Reverse-Engineering a Closed-Box Hardware and Software Linux Embedded System

Rafael Zurita
Email: rafael.zurita@fai.uncoma.edu.ar

Rodolfo Del Castillo
Email: rdc@fi.uncoma.edu.ar

Miriam Lechner
Email: mtl@fi.uncoma.edu.ar

Eduardo Grosclaude
Email: oso@fai.uncoma.edu.ar
Departamento de Ingeniería de Computadoras - Facultad de Informática
Universidad Nacional del Comahue
Neuquén, Argentina

***Resumen*—[Este trabajo expone el proceso, por el cual, un sistema Linux embebido de caja cerrada (dispositivo de hardware y software) ha sido estudiado, con el fin de determinar de que está hecho, qué lo hace funcionar con Linux, y cómo puede ser modificado o actualizado a nivel de software.**

El resultado final, luego de aplicar el procedimiento, es una placa electrónica de desarrollo Linux embebido genérica, que ha sido adaptada y reutilizada para otros fines de interés, en el ámbito de los sistemas Linux embebidos.

Para la experimentación práctica se utilizó un dispositivo ENCORE ENTC1000, que se comercializa como cliente liviano de escritorios remoto.]

Linux embedded systems are often found in closed-box products. The present work documents the process by which such a closed-box, off-the-shelf product, has been analysed to study how it is built, how it is powered by Linux, and how its software can be updated or modified to adapt the device to other usages. The final result, after the described procedure is applied, is a generic, embedded Linux development board, suitable for arbitrary purposes. Our practical experiments targeted an Encore ENTC1000 product marketed as a thin client device for remote desktops.

I. Introducción

[Existen decenas de dispositivos embebidos que utilizamos en la vida diaria. Tostadoras, teléfonos móviles, heladeras, aspiradoras, decodificadores de televisión, relojes, routers hogareños, consola de juegos, etc. Y la mayoría de estos sistemas, a diferencia de una PC que puede realizar miles de funciones diferentes, realizan una única función. Esta única tarea define que nos encontramos ante sistemas embebidos [1].] Dozens of embedded devices such as toasters, mobile phones, refrigerators, vacuum cleaners, TV decoders, clocks, home network routers, game consoles, are used in daily life. Most of these, unlike domestic PCs, are bound to perform a single function. This single function makes them qualify as embedded systems [REF].

[Sin embargo, estos artículos del hogar ya no son tan simples como antes. Algunos años atrás, la mayor parte de estos dispositivos fueron únicamente mecánicos y eléctricos. En cambio, hoy en día, la mayoría incluye placas de circuitos electrónicos, sensores, microcontroladores, microprocesadores y funcionalidades sofisticadas.] However, these household items are no longer as simple as they were once. Some years ago, most of these devices were just mechanical and electrical artifacts. Today's artifacts include electronic circuits, sensors, microcontrollers, microprocessors, and are capable to perform a sophisticate array of functionalities.

[Esta complejidad trajo la necesidad de ejecutar un sistema operativo dentro del dispositivo, ya que se necesita administrar varias funcionalidades al mismo tiempo. Actualmente, los fabricantes han optado por Linux en muchos casos, debido principalmente a su bajo costo y su gran adaptabilidad (referencia electrolux brasil, celular android, router linux)] Such complexity comes along with the need to have the device run an operating system, as several functionalities need to be concurrently managed. Vendors opt for Linux in many cases due to low cost and easy customisation [REF ELECTROLUX BRAZIL ANDROID ROUTER].

[En el marco regional, adquirir sistemas Linux embebidos para investigación o desarrollo es complejo y costoso. Si fuese necesario, por ejemplo, construir un sofisticado dispositivo robótico en nuestra realidad, reutilizaríamos, dentro de lo posible, la mayor cantidad de partes de otros trabajos previos, como así también componentes de hardware y software preexistentes, que puedan adaptarse cuando y donde se necesite.] In our region, acquisition of embedded Linux systems for research or development is a complex and expensive prospect. If we were to build, say, a sophisticated robotic device, then our best strategy would be reusing as many parts as possible from other previous work, and choosing off-the-shelf, ready-made hardware and software components to be repurposed.

[Existen decenas de dispositivos embebidos en el mercado regional, que pueden adquirirse para su estudio y adaptación. Y, aunque muchas veces no se encuentren visiblemente etiquetados o documentados, gran parte de estos sistemas contienen el hardware necesario para ejecutar un sistema operativo Linux embebido. Ejemplos de estos dispositivos capaces de ejecutar un sistema Linux son los routers hogareños TP-Link, y los teléfonos móviles con sistema Android.] Many embedded devices are available in our regional market for study or customization. Most of them come with the required hardware to run an embedded Linux system, although they not always claim so. Typical devices capable to run a Linux system are TP-Link home routers or Android-equipped mobile phones.

