/dev/pts on /dev/pts type devpts (rw,relatime,mode=600)
none on /dev/shm type tmpfs (rw,relatime)
/dev/mmcblk0p2 on /mnt/mmcblk0p2 type ext3 (rw,relatime,user_xattr,barrier=1,nodelalloc,data=ordered)
# pwd
/
# ls -la
total 4
drwxrwxrwx   18 admin    admin            0 Oct 14  2012 .
drwxrwxrwx   18 admin    admin            0 Oct 14  2012 ..
drwxr-xr-x    2 root     root             0 Oct 14  2012 bin
drwxr-xr-x   12 root     root          2860 Jan  1 01:01 dev
drwxr-xr-x   15 admin    admin            0 Jan  1 01:01 etc
drwxr-xr-x    5 root     root             0 Oct 14  2012 home
-rwxr-xr-x    1 root     root           176 Oct 14  2012 init
drwxr-xr-x    5 root     root             0 Jan  1 01:00 lib
lrwxrwxrwx    1 root     root            11 Oct 14  2012 linuxrc -> bin/busybox
drwx------    2 root     root             0 Oct 14  2012 lost+found
drwxr-xr-x    2 admin    admin            0 Oct 14  2012 media
drwxr-xr-x    3 admin    admin            0 Jan  1 01:00 mnt
drwxr-xr-x    2 root     root             0 Oct 14  2012 opt
dr-xr-xr-x   60 root     root             0 Jan  1 01:00 proc
drwx------    2 root     root             0 Dec  6  2012 root
lrwxrwxrwx    1 root     root             3 Oct 14  2012 run -> tmp
drwxr-xr-x    2 root     root             0 Oct 14  2012 sbin
drwxr-xr-x   12 root     root             0 Jan  1 01:00 sys
drwxrwxrwt    4 root     root             0 Jan  1 01:01 tmp
drwxr-xr-x    9 admin    admin            0 Oct  8  2012 usr
drwxr-xr-x    9 admin    admin            0 Oct  8  2012 var
# ls -la /mnt/
total 4
drwxr-xr-x    3 admin    admin            0 Jan  1 01:00 .
drwxrwxrwx   18 admin    admin            0 Oct 14  2012 ..
drwxr-xr-x    6 root     root          4096 Jan  1 01:00 mmcblk0p2
lrwxrwxrwx    1 root     root            14 Jan  1 01:00 odconf -> /mnt/mmcblk0p2
# cd /mnt/odconf
# pwd
/mnt/odconf
# ls -la
total 36
drwxr-xr-x    6 root     root          4096 Jan  1 01:00 .
drwxr-xr-x    3 admin    admin            0 Jan  1 01:00 ..
drwx------    2 root     root         16384 Oct 18  2012 lost+found
-rw-r--r--    1 admin    admin           15 Oct 18  2012 opendomo.cfg
drwxr-xr-x    2 root     root          4096 Jan  1 01:00 pdefault
drwx------    2 admin    admin         4096 Oct 18  2012 pkgcache
drwx------    2 admin    admin         4096 Oct 19  2012 sysconf
```

Ilustración 3.6: Visión general del sistema de ficheros de OpenDomo

- **Red**

Al arrancar, OpenDomo tratará de conectarse a la red automáticamente mediante DHCP. Éste ha sido el sistema empleado durante el desarrollo del proyecto por su simplicidad, ya que se dispone de una LAN con servidor DHCP. Sin embargo, es posible establecer manualmente su configuración TCP/IP e incluso transformarlo en un servidor DHCP, si bien estas configuraciones no han sido probadas, ya que no han sido necesarias en ningún momento.

Para establecer manualmente la configuración de red, puede utilizarse la interfaz web (odcgi) o

manipular con un editor de texto el correspondiente fichero de configuración:

a) Interfaz web: Acudir a "Configure → Network → Network Interface Configuration". Para acceder a la interfaz web, es necesario que el sistema se haya conectado previamente a la red, por ejemplo, por DHCP.

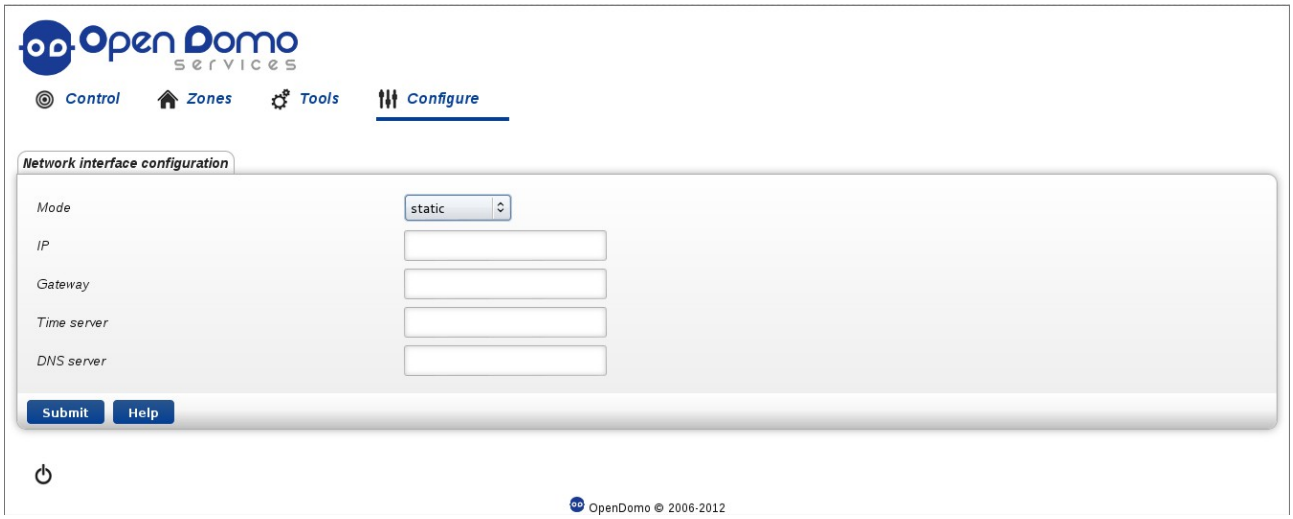


Ilustración 3.7: Configuración de red a través del interfaz web ODCGI

b) Fichero de configuración: Tras acceder a la shell, editar el archivo /etc/opendomo/network.conf (o crearlo si no existe) e indicar la IP y pasarela, como se indica a continuación:

```
IPADDRESS=192.168.0.12
DEFAULTGW=192.168.0.1
NTPSERVER=218.13.10.2
```

El servidor de tiempo (NTPSERVER) es opcional, y requiere disponer del paquete con el cliente NTP para que la hora sea sincronizada de forma efectiva.

- **Interacción con el usuario**

Una vez se encuentra arrancado el sistema sobre el dispositivo, existen tres formas en las que el usuario puede interaccionar con el mismo:

a) Conectando periféricos, es decir, un teclado y un monitor al menos. Este método permite al usuario acceder al sistema a través de una shell y, tras hacer login, trabajar con el mismo, alterando su configuración si fuera necesario. Es el único método viable en caso de no haber sido posible conectar el sistema a la red. Éste método ha sido el utilizado en la fase inicial del proyecto, ya que permite ver en detalle los mensajes de arranque, los errores que se producen, y examinar y probar el sistema en caso de no haber conexión a la red o de no funcionar el servidor SSH o la interfaz web. Se ha utilizado un teclado y ratón inalámbricos USB marca Logitech, que fueron detectados perfectamente por el dispositivo desde el primer momento, y un monitor LCD de 22" con entrada HDMI (ha sido la salida de vídeo del dispositivo utilizada) marca Blusens.

b) Accediendo por SSH desde un dispositivo remoto, como un PC. Resulta obvio que este

método sólo es posible en caso de que el sistema haya podido conectarse a la red. Ofrece las mismas posibilidades que el anterior, ya que a través de la shell es posible ejecutar tareas y llevar a cabo configuraciones. Ha sido el principal método utilizado en el proyecto una vez se disponía de una versión de OpenDomo que era capaz de conectarse a la red y de levantar el demonio SSH.

c) Accediendo a través del navegador web desde un dispositivo remoto, como un PC. Al igual que en el caso anterior, es evidente que el sistema debe estar conectado a la red, y adicionalmente, disponer de un paquete especial llamado “odcgi”. Este paquete despliega una atractiva interfaz web a través de la cual, es posible interaccionar con el dispositivo de forma cómoda y sencilla. Se ha utilizado ocasionalmente a lo largo del proyecto, principalmente para comprobar que el servidor HTTP se encontraba operativo y que el paquete ODCGI era correctamente cargado.

Ilustración 3.8: Página de inició de sesión en interfaz web ODCGI

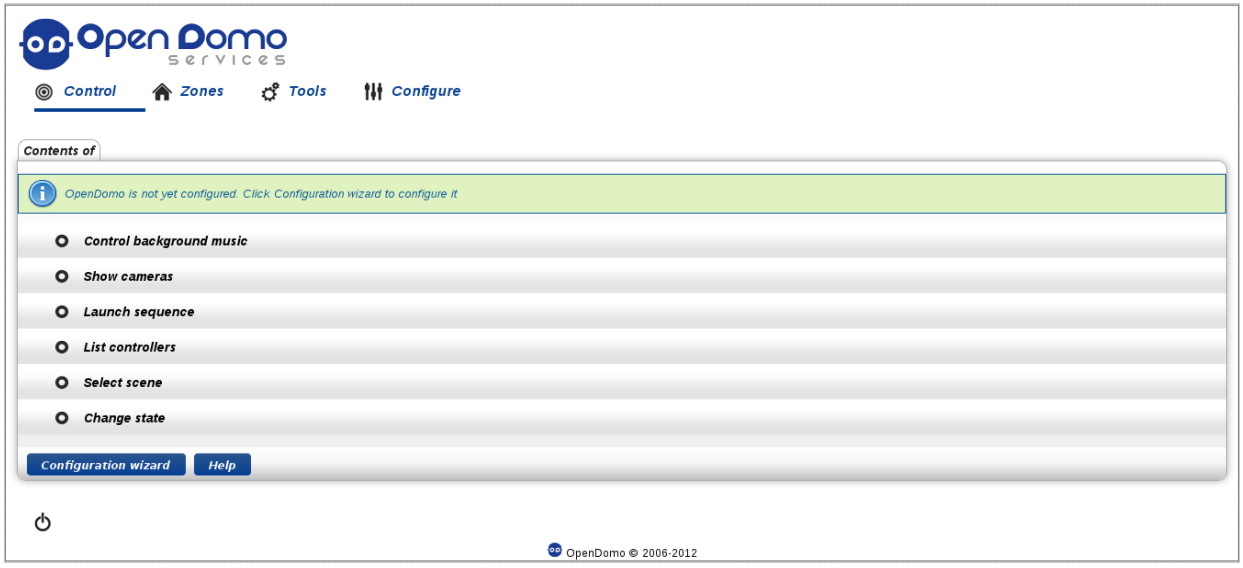


Ilustración 3.9: Página principal de interfaz web ODCGI

El funcionamiento del paquete “odcgi” puede verse a través del siguiente diagrama de flujo:

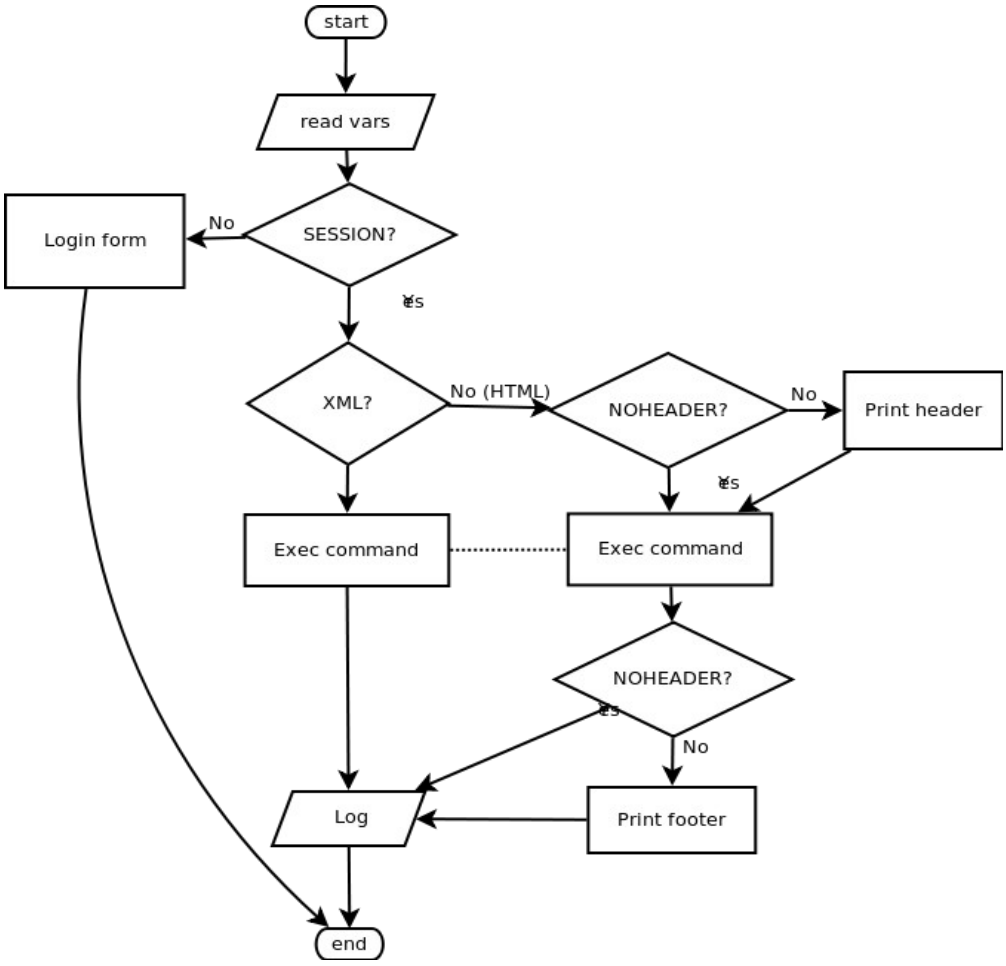


Ilustración 3.10: Diagrama de funcionamiento interno de ODCGI

Dentro del apartado denominado “Exec Command” se ejecutarán los comandos o scripts necesarios para llevar a cabo las tareas indicadas por el usuario.

Por defecto, el sistema dispone de dos usuarios configurados:

- Usuario “admin”, con contraseña “opendomo” (utilizado para conexiones SSH y desde la interfaz web, ya que el usuario “root” está bloqueado por seguridad).
- Usuario “root”, con contraseña “opendomo” (utilizable desde la consola del dispositivo, o en conexiones SSH tras hacer login como “admin” y escalar a “root” con el comando “su -”.

3.1.2 Herramientas utilizadas

Las principales herramientas software que intervienen en el proyecto son:

a) BuildRoot

BuildRoot es una potente herramienta que permite crear desde cero, a partir de ficheros fuente y parches, un sistema Linux embebido completo.

b) OpenDomo SDK:

Consiste en un conjunto de shell scripts que, por un lado, actúan como un frontend para trabajar con BuildRoot, y por otro, permiten utilizar los sistemas embebidos Linux generados por éste para generar el sistema OpenDomo. Durante este proceso de transformación, se incorporan al sistema embebido diferentes paquetes desarrollados por OpenDomo Services S.L. que constituyen el corazón del sistema OpenDomo, junto con algunas configuraciones adicionales.

3.1.2.1 BuildRoot

Ilustración 3.11: Logo de BuildRoot

BuildRoot está formado por un conjunto de ficheros de configuración para la compilación de aplicaciones de código abierto a partir de su código fuente (Makefiles) y parches, y permite generar sistemas Linux embebidos completos de forma fácil, cómoda y automatizada. Es capaz de generar cualquier componente de una cadena de herramientas de compilación cruzada (toolchain), o todas ellas, el sistema de ficheros raíz para el sistema embebido (root filesystem), una imagen del kernel Linux e incluso una imagen del cargador de arranque. Soporta la generación de sistemas embebidos para múltiples arquitecturas (X86, ARM, MIPS, PowerPC, etc.), y está protegido bajo la licencia GPLv2.

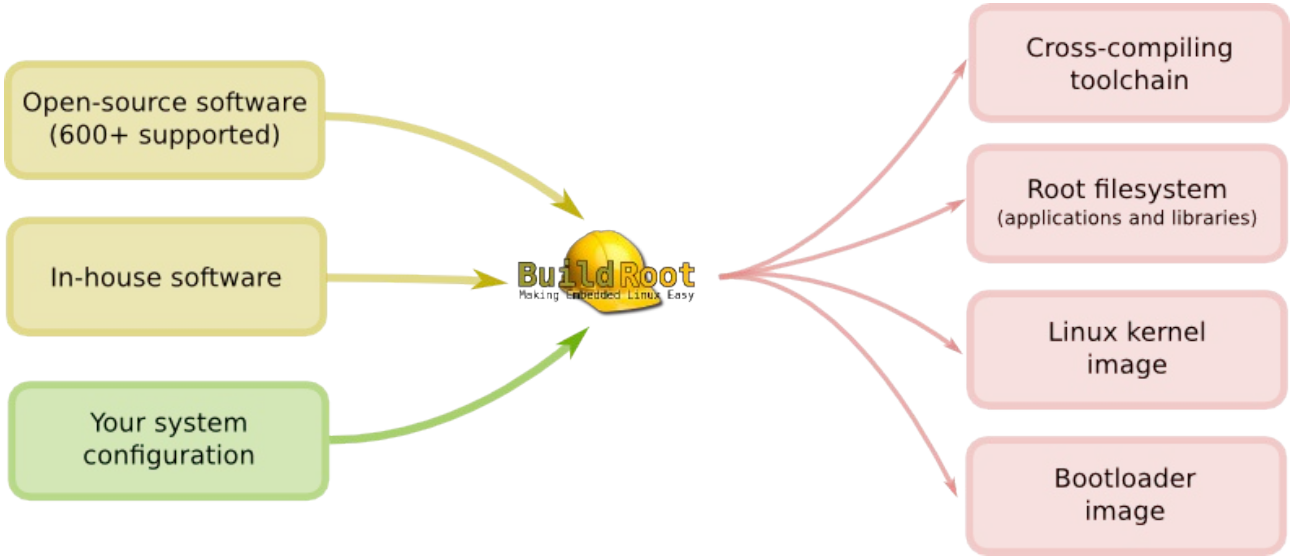


Ilustración 3.12: Esquema de entradas y salidas de BuildRoot

Las principales características de BuildRoot son:

- Puede gestionar todos los componentes de un proyecto de sistema embebido: la cadena de herramientas de compilación cruzada, generación del sistema de ficheros raíz, compilación del kernel y el cargador de arranque. Puede ser utilizado para todas estas tareas, o sólo algunas de ellas.
- Facilidad de configuración, gracias a sus interfaces menuconfig, gconfig y xconfig, que resultan ya conocidas y familiares para todos los desarrolladores y usuarios de los sistemas GNU/Linux.
- Soporta cientos de paquetes de aplicaciones de usuario y librerías.
- Soporta múltiples tipos de sistemas de ficheros para el sistema de ficheros raíz: JFFS2, UBIFS, tarballs, romfs, cramfs, squashfs...
- Puede generar una cadena de herramientas de compilación cruzada basadas en uClibc, o reutilizar una cadena ya existente basada en esta u otras librerías.
- Dispone de una estructura simple, sencilla de comprender y ampliar. Se basa en los sobradamente probados y conocidos Makefiles.

Una cadena de herramientas de compilación cruzada (toolchain), es un conjunto de herramientas software que son utilizadas en cadena, de manera que la salida de cada una de ellas es utilizada como entrada para la siguiente, hasta obtener el resultado deseado. Su objetivo es ejecutarse sobre una máquina con arquitectura A, para generar a partir de su código fuente, aplicaciones destinadas a ejecutarse sobre una máquina con arquitectura B. Generalmente, una cadena de herramientas de compilación cruzada se compone de un compilador, un enlazador, un ensamblador, librerías, y un depurador. En el caso particular del proyecto que nos ocupa, la arquitectura A será X86 o x86_64, y la arquitectura B sera ARM.

Centrándose en las herramientas del proyecto GNU, una completa cadena de herramientas de compilación cruzada incluiría a las siguientes:

- GNU make, para la automatización de la compilación.
- GCC (GNU Compiler Collection), con la colección de compiladores para diversos lenguajes.
- GNU Binutils, que incluye el enlazador y el ensamblador entre otras herramientas.
- GDB (GNU Debugger), para la depuración interactiva.
- GNU Build System (autotools), que incluye herramientas generadoras de makefiles.
- Librerías necesarias, como Glibc (Librería C de GNU) y/u otras,

En el caso que nos ocupa, usando BuildRoot se generará desde cero la cadena de herramientas de compilación cruzada a utilizar, que consistirá en:

- GCC
- GNU Binutils
- Librería uClibc, que será descrita a continuación en este mismo apartado.

Las principales herramientas utilizadas por BuildRoot para la generación de los sistemas Linux embebidos son las siguientes:

1. **Librería uClibc**

Ilustración 3.13: Logo de uClibc

Se trata de una versión reducida de la librería estándar de C, diseñada para ser utilizada en sistemas Linux embebidos, y protegido bajo licencia LGPL.

Es mucho más pequeña que Glibc, que es la librería de C utilizada normalmente en las distribuciones GNU/Linux. Esto es posible debido a que Glibc está preparada para implementar completamente todos los estándares de C a través de una amplia gama de plataformas hardware, mientras que uClibc se especializa en sistemas embebidos, siendo posible además habilitar o deshabilitar algunas de sus características en función del espacio disponible.

uClibc se emplea en sistemas Linux estándar o sin MMU, y soporta las arquitecturas i386, amd64, ARM (big/little endian), PowerPC, SuperH (big/little endian), SPARC y V850. Aunque ha sido escrita desde cero, incorpora parte de código de Glibc.

2. **Herramientas Busybox**

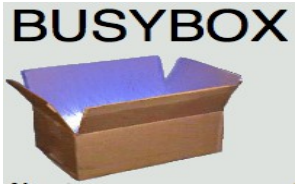


Ilustración 3.14: Logo de Busybox

Busybox es una herramienta que combina versiones simplificadas de muchas utilidades típicas en los sistemas UNIX en un único ejecutable de pequeño tamaño. Estas versiones simplificadas disponen de menos opciones y funcionalidad que los originales, pertenecientes en su mayor parte al proyecto GNU, sin embargo, ofrecen las opciones fundamentales y su comportamiento es muy similar. Gracias a su reducido tamaño, es posible combinarlos para obtener un entorno de trabajo completo para cualquier sistema embebido.

Busybox ha sido desarrollado bajo licencia GPLv2, centrándose principalmente en la optimización de tamaño y el consumo de recursos. Es extremadamente modular, siendo posible incluir o excluir utilidades y características en tiempo de compilación, lo que permite personalizar los sistemas embebidos de forma sencilla.

También dispone de un ejecutable “init” personalizado y simplificado, que entre otras cosas no

soporta runlevels. Este “init” se encuentra incluido en el ejecutable general que compone Busybox, junto con el resto de utilidades del sistema.

Para que Busybox funcione, es necesario tener un sistema de ficheros raíz adaptado, de manera que en la ubicación normal de las utilidades del sistema (comandos básicos como ls, mv, cp, grep, etc.), en lugar de los ejecutables correspondientes a cada una, se disponga en su lugar de enlaces simbólicos apuntando al ejecutable de Busybox (/bin/busybox). Este ejecutable debe disponer de permisos especiales para su correcto funcionamiento: el bit “set user ID” debe estar habilitado. Esto causa que cualquier usuario o programa que ejecute el fichero, acceda al sistema con los mismos privilegios que el propietario del mismo, en lugar de con los suyos propios.

A continuación, se muestra un ejemplo de ejecutable Busybox, visto a través del comando “ls”, donde se observa su tamaño y sus permisos:

```
$ ls -lah busybox
-rwsr-xr-x    1 root     root      676.3K Jan  2  2013 busybox
```

3. **Kernel Linux**

Ilustración 3.15: Logo de Linux

Linux es un núcleo libre de sistema operativo basado en Unix, licenciado bajo GPLv2, y desarrollado por colaboradores de todo el mundo, entre los que se encuentran desde particulares hasta empresas.

Actualmente se trata de un núcleo monolítico híbrido, donde a diferencia de los núcleos monolíticos tradicionales, los controladores de dispositivos y extensiones al núcleo pueden ser cargadas y descargadas fácilmente como módulos sin interrumpir el funcionamiento del sistema. También se diferencia de los núcleos monolíticos tradicionales en que los controladores pueden ser detenidos momentáneamente por actividades más importantes bajo ciertas condiciones, lo que permite gestionar adecuadamente interrupciones hardware y mejorar el soporte para multiprocesamiento simétrico.

Gracias a que desde sus inicios se diseñó para ser un núcleo portable, actualmente es uno de los núcleos más ampliamente portados, siendo capaz de funcionar sobre dispositivos muy diversos. Las principales arquitecturas soportadas son DEC Alpha, ARM, AVR32, Blackfin, ETRAX CRIS, FR-V, H8, IA64, M32R, m68k, MicroBlaze, MIPS, MN10300, PA-RISC, PowerPC, System/390, SuperH, SPARC, x86, x86 64 y Xtensa.

3.1.2.2 OpenDomo SDK

OpenDomo SDK (Software Development Kit) es un entorno de desarrollo pensado para facilitar las tareas de desarrollo del sistema OpenDomo, y se basa en un conjunto de shell scripts que automatizan las principales etapas del proceso de creación del sistema y sus componentes. Entre otras cosas, actúa como un frontend de BuildRoot, descargándolo en interaccionando con el mismo,

proporcionándole los parámetros y la configuración de entrada, y procesando posteriormente su salida.

Los principales componentes de OpenDomo SDK son:

a) Shell scripts para automatización de tareas. Aunque dispone de varios más, los principales scripts utilizados a lo largo del desarrollo del proyecto son:

   1. odsdk.sh: es el script principal, que constituye el punto de entrada al SDK. Permite descargar y descomprimir la versión de BuildRoot a utilizar, actualizar el SDK con la última versión del repositorio, lanzar la interfaz de configuración de BuildRoot y el kernel Linux, instalar los paquetes del sistema operativo necesarios para que la generación del sistema OpenDomo pueda ejecutarse correctamente, compilar los diferentes paquetes específicos del sistema OpenDomo, lanzar el proceso de generación del sistema OpenDomo, etc. Estas acciones no son ejecutadas directamente por el script, sino que se realizan a través de llamadas a scripts secundarios o aplicaciones.
   2. process_initrd.sh: se trata de un script secundario que procesa el initrd en formato EXT2 generado por BuildRoot, con el sistema embebido básico, y lo transforma en el initrd que constituye el sistema OpenDomo, eliminando todos los componentes no necesarios para reducir su tamaño al mínimo, e incorporando los componentes y configuraciones específicos del sistema (usuarios y contraseñas, grupos, scripts de inicio, ficheros de configuración de los servicios, etc.).
   3. create_iso.sh: se trata de un script secundario que permite generar un fichero ISO con una imagen de CD arrancable que incluye el sistema OpenDomo. Se trata de una herramienta que empaqueta el sistema OpenDomo para generar un Live CD del estilo a los existentes para muchas distribuciones GNU/Linux actuales. De esta forma, es posible probar el sistema OpenDomo en cualquier PC o máquina virtual (si ha sido compilado para la arquitectura x86).

b) Código fuente y makefiles de todos los paquetes específicos del sistema OpenDomo.

c) Ficheros de configuración, directorios y scripts que compondrán el sistema OpenDomo a generar. Como se ha mencionado anteriormente, se incluyen usuarios del sistema y contraseñas, grupos, scripts de inicio, ficheros de configuración de los servicios, etc.

d) Ficheros de configuración que deberá emplear BuildRoot para la compilación de los principales elementos del sistema y la generación del sistema de ficheros raíz, que posteriormente será modificado por el SDK para excluir e incluir ficheros y directorios. Estos ficheros permiten que el sistema generado funcione sobre cada dispositivo en particular, por lo que cada dispositivo requerirá diferentes ficheros de configuración, que son:

   1. Fichero de configuración general de BuildRoot, indicando parámetros como la arquitectura del procesador, versiones de los componentes (uClibc, Busybox, Linux, GCC, Binutils, etc.), orígenes para la descarga de los ficheros fuente del kernel Linux, formato del sistema de ficheros raíz a generar, herramientas y librerías a incluir, cadena de herramientas de compilación cruzada a utilizar, ficheros de configuración para la compilación de los principales componentes (uClibc, Busybox y Linux), etc.
   2. Fichero para la configuración de compilación de uClibc, indicando parámetros como la

arquitectura del procesador y características, formato de los ejecutables, configuraciones y funciones a habilitar en la librería, etc.

3. Fichero para la configuración de compilación de Busybox, indicando la plataforma sobre la que se ejecutará, personalización de sus librerías, así como las utilidades que debe incluir y configuración de cada una de ellas (desde editores de texto a herramientas de bajo nivel).
4. Fichero para la configuración de compilación del núcleo Linux, indicando la arquitectura sobre la que se ejecutará junto con numerosos parámetros hardware de bajo nivel, soporte que incluirá el kernel, módulos dinámicos que se generarán, y un largo etcétera.

3.1.3 Especificación de estándares, normas de diseño y construcción

Se han utilizado los diferentes formatos incluidos en el estándar ODF (OpenDocument Format), tanto para documentos susceptibles de sufrir modificaciones y/o ampliaciones, como para plantillas si existen. El software empleado para la manipulación de estos ficheros ha sido LibreOffice.

Los documentos finalizados, serán duplicados a formato PDF a efectos de comunicación o publicación, impidiendo su modificación en la medida de lo posible.

En el caso de los diagramas y esquemas, se ha utilizado la notación UML. Se ha empleado el software DIA para la realización de estos diagramas, así como el software Planner (aplicación de gestión de proyectos para el escritorio GNOME) para la realización de diagramas de Gantt. En caso de no disponer del software Planner, o de que éste no ofrezca las funcionalidades o capacidades requeridas, se recurrirá a la utilización del software ProjectLibre. El reemplazo para el software DIA en caso de no ofrecer éste las funcionalidades deseadas, será LibreOffice Draw.

3.2 Implementación

3.2.1 Metodología de trabajo

Para obtener una versión del sistema OpenDomo que corriera sobre el dispositivo Raspberry Pi, ha sido necesario llevar a cabo una serie de tareas, algunas de ellas basadas en un continuo ciclo de prueba y error hasta encontrar la solución adecuada, dado que no existía documentación exacta sobre la realización correcta de las mismas. Las tareas realizadas, así como el orden en que se han llevado a cabo, pueden resumirse de forma aproximada como sigue:

1. **Configuración de BuildRoot y herramientas**

Conocido el funcionamiento del SDK, BuildRoot, y las interacciones entre ambos, se procedió a configurar BuildRoot y sus herramientas para generar el port de OpenDomo para Raspberry Pi. El primer paso consistió en estudiar el dispositivo a nivel físico, para determinar la arquitectura de su CPU, modelo y características.

Una vez recopilada la información, se analizaron los ficheros de configuración utilizados para generar el sistema OpenDomo para ODNetwork, con arquitectura x86, y se estableció una lista de cambios a realizar, que afectaban a los ficheros de configuración de BuildRoot, Busybox, uClibc y

el kernel Linux:

- BuildRoot:

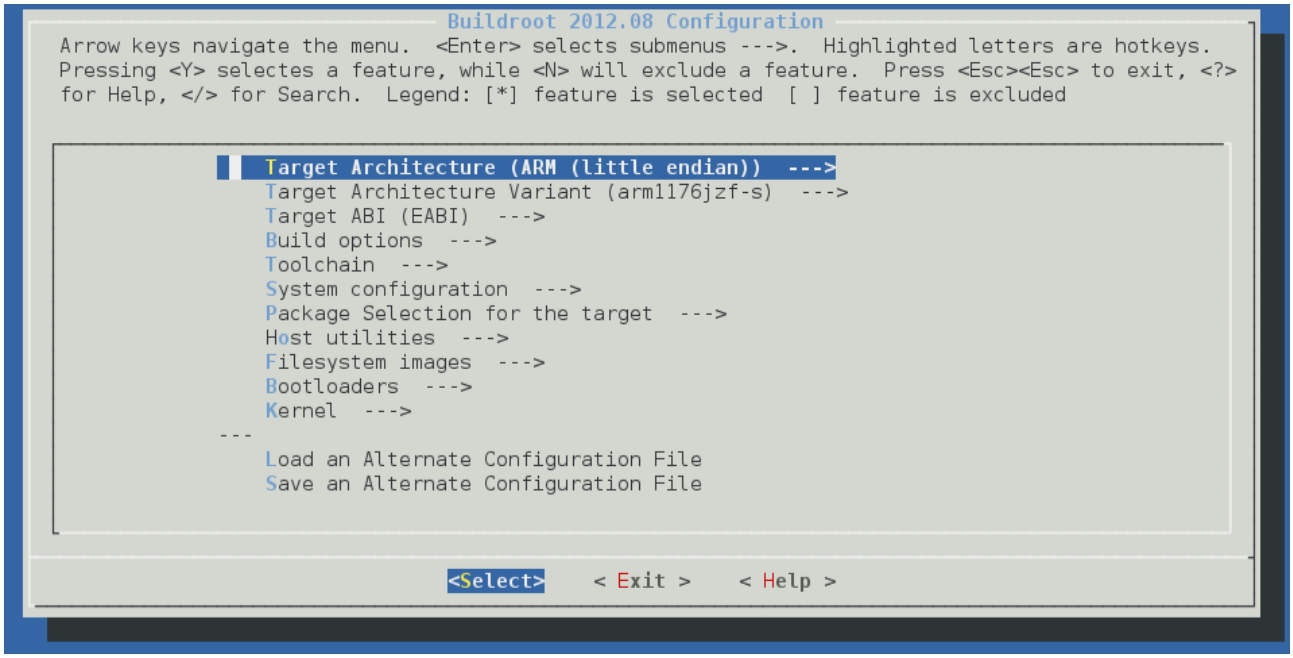


Ilustración 3.16: Utilidad de configuración de BuildRoot

Los cambios principales afectaron a la arquitectura y variante, sistema de ficheros raíz, así como a las versiones de los componentes utilizados:

- Kernel headers 3.2.x
- uClibc 0.9.33.x
- Binutils 2.22
- GCC 4.7.x
- Busybox 1.20.x
- Kernel Linux 3.2.x (actualmente 3.2.27). Además del cambio de versión, se ha pasado de la rama oficial a la rama adaptada específicamente para el dispositivo Raspberry Pi, con las modificaciones y los parches necesarios ya aplicados, y disponible en el repositorio git https://github.com/raspberrypi/linux.git.

- uClibc:

  Tras el cambio de versión, se procedió a la descarga de las nuevas fuentes, y se realizó la configuración a través de la interfaz propia del software, invocada con el comando “make menuconfig”.

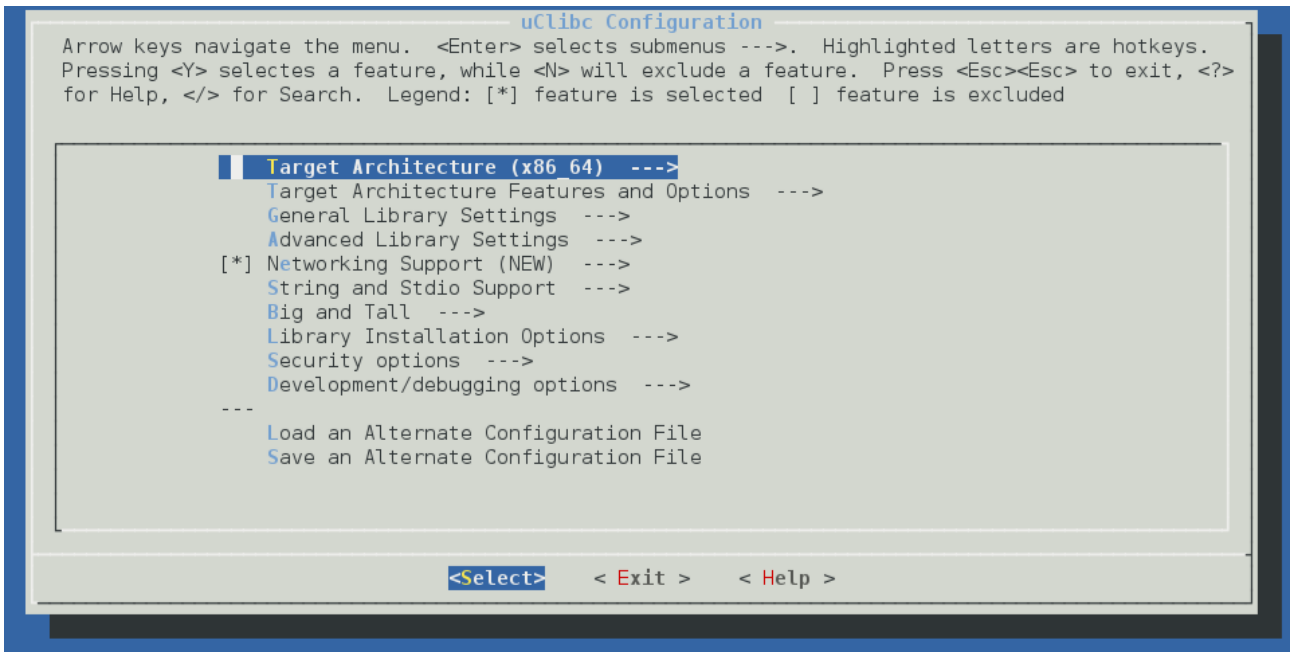


Ilustración 3.17: Utilidad de configuración de uClibc

Para ayudar a la correcta configuración, se analizaron previamente ficheros ya existentes, como el empleado en la generación del sistema para ODNetwork, u otros ejemplos encontrados en los foros Raspberry Pi. El nuevo fichero de configuración fue guardado y proporcionado a BuildRoot a través de su configuración para que fuese usado en las sucesivas compilaciones de la librería. Las principales modificaciones afectaron a la arquitectura, así como a sus características y opciones.

- Busybox:

  Al igual que con la librería uClibc, tras el cambio de versión se descargaron las nuevas fuentes y se configuró la herramienta a través de su interfaz de configuración invocada con el comando “make menuconfig”.

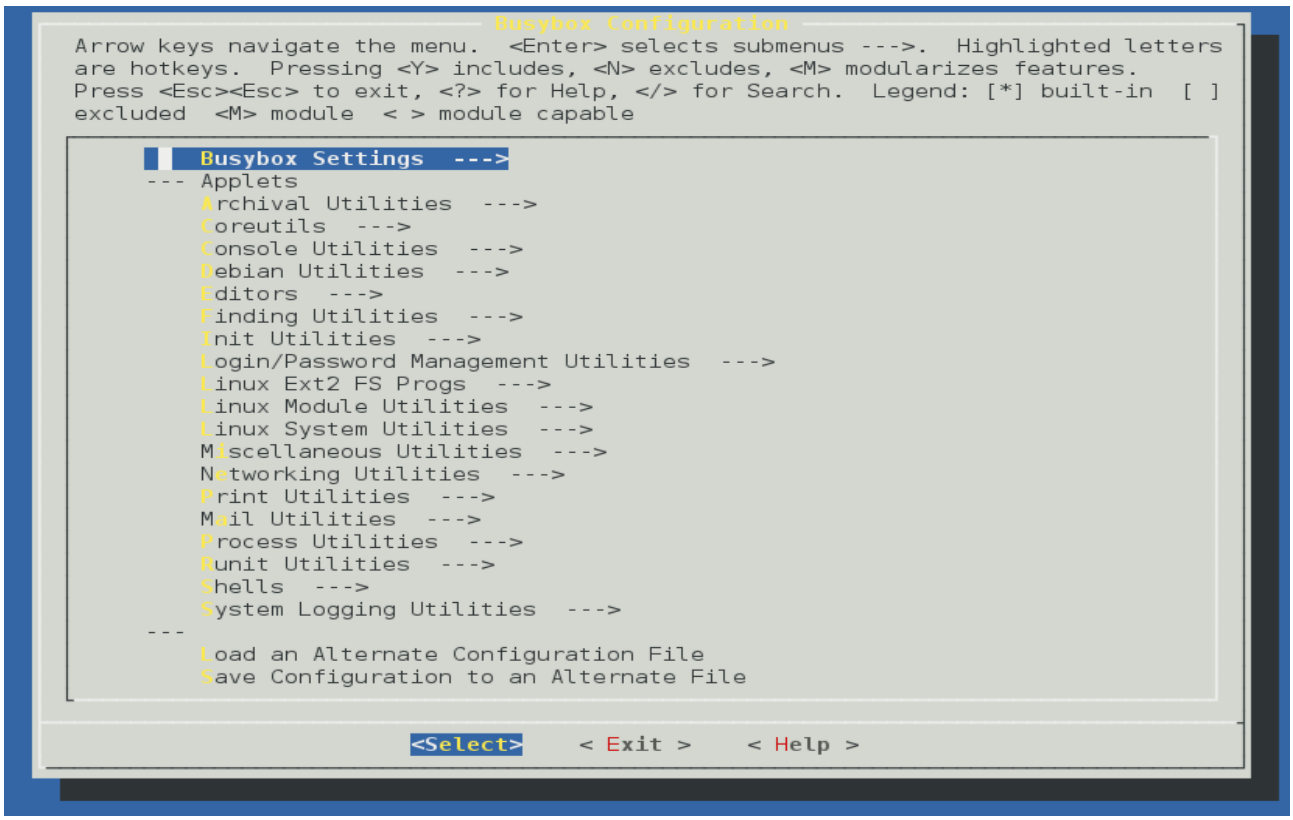


Ilustración 3.18: Utilidad de configuración de Busybox

Tras realizar las modificaciones pertinentes, el fichero fue guardado y proporcionado a BuildRoot a través de su configuración, para que fuera utilizado en las sucesivas compilaciones de Busybox.

- Kernel Linux:

  El proceso seguido con el fichero de configuración de compilación para el kernel Linux fue similar al seguido por Busybox y uClibc, pero dada la extensión de este fichero y la multitud de parámetros, algunos de ellos con una finalidad desconocida, se decidió emplear como base un fichero de configuración ya creado y adaptado para la Raspberry Pi. Este fichero se obtuvo descargando las fuentes del kernel para Raspberry Pi del repositorio git https://github.com/raspberrypi/linux.git, dentro del cual, existe un directorio con configuraciones del kernel predefinidas para diferentes dispositivos. Concretamente, para Raspberry Pi existen los siguientes ficheros:

  – bcmrpi_cutdown_defconfig → Configuración reducida.

  – bcmrpi_defconfig → Configuración estándar.

  – bcmrpi_emergency_defconfig → Configuración mínima para arranque de shell de emergencia.

  De estos tres ficheros, se utilizó como base “bcmrpi_cutdown_defconfig”, con la configuración reducida, ya que el objetivo era que la imagen del kernel fuera pequeña por

cuestiones de rendimiento y ocupación de memoria. De no haber obtenido con esta configuración todas las funcionalidades requeridas, hubiese sido necesario comenzar a utilizar el fichero “bcmrpi_defconfig”, pero finamente no fue necesario.

No obstante, este fichero no habilitaba en el kernel una característica indispensable para el funcionamiento del sistema OpenDomo. Este sistema, se empaqueta en forma de un disco en RAM comprimido con Gzip (initrd para ODNetwork, e initramfs para Raspberry), de manera que el kernel debe ser capaz de arrancar desde un disco en RAM de estas características, utilizándolo como sistema de ficheros raíz (root filesystem). Para ello, se deben modificar dos variables del fichero de configuración, que dotan al kernel compilado esta característica:

```
CONFIG_BLK_DEV_INITRD=y
CONFIG_RD_GZIP=y
```

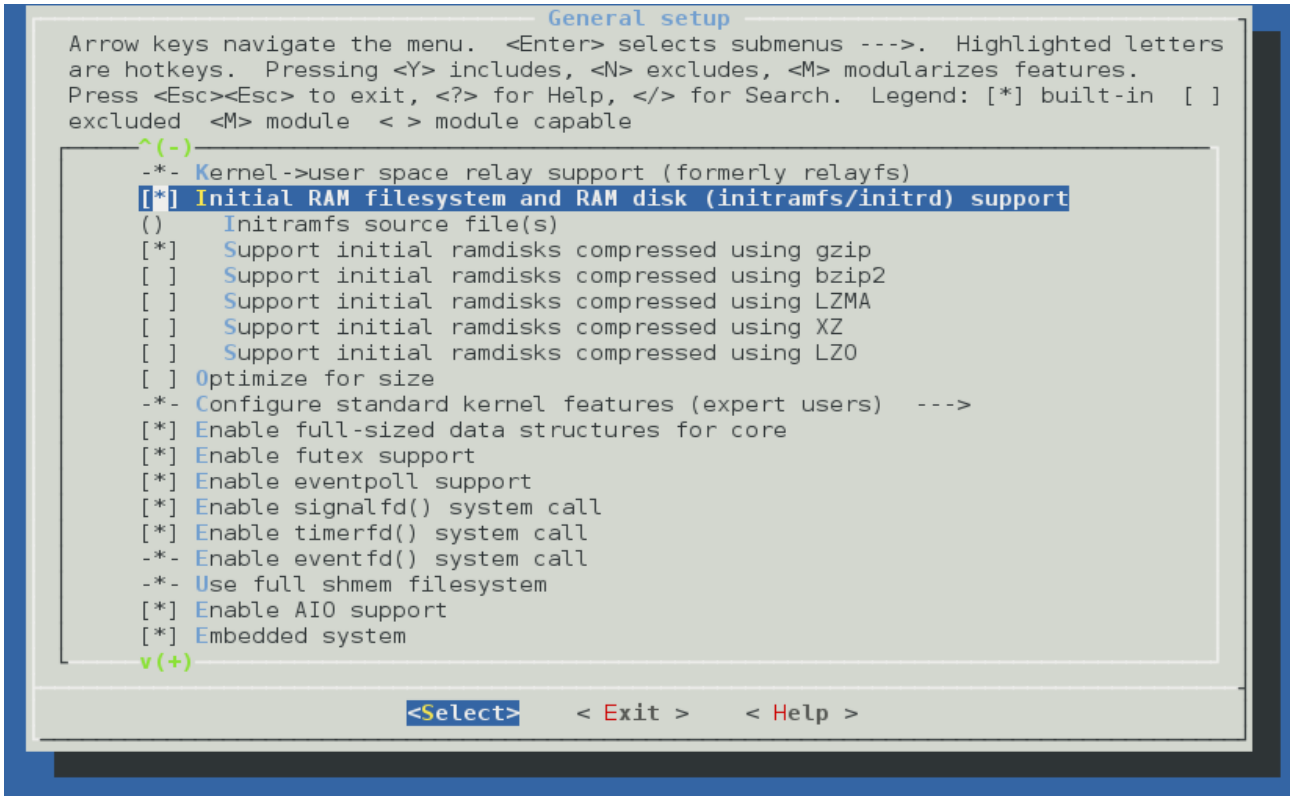


Ilustración 3.19: Utilidad de configuración del kernel Linux, apartado general

Por otro lado, también surgieron problemas con el sonido. El módulo necesario, llamado “snd_bcm2835”, se compilaba como módulo dinámico, pero no se cargaba automáticamente durante el arranque del sistema. Su posterior carga manual también planteaba problemas debido a las dependencias del mismo con otros módulos, por lo que finalmente se decidió incluir este módulo y sus dependencias en el kernel.

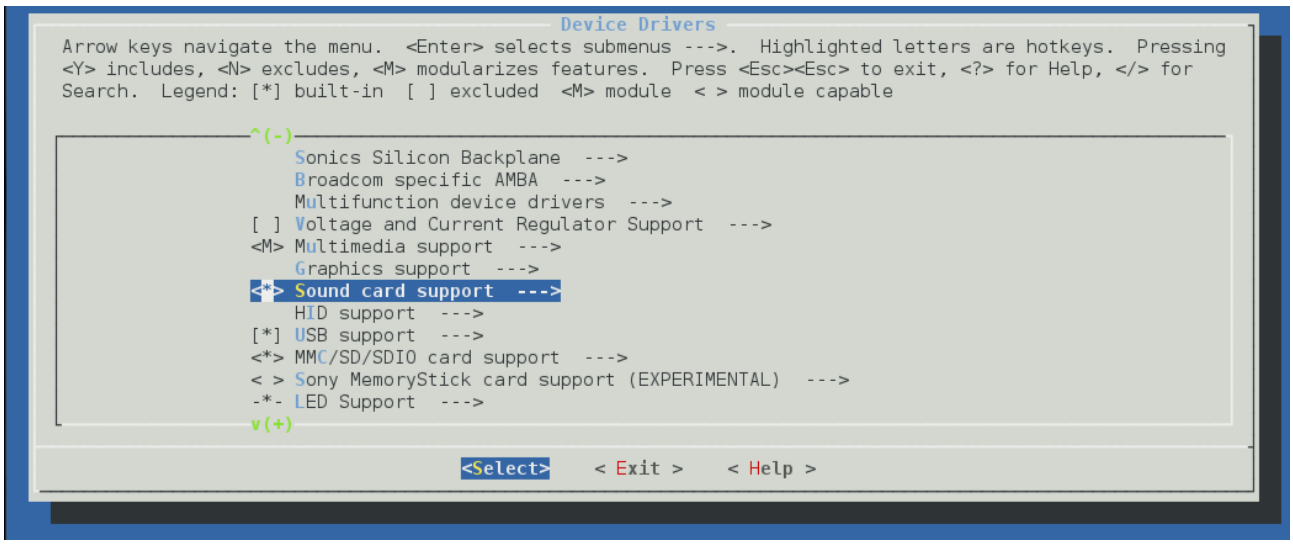


Ilustración 3.20: Utilidad de configuración del kernel Linux, apartado de controladores

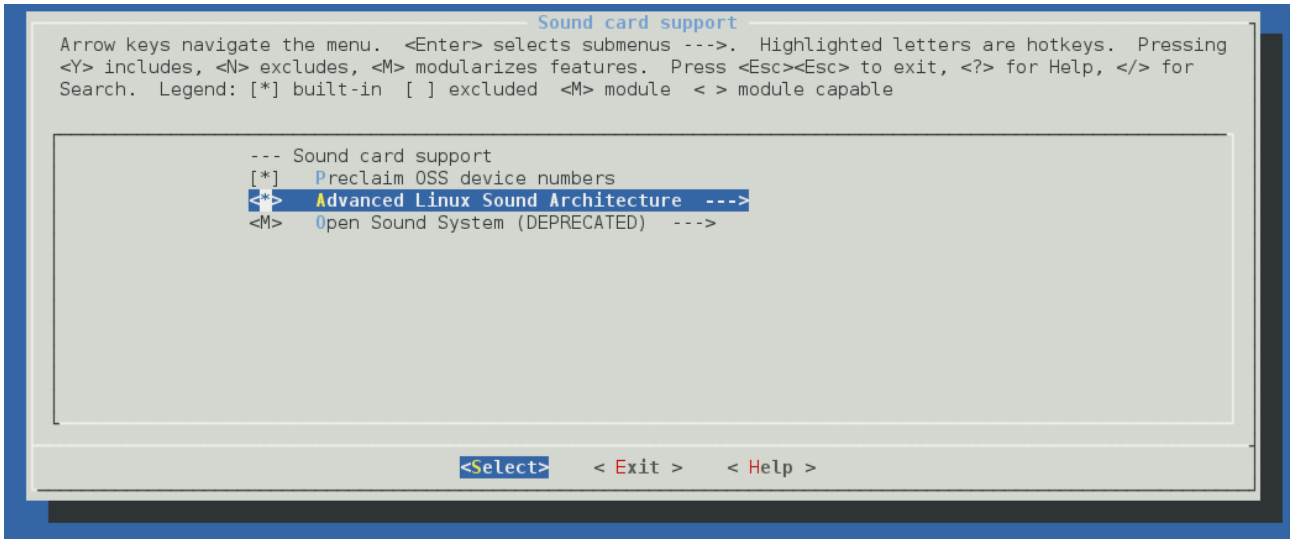


Ilustración 3.21: Utilidad de configuración del kernel Linux, apartado de controladores de sonido

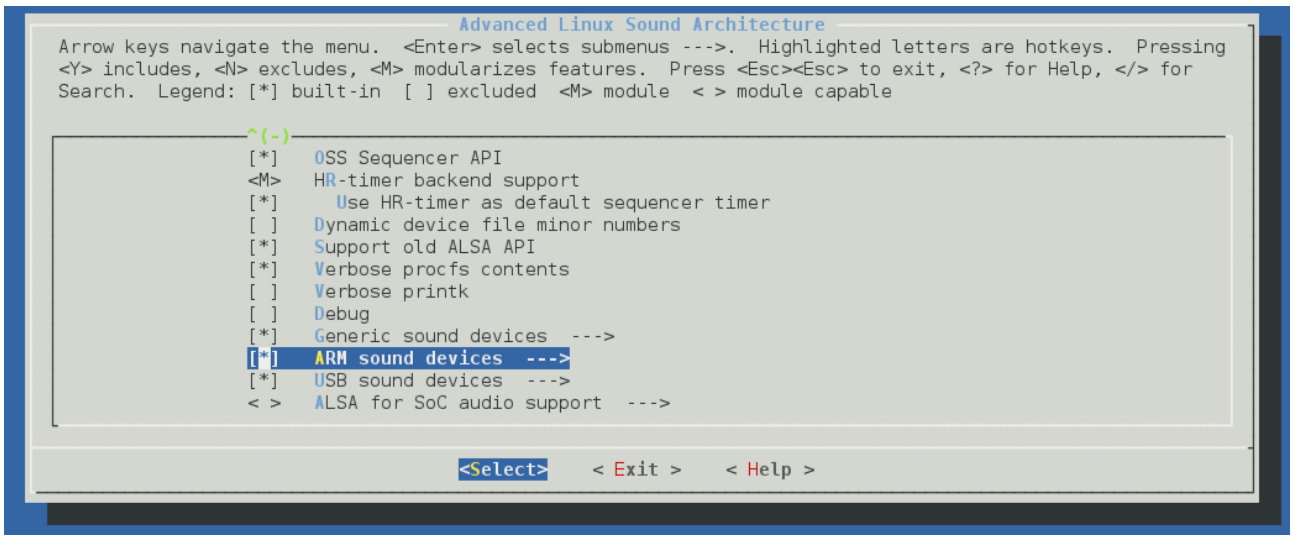


Ilustración 3.22: Utilidad de configuración del kernel Linux, apartado de ALSA

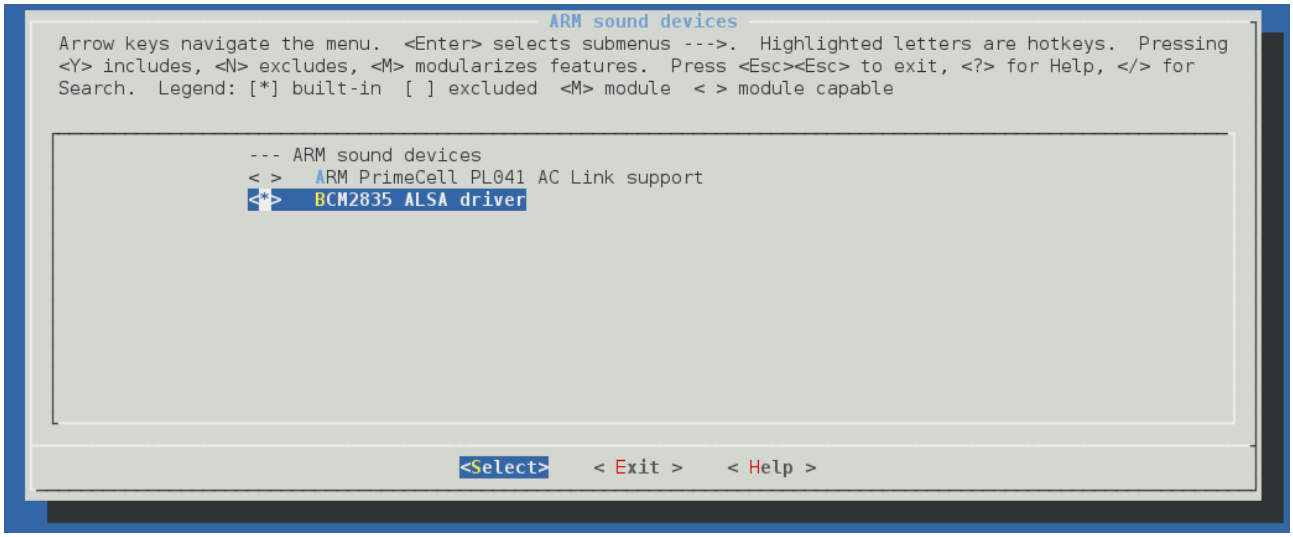


Ilustración 3.23: Utilidad de configuración del kernel Linux, apartado de controladores ALSA para dispositivos ARM

Tras modificar el fichero de configuración de compilación del núcleo Linux, fue guardado y proporcionado a BuildRoot a través de su configuración, para ser utilizado en las sucesivas compilaciones del kernel.

La dificultad de esta fase no solo radicaba en la correcta configuración de cada fichero para que los binarios resultantes fueran funcionales, sino que no todas las combinaciones de variables de cada fichero de configuración conseguían una compilación satisfactoria de la herramienta asociada al mismo, y además, no todas las combinaciones de variables de los diferentes ficheros conseguían la consecución completa del proceso de compilación del sistema. Estas herramientas se encuentran interrelacionadas, de manera que, por ejemplo, la desactivación de una característica en la

configuración de compilación de uClibc, puede hacer que esta librería se compile satisfactoriamente, pero que el proceso de compilación posterior de Busybox falle, en función de su configuración, por la ausencia de dicha característica de uClibc. Por este motivo, fue necesario realizar infinidad de pruebas de compilación, analizando tanto los errores obtenidos como el punto del proceso de compilación del sistema en el que se producían, para ir ajustando los diferentes ficheros de configuración hasta que todo el procedimiento se completó con éxito. En este punto, cabe mencionar la ayuda obtenida a través de la lista de correo de BuildRoot, cuyo histórico puede consultarse en http://buildroot-busybox.2317881.n4.nabble.com/.

Los ficheros de configuración generados, han sido incorporados a la nueva versión del SDK, en el directorio "opendomo/opendomo-distro/arch/arm/", y son:

a) buildroot-config
b) busybox-config
c) linux-config
d) uclibc-config

2. Arranque del port OpenDomo sobre la Raspberry Pi

Una vez obtenido el kernel y el initramfs con el sistema OpenDomo para su ejecución sobre la Raspberry Pi, mediante el uso combinado de BuildRoot con el SDK, llegó el momento de conseguir el arranque del sistema en el dispositivo. El particionado a realizar en la tarjeta SD, los sistemas de ficheros a utilizar, y los archivos binarios con el firmware necesario estaban claros, sin embargo, en este punto no se tenía la certeza de que el dispositivo soportase la utilización de estos discos en RAM. El cargador de arranque a utilizar se suministra en formato binario como parte de los ficheros de firmware, y es éste el responsable de cargar en la memoria tanto el kernel como el disco en RAM, y lanzar posteriormente la ejecución del kernel pasándole la dirección de memoria donde comienza el disco en RAM. Si esta funcionalidad no hubiera estado soportada por el firmware, hubiera sido necesario un cambio de enfoque para el empaquetamiento y la carga del sistema OpenDomo en el dispositivo, pero por suerte, no fue necesario.

Los puntos a tener en cuenta para la correcta carga del disco en RAM por el dispositivo son:

- El formato para el disco en RAM es initramfs, en lugar de initrd.
- El nombre del fichero con el initramfs no puede exceder los 8 caracteres de longitud.
- El kernel a utilizar debe ser compilado con soporte para la carga de discos en RAM como se vio anteriormente, ya que el kernel suministrado en binario y las configuraciones de compilación por defecto no incluyen esta característica.
- Es necesario incluir un fichero de configuración junto con los binarios del firmware, llamado "config.txt", con las siguientes variables de configuración:

```
boot_delay=2
initramfs init.gz
```

Estos parámetros fuerzan una pausa de 2 segundos entre la finalización de la carga del firmware y el inicio de la carga del kernel, y especifican el nombre y extensión del initramfs.

Puesto que OpenDomo SDK genera el sistema OpenDomo en formato initrd y no initramfs, fue necesario analizar las diferencias entre ambos formatos de discos en RAM, y realizar los siguientes cambios:

a) Transformar el initrd en formato EXT2 generado por el SDK en un initramfs, siguiendo los pasos indicados a continuación:

   I. Montar el initrd en un directorio como "loop device", que es un pseudo-dispositivo que hace que un fichero sea accesible como un dispositivo de bloques, que básicamente es en lo que consiste un initrd. Para realizar este paso, se emplea el comando "mount initrd directorio_montaje -o loop".

   II. Copiar todo el contenido del initrd en un fichero CPIO con un formato específico, y comprimirlo con Gzip. Los ficheros CPIO tienen un formato similar al de los archivadores Tar, y contienen ficheros y directorios junto con encabezados utilizados para extraer el contenido, así como encabezados extra como el nombre, fecha de creación, permisos y propietario de cada fichero y directorio, etc. Son el formato de archivo utilizado en los initramfs. Para realizar este paso, desde el directorio donde se encuentra montado el initrd, se emplea el comando "find | cpio -H newc -o | gzip > ../nombre_initramfs.gz", que generará el initramfs en el directorio padre.

   III. Si se desea desempaquetar el initramfs creado para su análisis o modificación, se recomienda crear un directorio para extraer el contenido, y desde su interior, ejecutar el comando "cat ../nombre_initramfs.gz | gzip -d | cpio -id". Este comando extraerá el contenido del initramfs en el directorio actual.

b) Modificar la configuración de BuildRoot para que, además de generar el sistema de ficheros raíz como initrd en formato EXT2 para su correcto procesamiento por OpenDomo SDK, generase el sistema de ficheros raíz en formato CPIO (initramfs).

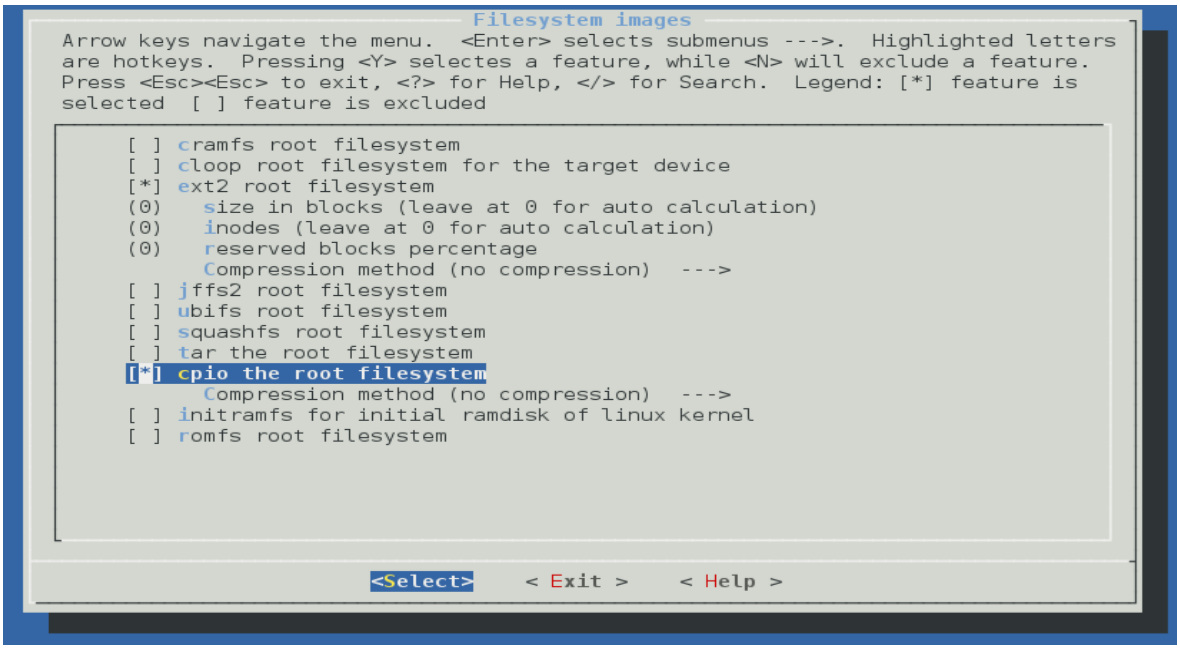


Ilustración 3.24: Utilidad de configuración de BuildRoot, apartado de selección del formato para el sistema de ficheros raíz

Con esta configuración, el sistema de ficheros raíz se genera simultáneamente en dos formatos diferentes desde BuildRoot, de los cuales, sólo uno (initrd en formato EXT2) es utilizado por OpenDomo SDK para generar el sistema, lo que parece no tener sentido. Sin embargo, la selección de esta opción causa que se realicen algunas modificaciones en ambos sistemas de ficheros raíz, que posibilitan posteriormente el arranque del sistema OpenDomo tras su transformación de initrd a initramfs. Por ejemplo, se incluye un script llamado "init" en la raíz del sistema de ficheros, con el siguiente código:

```
#!/bin/sh
# devtmpfs does not get automounted for initramfs
/bin/mount -t devtmpfs devtmpfs /dev
exec 0</dev/console
exec 1>/dev/console
exec 2>/dev/console
exec /sbin/init $*
```

Al finalizar la ejecución del script, se ejecuta el proceso "init" real parametrizado, que como se comentó anteriormente, es en realidad un enlace simbólico al ejecutable de Busybox, ya que se encuentra incluido en el mismo.

3. Modificación de OpenDomo SDK para la generación automática del port para Raspberry Pi

En este apartado, se incluyen tanto las modificaciones a realizar en el SDK para la generación automática del sistema OpenDomo, como las llevadas a cabo sobre los ficheros internos del propio sistema, así como las modificaciones necesarias para la correcta compilación de los paquetes específicos de OpenDomo, ya que hasta este punto, se dispone de un sistema que arranca, pero que no dispone de los paquetes software principales.

Cabe mencionar, que aunque inicialmente la modificación del SDK se incluía dentro del ámbito del proyecto, tras estudiar los cambios a realizar, OpenDomo Services S.L. ha decidido tomar las riendas de estas modificaciones, proponiéndose una reestructuración completa del mismo para permitir agregar cómodamente soporte para nuevas arquitecturas, así como otras funcionalidades fuera del ámbito de este proyecto. Por tanto, las tareas realizadas se han limitado a modificar los ficheros necesarios para conseguir el correcto funcionamiento del sistema en la Raspberry Pi, desde su generación a su ejecución, y posteriormente comunicar estas modificaciones a OpenDomo Services S.L., para que decidieran cómo incluirlas dentro de la nueva versión del SDK.

Todas las modificaciones realizadas van precedidas de la etiqueta "#RASPBERRY" para permitir su fácil identificación, y se han resaltado en negrita. También se han mantenido como comentarios las líneas originales en caso de existir. No se ha copiado el código completo de los ficheros modificados por cuestiones de espacio, y se han señalado las zonas con código omitido mediante puntos suspensivos. Las principales modificaciones han sido:

a) Cambios en el script "process_initrd.sh" para generar automáticamente el initramfs a partir del initrd de OpenDomo.

