ESCUELA SUPERIOR POLITECNICA DE CHIMBORAZO

FACULTAD DE INFORMATICA Y ELECTRONICA

ESCUELA DE INGENIERIA ELECTRONICA

"ESTUDIO DE LOS SISTEMAS EMBEBIDOS EN LINUX Y SU APLICACIÓN EN EL DESARROLLO DE UNA TARJETA CON MICRONÚCLEO LINUX PARA LABORATORIO DE COMUNICACIONES"

TESIS DE GRADO

Previa a la obtención del título de

INGENIERO EN ELECTRONICA Y COMPUTACION

Presentada por:

PABLO FERNANDO RAMÍREZ TORRES

RIOBAMBA – ECUADOR

2009

A mis padres, Silvio y Luz

A mis hermanos, Patricio y Danny

A mis familiares y amigos.

Pablo Fernando.

A Dios por iluminarme, guiarme y darme la paz necesaria y la sabiduría suficiente.

A mis padres por los años incondicionales.

A mis maestros por su guía.

Y a todas las personas que de una u otra manera hicieron posible la realización de esta tesis.

NOMBRE	FIRMA	FECHA
Dr. Romeo Rodríguez **DECANO DE LA FACULTAD** **INFORMATICA Y ELECTRONICA**	----------	----------
Ing. Paúl Romero **DIRECTOR DE ESCUELA DE** **INGENIERIA ELECTRONICA**	----------	----------
Ing. Hugo Moreno Avilés **DIRECTOR TESIS**	----------	----------
Ing. Daniel Haro Mendoza **MIEMBRO TRIBUNAL**	----------	----------
Tlgo. Carlos Rodríguez **DIRECTOR CENTRO** **DOCUMENTACION**	----------	----------
NOTA DE LA TESIS	----------	

"Yo **PABLO FERNANDO RAMÍREZ TORRES** soy responsable de las ideas, doctrinas y resultados expuestos en esta tesis; y, declaro que el patrimonio intelectual de la Tesis de Grado pertenece a la ESCUELA SUPERIOR POLITECNICA DE CHIMBORAZO".

_____

PABLO FERNANDO RAMÍREZ TORRES

ACRONIMOS

Big endian: se almacena de la forma en que se lo leería, primero MSB y luego LSB.
Board Support Package o Hardware Abstraction Layer (BSP o HAL): conjunto de programas usados para inicializar los dispositivos de hardware sobre la placa e implementar rutinas específicas de la placa que puedan ser usada por el kernel y los drivers.
Cross-compilation: Compilación Cruzada
ELF: Executable Linkage Format, es un formato ejecutable dentro de los binarios.
Endian: Hace referencia al orden en que se almacenan físicamente los bytes (2 bytes), si el byte más significativo (MSB) se almacena después o antes que el byte menos significativo (LSB).
initramfs (init ram filesystem): Reemplaza a initrd, initramfs se usa en el kernel 2.6
initrd (init ram disk): disco ram de inicio
Little endian: se almacena de manera inversa, es decir, primero LSB y luego el MSB.
Middle endian: sistemas en los que se utiliza ambos modos (bigendian y littleendian).
MTD: Memory Technology Devices, son memorias flash, RAM y chips similares.
Root file System (root fs): Sistema de archivos raíz
RTOS: Sistema Operativo de Tiempo Real
SBC: Computador de Placa única, hace referencia física a un sistema embebido
SO: Sistema Operativo
Toolchain: Herramientas de compilación cruzada
USB: Bus Serial Universal

INDICE GENERAL

INDICE DE FIGURAS

INDICE DE TABLAS

INTRODUCCION

Cuando se habla de un sistema embebido se está haciendo referencia a un dispositivo que, a diferencia de una computadora (PC) de propósito general, es de uso único. Está diseñado para realizar un conjunto específico de tareas u operaciones. Por lo general, son computadoras de una única placa o tarjeta, denominados SBC (del inglés, Single Board Computer). Estos dispositivos funcionan con un sistema operativo almacenado en memorias internas de la misma placa y en la mayoría de los casos son desarrollados por los mismos fabricantes de la SBC. Ejemplos de dispositivos que poseen sistemas embebidos son routers, switchs, firewalls, etc.

Linux en los sistemas embebidos: «Embedded Linux»

Linux está presente en muchas partes. Quizá aún no haya ganado la batalla en los ordenadores personales, pero definitivamente es el número uno en el área de los sistemas embebidos. Sin saberlo, nos rodean miles de dispositivos que funcionan con Linux.

Al contrario de lo que pueda parecer, Embedded Linux no es una versión reducida de Linux. El calificativo «embebido» realmente hace referencia a la funcionalidad de la aplicación, no a la funcionalidad de Linux. En otras palabras, Embedded Linux o Linux Embebido es exactamente lo mismo que Linux. La única diferencia radica en el conjunto de recursos fuera y dentro de Linux que se emplean para conseguir una aplicación.

El éxito de este sistema operativo, creado en 1991, radica en su desarrollo como proyecto «open source» a través de la Licencia Pública General (GPL), que permite a cualquiera usar libremente todo el código desarrollado para el kernel de Linux y obliga a acompañar cualquier distribución modificada del mismo, de todo el código fuente y sus modificaciones. La fiabilidad de Linux es consecuencia directa de esta filosofía que lleva implícita la aportación altruista de miles de programadores de todo el mundo

observando el código, mejorándolo, cambiándolo y probándolo en miles de configuraciones posibles del sistema.

El crecimiento inicial de la comunidad de desarrolladores del kernel de Linux derivó en un crecimiento aún mayor, si cabe, de la comunidad de desarrolladores de aplicaciones y tecnologías para programas que funcionan bajo Linux. Uno de los avances más recientes ha sido la adecuación de Linux para el mercado de los dispositivos embebidos.

Comenzó con el soporte del kernel y el compilador para los microprocesadores más populares de 32 bits: x86, ARM, PowerPC, MIPS y SH. Y luego continuó con la aparición de diferentes distribuciones de Linux con soporte para características específicas de los sistemas embebidos. Gracias a la disponibilidad del código fuente, a la ausencia de 'royalties' y al soporte de los micros y tecnologías modernas, Linux está actualmente atacando de forma feroz el mercado de los RTOS (Real-Time Operating Systems).

Las distribuciones de Linux embebido proporcionan compiladores cruzados (cross-compilers) que junto con la herramienta 'make' permiten compilar y mantener una aplicación para un sistema embebido con un simple comando. La ventaja de utilizar Linux como sistema de desarrollo para crear una aplicación destinada a una plataforma embebida basada en Linux es la posibilidad de probar las aplicaciones directamente en el host mientras todavía no está diseñado el hardware.

Linux se halla, entre otros, en PDAs y ordenadores de bolsillo, teléfonos móviles y teléfonos IP, webpads, reproductores de video y audio, gateways, servidores, firewalls y concentradores wireless, navegadores de automóvil, videocámaras, robots, relojes de bolsillo, minisatélites para los astronautas de la NASA.

Y eso es sólo el principio. Conforme los electrodomésticos vayan adoptando mayor inteligencia y necesiten conectividad de red, Linux irá entrando en nuestras vidas con mayor frecuencia, aunque muchas veces no nos demos ni cuenta.

CAPITULO I

RESEÑA DEL PROYECTO

1.1 ANTECEDENTES

Se entiende por sistemas embebidos a una combinación de hardware y software de computadora, sumado tal vez a algunas piezas mecánicas o de otro tipo, diseñado para tener una función específica. Es común el uso de estos dispositivos pero pocos se dan cuenta que hay un procesador y un programa ejecutándose que les permite funcionar. Esto ofrece un contraste con la computadora personal, que también está formada por una combinación de hardware y software más algunas piezas mecánicas (discos rígidos, por ejemplo). Sin embargo la computadora personal no está diseñada para un uso específico. Si no que es posible darle muchos usos diferentes. Muchas veces un sistema embebido es un componente de un sistema mucho más grande, como por ejemplo los sistemas de frenos o el sistema de inyección de combustible, en automóviles actuales, que son sistemas embebidos.

Esta combinación de software y hardware puede ser reemplazada en muchos casos por un circuito integrado que realice la misma tarea. Pero una de las ventajas de los sistemas embebidos es su flexibilidad. Ya que a la hora de realizar alguna modificación resulta mucho más sencillo modificar unas líneas de código al software del sistema embebido que reemplazar todo el circuito integrado. Un uso muy común de los sistemas embebidos es en los sistemas de tiempo real, entendiéndose por sistemas en tiempo real a aquellos sistemas en los que el control del tiempo es vital para el correcto funcionamiento. Los sistemas en tiempo real necesitan realizar ciertas operaciones o cálculos en un límite de tiempo. Donde ese límite de tiempo resulta crucial. Un ejemplo claro de un sistema de tiempo real es el control de tráfico aéreo.

Hay varias líneas nuevas de procesadores embebidos con soporte en Linux 2.6, incluyendo la serie Hitachi H8/300, el procesador NEC v850, y la línea de procesadores embebidos m68k diseñada por Motorola. Éstos últimos son los más familiares para el usuario corriente de Linux, ya que están en el corazón de las agendas Palm Pilot desde el principio (la Palm 1000). Otros modelos, con nombres tan sugerentes como DragonBall o ColdFire, son utilizados en sistemas y placas de evaluación fabricadas por Motorola, Lineo, Arcturus, y otras empresas. Por desgracia, la v2.6 todavía no permite usar otros procesadores m68k más antiguos sin MMU (como los procesadores 68000 utilizados en los primeros Macintosh), pero es bastante probable que surjan proyectos amateur para incluir éstos sistemas y otros parecidos.

A continuación se exponen varios ejemplos para ilustrar las posibilidades de los sistemas embebidos que usan Linux:

- En una fábrica, para controlar un proceso de montaje o producción. Una máquina que se encargue de una determinada tarea hoy en día contiene numerosos circuitos electrónicos y eléctricos para el control de motores, hornos, etc. que deben ser gobernados por un procesador, el cual ofrece un interfaz persona – máquina para ser dirigido por un operario e informarle al mismo de la marcha del proceso.

- Puntos de servicio o venta (POS, Point Of Service).

- Decodificadores y set-top boxes para la recepción de televisión. Cada vez existe un mayor número de operadores de televisión que aprovechando las tecnologías vía satélite y de red de cable ofrecen un servicio de televisión de pago diferenciado del convencional. En primer lugar envían la señal en formato digital MPEG-2 con lo que es necesario un procesado para decodificarla y mandarla al televisor. Además viaja cifrada para evitar que la reciban en claro usuarios sin contrato, lo que requiere descifrarla en casa del abonado. También ofrecen un servicio de televisión interactiva o web-TV que necesita de un software específico para mostrar páginas web y con ello un sistema basado en procesador con salida de señal de televisión.

- Sistemas radar de aviones

- Equipos de medicina en hospitales y ambulancias UVI – móvil.

- Máquinas de revelado automático de fotos.

- Cajeros automáticos.

- Pasarelas (Gateways) Internet-LAN.

1.2 JUSTIFICACION DEL PROYECTO DE TESIS

El presente proyecto de tesis plantea la posibilidad de desarrollar e implementar en una tarjeta microcontrolada para laboratorio de comunicaciones todo lo necesario para trabajar con prácticas, con su respectivo manual de usuario, cuya principal ventaja es la de incorporar microcontroladores, redes y comunicaciones de datos; donde el microcontrolador tendrá instalado un micronúcleo de Linux para su funcionamiento. La

administración o tareas de configuración se harán desde un PC mediante una conexión Ethernet, o a su vez también desde una conexión RS-232 (puerto serial), para correr o cargar las aplicaciones necesarias tanto para las prácticas como para la incorporación de hardware nuevo.

Existen muchas ventajas y también desventajas al utilizar Linux como sistema operativo embebido en vez del lenguaje propio del microcontrolador:

Ventajas:

- Ofrece una visión menos compleja del Hardware
- Facilita el manejo del Hardware aportando funciones al usuario.
- Portabilidad de las librerías y programas
- Hace parecer al usuario que se ejecutan todas las aplicaciones al mismo tiempo.
- Aporta seguridad en el funcionamiento del sistema y/o uso malintencionado de él mismo.
- Soporte ante posibles fallos de las aplicaciones que se ejecutan.

- El juego de instrucciones de un microcontrolador varia de marca en marca, incluso hasta en los de la misma marca varia el juego de instrucciones, esto hace que los programas no sean portables entre los diferentes micros, con un sistema operativo este problema desaparece.
- Se pueden usar lenguajes de alto nivel para interactuar con el micro, lo cual hace uso de un intérprete para dar al micro las instrucciones en bajo nivel. Se puede tener también una shell de comandos corriendo en un PC para interactuar de

una forma amigable con el dispositivo en cuestión. También es posible hacer la programación "in circuit", lo cual hace bastante amigable el proceso.

Desventajas:

- Se debe usar parte de la memoria para dejar en forma residente el sistema operativo, lo cual en los micros es bastante crítico y además se deben sacrificar algunos puertos de entrada/salida para la comunicación con el programa residente en el PC. Algunas opciones que hemos visto últimamente es construir una pequeña tarjeta con el micro, y un poco de memoria adicional en donde estaría residente el sistema operativo.

1.3 OBJETIVOS

1.3.1 Objetivo General

Estudiar los sistemas embebidos en Linux y sus aplicaciones para el desarrollo de una tarjeta con micronúcleo Linux para ser usada en el Laboratorio de Comunicaciones de la Escuela de Ingeniería Electrónica.

1.3.2 Objetivos Específicos

- Estudiar los sistemas embebidos y sus aplicaciones
- Estudiar y seleccionar el microcontrolador óptimo para el proyecto
- Estudiar y desarrollar un Micronúcleo Linux
- Diseñar la tarjeta con el sistema embebido
- Implementar Hardware y Software
- Desarrollar aplicaciones básicas

CAPITULO II

LINUX EMBEBIDO

2.1 SISTEMAS OPERATIVOS EMBEBIDOS

Nada limita más el corazón de un proyecto de desarrollo como la elección del SO. Si es un pequeño cronograma o kernel, una distribución open-source, un sencillo SO de tiempo real, un ampliamente caracterizado y comercial SO de tiempo real, este conduce todas las decisiones de software y muchas decisiones de hardware también. Un estudio muestra qué es importante a tener en consideración para elegir un SO embebido:

1. Desempeño en tiempo real
2. Compatibilidad con el procesador
3. Herramientas de Software
4. Derechos de autor
5. Precio
6. Huella de Memoria
7. Simplicidad

8. Middleware
9. Compatibilidad
10. Servicios y Características
11. Open source
12. Familiaridad
13. Soporte de Hardware
14. Customización
15. Reputación del fabricante
16. Soporte
17. Popularidad
18. Otros productos

Sistemas Embebidos que no son en Tiempo Real

No todos los SO para sistemas embebidos son diseñados para tiempo real, por ejemplo, Windows CE no es un SO de tiempo real, pero es un SO Multi-tarea, pero conforme el tiempo avanza se espera que estos sistemas usen kernels de tiempo real o parches, para hacerlos funcionales como SO en tiempo real.

Palm OS, corre sobre los CPU Motorola/Freescale 68K basado en el CPU Dragonball y el CPU ARM9, las herramientas de desarrollo están disponibles gratuitamente en el sitio web y varios mirrors.

EPOC/Symbian, soporta el CPU ARM, siendo el líder en el Mercado de los teléfonos móviles.

Sistemas Embebidos en Tiempo Real

a) Comercial

RTX, los productos CMX Real-Time Multi-Tasking Operating System soportan la mayoría de los microcontroladores embebidos, microprocesadores y DSPs (procesadores digitales de señal). Soporta además más de 30 compiladores. Esta línea cuenta con una gama de SO: CMX-RTX, CMX-TINY+, CMX-TCP/IP, CMXBug, y CMXTracker

Quadros, por Embedded Power Corportation. Posee middleware integrado para TCP/IP, CAN, USB, IrDA, FS, GUI. Las herramientas de desarrollo son RTXC Bridge y VisualRTXC, ha sido portado a ARM, ColdFire, Blackfin y otros.

INTEGRITY/velOSity/u-velOSity, por Integrity Green Hills Software ofrece además IDEs de MULTI, AdaMULTI, DoubeCheck, TimeMachine, además de muchos compiladores, software de comunicaciones, etc. Soporta plataformas de desarrollo sobre PCs Linux y Windows y una estación de trabajo SPARC. La lista de soporte a microprocesadores es amplia.

Keil ofrece SO/Kernels para procesadores 8051/C166/ARM. Estos son: **RTX51 tiny**, disponible gratuitamente en su kit, con características limitadas**, RTX51 full**, con capacidades expandidas para el bus CAN. **RTX166 Tiny**, similar al RTX51 pero para la plataforma C16x. **ARTX-166 Advanced**, C16x/ST10, características completas incluyendo sistema de archivos Flash y stack TCP/IP. Librería RL-ARM Real-Time, RTkernel completo para ARM7/9/Cortex-M3, con sistema de archivos Flash y suites TCP/IP.

AMX, desde su introducción por KADAK en 1980, **AMXTM** ha sido reconocido como un RTOS (Sistema Operativo en Tiempo Real) que concentra las más críticas necesidades de las más competitivas aplicaciones en tiempo real, y aún se mantiene fácil de usar y simple de entender. Soporta x86, 68K, ColdFire, PPC32 PowerPC, ARM7, ARM9, StrongARM, Xscale, y MIPS32.

embOS, desarrollado por SEGGER Microcontroller, es un RTOS para aplicaciones embebidas diseñado para ofrecer los beneficios de un completo sistema multi-tarea a costo mínimo. El kernel es completamente interrumpible y muy eficiente que puede ser usado en situaciones muy críticas de tiempo. La huella de memoria en RAM Y ROM es muy pequeña que puede ser usada en aplicaciones de un solo chip, dejando cuarto suficiente para el programa de usuario

iRMX, por tenAsys. El estándar de oro en software de tiempo real para la arquitectura x86, el SO iRMX III, ha sido probado en miles de aplicaciones que demandan tiempo real, alrededor del mundo. Diseñado específicamente para soportar la arquitectura Intel x86 CPUs de 32-bits y los chipsets asociados. Es altamente configurable desde lo más simple hasta una solución completa RTOS. Soporta arquitecturas de tipo PC y de Tipo no PC, como Multibus y Multibus II.

Nucleus, por Mentor Graphics, el SO Nucleus puede mejorar notablemente el desempeño, tiempos de respuesta y sistema de uso de memoria en muchos sistemas complejos, donde otros SO voluminosos agregan significante sobrecarga y no proveen respuesta en tiempo real.

OS-9, por RadiSys. Recientemente rankeado en el Top de los RTOS, es probado en miles de productos representando cientos de sistemas embebidos alrededor del mundo, incluyendo automatización y control industrial, automoción e instrumentación médica.

OSE, una poderosa plataforma para el diseño de sistemas embebidos en tiempo real. El mensaje de la arquitectura basada en OSE instantáneamente dice poderosa simplicidad en sistemas complejos y distribuidos.

pSOS, adquirido por Wind River Inc. Se usa para ser un SO embebido principal, pero sus estatus actual es desconocido.

Phoenix-RTOS, su mayor ventaja como proyecto es el desarrollo libre, portable, pequeño y bien diseñado de sistemas embebidos de tiempo real para plataformas de hardware embebido como las SBC (Single Board Computers, Computadores de una sola Placa), SOM (System on Module, Sistema en Módulo) y SoC (System On Chip, Sistema en Chip). Soporta IA32, PowerPC y ARM.

QNX, su naturaleza ultra confiable significa que su software se prefiere para sistemas críticos de vida como el control de tráfico aéreo, equipo de cirugía, y plantas de control nuclear. Y sus características multimedia lo vuelven ideal en lo referente a las radios, y el info-entretenimiento para las últimas terminales de juego de los casinos.

Salvo, tiene requerimientos de ROM modesta y una RAM minúscula. Esto significa que se puede tener manejo de eventos, basado en prioridades, aplicaciones multi-tarea en los recientes microcontroladores de un solo chip, con Suficiente cabida para aplicaciones. Es un producto de Pumpkin Inc. Soporta 8051, ARM, AVR, M68HC11, MSP430, PIC12, PIC24, PIC32, TMS320 DSP.

VxWorks, Wind River. El RTOS de Microsoft en sistemas embebidos. VxWorks podría ser el SO más usado en el mundo embebido.

b) Open Source

OSEK/VDX, es un estándar usado en aplicaciones distribuidas de automoción. OSEK no es un SO en sí, pero sí una interfaz estándar y una guía para el diseño de SO embebidos.

TRON, The Real-time Operating system Nucleus, ITRON es un estándar Japonés abierto para los RTOS bajo la guía de Ken Sakamura. Este proyecto tiene la intención de estandarizar los RTOS y especificaciones relacionadas para sistemas embebidos, particularmente los de menor escala. El ITRON RTOS apunta hacia dispositivos de consumidores electrónicos, como teléfonos móviles y máquinas de fax. Muchos fabricantes (Japoneses la mayoría) venden sus propias implementaciones de este RTOS.

FreeRTOS es un mini kernel de tiempo real o un RTOS gratuito, de código abierto y portable. Este website muestra como se puede crear un completo RTOS embebido a partir de un host Windows usando herramientas de desarrollo open source de calidad. Soporta ARM7, MSP430, AMD, AVR, 8051.

MicroC/OS-II o uC/OS-II, es un altamente portable, ROMable, muy escalable, pre-vaciable kernel multi-tarea en tiempo real para microprocesadores y microcontroladores. Soporta muchas arquitecturas de procesador como ARM7 y ARM9; y puede manejar por encima de las 255 tareas, brindando servicios como semáforos, correo de mensajes y administración de tareas. Ha sido portado a muchos microprocesadores y microcontroladores. El sistema no es gratuito para propósitos comerciales y posee muchísimo middleware de otros desarrolladores.

InfraBed es un RTOS portable que ha sido portado a la arquitectura ARM7 usando las herramientas de Keil. Es desarrollado por Embedded Artists. Gratuito para propósitos no comerciales.

TinyOS, de Crossbow Technology Inc. ha anunciado la disponibilidad del SO TinyOS 2.0 para el avanzado IRIS de Crossbow. Esto habilita a los programadores para usar la última generación de software sobre la última generación de hardware Sensor de Red. Este sensor usa el microcontrolador AVR con el compilador open source GNU.

eCos es un RTOS open source, libre de autor destinado a las aplicaciones embebidas. La naturaleza altamente configurable permite al SO ser perfeccionado para requerimientos de precisión de las aplicaciones, entregando lo mejor posible el desempeño en tiempos de corrido y una optimizada huella de recursos hardware. Su comunidad web ha crecido alrededor de la seguridad del SO sobre la marcha de la innovación técnica y amplio soporte de plataformas. Soporta ARM, CalmRISC, FR-V, H8, IA32, M86K, AM3x, MIPS, NEC v8xx, PowerPC, SPARC, SuperH.

Nut/OS es un intencionalmente pequeño RTOS para el ATmega128, que provee un mínimo de servicios para correr Nut/Net, el stack TCP/IP. Soporta AVR-GCC, ICCAVR, CodeVisionAVR.

AvrX es un kernel multi-tarea en tiempo real, escrito para las series de microcontroladores Atmel AVR. Contiene aproximadamente 40 APIs en las seis categorías. El kernel está escrito en ensamblador, y su tamaño total varía de ~500 a 700 palabras dependiendo de la versión que está siendo usada. Desde que el kernel se provee como una librería de rutinas, las aplicaciones prácticas toman menos espacio debido a que no todas las funciones son usadas. Soporta IAR assembler y el compilador C GCC.

Linux Embebido y plataformas UNIX

Comunidades

The Linux/Microcontroller Project es un port de Linux para sistemas sin una Memory Management Unit (MMU). **uCLinux** fue primeramente portado al Motorola MC68328: DragonBall Integrated Microprocessor. El primer sistema que booteó satisfactoriamente fue la PalmPilot usando una placa TRG SuperPilot con un bootloader creado específicamente para el port de Linux/PalmPilot.

ARM Linux es un port del kernel Linux completo para las máquinas basadas en el procesador ARM liderado mayormente por Russell King, con contribuciones de muchos otros. ARM Linux está siempre bajo constante desarrollo por varias personas y organizaciones alrededor del mundo.

Distribuciones

Linux/RT, por TimeSys

Red Hat, por muchos años ha desarrollado y usado sistemas embebidos con la ayuda de las herramientas open source de Red Hat y software de desarrollo. Debido al diverso rango de dispositivos y sistemas embebidos, se ha desarrollado un proceso para customizar herramientas y software para un amplio rango de arquitecturas, placas y sistemas.

BlueCat Embedded Linux de LynuxWorks, está basado en el kernel 2.6, es una implementación del modelo Linux, expandido para su uso en sistemas embebidos, desde dispositivos pequeños de tipo consumidor a los sistemas multi-CPU de larga escala.

MontaVista, con alrededor de 2000 clientes desarrollándolo a través de un amplio rango de industrias, es el indiscutible líder y proveedor de plataformas de desarrollo Linux a un grado comercial para dispositivos inteligentes e infraestructura de comunicaciones.

Debian, con su soporte multi-arquitectura, independencia de fabricante, contrato social y enorme base de software lo hace una atractiva elección para toda clase de sistemas, pero la distribución principal es mucho más intencionada para sistemas con por lo menos recursos de escritorio (grandes discos duros, mucha memoria). Embedded Debian trata de desnudar a Debian para que sea un sistema mucho más pequeño manteniendo todas las cosas buenas.

Embedix. Con sus 20 años de experiencia en el desarrollo de dispositivos embebidos y sólida realización como un pionero en Linux Embebido, Lineo se dispara fuertemente en la era de las redes brindando software sofisticado y robusto y servicios confiables para los clientes.

emKnoppix es una sub-distribución de Knoppix, la distribución LiveCD por excelencia, destinada a usarse en sistemas embebidos. La idea surge mientras un usuario evaluaba Knoppix para remasterizarlo y construir una plataforma Linux Embebido al mismo tiempo.

Pico/Linux, The Open Source Handheld OS distribuido por PicoGUI y Linux.

GeeXboX es una distribución libre de Linux Embebido que apunta a volver un PC en algo llamado HTPC (Home Theater PC, PC Teatro en Casa) o Media Center. Siendo una distribución basada en LiveCD, está lista para bootear un SO que puede trabajar sobre cualquier computador x86 clase Pentium o Macintosh PowerPC, sin implicaciones de requerimientos de software, se puede usar incluso en computadores sin disco, el sistema completo se carga en RAM.

UNIX

Embedded FreeBSD, este proyecto brinda herramientas y documentación para usar FreeBSD en un entorno embebido. La ventaja es que FreeBSD puede ser usado para fabricar productos y sistemas competitivos en el mundo embebido. Además su licencia permite publicar los binarios sin el código fuente.

NetBSD, este SO es el más portable del mundo, y muchas de las plataformas de hardware soportado son ideales para aplicaciones embebidas. Mientras el desarrollo embebido con NetBSD no difiere mucho del desarrollo regular de UNIX, algunos casos especiales están en proceso de desarrollo.

Plataforma Windows Embebido

Windows Embedded CE, diseñado para procesadores x86, ARM, MIPS y SH4, tiene muchos nombres como Windows CE, luego Pocket PC, Windows Mobile, Windows CE.NET, Windows automotives, Microsoft Auto. Pero todos se derivan del mismo tronco que por lo general es Windows NT, el cual no es un RTOS, pero se puede decir que el último Windows CE 6.0 es un SO nativo multi-tarea en tiempo real.

Además se han creado versiones embebidas de Windows XP y Windows Vista para procesadores x86, y muchos clones del Microsoft DOS. Existen también derivaciones del Windows CE destinadas a usos específicos en la automoción o puntos de venta.

Existen además muchísimos otros SO embebidos basados en la plataforma JAVA de Sun, sistemas multi-núcleo y una infinidad de arquitecturas, que no estudiaremos debido a que no es el propósito de esta tesis.

2.2 LINUX

Linux en realidad hace referencia al kernel Linux, un sistema Linux, o una distribución Linux aunque esta última en realidad debería denominarse GNU/Linux, debido a que Linux como tal es simplemente el núcleo del sistema pero un sistema operativo completo posee un sinnúmero de herramientas además del núcleo o kernel. Estrictamente hablando, Linux se refiere al kernel mantenido por Linus Torvalds y distribuido bajo el mismo nombre a través de varios repositorios. Esta base de código incluye solamente el kernel. El kernel provee facilidades al núcleo del sistema. Este puede no ser el primer software a correr sobre el sistema, como un bootloader el cual debe precederlo, pero una vez que está corriendo nunca es removido del control hasta que el sistema sea apagado. En efecto, este controla todo el hardware y provee

abstracciones de alto nivel como procesos, sockets y archivos al diferente software que está corriendo en el sistema.

Como el kernel es constantemente actualizado un esquema numérico es usado para identificar una cierta entrega. Ese esquema numérico usa tres números separados por puntos para identificar cada entrega. Los primeros dos números designan la versión y el tercero la entrega. Linux 2.4.20, por ejemplo, es una versión 2.4 entrega número 20 que por cierto significa que es estable. Los números impares como el 2.5 designan kernels en desarrollo y prueba para la siguiente entrega estable que es la actual 2.6. Usualmente, se suele usar un kernel estable de las últimas entregas para los sistemas embebidos.

Finalmente, Linux puede referirse como se mencionó anteriormente a una distribución Linux. Red Hat, Mandriva, SuSE, Debian, Slackware, Caldera, MontaVista, Embeddix, BlueCat, PeeWeeLinux, Ubuntu, Ututo, y otras muchísimas más son distribuciones Linux que incluyen un montón de herramientas complejas. Estas pueden variar en su propósito, tamaño y en algunos casos precio. Para la mayoría de nosotros nos son familiares las distribuciones mediante los CD-ROMs.

2.3 LINUX EMBEBIDO

Primeramente cabe aclarar que Linux como tal no posee una versión "embebida" suele denominársele así debido a los sistemas en los que es portado y usado como sistema operativo, eso no significa que el kernel no pueda ser embebido. Sólo significa que no se necesita un kernel especial para crear un sistema embebido. A menudo se puede usar una de las entregas oficiales del kernel para construir un sistema. Algunas veces se necesita un kernel modificado distribuido por terceros, alguno que haya sido específicamente alterado para una configuración especial de hardware o para el soporte de cierto tipo de aplicación. Los kernels provistos en las varias distribuciones embebidas, por ejemplo, a menudo incluyen algunas optimizaciones no encontradas en el árbol del kernel principal y son parchados para dar soporte a algunas herramientas de

depuración. Mayormente un kernel usado en un sistema embebido difiere de un kernel usado en una estación de trabajo o un servidor por su arquitectura de configuración.

Un sistema Linux embebido simplemente designa a un sistema embebido basado en el kernel Linux y no implica el uso de una librería específica o herramientas de usuario con este kernel, además que debido a su propósito tiene una huella muy reducida.

Una distribución de Linux Embebido puede incluir herramientas de desarrollo para sistemas embebidos, aplicaciones varias de software destinadas a ser usadas en un sistema embebido o ambas.

Las distribuciones con estructura de Desarrollo incluyen varias herramientas de desarrollo que facilitan la puesta a punto de un sistema embebido. Esto puede incluir navegadores especiales de Fuentes, compiladores cruzados, depuradores, software de manejo y administración de proyectos, constructores de imágenes de booteo. Estas distribuciones están pensadas para ser instaladas en un host de desarrollo.

Las distribuciones de tipo customización proveen un set de aplicaciones a ser usadas en el sistema embebido. Éstas suelen incluir librerías especiales, execu, y archivos de configuración destinados al sistema objetivo. Un método puede ser provisto para simplificar la generación de sistemas de archivos raíz para los sistemas embebidos.

2.4 LINUX EN TIEMPO REAL

Inicialmente, Linux en tiempo real fue designado por el proyecto RTLinux liberado en 1996 por Michael Barabanov bajo la supervisión de Victor Yodaiken. La característica de este proyecto fue la de brindar tiempos de respuesta determinísticos bajo un ambiente Linux.

Sin embargo, existen hoy muchos más proyectos que brindan de una forma u otra respuestas en tiempo real bajo Linux. RTAI, Kurt, y Linux/RK todos poseen alto desempeño en tiempo real bajo Linux. Algunas mejoras se obtienen insertando un

kernel secundario bajo el kernel Linux. Otras mejoras en los tiempos de respuesta del kernel Linux se obtienen parchando el sistema.

El adjetivo "tiempo real" se usa en conjunto con Linux para describir un sinnúmero de cosas distintas. Principalmente es usado para decir que el sistema o uno de sus componentes está supuesto a tener tiempos de respuesta fijos, pero si se usa una definición estricta eso no significa que lo que se ofrece sea necesariamente tiempo real.

2.5 RAZONES PARA ESCOGER LINUX

Aparte de las razones mencionadas anteriormente existen motivaciones adicionales para escoger Linux sobre los sistemas operativos tradicionales.

2.5.1 Calidad y Confianza en el Código

Calidad y confianza son medidas subjetivas del nivel de confidencia del código.

Sumado a esto una definición más exacta de calidad del código podría ser difícil de obtener, hay propiedades comunes que los programadores esperan del código:

Modularidad y Estructura

Cada funcionalidad por separado se debería encontrar en un modulo separado, y el archivo de esquema del proyecto debe reflejar eso.

Dentro de cada modulo, la funcionalidad compleja se subdivide en un número adecuado de funciones independientes.

Fácil de Reparar

El código debe ser más o menos fácil de depurar para cualquiera que entienda su estructura.

Extensibilidad

Agregar características al código debe ser justamente simple. En caso de modificaciones estructurales o lógicas deben ser fáciles de identificar.

Configurabilidad

Debe ser posible seleccionar que características del código serán parte de la aplicación final. Esta selección debe ser fácil de llevar a cabo.

Las propiedades de un código confiable deben ser:

Predecible

Durante la ejecución, el comportamiento del programa se supone debería ser como lo define su estructura definida y no debe volverse errática.

Recuperación de Errores

En caso de ocurrir una situación problemática, se espera que el programa tome medidas para recuperarse del problema y alertar a quién corresponda con un completo mensaje de diagnostico.

Longevidad

El programa debe correr sin asistencia por largos períodos de tiempo y conservar su integridad a pesar de las situaciones que encuentre.

La mayoría de los programadores están de acuerdo con que el kernel Linux y la mayoría de los proyectos usados en un sistema Linux llena esta descripción de calidad y

confianza, aunque se han presentado quejas acerca del orden del código comparado al kernel de los sistemas BSD.

2.5.2 Disponibilidad del Código

La disponibilidad del código se refiere a la ventaja de tener el código fuente de Linux y todas las herramientas de compilación disponibles sin restricciones. Los componentes más importantes de Linux, incluido el kernel por sí mismo son distribuidos bajo la Licencia Pública General (GPL) de la Free Software Foundation.

2.5.3 Soporte de Hardware

Amplio soporte de hardware significa que Linux soporta diferentes tipos de hardware, plataformas y dispositivos. Sin embargo un sinnúmero de fabricantes aún no proveen drivers para Linux, se ha hecho un progreso considerable y se espera aún más. Debido a un amplio número de drivers que son mantenidos por la comunidad Linux, se puede usar confiablemente componentes hardware sin el miedo que el fabricante pueda algún día descontinuar esa línea de productos. El amplio soporte de hardware significa que Linux corre en docenas de diferentes arquitecturas hardware.

Nuevamente, ningún otro SO posee este nivel de portabilidad. Dado un CPU y una plataforma, se puede razonablemente esperar que Linux corra en él o que alguien más pasó por el mismo proceso de portar Linux y que pueda asistir a alguna otra persona.

2.5.4 Protocolos de Comunicación y Estándares de Software

Linux también provee un amplio protocolo de comunicación y estándares de software. Esto hace fácil integrar Linux dentro de estructuras existentes.

Se puede fácilmente integrar un sistema Linux en una red Windows existente y esperar que esto sirva a los clientes a través de Samba por ejemplo. Se puede usar una Linux box para practicar radio amateur construyendo esto dentro del kernel.

2.5.5 Herramientas Disponibles

La variedad de herramientas existentes para Linux lo hacen muy versátil. Si se piensa en una aplicación necesaria, seguramente otras personas sintieron la misma necesidad antes. Esto quiere decir que probablemente la aplicación esté disponible bajo GPL y lista para descarga o en proceso de ser publicada, dos grandes sitios en el internet ayudan en este cometido: Freshmeat (http://www.freshmeat.net) y SourceForge (http://www.sourceforge.net).

2.5.6 Soporte de la Comunidad

Esta es una de las grandes fortalezas de Linux. Es aquí donde el espíritu del software libre y la comunidad open source (código abierto) se siente más fuerte. Tan pronto se necesite una aplicación seguramente habrá alguien que pasó por la misma situación y esté dispuesto a compartir sus soluciones, y estará muy contento de hacerlo.

2.5.7 Licenciamiento

El licenciamiento permite a los programadores hacer con Linux cosas que solo podrían soñar hacer con software propietario. En esencia se puede usar, modificar, redistribuir o vender el software con la única restricción de que el software obtenido no debe tener restricciones.

CAPITULO III

ARQUITECTURA DEL SISTEMA

3.1 ARQUITECTURA DE SISTEMAS EMBEBIDOS

3.1.1 Arquitectura Genérica de los Sistemas Embebidos

Desde que los sistemas Linux son hechos de distintos componentes es necesario revisar la arquitectura de un sistema Linux Genérico.

La Figura 1 presenta dicha arquitectura con todos los componentes involucrados. Además la figura abstrae a un alto nivel el contenido del kernel y los otros componentes, las abstracciones presentadas son suficientes para el análisis.

Cabe recalcar que hay una pequeña diferencia en la descripción hecha a continuación, un sistema embebido y una estación de trabajo o servidor, debido a que los sistemas Linux son todos estructurados de igual manera a este nivel de abstracción.

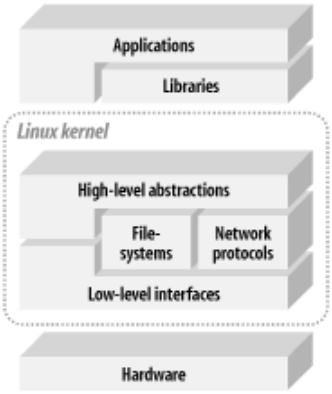


Fig. 1 Arquitectura Genérica de un Sistema Linux

Se requieren algunas características de hardware para que sea posible correr un sistema Linux. Primeramente Linux requiere al menos un CPU de 32-bits que contenga una Unidad de Manejo de Memoria (MMU). Segundo, una cantidad suficiente de memoria RAM disponible para acomodar el sistema. Tercero, capacidades I/O (Entrada/Salida) mínimas son requeridas si se trabajará en algún desarrollo para un sistema embebido y con razonables capacidades para depuración. Esto es también importante para alguna posterior resolución de problemas en el campo. Finalmente, el kernel debe ser capaz de cargar y/o accesar un sistema de archivos raíz a través de alguna clase de almacenamiento permanente o vía red.