[El sistema embebido ENTC-1000 es un producto propietario de ENCORE Electronics Inc [5], que funciona como una terminal de escritorio remoto, soportando los protocolos RDP (Remote-Desktop-Protocol) y Xwindow. Si bien el folleto comercial y su documentación proveen especificaciones mínimas, esto es, se publicita como un sistema con Linux embebido kernel 2.4 [1], no existe información adicional de la arquitectura o funcionamiento interno del software. Tampoco provee, ENCORE Electronic Inc., el código fuente del software instalado, ni las instrucciones para su compilación e instalación (como es requerido por la licencia GPL de Linux [2]).]

The ENTC-1000 embedded system is a proprietary product of ENCORE Electronics Inc. [5], working as a remote desktop terminal supporting RDP (Remote Desktop Protocol) and XWindow protocols. Commercial brochures and product documentation provide minimal specs, i.e. the device is marketed as a Linux 2.4 embedded system[2]. However, there is no additional documentation to architecture or software internals. ENCORE Electronics Inc. does not provide the source code to the installed software, nor any compiling and installing instructions as required by the GPL licence Linux is released under[2].

[Es de interés para nuestra facultad el entendimiento de la arquitectura y funcionamiento de los dispositivos ENTC-1000, ya que se trata de un sistema embebido moderno, y disponible para su adquisición en el país. Conocer su funcionamiento interno nos permite la experimentación académica con Linux embebido. Y, por último, pero no menos importante, permite su reuso para necesidades específicas en el marco de los sistemas embebidos.] As the ENTC-1000 device is a modern embedded system, available for purchase in our country, understanding its architecture and behavior is of interest to our Department. Knowing its internals will allow us an inexpensive way to experiment with embedded Linux for academic purposes.

II. Marco Jurídico

Cuando reutilizamos un sistema embebido de hardware y software cerrado, necesitamos la documentación completa de las especificaciones del dispositivo. Si esa documentación no existe, o si no existe completamente, es necesario descubrir su funcionamiento mediante diferentes pruebas. Este último método es el elegido en este artículo, y es lo que se denomina ingeniería inversa.

Esto plantea la inquietud acerca de la legalidad del proceso, debido a las posibles restricciones en las licencias de uso de cada producto específico. En este caso en particular, sólo se usa ingeniería inversa para poder reutilizar el hardware con un propósito diferente, lo cual es acorde con las Ley Argentina N° 24.240 de Defensa del Consumidor (referencia :), http://www.infoleg.gov.ar/infolegInternet/anexos/0-4999/638/texact.htm), siempre que el producto haya sido adquirido legalmente.

III. Arquitectura de un Sistema Linux Embebido

III-A. Arquitectura de Hardware

Para entender el funcionamiento de un dispositivo Linux embebido, es necesario, en primer lugar, conocer la arquitectura de hardware. En un sistema embebido moderno la arquitectura contiene como componente central, un microcontrolador o microprocesador (CPU). Mas aún, actualmente, muchos sistemas contienen un SOC (del inglés "system on chip"), que contiene la CPU y gran parte de los componentes de un sistema embebido, en un único chip [3].

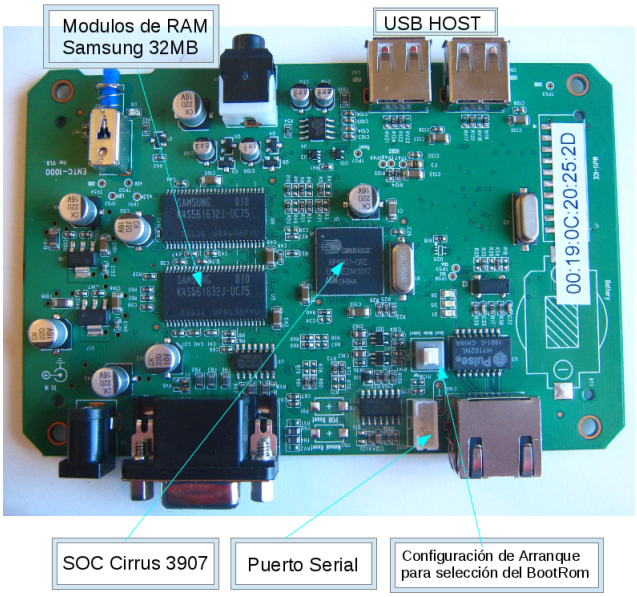

Figura 1. Placa del ENCORE ENTC-1000

Además, es de interés conocer cuáles otros componentes conforman el sistema. Los mas importantes, luego de conocer la CPU, son los módulos de memoria (de tipo RAM y de tipo FLASH), y los módulos de comunicaciones (serial, usb, etc). Ya que, en conjunto con el microprocesador, suelen dar una visión global de cómo inicia o funciona el sistema.