```
#!/bin/sh
```

```
#desc: Crea un initrd de OpenDomo a partir del de Busybox

# Copyright(c) 2011 OpenDomo Services SL. Licensed under GPL v3 or later

# ------------------------------------------------------------------------
#  Script Configuration
# ------------------------------------------------------------------------
ROOTFS="../../buildroot/output/images/rootfs.ext2"
WORK=/tmp
MNTORIG=/mnt/buildroot
MNTDIR=/mnt/opendomo
#INITRDSIZE=42

#RASPBERRY
#INITRDSIZE=64
INITRDSIZE=128

#RASPBERRY
ARCH="arm"

if test `whoami` != "root"
then
       echo "You must be root to run this script"
       exit 1
fi

...
...

for lst in `ls $HERE/drivers/*.lst`; do
       # this pkgname should be the one that odkoloader needs to know
       pkgname="`basename $lst | sed -e 's/.lst//'`"

       #RASPBERRY
       #rm -fr $HERE/drivers/${pkgname}-$VERSION.i486.tar.gz
       rm -fr $HERE/drivers/${pkgname}-$VERSION.$ARCH.tar.gz

       #RASPBERRY
       #if tar cf $HERE/drivers/${pkgname}-$VERSION.i486.tar --remove-files \
       #      -C $MNTDIR `cat $lst` 2> $HERE/drivers/${pkgname}.notfound
       if tar cf $HERE/drivers/${pkgname}-$VERSION.$ARCH.tar --remove-files \
              -C $MNTDIR `cat $lst` 2> $HERE/drivers/${pkgname}.notfound
       then

              if test -d $HERE/drivers/$pkgname
              then
                     chmod 755 -R $HERE/drivers/$pkgname
                     cd $HERE/drivers/$pkgname

                     #RASPBERRY
                     #tar rf ../${pkgname}-$VERSION.i486.tar * $EXCLUDE
                     tar rf ../${pkgname}-$VERSION.$ARCH.tar * $EXCLUDE

                     cd ../..
              fi
              touch $HERE/drivers/${pkgname}.deps
              echo -n " $pkgname "
       else

              #RASPBERRY
              #rm $HERE/drivers/${pkgname}-$VERSION.i486.tar 2>/dev/null
```

```
                rm $HERE/drivers/${pkgname}-$VERSION.$ARCH.tar 2>/dev/null
        fi
done

...
...

# -- Supported architectures (for packages selection) --
#RASPBERRY
#echo 'x86|i386|386|i486|486' > $MNTDIR/etc/archs
echo $ARCH > $MNTDIR/etc/archs

# Every ISO is prepared for a determined architecture (in this case, 486), so
# this file will tell the package manager which packages can use apart of noarch

# -- Support scripts --
echo '#!/bin/sh
grep $USER /etc/group | cut -f1 -d:' > $MNTDIR/bin/groups
chmod 555 $MNTDIR/bin/groups

...
...

if test -f $MNTDIR/usr/bin/sudo; then
        chmod 4755 $MNTDIR/usr/bin/sudo
fi

#RASPBERRY
#Before umount the new initrd root filesystem, it is used to create the initramfs
for Raspberry Pi
ACTUAL=`pwd`
cd $MNTDIR
find | cpio -H newc -o | gzip > $ACTUAL/bootcd/init.gz
cd $ACTUAL


umount $MNTDIR


rm -fr bootcd/initrd.gz
gzip bootcd/initrd

#RASPBERRY
#The initrd in EXT2 format can be deleted
rm -fr bootcd/initrd.gz

chmod ugo+rw bootcd/* -R
```

b) Cambios en los scripts de compilación de todos los paquetes dependientes de la arquitectura, para que utilizasen la nueva cadena de herramientas de compilación cruzada. Principalmente se han modificado los ficheros “mkpkg.sh” y “Makefile” como sigue (se han usado los ficheros de un paquete en particular como ejemplo):

1. mpkkg.sh

```
#!/bin/bash
DATE=`date +%Y%m%d`
#RASPBERRY
#ARCH=i486
```

```
ARCH=arm
PKG="oddiscovery"
USR="--owner 1000 --group 1000 --same-permissions"
EXCLUDE=' --exclude "*~" --exclude .svn '

...
...
```

2. Makefile

```
ifeq ($(gcc),1)
   CC=gcc
else
   TOOLCHAIN=../../buildroot/output/host
   #RASPBERRY
   #CC=$(TOOLCHAIN)/usr/bin/i486-unknown-linux-uclibc-gcc
   CC=$(TOOLCHAIN)/usr/bin/arm-unknown-linux-uclibcgnueabi-gcc
endif

...
...
```

c) Cambios en el script "/etc/init.d/mount" del sistema OpenDomo para que incluyese el análisis del dispositivo en que el kernel asociaba la tarjeta SD (mmcblk0), junto con sus particiones (mmcblk0p1 y mmcblk0p2).

```
#!/bin/sh
#desc:Local storage

# Copyright(c) 2011 OpenDomo Services SL. Licensed under GPL v3 or later

mkdir -p /media/

# Por precauciÃ³n, reaplicamos permisos importantes
chown admin:admin /media
chown admin:admin /mnt/
chown admin:admin -R /usr/share/
ls -lah /var > /tmp/varlist.txt

/usr/bin/logger "Mounting drives"


# En el arranque predeterminado solamente funcionan las primeras particiones
#RASPBERRY
#DRIVEORDER="sr0 sda sda1 sdb sdb1 sdc sdc1 hda hda1 hdb hdb1 hdc hdc1 hdd hdd1"
#Se agrega la partición 2 de la tarjeta SD, tal y como la reconoce el sistema al
arrancar
DRIVEORDER="sr0 sda sda1 sdb sdb1 sdc sdc1 hda hda1 hdb hdb1 hdc hdc1 hdd hdd1
mmcblk0p2"


start(){
  # Hack for ISO auto-mounting and misconfigured systems
  OPTIONS="rw,nosuid,uid=1000,gid=100,dmask=0027,fmask=0027,flush"

  #RASPBERRY
  #for d in sr0 sda sdb sdc hda hdb hdc hdd; do
  #Se agrega la partición de datos de la tarjeta, tal y como la reconoce el
sistema en el arranque
```

```
  for d in sr0 sda sdb sdc hda hdb hdc hdd mmcblk0p2; do
        if test -e /dev/$d && ! test -e /mnt/$d; then
             mkdir /mnt/$d
             # Try to mount with options, otherwise without.
             mount /dev/$d /mnt/$d -o $OPTIONS || mount /dev/$d /mnt/$d
        fi
  done


  FSORDER="ext3 ext2 iso9660 vfat"
  for dev in $DRIVEORDER; do
      #for fs in $FSORDER; do
             #if ! test -d /mnt/$dev; then
             #      # Tener en cuenta los tipos particulares de FS
             #      if test "$fs" = "vfat"; then
             #             # Posiblemente se trate de un pendrive, ya que
             #             # no es ningÃºn sistema nativo de Linux. Hay que
             #             # dar acceso a todos los usuarios, ya que el
             #             # owner será root:root.
             #             op=" -o fmask=0111,dmask=0000,sync "
             #      else
             #             op=""
             #      fi
             #      mkdir -p /mnt/$dev
             #      if mount -t $fs /dev/$dev /mnt/$dev $op 2>/dev/null
                    if test -d /mnt/$dev
                    then
                           #RASPBERRY
                           #El script process_disk no crea el enlace. Esto es un
workaround
                           ln -s /mnt/$dev /mnt/odconf
                           /usr/bin/logger "Mounting $dev"
                           echo -n "($dev)"
                           logger "Device $dev detected"
                           if test -e /mnt/$dev/opendomo.cfg; then
                                  /sbin/process_disk $dev
                           else
...
...
```

4. Optimizaciones de rendimiento

Disponiendo ya de un sistema funcional sobre el dispositivo, se buscaron las configuraciones necesarias para lograr el máximo rendimiento del sistema OpenDomo sobre la Raspberry Pi. Estas configuraciones se centraron en el correcto reparto de la memoria RAM del dispositivo entre la CPU y la GPU, ya que su firmware ofrece la posibilidad de manipular estos parámetros de forma sencilla.

El enfoque adoptado consistió en asignar la máxima cantidad de memoria RAM a la CPU, de manera que esta memoria esté disponible para el funcionamiento del sistema OpenDomo, consiguiendo el mejor rendimiento posible del mismo. A la GPU se le asignará la cantidad de memoria mínima necesaria, ya que no se espera que tenga que realizar ninguna tarea, puesto que no se dispone de entorno gráfico, y en general, se espera que el dispositivo sea gestionado a través de la red, no siendo necesario utilizar ninguna de sus salidas de vídeo.

La cantidad mínima de memoria asignable a la CPU son 16 MB, y con el firmware actual, esta asignación se realiza de la siguiente forma:

- El reparto de la memoria se realiza mediante un parámetro incluidos en el fichero "config.txt" dentro de la partición de arranque, junto con los ficheros binarios del firmware.
- Este parámetro es "gpu_mem", e indica los MB de RAM, en bloques de 16 MB, que se asignarán a la GPU. El resto de la RAM se asigna automáticamente a la CPU.
- Usando el valor mínimo de este parámetro (gpu_mem=16), se reservan únicamente 16 MB para la GPU, dejando los 240 restantes para la CPU. Este reparto de memoria hace que el sistema arranque con los ficheros de firmware "start_cd.elf" y "fixup_cd.dat", en lugar de usar los habituales "start.elf" y "fixup.dat".

Por tanto, el fichero "config.txt" final deberá contener las siguientes líneas para conseguir el arranque del sistema desde el initramfs, y con 16 MB de RAM para la GPU:

```
boot_delay=2
initramfs init.gz
gpu_mem=16
```

5. **Documentación de usuario**

Junto con esta memoria, se ha elaborado un manual de usuario (5.2) que describe detalladamente todos los pasos a realizar para la puesta en marcha desde cero del sistema OpenDomo sobre la Raspberry Pi, abarcando desde la descarga del SDK, hasta la instalación del sistema en la tarjeta SD y posterior optimización de su funcionamiento.

3.2.2 Planificación de tareas

Seguidamente, se muestra la planificación establecida para llevar a cabo las tareas implicadas en el desarrollo del port de OpenDomo para Raspberry Pi.

WBS	Nombre	Inicio	Fin	Trabajo
1	Análisis del sistema OpenDomo	sep 20	oct 5	12d
2	Análisis interno de OpenDomo SDK	oct 8	oct 23	12d
3	Análisis del funcionamiento de BuildRoot	oct 24	nov 26	24d
4	Análisis del dispositivo Raspberry Pi	nov 27	dic 12	12d
5	Configuración de BuildRoot y herramientas	dic 13	mar 22	72d
6	Arranque del port de OpenDomo sobre la Raspberry Pi	mar 25	may 13	36d
7	Modificación de OpenDomo SDK para generación del port	may 14	may 29	12d
8	Optimizaciones de rendimiento	may 30	jun 14	12d
9	Documentación de usuario	jun 17	jun 20	4d

Ilustración 3.25: Tabla de planificación de tareas

Junto con su orden, fecha de inicio y finalización, y duración en días, el siguiente gráfico permite observar más claramente el reparto de estas tareas a lo largo del periodo de vida del proyecto:

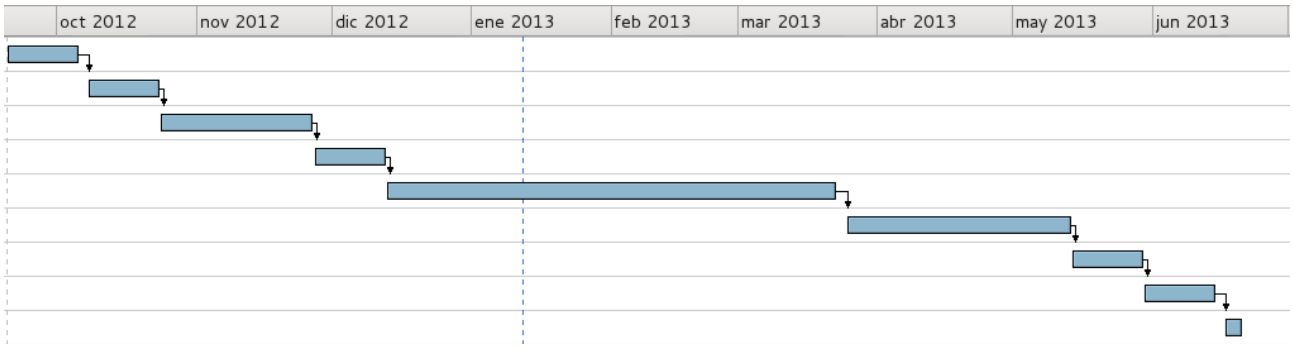


Ilustración 3.26: Esquema de representación de tareas respecto a su orden y duración

3.2.3 Elección de licencia

No ha sido necesario llevar a cabo un proceso de elección de licencia, ya que el proyecto se ha basado en la utilización, configuración y modificación de diverso software ya existente, con licencias ya establecidas. Estas licencias son libres en su mayor parte, principalmente LGPL y GPL en sus diferentes versiones, aunque hay otras.

3.2.4 Entorno de desarrollo

Para el desarrollo del proyecto se ha empleado una Raspberry Pi modelo B, que se espera sea el modelo también usado en producción, por lo que no se plantean problemas. No obstante, cabe mencionar que existe un modelo A, anterior al utilizado en el desarrollo, con ligeras diferencias hardware (mitad de memoria RAM, carencia de controladora Ethernet, y un único puerto USB) que pueden hacer necesaria la realización de algunas adaptaciones. Estas adaptaciones afectarían en principio a la configuración relacionada con la optimización de rendimiento, ya que el reparto de la memoria RAM sería diferente, y a los módulos necesarios para el funcionamiento de la controladora de red, ya que al carecer de una incorporada será necesario conectarla vía USB, y se requerirá un análisis de las alternativas disponibles.

El proyecto no contempla futuros modelos del dispositivo.

4 Resultados, valoración y conclusiones

4.1 Análisis de los resultados

A continuación, se analizarán los resultados obtenidos tras la finalización del proyecto, en función de los objetivos alcanzados respecto a los planteados inicialmente, y el cumplimiento de la planificación establecida.

4.1.1 Cumplimiento de objetivos

Los objetivos inicialmente propuestos fueron:

- Portabilidad del sistema OpenDomo para su ejecución sobre el dispositivo Raspberry Pi.
- Modificación de OpenDomo SDK para que, mediante simple parametrización de los scripts, sea capaz de generar el sistema tanto para ODNetwork (x86) como para Raspberry Pi (ARM).

Ambos objetivos han sido cumplidos, si bien el segundo de ellos no se ha realizado conforme se esperaba al comienzo del proyecto. Inicialmente, la tarea de modificación del SDK iba a llevarse a cabo bajo nuestra responsabilidad, sin embargo, finalmente Opendomo Services S.L. decidió encargarse de la misma debido a un cambio en el alcance planteado al inicio. Además de las modificaciones necesarias para generar el sistema para Raspberry Pi y ODNetwork, decidieron abordar otras más profundas destinadas a facilitar el soporte de nuevas arquitecturas en el futuro, junto con nuevas funcionalidades del sistema en las que estaban trabajando. Por este motivo, las tareas llevadas a cabo se centraron en la modificación de una copia local del SDK para lograr un sistema funcional sobre Raspberry Pi, para posteriormente transmitir estas modificaciones a OpenDomo Services S.L., de manera que analizasen la mejor forma de incluirlas en la nueva versión del SDK.

4.1.2 Cumplimiento de la planificación

En cuanto a la planificación establecida, sólo se ha conseguido un cumplimiento parcial, si bien cabe mencionar que en la mayoría de los casos las desviaciones han sido como consecuencia de la finalización de tareas antes de la fecha prevista, lo que normalmente no suele ocurrir. Aunque la planificación se llevó a cabo contando una duración total de entre 8 y 9 meses, la realización de algunas tareas previamente al inicio oficial del curso académico, junto con el buen ritmo de desarrollo, hubieran permitido la realización del proyecto en un máximo de 6 meses. Prueba de ello, es que a fecha de redacción de estas líneas (febrero de 2013), el proyecto se encuentra casi concluido, a la espera de finalizar algunas modificaciones al SDK, mientras que atendiendo a la planificación, aún se debería estar ajustando la configuración de BuildRoot y el resto de herramientas.

Otro aspecto a tener en cuenta, es que ha sido necesario realizar algunas tareas fuera de la planificación establecida inicialmente, debido a necesidades de Opendomo Services S.L. y para adaptarse adecuadamente al calendario de entregas fijado por la UOC.

4.2 Valoración y conclusiones

Además los objetivos del propio proyecto, inicialmente también se plantearon otros a nivel personal, que fueron:

- Poner en práctica los conocimientos adquiridos a lo largo de estos estudios.
- Adquirir conocimientos técnicos sobre sistemas y dispositivos empotrados, que hasta ahora se encontraban fuera de mi ámbito académico y profesional.
- Mejorar mis habilidades en el desarrollo de la documentación asociada a un proyecto

compuesto íntegramente por Software Libre, especialmente en el ámbito de la redes y los sistemas operativos.

- Familiarizarme y perfeccionar los métodos de trabajo utilizados en los proyectos libres, basados no sólo en la lectura de documentación, sino en la experimentación y el contacto directo con los desarrolladores y las comunidades de usuarios, generando retroalimentación cuando sea posible (reporte de bugs, sugerencias de funcionalidades, etc.).

Al igual que con los objetivos del proyecto, considero alcanzados todos los objetivos personales. El proyecto realizado ha permitido poner en práctica muchos de los conocimientos adquiridos a lo largo de los estudios, y sin duda he ampliado profundamente mis conocimientos sobre sistemas y dispositivos empotrados, un campo que desconocía casi por completo. También he tenido la oportunidad de mejorar mis habilidades a la hora de desarrollar documentación asociada a un proyecto de Software Libre, si bien considero que el tipo de licencias asociadas al software del proyecto tienen poco impacto sobre la documentación a desarrollar.

Por otro lado, haber trabajado íntegramente con herramientas libres ha permitido mejorar mis habilidades a la hora de buscar información y documentación a través de los canales más comúnmente utilizados en este ámbito, como foros, wikis, IRC, listas de correo, etc., intercambiando información y opiniones de forma directa con otros usuarios y desarrolladores, reportando bugs, realizando sugerencias, y colaborando en las pruebas de algunas funcionalidades. Esta estrecha relación con usuarios y desarrolladores ha sido una de las mejores experiencias del proyecto, y me ha hecho sentir parte de una comunidad global en la que es posible tanto aportar como recibir, hecho que considero imposible de alcanzar mediante el uso de herramientas de código cerrado. También he tenido ocasión de poner en práctica mis conocimientos de inglés, sin los cuales no hubiera sido posible comunicarme ni consultar gran parte de la información que he necesitado.

Como conclusión, considero globalmente el proyecto como satisfactorio, aunque como sucede casi siempre, desde la perspectiva que aporta la experiencia hubiera enfocado de diferente forma algunas de las tareas llevadas a cabo. A pesar de todo, aunque evidentemente es posible encontrar aspectos a mejorar o sobre los que puede profundizarse, creo que se ha alcanzado como mínimo una base sólida sobre la que seguir trabajando en el futuro, y a la que aportar mejoras y nuevas funcionalidades.

5 Apéndice

5.1 Batería de pruebas estándar realizadas por Opendomo Services S.L.

Las batería de pruebas estándar que será ejecutada por Opendomo Services S.L. se resume en la siguiente tabla:

Número Prueba	Número Test	Descripción
1	1	Log-in con nombre de usuario incorrecto (admin/admin)
	2	Log-in con nombre de usuario correcto (admin/opendomo)
2	1	Control → Listar controladores
	2	Activar / desactivar luz02
	3	Configurar puertos
	4	Elegir “luz02”
	5	Asignar zona (comedor) y etiqueta (iluminación)
3	1	Control → Seleccionar escena
	2	Añadir
	3	Seleccionar “luz02” → Añadir escena
	4	Nombre “Escena de prueba”
	5	Finalizar
	6	Administrar escenas
	7	Clicar en “Escena de prueba”. Verificar que se abre correctamente
4	1	Control → Lanzar secuencia
	2	Añadir
	3	Nombre “Secuencia de prueba”
	4	Añadir
	5	Añadir nuevo paso (establecer luz02 on, esperar 1, establecer luz02 off)
	6	Volver al listado
	7	Clicar en “Secuencia de prueba” y verificar que se abre correctamente
5	1	Configurar → Administrar condiciones
	2	Clicar en “Actualizar paquetes”
	3	Clicar en “Añadir requisito”
	4	“hora igual a 20:00”. Añadir
	5	Borrar requisito anterior
6	1	Configurar → Administrar zonas
	2	Seleccionar “Comedor”
	3	Guardar cambios
7	1	Log-out and log-in con nombre de usuario raso (user/opendomo)
	2	Control → Lanzar secuencia

Número Prueba	Número Test	Descripción
	3	Seleccionar “Secuencia de prueba”
8	1	Control → Seleccionar escena
	2	Seleccionar “Escena de prueba”
9	1	Clicar en “Zonas”
	2	Clicar en “Comedor”
	3	Verificar que luz02 aparece en Iluminación
	4	Clicar en otra estancia
10	1	Configurar → Paquetes → Instalar paquete
	2	Verificar que los paquetes instalados y no instalados se distinguen
	3	Intentar instalar un paquete ya instalado
	4	Intentar instalar un paquete no instalado

5.2 Manual de usuario

El siguiente apartado contiene una guía rápida donde se indican, de forma resumida, todos los pasos necesarios para generar desde cero y poner en funcionamiento el sistema OpenDomo sobre el dispositivo Raspberry Pi, utilizando un sistema GNU/Linux Debian o derivados, ya que los scripts del SDK utilizan el gestor de paquetes APT.

1. Descarga del SDK

Como usuario sin privilegios, crear una carpeta en algún lugar del directorio “home” correspondiente donde se realizará la descarga. Se debe tener en cuenta que una vez haya sido realizada la primera compilación, el cambio de nombre de este directorio puede acarrear problemas.

Descargar el software subversión si no se dispone del mismo, y desde una shell situarse en el directorio creado. Una vez dentro, ejecutar el comando siguiente:

```
svn checkout https://opendomo.googlecode.com/svn/trunk/ opendomo
```

Este comando descargará en el directorio “opendomo”, dentro del directorio actual, la última versión del SDK.

2. Instalación de dependencias

Desde la shell, acceder al directorio “opendomo” recién creado, y ejecutar el siguiente comando, que descargará las dependencias básicas para el proceso de compilación y generación:

```
./odsdk.sh deps
```

Es posible que haga falta alguna dependencia adicional, que deberá ser identificada en pasos posteriores según los mensajes de error obtenidos a lo largo del proceso.

3. Actualización de la versión del SDK y descarga de BuildRoot

Desde el directorio “opendomo” anteriormente creado, ejecutar el siguiente comando, que actualizará la versión del SDK si fuera necesario, y descargará y desempaquetará la versión necesaria de BuildRoot si no se dispone de la misma:

```
./odsdk.sh update
```

4. Configuración del SDK para compilar el sistema para Raspberry Pi

Para configurar el SDK de manera que genere el sistema adaptado para el dispositivo Raspberry Pi, y elimine los paquetes generados anteriormente, se deben ejecutar los siguientes comandos:

```
./odsdk.sh arch arm
./odsdk.sh rmpkg
```

5. Generación del sistema

Para lanzar el proceso de generación del sistema, que incluye la descarga del software necesario, la compilación del mismo, y la generación del initramfs, se debe ejecutar el siguiente comando:

```
./odsdk.sh brmake
```

Casi al final del proceso, que puede tardar alrededor de una hora dependiendo de la máquina utilizada y la velocidad de la conexión a Internet, se pedirán las credenciales del usuario para lanzar una serie de tareas con privilegios administrativos mediante el comando “sudo”, por lo que será necesario tenerlo configurado en el sistema.

Al finalizar, se tendrá un initrd de OpenDomo en el directorio “opendomo/opendomo-distro/bootcd/” llamado “initrd”, y una imagen del kernel para Raspberry Pi, también comprimida con Gzip, en “buildroot/output/images/”, llamada “zImage”. Esta imagen del kernel ya es funcional sobre el dispositivo, pero aún debe transformarse el initrd en un initramfs para que funcione adecuadamente sobre la Raspberry Pi.

6. Compilación del resto de paquetes adicionales

Una vez se tiene generado el sistema, y por tanto, se dispone de todas las herramientas de compilación cruzada, se deben compilar los paquetes adicionales del sistema mediante el siguiente comando:

```
./odsdk.sh pkg
```

Si todo va bien, se habrán generado archivadores comprimidos con los binarios de cada paquete adicional, que se encontrarán en los correspondientes subdirectorios “pkg/” de cada paquete, dentro del directorio “opendomo/”.

7. Generación de los ficheros de arranque para la Raspberry Pi

El siguiente comando, procesa el initrd de Opendomo para transformarlo en un initramfs para Raspberry Pi, y adicionalmente, descarga todos los ficheros de firmware necesarios:

```
./odsdk.sh mkrpi
```

Al finalizar su ejecución, se dispondrá de un directorio llamado “opendomo/raspberryfiles/”, que contendrá a su vez los subdirectorios “firmware” y “boot”.

El directorio “firmware” contiene un clon del repositorio oficial de firmware para la Raspberry Pi, con excepción de las imágenes binarias del kernel (ficheros “kernel.img” y “kernel_emergency.img”), que son borradas para sustituirlas por la imagen ya generada para el

sistema OpenDomo. Es necesario tener instalado Git para poder clonar correctamente el repositorio.

El directorio “boot”, por su parte, contiene todos los ficheros a copiar directamente en la tarjeta SD, dentro del directorio raíz de la partición de arranque (FAT32), incluyendo el firmware, ficheros de licencia, initramfs de OpenDomo, imagen del kernel, y fichero de configuración.

8. Preparación de la tarjeta SD

La preparación de la tarjeta SD borrará todo el contenido de la misma. A continuación se muestran los pasos para la preparación de la tarjeta utilizando el particionamiento recomendado, basado en dos particiones separadas, una para el arranque y otra para los datos.

- ✔ El primer paso, será determinar el dispositivo que corresponde a la tarjeta. Para ello se recomienda analizar con los comandos “mount” y “fdisk” los discos montados y conectados al sistema. Analizar la salida del comando “dmesg” o el fichero “/var/log/messages” tras conectar la tarjeta al equipo también puede ser de ayuda. En el ejemplo, la tarjeta se encuentra en “/dev/sdc”.

  Líneas escritas en “/var/log/messages” al conectar la tarjeta:

```
Jan 19 11:47:05 kernel: [ 9487.025957] sd 12:0:0:0: [sdc] 3842048 512-byte logical blocks: (1.96 GB/1.83 GiB)
Jan 19 11:47:05 kernel: [ 9487.031490]  sdc: unknown partition table
```

- ✔ Seguidamente, se ejecuta “fdisk” sobre el dispositivo, y se analizan sus particiones con la opción “p”, eliminándolas con la opción “o” si fuera necesario.