Existe una versión especialmente modificada de Linux llamada uCLinux (you-see-Linux, Ves Linux) que es capaz de correr en algunas CPUs que no poseen MMU. El desarrollo de aplicaciones para Linux en dichos procesadores varía, sin embargo, debido al tema expuesto en esta tesis, no se trabajará con dichos procesadores.

Inmediatamente por encima del hardware se sitúa el kernel. El kernel es el componente del núcleo del sistema operativo. Su propósito es manejar el hardware en una manera coherente mientras provee abstracciones familiares de alto nivel para el software de nivel de usuario. Como con otros kernels de tipo Unix, drives Linux de dispositivos, manejos de acceso I/O, agendas de control de procesos, cumplimiento de la memoria compartida, manipulación de la distribución de señales, y otras tareas administrativas. Se debe esperar que las aplicaciones que usan las APIs provistas por un kernel deban

ser portables entre varias arquitecturas soportadas por este kernel con pocos o nulos cambios. Este es el caso usual en Linux, como se puede ver por el cuerpo de aplicaciones uniformemente disponibles en todas las arquitecturas soportadas por Linux.

Dentro del kernel dos amplias categorías de servicios en capas proveen la funcionalidad requerida por las aplicaciones. Las interfaces de bajo nivel son específicas a la configuración del hardware sobre la cual corre el kernel y provee recursos para el control directo del hardware usando una API de hardware independiente. Aquello es, que manejos de registros o páginas de memoria serán hechos de manera diferente sobre un sistema PowerPC y sobre un sistema ARM, pero deberá ser accesible usando una API común para los componentes de alto nivel del kernel, con algunas raras excepciones. Típicamente, los servicios de bajo nivel pueden manejar operaciones específicas del CPU, operaciones de arquitecturas específicas de memoria, e interfaces básicas para los dispositivos.

Sobre los servicios de bajo nivel provistos por el kernel, componentes de alto nivel proveen las abstracciones comunes a todos los sistemas Unix, incluyendo procesos, archivos, sockets y señales.

Los dispositivos de disco han sido y seguirán siendo el medio de almacenamiento principal para los datos computarizados. En el caso de los sistemas embebidos se suele usar además medios disponibles en la red, debido a su poca o nula capacidad de almacenamiento físico, pero esto no es tan recomendable.

Se podría esperar que justo encima del kernel se encontrarían las aplicaciones y utilidades construidas y corriendo sobre el SO. Sin embargo los servicios exportados por el kernel son a menudo no aptos para usarse directamente por las aplicaciones. En vez de eso, las aplicaciones confían en librerías para ser provistas de APIs familiares y servicios abstractos que interactúan con el kernel en nombre de las aplicaciones para obtener la funcionalidad deseada. La principal librería usada por la mayoría de aplicaciones Linux es la Librería C de la GNU. Para los sistemas Linux embebidos, substitutos a esta librería pueden ser usados, para compensar la mayor deficiencia de la

Librería GNU C, su tamaño. Otras además de la librería C, como las librerías Qt, XML, o MD5 proveen varias APIs útiles y funcionales sirviendo a toda clase de propósitos.

Las librerías son típicamente enlazadas dinámicamente con las aplicaciones. Esto es, que ellas no son parte del binario de las aplicaciones, pero son cargadas dentro del espacio de memoria de las aplicaciones durante el inicio de las mismas. Esto permite a muchas aplicaciones usar la misma instancia de una librería en vez de que cada una tenga su propia copia.

3.2 ARRANQUE DEL SISTEMA

Tres principales componentes de software participan en el arranque del sistema: el bootloader, el kernel y el proceso init. El bootloader es el primer software en correr desde el inicio y es altamente dependiente del hardware. Como veremos más adelante existen muchos bootloader disponibles para Linux. El bootloader conduce a la inicialización de hardware de bajo nivel y de allí salta al código de inicio del kernel.

El resto del arranque del sistema es conducido al espacio de usuario por el programa init encontrado en el sistema de archivos raíz.

3.3 TIPOS DE CONFIGURACIÓN DE BOOTEO

El tipo de configuración de booteo elegido para un sistema influencia enormemente la selección de un Bootloader, su configuración, y el tipo de software y hardware que se encuentra en el host. Una configuración de booteo mediante la red, por ejemplo, requiere que el host provea algunos tipos de servicios de red al sistema objetivo. En el diseño del sistema primeramente se necesita identificar las configuraciones de booteo necesarias que probablemente se usarán durante el desarrollo y en el producto final. Luego se necesitará elegir el bootloader más apropiado o un set de éstos que satisfagan las necesidades del sistema. No todos los bootloaders, por ejemplo, pueden arrancar kernels desde dispositivos de disco.

Los sistemas Linux Embebido son tan diversos como sus contrapartes. Los sistemas Linux Embebido se caracterizan sin embargo por el requerimiento de cargar un kernel Linux y su sistema de archivos designado. Cómo estos son cargados y operados depende enormemente de los requerimientos del sistema y algunas veces, del estado de su desarrollo.

Existen tres diferentes configuraciones usadas para realizar un bootstrap a un sistema Linux Embebido: la configuración de dispositivos de almacenamiento de estado sólido, la configuración de disco, y la configuración de red. Cada configuración tiene sus propias configuraciones típicas y usos.

3.3.1 Medios de Almacenamiento de Estado Sólido

En este setup, un dispositivo de almacenamiento de estado sólido (discos duros, pendrives, CompactFlash, etc.) mantiene el bootloader inicial, sus parámetros de configuración, el kernel, y el sistema de archivos raíz. Aunque el desarrollo de un sistema Linux Embebido puede usar otros setups de booteo, dependiendo de la fase de desarrollo, la mayoría de los sistemas de producción contienen un dispositivo de almacenamiento de estado sólido para mantener todos los componentes del sistema. La Figura 2 muestra el esquema más común de estos dispositivos con todos los componentes del sistema.

Fig. 2 Diagrama típico de un Dispositivo de Almacenamiento Sólido

Aunque la Figura 2 muestra los dispositivos de almacenamiento separados en cuatro partes diferentes, estas pueden contener partes más pequeñas. Los parámetros de booteo pueden estar contenidos en el espacio reservado para el bootloader. El kernel puede estar también contenido en el sistema de archivos raíz. Esto sin embargo, requiere que el bootloader sea capaz de leer el sistema de archivos raíz. También el

kernel y sistema de archivos raíz pueden ser empaquetados como una única imagen que se descomprime en RAM antes de ser usada. Dependiendo de las capacidades provistas por el bootloader, podría haber otras posibles configuraciones, con sus ventajas y desventajas. Usualmente se pueden categorizar usando los siguientes criterios: uso de memoria flash, uso de RAM, facilidad de actualización, y tiempo de boteo.

3.3.2 Red

En este setup solamente el sistema de archivos raíz o ambos, el sistema de archivos raíz y el kernel se cargan mediante la red. En el primer caso, el kernel reside en un dispositivo de almacenamiento de estado sólido y el sistema de archivos raíz se monta vía NFS. En el segundo caso solo el bootloader reside sobre un dispositivo de almacenamiento local. El kernel es luego descargado vía TFTP y sistema de archivos raíz montado vía NFS. Para automatizar la localización del servidor, el bootloader debe usar BOOTP/DHCP.

3.4 ARQUITECTURAS DE PROCESADOR

3.4.1 x86

La familia x86 empezó con el 386 introducido por Intel en 1985 y fue incluyendo a todos los descendientes de este procesador, incluyendo al 486 y la familia Pentium, junto con procesadores compatibles de otros fabricantes como AMD y National Semiconductor. Últimamente, una nueva línea está agrupando la funcionalidad de un PC típico con un núcleo basado en un CPU de uno de los procesadores de la familia x86 para formar un System-on-Chip (SoC). La familia Geode de National Semiconductor y el ZFx86 de ZF Micro Devices son parte de esta línea de SoC.

Aunque la x86 es la más popular y más publicitada plataforma para correr Linux, representa una pequeña fracción del mercado de sistemas embebidos tradicionales. En la mayoría de los casos los diseñadores prefieren procesadores ARM, MIPS, y PowerPC por razones de complejidad y costo.

3.4.2 ARM

El ARM, que son las siglas para Advanced RISC Machine, es una familia de procesadores mantenidos y promovidos por ARM Holdings Ltd. Contrario a otros fabricantes de chips como IBM, Motorola e Intel, ARM Holdings no manufactura sus propios procesadores. En cambio ARM diseña los núcleos del CPU para sus clientes que basan sus productos en el núcleo ARM, carga a los clientes honorarios por el licenciamiento en el diseño, y les permite manufacturar el chip dónde les plazca. Esto ofrece varias ventajas para las partes involucradas, pero crea una cierta confusión al desarrollador que desea usar dicha arquitectura por primera vez, debido a que no hay un productor principal del procesador. Existe sin embargo, una característica unificante que es importante mencionar: todos los procesadores ARM comparten el mismo set de instrucciones, lo que hace a todas sus variantes completamente compatibles con el software. Esto no significa que todos los CPUs ARM y tarjetas basadas en estos pueden ser programadas y configuradas de la misma manera, solo que el lenguaje ensamblador y el binario resultante son idénticos para todos los procesadores ARM. Actualmente Intel, Toshiba, Samsung, y muchos otros fabrican procesadores ARM.

3.4.3 IBM/Motorola PowerPC

La arquitectura PowerPC es resultado de la colaboración entre IBM, Motorola, y Apple. Hereda ideas del trabajo hecho por las tres firmas, especialmente de la arquitectura de IBM Performance Optimization With Enhanced RISC (POWER), la cual aún existe.

El PowerPC es mayormente conocido por su uso en las Macs de Apple, pero existen otras estaciones de trabajo IBM basadas en PowerPC y otros fabricantes, al igual que sistemas embebidos basados en PowerPC. El popular sistema TiVo, es uno de ellos.

Al igual que el i386 y el ARM, el PowerPC (PPC) es una arquitectura muy bien soportada en Linux. Este nivel de soporte puede ser particularmente visto en la infinidad de sistemas PPC en los que Linux corre.

Para brindar compatibilidad con los varios hardwares PPC, cada tipo de arquitectura PPC tiene sus propias funciones de bajo nivel agrupadas en archivos designados por arquitectura, por ejemplo, para máquinas CHRP, Gemini y PReP. Sus nombres reflejan las arquitecturas, como chrp_pci.c o gemini_pci.c. En igual manera, existen las cuentas del kernel para las versiones embebidas del PPC, como las series de IBM 4xx y las series Motorola 8xx.

3.4.4 MIPS

El MIPS es el cerebro niño de John Hennessey, mayormente conocido por sus todos sus estudiantes de ciencias computacionales en el mundo por sus libros sobre arquitectura de computadores escritos con David Patterson, y es el resultado del Stanford Microprocessor without Interlocked Pipeline Stages Project (MIPS). MIPS es afamado por ser la base de las estaciones de trabajo y servidores vendidos por SGI y de las consolas de juego como la Nintendo de 64-bits y las Sony PlayStation 1 y 2. Pero también se encuentra en muchísimos sistemas embebidos. Al igual que ARM, la compañía propietaria del MIPS, MIPS Technologies Inc., licencia sus núcleos de CPU a terceros, pero a diferencia de ARM, existen muchos set de instrucciones para implementación, los cuales difieren en varios grados. Implementaciones del MIPS de 32-bits están disponibles desde IDT, Toshiba, Alchemy, y LSI. Implementaciones de 64-bits están disponibles desde IDT, LSI, NEC, QED, SandCraft, y Toshiba.

El port inicial para MIPS de Linux fue principalmente para dar soporte a las estaciones de trabajo basadas en él. Eventualmente, el port también vino para incluir tarjetas o places de desarrollo y sistemas embebidos basados en MIPS. Para acomodar los varios CPUs y sistemas construidos alrededor de él, el esquema de la porción del MIPS en el kernel está dividido en directorios basados en el tipo del sistema en el cual correrá el kernel.

Similarmente, la configuración del kernel para un sistema MIPS dado, está mayormente influenciado por el tipo de tarjeta usado. El tipo actual del chip MIPS en la tarjeta es mucho menos importante.

3.4.5 Hitachi SuperH

En un esfuerzo por mejorar su línea de microcontroladores H8, Hitachi introdujo la línea de procesadores SuperH. Estos manipulan datos de 32-bits internamente y ofrece varios tipos de buses externos. Luego Hitachi formó la compañía SuperH Inc. junto a STMicroelectronics (formalmente SGS-Thomson Microelectronics). SuperH Inc. licencia y encabeza al SuperH de manera muy parecida a ARM y MIPS Technologies Inc. las primeras versiones del SuperH como el SH-1, SH-2, y sus variantes no tenían MMU. Empezando con el SH-3, sin embargo, todos los SuperH posteriores ya poseen MMU. El SuperH es usado en los propios productos de Hitachi, en muchos sistemas embebidos orientados al consumidor electrónico como PDAs y las consolas de juego Sega Saturn y Dreamcast.

3.4.6 Motorola 68000

La familia Motorola 68000 es conocida en la jerga Linux como el m68k y ha sido soportado en las variantes que incluyen MMU y a partir del kernel 2.5 las que no lo poseen. El m68k es el segundo después del x86, como la arquitectura más popular de los 80's. Aparte de ser usado en muchos sistemas grandes y populares por Atari, Apple, y Amiga, y en populares estaciones de trabajo por HP, Sun, y Apollo, el m68k fue también una plataforma estrella a elegir para sistemas embebidos.

3.5 BUSES E INTERFACES

Los buses e interfaces son la fábrica que conecta el CPU con los periféricos que son parte del sistema. Cada bus e interfaz tiene sus propias características, y el nivel de soporte provisto por Linux a ellos varía de acuerdo a éstas. Se discutirán a continuación algunos de los buses en interfaces típicas en los sistemas embebidos.

Linux soporta muchos otros buses como SBus, NuBus, TurboChannel y MCA, pero son específicos de estaciones de trabajo o servidores.

3.5.1 ISA

El bus Industry Standard Architecture (ISA) fue diseñado y ocupado en el núcleo de la arquitectura PC-AT. Es un poco obsoleto y raro y no brindaba las facilidades que otros buses sí, incluyendo la facilidad de mapeo de las direcciones normales de espacio físico del procesador. Su simplicidad sin embargo, favoreció la proliferación para muchos dispositivos para el PC, favoreciendo a su vez el desarrollo de sistemas embebidos mediante PC.

Los dispositivos ISA son accesados mayormente mediante la programación de puertos I/O disponibles en el set de instrucciones del x86, además el kernel no tiene que hacer ningún trabajo para habilitar los drivers de dispositivo para usar este bus.

3.5.2 PCI

El bus Peripheral Component Interconnect (PCI), administrado por el PCI Special Interest Group (PCI-SIG), es posiblemente el bus más popular disponible actualmente. Diseñado para reemplazar al ISA se usa en combinación con muchas diferentes arquitecturas, incluyendo la PPC y la MIPS, para construir diferentes tipos de sistemas, incluyendo dispositivos embebidos.

El PCI requiere software de soporte para habilitarlo y ser usado por los drivers de dispositivo. El soporte en Linux para este bus es completamente amplio y maduro.

3.5.3 PCMCIA

El Personal Computer Memory Card International Association (PCMCIA) es ambos, el nombre común del bus y el nombre de la organización que promueve y mantiene los estándares relacionados. Se han publicado muchos estándares como los actuales CardBus de 32-bits y las especificaciones del USB CardBay. Como parte de un sistema embebido, PCMCIA presta sus flexibilidad y facilidad de extensión. En el iPAQ, por ejemplo, habilita al usuario para conectarse a una LAN usando una tarjeta de red

inalámbrica. En otros sistemas, hace disponible un amplio espacio de almacenamiento permanente a través de las tarjetas CompactFlash. Pero su uso en Linux puede ser confuso debido a una gran cantidad de proyectos dedicados a dar soporte a este bus.

3.5.4 CompactPCI

La especificación CompactPCI fue iniciada por Ziatech y fue desarrollada por miembros del PCI Industrial Computer Manufacturer's Group (PICMG). La especificación CompactPCI provee una abierta y versátil plataforma para aplicaciones de alto desempeño y alta disponibilidad

Técnicamente, el bus CompactPCI es eléctricamente idéntico al bus PCI. En vez de usar conexiones de tipo slot, se usan conectores de pin para conectar las tarjetas CompactPCI.

3.5.5 USB

El Universal Serial Bus (USB) fue desarrollado y es mantenido por un grupo de compañías que forman el USB Implementers Forum (USB-IF). Fue inicialmente desarrollado para reemplazar aquellas interfaces de conexión fragmentadas y lentas como los puertos paralelos y serias tradicionalmente usados para conectar periféricos a las PCs, el USB se ha establecido rápidamente como la interfaz a elegir al proveer comunicación barata, fácil de usar y de alta velocidad. Muchos sistemas embebidos como los SBCs y SoCs lo usan.

3.5.6 I^2C

Inicialmente introducido por Philips para habilitar la comunicación entre componentes dentro de los sets de TV, el bus Inter-Integrated Circuit (I^2C) se puede encontrar en muchos sistemas embebidos de todos los tamaños y propósitos. Como otros muchos buses de menor escala como el SPI y MicroWire, el bus I^2C es un simple bus serial que

permite el intercambio de cantidades limitadas de datos entre los componentes IC (Circuitos Integrados) de un sistema embebido.

3.6 ENTRADA/SALIDA (I/O)

Las entradas y salidas son vitales para el rol de cualquier dispositivo computarizado. Como con otros SO, Linux soporta un amplio rango de dispositivos I/O.

Algunos dispositivos necesarios para el desarrollo de esta tesis son soportados en Linux de dos formas en el kernel, primero por un driver nativo que toma a cargo la conexión directa del dispositivo con el sistema, y la segunda a través de la capa USB para la cual el dispositivo fue agregado. Existen por ejemplo, teclados PS/2 y teclados USB así como mouses PS/2 y mouses USB.

3.6.1 Puerto Serial

El Puerto serial es probablemente el mayor amigo de cada desarrollador de sistemas embebidos (o su peor enemigo, dependiendo de la experiencia pasada con su interfaz un poco complicada). Muchísimos sistemas embebidos son desarrollados y depurados usando un enlace serial RS232 entre el host y el sistema objetivo. La simplicidad de la interfaz RS232 ha reforzado su ampliamente expandido uso y adopción, a pesar de su limitado ancho de banda comparado al de otras alternativas. Nótese que existen otras interfaces seriales detrás del RS232, algunas de las cuales son menos sensibles al ruido y por ende mucho más adaptable a un ambiente industrial. El protocolo de hardware serial sin embargo, no es tan importante como la actual interfaz de programación provista por el hardware serial del dispositivo.

Debido a que el hardware RS232 es una interfaz hardware, el kernel no necesita tener soporte para el mismo. El kernel incluye drivers para los chips que actualmente representan comunicación RS232, los Universal Asynchronous Receiver-Transmitters (UARTs). Los UARTs varían de una arquitectura a otra, aunque algunos UARTs, como el 16550, son usados en más de una arquitectura.

El principal driver serial (UART) en el kernel está en: /drivers/char/serial.c.

3.6.2 Puerto Paralelo

En comparación al Puerto serial, el Puerto Paralelo es raramente usado en sistemas embebidos. En realidad los sistemas embebidos que incluyen este puerto lo hacen debido al uso de alguna impresora antigua o alguna suerte de dispositivo, que con la venida del USB han ido desapareciendo rápidamente. Este puerto es una simple Entrada/Salida Multi-bit.

3.6.3 Módem

Los sistemas embebidos que usan un módem para llamar a un centro de datos son muy comunes. Sistemas de Alarma, máquinas bancarias y hardware de monitoreo remoto son todos ejemplos de sistemas embebidos que necesitan comunicarse con un sistema central para completar sus propósitos primarios. Las ventajas son distintas, pero todos estos sistemas usan módems convencionales para conectarse con el viejo sistema telefónico plano para accesar a un host remoto.

Los módems en Linux son vistos como puertos seriales, al igual que en muchos SO, incluyendo Unix.

3.6.4 Data Acquisition (Adquisición de Datos)

El DAQ es la base de cualquier sistema de automatización de procesos. Cualquier fábrica moderna o laboratorio científico se complementa con equipo DAQ conectado, de una manera u otra, con computadores corriendo software de análisis de datos.

Los eventos que ocurren en el mundo real se miden por medios de transducción que convierten un fenómeno físico en un valor eléctrico. Estos valores son luego muestreados usando hardware DAQ para luego ser accesibles al software.

No existe una interfaz estándar en Unix, o algún otro SO, para interactuar con hardware DAQ. Comedi, la interfaz Linux de dispositivos de control y medida, es el paquete principal para interactuar con DAQs y contiene drivers para un sinnúmero de tarjetas DAQ.

Los DAQ toman actualmente un sinnúmero de formas, pueden ser dispositivos Ethernet, una tarjeta PCI, o usar algún otro tipo de conexión.

Ninguna discusión acerca de DAQ estaría completa sin cubrir algunos de los paquetes comerciales (propietarios) más conocidos usados con y como DAQs como LabVIEW, Matlab, y Simulink. Dada la popularidad de Linux en este campo y su posible uso en tiempo real, los tres paquetes están disponibles y soportados para Linux por sus fabricantes, nótese también que existe un amplio número de paquetes desarrollados bajo GPL que con ventajas o desventajas reemplazan estos paquetes cubriendo casi todas sus funciones.

3.7 ALMACENAMIENTO

Todos los sistemas embebidos requieren al menos una forma de almacenamiento permanente para arrancar incluso las primeras etapas del proceso de booteo. La mayoría de los sistemas, incluyendo los sistemas embebidos Linux, continúan usando el mismo sistema de almacenamiento inicial para el resto de su operación incluso para ejecutar código o para accesar datos. En comparación al software embebido tradicional, sin embargo, el uso de Linux implica grandes requerimientos sobre el hardware de almacenamiento del sistema embebido, ambos en términos de tamaño y organización.

3.7.1 Memory Technology Devices

En terminología Linux, los memory technology devices o MTDs (Dispositivos de Tecnología de Memoria) incluyen a todos los dispositivos de memoria, como la ROM convencional, RAM, flash, y M-Systems' DiskOnChip (DOC). Estos dispositivos tienen sus propias capacidades, particularidades y limitaciones. Para programar y usar un

dispositivo MTD en sus sistemas, desarrolladores de sistemas embebidos tradicionalmente usan herramientas y métodos específicos para ese tipo de dispositivo.

Para evitar tanto como sea posible, teniendo herramientas diferentes para diferentes tecnologías y para brindar capacidades comunes a través de las diferentes tecnologías, el kernel Linux incluye el subsistema MTD. Esto provee una capa unificada y uniforme que permite la combinación de drivers de chips MTD de bajo nivel con interfaces de alto nivel llamadas "módulos de usuario", los mismos que, no se deben confundir con módulos del kernel o cualquier caso de abstracción de usuario.

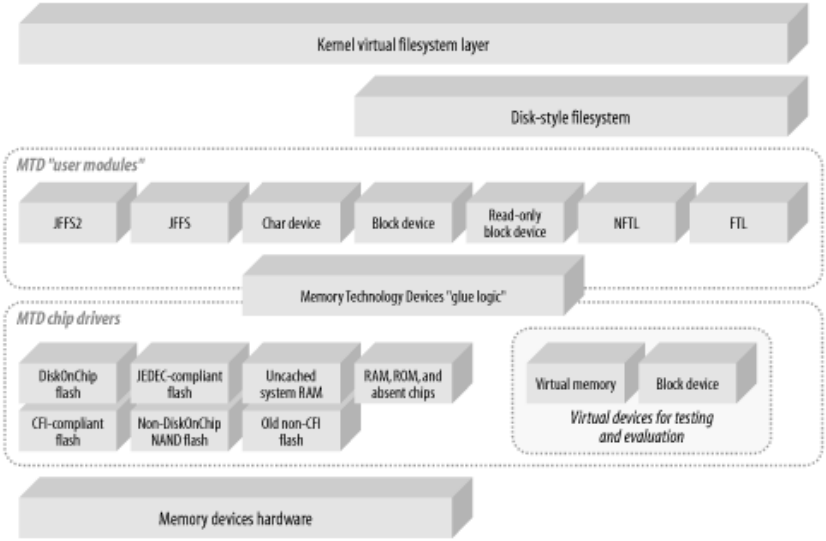


Fig. 3 MTD Subsystem

La siguiente es una lista de algunos de los drivers disponibles en Linux para los MTDs:

- DiskOnChip
- Common Flash Interface (CFI)
- JEDEC
- Non-DOC NAND flash
- Old non-CFI flash
- RAM, ROM, y los absent chips
- Uncached RAM

3.8 NETWORKING DE PROPÓSITO GENERAL

Un creciente número de sistemas embebidos se adjuntan a redes de propósito general. Estos dispositivos, aunque más específicos que muchos otros sistemas computarizados, en muchas maneras, son a menudo exigidos a brindar los mismos servicios de red encontrados en muchos servidores modernos, capacidad que Linux posee sin necesidad de instalar software o drivers adicionales. Aunque existen muchos dispositivos que permiten el Networking, como: IrDA (Infrarrojos), Bluetooth, WiFi, etc, nos enfocaremos principalmente en las redes de tecnología Ethernet.

3.8.1 Ethernet

Inicialmente desarrollado en el centro de investigación PARC de Xerox en Palo Alto, California, es actualmente la tecnología por excelencia usada en redes de computadores y sobre todo en sistemas embebidos.

Linux soporta redes Ethernet de 1, 10 y 100 Megabits. También soporta unos cuantos dispositivos Gigabit Ethernet.

3.9 BOOTLOADERS

Como se dijo anteriormente, se pueden usar muchos bootloaders con Linux en hardware vario. Aquí se discutirá los bootloaders más populares y versátiles para cada arquitectura centrándonos específicamente en aquellos que soportan ARM, debido a que como se verá más adelante, la tesis se va a basar en esta arquitectura. Hay muchos autores que hacen una distinción entre bootloader y monitor. En esos casos, el bootloader se puede definir simplemente como el componente que arranca un dispositivo y lanza el software principal, un monitor provee además de dichas capacidades, una interfaz de línea de comando que puede ser usada para depuración, lectura/escritura de memoria, reprogramación de flash, configuración, etc. En este trabajo nos referiremos al bootloader como tal con o sin capacidad de monitor.

He aquí una lista de algunos populares bootloaders:

			Arquitecturas					
Bootloader	**Monitor**	**Descripción**	**x86**	**ARM**	**PowerPC**	**MIPS**	**SuperH**	**m68k**
LILO	No	El principal bootloader para Linux	X					
GRUB	No	Sucesor GNU de LILO	X					
ROLO	No	Carga Linux desde ROM sin BIOS	X					
Loadlin	No	Carga Linux desde DOS	X					
Etherboot	No	Cargador ROMable para sistemas de booteo a través de tarjetas Ethernet	X					
LinuxBIOS	No	Reemplazo Linux para el BIOS	X					
Compaq's bootldr	Si	Cargador versátil para Compaq iPAQ		X				
blob	No	Cargador del LART hardware project		X				
PMON	Si	Usado en Agenda VR3				X		
sh-boot	No	Cargador del LinuxSH Project					X	
U-Boot	Si	Cargador de PPCBoot y ARMBoot	X	X	X			
RedBoot	Si	Basado en eCos	X	X	X	X	X	X

Tabla 1. Bootloaders

3.9.1 Compaq's bootldr

Aunque inicialmente desarrollado solamente para el Compaq iPAQ, el Compaq's bootldr actualmente soporta al Assabet de Intel y al Jornada 720 de HP. Aunque está muy limitado en el hardware que soporta, provee un enriquecido set de comandos y es capaz de cargar kernels desde particiones MTD JFFS2.

3.9.2 blob

Blob fue introducido como el bootloader para el LART hardware project. Desde su lanzamiento, ha sido portado a muchos otros sistemas basados en ARM, incluyendo al Assabet y al Brutus de Intel, las tarjetas Nesa y Shannon.

3.9.3 U-Boot

Aunque hay unos cuántos bootloaders, "Das U-Boot" el bootloader universal, es seguramente el más rico, más flexible y más activamente desarrollado bootloader open source disponible. Es actualmente mantenido por Wolfgang Denk de DENX Software Engineering, y es contribuido por un amplio número de desarrolladores. U-Boot está basado en los proyectos PPCBoot y ARMBoot.

Entre otras cosas, U-Boot es capaz de bootear un kernel a través de TFTP, desde un disco IDE o SCSI, y desde un DOC. También, incluye, soporte sólo lectura para JFFS2.

3.9.4 RedBoot

RedBoot está supuesto a ser la siguiente generación de bootloaders de Red Hat, reemplazando a CygMon y GDB con un firmware que soporta un amplio rango de software. Aunque Red Hat ha parado el desarrollo activo de eCos, el SO sobre el que RedBoot está basado, eCos ha sido ahora licenciado bajo GPL y continua siendo mantenido por algunos desarrolladores del núcleo de Red Hat. Es actualmente un bootloader muy poderoso, bien soportado y documentado.

CAPITULO IV

TARJETA CS-E9302

4.1 TARJETA SBC DE DESARROLLO OLIMEX CS-E9302

Considerando las características del microcontrolador a usarse, se eligió una tarjeta acorde a las necesidades de esta tesis, y entre abundantes opciones se eligió la CS-E9302 de Olimex, importada desde Chile y manufacturada en Bulgaria.

Descripción

La CS-E9302 es una poderosa tarjeta de desarrollo para microcontroladores.

Se basa en un Cirrus Logic EP9302 ARM920T a 200 MHz con dos host USB, dos puertos RS232, Ethernet de 100 Mbps, tarjeta SD-MMC, NOR Flash de 16 MB, SDRAM de 32 MB, Transceptor IrDA (Infrarrojo), JTAG, UEXT, EXT, conectores ADC para periféricos adicionales, etc.

Características

- MCU: EP9302 ARM920T 200Mhz, caché 16+16KB de instrucciones y datos, MMU, SDRAM, SRAM, controlador de bus externo FLASH a 100 MHz, 100 Mbps Ethernet MAC, dos UARTS, dos puertos USB 2.0, IrDA, ADC, SPI, I^2S Audio, AC'97, DMA 12 ch, RTC, dual PLL, WDT, 2- 16 bit, 1- 32 bit TIMERS, boot ROM, controlador de interrupciones.

- SDRAM Externa de 32MB (16MB x16bit) 7.5 ns /133 MHz

- Flash Externa de 16MB (8MB x16 bit) 80 ns.

- ETHERNET 10/100 PHY KS8721BL

- 2 x RS232 drivers y conectores

- 2 x USB conectores host

- Tarjeta SD/MMC

- Transceptor IrDA

- Conector UEXT con I^2C, SPI, RS232.

- Suplemento de energía para conexión de módulos add-on como RF link, MP3, etc.

- Interfaz JTAG

- Puerto de extensión ADC

- Plug in jack, para fuente de energía

Especificaciones Físicas

La CS-E9302 no debe ser sometida a potenciales electrostáticos altos, está diseñada a doble capa y debe ser manipulada con cuidado. Las dimensiones de la tarjeta son 110 x 90 mm. Esta tarjeta es ideal para desarrollo de sistemas embebidos. El consumo de la tarjeta es de alrededor de 400 a 450 mA con CPU a 200 MHz de reloj y 100 MHz de bus de sistema.

Requerimientos de Uso de la Tarjeta

Cables

- Un cable DB9 hembra-hembra de tipo null modem para conexión serial COM con el PC.

- Cable LAN cruzado para conexión Ethernet con el PC o Cable LAN directo para conexión con switch o router.

Hardware

- Fuente de alimentación regulada de +5 V/1 A

- ARM-JTAG, ARM-USB-OCD, ARM-USB-TINY o alguna herramienta compatible si se desea trabajar con JTAG, aunque no es necesario si se va a trabajar con un RTOS.

Software

- En vista que el objetivo principal de esta tesis es el desarrollo de un sistema Linux Embebido, los requerimientos de software se irán detallando más adelante conforme avance el proyecto y sus propósitos.

4.1.1 Fotos

Cara de Componentes

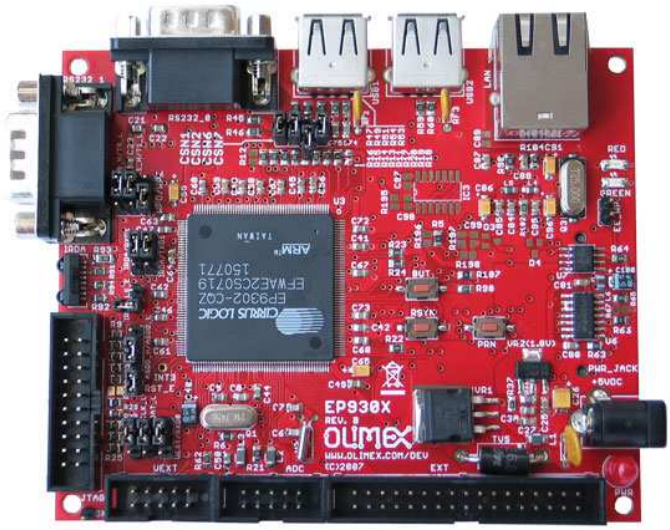


Fig. 4 Vista Superior CS-E9302

Cara de Soldadura

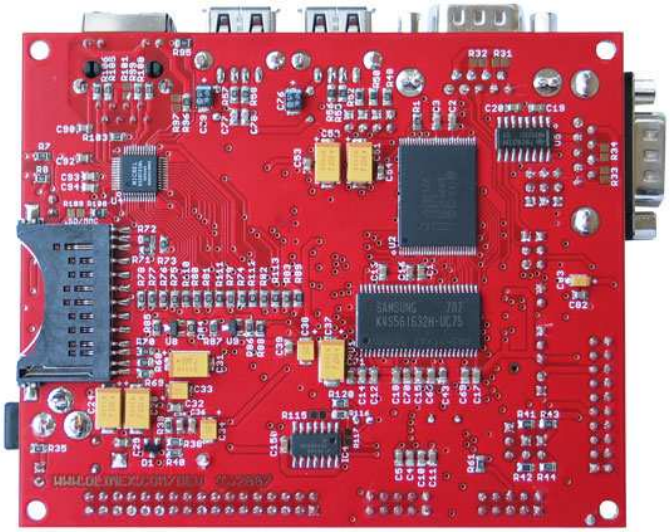


Fig. 5 Vista Inferior CS-E9302

4.1.2 Layout

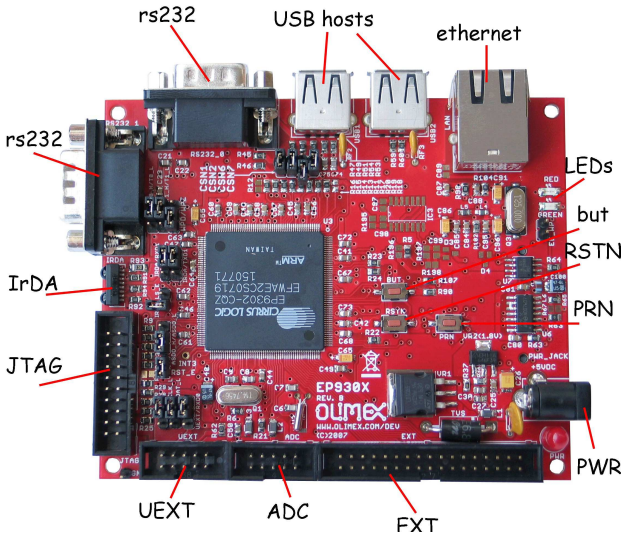


Fig. 6 Board Layout

4.1.3 Microcontrolador Cirrus Logic EP9302 (SoC)

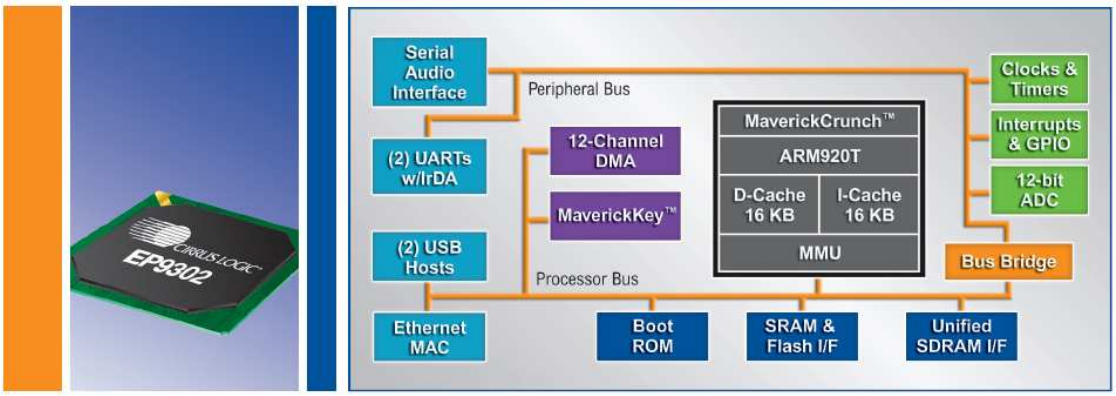


Fig. 7 Diagrama de bloques del EP9302

Como se muestra en la Fig. 7, se trata de un microcontrolador System-on-Chip (Sistema en Chip) con procesador de arquitectura ARM9, con un amplio rango de aplicaciones.

Provee un diseño ágil que incluye un procesador ARM920T a 200MHz con capacidad de ser usado en aplicaciones a nivel industrial y consumidor.

El procesador está basado en RISC de bajo consumo, operando desde los 1.8 V, mientras las I/O operan a 3.3 V con un consumo entre los 100 mW hasta los 750 mW, dependiendo de la velocidad.

Características

- Procesador ARM920T a 200 MHz
  - 16 KB de cache de datos y 16 KB de cache de instrucciones
  - MMU
  - Bus de sistema a 100 MHz

- Motor Matemático *MaverickCrunch*
  - Punto flotante, enteros y señal de procesamiento de instrucciones
  - Optimizado para algoritmos de compresión digital de música
  - Interbloqueo de hardware permitido en la codificación de línea

- IDs *MaverickKey* para Manejo de Derechos Digitales o Diseño de Seguridad IP
  - ID única de 32-bits
  - ID randómica de 128-bits

- Interfaces de Periféricos Integradas
  - Cinco entradas A/D con resolución de 12-bits
  - MAC Ethernet de 1/10/100 Mbps
  - Dos puertos USB 2.0 de alta velocidad (OHCI)
  - Dos UARTs (tipo 16550), incluyendo soporte soft módem
  - Interfaz IrDA, modo lento
  - Puerto SPI
  - Interfaz AC'97
  - Interfaz I^2S

- Opciones de Memoria Externa
  - Interfaz SDRAM de 16-bits, arriba de dos bancos

- SRAM/Flash/ROM I/F de 16/8 bits
- Interfaz serial EEPROM

- Periféricos Internos
  - Reloj de tiempo real con trim por software
  - 12 canales DMA para transferencia de datos que maximiza el desempeño del sistema
  - Boot ROM
  - PLLs duales controlan todo el dominio del reloj
  - Temporizador Watchdog
  - Dos temporizadores de propósito general de 16-bits
  - Temporizador de propósito general de 32-bits
  - Temporizador de depuración de 40-bits

- Entradas y Salidas de Propósito General (GPIO)
  - 16 GPIOs extendidas incluyendo capacidad de interrupciones
  - 8 GPIOs adicionales opcionales multiplexadas sobre periféricos

Descripción

El EP9302 posee un avanzado procesador ARM920T de 32-bits con un pipeline de cinco etapas, entregando un desempeño impresionante con un bajo consumo de energía. Las dos caché incluidas de 16 KB para datos e instrucciones proveen una latencia de ciclo de cero a los datos y programas, o pueden ser bloqueadas para brindar acceso garantizado y sin latencia a datos e instrucciones críticas. Para aplicaciones con tamaño de memoria de instrucciones reducido, el set de instrucciones comprimido *Thumb* brinda un diseño de espacio eficiente que maximiza el uso de memoria externa de instrucciones.