En la práctica, la arquitectura de un sistema embebido particular puede conocerse desmontando la placa del circuito impreso de la caja exterior. Esto posibilita inspeccionar visualmente y listar todos los componentes principales con los que está compuesto el sistema. A partir de este listado, puede obtenerse, en la mayoría de los casos, la documentación pública de cada uno de los componentes o chips listados. A este tipo de documentación se conoce como hoja de datos, o en algunos casos (como el de las memorias o CPU) manuales de programación del chip.

En nuestro caso de estudio, al desmontar el ENTC-1000 (ver la Figura 1) , identificamos un SOC Cirrus EP3907A como el componente principal, que contiene un procesador central de arquitectura ARM, modelo ARM920T a una velocidad de 200Mhz. Además, la placa presenta dos módulos de memoria SDRAM Samsung de 32MB cada uno, y un modulo de memoria FLASH embebida INTEL de 16MB.

Afortunadamente para nuestra investigación, la empresa Cirrus publica documentación detallada del SOC Cirrus 3907A [6]. Junto con esta documentación Cirrus también publica software para su programación, y código fuente de Linux para experimentación. Completando la búsqueda, hemos encontrado

[1] Año 2001
[2] This kernel version was released in 2001.

documentación de placas de laboratorios experimentales, que la fabrica Cirrus publica. Estos diseños son muy útiles a las empresas que utilizan el SOC de Cirrus, ya que lo pueden tomar como diseño de referencia para sus circuitos particulares.

III-B. Arquitectura de Software

En el lado software, la arquitectura consiste típicamente de tres componentes esenciales :

- Un gestor de arranque (del inglés "bootloader") para un sistema embebido, que generalmente es Das U-Boot [4]. Si no se utiliza U-Boot entonces es posible encontrar un gestor de arranque de código cerrado.
- Un kernel Linux o uClinux en caso de ser un sistema sin una unidad de manejo de memoria, MMU (del inglés "Memory Management Unit").
- Un sistema de archivos mínimo que contiene, al menos, una biblioteca de C (generalmente uClibc o eglibc), y busybox, el cual es un pequeño programa ejecutable, optimizado para sistemas embebidos, que puede realizar las operaciones de muchas de las utilidades básicas de UNIX (ls, cp, etc.).

Algunas veces existe un cuarto componente de software, que son las aplicaciones y controladores de hardware (del inglés "drivers") del fabricante. Las aplicaciones muchas veces no son de interés, pero si los controladores. Si existiesen controladores de código cerrado entonces el trabajo de preparar un Linux embebido con soporte para todo el hardware será difícil (debido a que no se tiene acceso al código fuente). En nuestra experiencia con el ENTC-1000 no encontramos drivers de código cerrado, por lo que este punto está fuera del alcance de este artículo.

El trabajo realizado en este punto fue la preparación de dos sistemas Linux embebidos completos de reemplazo: Amstrong y emDebian. La documentación para preparar sistemas Linux embebidos es amplia y públicamente disponible (referencias). Además, existen decenas de distribuciones Linux para embebidos, o entornos de desarrollo de distribuciones Linux para embebidos [3]. Estas herramientas facilitan la preparación de un sistema para un dispositivo específico. Algunas de estos proyectos son Openwrt, Buildroot, u OpenEmbedded (referencias).

Una vez que se tiene el conocimiento en cómo construir (compilar y configurar) y operar los componentes de software para la arquitectura destino, se debe realizar un análisis de si es posible o no modificar el firmware original.

IV. Accediendo por Software

IV-A. Interfaz Serial

Existen varias posibilidades para conectarse a un dispositivo Linux embebido, pero una de las disponibles, mas comúnmente encontrada es a través de una consola serial. En los sistemas Linux, la consola serial permite ver los mensajes de error del kernel, y también interactuar con el gestor de arranque.

La mayoría de los sistemas embebidos contienen un controlador UART (del inglés "Universal Asynchronous Receiver-Transmitter"), utilizado principalmente para depuración y desarrollo. Pero que también es utilizado por Linux como la interfaz serie predeterminada. El UART traduce la información en formato paralelo provenientes del bus del sistema, a datos en formato serie, para que puedan ser transmitidos a través de diferentes puertos. En los sistemas embebidos, generalmente, el controlador UART es parte de la CPU, por lo que, aunque no exista un conector expuesto en la placa, generalmente está presente.