```
# fdisk /dev/sdc

Orden (m para obtener ayuda): p

Disco /dev/sdc: 1967 MB, 1967128576 bytes
61 heads, 62 sectors/track, 1015 cylinders, 3842048 sectores en total
Units = sectores of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Identificador del disco: 0x33897119

Disposit. Inicio    Comienzo      Fin      Bloques  Id  Sistema

Orden (m para obtener ayuda): o
Building a new DOS disklabel with disk identifier 0xe85b38ee.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Atención: el indicador 0x0000 inválido de la tabla de particiones 4 se corregirá mediante w(rite)
```

```
Orden (m para obtener ayuda): p

Disco /dev/sdc: 1967 MB, 1967128576 bytes
61 heads, 62 sectors/track, 1015 cylinders, 3842048 sectores en total
Units = sectores of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Identificador del disco: 0xe85b38ee

Disposit. Inicio    Comienzo      Fin      Bloques  Id  Sistema
```

- ✔ En caso de usar “o” para eliminar las particiones, se recomienda pulsar “w” para salvar cambios, y reconectar la tarjeta. Si esto no elimina completamente la información, se recomienda usar el comando “dd” para llenar el dispositivo de ceros.

```
# dd if=/dev/zero of=/dev/sdc
dd: escribiendo en «/dev/sdc»: No queda espacio en el dispositivo
3842049+0 registros leídos
3842048+0 registros escritos
1967128576 bytes (2,0 GB) copiados, 511,269 s, 3,8 MB/s
```

- ✔ Se debe anotar el número de bytes totales de la tarjeta, visibles desde “fdisk” con la opción “p”, pues será utilizado más adelante para establecer la geometría. En el ejemplo, la tarjeta tiene un total de 1967128576 bytes.
- ✔ Se selecciona “x” para entrar en el modo experto, y se establece una geometría de 255 cabezas con la opción “h” (heads), y 63 sectores con la opción “s” (sectors).

```
Orden (m para obtener ayuda): x

Orden avanzada (m para obtener ayuda): h
Número de cabezas (1-256, valor predeterminado 61): 255

Orden avanzada (m para obtener ayuda): s
Número de sectores (1-63, valor predeterminado 62): 63
```

- ✔ Se calculan los cilindros a partir de los bytes de la tarjeta (1967128576 en este caso) y la formula siguiente, y se establecen con la opción “c” (cylinders). El resultado de la fórmula debe ser redondeado siempre a la baja.

  Cilindros = 1967128576 / 255 / 63 / 512 = 239,156427015 = 239

```
Orden avanzada (m para obtener ayuda): c
Número de cilindros (1-1048576, valor predeterminado 1015): 239
```

- ✔ Establecida la geometría, se sale del modo experto con la opción “r”, y se crean dos particiones primarias con la opción “n”. La primera tendrá 500 MB de tamaño (opcional, se recomienda un tamaño mínimo de 100 MB), que se establecerá dejando por defecto el valor del primer sector y marcando “+500M” para el último, será de tipo FAT32, que se establecerá con la opción “t” y el código de partición “c”, y además se marcará como de arranque con la opción “a”. La segunda se dejará con los valores por defecto, ocupando el resto de la tarjeta.

```
Orden avanzada (m para obtener ayuda): r

Orden (m para obtener ayuda): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Número de partición (1-4, valor predeterminado 1): 1
Primer sector (2048-3842047, valor predeterminado 2048):
Se está utilizando el valor predeterminado 2048
Last sector, +sectores or +size{K,M,G} (2048-3842047, valor predeterminado
3842047): +500M

Orden (m para obtener ayuda): t
Se ha seleccionado la partición 1
Código hexadecimal (escriba L para ver los códigos): c
Se ha cambiado el tipo de sistema de la partición 1 por c (W95 FAT32 (LBA))

Orden (m para obtener ayuda): a
Número de partición (1-4): 1
Orden (m para obtener ayuda): p

Disco /dev/sdc: 1967 MB, 1967128576 bytes
255 heads, 63 sectors/track, 239 cylinders, 3842048 sectores en total
Units = sectores of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Identificador del disco: 0xe85b38ee

Disposit. Inicio    Comienzo      Fin      Bloques  Id  Sistema
/dev/sdc1   *        2048     1026047      512000    c  W95 FAT32 (LBA)

Orden (m para obtener ayuda): n
```

```
Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p): p
Número de partición (1-4, valor predeterminado 2): 2
Primer sector (1026048-3842047, valor predeterminado 1026048):
Se está utilizando el valor predeterminado 1026048
Last sector, +sectores or +size{K,M,G} (1026048-3842047, valor predeterminado
3842047):
Se está utilizando el valor predeterminado 3842047
```

✔ Una vez comprobado que el particionado es correcto con la opción “p”, se escribe ta tabla de particiones al dispositivo con la opción “w”.

```
Orden (m para obtener ayuda): p

Disco /dev/sdc: 1967 MB, 1967128576 bytes
255 heads, 63 sectors/track, 239 cylinders, 3842048 sectores en total
Units = sectores of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Identificador del disco: 0xe85b38ee

Disposit. Inicio    Comienzo      Fin      Bloques  Id  Sistema
/dev/sdc1   *        2048     1026047      512000    c  W95 FAT32 (LBA)
/dev/sdc2         1026048     3842047     1408000   83  Linux

Orden (m para obtener ayuda): w
¡Se ha modificado la tabla de particiones!

Llamando a ioctl() para volver a leer la tabla de particiones.

ATENCIÓN: Si ha creado o modificado alguna de las
particiones DOS 6.x, consulte la página man de fdisk
para ver información adicional.
Se están sincronizando los discos.
```

✔ En este punto, se recomienda desconectar y volver a conectar la tarjeta para que el sistema detecte los cambios, y se pasará a formatear las particiones como sigue, marcando cada una con la etiqueta correspondiente si se desea (en el ejemplo, “boot” para la de arranque y “datos” para la de almacenamiento).

```
# mkfs.msdos -F 32 /dev/sdc1 -n boot
```

```
mkfs.msdos 3.0.13 (30 Jun 2012)
# mkfs.ext3 /dev/sdc2 -L datos
mke2fs 1.42.5 (29-Jul-2012)
Etiqueta del sistema de ficheros=datos
OS type: Linux
Tamaño del bloque=4096 (bitácora=2)
Tamaño del fragmento=4096 (bitácora=2)
Stride=0 blocks, Stripe width=0 blocks
88000 inodes, 352000 blocks
17600 blocks (5.00%) reserved for the super user
Primer bloque de datos=0
Número máximo de bloques del sistema de ficheros=360710144
11 bloque de grupos
32768 bloques por grupo, 32768 fragmentos por grupo
8000 nodos-i por grupo
Respaldo del superbloque guardado en los bloques:
       32768, 98304, 163840, 229376, 294912

Allocating group tables: hecho
Escribiendo las tablas de nodos-i: hecho
Creating journal (8192 blocks): hecho
Escribiendo superbloques y la información contable del sistema de ficheros: hecho
```

9. Copia de los ficheros a la partición de arranque de la tarjeta SD

Con la tarjeta preparada, es hora de copiar los ficheros necesarios en la partición de arranque, que será la partición FAT32. Se deben copiar en la misma todos los ficheros contenidos en el directorio “opendomo/raspberryfiles/boot/”.

Con estos ficheros ya será posible arrancar el sistema, pero no dispondremos de ninguno de los paquetes adicionales, como ODCGI para la interfaz web de administración.

10. Copia de los ficheros a la partición de datos (paquetes, configuraciones, etc.)

En la partición de datos, en formato EXT3 si se ha seguido el manual, se deberán crear los siguientes directorios y ficheros:

a) Directorio “pkgcache”, que contendrá los binarios de todos los paquetes a utilizar, y que deberán ser copiados a mano de los subdirectorios “pkg” de cada paquete, dentro del directorio “opendomo”. Algunos de ellos son independientes de la arquitectura, pero otros llevarán el sufijo “arm” indicando que son binarios funcionales sobre esta arquitectura, a la que pertenece la Raspberry Pi. También deberán ser incluidos los archivadores del directorio “packages” dentro de “opendomo/opendomo-distro”.

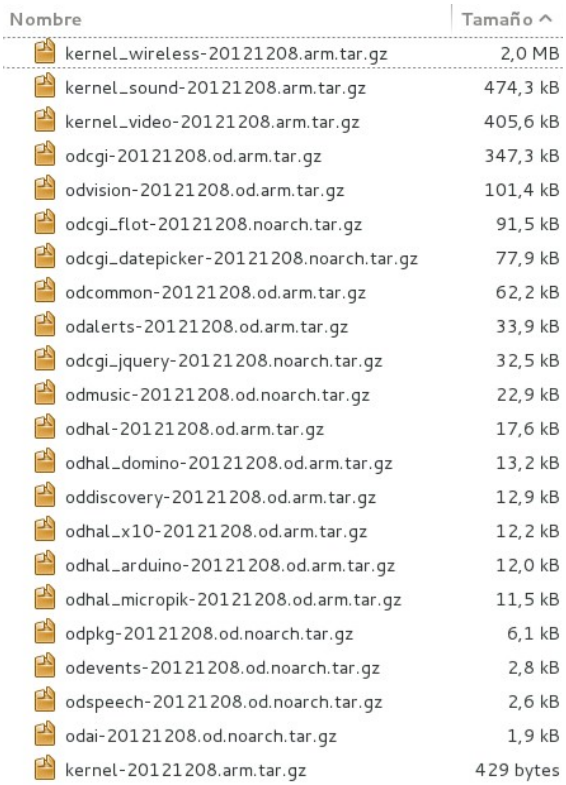


Ilustración 5.1: Contenido del directorio "pkgcache"

b) Fichero de configuración, que debe ser generado a mano. Se trata de un fichero de texto, que debe llamarse "opendomo.cfg" y contener las siguientes líneas:

```
CONFDEVICE="1"
```

Tras el primer arranque del sistema, se crearán automáticamente otros directorios en esta partición, pero los descritos son básicos y deben ser creados a mano.

Con estos pasos se tendrá el sistema OpenDomo instalado en la tarjeta SD, listo para ser arrancado en el dispositivo.

6 Referencias

- Web del proyecto OpenDomo: http://es.opendomo.org/
- Web oficial de Raspberry Pi: http://www.raspberrypi.org/
- Enciclopedia libre Wikipedia: http://www.wikipedia.org/
- Web para compra de dispositivo Carambola: http://shop.8devices.com/carambola
- Web de dispositivo Cubieboard: http://cubieboard.org

- Web para adquisición de dispositivo A13-OLinuXino: https://www.olimex.com/Products/OLinuXino/A13/A13-OLinuXino-WIFI/
- Web del dispositivo Hackberry A10: https://www.miniand.com/products/Hackberry%20A10%20Developer%20Board
- Web para adquisición de dispositivo Open Exynos4 Quad: http://www.hardkernel.com/renewal_2011/products/prdt_info.php?g_code=G133999328931
- Embedded Linux Wiki: http://elinux.org
- Respositorios para Raspberry Pi alojados en GitHub: https://github.com/raspberrypi
- Web de BuildRoot: http://buildroot.uclibc.org/
- Web de uClibc: http://www.uclibc.org
- Web de Busybox: http://www.busybox.net/
- Web de VirtualBox: https://www.virtualbox.org/
- Web de Free Electrons: http://free-electrons.com
- Web de la forja de software Launchpad: https://launchpad.net/
- Web de la distribución Raspbian: http://www.raspbian.org/
- Histórico de la lista de correo de BuildRoot: http://buildroot-busybox.2317881.n4.nabble.com/