Motor Matemático MaverickCrunch de Procesamiento Ultrarrápido

El Motor *MaverickCrunch* es un avanzado coprocesador matemático de modo mixto, que acelera enormemente las capacidades de procesamiento de los enteros y puntos

flotantes de precisión simples y dobles, del núcleo del procesador ARM920T. El motor simplifica las tareas de programación de usuario final usando instrucciones coprocesadas predefinidas, utilizando herramientas de compilación estándar para ARM, y requiriendo sólo una sesión de depuramiento para el sistema entero.

Es más, el diseño integrado brinda un flujo de instrucciones simple y la ventaja de la latencia cero para instrucciones en caché. Para emular esta capacidad, las soluciones de otros fabricantes agregan un DSP al sistema, el cual requiere por separado sets de herramientas para compilar, enlazar y depurar. Este DSP adicional requiere que el programador escriba dos programas por separado y depurarlos simultáneamente, lo cual suele resultar en frustración y costosos retrasos.

La instrucción de multiplicar y acumular enteros de ciclo simple en el motor, le permite al EP9302 ofrecer una velocidad y desempeño únicos mientras se codifican formatos de audio y video digital, procesamiento de datos vía Ethernet y desarrollar otro tipo de cómputos matemáticos intensivos y funciones de procesamiento de datos en electrónica comercial e industrial.

Controlador Integrado Multiplexado A/D de cinco entradas

El EP9302 integra un señalizado conversor analógico a digital con una terminación multiplexada que es controlado a través de una matriz de switcheo configurada por software. Esta matriz permite el ruteo de las cinco señales de entrada hacia la entrada A/D de muestreo o a las entrada +REF Y –REF.

Si entradas REF externas no son requeridas las entradas A/D REF pueden ser internamente switcheadas a VDD y GND permitiendo que sean muestreadas las cinco entradas.

Al leer un resultado muestreado se habilita un comando de conversión para la siguiente muestra. Estas capacidades son especialmente útiles en control de procesos y aplicaciones de instrumentación así como también monitoreo de temperatura o voltaje.

ID Única MaverickKey, que asegura Contenido Digital y Diseños OEM

Las IDs únicas programadas en hardware brindan una excelente solución para el crecimiento concerniente al contenido y comercio web seguro. Con la seguridad en el Internet jugando un rol importante en el envío de medios digitales, como libros o música, los métodos tradicionales de software se vuelven rápidamente desconfiables.

Las IDs MaverickKey proveen a los OEMs con un método de utilizar IDs hardware específicas para mecanismos de Manejo de Derechos Digitales (DRM).

Ambos, una ID específica de 32-bits así como también una ID randómica de 128-bits se programan dentro del EP9302 a través del uso de tecnología de prueba láser.

Estas pueden ser luego usadas para coincidir de manera segura contenido con derechos de autor con el ID del dispositivo final, que es alimentado por el EP9302 y luego envía la información con derechos de autor sobre una conexión segura.

Además, las transacciones seguras se pueden beneficiar emparejando las IDs del dispositivo con las IDs del servidor.

Las IDs MaverickKey pueden también ser usadas por OEMs y diseñar casas que las protejan contra diseños piratas seteando rangos para IDs únicas.

Dos Puertos USB 2.0 integrados, Host de alta velocidad con Transceptores

Completamente compatibles con la especificación OCHI USB 2.0 Full Speed a 12 Mbps, los puertos pueden ser usados para brindar conexión a un sinnúmero de dispositivos externos incluyendo a los de almacenamiento masivo, dispositivos externos portables como reproductores de audio o cámaras, impresoras o hubs USB.

Naturalmente estos puertos también soportan el estándar USB de baja velocidad. Esto brinda la oportunidad de crear un amplio arreglo de configuraciones de sistema flexibles.

MAC Ethernet integrado, reduciendo costos de BOM

El EP932 integra un Ethernet Media Access Controller (MAC) de 1/10/100 Mbps. Con una simple conexión a un PHY externo basado en MII, un sistema basado en el EP9302 fácilmente tendrá una conexión a Internet de alta velocidad.

Sonido de Alta Calidad enviado en múltiples Formatos de Audio

El EP9302 entrega soporte para interfaces seriales SPI, I^2S y AC'97. El puerto AC'97 soporta múltiples CODECs para salida efectiva de audio estéreo. Cirrus Logic provee algoritmos de codificación y decodificación de audio de alto desempeño para un sinnúmero de formatos populares de audio incluyendo Windows Media Audio, MP3, AAC, etc.

Interfaz de Memoria de Propósito General (SDRAM, SRAM, ROM & FLASH)

Una de las características del EP9302 es un modelo de dirección de memoria unificado en el cual todos los dispositivos de memoria son accesados sobre un bus de dirección/datos común.

El controlador de memoria SRAM soporta dispositivos de 8 y 16 bits y acomoda una ROM de booteo concurrentemente con una memoria SRAM de 8 o 16 bits.

Múltiples Mecanismos de Booteo, que incrementan la Flexibilidad

El procesador incluye una memoria de booteo ROM de 16 KB para levantar configuraciones estándar.Opcionalmente el procesador puede ser arrancado desde la memoria FLASH, sobre la interfaz serial SPI o a través de un UART.

Esta flexibilidad en el arranque hace fácil diseñar sistemas controlados por el usuario y actualizables en campo.

Abundantes I/Os de Propósito General para construir Sistemas Flexibles

El EP9302 incluye ambos, pines de I/O estándar y expandidas (GPIO).

Las 16 diferentes GPIOs expandidas pueden ser manualmente configuradas como entradas, salidas o entradas habilitadas de interrupción.

Hay adicionalmente 8 GPIOs estándar, que pueden ser individualmente configuradas como entradas, salidas o entradas de drenado abierto.

Las GPIOs estándar son multiplexadas con pines de funciones de periférico, así el número disponibles depende de la utilización de periféricos.

Juntas, las GPIOs estándar y extendidas facilitan un diseño de sistema fácil con periféricos externos no integrados en el EP9302.

Aplicaciones

- Radios Internet
- Terminales de Puntos de Venta (PoS)
- Controles Industriales y de Construcción
- Rockolas Electrónicas
- Sistemas de Control Telemático
- Sistemas de Seguridad Biométricos
- Máquinas de Lotería
- Equipo de Fitness
- Sistemas de Seguridad
- Cámaras de Internet
- Mezcladores MP3
- Módems GSM
- VoIP

4.1.4 Circuitería y Jumpers

Circuito de Reset

El chip EP9302 se puede resetear a través del pin PRSTN (power on reset) o a través del pin de reseteo común de drenado abierto (user reset) o RSTON.

El PRSTN es un Power on Reset realizado con un circuito de reset MCP130T con el típico umbral de 2.9 V. En la tarjeta se presenta un botón PRN el cual es usado para un reset Power-on Reset manual.

El reset de usuario RSTON es común para el microcontrolador EP9203, el chip PHY, la NOR flash, el JTAG y el circuito de supervisión MCP130. Se puede generar desde el pin 15 JTAG, botón RSTN, circuito de reset CS9302 o MCP130T.

Circuito de Reloj

El EP9302 genera su propio reloj interno con PLL y usa un simple cristal de 14.7456 MHz conectado al pin 119 (XTAL0) y el pin 118 (XTAL1).

El Reloj de Tiempo Real opera desde un cristal de 32.768 KHz.

Descripción de Jumpers

Jumpers de Configuración de Boot

Los controles de configuración de hardware se definen por un set de pines de dispositivo que son apestillados dentro de los bits de control de configuración sobre la afirmación del reset del chip cuando se activan PRSTN o RSTON. Las diferentes configuraciones de bits hardware definen el comportamiento del Watchdog, modo de booteo (interno o externo), sincronismo en el boot, y ancho de booteo externo. Los pines apestillados son:

EECLK, selecciona booteo interno o externo el jumper correspondiente es EECLK_L. cuando el jumper se cierra el log es "0", de otra manera el log es "1".

EEDAT, debe ser puesto a "1". El jumper correspondiente es EEDAT_L. Cuando el jumper se cierra EEDAT vale "0", de otra manera vale"1".

TEST[1:0], selecciona el modo de booteo. Los jumpers correspondientes son T0_H/T0_L y T1_H/T1_L. Cuando el jumper T0_H/T0_L se cierra hacia el lado de T0_H TEST0 vale "1". Cuando el jumper T0_H/T0_L se cierra hacia el lado de T0_L, TEST0 vale "0". Cuando el jumper T1_H/T1_L se cierra hacia el lado de T1_H, TEST1 vale "1". Cuando el jumper T1_H/T1_L se cierra hacia la cara de T1_L, TEST1 vale "0".

ASDO, selecciona booteo síncrono o asíncrono, los 3 pines correspondientes al jumper son ASDO_H/ASDO_L. Si el jumper se cierra hacia el lado de ASDO_H, ASDO vale "1". Hacia el otro lado vale "0".

CSN[7:6], selecciona el tamaño de boot externo, CSN6 y CSN7 son los jumpers que definen el nivel lógico de estos pines. Si están cerrados el pin correspondiente vale "0".

EECLK	EEDAT	TEST1	TEST0	ASDO	CSN[7:6]	Configuración de Booteo
0	1	0	0	1	00 01	Booteo Externo desde el espacio de memoria Sync seleccionado por DevCfg3 a través del controlador SDRAM. El tipo de medio debe ser a su vez SROM o SyncFLASH. La selección del tamaño SRAM es determinado por el apestillado del valor de CSN[7:6]: 8-bit SFLASH 16-bit SROM
0	1	0	0	0	00 01	Booteo externo desde el espacio de memoria Async seleccionado por nCS0 a través del Controlador de Memoria Síncrona. La selección del tamaño de SRAM se determina por el apestillado del valor de CSN[7:6]: 8-bit SRAM 16-bit SRAM

1	1	0	1	X	01	Booteo serial de 16-bits
1	1	0	0	1	00 01	Booteo Externo desde una NOR-FLASH. La selección del tamaño del bus se determina por el valor del apestillamiento de CSN[7:6]: 8-bit 16-bit–DEFAULT POSITION–NORMAL BOOT
1	1	0	0	0	00 01	Boot Interno desde el chip ROM. La selección del tamaño de la ROM s determina por el valor del apestillamiento de CSN[7:6]: 8-bit 16-bit

Tabla 2. Configuraciones de booteo

Solo dos selecciones son funcionales para el usuario, Booteo Normal y Booteo Serial. Los otros modos se reservan para uso de fábrica solamente.

La selección marcada en azul es la posición por defecto de los jumpers.

Configuración de Jumpers de Watchdog

CSN1, deshabilita el temporizador reset del Watchdog. Si el jumper CSN1 se cierra en el pin correspondiente (CSN1) vale "0", de otra manera vale"1".

CSN2, deshabilita la duración del reset del Watchdog. Si el jumper CSN2 se cierra en el pin correspondiente (CSN2) vale "0", de otra manera vale "1".

CSN1	CSN2	Opción de Inicio
0	0	Watchdog deshabilitado, Duración de reset deshabilitado
0	1	Watchdog deshabilitado, Duración de reset activo
1	0	Watchdog activo, Duración de reset deshabilitado
1	1	Watchdog activo, Duración de reset activo

Tabla 3. Watchdog Jumpers

La selección marcada en azul es la posición por defecto de los jumpers.

Jumpers de Uso General

WPD/WPE, define el estado de protección de escritura de la NOR-FLASH U2(JS28F128J3C120) el estado de WPD (1-2 en corto) habilita la escritura de la flash, el estado WPE (2-3 en corto) deshabilita la escritura de la flash. La posición por defecto es 1-2 en corto.

Fig. 8 WPD

EE_WP, cuando el jumper EE_WP se cierra deshabilita la escritura de la EEPROM U7(AT25F1024). La posición por defecto es abierto.

Fig. 9 EE_WP

IRDA/RXD1, el jumper IRDA/RXD1 define conexión RXD1 (pin 110 del MCU). Cuando el jumper es instalado en el estado IRDA (2-3 en corto), entonces la función del RDX1 del MCU es conectado al módulo IrDA. Si se selecciona el estado RXD1 (1-2 en corto), el pin RXD1 está conectado a través del driver RS232 U5 al conector RS232_1. La posición por defecto es 1-2 en corto.

Fig. 10 RXD1

IRDA/TXD1, el jumper IRDA/TXD1 define la conexión TXD1 (pin 114 del MCU). Cuando el jumper está instalado en estado IRDA (2-3 en corto), entonces la función TXD1 del MCU es conectado al módulo IRDA. Si el estado TXD1 se selecciona (1-2 en

corto), el pin TXD1 está conectado a través del driver U5 RS232 al conector RS232_1. La posición por defecto es 1-2 en corto.

Fig. 11 TXD1

IRDA_LS_E, el jumper IRDA_LS_E se usa si se debe seleccionar el modo baja velocidad del chip IRDA. Cuando el jumper se cierra, el usuario habilita el modo baja velocidad. La posición por defecto es abierto.

Fig. 12 IRDA alta velocidad

RST_E, cuando el jumper RST_E está cerrado, el pin15 del conector JTAG está conectado a la línea RSTN (User Reset) o permite al JTAG reseteado. La posición por defecto es cerrado

Fig. 13 RST_E

UEXT/RXD0, el jumper UEXT/RXD0 define conexión RXD0 (pin 109 del MCU).cuando el jumper se instala en estado UEXT, entonces las función RXD0 del MCU es conectado al conector UEXT.

Si el estado RXD0 se selecciona, el pin RXD0 está conectado a través del driver U5 RS232 al conector RS232_0. La posición por defecto es 1-2 en corto.

Fig. 14 UEXT/RXD0

ENTRADA/SALIDA

Dos botones de reset con nombres RSTN y PRN conectados al pin 124 (RSTON) y al pin 125 (PRSTN) del chip EP9302.

Un botón de usuario con nombre BUT conectado al pin 201 HGPIO[4].

Dos LEDs, rojo conectado al pin 98 RDLED y verde conectado al pin 97 GRLED.

LED rojo de Fuente de alimentación con nombre PWR que indica que la fuente está conectada.

Descripción de Conectores

USB1, USB2

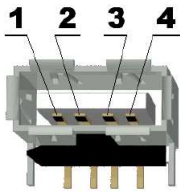


Fig. 15 Pines USB

USB1		USB2	
Pin #	**Signal Name**	**Pin #**	**Signal Name**
1	+5V	1	+5V
2	USBM0	2	USBM2
3	USBP0	3	USBP2
4	GND	4	GND

Tabla 4. Conectores USB

- USBM0 y USBP0 forman la entrada/salida diferencial del USB host 0.
- USBM1 y USBP1 forman la entrada/salida diferencial del USB host 1.

La USB Open Host Controller Interface (Open HCI) provee puertos de comunicaciones a alta velocidad a un baud rate de 12 Mbps, por encima de 127 dispositivos USB (impresora, mouse, cámara, teclado, etc.) y hubs USB se pueden conectar en el USB host en la topología USB "tieredstart". El puerto USB está de acuerdo a las especificaciones de USB 2.0 y soporta ambas baja velocidad (1.5 Mbps) y alta velocidad (12 Mbps).

RS232_0, RS232_1

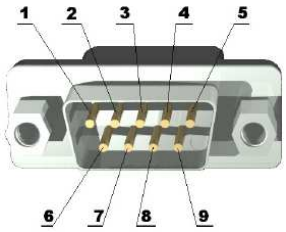


Fig. 16 Pines DB9

RS232_0		RS232_1	
Pin #	**Signal Name**	**Pin #**	**Signal Name**
1	NC	1	NC
2	RXD_0	2	RXD_1
3	TXD_0	3	TXD_1
4	pin 6	4	pin 6
5	GND	5	GND
6	pin 4	6	pin 4
7	pin 8	7	pin 8
8	pin 7	8	pin 7
9	NC	9	NC

Tabla 5. Conectores RS232

El UART0 soporta un flujo de bits arriba de los 115.2 Kbps, soporta HDLC e incluye 16 bytes FIFO para recibir y 16 byte FIFO para transmitir.

El Shell del sistema Linux embebido a instalarse será accesible desde el conector DB9 macho en RS232_0. La configuración del terminal será: **57600 Kbps, 8 data bits, no parity, 1 stop bit, no flow control**.

El UART1 contiene un codificador IrDA operando en modo lento (arriba de 115 Kbps), medio (0.576 o 1.152 Mbps), o rápido (4 Mbps). También tiene 16 bytes FIFO para recibir y 16 bytes FIFO para transmitir.

- TXD_0, Transmit Data 0 (Salida). Esta es la salida serial asíncrona de datos (RS232) para el cambio de registro en el controlador UART0 (éste pin es entrada para el RS232 y entrada para el LPC2478)

- RXD_0, Receive Data 0 (Entrada). Esta es la entrada serial asíncrona de datos (RS232) para el cambio de registro en el controlador UART0 (éste pin es salida para el RS232 y entrada para el LPC2478).

- TXD_1, Transmit Data 1 (Salida). Esta es la salida serial asíncrona de datos (RS232) para el cambio de registro en el controlador UART1 (éste pin es entrada para el RS232 y entrada para el LPC2478).

- RXD_1, Receive Data 1 (Entrada). Esta es la entrada serial asíncrona de datos (RS232) para el cambio de registro en el controlador UART1 (éste pin es salida para el RS232 y entrada para el LPC2478).

ETHERNET:

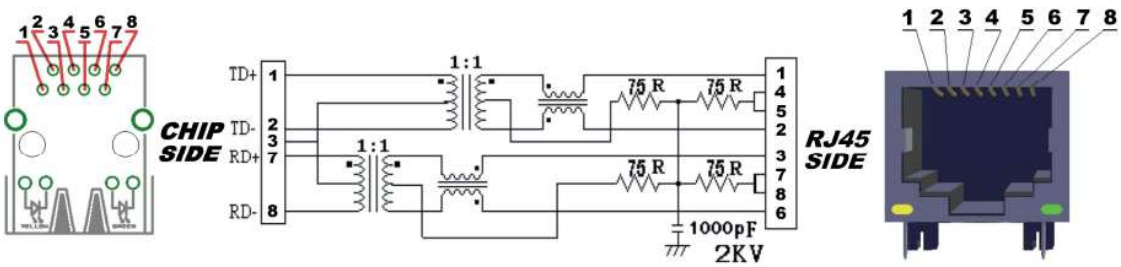


Fig. 17 Circuito Ethernet

La CS-E9302 posee un subsistema MAC físico completo. El EP9302 soporta 1/10/100 Mbps de transferencia de datos. El subsistema MAC cumple con la topología ISO/TEC 802.3 para un único medio compartido con varias estaciones.

Están soportadas múltiples PHYs que cumplen con MII. La capa PHY está hecha con el chip MICREL PHY KS8721BL.

Pin #	Signal Name Chip Side	Pin #	Signal Name Chip Side
1	TD+	5	NC
2	TD-	6	2.5V
3	2.5V	7	RD+
4	NC	8	RD-

Tabla 6. Conector RJ45

LED	Color	Uso
Izquierda	Amarillo	Actividad
Derecha	Verde	100 Mbps (Half/Full dúplex)

Tabla 7. LEDs Ethernet

- TD- OutputDifferential. Esta señal es una salida del MCU.
- TD+ OutputDifferential. Esta señal es una salida del MCU.
- RD- InputDifferential. Esta señal es una entrada del MCU.
- RD+ InputDifferential. Esta señal es una entrada del MCU.

UEXT:

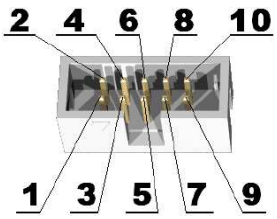


Fig. 18 Pines UEXT

Pin #	Signal Name	Pin #	Signal Name
1	3.3V	2	GND
3	TXD0	4	RXD0_UEXT
5	H2	6	H3
7	SSPRX1	8	SSPTX1
9	SCLK1	10	SFRM1

Tabla 8. UEXT

El UEXT es un conector universal OLIMEX con 3.3 V de alimentación e interfaz UART, SPI más los puertos del MCU, HGPIO[2] y HGPIO[3] conectados a los pines del UEXT 5 y 6. Otros dispositivos o módulos se pueden conectar con estas interfaces a través del UEXT, por ejemplo:

- MOD-NRF24L - http://www.olimex.com/dev/mod-nrf24L.html
- MOD-RFID125- http://www.olimex.com/dev/mod-rfid125.html
- MOD-MP3 - http://www.olimex.com/dev/mod-mp3.html

ADC:

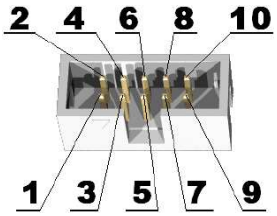


Fig. 19 Pines ADC

Pin #	Signal Name	Pin #	Signal Name
1	GND	2	GND
3	ADC0	4	ADC1
5	ADC2	6	ADC3
7	ADC4	8	ADC_VDD(3.3V)
9	GND	10	GND

Tabla 9. ADC

El bloque ADC consiste en un conversor Analógico a Digital de 12-bits con una entrada analógica multiplexora. El multiplexor puede seleccionarse para medir voltaje de baterías y otros voltajes misceláneos en los pines externos de medición. La señal debe estar entre los 0 a 3.3 V, el ADC se completa con la posibilidad de interrupción.

Interfaz de Tarjetas SD/MMC

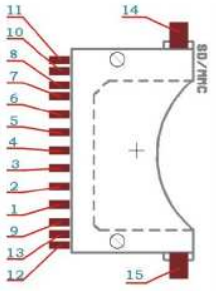


Fig. 20 Slot SD/MMC

Pin #	Signal Name
1	CD_MMC_E/SFRM1
2	SSPTX1
3	GND
4	VCC (470nH to 3.3V)
5	SCLK1
6	GND
7	SSPRX1
8	10K to VCC
9	10K to VCC
10	WP (33K to VCC)
11	WP (2K to GND)
12	CP (2K to GND)
13	WP (33K to VCC)
14	2K to GND
15	2K to GND

Tabla 10. SD/MMC

EXT

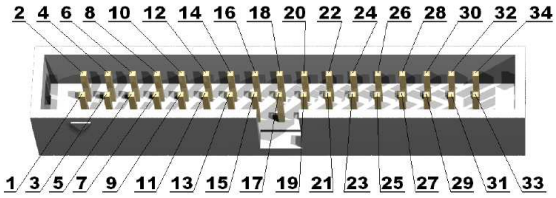


Fig. 21 Pines EXT

Pin #	Signal Name	Pin #	Signal Name
1	CGPIO[0]	2	EGPIO[15]
3	EGPIO[14]	4	EGPIO[13]
5	EGPIO[12]	6	EGPIO[11]
7	EGPIO[10]	8	EGPIO[9]
9	EGPIO[8]	10	EGPIO[7]

11	EGPIO[6]	12	EGPIO[5]
13	EGPIO[4]	14	EGPIO[3]
15	EGPIO[2]	16	EGPIO[1]
17	EGPIO[0]	18	FGPIO[1]
19	FGPIO[2]	20	FGPIO[3]
21	HGPIO[2]	22	HGPIO[3]
23	HGPIO[4]	24	HGPIO[5]
25	ABITCLK	26	ASYNC
27	ASDI	28	ASDO
29	ARSTN	30	+5V(Out)
31	3.3V(Out)	32	1.8V(Out)
33	GND	34	PRN(Power on Reset)

Tabla 11. EXT

En este conector se encuentran disponibles todas las GPIOs del EP9302 las cuales no están siendo usadas para otros propósitos. Con éstas se pueden agregar módulos hardware add-on para ser controlados con la CS-E9302 como GSM/GPS, tarjetas input/output add-on.

JTAG

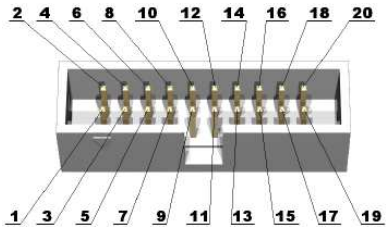


Fig. 22 Pines JTAG

Pin #	Signal Name	Pin #	Signal Name
1	+3.3V	2	+3.3V
3	TRST	4	GND
5	TDI	6	GND
7	TMS	8	GND
9	TCK	10	GND

11	TCK/GND	12	GND
13	TDO	14	GND
15	RSTN/NC	16	GND
17	NC	18	GND
19	NC	20	GND

Tabla 12. JTAG

El conector JTAG permite al software de depuración comunicarse vía puerto JTAG (Joint Test Action Group) directamente al núcleo. Las instrucciones deben ser insertadas y ejecutadas por el núcleo permitiéndole a la memoria del MCU ser programada con código ejecutado paso a paso. Cumple con el estándar IEEE 1149.1 - 1990 Standard Test Access Port.

- TDI, Test Data In. Esta es la entrada serial de datos para cambio de registro

- TDO, Test Data Out. Esta es la salida serial de datos para cambio de registro. Los datos se cambian en el dispositivo en el límite negativo de la señal TCK.

- TMS, Test Mode Select. El pin TMS selecciona el estado siguiente en la máquina de estado TAP.

- TCK, Test Clock. Esto permite el cambio interno de los datos, en los pines TMS y TDI. Es un reloj de disparo positivo con las señales TMS and TCK que definen el estado interno del dispositivo.

- TRST, Test Reset. Esta señal resetea el controlador JTAG

- TCK, Clock. Esta es una señal de sincronización usada por el conector JTAG para saber si está listo para recibir/transmitir.

- RSTN, Reset. Esta señal resetea el MCU.

Este conector comparte el esquema ARM JTAG por defecto.

PWR

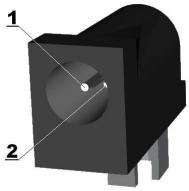


Fig. 23 Power Jack

Pin #	Signal Name
1	+5VDC ONLY
2	GND

Tabla 13. Power Jack

Dimensiones Mecánicas

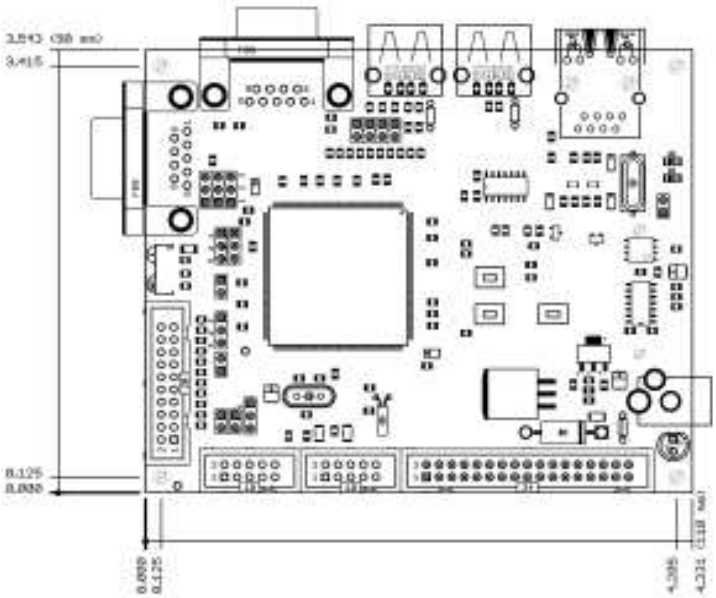


Fig. 24 Dimensiones

CAPITULO V

IMPLEMENTANDO LINUX EMBEBIDO EN LA CS-E9302

5.1 HERRAMIENTAS DE DESARROLLO

Muchos desarrolladores de software mainstream, y desarrolladores de sistemas embebidos, necesitan compiladores, enlazadores, intérpretes, entornos de desarrollo integrado, y otras herramientas de ese tipo.

Las herramientas de desarrollo embebido son diversas y diferentes, sin embargo, aquellas se corren típicamente sobre una plataforma mientras se construyen en otra.

Esta es la razón por la que estas herramientas son denominadas herramientas de desarrollo de compilación cruzada.

En este capítulo trataremos de explicar la construcción del entorno de desarrollo Linux y este tipo de herramientas además de algunas alternativas referentes a las mismas. A continuación se muestra el esquema de Desarrollo/Booteo del sistema:

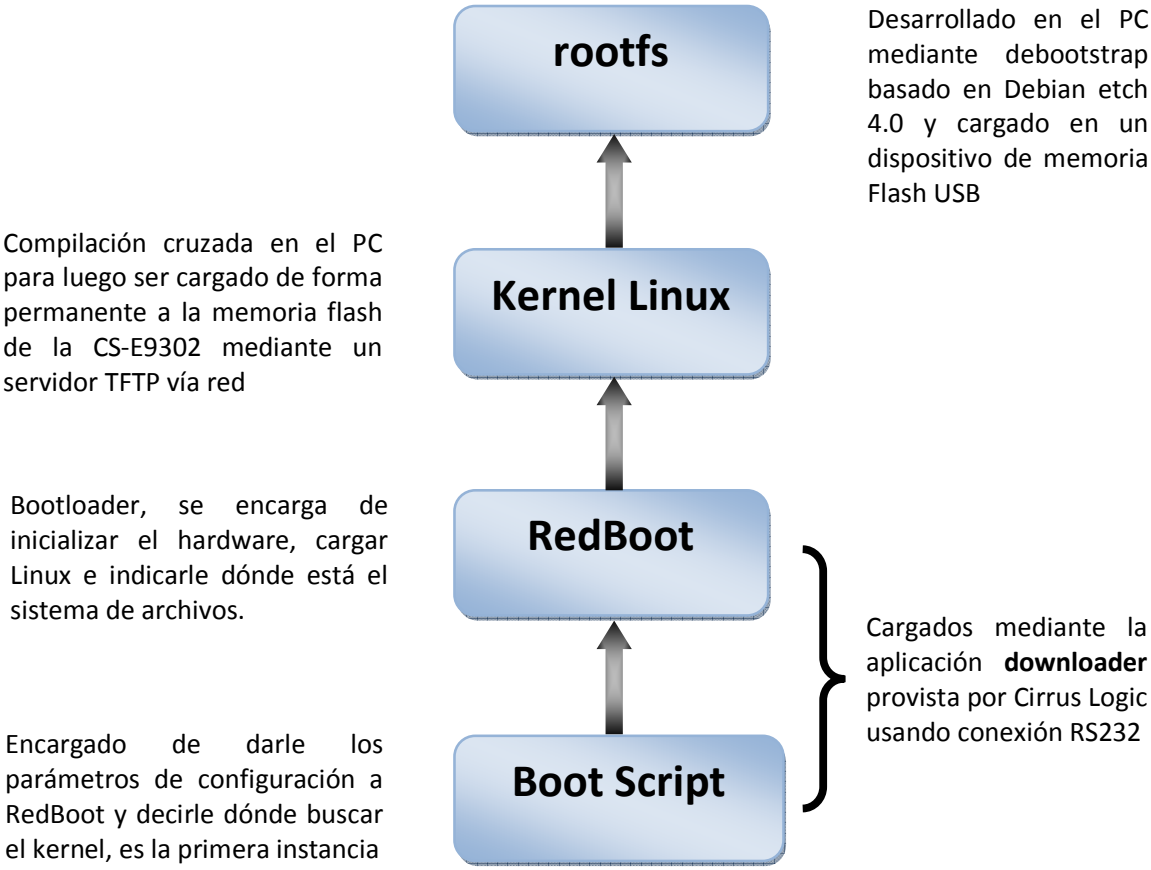


Fig. 25 Esquema de Desarrollo/Booteo del Sistema Linux Embebido

La Figura 25 muestra de manera sencilla el proceso en el cual la tarjeta correrá el sistema Linux embebido, y a su vez muestra como se desarrolló cada una de estas fases, las cuales están en el mismo orden de booteo. Lo primero que correrá es el Boot Script, que contiene los parámetros de configuración del bootloader RedBoot y viene incluido en este, ambos cargados en el sector boot ROM de la SBC.

RedBoot es el encargado de cargar el kernel, la primera vez desde un servidor TFTP para dejarlo residente en la memoria no volátil NOR-Flash, que posee la CS-E9302 y que luego de inicializar el hardware carga Linux, para acto seguido desde un dispositivo de memoria Flash USB de 2 GB, cargar el sistema de archivos root, el cual conforme avance el desarrollo de la tesis, contendrá las aplicaciones, librerías, archivos y demás datos necesarios para desarrollar prácticas y demostraciones.

5.1.1 Kubuntu 8.10 i38

Kubuntu GNU/Linux, es una variante derivada de Ubuntu que a su vez es una distribución basada en Debian, considerado por muchos el sistema Linux por excelencia debido a su estabilidad e infinidad de repositorios.

Kubuntu se diferencia de Ubuntu al implementar el escritorio KDE, el cual es mucho más vistoso visualmente que el escritorio Gnome usado por excelencia en Ubuntu.

Esta versión en particular implementa un escritorio KDE 4.1 bastante agradable y manejable.

Algunos Consejos Útiles

- Técnicamente se debería usar el usuario **root** (superusuario) para operaciones que requieran privilegio de superusuario, como montar un sistema de archivos, cargar un módulo del kernel, cambiar permisos, configuración de la red, etc. La mayoría de tareas regulares como descargar, extraer fuentes, compilar, pueden ser realizadas como usuario regular.
- Si los comandos son corridos desde un shell root por error, el usuario regular ya no podrá mas manipular los archivos generados correspondientes a la ejecución del comando. En este caso se debe reasignar permisos con el comando chown -R por ejemplo: `chown –R myuser:myuser /mnt/labs/`
- En Debian, Ubuntu, Kubuntu y sus derivados, no debe causar sorpresa si las aplicaciones gráficas no corren desde el usuario root. Se debe setear la variable DISPLAY con las mismas opciones que en el usuario regular, pero por motivos de seguridad eso no es recomendable, es preferible correr dichas aplicaciones desde un usuario regular.
- Si se está corriendo Kubuntu en modo LiveCD, no guardar datos en /home/Ubuntu. De otra manera el sistema se congelará rápidamente debido a que /home en este modo está residente en la RAM.

Diferentes Opciones de Configuración

Existen varias opciones para usar Kubuntu en el desarrollo de aplicaciones:

- La mejor opción es instalar Kubuntu en el disco duro. Esto permite al sistema ser rápido y ofrecer todas las características de un sistema instalado.
- La segunda opción es usar Kubuntu en modo LiveCD. Sin embargo, en este modo, la información personal se guarda en RAM, lo cual puede no ser suficiente para el propósito de nuestro sistema. La solución es crear un gran archivo dentro de una partición existente.
- La tercera opción es instalar Kubuntu en una máquina virtual.

Debido al propósito de esta tesis se eligió instalar Kubuntu en el disco duro, conviviendo con Windows XP con la opción de booteo dual, en un PC con las siguientes características:

- CPU Core 2 Duo E4500 a 2.2 GHz
- 2 GB de RAM
- Fastethernet 1/10/100 Mbps
- 8 hosts USB
- Unidad súper multi DVD
- Puerto COM RS232
- Periféricos normales de un PC

Instalación de Kubuntu

Para poder trabajar en buenas condiciones, se necesitan al menos 10 GB de espacio disponible en disco.

La manera estándar para crear espacio es particionando una partición Windows. Esto se puede hacer directamente desde la instalación de Kubuntu, siendo la Fig. 27 un ejemplo típico:

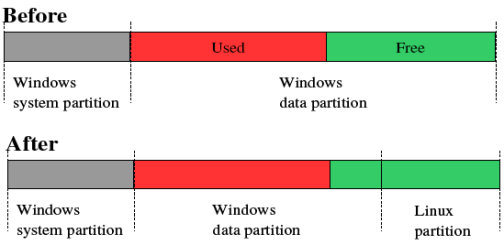


Fig. 26 Particionamiento

Configurando la Red y los Repositorios

Se debe asegurar primeramente que se puede acceder a la red propiamente, si esto no sucede se deberá ajustar las opciones en "System Settings", disponibles en el menú KDE.

Luego se debe verificar que los repositorios de paquetes están propiamente habilitados, caso contrario se debe correr el gestor de paquetes Adept (System -> Adept Manager en el menú KDE).

Si el acceso a Internet funciona usando un proxy se agregará la siguiente línea a /etc/apt/apt.conf :

```
Acquire::http::Proxy "http://ProxyServer:Port";
```

Finalmente aunque se comentó sobre el uso del usuario **root** sólo para tareas administrativas que requieran privilegios de superusuario, es bastante tedioso tener que cambiar a cada momento de usuario, por lo tanto la mayor parte del trabajo se realizó desde un shell root. En las distribuciones basadas en Debian como Ubuntu y Kubuntu el usuario root viene deshabilitado por defecto, pero esto se soluciona fácilmente dándole una contraseña a este usuario con:

```
$ passwd root
```

Y para cambiar desde cualquier usuario a root:

```
$ su
```

Y cuando sea necesario cambiar a algún otro usuario:

```
$ su <usuario>
```

apt-get

Este comando es propio de Debian y es muy útil e importante como se verá más adelante para lo que a este sistema embebido se refiere, ya que baja los programas de los repositorios, y además calcula dependencias y las instala también. Sin esta herramienta, se tendría que compilar los paquetes y resolver sus dependencias. Dependencias son los paquetes o librerías que necesita un programa para funcionar. Sin esta herramienta pues, es un verdadero lío instalar software.

Cuando se instala Kubuntu, por defecto esta herramienta buscará en el CD de instalación los paquetes a instalar. Para configurar la búsqueda en internet, habrá que editar un fichero, que está en /etc/apt/sources.list . En este fichero están las direcciones donde ir a buscar los paquetes. Siendo root, habrá que comentar (poner un # delante) la línea donde dice que busque en el CD y descomentar las otras líneas dónde hayan direcciones. Cada una de estas direcciones indica las diferentes categorías de paquetes soportados por Kubuntu.

Una vez descomentadas esas líneas, ya se puede actualizar el sistema. Lo primero que se debe hacer es un "apt-get update", con ello actualizamos toda la lista de paquetes de la que se tiene constancia. Es decir, se conecta a las direcciones anteriores y nos bajamos la lista de paquetes que hay.