En nuestro trabajo con el ENTC-1000, los contactos (en inglés "pins") del UART no se encontraban identificados o etiquetados, pero estaban presentes. Utilizando un multímetro y un osciloscopio, se identificaron contactos de tierra (GND) y de voltaje de corriente directa (VCC). Luego, utilizando el contacto tierra, se probaron los demás contactos expuestos en la placa, para identificar el contacto de transmisión (TX), y recepción (RX).

Para corroborar que los contactos del UART son los correctos, se conectó el conversor de nivel RS-232 a los mismos, y luego a una PC. Finalmente, se utilizó el programa de comunicaciones minicom para obtener, al menos, información visual sin sentido, ya que no se conocía a que velocidad de transmisión se encuentra operando el firmware original.

IV-B. Tomando Control

Conociendo los componentes de software, se realizaron pruebas de acceso al sistema original en al menos tres niveles de la arquitectura:

- A través de conexiones TCP/IP una vez que el firmware original está en ejecución,
- a través del conexiones al gestor de arranque,
- a través de accesos al software del boot rom.

En los primeros dos casos no se tuvieron resultados promisorios. La tarea en estos puntos es obtener acceso como administrador del sistema Linux cerrado. O en el caso del gestor de arranque, se busca el acceso al interprete de comandos mínimo, que algunos gestores proveen. Por lo que se continuó a mas bajo nivel, para obtener acceso a niveles anteriores a la ejecución del gestor de arranque.

Leyendo la documentación obtenida de la arquitectura y programación del hardware, se analizó el proceso de reinicio del hardware y como la CPU interna del ENTC-1000 comienza a ejecutar instrucciones.

Comunicación con la CPU

El SOC Cirrus EP9307 tiene una ROM de arranque[4]. En un reinicio (en inglés reset) del SOC, la CPU (ARM920t) comienza a ejecutar código en la dirección cero.

Por otro lado, el manual del SOC indica que éste utiliza los controles de la "configuración del hardware" para seleccionar qué dispositivo aparece en la dirección "cero". Al observar la placa ENTC-1000, se identificó un interruptor etiquetado como

[3] No confundir distribución con entorno de desarrollo

[4] boot ROM

"boot mode select". Por lo que se procedió a experimentar con el mismo, con el objetivo de seleccionar el BootROM y así mapearlo a la dirección cero, y por ende, lograr su ejecución.

Afortunadamente, en un reinicio, el Cirrus EP9307, en conjunción con el software de la BootROM, operan en un modo llamado "download en serie", que nos permite enviar unos pocos bytes para ser ejecutados por la CPU. Internamente, en un evento de reinicio, estos componentes (CPU y BootROM) trabajan de la siguiente manera:

1. Se inicializa el UART1 a 9600 baudios, 8 bits, sin paridad, 1 bit de stop.
2. Se envía por esa conexión serie el caracter ">"
3. Se leen desde el sistema embebido, 2048 bytes provenientes de la conexión serie, y se almacenan en una buffer de inicio interno.
4. Decisión:
   - Si el otro extremo de la conexión serie (por ejemplo, nuestra PC) no envía los 2048 bytes, el sistema continúa el proceso de arranque utilizando la memoria interna FLASH.
   - Si el extremo de la conexión serie envió los 2048 bytes, entonces se envía desde el sistema (SOC) otro caracter ">" a través de la conexión serie, para indicar que se leyeron correctamente.
5. Finalmente, la CPU "salta" a la dirección del buffer de boot interno, para continuar leyendo instrucciones desde los 2048 bytes leídos.

En nuestras pruebas logramos obtener el caracter ">" desde el programa de comunicación minicom, por lo que pudimos confirmar varias hipótesis. En principio toda la conexión física entre la placa ENTC-1000 y nuestra PC era funcional. También a nivel de comunicación vía software.

Posteriormente, preparamos un pequeño programa en lenguaje ensamblador que no supere los 2048 bytes, y que nos permita ejecutar instrucciones en la CPU del ENTC-1000. Tomamos como base un ejemplo publicado por Cirrus, y lo modificamos para realizar las pruebas con respecto a la información que nos faltaba por conocer.

En particular, hicimos que nuestro pequeño programa encontrara la dirección física de la memoria FLASH interna. El interés por encontrar esta dirección base, es porque en esta memoria residen los componentes de software: gestor de arranque, el kernel Linux, y el sistemas de archivos raíz.