Ahora ya se puede instalar aplicaciones o paquetes, para ello se ejecuta "`apt-get install <programa>`". Como por ejemplo: "`apt-get install apache2`" instalará el servidor web apache en la máquina y lo arrancará (se puede probar tecleando la dirección localhost en firefox).

Cuando no se recuerda el nombre completo de un paquete, puede ser buscado con "`apt-cache search <aproximación>`", por ejemplo, si no se recuerda como era el servidor para telnet, se teclea "`apt-cache search telnet`" y aparecen varios paquetes, se deben examinar un poco y se verá que hay un paquete llamado "`telnetd - telnet server`", así pues, se teclea "`apt-get install telnetd`" y ya se tiene el servidor de telnet en el sistema.

Para desinstalar programas, se usa "`apt-get remove programa`". Si se quiere borrar también los ficheros de configuración del programa, se usa "`apt-get remove --purge <programa>`". El sistema de apt-get no es muy óptimo desinstalando paquetes. Ya que si para instalarlo tuvo que instalar también una librería, a la hora de desinstalar no quita ésta. Para solucionarlo se puede usar otro gestor de paquetes, como Aptitude, que funciona igual que apt-get, para esto se hace un "`apt-get install aptitude`".

Para actualizar todos los paquetes del sistema, se usa "`apt-get dist-upgrade`", esto bajará todos los nuevos programas.

De vez en cuando, es recomendable hacer un "`apt-get update`" y "`apt-get dist-upgrade`" para conseguir las actualizaciones de seguridad.

5.1.2 Compilación Cruzada

La compilación cruzada es el proceso mediante el cual se realiza la compilación en una PC, denominada **host** (huésped), para ser ejecutado sobre otra arquitectura de hardware, denominado **target** (objetivo). Se realiza una compilación cruzada sobre un sistema, cuando este produce ejecutables para otro sistema. Esto sucede cuando el sistema **target** no posee el conjunto de herramientas de compilación nativas o cuando el sistema **host** es más rápido o posee mejores recursos (CPU, memoria, etc.).

El esquema típico de una plataforma hardware de compilación cruzada es el siguiente:

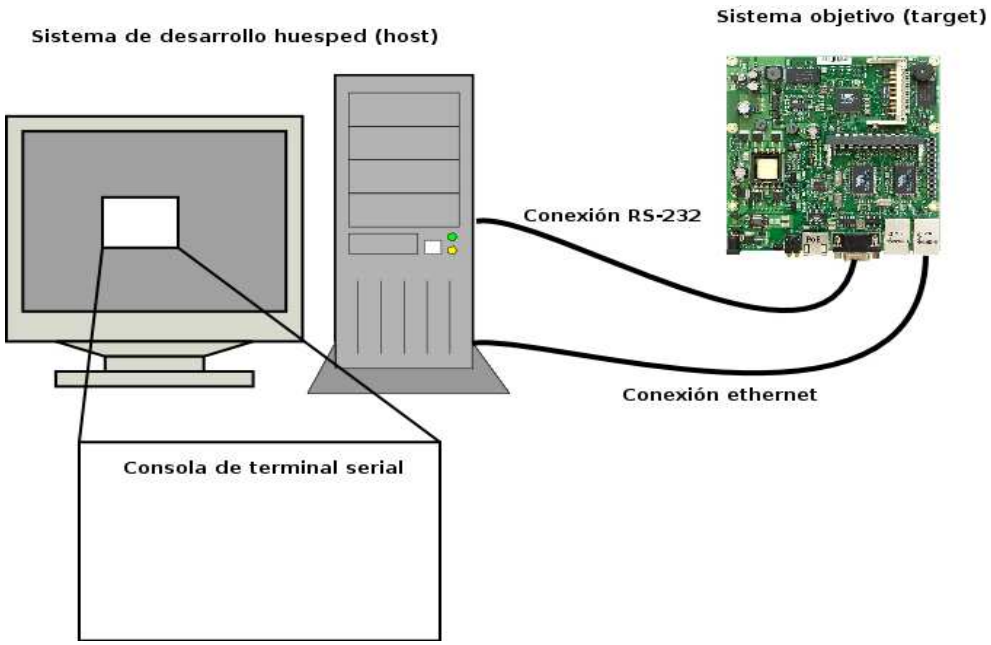


Fig. 27 Entorno de Desarrollo de Compilación Cruzada

5.1.3 Toolchain

Es un conjunto de herramientas, binarios y librerías necesarias para realizar la compilación cruzada (binutils, gcc, glibc). Algunos de los componentes son:

- **Compilador C GNU**(gcc): compilador C usado para generar código objeto.

- **Librería C GNU**: esta librería implementa las llamadas al sistema, *API*, como open, read. Pueden ser, glibc, uClib, u otras.

- **Binutils** (binary utilities): conjunto de programas necesarios para compilar/enlazar/ensamblar y realizar otras tareas de debugging. Los binarios principales son: **ld** (GNU linker), **as** (GNU assembler). Otros son:

    - addr2line – Convierte direcciones en nombres de archivo y números de línea.

- ar – Una utilidad para creación, modificación y extracción desde archivos.
- c++filt – Filtro para símbolos C++.
- gprof – Muestra información de perfiles.
- nlmconv – Convierte código de objeto en un NLM.
- nm – Lista todos los símbolos desde archivos de objeto.
- objcopy – Copia y traduce archivos de objeto.
- objdump – Muestra información desde los archivos de objeto.
- ranlib – Genera un índice de los contenidos de un archivo.
- readelf – Muestra la información de cualquier formato de archivo objeto ELF.
- size – Lista la sección de tamaños de un objeto o archivo.
- strings – Lista las cadenas imprimibles desde archivos.
- strip – Descarta símbolos.
- windres – Un compilador para recursos de archivo Windows.

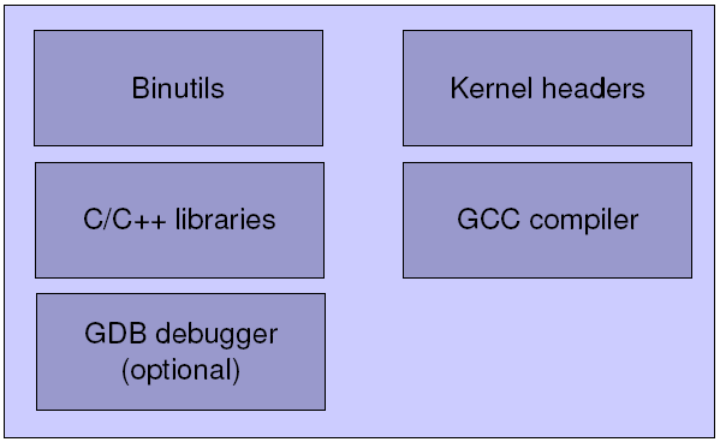


Fig. 28 Esquema Toolchain

La toolchain y la compilación cruzada, siguen un proceso específico dentro de la arquitectura hardware del entorno físico de compilación cruzada, el cual se puede representar de la siguiente manera:

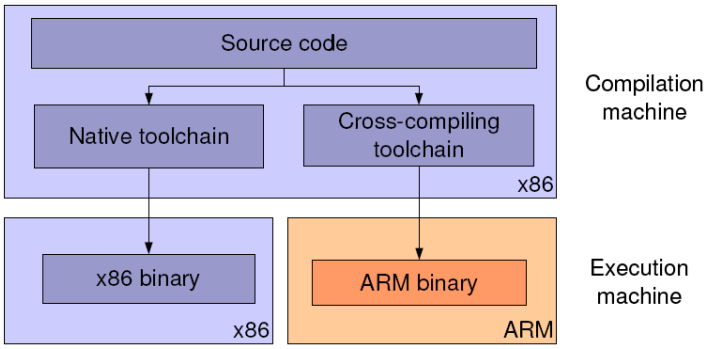


Fig. 29 Esquema de Compilación Cruzada y Binario resultante

5.1.4 Variables de Entorno en Linux

Una variable de entorno es un nombre asociado a una cadena de caracteres.

Dependiendo de la variable, su utilidad puede ser distinta. Algunas son útiles para no tener que escribir muchas opciones al ejecutar un programa, otras las utiliza el propio shell (PATH, PS1,…). La tabla siguiente muestra la lista de variables más usuales:

Variable	Descripción
DISPLAY	Donde aparecen las salidas de X-Windows.
HOME	Directorio personal.
HOSTNAME	Nombre de la máquina.
MAIL	Archivo de correo.
PATH	Lista de directorios donde buscar los programas.
PS1	Prompt.
SHELL	Intérprete de comandos por defecto.
TERM	Tipo de terminal.
USER	Nombre del usuario.

Tabla 14. Variables de Entorno en Linux

La forma de definir una variable de entorno cambia con el intérprete de comandos, se muestra tcsh y bash siendo los dos más populares en el ámbito Linux:

bash: `export VARIABLE=Valor`
tcsh: `setenv VARIABLE Valor`

Por ejemplo, para definir el valor de la variable DISPLAY:

bash: `export DISPLAY=localhost:0.0`
tcsh: `setenv DISPLAY localhost:0.0`

La variable PATH se exporta de la siguiente manera:

```
export PATH=……:$PATH
```

Para ver el contenido de las variables de entorno

```
env
```

Y si se quiere ver el de una en particular:

```
echo $PATH
```

5.1.5 ELDK (Embedded Linux Developer's Kit)

Para la instalación de las herramientas de compilación cruzada específicas para sistemas basados en ARM, se podrían usar algunas herramientas el set de herramientas que provee ELDK de DENX Software, el cual es una imagen .iso disponible en el internet bajo GPL el cual provee un sinnúmero de herramientas necesarias y listas para su instalación en un sistema Linux, sobre todo la principal herramienta para compilación cruzada ARM: arm-linux-gcc. Dicha imagen posee su propio instalador haciendo la tarea mucho más fácil, el .iso se descarga de la siguiente dirección:

http://mirror.switch.ch/ftp/mirror/eldk/eldk/4.1/arm-linux-x86

5.1.6 arm-linux-gcc

Este es el compilador cruzado, muchas veces resulta tedioso instalar ELDK e incluso infructuoso debido a que sus paquetes son rpm's que son paquetería hecha para sistemas basados en RedHat, siendo Kubuntu una derivación de Debian el tipo de paquetes que usa son .deb por lo tanto algunas veces suele dar problemas de compatibilidad aunque esto es relativo.

Por eso resulta conveniente la descarga directa del arm-linux-gcc en su versión comprimida .tar.bz2 para luego ser descomprimido y ubicado correctamente donde corresponde.

Dicho paquete se puede descargar de la siguiente dirección:

http://arm.cirrus.com/files/tools/arm-linux-gcc-4.1.1-920t.tar.bz2

Una vez descargado se descomprime y se copia a su ubicación correspondiente de la siguiente manera, como se mencionó anteriormente y de aquí en adelante no se volverá a mencionar, es preferible hacer esto desde un shell root:

```
# cd cs-e9302
# gunzip arm-linux-gcc-4.1.1-920t.tar.bz2
# tar -xvf arm-linux-gcc-4.1.1-920t.tar
# cp –r /4.1.1-920t/* /usr/local/arm
```

Para el correcto funcionamiento del compilador cruzado es necesario cambiar la variable de entorno PATH, indicándole la ubicación de los binarios:

```
# export PATH=/usr/local/arm/bin:$PATH
```

5.1.7 Consideraciones sobre el kernel

Ya se conoce el hardware a usarse en el sistema embebido y parte del software a usarse para el desarrollo de aplicaciones.

Ahora se necesita configurar correctamente el kernel Linux para que tenga una huella pequeña pero con toda la funcionalidad requerida. Aquí se brindarán unas pautas regulares para obtener un kernel con todo lo necesario para el sistema:

1. Seleccionar la arquitectura más cercana/compatible posible.

2. Deshabilitar "Enable loadable modules support", pues lo que se quiere es incrustar los controladores dentro de la imagen y no depender de módulos externos.

3. Seleccionar el modo apropiativo "Preemption model -> Preemptive kernel" dependiendo de si el sistema interactúa mucho con el usuario, caso contrario si se procesa a modo servidor, es conveniente usar "No force preemption". El kernel por defecto usa un modo medio.

4. Dispositivos de bloque. Activar únicamente los controladores SATA/PATA/IDE.

5. USB. Es necesario habilitar el uso de USB 2.0 y la clase USB HID para poder usar teclados ratones, etc. Se deshabilita todo lo demás.

6. Sistema de archivos. Se selecciona ext3.

La estructura típica del kernel se muestra claramente en la Figura 30, y no se deben cometer errores al configurar el kernel, puesto que si dicha estructura se altera, podría resultar en errores de pánico de kernel y posibles daños al sistema físico.

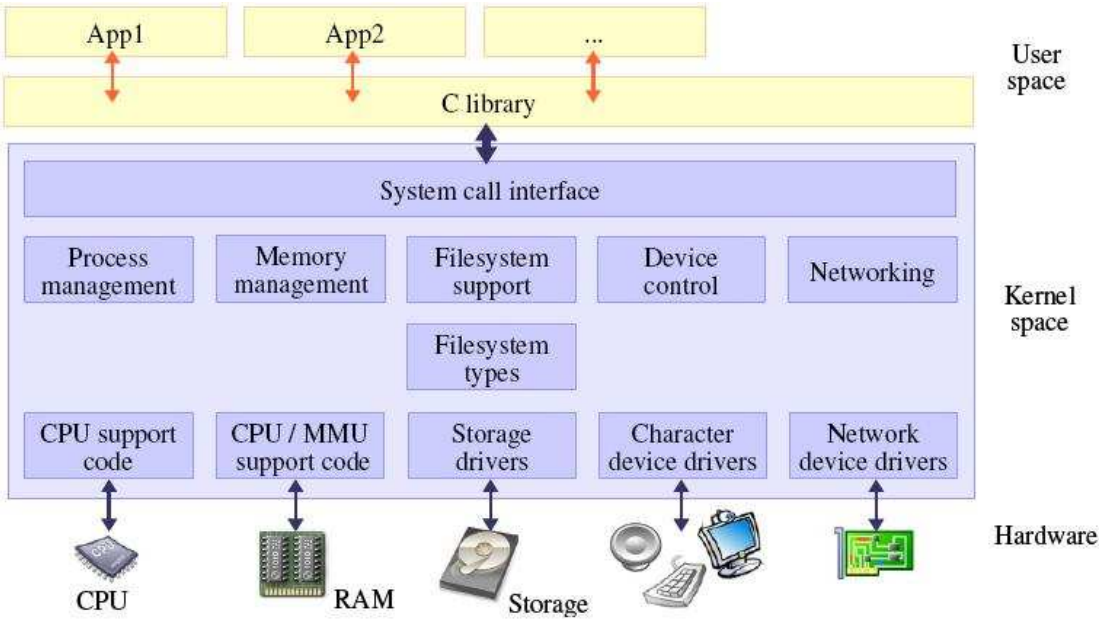


Fig. 30 Estructura del kernel Linux

Compilando y Parchando el Kernel

La CS-E9032 está implementada con un MCU EP9302, y es compatible con las tarjetas de producción de Cirrus Logic EDB9302. Pero ésta no soporta los puertos I/O de audio. Si se quiere usar el sistema de audio, el DAC CS4271 o el CS4272 se deben agregar como un módulo hardware adicional a la tarjeta.

Se eligió el kernel **linux-2.6.24-rc8** el cual es un kernel de prueba que va bastante bien con la tarjeta y posee una huella muy pequeña, cabe resaltar que se instalaron algunos kernels para probar la funcionalidad de la tarjeta y las funciones que brindaban los mismos, incluso con un kernel NetBSD:

- linux-2.4.21
- linux-2.6.8.1
- linux-2.6.17.14
- linux-2.6.20.4

- linux-2.6.27.8
- linux-2.6.28.1
- linux-2.6.24-rc7
- NetBSD

Cabe aclarar que los últimos núcleos van muy bien pero presentan una huella un poco más grande y como hablamos de un sistema con recursos limitados, se tomó la decisión de usar dicho núcleo experimental parchado, el cual a su vez ha sido optimizado siguiendo los pasos anteriores para reducir su huella dentro de lo posible sin alterar la funcionalidad de la tarjeta. Ahora se compilará y parchará el kernel Linux-2.6.24 siguiendo los siguientes pasos:

Primeramente Kubuntu no trae instalada la utilidad **patch** así que debe ser agregada e instalada en el sistema, siendo ésta una tarea sencilla mediante el comando **apt-get**.

```
# apt-get install patch
```

Y los pasos que siguen se refieren propiamente al kernel y su parchado para soporte SD/MMC:

```
# cd cs-e9302
# wget
http://www.kernel.org/pub/linux/kernel/v2.6/testing/v2.6.24/linux
-2.6.24-rc8.tar.bz2
# wget http://dev.ivanov.eu/projects/cs-e9302/linux-2.6.24-
rc8_ep93xx_mmc.patch.gz
# gunzip linux-2.6.24-rc8.tar.bz2
# tar -xvf linux-2.6.24-rc8.tar
# cd linux-2.6.24-rc8
# zcat ../linux-2.6.24-rc8_ep93xx_mmc.patch.gz | patch -p1

patching file arch/arm/mach-ep93xx/core.c
patching file cs-e9302_kernel_config
```

```
patching file drivers/net/arm/ep93xx_eth.c
patching file drivers/net/arm/ep93xx_eth.h
patching file drivers/spi/Kconfig
patching file drivers/spi/Makefile
patching file drivers/spi/spi_ep93xx.c
patching file include/asm-arm/arch-ep93xx/ep93xx-regs.h
```

```
# cp cs-e9302_kernel_config .config
```

Antes de configurar el kernel, el Makefile del linux-2.6.24.rc8 debe modificarse:

```
# cd linux-2.6.24-rc8
# vim Makefile
```

Se debe reemplazar ARCH ?= $(SUBARCH) y CROSS_COMPILE ?= como sigue:

```
ARCH ?= arm
CROSS_COMPILE = arm-linux-
```

Y, también se debe reemplazar el cs-e9302_kernel_config .config el cual tiene la información necesaria de la tarjeta CS-E9302 para compilar el kernel V2.6. Más aún, si se quiere que el kernel soporte más funciones como smbfs, network file system (NFS), se debe tratar con "`make menuconfig`":

```
# cd linux-2.6.24-rc8
# make oldconfig
# make menuconfig
```

Asegurarse de que el tipo de sistema es "EP93xx-based" y la plataforma "Support Cirrus Logic EDB9302":

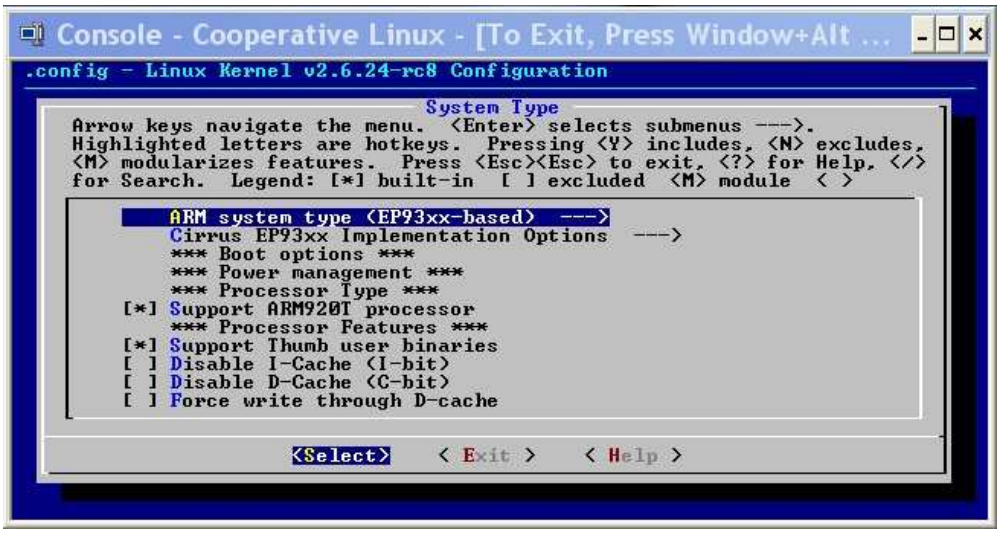


Fig. 31 Salida del comando make menuconfig

Y para más funciones se puede marcar "Root file system on NFS" y "SMB file system support" para usar un sistema de archivos raíz vía red y montar entornos compartidos de Windows.

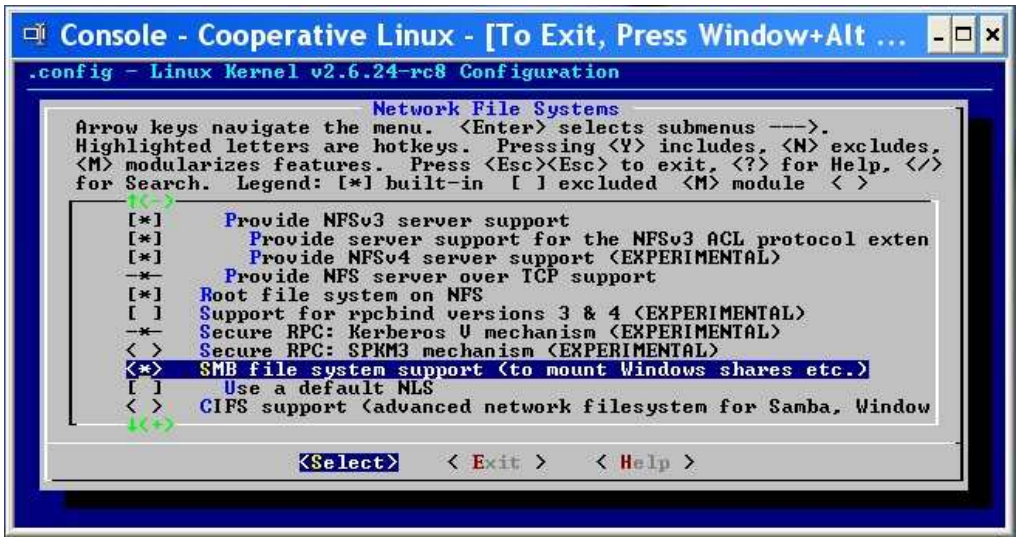


Fig. 32 Selección de paquetes adicionales

Finalmente, se usa el comando make para compilar el kernel y crear la imagen:

```
# make
# make zImage
```

5.1.8 Cable NULL MODEM

El PC se conecta a la CS-E9302 via conexión UART (RS232_0 para el lado de la CS-E9302). No se necesita la señal de feedback en el PC, pero el cable NULL MODEM es necesario para una correcta operación, de otra manera no se podrá escribir ningún carácter en la ventana del HyperTerminal desde el teclado.

He aquí el cable NULL MODEM con loopback handshaking, que se fabricará:

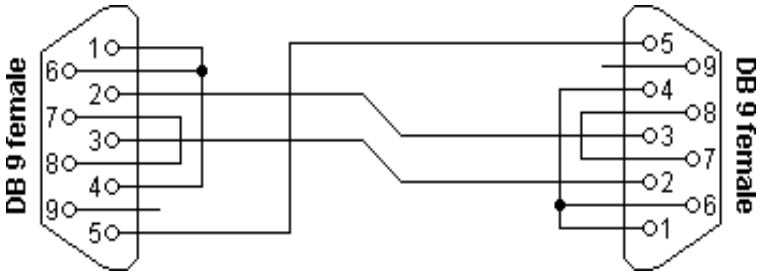


Fig. 33 Cable NULL MODEM

Conector 1	Conector 2	Función
2	3	Rx <= Tx
3	2	Tx => Rx
5	5	GND
1 + 4 + 6	-	DTR => CD + DSR
-	1 + 4 + 6	DTR => CD + DSR
7 + 8	-	RTS => CTS
-	7 + 8	RTS => CTS

Tabla 15. Descripción del cable NULL MODEM

5.1.9 HyperTerminal

Se usará el HyperTerminal de Windows como una ventana de prompt de comandos, el cual es un programa de comunicación serial de datos que usa el puerto COM del computador con conexión RS232, para mostrar la línea de comandos o la consola de un dispositivo, por lo general, carente de pantalla. Posee las siguientes capacidades:

- Consola serial a un sistema Unix remoto
- Transferencia de archivos
- Control de módem y marcado dial-up
- Configuración del puerto serial

Es posible también conectarse desde Linux mediante la herramienta Minicom, que cumple la misma función que el HyperTerminal, a diferencia que es open-source.

Primero se conecta la tarjeta al puerto serial (COM1/COM2) en el computador y después se corre HyperTerminal. Se configura el puerto serial a 57600; 8,N,1 y sin control de flujo. Luego se enciende la tarjeta.

Fig. 34 Nombre de conexión

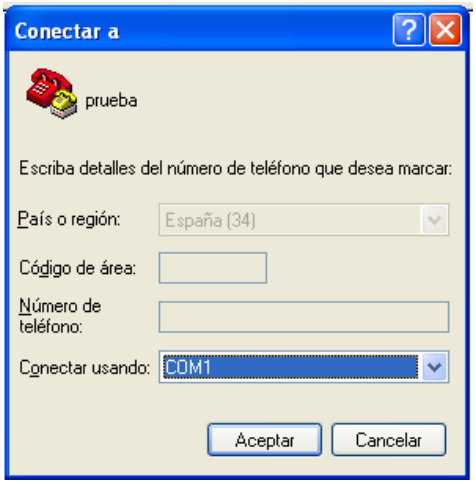


Fig. 35 Selección de puerto COM

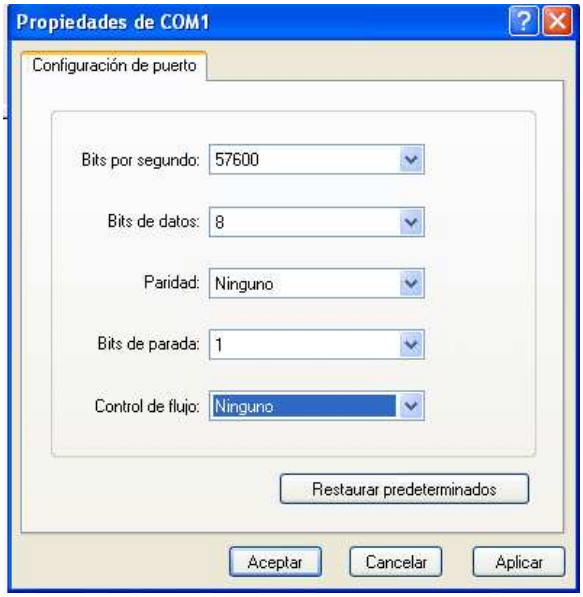


Fig. 36 Configuración puerto COM

5.1.10 Boot Script

En este sistema el Boot Script viene incluido junto a RedBoot, que es el bootloader que se va a utilizar, y su función principal es permitir configurar los parámetros de booteo, variables de entorno, y en este caso lo que principalmente se desea, lo cual es el

dispositivo desde el cual se cargará el kernel y el sistema de archivos o rootfs (sistema de archivos raíz).

Al encender, RedBoot usando el temporizador de conteo regresivo (Boot script timeout), después de 5 segundos, correrá el script por defecto, e inicializará hardware y kernel. Se puede detener este temporizador y reconfigurar las variables de entorno presionando Ctrl-C. En este caso, se debe cambiar los contenidos de los scripts de booteo y el valor de los parámetros dados para la correcta operación y sobre todo para que el sistema haga lo que se supone debe hacer. Más adelante se explicará su instalación y configuración para el proyecto junto con RedBoot debido, a que como se dijo, viene incluido en este. Por lo pronto daremos un vistazo general a los parámetros de booteo y ejemplos de cómo se configura.

Al encender la tarjeta, se debe presionar en un plazo de tiempo de 5 segundos Ctrl-C para obtener un prompt de RedBoot y configurar el script por defecto o ver su contenido de la siguiente manera:

```
RedBoot> fconfig
Run script at boot: true
Boot script:
.. fis load zImage
.. exec -c "console=ttyAM root=/dev/mmcblk0p1 rootdelay=5"
Enter script, terminate with empty line
>> fis load zImage
>> exec -c "console=ttyAM root=/dev/sda1 rootdelay=5"
>>
Boot script timeout (1000ms resolution): 5
Use BOOTP for network configuration: false
Gateway IP address: 192.168.0.1
Local IP address: 192.168.0.93
Local IP address mask: 255.255.255.0
Default server IP address: 192.168.0.94
DNS server IP address: 192.168.0.1
```

```
Set eth0 network hardware address [MAC]: true
eth0 network hardware address [MAC]:
0x00:0x00:0x00:0x00:0x48:0x33
GDB connection port: 9000
Force console for special debug messages: false
Network debug at boot time: false
Update RedBoot non-volatile configuration - continue (y/n)?
```

Cada uno de estos parámetros corresponde a lo siguiente:

Parámetro	**Valor**	**Función**
Run script at boot	True	Al encender, correrá un script
Boot script		Entrada de línea de comandos Linux
Boot script timeout	5	Después de 5s, RedBoot correrá la configuración de la variable script
Use BOOTP for network configuration	False	Bootear usando protocolo BOOTP
Gateway IP address	192.168.0.1	IP Gateway
Local IP address	192.168.0.93	Direción IP estática CS-E9302
Local IP address mask	255.255.255.0	Máscara de la IP Local
Default server IP address	192.168.0.94	IP del servidor TFTPD32
DNS server IP address	192.168.0.1	IP del servidor DNS
eth0 network hardware address [MAC]	0x00:0x00:0x00:0x00:0x48:0x33	Dirección MAC
GDB connection port	9000	Número de Puerto del servidor de depurado GDB
Force console for special debug messages	False	Habilita consola de mensajes especiales de depurado
Network debug at boot	False	Soporte de depurado de Red

time		
Update RedBoot non-volatile configuration	y/n	Grabar variables de configuración en la memoria Flash

Tabla 16. Parámetros del Boot Script

5.1.11 RedBoot

A diferencia de un bootloader de PC, en un sistema embebido el bootloader tiene a su cargo muchas más funciones, necesarias para la puesta a punto del hardware antes de cargar el núcleo. Es decir, el bootloader cumple la función del BIOS de un PC normal, debido a que es un sistema de recursos limitados o específicos esta tarea no se puede manejar de otra manera. Un esquema de lo que un bootloader realiza dentro de un sistema embebido típico es el siguiente:

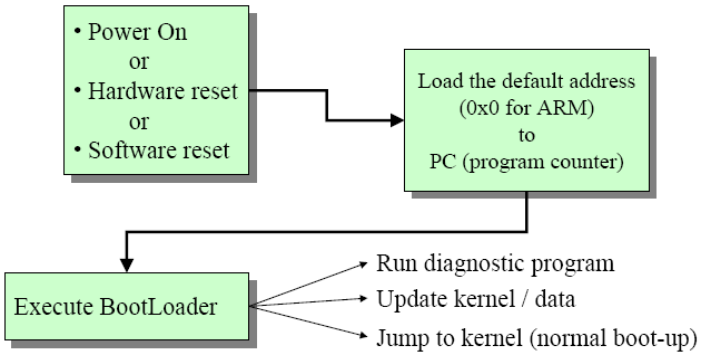


Fig. 37 Carga de RedBoot

RedBoot es un completo ambiente bootstrap o bootloader para sistemas embebidos. Se basa en la capa de abstracción de hardware de eCos, RedBoot hereda las cualidades de eCos de confianza, configurabilidad, portabilidad y huella reducida. RedBoot, es capaz de:

- Parar todas las interrupciones
- Parar el temporizador Watchdog
- Iniciar la Interfaz de memoria
- Copiar la sección de DATA & BSS
- Iniciar otros ítems, como, las GPIO
- Localizar los stacks de sistema
- Detectar las entradas de usuario y decidir que hacer:
    - Correr programas de diagnóstico
    - Actualizar el kernel / datos
    - Saltar al kernel (booteo normal)

Los diagramas siguientes son parte del proceso de booteo:

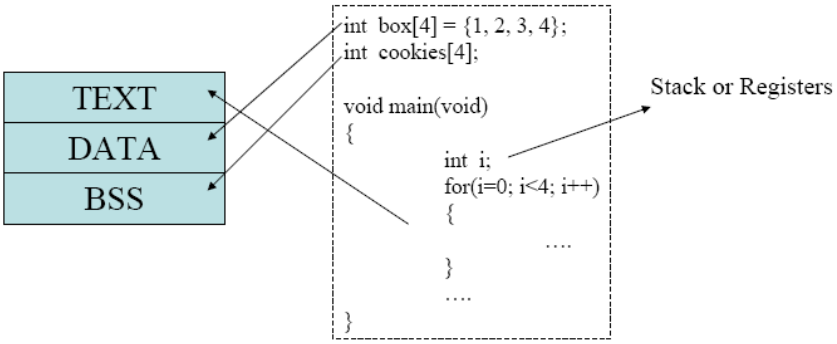


Fig. 38 DATA & BSS

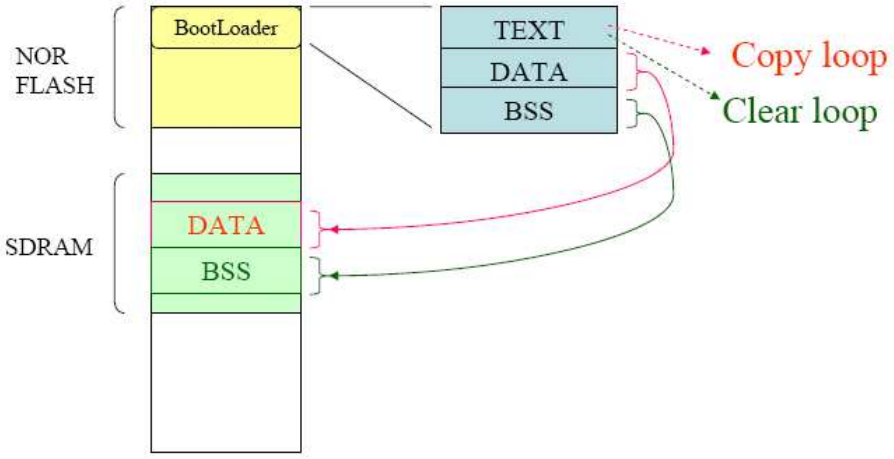


Fig. 39 Inicialización de la Interfaz de Memoria

RedBoot permite bajar y ejecutar aplicaciones embebidas via RS232 o Ethernet, incluyendo Linux embebido y aplicaciones eCos. Puede ser usado para ambos: desarrollo de productos y depuramiento de productos en campo (actualización de flash y booteo via red).

El soporte para depuración y descarga vía Ethernet está incluido permitiéndole recuperar sus parámetros IP via BOOTP o DHCP y descarga de imágenes de programas via TFTP. Las imágenes pueden ser también descargadas vía serial, o usando X- o Y-modem.

RedBoot puede ser usado para comunicarse con GDB (GNU Debugger – Depurador GNU) para depurar aplicaciones via serial o Ethernet, incluyendo la habilidad de interrumpir una aplicación que está corriendo empezada por GDB.

Una Interfaz de línea de comando interactiva se provee para permitir el manejo de las imágenes Flash, descarga de imágenes, configuración de RedBoot, etc.

Para booteo desatendido o automático, se pueden almacenar boot scripts en Flash permitiendo por ejemplo, cargar imágenes desde Flash, TFTP, USB, SD/MMC, etc.

Instalación

RedBoot corre usualmente desde el boot sector de la Flash en la plataforma embebida o desde una ROM de booteo, y está diseñado para correr cuando el sistema es alimentado. El método usado para instalar la imagen de RedBoot en almacenamiento no volátil varía de plataforma a plataforma.

En nuestro caso es muy sencillo programar la NOR-Flash vía RS232 mediante la utilidad **download** provista por Cirrus Logic mediante:

```
~$./download -b 57600 -o 0k -p S0 redboot_9302.bin
```

Comandos de Edición

- Delete (0x7F) o Backspace (0x08) borra el carácter a la izquierda del cursor.
- ^A o Inicio mueve el cursor al inicio de la línea.
- ^K borra todos los caracteres sobre la línea del cursor hasta el final.
- ^E o FIN posiciona el cursor al final de la línea.
- ^D o SUPR borra el carácter bajo el cursor.
- ^F o FLECHA DERECHA mueve el cursor un carácter a la derecha.
- ^B o FLECHA IZQUIERDA mueve el cursor un carácter a la izquierda.
- ^P o FLECHA ARRIBA reemplaza la línea actual por una línea previa. Un pequeño número de líneas puede ser guardado en el historial. Usando ^P (y ^N), la línea actual puede ser reemplazada por cualquiera de las líneas previas.
- ^N o FLECHA ABAJO reemplaza la línea actual por la línea siguiente en el historial.

En el caso de fconfig algunos comandos de edición adicionales son posibles:

- ^ (elevado a) salta a editar el ítem previo en la lista de fconfig. Si fconfig edita el ítem A, seguido del ítem B, presionando ^ cuando se cambia el ítem B, permite cambiar el ítem A.

- . (punto) para la edición de ítems y no cambia el ítem actual.
- Return deja el valor de este ítem sin cambio. Actualmente no es posible pasar a través del valor del script de inicio; este debe ser reiniciado.

También, al igual que en el BIOS, este tiene un modo de configuración que permite setear el modo en que bootea la SBC. Sin embargo, en vez de pantallas y mensajes de diálogo, RedBoot es un programa de línea de comandos que corre desde el puerto serial.

Nótese, que a cualquier tiempo se puede presionar el botón "PRN" de la tarjeta para resetear la misma. Si no se quiere grabar nada, esta es la opción para resetear RedBoot y reiniciarlo sin peligro a daño alguno.

Cuando se enciende la tarjeta o después de reiniciar la tarjeta, si se presiona Ctrl-C dentro de 5 segundos se obtiene una consola RedBoot:

```
RedBoot>
```

Desde este modo algunos comandos pueden ser usados:

`fconfig` – configuración basada en flash, permite ingresar parámetros para cada opción de configuración disponible, y permite subir los mismos a la flash, esto bajo riesgo propio del programador.

`fconfig -l` – lista la configuración actual

`fis xxx` – manipula el Flash Image System

`fis list` – lista la cantidad de memoria flash reservada

`help` – lista todos los comandos de RedBoot

`ip_address` – muestra la dirección IP de RedBoot

`ping -h <ip>` – hace ping hacia una dirección IP dada.

`reset` – resetea el chip y reinicia RedBoot, pero esto a veces no es suficiente suele ser necesario un reset físico mediante el botón "PRN"

`version` – muestra la información de versión de RedBoot

Algunas salidas de comando:

```
RedBoot> fis list
Name              FLASH addr Mem addr   Length     Entry point
RedBoot           0x60000000 0x60000000 0x00040000 0x00000000
RedBoot config    0x60FC0000 0x60FC0000 0x00001000 0x00000000
FIS directory     0x60FE0000 0x60FE0000 0x00020000 0x00000000
netbsd            0x60040000 0x00200000 0x00500000 0x00200000
netbsd_install    0x60540000 0x00200000 0x00500000 0x00200000
ramdisk.gz        0x60A40000 0x00800000 0x00300000 0x00800000
zImage            0x60D40000 0x00080000 0x000E0000 0x00080000

RedBoot> version

RedBoot(tm) bootstrap and debug environment [ROMRAM]
Non-certified release, version v2_0 - built 08:14:43, Aug 22 2006

Platform: Cirrus Logic EDB9302 Board (ARM920T) Rev A
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x02000000, 0x00041e68-0x01fdd000 available
FLASH: 0x60000000 - 0x61000000, 128 blocks of 0x00020000 bytes
each.
```

```
RedBoot> ip_address
IP: 192.168.0.93/255.255.255.0, Gateway: 0.0.0.0
Default server: 192.168.0.13, DNS server IP: 0.0.0.0
RedBoot>
```

5.1.12 root fs

Para poder cargar aplicaciones, realizar tareas administrativas y para manejo de funciones críticas del kernel, es necesario manejar archivos, ya que en Linux, todo se maneja por medio de archivos, desde los drivers y enlaces entre librerías y aplicaciones, hasta los dispositivos hardware, por eso es necesario instalar un sistema de archivos en algún dispositivo de almacenamiento no volátil, el cual cumpla las funciones que a su vez hace un disco duro en un PC, el cual contendrá todo lo que se haya instalado o agregado al sistema de forma que no se pierda en el apagado de la tarjeta.

Como se implementará un kernel Linux estándar, éste puede correr perfectamente cualquiera de los sistemas de archivos raíz estándar de cualquier distribución. El root fs Debian fue desde un principio la opción a considerar por su facilidad de implementar y porque soporta casi todos los procesadores ARM de sistemas embebidos existentes.

Se eligió la versión estable Debian 4.0 etch, la cual contiene paquetes que han pasado todas las fases y escenarios de prueba lo cual brinda confiabilidad al sistema. En contraste con muchísimas otras distribuciones, Debian GNU/Linux no se enfoca primariamente en usos de PC o Servidores. Esta es una de las poquísimas distribuciones que consiste en una base de desarrollo extremadamente amplia que a su vez trabaja en un amplio espectro de aspectos para sistemas operativos. Es así, que por lo general la gente se refiere a Debian como el "Sistema Operativo Universal". En efecto, el sistema es tan flexible que terceros lo consideran una meta distribución, y basan sus distribuciones GNU/Linux más específicas en él, tales como: Knoppix, Ubuntu, Kubuntu, etc, mientras Debian se mantiene como una distribución completamente usable en su propio derecho, tanto así que es portado a muchos sistemas embebidos y muchas de sus herramientas se usan en ellos.

Instalando y corriendo debootstrap

La herramienta que el instalador de Debian usa para crear el sistema de archivos, la cual es reconocida como la manera oficial un sistema base Debian, es **debootstrap**. Este usa **wget** y **ar**, pero en otros ambientes lo hace solo desde /bin/sh. Se deber verificar y/o instalar wget y luego instalar debootstrap mediante:

```
# apt-get install wget
# apt-get install ar
# apt-get install debootstrap
```

Se debe editar el contenido de /usr/sbin/debootstrap:

```
# vim ./usr/sbin/debootstrap
```

Luego se cambia la dirección para apuntar al directorio donde se desarrollará el root fs.

```
DEBOOTSTRAP_DIR= /mnt/debinst/work/usr/lib/debootstrap
```

Se debe substituir la línea para la arquitectura correspondiente a ARM, en la línea de comandos de debootstrap:

```
# ./usr/sbin/debootstrap --arch arm etch /mnt/debinst
http://ftp.debonaras.org/etch/debian
```

Para configurar el **sistema base**, se debe sobrescribir el archivo de interfaces activas, más adelante se agregará más a él:

```
# cp /etc/network/interfaces /mnt/debinst/etc/network/interfaces
```

Ahora se tiene un sistema Debian real, aunque bastante escueto, en el disco. Se ejecuta chroot en él:

```
# LANG= chroot /mnt/debinst /bin/bash
```

El siguiente paso para poner a punto el root fs es **montar la partición** y agregar lo necesario a ella:

```
# editor /etc/fstab

##################START#########################################
etc/fstab: static file system information.
#
# file system    mount point   type    options
/dev/sda1       /              ext3    defaults        1  1
/dev/sda1       swap           swap    defaults        0  0
proc            /proc          proc    defaults        0  0
usbfs           /proc/bus/usb  usbfs   defaults        0  0
####################END#########################################
```

Para configurar el **teclado**:

```
# dpkg-reconfigure console-data
```

Para configurar el **networking**:

```
# vim /etc/network/interfaces

##########################START#########################################
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)
# See the interfaces(5) manpage for information on what options are
# available.
#########################################################################
# We always want the loopback interface.
#
auto lo
iface lo inet loopback
```

```
#
# The NSLU2 built-in ethernet
auto eth0
# The pre-up option must always be supplied, regardless
# of configuration, to set the hardware correctly.
# Severe network problems may result if this option is
# removed.
iface eth0 inet dhcp
pre-up modprobe -f ixp425_eth
        pre-up modprobe -f ixp400
        pre-up ifconfig eth0 hw ether 00:04:5A:XX:YY:ZZ
# make sure you copy in your own ethernet MAC address from the sticker
on the bottom of your slug, replace XX:YY:ZZ

# set static fallback address ...
# It would be a good idea to set this, really.
# address give.me.an.ip
# netmask give.me.a.netmask
# gateway give.me.a.gw
#
########################END################################################
#####
```

Se editan las directivas de nombre de dominio en:

```
# vim /etc/resolv.conf
##########START##################
search hqdom.local\000
nameserver 10.1.1.36
nameserver 192.168.9.100
##################END#############
```

Se pone un nombre de host al sistema embebido, será cs-e9302:

```
# echo cs-e9302 > /etc/hostname
```

Bien es cierto que este proceso es tedioso y existen muchos root fs basados en Debian que están listos para usarse con la arquitectura ARM, por tanto y en vista de lo complicado que resultan algunas cosas, se subió el root fs al internet como parte de una comunidad dispuesta a portar Linux a la CS-E9302, y está disponible para descarga así que obviaremos los pasos anteriores, debido a que este sistema tiene el propósito de enseñar a manejar un sistema embebido y mas no seguir los pasos más complicados de su construcción. Por tanto los pasos siguientes muestran cómo poner a punto el root fs y dejarlo residente en una memoria Flash USB.

Primeramente, en el PC se crea un dispositivo ext3 para el root fs, se conecta un drive USB FLASH, asumiendo que el nombre del dispositivo de almacenamiento es /dev/sdb1:

```
# cd cs-e9302
# wget http://dev.ivanov.eu/projects/cs-e9302/cs-e9302_root2.tar.gz
# mkfs.ext3 -L root /dev/sdb1
# mount /dev/sdb1 /mnt
# cd /mnt
# gunzip cs-e9302_root2.tar.gz
# tar -xvf cs-e9302_root2.tar
# umount /mnt
```

Y hecho esto el sistema de archivos ha sido copiado y es bastante funcional, basta conectar el Flash USB al host USB 1 en la CS-E9302 antes de ser encendida, y configurar RedBoot para que lo cargue.

Cabe aclarar que este sistema de archivos raíz estuvo primariamente pensado para ser montado en una Memoria SD, y en la página desde la cual se descarga el root fs y la comunidad mencionada existen instrucciones solo para este propósito; pero debido a su vida corta y a la lentitud de acceso a los datos, y como parte de investigación en esta tesis se decidió implementarlo en una memoria USB, ya que los host de la CS-E9302 son USB 2.0, es decir de alta velocidad.

5.1.13 Cargando Linux y root fs en la CS-E9302

Después de compilar por completo el kernel, la imagen del kernel Linux (zImage) se debió crear en el directorio /linux-2.6.24-rc8/arch/arm/boot.

Para bootear el kernel, se debe configurar la variable de entorno de RedBoot. Resulta sencillo si se usa el RS232 para cargar la imagen del kernel, pero debido a la muy baja transferencia de datos que el RS232 proporciona, lo más seguro es que tomará varios minutos para que se complete la carga. Afortunadamente, RedBoot soporta el protocolo TFTP, y se puede cargar la imagen del kernel a la CS-E9302 via red con una muchísima mejor tasa de transferencia. En este caso se necesita un servidor TFTP, que se puede descargar gratis del internet. Se usará el TFTPD32 server para WinXP. El TFTPD32 server no necesita instalación y está disponible en http://www.esnips.com/doc/e8c33ffb-e341-402c-be53-ed259daf6ac3/Tools. Se crea un nuevo directorio bajo el drive C:\ por ejemplo:

```
C:\TFTPD32\
```

Simplemente se copia tftpd32.exe en esta carpeta y:

1. Elegir la pestaña Tftp Server
2. Click en el botón Browse, y referenciarlo a C:\TFTPD32
3. Copiar el archivo zImage a C:\TFTPD32

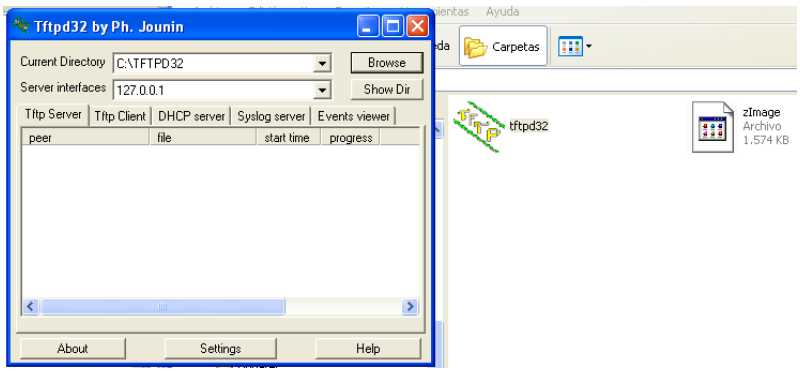


Fig. 40 Tftpd32 Server

"Server interfaces" indica la dirección IP del servidor (la dirección IP de WinXP), la cual es 192.168.0.94. Ahora el servidor TFTPD32 está listo para cargar la imagen del kernel Linux. En el prompt del Hyper Terminal, se setearán las variables de entorno, presionando Ctrl-C 5 segundos antes de que inicie RedBoot:

```
RedBoot> fconfig
Run script at boot: true
Boot script:
.. fis load zImage
.. exec -c "console=ttyAM root=/dev/mmcblk0p1 rootdelay=5"
Enter script, terminate with empty line
>> fis load zImage
>> exec -c "console=ttyAM root=/dev/sda1 rootdelay=5"
>>
Boot script timeout (1000ms resolution): 5
Use BOOTP for network configuration: false
Gateway IP address: 192.168.0.1
Local IP address: 192.168.0.93
Local IP address mask: 255.255.255.0
Default server IP address: 192.168.0.94
DNS server IP address: 192.168.0.1
Set eth0 network hardware address [MAC]: true
eth0 network hardware address [MAC]:
0x00:0x00:0x00:0x00:0x48:0x33
GDB connection port: 9000
Force console for special debug messages: false
Network debug at boot time: false
Update RedBoot non-volatile configuration - continue (y/n)?y
```

Se checa las direcciones de memoria en flash y lo que existe en ellas:

```
RedBoot> fis list
```

```
Name            FLASH addr Mem addr   Length     Entry point
Name            0x60000000 0x60000000 0x00040000 0x00000000
RedBoot         0x60FC0000 0x60FC0000 0x00001000 0x00000000
RedBoot config  0x60FE0000 0x60FE0000 0x00020000 0x00000000
FIS directory   0x60040000 0x00A00000 0x00A00000 0x00A00000
ramdisk         0x60A40000 0x00080000 0x00300000 0x00080000
```

La imagen pre-cargada será guardada en la dirección 0x60A40000, cuando se inicie el kernel, RedBoot copiará la zImage desde la memoria Flash de esa dirección a la dirección de memoria principal 0x00080000, ahora se debe cargar la imagen desde el servidor Tftp, y en caso de existir algún kernel en dicha dirección, primero se borra:

```
RedBoot> fis delete zImage
```

Una vez hecho esto se ejecuta:

```
RedBoot> load -r -v -b 0x80000 zImage
```

Ahora se crea un nuevo archivo y se copia la zImage en la memoria Flash:

```
RedBoot> fis create -b 0x80000 -l 0x2F0000 zImage
```

Dónde 0x2F0000 es el valor de la huella o tamaño del kernel que varía de 1.5 a aproximadamente 3 MB dependiendo de las funciones soportadas por el kernel.

Hecho esto la tarjeta quedó lista para correr un sistema GNU/Linux completo basado en Debian, llamado ahora **C@M@LEON Embedded GNU/Linux**, con la mayoría de las funciones necesarias para un PC incluido un cliente DHCP, dentro de un sistema embebido. Ahora cada vez que se encienda la tarjeta, automáticamente correrá Linux y cargará el sistema de archivos sin necesidad de ingresar comandos adicionales.

CAPITULO VI

DESARROLLO DE APLICACIONES PARA C@M@LEON

6.1 USANDO C@M@LEON Embedded GNU/Linux

Cuando C@M@LEON bootea satisfactoriamente, durante las primeras sesiones y cada cierto tiempo, se debe actualizar los paquetes, por tanto al igual que otras distribuciones basadas en Debian, se puede usar la mayoría de sus comandos, repositorios, aplicaciones y paquetería .deb además de las típicas funcionalidades de cualquier sistema basado en Linux.

Cabe recalcar que debido a las prestaciones se ha elegido usar para todas las aplicaciones el editor **vim**, el cual debido a ser manejado desde una consola, necesita que el usuario sepa sus comandos básicos, en tal caso en la parte de los anexos se incluirá la manera correcta de utilizarlo la cual también sirve para el editor **vi**. Además de esto no se explicará en detalle el funcionamiento de los servidores ni mucho menos su marco teórico debido a que esta tesis se centra en el aprendizaje de sistemas

embebidos y sus posibles usos y mas no en conocer a fondo sus aplicaciones, de esto existe bastante literatura disponible en el internet y en libros afines.

Simplemente se corre el comando apt-get en el prompt de la consola. Hay que asegurarse de que la conexión a internet de alta velocidad funciona correctamente:

```
cs-e9302:~# ifconfig
```

```
eth0      Link encap:Ethernet  HWaddr 00:DE:AD:B0:05:00
          inet addr:192.168.0.93  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: 2001:a001:14c4:0:2de:adff:feb0:500/64 Scope:Global
          inet6 addr: fe80::2de:adff:feb0:500/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:86 errors:0 dropped:86 overruns:0 frame:0
          TX packets:80 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Interrupt:39

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:560 (560.0 b)  TX bytes:560 (560.0 b)
```

En caso de ser necesario configurar la interfaz manualmente:

```
cs-e9302:~# ifconfig eth0 inet 192.168.0.93
```

Y por si se necesita en algún momento bajar la interfaz:

```
cs-e9302:~# ifconfig eth0 down
```

Ahora se checa la conexión al PC:

```
cs-e9302:~# ping -c 2 192.168.24.10

PING 192.168.24.10 (192.168.24.10) 56(84) bytes of data.
64 bytes from 192.168.24.10: icmp_seq=1 ttl=128 time=2.33 ms
64 bytes from 192.168.24.10: icmp_seq=2 ttl=128 time=0.954 ms

--- 192.168.24.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.954/1.645/2.336/0.691 ms
```

Checar la conexión a internet:

```
cs-e9302:~# ping -c 2 www.google.com

2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.954/1.645/2.336/0.691 ms
cs-e9302:~# ping -c 2 www.google.com
PING www.l.google.com (66.249.89.147) 56(84) bytes of data.
64 bytes from jp-in-f147.google.com (66.249.89.147): icmp_seq=1 ttl=234 time=17.
9 ms
64 bytes from jp-in-f147.google.com (66.249.89.147): icmp_seq=2 ttl=234 time=17.
8 ms

--- www.l.google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 17.816/17.897/17.979/0.156 ms
```

Una vez lista la configuración y revisada la conectividad, se puede instalar y actualizar aplicaciones. Se actualizamos la lista de paquetes:

```
cs-e9302:~# apt-get update
```

Se instalan los paquetes correspondientes a la última versión estable de Debian y presentes en C@M@LEON Embedded GNU/Linux:

```
cs-e9302:~# apt-get upgrade
```

Instalando make

```
cs-e9302:~# apt-get install make
```

Instalando gmake

```
cs-e9302:~# apt-get install gmake
```

Instalando el compilador GNU C

```
cs-e9302:~# apt-get install gcc
```

Instalando el compilador GNU C++

```
cs-e9302:~# apt-get install g++
```

Instalando las librerías C estándar

```
cs-e9302:~# apt-get install libc6-dev
```

Instalando las librerías gráficas X

```
cs-e9302:~# apt-get install libx11-dev
```

Instalando el servidor Samba

```
cs-e9302:~# apt-get install smbfs
```

Instalando el demonio hotplugger udev

```
cs-e9302:~# apt-get install udev
```

6.1.1 Compilando Programas C

Aunque es un sistema con recursos no tan provistos como los de un PC, este sistema embebido es capaz de correr sus propias aplicaciones sin necesidad de compilación cruzada, como se verá más adelante en el desarrollo de las aplicaciones.

Ahora se escribirá un simple programa en C, y lo se ejecutará como sigue:

```
cs-e9302:~# vim hola.c


#include <stdio.h>

int main (int agc, char* argc[])
{
  printf ("Hola!!  \n");
  printf ("Tu comando es: %s\n", argc[0]);
  return 0;
}


cs-e9302:~# gcc -o hola hola.c
cs-e9302:~# ./hola


Hola!!
Tu comando es:./hola


cs-e9302:~/temp#
```

El programa se explica por sí solo.

6.1.2 Usando el sistema X-Window vía Red

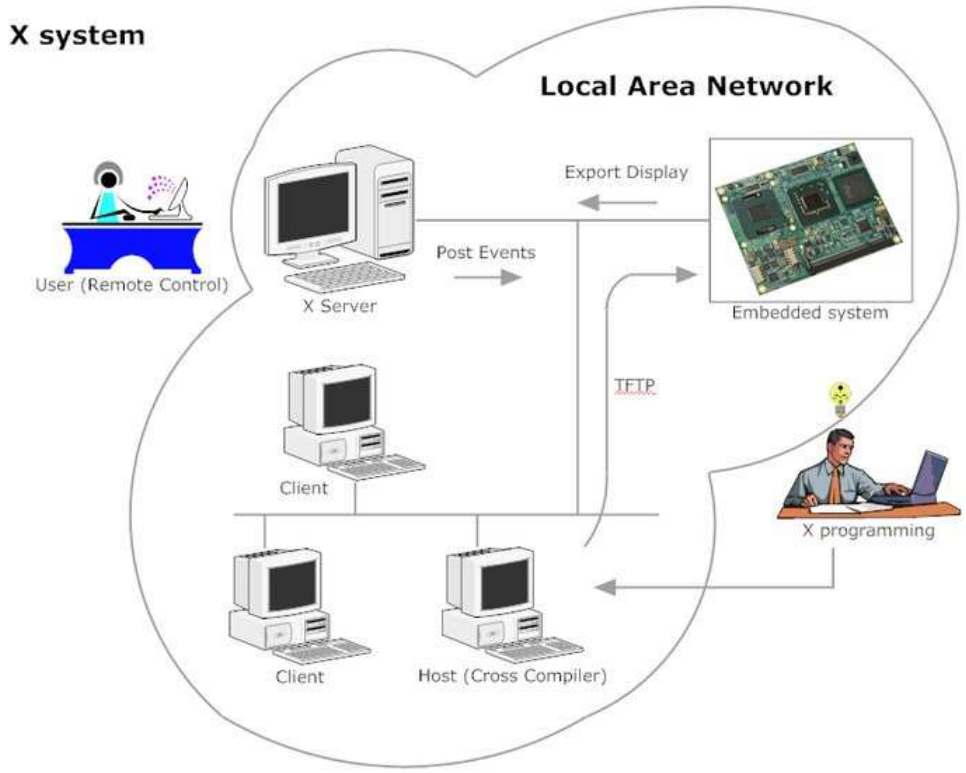


Fig. 41 Escenario del proceso de creación de C@M@LEON incluyendo aplicaciones gráficas

Se usará la CS-E9302 como un cliente de aplicaciones, el cual envía el entorno gráfico a los dispositivos de pantalla (el Servidor X) vía conexión LAN. El escritorio del PC Windows XP correrá como un Servidor X, este recibe las informaciones de pantalla como mapas de bits desde el cliente, da soporte al mouse, eventos de presionado de teclas y otros funciones, y las envía al cliente.

Para el servidor X se usará una aplicación muy poderosa, completa y fácil de instalar así como de usar, llamada X-Win32 de la empresa StarNet el único inconveniente que se tiene con ella es que no es open-source, sólo se puede obtener una licencia de evaluación de 30 días. Existen otras aplicaciones similares como Xming o Cygwin/X que son gratuitas y open-source, pero son mucho más complejas de instalar y manejar, y necesitan software adicional.

En el sistema embebido se instala la aplicación **xterm** para lanzar un terminal con entorno gráfico, simple pero bastante funcional:

```
cs-e9302:~# apt-get install xterm
```

Se corre el servidor X-Win32, y se exporta la variable de pantalla desde la consola de C@M@LEON, con la dirección IP para que pueda ser vista en WinXP vía red:

```
cs-e9302:~# export DISPLAY=192.168.0.94:0.0
```

En los sistemas basados en Debian, la opción de lanzar aplicaciones gráficas desde root está deshabilitada por defecto, por motivos de seguridad, por tanto para correr cualquier aplicación gráfica debemos crear un usuario regular y cambiarnos al mismo:

```
cs-e9302:~# adduser terminal
cs-e9302:~# su terminal
cs-e9302:~terminal$
```

Y ahora se lanzará un terminal gráfico a la dirección de red inicializado la aplicación xterm:

```
cs-e9302:~terminal$ xterm -bg black -fg white +cm +dc -geometry 80x20+100+50 &
```

Lo cual dará el siguiente resultado en el PC WinXP:

Fig. 42 xterm en WinXP

Lo cual permitirá cambiar las opciones del xterm. Se comprueba si la respuesta del sistema X-Window es satisfactoria.

6.1.3 Compilado y Ejecución de una Aplicación X

Para esta aplicación se uso la consola en modo gráfico que generó la aplicación anterior y todo se realizó desde el mismo terminal gráfico en el PC WinXP. No se profundizará en cómo funciona el sistema X-Window ni la librería Xlib, simplemente nos limitaremos a constatar y explicar que ambos permiten la construcción de entornos gráficos sencillos, que permite en cualquier sistema Linux construir aplicaciones que funcionan en modo gráfico como se muestra a continuación. Se escribe una simple aplicación X "hello", que se compilará y ejecutará para probar la librería xlib11, se creará una ventana, luego se imprime la cadena "Hola, X Window System!" y se muestra la misma sobre la pantalla del servidor X en el PC. La ventana se cerrará cuando se presione un botón del mouse o una tecla.

```
cs-e9302:~# vim hello.c
```

```
/**** hello.c demo ******/

/*  Declaración Xlib11 */
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>

/*  Declaración librerías C estándar  */
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

/*  Variables Globales  */
Display *     display;
int           screen_num;
static char * appname;
```

```
/*   main() function   */
int main( int argc, char * argv[] ) {

    /*  Window variables  */
    Window       win;
    int          x, y;
    unsigned int width, height;
    unsigned int border_width;
    char *       window_name = "CS-E9302 DEMO!";
    char *       icon_name   = "Hello";

    /*  Display variables  */
    char *       display_name = NULL;
    unsigned int display_width, display_height;

    /*  Miscellaneous X variables  */
    XSizeHints *  size_hints;
    XWMHints   *  wm_hints;
    XClassHint *  class_hints;
    XTextProperty windowName, iconName;
    XEvent        report;
    XFontStruct * font_info;
    XGCValues     values;
    GC            gc;
    appname = argv[0];

    /*  dar lugar en memoria a nuestras estructuras  */
    if ( !( size_hints  = XAllocSizeHints() ) ||
           !( wm_hints    = XAllocWMHints()   ) ||
           !( class_hints = XAllocClassHint() )    ) {
        fprintf(stderr, "%s: couldn't allocate memory.\n", appname);
        exit(EXIT_FAILURE);
    }

    /*  Conectar al X server  */
    if ( (display = XOpenDisplay(display_name)) == NULL ) {
        fprintf(stderr, "%s: couldn't connect to X server %s\n", appname,
display_name);
        exit(EXIT_FAILURE);
    }

    /*  Configurar la pantalla a mostrar */
```

```
    screen_num     = DefaultScreen(display);
    display_width  = DisplayWidth(display, screen_num);
    display_height = DisplayHeight(display, screen_num);

    x = y = 0;
    width  = display_width / 6;
    height = display_width / 7;

    win = XCreateSimpleWindow(
            display,
            RootWindow(display, screen_num),
                  x, y, width, height, border_width,
                  WhitePixel(display, screen_num),
                  BlackPixel(display, screen_num)
              );

    if ( XStringListToTextProperty(&window_name, 1, &windowName) == 0 ) {
        fprintf(stderr, "%s: structure allocation for windowName failed.\n",
appname);
        exit(EXIT_FAILURE);
    }

    if ( XStringListToTextProperty(&icon_name, 1, &iconName) == 0 ) {
        fprintf(stderr, "%s: structure allocation for iconName failed.\n",
appname);
        exit(EXIT_FAILURE);
    }

    size_hints->flags       = PPosition | PSize | PMinSize;
    size_hints->min_width   = 100;
    size_hints->min_height  = 50;

    wm_hints->flags         = StateHint | InputHint;
    wm_hints->initial_state = NormalState;
    wm_hints->input         = True;

    class_hints->res_name   = appname;
    class_hints->res_class  = "hellox";

    XSetWMProperties(display, win, &windowName, &iconName, argv, argc,
             size_hints, wm_hints, class_hints);
```

```
    XSelectInput(display,
                 win,
                 ExposureMask    |
                 KeyPressMask    |
                 ButtonPressMask |
                 StructureNotifyMask);

    if ( (font_info = XLoadQueryFont(display, "9x15")) == NULL ) {
        fprintf(stderr, "%s: cannot open 9x15 font.\n", appname);
        exit(EXIT_FAILURE);
    }

    gc = XCreateGC(display, win, 0, &values);
    XSetFont(display, gc, font_info->fid);
    XSetForeground(display, gc, WhitePixel(display, screen_num));

    /*  Mostrar Ventana  */
    XMapWindow(display, win);

    while ( 1 ) {
        static char * message = "Hello, X Window System!";
        static int    length;
        static int    font_height;
        static int    msg_x, msg_y;

        XNextEvent(display, &report);

        switch ( report.type ) {
          case Expose:
            if ( report.xexpose.count != 0 ){
                break;
              }

            length = XTextWidth(font_info, message, strlen(message));
            msg_x  = (width - length) / 2;

            font_height = font_info->ascent + font_info->descent;
            msg_y  = (height + font_height) / 2;

            XDrawString(display, win, gc, msg_x, msg_y, message,
strlen(message));
```

```
          break;

          case ConfigureNotify:
            width  = report.xconfigure.width;
            height = report.xconfigure.height;
          break;

          case ButtonPress:
          case KeyPress:
            /*  Clean up and exit  */
            XUnloadFont(display, font_info->fid);
            XFreeGC(display, gc);
            XCloseDisplay(display);
            exit(EXIT_SUCCESS);

        }
   }
    return EXIT_SUCCESS;
}
```

Se compila y ejecuta:

```
cs-e9302:~# gcc -o hello hello.c -lX11
cs-e9302:~# ./hello
```

Y se obtiene:

Fig. 43 Programa usando el sistema X-Window via Red

6.1.4 Instalando y Lanzando Aplicaciones Gráficas Complejas

Mediante las dos aplicaciones anteriores se pudo demostrar que el sistema X-Window está correctamente instalado y funciona perfectamente, por tanto para probar su funcionalidad y la capacidad de este sistema embebido se instalarán aplicaciones gráficas más complejas que podrían ser: Firefox, juegos, escritorios simples, etc, la imaginación es el límite siempre y cuando se respeten las limitaciones de la CS-E9302.

Para este caso se decidió instalar el editor de programación **bluefish**:

```
cs-e9302:~# apt-get install bluefish
```

Luego con el terminal gráfico con shell root aún abierto, se ejecuta el editor mediante:

```
cs-e9302:~# bluefish
```

Lo cual dará el siguiente resultado:

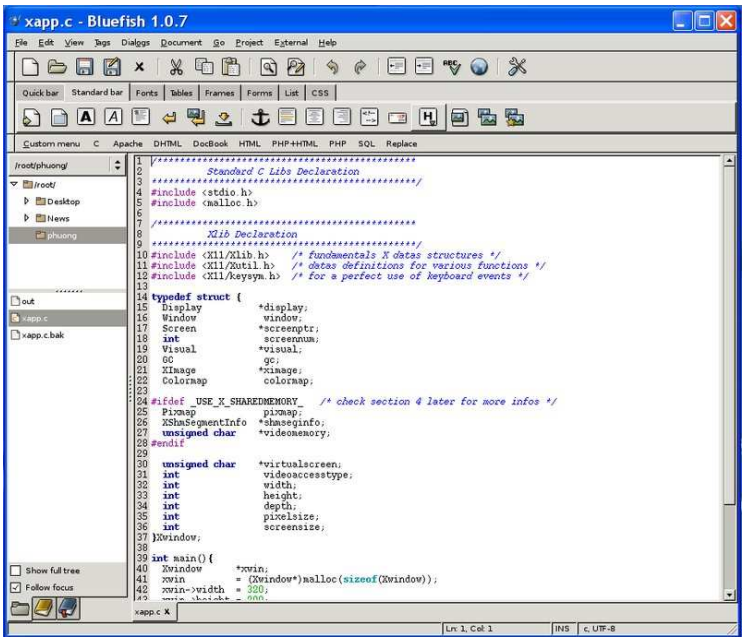


Fig. 44 Bluefish Editor

6.1.5 Servidor Web Apache 2

Una de las aplicaciones más comunes que tienen los sistemas embebidos es la de funcionar como servidores web, ya que esta función no requiere de grandes recursos físicos ni lógicos y resulta mucho más económico montar un servidor web en un sistema embebido que en un PC.

La aplicación de software libre por excelencia para montar servidores web y una de las favoritas en el mundo es el servidor web Apache (www.apache.org), fácil de instalar y configurar, además de ser liviana y robusta, pudiendo ser dotada de muchos módulos de propósitos varios, orientados desde la seguridad hasta el soporte para nuevas características del desarrollo web. Se instala mediante:

```
cs-e9302:~# apt-get install apache2
```

Al ser este un demonio, cada vez que inicie el sistema, este se cargará automáticamente verificando y cargando según la disponibilidad, todo aquello que conste en su archivo de configuración. Esta tarea no será detallada debido a que como se dijo este sistema está diseñado para el aprendizaje, y uno de los deberes de un estudiante es investigar, por tanto sólo se relatará lo esencial:

```
cs-e9302:~# vim /etc/apache/apache2.conf
```

Aquí se comentará o descomentará las líneas según sea el propósito del servidor o la funcionalidad que se desee, como los usuarios, uso de https, dirección de DNS, carpetas de destino, etc. La página que se muestra por defecto se encuentra en /usr/lib/apache2-default/, está deberá ser cambiada por la que se desee mostrar y se guardará con el nombre de index.html.

Para iniciar, parar o reiniciar el demonio hacemos lo siguiente:

```
cs-e9302:~# /etc/init.d/apache2 start
```

```
cs-e9302:~# /etc/init.d/apache2 stop
cs-e9302:~# /etc/init.d/apache2 restart
```

6.1.6 nmap

Nmap es una aplicación de software libre apoyada por toda una comunidad, cuyo principal objetivo es detectar las fallas de seguridad existentes en una red o sistema. Es conocido como el más potente escáner de puertos, aunque esto es relativo ya que su funcionalidad va más allá. También es considerada una herramienta hacker por excelencia debido a que detecta en indica de manera sencilla todos los agujeros del sistema y brinda información crítica sobre el mismo.

Al tener una conexión Ethernet de alta velocidad, se podría pensar en el sistema embebido como un servidor para auditorías de seguridad, o incluso hasta un firewall, por eso pareció atractiva la idea de implementar esta aplicación dentro de C@AM@LEON.

```
cs-e9302:~# apt-get install nmap
```

Y para escanear un host específico:

```
cs-e9302:~# nmap 192.168.0.94
```

6.1.6 Servidor MP3

Una de las principales inquietudes al trabajar con un sistema embebido, es la de preguntarse de qué es capaz el mismo, hasta dónde se puede llegar y explotarlo hasta sus límites. Pues bien para esto y dadas las características del EP-9302 se decidió trabajar con audio, a pesar de que la CS-E9302 no posee salidas ni entradas de audio, así que se planteó la posibilidad de al igual que las aplicaciones gráficas exportarlo vía

red y ver si era factible. Para ello se decidió implementar un servidor bastante novedoso y poco visto, y nunca antes visto en un sistema embebido: un MP3 Server.

Creando el Servidor

Es impresionante lo sencillo que es crear un robusto servidor Perl.

Para ello se debe asegurar de que se tiene el lenguaje Perl en el sistema, caso contrario se instala:

```
cs-e9302:~# apt-get install perl
```

Ahora se crea una carpeta de trabajo llamada MP3 que contendrá las canciones, la lista de reproducción y el código Perl del servidor.

```
cs-e9302:~# mkdir /MP3/
```

Perl es el lenguaje de programación en el que se escriben la mayoría de los módulos Debian y es un lenguaje derivado de C, muy potente, fácil y ágil, con la única desventaja de que no tiene interfaz gráfica, aunque este no es un problema en un sistema embebido. El manejo de audio y de red es más sencillo desde este lenguaje.

Para esto se usa el módulo IO::Socket, que da una sencilla interfaz orientada a objetos para tomar control de los sockets. Para empezar, este servidor necesita un socket para escuchar, comúnmente llamado "listening socket", a este socket es a donde los clientes se conectan. No es cosa del otro mundo crear este socket:

```
#creando un socket para escuchar
my $listen_socket = IO::Socket::INET->new(LocalPort => 8000,
Listen => 10,
Proto => 'tcp',
Reuse => 1);
```

Esto creará un socket para escuchar en el host local bajo el puerto 8000, usando el protocolo TCP. El parámetro "Listen" es el máximo de clientes que se puede tener en "queue" o cola. Y "Reuse" permite reiniciar el puerto 8000. Estos son los parámetros básicos a usar.

Para manejar un cliente intentando conectar, la siguiente línea creará el socket para el cliente:

```
my $connection = $listen_socket->accept;
```

Aquí la variable $connection es un objeto de socket, que puede ser manejado como un objeto de archivo. Así que se puede imprimir o leer de él como se haría con un archivo:

```
#Escribir al socket del cliente
print $connection "Hola Cliente!!";
#leer del socket del cliente
my $message = <$connection>;
```

Lo siguiente que debemos hacer tiene que ver con el "fork" de nuestro servidor.

Cuando un proceso hijo muere, no libera los recursos del sistema hasta que el proceso padre reconoce que está muerto con la llamada de las funciones "wait" o "waitpid".

Como los servidores por lo general trabajan por mucho tiempo, y se terminan varios procesos hijos, se hace necesario asegurarse de que el proceso padre se esté enterando que los procesos hijos se están muriendo, si no se da cuenta el proceso padre, entonces los procesos hijos se convierten en "zombies".

Los servidores normalmente se la pasan la mayoría de su tiempo en la llamada "accept" esperando llamadas de nuevos clientes esperando conectarse. Pero el problema, es que también está esperando que los procesos hijos se mueran, entonces ¿cómo puede hacer el servidor dos cosas a la vez? Fácil: con señales.

Cuando un proceso hijo se muere manda una señal SIGCHLD al proceso padre. Entonces nuestro servidor necesita registrar una llave de señal para llamar a la función "waitpid" cada vez que se manda la señal SIGCHLD:

```
#crear la llave de la señal
$SIG{CHLD} = \&REAPER;
#rutina para prevenir los zombies
sub REAPER{

#WNOHANG significa regresar inmediatamente si no hay hijo.
while ((waitpid(-1, WNOHANG)) >0 ){}

#resetea la señal para que el otro proceso hijo muera
$SIG{CHLD} = \&REAPER;
}
```

Entonces ya para el reproductor de MP3 no hay mucho a añadirse. Básicamente el servidor empieza, entonces un cliente llega (por ejemplo xmms o mpgl23) abriendo un socket. Entonces el servidor hace un "fork" y regresa un socket al proceso hijo. Finalmente en proceso padre regresa para seguir escuchando y esperando a otro cliente. El proceso hijo simplemente irá en un loop continúo tocando canciones de la lista aleatoriamente hasta que el cliente deje de escuchar.

Se puede crear una lista de canciones MP3 usando el Winamp o su clon GPL el XMMS, pero debido a que se está trabajando en un sistema embebido, lo más conveniente sería crear la lista desde un editor, y aunque parezca difícil no lo es.

El **M3U** (MPEG Versión 3.0 URL) es un formato de archivo que almacena listas de reproducción de medios. Es decir, es una forma de agrupar los archivos que se quieran reproducir, de manera que no haya que ir buscándolos uno a uno cada vez que se quiera reproducirlos todos a la vez. Aunque en un principio solo se podía hacer con Winamp, actualmente está disponible en varios otros como XMMS, foobar2000, JuK, Windows Media Player, iTunes, VLC y otros.

Aquí se muestra un ejemplo de archivo M3U. Muestra.mp3 y Ejemplo.ogg son los archivos de audio. 123 y 321 son las duraciones en segundos. Se utiliza una longitud de -1 para indicar que el archivo es de streaming, por lo que es imposible determinar su duración. El valor siguiente de la duración es el título que será mostrado, que normalmente es similar a la ruta del archivo que se coloca en la línea siguiente.