Para lograr el cometido, enviamos nuestro programa en los 2048 bytes que el sistema lee desde el arranque vía "download en serie". Nuestro programa leyó bytes de direcciones consecutivas, comenzando en la dirección 0x60000000 (que tomamos como referencia del manual de Cirrus). Encontramos la dirección base de la memoria FLASH en la dirección 0x60001000, ya que el programa leyó la palabra "CRUS" en cuatro bytes consecutivos. Esos cuatro bytes "CRUS" indican el inicio del software de arranque, ya que está especificado en el manual de la programación del SOC.

De una manera similar, tuvimos acceso al conocimiento de las direcciones físicas de la memoria RAM.

IV-C. Iniciando un Sistema Linux Embebido Alternativo

Conociendo como ejecutar pequeños programas al momento de un reinicio del hardware, y conociendo las direcciones de la memoria RAM y FLASH, preparamos otro pequeño programa para enviar a la memoria RAM un gestor de arranque (a través de la comunicación serie). En nuestro caso, utilizamos u-boot configurado para este dispositivo.

Una vez que nuestro gestor de arranque estuvo en ejecución, y que pudimos utilizarlo a través de la comunicación serie, tuvimos varias posibilidades, en lo que a software se refiere, para modificar el sistema.

En particular, modificamos la memoria FLASH interna, colocando nuestro gestor de arranque en la dirección posterior a los cuatros bytes "CRUS". También, escribimos en la memoria FLASH un kernel Linux actual, versión 3.X., tal que está preparado para inicializar el hardware y leer, como sistema de archivos raíz, un sistema de archivos contenido en una memoria flash vía usb (pendrive, o similar).

Esto modificó el inicio del firmware interno del ENCORE ENTC-1000, ofreciendo al menos dos interesantes funcionalidades :

- Un gestor de arranque moderno y accesible para ser operado a través de una comunicación serie.
- Un kernel Linux actualizado, y preparado para iniciar diferentes sistemas operativos Linux, ya que lee el sistema de archivos raíz desde memorias conectadas al puerto USB.

Con estas funcionalidades mínimas se pueden realizar pruebas de programación para sistemas Linux embebidos, sin necesidad de acceder continuamente al gestor de arranque.

V. Conclusiones y Trabajo Futuro

Se utilizó ingeniería inversa para entender la arquitectura, funcionamiento y la programación de un sistema Linux embebido moderno.

La experiencia y documentación del proceso nos permitió obtener mayor conocimiento del tema, posibilitó la experimentación académica con Linux embebido, y permitió el reuso de un sistema embebido comercial para necesidades específicas.

Se seleccionó el hardware Encore ENTC-1000 porque cubre las expectativas de prestaciones, aún cuando no se conocía, en un primer momento, si sería posible estudiarlo y reutilizarlo.

Afortunadamente, y luego de esta experiencia, no sólo se ha logrado entender el funcionamiento interno, sino que tambien, se está reutilizando de manera productiva para cumplir disintos roles :

- Como servidor de impresión para impresoras USB sin conexiones de red.
- Como servidor DHCP en pequeñas redes internas de la Universidad.
- Como cerebro de un robot experimental, que se comunica con un microcontrolador y sensores.

- Como sistema Linux embebido para aprendizaje y experimentación.
- Y como un servidor DNS experimental.

Como trabajo futuro, se espera repetir la experiencia [2] en futuros sistemas Linux embebidos comerciales, para estandarizar las tareas necesarias en una metodología, y hasta que la posibilidad de adquirir hardware especializado para desarrollos de sistemas Linux embebidos sea mucho mas accesible para las instituciones publicas. En cuyo caso, la experiencia de ingeniería inversa se puede mantener sobre productos comerciales, como una manera de incentivar el aprendizaje del funcionamiento y la arquitectura de los sistemas embebidos.

REFERENCIAS

[1] S. Heath, *Embedded Systems Design*, Second Edition. Newnes, 2002. ISBN-13: 978-0750655460

[2] Free Software Foundation, *GNU General Public License*. http://www.gnu.org/licenses/gpl.html

[3] S.B. Furber, *ARM System-on-chip Architecture*, Second Edition. Addison-Wesley Professional, 2000. ISBN: 8131708403

[4] Denx, *U-Boot bootloader*. http://www.denx.de/wiki/U-Boot/

[5] Encore Electronics, *Encore Thin Client ENTC-1000*. http://www.encore-usa.com/ar/cat/Thin-Client/Encore-Thin-Client

[6] Cirrus Logic, *EP93xx User's Guide*. http://www.cirrus.com/en/pubs/manual/EP93xx_Users_Guide_UM1.pdf