Primero se deben almacenar las canciones en la carpeta MP3, y luego se crea el archivo playlist.m3u:

```
cs-e9302:~# cd /MP3/
cs-e9302:/MP3# vim playlist.m3u
```

Y dentro de él se incluye lo siguiente:

```
#EXTM3U

 #EXTINF:123,Título de Muestra
/MP3/Muestra.mp3

 #EXTINF:321,Título de Ejemplo
/MP3/Ejemplo.ogg
```

Código del Servidor

Primero se crea y edita el archivo con extensión .pl, agregando el código final:

```
cs-e9302:/MP3# vim MP3Server.pl
```

```
#!/usr/bin/perl -w
use strict;
use IO::Socket;
#tomar el puerto a controlar o por default 8000
my $port = $ARGV[0] || 8000;
#ignorar procesos hijos para evitar zombies
$SIG{CHLD} = 'IGNORE';
```

```
#crear el socket a escuchar
my $listen_socket = IO::Socket::INET->new(LocalPort => $port,
Listen => 10,
Proto => 'tcp',
Reuse => 1);
#asegurarnos que estamos controlando el puerto
die "No se puede crear el listening socket: $@" unless $listen_socket;
warn "Servidosr listo!!!. Esperando conexiones ... \n";
#esperar conexiones
while (my $connection = $listen_socket->accept){
my $child;
# crear el fork para salir
die "No se puede hacer fork: $!" unless defined ($child = fork());
#¡el hijo!
if ($child == 0){
#cerrar el puerto para escuchar el proceso hijo
$listen_socket->close;
#llamar la funcion principal del hijo
play_songs($connection);
#si el hijo regresa salte
exit 0;
}
#¡soy el padre!
else{
#¿quién se conectó?
warn "Conexión recibida ... ",$connection->peerhost,"\n";
#cerrar la conexión, ya fue mandada a un hijo
$connection->close();
}
#regresa a escuchar otras conexiones
}
sub play_songs{
my $socket = shift;
#sacar todas las canciones posibles
open PLAYLIST, "playlist.m3u" or die;
my @songs = <PLAYLIST>;
```

```
close PLAYLIST;
chomp @songs;
#creador de canciones randómico
srand(time / $$);
#crear un loop eterno hasta que el cliente deje de escuhar
while(){
#Mandar el header necesario
print $socket "HTTP/1.0 200 OK\n";
print $socket "Content-Type: audio/x-mp3stream\n";
print $socket "Cache-Control: no-cache \n";
print $socket "Pragma: no-cache \n";
print $socket "Connection: close \n";
print $socket "x-audiocast-name: My MP3 Server\n\n";
#seleccionar una canción aleatoria de la lista
my $song = $songs[ rand @songs ];
#que cancion estamos tocando
warn( "play song: $song\n");
#abrir la cancion o intentar con otra
open (SONG, $song) || next;
binmode(SONG); #para usarios de windows
my $read_status = 1;
my $print_status = 1;
my $chunk;
#Esta parte imprime el binario al socket
#Lo hace lo más rápido posible
while( $read_status && $print_status ){
$read_status = read (SONG, $chunk, 1024);
if( defined $chunk && defined $read_status){
$print_status = print $socket $chunk;
}
undef $chunk;
}
close SONG;
unless( defined $print_status ){
$socket->close();
exit(0);
}
```

```
}
}
```

Ahora corremos el servidor mediante:

```
cs-e9302:/MP3# perl MP3Server.pl
```

Conectando a los clientes

Como se va a usar Winamp todo lo que hay que hacer es presionar Ctrl-L y poner:

```
http://192.168.0.94:8000
```

El número de clientes que se pueden conectar al mismo tiempo depende del ancho de banda. Se debe considerar bajar el firewall de Windows porque esto puede afectar la conexión y es necesario tener un IP único para que el servidor pueda ser accesado.

CONCLUSIONES

1. Para la construcción del sistema embebido es necesario un PC con booteo dual, Linux y Windows XP, o en su defecto dos PCs, uno Linux y uno Windows XP.

2. Para comprobar la funcionalidad del sistema embebido y sus aplicaciones es necesario manejarlo con una LAN, que incluya hosts Linux y Windows XP.

3. Al encenderse la tarjeta siempre se va a necesitar la conexión mediante RS232 para propósitos de configuración y levantamiento de ciertos parámetros, si su uso va a ser para algún tipo de servidor es necesario configurar la dirección IP estática, aunque una vez hecho esto se puede realizar mediante telnet o xterm vía red.

4. El sistema mostró una potencialidad abrumadora para aplicaciones pesadas y que manejan entornos gráficos y audio, siendo posible la implementación de prácticas referentes a estos tópicos que son muy poco vistos en sistemas embebidos de aprendizaje.

5. El resultado costo – beneficio es realmente bastante prometedor, ya que la capacidad del sistema permite divisar un amplio rango de aplicaciones dirigidas al estudiante de Ingeniería Electrónica.

6. Aunque las aplicaciones que se muestran en este trabajo están dirigidas a un área, las posibilidades de la tarjeta fácilmente la hacen útil para desarrollar aplicaciones en el área de Telecomunicaciones, Control Automático, Redes, etc.

RECOMENDACIONES

1. Se debe tener en consideración las limitaciones de la tarjeta al momento de querer instalar y correr aplicaciones, los entornos gráficos van muy bien pero tienen limitaciones.

2. Al momento en que se enciende la tarjeta y esta cargó el kernel, debe ser aislada por completo de manipulación física externa, los dispositivos conectados a ella como la memoria Flash USB y el cable Null Módem, no deben ser removidos nunca durante ejecución.

3. Checar siempre que los parámetros de Networking, variables de entorno, usuarios autorizados y permisos sean siempre los correctos dependiendo de lo que se quiera hacer.

4. En caso de problemas en el sistema de archivos, que directamente envíen a una consola de mantenimiento, correr: `fsck.ext3 /dev/sda1`.

5. En caso de redundancia cíclica, el reset por software puede no siempre funcionar, si es inevitable, presionar el botón de reset físico PRN.

6. Sería bastante importante que la Escuela de Ingeniería Electrónica pensara en un futuro próximo adquirir más de estas tarjetas y montar un Laboratorio de Sistemas Embebidos, Control y Redes.

RESUMEN

Se elaboró una tarjeta embebida con Linux que funciona como un computador en tarjeta, previo estudio comparativo, facilitando desarrollar aplicaciones embebidas, en el Laboratorio de Comunicaciones de la Escuela de Ingeniería Electrónica de la Escuela Superior Politécnica de Chimborazo, para facilitar el proceso de enseñanza-aprendizaje.

Se implementó con la tarjeta CS-E9302, mediante dos computadores Linux y WinXP, además de la incorporación de software libre como Bluefish Editor, Firefox, Nmap, Samba, etc., todo en un ambiente de desarrollo Debian, basado en el método experimental mediante investigación previa de sistemas embebidos.

La tarjeta obtuvo buenos resultados en diferentes pruebas, innovando sus aplicaciones y desempeño comparada a sistemas similares. La carga del kernel es 68% más rápida, la compilación de programas resultó ser 50% más rápida que la Cirrus Logic SBC EP9302. En la tarjeta fue montado un Servidor MP3 respondiendo a 10 aplicaciones cliente a la vez, pudiendo además lanzar un entorno gráfico vía red. Las aplicaciones demostraron funcionalidad similar a un computador de escritorio siendo éstas creadas en la misma tarjeta, además se corrió código de manipulación de periféricos de la tarjeta. La tarjeta fue elaborada en Chile por OLIMEX, siendo posible agregar módulos de expansión comerciales o desarrollados como parte de proyectos estudiantiles.

La tarjeta demostró ser una herramienta poderosa para los estudiantes en las materias de Telecomunicaciones, Control Automático y Redes, siendo más sencilla y menos costosa de usar que otros equipos como routers, firewalls, etc.

Se recomienda al Laboratorio de Comunicaciones la investigación detallada de las funciones y entornos de los sistemas embebidos.

SUMMARY

It was made an Embedded Linux board that works like a board computer, previous to a comparative study, making it easy to develop embedded applications, in Escuela Superior Politecnica de Chimborazo's Electronics School's Communication's Lab, to be easy the process of teaching-learning.

It was implemented with CS-E9302 board, using two Linux and WinXP computers, also the incorporation of free software like Bluefish Editor, Firefox, Nmap, Samba, etc., all in a Debian development environment, based on the experimental method through previous research of embedded systems.

The Board obtained good results in different tests, innovating its applications and performance compared to similar systems. Kernel's load was 68% faster, program compilation results to be 50% faster than Cirrus Logic SBC EP9302. Into the board as mounted a MP3 Server responding to 10 client applications at the same time, and it can launch also a graphical environment network via. Applications demostrated similar functionality like personal computers being this created into the Board, also peripherical Board manipulating code was ran. Board was made in Chile by OLIMEX, being possible to add commercial or students made modules.

Board demostrates to be a powerful tool for students in signatures like Telecommunications, Automated Control and Networks, being more easy and less expensive than other equipment like routers, firewalls, etc.

Is necessary that Communications Lab do research about functions and environments of embedded dydtems.

BIBLIOGRAFIA

1. BOVET, D.; CESATI, M. Understanding the Linux Kernel. 3rd ed. , San Diego: O'Reilly, 2005. 455 p.

2. CORBET, J.; RUBINI, A.; KROAHHARTMAN, G. Linux Device Drivers. 3rd ed. , New York: O'Reilly, 2005. 645 p.

3. KROAHHARTMAN, G. Linux Kernel in a Nutshell, Montreal: OReilly, 2006. 700 p.

4. LOVE, R. Linux Kernel Development. 2nd ed. , Chicago: Novell Press, 2004. 587 p.

5. RAGHAVAN, P.; LAD, A.; NEELAKANDAN, S. Embedded Linux System Design and Development, Berlin: Auerbach, 2005. 813 p.

Recursos Web:

ARCHIVOS NECESARIOS PARA CARGAR LINUX EN LA CS-E9302: kernel, patch, etc.

http://dev.ivanov.eu/projects/cs-e9302/

2009-02-14

DEBIAN GNU/Linux

http://www.debian.org

2009-05-02

LINUX EN PROCESADORES ARM

http://www.linuxarm.com/

2009-02-14

LISTA DE CORREO KERNEL LINUX

http://kerneltrap.org/

2008-12-06

LISTA DE CORREO SISTEMAS EMBEBIDOS

http://lwn.net/

2008-12-06

PORT ARM LINUX

http://www.arm.linux.org.uk/

2009-02-14

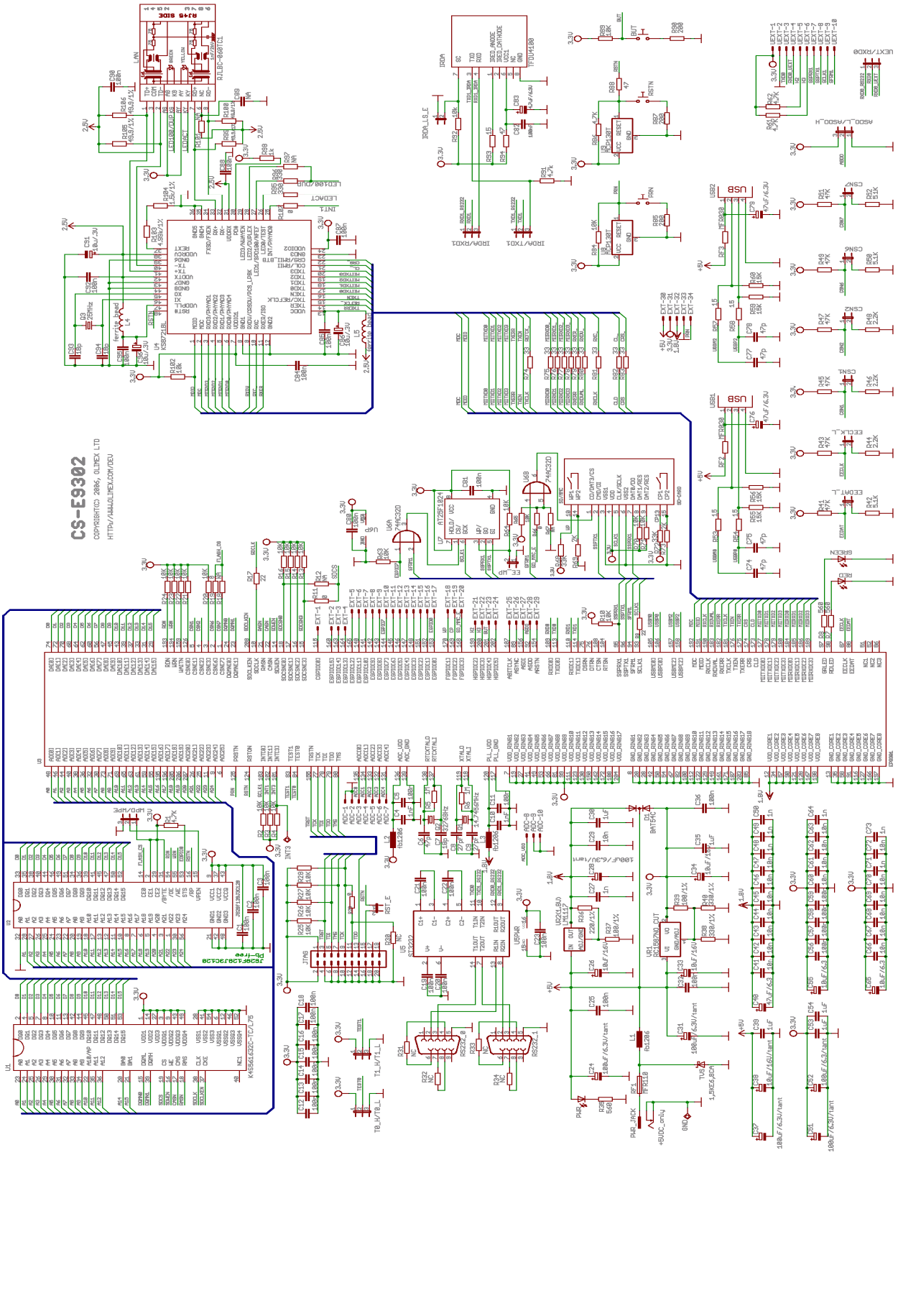
SISTEMAS EMBEBIDOS: Desarrollo de aplicaciones, audio, video, java,etc.

http://freeelectrons.com/articles/

2008-12-06

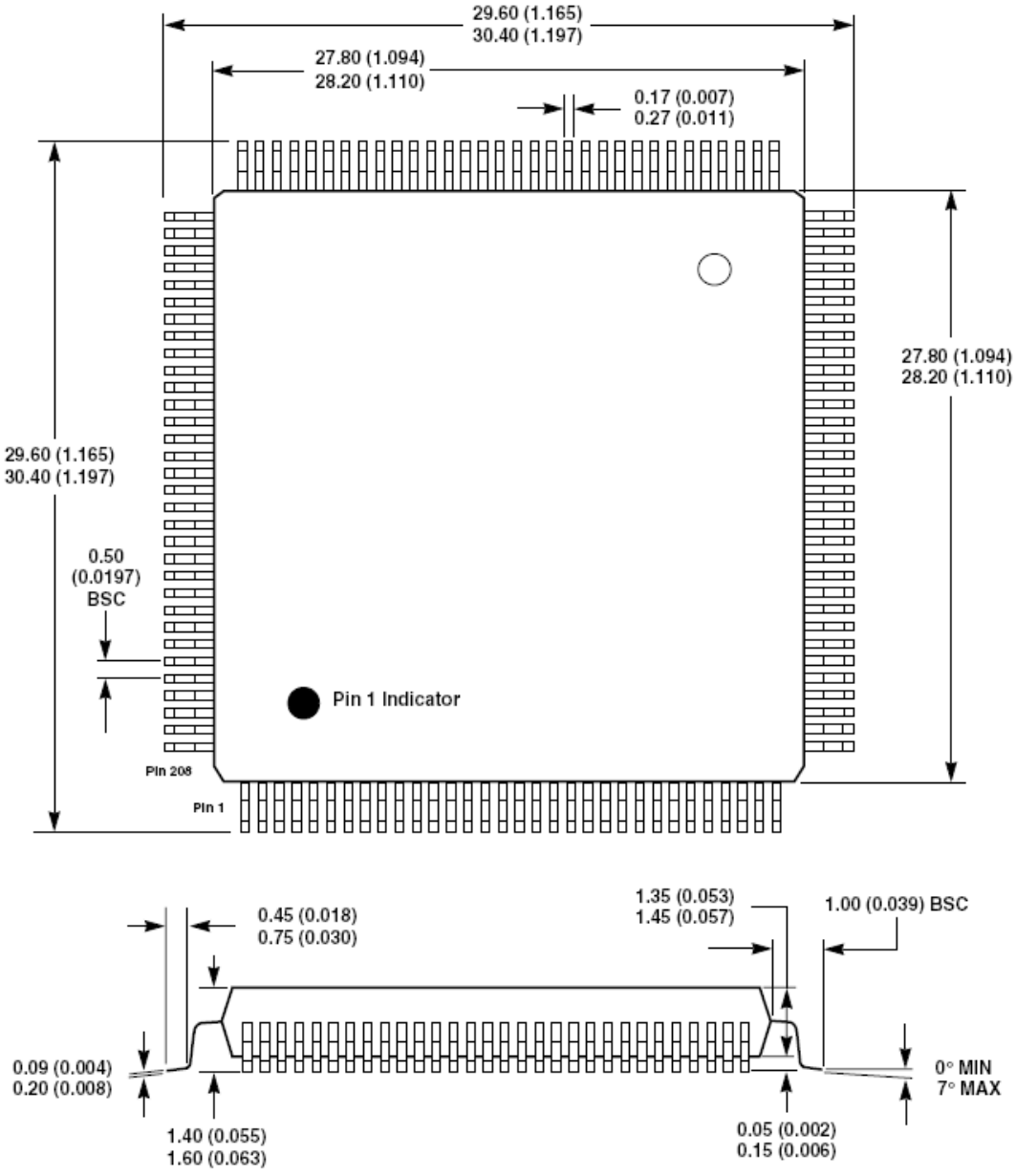
ANEXOS

ESQUEMÁTICO CS-E9302



DIMENSIONES FÍSICAS EP9302

208-Pin LQFP (28 × 28 × 1.40-mm Body)



LISTA DE PINES EP9302

Pin Number	Pin Name	Pin Number	Pin Name	Pin Number	Pin Name	Pin Number	Pin Name	Pin Number	Pin Name	Pin Number	Pin Name
1	CSn[7]	36	AD[5]	71	AD[9]	106	USBp[0]	141	EGPIO[10]	176	TXEN
2	CSn[6]	37	DA[12]	72	DA[1]	107	ABITCLK	142	EGPIO[9]	177	MIITXD[0]
3	CSn[3]	38	AD[4]	73	AD[8]	108	CTSn	143	EGPIO[8]	178	MIITXD[1]
4	CSn[2]	39	DA[11]	74	DA[0]	109	RXD[0]	144	EGPIO[7]	179	MIITXD[2]
5	CSn[1]	40	AD[3]	75	DSRn	110	RXD[1]	145	EGPIO[6]	180	MIITXD[3]
6	AD[25]	41	vdd_ring	76	DTRn	111	vdd_ring	146	EGPIO[5]	181	TXCLK
7	vdd_ring	42	gnd_ring	77	TCK	112	gnd_ring	147	EGPIO[4]	182	RXERR
8	gnd_ring	43	DA[10]	78	TDI	113	TXD[0]	148	EGPIO[3]	183	RXDVAL
9	AD[24]	44	AD[2]	79	TDO	114	TXD[1]	149	gnd_ring	184	MIIRXD[0]
10	SDCLK	45	DA[9]	80	TMS	115	CGPIO[0]	150	vdd_ring	185	MIIRXD[1]
11	AD[23]	46	AD[1]	81	vdd_ring	116	gnd_core	151	EGPIO[2]	186	MIIRXD[2]
12	vdd_core	47	DA[8]	82	gnd_ring	117	PLL_GND	152	EGPIO[1]	187	gnd_ring
13	gnd_core	48	AD[0]	83	BOOT[1]	118	XTALI	153	EGPIO[0]	188	vdd_ring
14	SDWEn	49	vdd_ring	84	BOOT[0]	119	XTALO	154	ARSTn	189	MIIRXD[3]
15	SDCSn[3]	50	gnd_ring	85	gnd_ring	120	PLL_VDD	155	TRSTn	190	RXCLK
16	SDCSn[2]	51	NC	86	NC	121	vdd_core	156	ASDI	191	MDIO
17	SDCSn[1]	52	NC	87	EECLK	122	gnd_ring	157	USBm[2]	192	MDC
18	SDCSn[0]	53	vdd_ring	88	EEDAT	123	vdd_ring	158	USBp[2]	193	RDn
19	vdd_ring	54	gnd_ring	89	ASYNC	124	RSTOn	159	WAITn	194	WRn
20	gnd_ring	55	AD[15]	90	vdd_core	125	PRSTn	160	EGPIO[15]	195	AD[16]
21	RASn	56	DA[7]	91	gnd_core	126	CSn[0]	161	gnd_ring	196	AD[17]
22	CASn	57	vdd_core	92	ASDO	127	gnd_core	162	vdd_ring	197	gnd_core
23	DQMn[1]	58	gnd_core	93	SCLK1	128	vdd_core	163	EGPIO[14]	198	vdd_core
24	DQMn[0]	59	AD[14]	94	SFRM1	129	gnd_ring	164	EGPIO[13]	199	HGPIO[2]
25	AD[22]	60	DA[6]	95	SSPRX1	130	vdd_ring	165	EGPIO[12]	200	HGPIO[3]
26	AD[21]	61	AD[13]	96	SSPTX1	131	ADC[4]	166	gnd_core	201	HGPIO[4]
27	vdd_ring	62	DA[5]	97	GRLED	132	ADC[3]	167	vdd_core	202	HGPIO[5]
28	gnd_ring	63	AD[12]	98	RDLED	133	ADC[2]	168	FGPIO[3]	203	gnd_ring
29	DA[15]	64	DA[4]	99	vdd_ring	134	ADC[1]	169	FGPIO[2]	204	vdd_ring
30	AD[7]	65	AD[11]	100	gnd_ring	135	ADC[0]	170	FGPIO[1]	205	AD[18]
31	DA[14]	66	vdd_ring	101	INT[3]	136	ADC_VDD	171	gnd_ring	206	AD[19]
32	AD[6]	67	gnd_ring	102	INT[1]	137	RTCXTALI	172	vdd_ring	207	AD[20]
33	DA[13]	68	DA[3]	103	INT[0]	138	RTCXTALO	173	CLD	208	SDCLKEN
34	vdd_core	69	AD[10]	104	RTSn	139	ADC_GND	174	CRS		
35	gnd_core	70	DA[2]	105	USBm[0]	140	EGPIO[11]	175	TXERR		

GNU/Linux most wanted

Summary of most useful commands

©Copyright 2006-2006, Free Electrons.
Free to share under the terms of the Creative Commons Attribution-ShareAlike 2.5 license
(http://creativecommons.org)

Sources, translations, updates, command and concepts details on our free training materials:
http://free-electrons.com/training/intro_unix_linux

Thanks to Michel Blanc, Hermann J. Beckers and Thierry Grellier.

Latest update: Jan 17, 2009

Handling files and directories

Create a directory:
`mkdir dir`

Create nested directories:
`mkdir -p dir1/dir2`

Changing directories:
`cd newdir`
`cd ..` (parent directory)
`cd -` (previous directory)
`cd` (home directory)
`cd ~bill` (home directory of user `bill`)

Print the working (current) directory:
`pwd`

Copy a file to another:
`cp source_file dest_file`

Copy files to a directory:
`cp file1 file2 dir`

Copy directories recursively:
`cp -r source_dir dest_dir`
`rsync -a source_dir/ dest_dir/`

Create a symbolic link:
`ln -s linked_file link`

Rename a file, link or directory:
`mv source_file dest_file`

Remove files or links:
`rm file1 file2`

Remove empty directories:
`rmdir dir`

Remove non-empty directories:
`rm -rf dir`

Listing files

List all "regular" files (not starting with `.`) in the current directory:
`ls`

Display a long listing:
`ls -l`

List all the files in the current directory, including "hidden" ones (starting with `.`):
`ls -a`

List by time (most recent files first):
`ls -t`

List by size (biggest files first)
`ls -S`

List with a reverse sort order:
`ls -r`

Long list with most recent files last:
`ls -ltr`

Displaying file contents

Concatenate and display file contents:
`cat file1 file2`

Display the contents of several files (stopping at each page):
`more file1 file2`
`less file1 file2` (better: extra features)

Display the first 10 lines of a file:
`head -10 file`

Display the last 10 lines of a file:
`tail -10 file`

File name pattern matching

Concatenate all "regular" files:
`cat *`

Concatenate all "hidden" files:
`cat .*`

Concatenate all files ending with `.log`:
`cat *.log`

List "regular" files with `bug` in their name:
`ls *bug*`

List all "regular" files ending with `.` and a single character:
`ls *.?`

Handling file contents

Show only the lines in a file containing a given substring:
`grep substring file`

Case insensitive search:
`grep -i substring file`

Showing all the lines but the ones containing a substring:
`grep -v substring file`

Search through all the files in a directory:
`grep -r substring dir`

Sort lines in a given file:
`sort file`

Sort lines, only display duplicate ones once:
`sort -u file` (unique)

Changing file access rights

Add write permissions to the current user:
`chmod u+w file`

Add read permissions to users in the file group:
`chmod g+r file`

Add execute permissions to other users:
`chmod o+x file`

Add read + write permissions to all users:
`chmod a+rw file`

Make executable files executable by all:
`chmod a+rX *`

Make the whole directory and its contents accessible by all users:
`chmod -R a+rX dir` (recursive)

Comparing files and directories

Comparing 2 files:
`diff file1 file2`

Comparing 2 files (graphical):
`gvimdiff file1 file2`
`tkdiff file1 file2`
`kompare file1 file2`

Comparing 2 directories:
`diff -r dir1 dir2`

Looking for files

Find all files in the current (`.`) directory and its subdirectories with `log` in their name:
`find . -name "*log*"`

Find all the `.pdf` files in `dir` and subdirectories and run a command on each:
`find . -name "*.pdf" -exec xpdf {} ';'`

Quick system-wide file search by pattern (caution: index based, misses new files):
`locate "*pub*"`

Redirecting command output

Redirect command output to a file:
`ls *.png > image_files`

Append command output to an existing file:
`ls *.jpg >> image_files`

Redirect command output to the input of another command:
`cat *.log | grep error`

Job control

Show all running processes:
`ps -ef`

Live hit-parade of processes (press `P`, `M`, `T`: sort by Processor, Memory or Time usage):
`top`

Send a termination signal to a process:
`kill <pid>` (number found in `ps` output)

Have the kernel kill a process:
`kill -9 <pid>`

Kill all processes (at least all user ones):
`kill -9 -1`

Kill a graphical application:
`xkill` (click on the program window to kill)

File and partition sizes

Show the total size on disk of files or directories (**d**isk **u**sage):
`du -sh dir1 dir2 file1 file2`

Number of bytes, words and lines in file:
`wc file` (**w**ord **c**ount)

Show the size, total space and free space of the current partition:
`df -h .`

Display these info for all partitions:
`df -h`

Compressing

Compress a file:
`gzip file` (`.gz` format)
`bzip2 file` (`.bz2` format, better)
`lzma file` (`.lzma` format, best compression)

Uncompress a file:
`gunzip file.gz`
`bunzip2 file.bz2`
`unlzma file.lzma`

Archiving

Create a compressed archive (**ta**pe **r**chive):
`tar zcvf archive.tar.gz dir`
`tar jcvf archive.tar.bz2 dir`
`tar cvf - dir | lzma > archive.tar.lzma`

Test (list) a compressed archive:
`tar ztvf archive.tar.gz`
`tar jtvf archive.tar.bz2`
`unlzma -c archive.tar.lzma | tar tvf -`

Extract the contents of a compressed archive:
`tar zxvf archive.tar.gz`
`tar jxvf archive.tar.bz2`
`unlzma -c archive.tar.lzma | tar xvf -`

`tar` options:
`c`: **c**reate
`t`: **t**est
`x`: e**x**tract
`j`: on the fly `bzip2` (un)compression
`z`: on the fly `gzip` (un)compression

Handling zip archives
`zip -r archive.zip <files>` (create)
`unzip -t archive.zip` (**t**est / list)
`unzip archive.zip` (extract)

Printing

Send PostScript or text files to `queue`:
`lpr -Pqueue f1.ps f2.txt` (**l**ocal **pr**inter)

List all the print jobs in `queue`:
`lpq -Pqueue`

Cancel a print job number in `queue`:
`cancel 123 queue`

Print a PDF file:
`pdf2ps doc.pdf`
`lpr doc.ps`

View a PostScript file:
`ps2pdf doc.ps`
`xpdf doc.pdf`

User management

List users logged on the system:
`who`

Show which user I am logged as:
`whoami`

Show which groups `user` belongs to:
`groups user`

Tell more information about `user`:
`finger user`

Switch to user `hulk`:
`su - hulk`

Switch to super user (`root`):
`su -` (**s**witch **u**ser)
`su` (keep same directory and environment)

Time management

Wait for 60 seconds:
`sleep 60`

Show the current date:
`date`

Count the time taken by a command:
`time find_charming_prince -cute -rich`

Command help

Basic help (works for most commands):
`grep --help`

Access the full manual page of a command:
`man grep`

Misc commands

Basic command-line calculator
`bc -l`

Basic system administration

Change the owner and group of a directory and all its contents:
`chown -R newuser:newgroup dir`

Reboot the machine in 5 minutes:
`shutdown -r +5`

Shutdown the machine now:
`shutdown -h now`

Display all available network interfaces:
`ifconfig -a`

Assign an IP address to a network interface:
`ifconfig eth0 207.46.130.108`

Bring down a network interface:
`ifconfig eth0 down`

Define a default gateway for packets to machines outside the local network:
`route add default gw 192.168.0.1`

Delete the default route:
`route del default`

Test networking with another machine:
`ping 207.46.130.108`

Create or remove partitions on the first IDE hard disk:
`fdisk /dev/hda1`

Create (format) an ext3 filesystem:
`mkfs.ext3 /dev/hda1`

Create (format) a FAT32 filesystem:
`mkfs.vfat -v -F 32 /dev/hda2`

Mount a formatted partition:
`mkdir /mnt/usbdisk` (just do it once)
`mount /dev/ubal /mnt/usbdisk`

Mount a filesystem image (loop device):
`mount -o loop initrd.img /mnt/initrd`

Unmount a filesystem:
`umount /mnt/usbdisk`

Check the system kernel version:
`uname -a`

Free Electrons — Free software for Electronic devices

LISTA DE COMANDOS LINUX UTILES POR FREE-ELECTRONS

vi basic commands

Summary of most useful commands

©Copyright 2006-2005, Free Electrons, http://free-electrons.com. Latest update: Sep 8, 2006
Free to share under the terms of the Creative Commons Attribution-ShareAlike 2.5 license
(http://creativecommons.org/licenses/by-sa/2.5/)
Sources, translations and updates on our free training materials: http://free-electrons.com/training/intro_unix_linux
Thanks to: Liubo Chen.

Entering command mode

`[Esc]` Exit editing mode. Keyboard keys now interpreted as commands.

Moving the cursor

`h` (or left arrow key) move the cursor left.
`l` (or right arrow key) move the cursor right.
`j` (or down arrow key) move the cursor down.
`k` (or up arrow key) move the cursor up.
`[Ctrl] f` move the cursor one page **f**orward .
`[Ctrl] b` move the cursor one page **b**ackward.
`^` move cursor to the first non-white character in the current line.
`$` move the cursor to the end of the current line.
`G` **g**o to the last line in the file.
nG **g**o to line number *n*.
`[Ctrl] G` display the name of the current file and the cursor position in it.

Entering editing mode

`i` **i**nsert new text before the cursor.
`a` **a**ppend new text after the cursor.
`o` start to edit a new line after the current one.
`O` start to edit a new line before the current one.

Replacing characters, lines and words

`r` **r**eplace the current character (does not enter edit mode).
`s` enter edit mode and **s**ubstitute the current character by several ones.
`cw` enter edit mode and **c**hange the **w**ord after the cursor.
`C` enter edit mode and **c**hange the rest of the line after the cursor.

Copying and pasting

`yy` copy (yank) the current line to the copy/paste buffer.
`p` **p**aste the copy/paste buffer after the current line.
`P` **P**aste the copy/paste buffer before the current line.

Deleting characters, words and lines

All deleted characters, words and lines are copied to the copy/paste buffer.

`x` delete the character at the cursor location.
`dw` **d**elete the current **w**ord.
`D` **d**elete the remainder of the line after the cursor.
`dd` **d**elete the current line.

Repeating commands

`.` repeat the last insertion, replacement or delete command.

Looking for strings

`/string` find the first occurrence of *string after the cursor.*
`?string` find the first occurrence of *string* before the cursor.
`n` find the **n**ext occurrence in the last search.

Replacing strings

Can also be done manually, searching and replacing once, and then using `n` (next occurrence) and `.` (repeat last edit).

`n,ps/str1/str2/g` between line numbers *n* and *p*, **s**ubstitute all (**g**: global) occurrences of *str1* by *str2*.
`1,$s/str1/str2/g` in the whole file ($: last line), **s**ubstitute all occurrences of *str1* by *str2*.

Applying a command several times - Examples

`5j` move the cursor 5 lines down.
`30dd` **d**elete 30 lines.
`4cw` **c**hange 4 **w**ords from the cursor.
`1G` **g**o to the first line in the file.

Misc

`[Ctrl] l` redraw the screen.
`J` **j**oin the current line with the next one

Exiting and saving

`ZZ` save current file and exit vi.
`:w` **w**rite (save) to the current file.
`:w file` **w**rite (save) to the *file* file.
`:q!` **q**uit vi without saving changes.

Going further

`vi` has much more flexibility and many more commands for power users!
It can make you extremely productive in editing and creating text.

Learn more by taking the quick tutorial: just type `vimtutor`.

Find many more resources on the net!

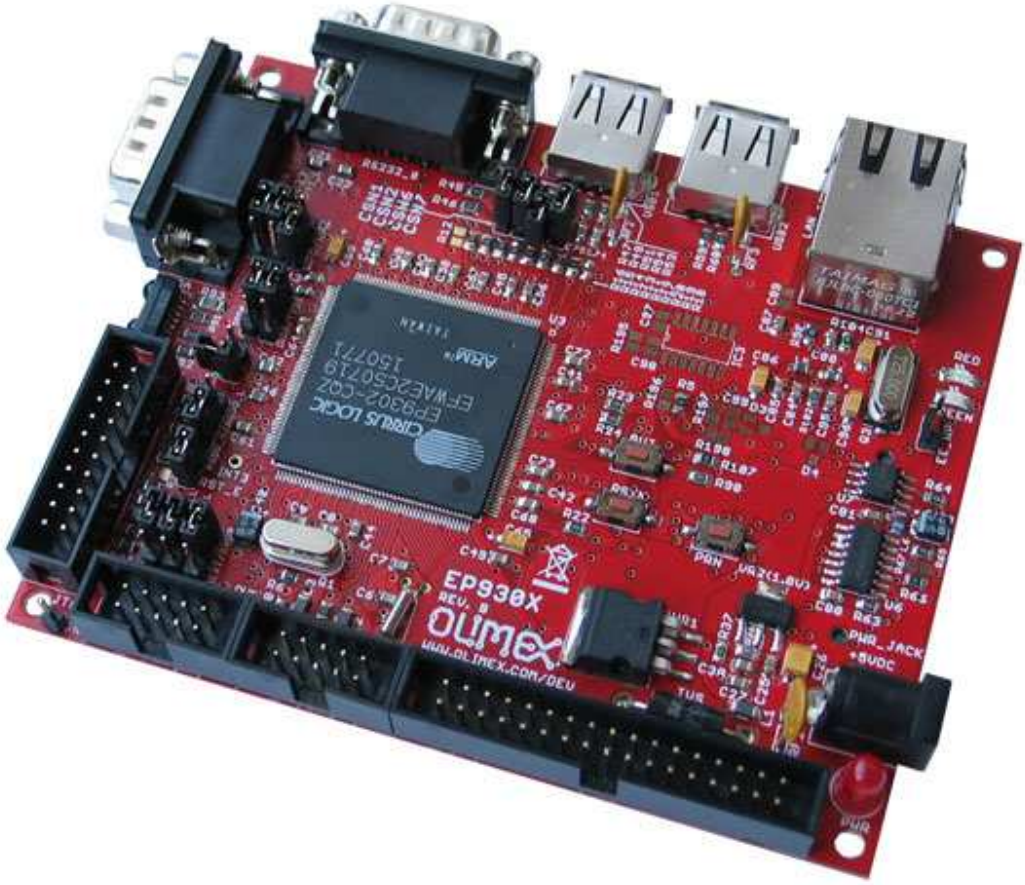
COMANDOS ÚTILES EDITOR VI Y VIM POR FREE-ELECTRONS

"Bueno, llegado a este punto cabe decir que, la mayoría del código fuente, instaladores, documentos, recursos y demás herramientas utilizadas en la presente tesis se encuentran disponibles en un DVD adjunto al Manual de Usuario que viene con esta tesis y por supuesto con la tarjeta."

Pablo

USA LA FUENTE, LUKE!!!

MANUAL DE USUARIO



C@M@LEON EMBEDDED GNU/LINUX EN LA CS-E9302

TARJETA SBC DE DESARROLLO OLIMEX CS-E9302

Descripción

La CS-E9302 es una poderosa tarjeta de desarrollo orientada al entorno Linux Embebido para los microprocesadores Cirrus Logic EP9302 ARM920T a 200 MHz con dos host USB, dos puertos RS232, Ethernet de 100 Mbps, tarjeta SD-MMC, NOR Flash de 16 MB, SDRAM de 32 MB, Transceptor IrDA (Infrarrojo), JTAG, UEXT, EXT, conectores ADC para periféricos adicionales, etc.

MICROCONTROLADOR CIRRUS LOGIC EP9302 (SoC)

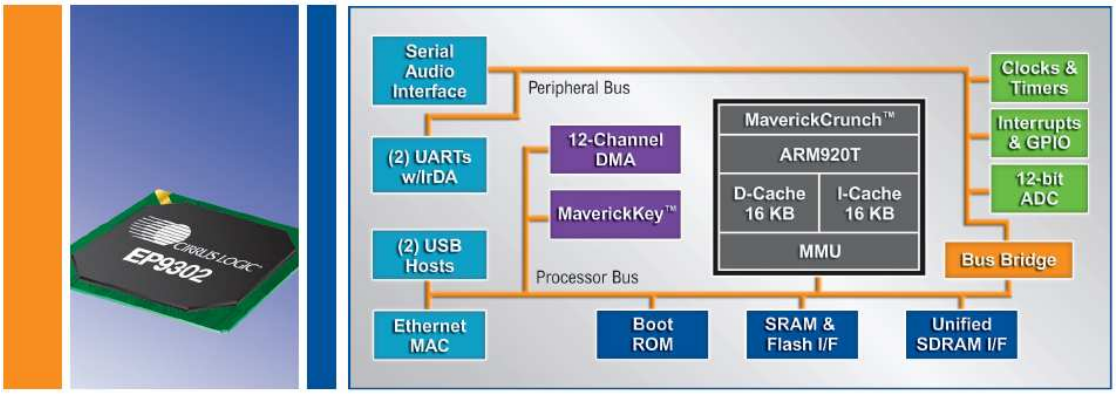


Diagrama de bloques del EP9302

Como se muestra en la Figura, se trata de un microcontrolador System-on-Chip (Sistema en Chip) con procesador de arquitectura ARM9, con un amplio rango de aplicaciones. Provee un diseño ágil que incluye un procesador ARM920T a 200 MHz con capacidad de ser usado en aplicaciones a nivel industrial y consumidor.

El procesador está basado en RISC de bajo consumo, operando desde los 1.8 V, mientras las I/O operan a 3.3 V con un consumo entre los 100 mW hasta los 750 mW, dependiendo de la velocidad.

Características

- Procesador ARM920T a 200 MHz
    - 16 KB de cache de datos y 16 KB de cache de instrucciones
    - MMU
    - Bus de sistema a 100 MHz

- Motor Matemático *MaverickCrunch*
    - Punto flotante, enteros y señal de procesamiento de instrucciones
    - Optimizado para algoritmos de compresión digital de música
    - Interbloqueo de hardware permitido en la codificación de línea

- IDs *MaverickKey* para Manejo de Derechos Digitales o Diseño de Seguridad IP
    - ID única de 32-bits
    - ID randómica de 128-bits

- Interfaces de Periféricos Integradas
    - Cinco entradas A/D con resolución de 12-bits
    - MAC Ethernet de 1/10/100 Mbps
    - Dos puertos USB 2.0 de alta velocidad (OHCI)
    - Dos UARTs (tipo 16550), incluyendo soporte soft módem
    - Interfaz IrDA, modo lento
    - Puerto SPI
    - Interfaz AC'97
    - Interfaz I^2S

- Opciones de Memoria Externa
    - Interfaz SDRAM de 16-bits, arriba de dos bancos

- SRAM/Flash/ROM I/F de 16/8 bits
- Interfaz serial EEPROM

- Periféricos Internos
  - Reloj de tiempo real con trim por software
  - 12 canales DMA para transferencia de datos que maximiza el desempeño del sistema
  - Boot ROM
  - PLLs duales controlan todo el dominio del reloj
  - Temporizador Watchdog
  - Dos temporizadores de propósito general de 16-bits
  - Temporizador de propósito general de 32-bits
  - Temporizador de depuración de 40-bits

- Entradas y Salidas de Propósito General (GPIO)
  - 16 GPIOs extendidas incluyendo capacidad de interrupciones
  - 8 GPIOs adicionales opcionales multiplexadas sobre periféricos

Aplicaciones

- Radios Internet
- Terminales de Puntos de Venta (PoS)
- Controles Industriales y de Construcción
- Rockolas Electrónicas
- Sistemas de Control Telemático
- Sistemas de Seguridad Biométricos
- Máquinas de Lotería
- Equipo de Fitness
- Sistemas de Seguridad

- Cámaras de Internet
- Mezcladores MP3
- Módems GSM
- VoIP

Características CS-E9302

- MCU: EP9302 ARM920T 200Mhz, caché 16+16KB de instrucciones y datos, MMU, SDRAM, SRAM, controlador de bus externo FLASH a 100 MHz, 100 Mbps Ethernet MAC, dos UARTS, dos puertos USB 2.0, IrDA, ADC, SPI, I^2S Audio, AC'97, DMA 12 ch, RTC, dual PLL, WDT, 2- 16 bit, 1- 32 bit TIMERS, boot ROM, controlador de interrupciones.

- SDRAM Externa de 32MB (16MB x16bit) 7.5 ns /133 MHz

- Flash Externa de 16MB (8MB x16 bit) 80 ns.

- ETHERNET 10/100 PHY KS8721BL

- 2 x RS232 drivers y conectores

- 2 x USB conectores de host

- Tarjeta SD/MMC

- Transceptor IrDA

- Conector UEXT con I^2C, SPI, RS232 y suplemento de energía para conexión de módulos add-on como RF link, MP3, etc.

- Interfaz JTAG
- Puerto de extensión ADC
- Plug in jack, para fuente de energía

Especificaciones Físicas

La CS-E9302 no debe ser sometida a potenciales electrostáticos altos, está diseñada a doble capa y debe ser manipulada con cuidado. Las dimensiones de la tarjeta son 110 x 90 mm. Esta tarjeta es ideal para desarrollo de sistemas embebidos. El consumo de la tarjeta es de alrededor de 400 a 450 mA con CPU a 200 MHz de reloj y 100 MHz de bus de sistema.

Requerimientos de Uso de la Tarjeta

Cables

- ✓ 1 cable DB9 hembra-hembra de tipo null modem para conexión serial COM con el PC.
- ✓ Cable LAN cruzado para conexión Ethernet con el PC o Cable LAN directo para conexión con switch o router.

Hardware

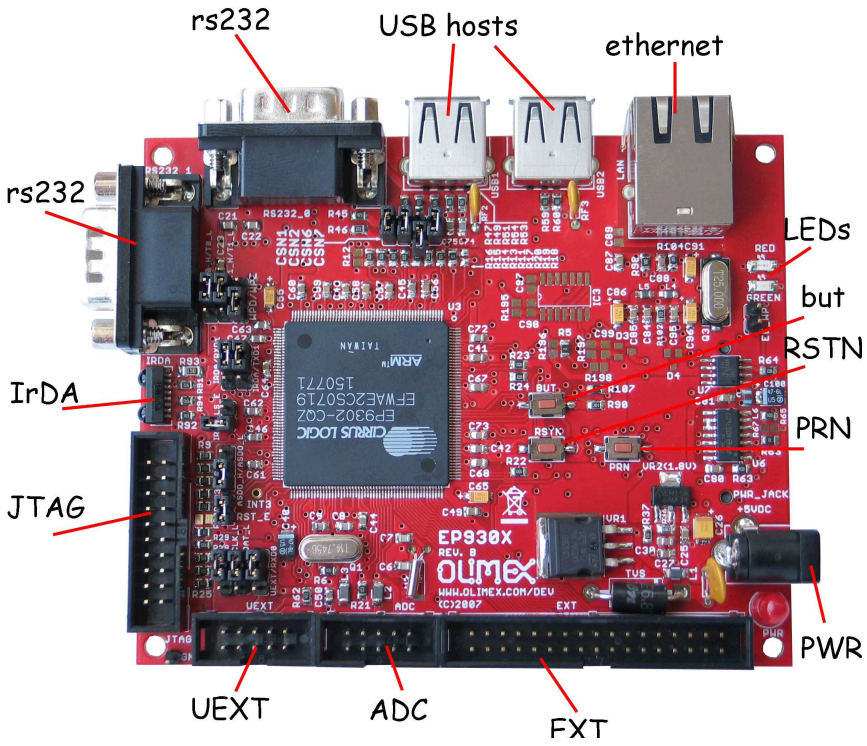
- ✓ Fuente de alimentación regulada de +5 V/1 A

- ✓ ARM-JTAG, ARM-USB-OCD, ARM-USB-TINY o alguna herramienta compatible si se desea trabajar con JTAG, aunque no es necesario si se va a trabajar con un RTOS.

Software

- ✓ Herramientas de compilación cruzada, loader, bootloader, kernel Linux 2.4 o superior, etc. Las opciones son infinitas debido a que es una tarjeta dedicada a la investigación las opciones software dependen de la pericia o necesidades del usuario.

4.3.2 Layout



Circuitería y Jumpers

Circuito de Reset

El chip EP9302 se puede resetear a través del pin PRSTN (power on reset) o a través del pin de reseteo común de drenado abierto (user reset) o RSTON.

El PRSTN es un Power on Reset realizado con un circuito de reset MCP130T con el típico umbral de 2.9 V. En la tarjeta se presenta un botón PRN el cual es usado para un reset Power-on Reset manual.

El reset de usuario RSTON es común para el microcontrolador EP9203, el chip PHY, la NOR flash, el JTAG y el circuito de supervisión MCP130. Se puede generar desde el pin 15 JTAG, botón RSTN, circuito de reset CS9302 o MCP130T.

Circuito de Reloj

El EP9302 genera su propio reloj interno con PLL y usa un simple cristal de 14.7456 MHz conectado al pin 119 (XTAL0) y el pin 118 (XTAL1).

El Reloj de Tiempo Real opera desde un cristal de 32.768 KHz.

Descripción de Jumpers

Jumpers de Configuración de Boot

Los controles de configuración de hardware se definen por un set de pines de dispositivo que son apestillados dentro de los bits de control de configuración sobre la

afirmación del reset del chip cuando se activan PRSTN o RSTON. Las diferentes configuraciones de bits hardware definen el comportamiento del Watchdog, modo de booteo (interno o externo), sincronismo en el boot, y ancho de booteo externo. Los pines apestillados son:

EECLK, selecciona booteo interno o externo el jumper correspondiente es EECLK_L. cuando el jumper se cierra el log es “0”, de otra manera el log es “1”.

EEDAT, debe ser puesto a “1”. El jumper correspondiente es EEDAT_L. Cuando el jumper se cierra EEDAT vale “0”, de otra manera vale”1”.

TEST[1:0], selecciona el modo de booteo. Los jumpers correspondientes son T0_H/T0_L y T1_H/T1_L. Cuando el jumper T0_H/T0_L se cierra hacia el lado de T0_H TEST0 vale “1”. Cuando el jumper T0_H/T0_L se cierra hacia el lado de T0_L, TEST0 vale “0”. Cuando el jumper T1_H/T1_L se cierra hacia el lado de T1_H, TEST1 vale “1”. Cuando el jumper T1_H/T1_L se cierra hacia la cara de T1_L, TEST1 vale “0”.

ASDO, selecciona booteo síncrono o asíncrono, los 3 pines correspondientes al jumper son ASDO_H/ASDO_L. Si el jumper se cierra hacia el lado de ASDO_H, ASDO vale “1”. Hacia el otro lado vale “0”.

CSN[7:6], selecciona el tamaño de boot externo, CSN6 y CSN7 son los jumpers que definen el nivel lógico de estos pines. Si están cerrados el pin correspondiente vale “0”.

EECLK	**EEDAT**	**TEST1**	**TEST0**	**ASDO**	**CSN[7:6]**	**Configuración de Booteo**
0	1	0	0	1	00 01	Booteo Externo desde el espacio de memoria Sync seleccionado por DevCfg3 a través del controlador SDRAM. El tipo de medio debe ser a su vez SROM o SyncFLASH. La selección del tamaño SRAM es determinado por el apestillado del valor de CSN[7:6]:

						8-bit SFLASH 16-bit SROM
0	1	0	0	0	00 01	Booteo externo desde el espacio de memoria Async seleccionado por nCS0 a través del Controlador de Memoria Síncrona. La selección del tamaño de SRAM se determina por el apestillado del valor de CSN[7:6]: 8-bit SRAM 16-bit SRAM
1	1	0	1	X	01	Booteo serial de 16-bits
1	1	0	0	1	00 01	Booteo Externo desde una NOR-FLASH. La selección del tamaño del bus se determina por el valor del apestillamiento de CSN[7:6]: 8-bit 16-bit–DEFAULT POSITION–NORMAL BOOT
1	1	0	0	0	00 01	Boot Interno desde el chip ROM. La selección del tamaño de la ROM s determina por el valor del apestillamiento de CSN[7:6]: 8-bit 16-bit

Solo dos selecciones son funcionales para el usuario, Booteo Normal y Booteo Serial. Los otros modos se reservan para uso de fábrica solamente.

La selección marcada en azul es la posición por defecto de los jumpers.

Configuración de Jumpers de Watchdog

CSN1, deshabilita el temporizador reset del Watchdog. Si el jumper CSN1 se cierra en el pin correspondiente (CSN1) vale “0”, de otra manera vale“1”.

CSN2, deshabilita la duración del reset del Watchdog. Si el jumper CSN2 se cierra en el pin correspondiente (CSN2) vale “0”, de otra manera vale “1”.

CSN1	CSN2	Opción de Inicio
0	0	Watchdog deshabilitado, Duración de reset deshabilitado
0	1	Watchdog deshabilitado, Duración de reset activo
1	0	Watchdog activo, Duración de reset deshabilitado
1	1	Watchdog activo, Duración de reset activo

La selección marcada en azul es la posición por defecto de los jumpers.

Jumpers de Uso General

WPD/WPE, define el estado de protección de escritura de la NOR-FLASH U2(JS28F128J3C120) el estado de WPD (1-2 en corto) habilita la escritura de la flash, el estado WPE (2-3 en corto) deshabilita la escritura de la flash. La posición por defecto es 1-2 en corto.

EE_WP, cuando el jumper EE_WP se cierra deshabilita la escritura de la EEPROM U7(AT25F1024). La posición por defecto es abierto.

IRDA/RXD1, el jumper IRDA/RXD1 define conexión RXD1 (pin 110 del MCU). Cuando el jumper es instalado en el estado IRDA (2-3 en corto), entonces la función del RDX1 del MCU es conectado al módulo IrDA. Si se selecciona el estado RXD1 (1-2 en corto), el pin RXD1 está conectado a través del driver RS232 U5 al conector RS232_1. La posición por defecto es 1-2 en corto.

IRDA/TXD1, el jumper IRDA/TXD1 define la conexión TXD1 (pin 114 del MCU). Cuando el jumper está instalado en estado IRDA (2-3 en corto), entonces la función TXD1 del MCU es conectado al módulo IRDA. Si el estado TXD1 se selecciona (1-2 en corto), el pin TXD1 está conectado a través del driver U5 RS232 al conector RS232_1. La posición por defecto es 1-2 en corto.

IRDA_LS_E, el jumper IRDA_LS_E se usa si se debe seleccionar el modo baja velocidad del chip IRDA. Cuando el jumper se cierra, el usuario habilita el modo baja velocidad. La posición por defecto es abierto.

RST_E, cuando el jumper RST_E está cerrado, el pin15 del conector JTAG está conectado a la línea RSTN (User Reset) o permite al JTAG reseteado. La posición por defecto es cerrado

UEXT/RXD0, el jumper UEXT/RXD0 define conexión RXD0 (pin 109 del MCU).cuando el jumper se instala en estado UEXT, entonces las función RXD0 del MCU es conectado al conector UEXT. Si el estado RXD0 se selecciona, el pin RXD0 está conectado a través del driver U5 RS232 al conector RS232_0. La posición por defecto es 1-2 en corto.

ENTRADA/SALIDA

Dos botones de reset con nombres RSTN y PRN conectados al pin 124 (RSTON) y al pin 125 (PRSTN) del chip EP9302.

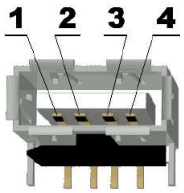
Un botón de usuario con nombre BUT conectado al pin 201 HGPIO[4].

Dos LEDs, rojo conectado al pin 98 RDLED y verde conectado al pin 97 GRLED.

LED rojo de Fuente de alimentación con nombre PWR que indica que la fuente está conectada.

Descripción de Conectores

USB1, USB2

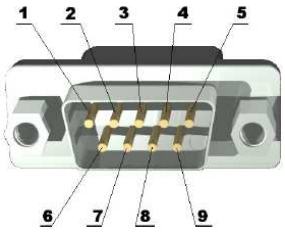


USB1		USB2	
Pin #	**Signal Name**	**Pin #**	**Signal Name**
1	+5V	1	+5V
2	USBM0	2	USBM2
3	USBP0	3	USBP2
4	GND	4	GND

- USBM0 y USBP0 forman la entrada/salida diferencial del USB host 0.
- USBM1 y USBP1 forman la entrada/salida diferencial del USB host 1.

La USB Open Host Controller Interface (Open HCI) provee puertos de comunicaciones a alta velocidad a un baud rate de 12 Mbps Encima de los 127 dispositivos USB (impresora, mouse, cámara, teclado, etc.) y hubs USB se pueden conectar en el USB host en la topología USB “tieredstart”. El puerto USB está de acuerdo a las especificaciones de USB 2.0 y soporta ambas baja velocidad (1.5 Mbps) y alta velocidad (12 Mbps).

RS232_0, RS232_1



RS232_0		RS232_1	
Pin #	**Signal Name**	**Pin #**	**Signal Name**
1	NC	1	NC
2	RXD_0	2	RXD_1
3	TXD_0	3	TXD_1
4	pin 6	4	pin 6
5	GND	5	GND
6	pin 4	6	pin 4
7	pin 8	7	pin 8
8	pin 7	8	pin 7
9	NC	9	NC

El UART0 soporta un flujo de bits arriba de los 115.2 Kbps, soporta HDLC e incluye 16 bytes FIFO para recibir y 16 byte FIFO para transmitir.

El Shell del sistema Linux embebido a instalarse será accesible desde el conector DB9 macho en RS232_0. La configuración del terminal será: 57600 Kbps, 8 data bits, no parity, 1 stop bit, no flow control.

El UART1 contiene un codificador IrDA operando en modo lento (arriba de 115 Kbps), medio (0.576 o 1.152 Mbps), o rápido (4 Mbps). También tiene 16 bytes FIFO para recibir y 16 bytes FIFO para transmitir.

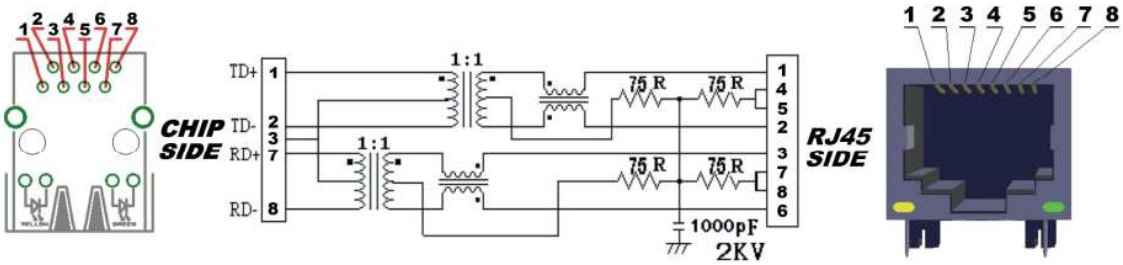
- TXD_0, Transmit Data 0 (Salida). Esta es la salida serial asíncrona de datos (RS232) para el cambio de registro en el controlador UART0 (éste pin es entrada para el RS232 y entrada para el LPC2478)

- RXD_0, Receive Data 0 (Entrada). Esta es la entrada serial asíncrona de datos (RS232) para el cambio de registro en el controlador UART0 (éste pin es salida para el RS232 y entrada para el LPC2478).

- TXD_1, Transmit Data 1 (Salida). Esta es la salida serial asíncrona de datos (RS232) para el cambio de registro en el controlador UART1 (éste pin es entrada para el RS232 y entrada para el LPC2478).

- RXD_1, Receive Data 1 (Entrada). Esta es la entrada serial asíncrona de datos (RS232) para el cambio de registro en el controlador UART1 (éste pin es salida para el RS232 y entrada para el LPC2478).

ETHERNET:



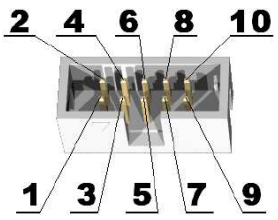
La CS-E9302 posee un subsistema MAC físico completo. El EP9302 soporta 1/10/100 Mbps de transferencia de datos. El subsistema MAC cumple con la topología ISO/TEC 802.3 para un único medio compartido con varias estaciones. Están soportadas múltiples PHYs que cumplen con MII. La capa PHY está hecha con el chip MICREL PHY KS8721BL.

Pin #	Signal Name Chip Side	Pin #	Signal Name Chip Side
1	TD+	5	NC
2	TD-	6	2.5V
3	2.5V	7	RD+
4	NC	8	RD-

LED	Color	Uso
Izquierda	Amarillo	Actividad
Derecha	Verde	100 Mbps (Half/Full dúplex)

- TD- OutputDifferential. Esta señal es una salida del MCU.
- TD+ OutputDifferential. Esta señal es una salida del MCU.
- RD- InputDifferential. Esta señal es una entrada del MCU.
- RD+ InputDifferential. Esta señal es una entrada del MCU.

UEXT:

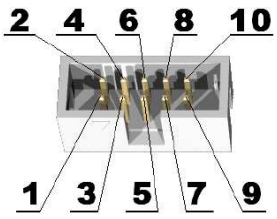


Pin #	Signal Name	Pin #	Signal Name
1	3.3V	2	GND
3	TXD0	4	RXD0_UEXT
5	H2	6	H3
7	SSPRX1	8	SSPTX1
9	SCLK1	10	SFRM1

El UEXT es un conector universal OLIMEX con 3.3 V de alimentación e interfaz UART, SPI más los puertos del MCU, HGPIO[2] y HGPIO[3] conectados a los pines del UEXT 5 y 6. Otros dispositivos o módulos se pueden conectar con estas interfaces a través del UEXT, por ejemplo:

- MOD-NRF24L - http://www.olimex.com/dev/mod-nrf24L.html
- MOD-RFID125- http://www.olimex.com/dev/mod-rfid125.html
- MOD-MP3 - http://www.olimex.com/dev/mod-mp3.html

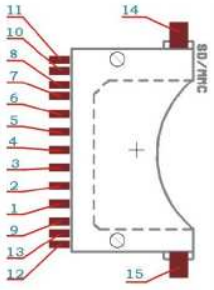
ADC:



Pin #	Signal Name	Pin #	Signal Name
1	GND	2	GND
3	ADC0	4	ADC1
5	ADC2	6	ADC3
7	ADC4	8	ADC_VDD(3.3V)
9	GND	10	GND

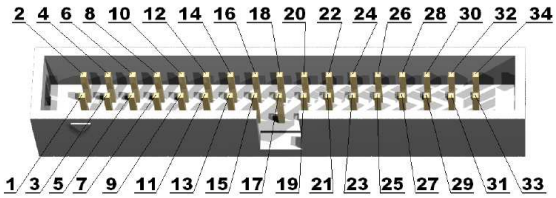
El bloque ADC consiste en un conversor Analógico a Digital de 12-bits con una entrada analógica multiplexora. El multiplexor puede seleccionarse para medir voltaje de baterías y otros voltajes misceláneos en los pines externos de medición. La señal debe estar entre los 0 a 3.3 V, el ADC se completa con la posibilidad de interrupción.

Interfaz de Tarjetas SD/MMC



Pin #	Signal Name
1	CD_MMC_E/SFRM1
2	SSPTX1
3	GND
4	VCC (470nH to 3.3V)
5	SCLK1
6	GND
7	SSPRX1
8	10K to VCC
9	10K to VCC
10	WP (33K to VCC)
11	WP (2K to GND)
12	CP (2K to GND)
13	WP (33K to VCC)
14	2K to GND
15	2K to GND

EXT

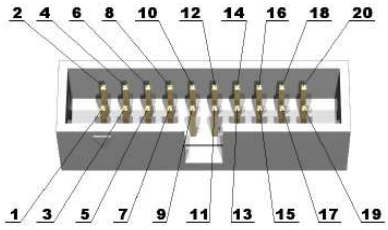


Pin #	Signal Name	Pin #	Signal Name
1	CGPIO[0]	2	EGPIO[15]
3	EGPIO[14]	4	EGPIO[13]
5	EGPIO[12]	6	EGPIO[11]
7	EGPIO[10]	8	EGPIO[9]
9	EGPIO[8]	10	EGPIO[7]

11	EGPIO[6]	12	EGPIO[5]
13	EGPIO[4]	14	EGPIO[3]
15	EGPIO[2]	16	EGPIO[1]
17	EGPIO[0]	18	FGPIO[1]
19	FGPIO[2]	20	FGPIO[3]
21	HGPIO[2]	22	HGPIO[3]
23	HGPIO[4]	24	HGPIO[5]
25	ABITCLK	26	ASYNC
27	ASDI	28	ASDO
29	ARSTN	30	+5V(Out)
31	3.3V(Out)	32	1.8V(Out)
33	GND	34	PRN(Power on Reset)

En este conector se encuentran disponibles todas las GPIOs del EP9302 las cuales no están siendo usadas para otros propósitos. Con éstas se pueden agregar módulos hardware add-on para ser controlados con la CS-E9302 como GSM/GPS, tarjetas input/output add-on.

JTAG



Pin #	Signal Name	Pin #	Signal Name
1	+3.3V	2	+3.3V
3	TRST	4	GND
5	TDI	6	GND
7	TMS	8	GND
9	TCK	10	GND

11	TCK/GND	12	GND
13	TDO	14	GND
15	RSTN/NC	16	GND
17	NC	18	GND
19	NC	20	GND

El conector JTAG permite al software de depuración comunicarse vía puerto JTAG (Joint Test Action Group) directamente al núcleo. Las instrucciones deben ser insertadas y ejecutadas por el núcleo permitiéndole a la memoria del MCU ser programada con código ejecutado paso a paso. Cumple con el estándar IEEE 1149.1 - 1990 Standard Test Access Port.

- TDI, Test Data In. Esta es la entrada serial de datos para cambio de registro

- TDO, Test Data Out. Esta es la salida serial de datos para cambio de registro. Los datos se cambian en el dispositivo en el límite negativo de la señal TCK.

- TMS, Test Mode Select. El pin TMS selecciona el estado siguiente en la máquina de estado TAP.

- TCK, Test Clock. Esto permite el cambio interno de los datos, en los pines TMS y TDI. Es un reloj de disparo positivo con las señales TMS and TCK que definen el estado interno del dispositivo.

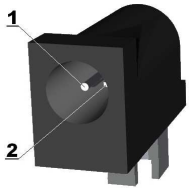
- TRST, Test Reset. Esta señal resetea el controlador JTAG

- TCK, Clock. Esta es una señal de sincronización usada por el conector JTAG para saber si está listo para recibir/transmitir.

- RSTN, Reset. Esta señal resetea el MCU.

Este conector comparte el esquema ARM JTAG por defecto.

PWR



Pin #	Signal Name
1	+5VDC ONLY
2	GND

LABORATORIO 1

Kubuntu 8.10 i38

Kubuntu GNU/Linux, es una variante derivada de Ubuntu que a su vez es una distribución basada en Debian, considerado por muchos el sistema Linux por excelencia debido a su estabilidad e infinidad de repositorios. Kubuntu se diferencia de Ubuntu al implementar el escritorio KDE, el cual es mucho más vistoso visualmente que el escritorio Gnome usado por excelencia en Ubuntu. Esta versión en particular implementa un escritorio KDE 4.1 bastante agradable y manejable.

Algunos Consejos Útiles

- ✓ Técnicamente se debería usar el usuario **root** (superusuario) para operaciones que requieran privilegio de superusuario, como montar un sistema de archivos, cargar un módulo del kernel, cambiar permisos, configuración de la red, etc. La mayoría de tareas regulares como descargar, extraer fuentes, compilar, pueden ser realizadas como usuario regular.
- ✓ Si los comandos son corridos desde un shell root por error, el usuario regular ya no podrá mas manipular los archivos generados correspondientes a la ejecución del comando. En este caso se debe reasignar permisos con el comando chown -R por ejemplo: chown –R myuser:myuser /mnt/labs/
- ✓ En Debian, Ubuntu, Kubuntu y sus derivados, no debe causar sorpresa si las aplicaciones gráficas no corren desde el usuario root. Se debe setear la variable DISPLAY con las mismas opciones que en el usuario regular, pero por motivos de seguridad eso no es recomendable, es preferible correr dichas aplicaciones desde un usuario regular.
- ✓ Si se está corriendo Kubuntu en modo LiveCD, no guardar datos en /home/Ubuntu. De otra manera el sistema se congelará rápidamente debido a que /home en este modo está residente en la RAM.

Diferentes Opciones de Configuración

Existen varias opciones para usar Kubuntu en el desarrollo de aplicaciones:

- ✓ La mejor opción es instalar Kubuntu en el disco duro. Esto permite al sistema ser rápido y ofrecer todas las características de un sistema instalado.
- ✓ La segunda opción es usar Kubuntu en modo LiveCD. Sin embargo, en este modo, información personal se guarda en RAM, lo cual puede no ser suficiente para el propósito de nuestro sistema. La solución es crear un gran archivo dentro de una partición existente.
- ✓ La tercera opción es instalar Kubuntu en una máquina virtual.

Debido al propósito de esta tesis se eligió instalar Kubuntu en el disco duro, conviviendo con Windows XP con la opción de booteo dual, en un PC con las siguientes características:

- CPU Core 2 Duo E4500 a 2.2 GHz
- 2 GB de RAM
- Fastethernet 1/10/100 Mbps
- 8 hosts USB
- Unidad súper multi DVD
- Puerto COM RS232
- Periféricos normales de un PC

Instalación de Kubuntu

Para poder trabajar en buenas condiciones, se necesitan al menos 10 GB de espacio disponible en disco.

La manera estándar para crear espacio es particionando una partición Windows. Esto se puede hacer directamente desde la instalación de Kubuntu, siendo la Fig. 27 un ejemplo típico:

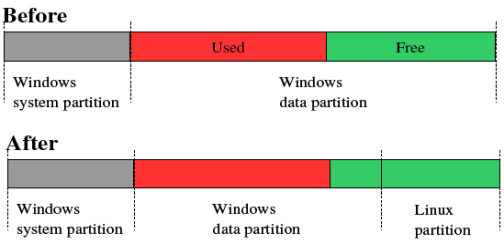


Fig. 26 Particionamiento

Configurando la Red y los Repositorios

Hay que asegurarse primeramente que se puede acceder a la red propiamente, si esto no sucede se deberá ajustar las opciones en “System Settings” disponibles en el menú KDE.

Luego debemos asegurarnos que los repositorios de paquetes están propiamente habilitados, corriendo el gestor de paquetes Adept (System -> Adept Manager en el menú KDE).

Si el acceso a Internet funciona usando un proxy agregar la siguiente línea a /etc/apt/apt.conf :

Acquire::http::Proxy "http://ProxyServer:Port";

Finalmente aunque se comentó sobre el uso del usuario **root** sólo para tareas administrativas que requieran privilegios de superusuario, es bastante tedioso tener que cambiar a cada momento de usuario, por lo tanto la mayor parte del trabajo se realizó desde un shell root. En las distribuciones basadas en Debian como Ubuntu y Kubuntu el usuario root viene deshabilitado por defecto, pero esto se soluciona fácilmente dándole una contraseña a este usuario con:

$ passwd root

Y para cambiar desde cualquier usuario a root:

$ su

Y cuando sea necesario cambiar a algún otro usuario:

$ su <usuario>

apt-get

Este comando es propio de Debian y es muy útil e importante como se verá más adelante para lo que a nuestro sistema embebido se refiere, ya que baja los programas de los repositorios, y además calcula dependencias y las instala también. Sin esta herramienta, tendríamos que compilar nosotros los paquetes y resolver dependencias. Dependencias son los paquetes o librerías que necesita un programa para funcionar. Sin esta herramienta pues, es un verdadero lío instalar cosas.

Cuando instalamos Kubuntu, por defecto esta herramienta buscará en el CD de instalación los paquetes a instalar. Para decirle que los busque en internet, habrá que editar un fichero, está en /etc/apt/sources.list . En este fichero están las direcciones donde ir a buscar los paquetes. Siendo root, habrá que comentar (poner un # delante) la línea donde dice que busque en el CD y descomentar las otras líneas dónde hayan direcciones. Cada una de estas direcciones indica las diferentes categorías de paquetes soportados por Kubuntu.

Una vez descomentadas esas líneas, ya podemos actualizar nuestro sistema. Lo primero es hacer un "apt-get update", con ello actualizamos toda la lista de paquetes de la que tenemos constancia. Es decir, nos conectamos a las direcciones anteriores y nos bajamos la lista de paquetes que hay.

Ahora ya se puede instalar aplicaciones o paquetes, para ello se ejecuta "apt-get install <programa>". Como por ejemplo "apt-get install apache2" instalará el servidor web apache en nuestra máquina y lo arrancará (lo podemos probar tecleando la dirección localhost en firefox).

Cuando no se recuerda el nombre completo de un paquete, puede ser buscado con "apt-cache search aproximación", por ejemplo, si no se recuerda como era el servidor para telnet, se teclea "apt-cache search telnet" y me aparecen varios paquetes, los examino un poco y veré que hay un paquete llamado "telnetd - telnet server", así pues, tecleo "apt-get install telnetd" y ya tengo el servidor de telnet en el sistema.

Para desinstalar programas, se usa "apt-get remove programa". Si queremos borrar también los ficheros de configuración del programa, usamos "apt-get remove --purge programa". El sistema de apt-get no es muy óptimo desinstalando paquetes. Ya que si para instalarlo tuvo que instalar

también una librería, a la hora de desinstalar no quita ésta. Para solucionarlo podemos usar otro gestor de paquetes, como Aptitude, que funciona igual que apt-get, para esto hago un "apt-get install aptitude".

Para actualizar todos los paquetes del sistema, se usa "apt-get dist-upgrade", esto bajará todos los nuevos programas.

De vez en cuando, es recomendable hacer un "apt-get update" y "apt-get dist-upgrade" para ir consiguiendo las actualizaciones de seguridad.

LABORATORIO 2

arm-linux-gcc

Este es nuestro compilador cruzado, resulta conveniente bajarse directamente el arm-linux-gcc en su versión comprimida .tar.bz2 para luego ser descomprimido y ubicado correctamente donde corresponde. Dicho paquete se puede descargar de la siguiente dirección:

http://arm.cirrus.com/files/tools/arm-linux-gcc-4.1.1-920t.tar.bz2

Una vez descargado se descomprime y copia a su ubicación correspondiente de la siguiente manera, como se mencionó anteriormente y de aquí en adelante no se volverá a mencionar, es preferible hacer esto desde un shell root:

```
# cd cs-e9302
# gunzip arm-linux-gcc-4.1.1-920t.tar.bz2
# tar -xvf arm-linux-gcc-4.1.1-920t.tar
# cp –r /4.1.1-920t/* /usr/local/arm
```

Para el correcto funcionamiento del compilador cruzado es necesario cambiar la variable de entorno PATH, indicándole la ubicación de los binarios:

```
# export PATH=/usr/local/arm/bin:$PATH
```

LABORATORIO 3

Compilando y Parchando el Kernel

Primeramente Kubuntu no trae instalada la utilidad **patch** así que toca agregarla e instalarla en el sistema, siendo ésta una tarea sencilla mediante el comando **apt-get**.

```
# apt-get install patch
```

Y los pasos que siguen se refieren propiamente al kernel y su parchado para soporte SD/MMC:

```
# cd cs-e9302
# wget http://www.kernel.org/pub/linux/kernel/v2.6/testing/v2.6.24/linux-2.6.24-rc8.tar.bz2
# wget http://dev.ivanov.eu/projects/cs-e9302/linux-2.6.24-rc8_ep93xx_mmc.patch.gz
# gunzip linux-2.6.24-rc8.tar.bz2
# tar -xvf linux-2.6.24-rc8.tar
# cd linux-2.6.24-rc8
# zcat ../linux-2.6.24-rc8_ep93xx_mmc.patch.gz | patch -p1

patching file arch/arm/mach-ep93xx/core.c
patching file cs-e9302_kernel_config
patching file drivers/net/arm/ep93xx_eth.c
patching file drivers/net/arm/ep93xx_eth.h
patching file drivers/spi/Kconfig
patching file drivers/spi/Makefile
patching file drivers/spi/spi_ep93xx.c
patching file include/asm-arm/arch-ep93xx/ep93xx-regs.h

# cp cs-e9302_kernel_config .config
```

Antes de configurar el kernel, el Makefile del linux-2.6.24.rc8 debe modificarse:

```
# cd linux-2.6.24-rc8
# vim Makefile
```

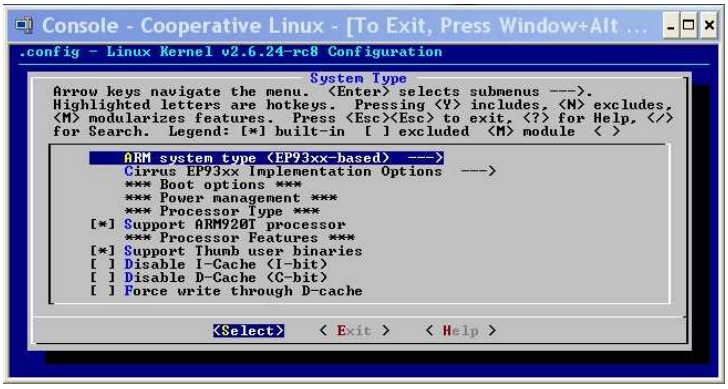
Reemplazar ARCH ?= $(SUBARCH) y CROSS_COMPILE ?= como sigue:

```
ARCH ?= arm
CROSS_COMPILE = arm-linux-
```

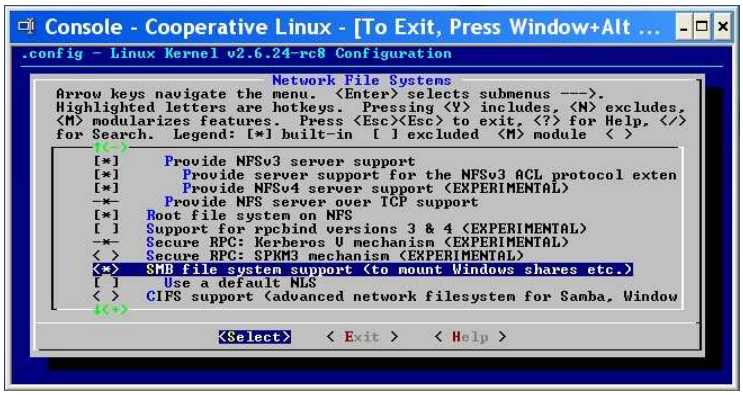
Y, el cs-e9302_kernel_config .config el cual tiene la información necesaria de la tarjeta CS-E9302 para compilar el kernel V2.6. Más aún, si se quiere que el kernel soporte más funciones como smbfs, network file system (NFS), se debe tratar con "make menuconfig":

```
# cd linux-2.6.24-rc8
# make oldconfig
# make menuconfig
```

Asegurarse de que el tipo de sistema es "EP93xx-based" y la plataforma "Support Cirrus Logic EDB9302":



Y para más funciones se puede marcar "Root file system on NFS" y "SMB file system support" para usar un sistema de archivos raíz vía red y montar entornos compartidos de Windows.



Finalmente, se usa el comando make para compilar el kernel y crear la imagen:

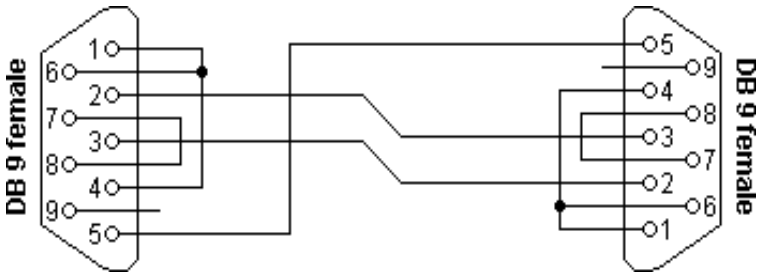
```
# make
# make zImage
```

LABORATORIO 4

Cable NULL MODEM y HyperTerminal

El PC se conecta a la CS-E9302 via conexión UART (RS232_0 para el lado de la CS-E9302). No se necesita la señal de feedback en el PC, pero el cable NULL MODEM es necesario para una correcta operación, de otra manera no se podrá escribir ningún carácter en la ventana del HyperTerminal desde el teclado.

He aquí el cable NULL MODEM con loopback handshaking, que se fabricará:

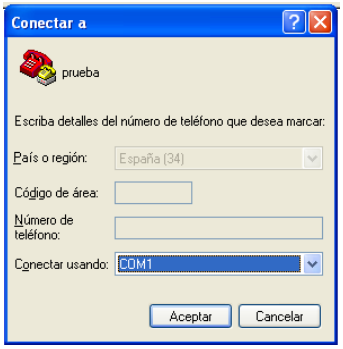


Conector 1	Conector 2	Función

2	3	Rx <= Tx
3	2	Tx => Rx
5	5	GND
1 + 4 + 6	-	DTR => CD + DSR
-	1 + 4 + 6	DTR => CD + DSR
7 + 8	-	RTS => CTS
-	7 + 8	RTS => CTS

Se usará el HyperTerminal de Windows como una ventana de prompt de comandos, el cual es un programa de comunicación serial de datos que usa el puerto COM del computador con conexión RS232, para mostrar la línea de comandos o la consola de un dispositivo, por lo general, carente de pantalla.

Primero se conecta la tarjeta al puerto serial (COM1/COM2) en el computador y después se corre HyperTerminal. Se configura el puerto serial a 57600; 8,N,1 y sin control de flujo. Luego se enciende la tarjeta. Los pasos para configurarlo son:


Conectar a
prueba
Escriba detalles del número de teléfono que desea marcar:
País o región:
España (34)
Código de área:
Número de teléfono:
Conectar usando:
COM1
Aceptar
Cancelar


Propiedades de COM1
Configuración de puerto
Bits por segundo:
57600
Bits de datos:
8
Paridad:
Ninguno
Bits de parada:
1
Control de flujo:
Ninguno
Restaurar predeterminados
Aceptar
Cancelar
Aplicar

LABORATORIO 5

root fs basado en Debian 4.0 etch

Para poder cargar aplicaciones, realizar tareas administrativas y para manejo de funciones críticas del kernel, es necesario manejar archivos, ya que en Linux, todo se maneja por medio de archivos, desde los drivers y enlaces entre librerías y aplicaciones, hasta los dispositivos hardware, por eso es necesario instalar un sistema de archivos en algún dispositivo de almacenamiento no volátil, el cual cumpla las funciones que a su vez hace un disco duro en un PC, en nuestro caso una memoria Flash USB.

Primeramente, en el PC creamos un dispositivo ext3 para el root fs, conectamos un drive USB FLASH. Descargamos un root fs disponible por una comunidad de desarrolladores, y asumiendo que el nombre del dispositivo de almacenamiento es /dev/sdb1:

```
# cd cs-e9302
# wget http://dev.ivanov.eu/projects/cs-e9302/cs-e9302_root2.tar.gz
# mkfs.ext3 -L root /dev/sdb1
# mount /dev/sdb1 /mnt
# cd /mnt
# gunzip cs-e9302_root2.tar.gz
# tar -xvf cs-e9302_root2.tar
# umount /mnt
```

Y hecho esto el sistema de archivos ha sido copiado y es bastante funcional, basta conectar el Flash USB al host USB 1 en la CS-E9302 antes de ser encendida, y configurar RedBoot para que lo cargue.

LABORATORIO 6

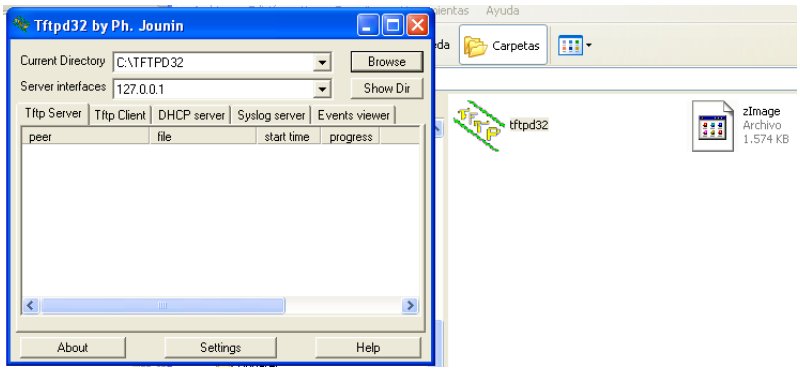
Cargando Linux y root fs en la CS-E9302 mediante RedBoot

Después de compilar por completo el kernel, la imagen del kernel Linux (zImage) se debió crear en el directorio /linux-2.6.24-rc8/arch/arm/boot. RedBoot soporta el protocolo TFTP, y se puede cargar la imagen del kernel a la CS-E9302 via red. En este caso se necesita un servidor TFTP, se usará el TFTPD32 server para WinXP. El TFTPD32 server no necesita instalación y está disponible en http://www.esnips.com/doc/e8c33ffb-e341-402c-be53-ed259daf6ac3/Tools. Se crea un Nuevo directorio bajo el drive C:\ por ejemplo:

```
C:\TFTPD32\
```

Simplemente se copia tftpd32.exe en esta carpeta y se corre:

4. Elegir la pestaña Tftp Server
5. Click en el botón Browse, y referenciarlo a C:\TFTPD32
6. Copiar el archivo zImage a C:\TFTPD32



"Server interfaces" indica la dirección IP del servidor (la dirección IP de WinXP), la cual es 192.168.0.94. Ahora el servidor TFTPD32 está listo para cargar la imagen del kernel Linux. En el

prompt del Hyper Terminal, se setearán las variables de entorno, presionando Ctrl-C, 5 segundos antes de que inicie RedBoot:

```
RedBoot> fconfig
Run script at boot: true
Boot script:
.. fis load zImage
.. exec -c "console=ttyAM root=/dev/mmcblk0p1 rootdelay=5"
Enter script, terminate with empty line
>> fis load zImage
>> exec -c "console=ttyAM root=/dev/sda1 rootdelay=5"
>>
Boot script timeout (1000ms resolution): 5
Use BOOTP for network configuration: false
Gateway IP address: 192.168.0.1
Local IP address: 192.168.0.93
Local IP address mask: 255.255.255.0
Default server IP address: 192.168.0.94
DNS server IP address: 192.168.0.1
Set eth0 network hardware address [MAC]: true
eth0 network hardware address [MAC]: 0x00:0x00:0x00:0x00:0x48:0x33
GDB connection port: 9000
Force console for special debug messages: false
Network debug at boot time: false
Update RedBoot non-volatile configuration - continue (y/n)?y
```

Checamos las direcciones de memoria en flash y lo que existe en ellas:

```
RedBoot> fis list
```

Name	**FLASH addr**	**Mem addr**	**Length**	**Entry point**
Name	0x60000000	0x60000000	0x00040000	0x00000000
RedBoot	0x60FC0000	0x60FC0000	0x00001000	0x00000000
RedBoot config	0x60FE0000	0x60FE0000	0x00020000	0x00000000
FIS directory	0x60040000	0x00A00000	0x00A00000	0x00A00000

```
ramdisk          0x60A40000 0x00080000 0x00300000 0x00080000
```

La imagen pre-cargada será guardada en la dirección 0x60A40000, cuando se inicie el kernel, RedBoot copiará la zImage desde la memoria Flash de esa dirección a la dirección de memoria principal 0x00080000, ahora debemos cargar la imagen desde el servidor Tftp, y en caso de existir algún kernel en dicha dirección, primero lo borramos, luego ejecutamos cada uno de los siguientes pasos:

```
RedBoot> fis delete zImage
```

```
RedBoot> load -r -v -b 0x80000 zImage
RedBoot> fis create -b 0x80000 -l 0x2F0000 zImage
```

Dónde 0x2F0000 es el valor de la huella o tamaño del kernel que varía de 1.5 a aproximadamente 3 MB dependiendo de las funciones soportadas por nuestro kernel.

Hecho esto la tarjeta quedó lista para correr un sistema GNU/Linux completo basado en Debian, llamado ahora **C@M@LEON Embedded GNU/Linux**, con la mayoría de las funciones necesarias para un PC incluido un cliente DHCP, dentro de un sistema embebido. Ahora cada vez que se encienda la tarjeta, automáticamente correrá Linux y cargará el sistema de archivos sin necesidad de ingresar comandos adicionales

LABORATORIO 7

Remasterizando C@M@LEON Embedded GNU/Linux

Cuando C@M@LEON bootea satisfactoriamente, durante las primeras sesiones y cada cierto tiempo, se debe actualizar los paquetes, por tanto al igual que otras distribuciones basadas en Debian, podemos usar la mayoría de sus comandos, repositorios, aplicaciones y paquetería .deb además de las típicas funcionalidades de cualquier sistema basado en Linux.

Cabe recalcar que debido a las prestaciones se ha elegido usar para todas las aplicaciones el editor **vim**. Simplemente se corre el comando apt-get en el prompt de la consola. Hay que asegurarse de que la conexión a internet de alta velocidad funciona correctamente:

```
cs-e9302:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:DE:AD:B0:05:00
          inet addr:192.168.0.93  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: 2001:a001:14c4:0:2de:adff:feb0:500/64 Scope:Global
          inet6 addr: fe80::2de:adff:feb0:500/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:86 errors:0 dropped:86 overruns:0 frame:0
          TX packets:80 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Interrupt:39

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:560 (560.0 b)  TX bytes:560 (560.0 b)
```

En caso de ser necesario configurar la interfaz manualmente:

```
cs-e9302:~# ifconfig eth0 inet 192.168.0.93
```

Y por si necesitamos en algún momento bajar la interfaz:

```
cs-e9302:~# ifconfig eth0 down
```

Ahora checamos la conexión al PC:

```
cs-e9302:~# ping -c 2 192.168.24.10

PING 192.168.24.10 (192.168.24.10) 56(84) bytes of data.
64 bytes from 192.168.24.10: icmp_seq=1 ttl=128 time=2.33 ms
64 bytes from 192.168.24.10: icmp_seq=2 ttl=128 time=0.954 ms

--- 192.168.24.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.954/1.645/2.336/0.691 ms
```

Checamos la conexión a internet:

```
cs-e9302:~# ping -c 2 www.google.com

2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.954/1.645/2.336/0.691 ms
cs-e9302:~# ping -c 2 www.google.com
PING www.l.google.com (66.249.89.147) 56(84) bytes of data.
64 bytes from jp-in-f147.google.com (66.249.89.147): icmp_seq=1 ttl=234 time=17.
9 ms
64 bytes from jp-in-f147.google.com (66.249.89.147): icmp_seq=2 ttl=234 time=17.
8 ms

--- www.l.google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 17.816/17.897/17.979/0.156 ms
```

Una vez lista la configuración y revisada la conectividad, podemos instalar y actualizar aplicaciones. Actualizamos la lista de paquetes:

```
cs-e9302:~# apt-get update
```

Instalamos los paquetes correspondientes a la última versión estable de Debian y presentes en C@M@LEON Embedded GNU/Linux:

```
cs-e9302:~# apt-get upgrade
```

Instalamos make

```
cs-e9302:~# apt-get install make
```

Instalamos gmake

```
cs-e9302:~# apt-get install gmake
```

Instalamos el compilador GNU C

```
cs-e9302:~# apt-get install gcc
```

Instalamos el compilador GNU C++

```
cs-e9302:~# apt-get install g++
```

Instalamos las librerías C estándar

```
cs-e9302:~# apt-get install libc6-dev
```

Instalamos las librerías gráficas X

```
cs-e9302:~# apt-get install libx11-dev
```

Instalamos el servidor Samba

```
cs-e9302:~# apt-get install smbfs
```

Instalamos el demonio hotplugger udev

```
cs-e9302:~# apt-get install udev
```

LABORATORIO 8

Compilando Programas C

Aunque es un sistema con recursos no tan provistos como los de un PC, este sistema embebido es capaz de correr sus propias aplicaciones sin necesidad de compilación cruzada, como se verá más adelante en el desarrollo de las aplicaciones. Ahora se escribirá un simple programa en C, y lo ejecutaremos:

```
cs-e9302:~# vim hola.c

#include <stdio.h>

int main (int agc, char* argc[])
{
  printf ("Hola!!  \n");
  printf ("Tu comando es: %s\n", argc[0]);
  return 0;
}

cs-e9302:~# gcc -o hola hola.c
cs-e9302:~# ./hola

Hola!!
Tu comando es:./hola

cs-e9302:~/temp#
```

LABORATORIO 9

X-Window System

Usaremos la CS-E9302 como un cliente de aplicaciones, el cual envía el entorno gráfico a los dispositivos de pantalla (el Servidor X) vía conexión LAN. El escritorio del PC Windows XP correrá como un Servidor X, este recibe las informaciones de pantalla como mapas de bits desde el cliente, da soporte al mouse, eventos de presionado de teclas y otros funciones, y las envía al cliente. Para el servidor X usaremos una aplicación muy poderosa, completa y fácil de instalar así como de usar, llamada X-Win32 de la empresa StarNet el único inconveniente que se tiene con ella es que no es open-source, sólo se puede obtener una licencia de evaluación de 30 días. En el sistema embebido instalamos la aplicación **xterm** para lanzar un terminal con entorno gráfico, simple pero bastante funcional:

```
cs-e9302:~# apt-get install xterm
```

Corremos el servidor X-Win32, y exportamos la variable de pantalla desde la consola de C@M@LEON, con la dirección IP para que pueda ser vista en WinXP vía red:

```
cs-e9302:~# export DISPLAY=192.168.0.94:0.0
```

En los sistemas basados en Debian, la opción de lanzar aplicaciones gráficas desde root está deshabilitada por defecto, por motivos de seguridad, por tanto para correr cualquier aplicación gráfica debemos crear un usuario regular y cambiarnos al mismo:

```
cs-e9302:~# adduser terminal
cs-e9302:~# su terminal
cs-e9302:~terminal$
```

Y ahora lanzamos un terminal gráfico a la dirección de red inicializado la aplicación xterm:

```
cs-e9302:~terminal$ xterm -bg black -fg white +cm +dc -geometry 80x20+100+50 &
```

Ver los resultados en el PC WinXP

LABORATORIO 10

Compilado y Ejecución de una Aplicación X

Para esta aplicación se uso la consola en modo gráfico que generó la aplicación anterior y todo se realizó desde el mismo terminal gráfico en el PC WinXP. Escribiremos una simple aplicación X "hello", que se compilará y ejecutará para probar la librería xlib11, crearemos una ventana, imprimiremos la cadena "Hello, X Window System!" y la mostraremos sobre la pantalla del servidor X en el PC. La ventana se cerrará cuando se presione un botón del mouse o una tecla.

```
cs-e9302:~# vim hello.c

/**** hello.c demo ******/

/*  Declaración Xlib11 */
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>

/*  Declaración librerías C estándar  */
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

/*  Variables Globales  */
Display *     display;
int           screen_num;
static char * appname;

/*   main() function   */
int main( int argc, char * argv[] ) {

    /*  Window variables  */
    Window       win;
    int          x, y;
    unsigned int width, height;
```

```
    unsigned int border_width;
    char *       window_name = "CS-E9302 DEMO!";
    char *       icon_name   = "Hello";

    /*  Display variables  */
    char *       display_name = NULL;
    unsigned int display_width, display_height;

    /*  Miscellaneous X variables  */
    XSizeHints *  size_hints;
    XWMHints   *  wm_hints;
    XClassHint *  class_hints;
    XTextProperty windowName, iconName;
    XEvent        report;
    XFontStruct * font_info;
    XGCValues     values;
    GC            gc;
    appname = argv[0];

    /*  dar lugar en memoria a nuestras estructuras  */
    if ( !( size_hints  = XAllocSizeHints() ) ||
           !( wm_hints    = XAllocWMHints()   ) ||
           !( class_hints = XAllocClassHint() )    ) {
        fprintf(stderr, "%s: couldn't allocate memory.\n", appname);
        exit(EXIT_FAILURE);
    }

    /*  Conectar al X server  */
    if ( (display = XOpenDisplay(display_name)) == NULL ) {
        fprintf(stderr, "%s: couldn't connect to X server %s\n", appname,
display_name);
        exit(EXIT_FAILURE);
    }

    /*  Configurar la pantalla a mostrar */
    screen_num     = DefaultScreen(display);
    display_width  = DisplayWidth(display, screen_num);
    display_height = DisplayHeight(display, screen_num);

    x = y = 0;
    width  = display_width / 6;
    height = display_width / 7;
```

```
    win = XCreateSimpleWindow(
            display,
            RootWindow(display, screen_num),
                  x, y, width, height, border_width,
                  WhitePixel(display, screen_num),
                  BlackPixel(display, screen_num)
              );

    if ( XStringListToTextProperty(&window_name, 1, &windowName) == 0 ) {
        fprintf(stderr, "%s: structure allocation for windowName failed.\n",
appname);
        exit(EXIT_FAILURE);
    }

    if ( XStringListToTextProperty(&icon_name, 1, &iconName) == 0 ) {
        fprintf(stderr, "%s: structure allocation for iconName failed.\n",
appname);
        exit(EXIT_FAILURE);
    }

    size_hints->flags       = PPosition | PSize | PMinSize;
    size_hints->min_width   = 100;
    size_hints->min_height  = 50;

    wm_hints->flags         = StateHint | InputHint;
    wm_hints->initial_state = NormalState;
    wm_hints->input         = True;

    class_hints->res_name   = appname;
    class_hints->res_class  = "hellox";

    XSetWMProperties(display, win, &windowName, &iconName, argv, argc,
             size_hints, wm_hints, class_hints);


    XSelectInput(display,
                 win,
                 ExposureMask    |
                 KeyPressMask    |
                 ButtonPressMask |
                 StructureNotifyMask);
```

```
    if ( (font_info = XLoadQueryFont(display, "9x15")) == NULL ) {
        fprintf(stderr, "%s: cannot open 9x15 font.\n", appname);
        exit(EXIT_FAILURE);
    }

    gc = XCreateGC(display, win, 0, &values);
    XSetFont(display, gc, font_info->fid);
    XSetForeground(display, gc, WhitePixel(display, screen_num));

    /*  Mostrar Ventana  */
    XMapWindow(display, win);

    while ( 1 ) {
        static char * message = "Hello, X Window System!";
        static int    length;
        static int    font_height;
        static int    msg_x, msg_y;

        XNextEvent(display, &report);

        switch ( report.type ) {
          case Expose:
            if ( report.xexpose.count != 0 ){
                break;
              }

            length = XTextWidth(font_info, message, strlen(message));
            msg_x  = (width - length) / 2;

            font_height = font_info->ascent + font_info->descent;
            msg_y  = (height + font_height) / 2;

            XDrawString(display, win, gc, msg_x, msg_y, message,
strlen(message));
          break;

          case ConfigureNotify:
            width  = report.xconfigure.width;
            height = report.xconfigure.height;
          break;
```

```
          case ButtonPress:
          case KeyPress:
            /*  Clean up and exit  */
            XUnloadFont(display, font_info->fid);
            XFreeGC(display, gc);
            XCloseDisplay(display);
            exit(EXIT_SUCCESS);

        }
   }
    return EXIT_SUCCESS;
}
```

Compilamos y ejecutamos:

```
cs-e9302:~# gcc -o hello hello.c -lX11
cs-e9302:~# ./hello
```

LABORATORIO 11

Instalando y Lanzando Aplicaciones Gráficas

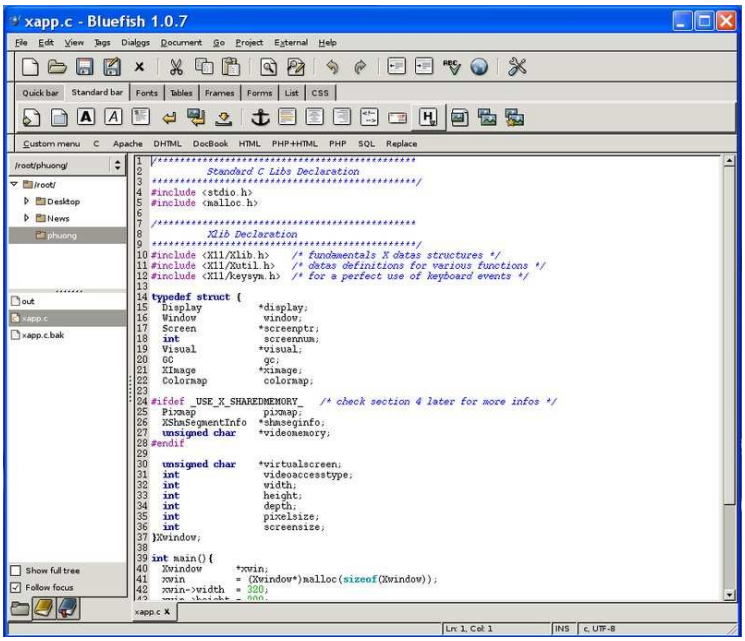
Instalaremos aplicaciones gráficas más complejas que podrían ser: Firefox, juegos, escritorios simples, etc, la imaginación es el límite siempre y cuando respetemos las limitaciones de la CS-E9302. Para nuestro caso instalaremos el editor de programación **bluefish**:

```
cs-e9302:~# apt-get install bluefish
```

Luego con el terminal gráfico con shell root aún abierto, ejecutamos el editor mediante:

```
cs-e9302:~# bluefish
```

Lo cual nos dará el siguiente resultado:


xapp.c - Bluefish 1.0.7
File Edit View Tags Dialogs Document Go Project External Help
Quick bar Standard bar Fonts Tables Frames Forms List CSS
Custom menu C Apache DHTML DocBook HTML PHP+HTML PHP SQL Replace
/root/phuong/
/root/
Desktop
News
phuong
out
xapp.c
xapp.c.bak
Show full tree
Follow focus
/***
Standard C Libs Declaration
***/
#include <stdio.h>
#include <malloc.h>
/***
Xlib Declaration
***/
#include <X11/Xlib.h> /* fundamentals X datas structures */
#include <X11/Xutil.h> /* datas definitions for various functions */
#include <X11/keysym.h> /* for a perfect use of keyboard events */
typedef struct {
Display *display;
Window window;
Screen *screenptr;
int screennum;
Visual *visual;
GC gc;
XImage *ximage;
Colormap colormap;
#ifdef _USE_X_SHAREDMEMORY_ /* check section 4 later for more infos */
Pixmap pixmap;
XShmSegmentInfo *shmseginfo;
unsigned char *videomemory;
#endif
unsigned char *virtualscreen;
int videoaccesstype;
int width;
int height;
int depth;
int pixelsize;
int screensize;
}XWindow;
int main(){
Xwindow *xvin;
xvin = (XWindow*)malloc(sizeof(XWindow));
xvin->width = 320;
xapp.c
Ln: 1, Col: 1
INS
c, UTF-8

LABORATORIO 12

APACHE WEB SERVER

La aplicación de software libre por excelencia para montar servidores web y una de las favoritas en el mundo es el servidor web Apache (www.apache.org), fácil de instalar y configurar, además de ser liviana y robusta, pudiendo ser dotada de muchos módulos de propósitos varios, orientados desde la seguridad hasta el soporte para nuevas características del desarrollo web. Lo instalamos mediante:

```
cs-e9302:~# apt-get install apache2
```

Al ser este un demonio, cada vez que inicie el sistema, este se cargará automáticamente verificando y cargando según la disponibilidad, todo aquello que conste en su archivo de configuración. Esta tarea no será detallada debido a que como se dijo este sistema está diseñado para el aprendizaje, y uno de los deberes de un estudiante es investigar, por tanto sólo se relatará lo esencial:

```
cs-e9302:~# vim /etc/apache/apache2.conf
```

Aquí se comentará o descomentará las líneas según sea el propósito del servidor o la funcionalidad que se desee, como los usuarios, uso de https, dirección de DNS, carpetas de destino, etc. La página que se muestra por defecto se encuentra en /usr/lib/apache2-default/, está deberá ser cambiada por la que se desee mostrar y se guardará con el nombre de index.html. Para iniciar, parar o reiniciar el demonio hacemos lo siguiente:

```
cs-e9302:~# /etc/init.d/apache2 start
cs-e9302:~# /etc/init.d/apache2 stop
cs-e9302:~# /etc/init.d/apache2 restart
```

LABORATORIO 13

Creando un MP3 Server con Perl

Debemos asegurarnos de que tenemos el lenguaje Perl en el sistema, caso contrario lo instalamos:

```
cs-e9302:~# apt-get install perl
```

Ahora creamos una carpeta de trabajo llamada MP3 que contendrá las canciones, la lista de reproducción y el código Perl del servidor.

```
cs-e9302:~# mkdir /MP3/
```

Primero debemos almacenar las canciones en la carpeta MP3, y luego creamos el archivo playlist.m3u:

```
cs-e9302:~# cd /MP3/
cs-e9302:/MP3# vim playlist.m3u
```

Y dentro de él pondremos lo siguiente:

```
#EXTM3U

 #EXTINF:123,Título de Muestra
/MP3/Muestra.mp3

 #EXTINF:321,Título de Ejemplo
/MP3/Ejemplo.ogg
```

Ahora creamos y editamos el archivo con extensión .pl, MP3Server.pl, agregando el código final:

```
cs-e9302:/MP3# vim MP3Server.pl
```

```
#!/usr/bin/perl -w
```

```
use strict;
use IO::Socket;
#tomar el puerto a controlar o por default 8000
my $port = $ARGV[0] || 8000;
#ignorar procesos hijos para evitar zombies
$SIG{CHLD} = 'IGNORE';
#crear el socket a escuchar
my $listen_socket = IO::Socket::INET->new(LocalPort => $port,
Listen => 10,
Proto => 'tcp',
Reuse => 1);
#asegurarnos que estamos controlando el puerto
die "No se puede crear el listening socket: $@" unless $listen_socket;
warn "Servidosr listo!!!. Esperando conexiones ... \n";
#esperar conexiones
while (my $connection = $listen_socket->accept){
my $child;
# crear el fork para salir
die "No se puede hacer fork: $!" unless defined ($child = fork());
#¡el hijo!
if ($child == 0){
#cerrar el puerto para escuchar el proceso hijo
$listen_socket->close;
#llamar la funcion principal del hijo
play_songs($connection);
#si el hijo regresa salte
exit 0;
}
#¡soy el padre!
else{
#¿quién se conectó?
warn "Conexión recibida ... ",$connection->peerhost,"\n";
#cerrar la conexión, ya fue mandada a un hijo
$connection->close();
}
#regresa a escuchar otras conexiones
}
sub play_songs{
my $socket = shift;
#sacar todas las canciones posibles
open PLAYLIST, "playlist.m3u" or die;
my @songs = <PLAYLIST>;
```

```
close PLAYLIST;
chomp @songs;
#creador de canciones randómico
srand(time / $$);
#crear un loop eterno hasta que el cliente deje de escuhar
while(){
#Mandar el header necesario
print $socket "HTTP/1.0 200 OK\n";
print $socket "Content-Type: audio/x-mp3stream\n";
print $socket "Cache-Control: no-cache \n";
print $socket "Pragma: no-cache \n";
print $socket "Connection: close \n";
print $socket "x-audiocast-name: My MP3 Server\n\n";
#seleccionar una canción aleatoria de la lista
my $song = $songs[ rand @songs ];
#que cancion estamos tocando
warn( "play song: $song\n");
#abrir la cancion o intentar con otra
open (SONG, $song) || next;
binmode(SONG); #para usarios de windows
my $read_status = 1;
my $print_status = 1;
my $chunk;
#Esta parte imprime el binario al socket
#Lo hace lo más rápido posible
while( $read_status && $print_status ){
$read_status = read (SONG, $chunk, 1024);
if( defined $chunk && defined $read_status){
$print_status = print $socket $chunk;
}
undef $chunk;
}
close SONG;
unless( defined $print_status ){
$socket->close();
exit(0);
}
}
}
```

Ahora corremos el servidor mediante:

```
cs-e9302:/MP3# perl MP3Server.pl
```

Para conectar un cliente desde Winamp todo lo que hay que hacer es presionar Ctrl-L y poner:

```
http://192.168.0.94:8000
```

LABORATORIO 14

Poniendo un logo de Debian al iniciar el Sistema

Para los amantes de Debían, ya que soy ubuntero por naturaleza (casi debianizado :-p), aquí les traigo algo muy particular, simplemente voy a explicar en sencillos pasos para su instalación y configuración, para luego mostrarle la imagen, así como dice el dicho "una imagen vale mas que mil palabras.... "

Instalación:

* Como usuario administrado: apt-get install linuxlogo

* Copiamos y Renombramos a issue: cp /etc/issue.linuxlogo /etc/issue

* Listo, para mostrar el logo: cat /etc/issue

* ahora: cp /etc/mod /etc/mod.tail

LABORATORIO 15

Recopilación de datos y métodos de control en un sistema GNU/Linux

Las máquinas proveen mucha información acerca del hardware y del software instalados. He aquí un compendio de comandos y rutas para encontrar esta información.

El sistema nos entrega mucha información durante el arranque:

```
sebb@nix ~ dmesg | less
```

Con:

```
sebb@nix ~ dmesg | head -1
Linux version 2.6.18-4-k7 (Debian 2.6.18.dfsg.1-12etch2)
(dannf@debian.org)
(gcc version 4.1.2 20061115 (prerelease) (Debian 4.1.1-21))
~1 SMP Wed May 9 23:42:01 UTC 2007
```

Obtenemos los detalles del sistema operativo, kernel, gcc, etc.

El comando hostname enseña o fija el nombre de una máquina.

Con uname, tenemos un resumen, datos acerca del último inicio y la carga del procesador:

```
sebb@nix ~ uname -a
Linux debian 2.6.18-4-k7 #1 SMP Wed May 9 23:42:01 UTC 2007 i686
GNU/Linux
```

El fichero /etc/issue indica el nombre y versión del sistema operativo:

```
sebb@nix ~ cat /etc/issue
C@m@leon Embedded GNU/Linux 4.0 \n \l
```

La estructura de directorios debajo de /proc nos entrega datos sobre el hardware:

```
sebb@nix ~ cat /proc/version
```

```
Linux version 2.6.18-4-k7 (Debian 2.6.18.dfsg.1-12etch2)
(dannf@debian.org)
(gcc version 4.1.2 20061115 (prerelease) (Debian 4.1.1-21))
~1 SMP Wed May 9 23:42:01 UTC 2007
```

La herramienta lsb_release da más detalles:

```
sebb@nix ~ lsb_release -a
LSB Version:
core-2.0-noarch:core-3.0-noarch:core-3.1-noarch:core-2.0-ia32:core-3.0-
ia32:core-3.1-ia32
Distributor ID: Debian
Description:    Debian GNU/Linux 4.0r0 (etch)
Release:        4.0r0
Codename:       etch
```

mount presenta todos los discos montados:

```
#       mount
```

ls -l /dev/disk/by-uuid presenta una cadena alfanumérica que precisa la ubicación de los discos:

```
sebb@nix ~ ls -l /dev/disk/by-uuid
total 0
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 5da97b22-8307-4fa7-a29e-
67f3843fe45a -> ../../sda5
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 8654CF6A54CF5C15 ->
../../sda1
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 a12a9d27-f17a-48f7-979e-
7a31d4f934fd -> ../../sda6
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 A83CD1B63CD17FAC ->
../../sda2
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 BC82D3FB82D3B85A ->
../../sda3
```

Esta presentación tiene la ventaja obtener la lista de dispositivos siempre en el mismo orden alfabético.

ls -l /dev/disk/by-path presenta los discos con detalles de ubicación:

```
sebb@nix ~ ls -l /dev/disk/by-path

total 0
lrwxrwxrwx 1 root root  9 2008-12-28 12:44 pci-0000:00:1f.2-scsi-0:0:0:0 -> ../../sda
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 pci-0000:00:1f.2-scsi-0:0:0:0-part1 -> ../../sda1
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 pci-0000:00:1f.2-scsi-0:0:0:0-part2 -> ../../sda2
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 pci-0000:00:1f.2-scsi-0:0:0:0-part3 -> ../../sda3
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 pci-0000:00:1f.2-scsi-0:0:0:0-part4 -> ../../sda4
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 pci-0000:00:1f.2-scsi-0:0:0:0-part5 -> ../../sda5
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 pci-0000:00:1f.2-scsi-0:0:0:0-part6 -> ../../sda6
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 pci-0000:00:1f.2-scsi-1:0:0:0 -> ../../scd0
```

Aún más detalles ls -l /dev/disk/by-id:

```
sebb@nix ~ ls -l /dev/disk/by-id
total 0
lrwxrwxrwx 1 root root  9 2008-12-28 12:44 ata-TOSHIBA_MK2552GSX_88PBC1JIT -> ../../sda
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 ata-TOSHIBA_MK2552GSX_88PBC1JIT-part1 -> ../../sda1
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 ata-TOSHIBA_MK2552GSX_88PBC1JIT-part2 -> ../../sda2
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 ata-TOSHIBA_MK2552GSX_88PBC1JIT-part3 -> ../../sda3
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 ata-TOSHIBA_MK2552GSX_88PBC1JIT-part4 -> ../../sda4
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 ata-TOSHIBA_MK2552GSX_88PBC1JIT-part5 -> ../../sda5
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 ata-TOSHIBA_MK2552GSX_88PBC1JIT-part6 -> ../../sda6
lrwxrwxrwx 1 root root  9 2008-12-28 12:44 scsi-1ATA_TOSHIBA_MK2552GSX_88PBC1JIT -> ../../sda
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 scsi-1ATA_TOSHIBA_MK2552GSX_88PBC1JIT-part1 -> ../../sda1
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 scsi-1ATA_TOSHIBA_MK2552GSX_88PBC1JIT-part2 -> ../../sda2
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 scsi-1ATA_TOSHIBA_MK2552GSX_88PBC1JIT-part3 -> ../../sda3
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 scsi-1ATA_TOSHIBA_MK2552GSX_88PBC1JIT-part4 -> ../../sda4
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 scsi-1ATA_TOSHIBA_MK2552GSX_88PBC1JIT-part5 -> ../../sda5
lrwxrwxrwx 1 root root 10 2008-12-28 12:44 scsi-1ATA_TOSHIBA_MK2552GSX_88PBC1JIT-part6 -> ../../sda6
```

lshw presenta una tabla del hardware:

```
sebb@nix ~ lshw -short
```

El comando lsusb presenta detalles del concentrador USB y de los dispositivos usb conectados.

```
sebb@nix ~ lsusb
```

Con lspci podemos listar los recursos pci de la máquina.

```
sebb@nix ~ lspci
```

En /proc/acpi/ está la configuración de acpi en funcionamiento.

En /proc/cpuinfo se encuentran datos acerca del procesador.

En /proc/meminfo se encuentran datos acerca de la memoria.

Una descripción de las particiones está en /proc/partitions, y se puede usar fdisk -l para más detalles.

El comando df reseña el espacio usado y disponible en todos los sistemas de ficheros montados.

Con ps axu, listamos todos los procesos ejecutándose. Con ps faxu tenemos un arbol de procesos.

top nos informa de los procesos cargados y del uso del sistema.

Con la herramienta fuser podemos obtener datos de conexión por puerto:

```
sebb@nix ~ puerto=80 ; echo -e "Conexiones a puerto $puerto:" ; ps -fea
| \
grep `fuser $puerto/tcp | awk '{ print $1 }'`
Conexiones a puerto 80:
80/tcp:
root      3810     1  0 10:53 ?        00:00:00 /usr/sbin/apache
www-data  3816  3810  0 10:53 ?        00:00:00 /usr/sbin/apache
www-data  3817  3810  0 10:53 ?        00:00:00 /usr/sbin/apache
www-data  3818  3810  0 10:53 ?        00:00:00 /usr/sbin/apache
www-data  3819  3810  0 10:53 ?        00:00:00 /usr/sbin/apache
www-data  3820  3810  0 10:53 ?        00:00:00 /usr/sbin/apache
```

Otro ejemplo, las conexiones a ssh:

```
sebb@nix ~ puerto=21 ; echo -e "Conexiones a puerto $puerto:" ; ps -fea
| \
grep `fuser $puerto/tcp | awk '{ print $1 }'`
```

El comando netstat: Todas las conexiones:

```
sebb@nix ~ netstat -an
```

Más detalles:

```
sebb@nix ~ netstat -en
```

Sobre tcp:

```
sebb@nix ~ netstat -nat
```

Formato largo:

```
sebb@nix ~ netstat -luta
```

Solo ethernet eth0:

```
sebb@nix ~ ifconfig eth0
```

La información acerca de los DNS también puede estar en /etc/resolv.conf

El comando route permite ver o cambiar la tabla de enrutamiento IP.

```
sebb@nix ~ route -v
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.0.0     *               255.255.255.0   U     0      0        0 eth0
link-local      *               255.255.0.0     U     0      0        0 eth0
default         pared.jacla     0.0.0.0         UG    0      0        0 eth0
```

Actividad de red:

```
sebb@nix ~ tcpdump  -vv
```

Usuarios logeados y su actividad:

```
sebb@nix ~ x
```

Máquinas en la red local:

```
sebb@nix ~ nmap -vv -sL -P0 192.168.0.*
```

lsof sirve para ver archivos abiertos, consultar el manual con man lsof:

```
sebb@nix ~ lsof /dev/log
COMMAND  PID USER   FD   TYPE     DEVICE SIZE   NODE NAME
syslogd 2307 root    0u  unix 0xdf05dba0        5814 /dev/log
kdm     3840 root    7u  unix 0xdbf0ca80        9149
/var/run/xdmctl/dmctl-
...
```

LABORATORIO 16

Programación de Drivers y Módulos para dispositivos

¿Qué es un driver?

Un driver es una capa de código entre el dispositivo de hardware y la aplicación. Un driver usa los privilegios con los que se ejecuta su código para definir exactamente como se quiere que un dispositivo sea visto por una aplicación. Pueden existir diferentes drivers para un mismo dispositivo.

¿Por qué escribir un driver?

Existen muchas razones para querer escribir un driver.

- Para dar soporte a nuevo hardware
- Para mantener un producto propio
- Se está creando hardware a un ritmo rápido, y los programadores de drivers van a tener trabajo por un buen tiempo

Uso de módulos

Los módulos tienen la ventaja de permitir adicionar y remover funcionalidades del kernel mientras el sistema está corriendo.

Módulos de ejemplo

Para obtener los módulos de ejemplo, ejecute:

```
$ svn co http://svn.arhuaco.org/svn/src/linux/examples/modules/hello/
$ cd hello
```

Hola Mundo

El siguiente ejemplo hello1.c es el código de un módulo que puede ser cargado y descargado de memoria con insmod y rmmod.

```
#include <linux/init.h>
#include <linux/module.h>

MODULE_LICENSE("Dual BSD/GPL");

static int hello_init(void)
{
        printk(KERN_ALERT "Hola, mundo\n");
        return 0;
}

static void hello_exit(void)
{
        printk(KERN_ALERT "Adios, mundo cruel\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

La macro MODULE_LICENSE especifica la licencia del módulo. Si no se especifica esta cadena, se produce un warning, por ejemplo:

```
hello1: module license 'unspecified' taints kernel.
```

Se recomienda especificar la licencia, a no ser que se esté creando un módulo propietario. Las macros module_init y module_exit son para inicialización. Funcionan si código se compila como módulo y directamente en el kernel. Cuando el código se compila en el kernel module_init se comporta igual a __initcall(). Para compilar el módulo, se usa el siguiente Makefile:

```
obj-m += hello1.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

La compilación de los módulos se ha simplificado, antes era necesario especificar múltiples parámetros en un makefile para un módulo. El nuevo sistema para compilar el kernel facilita el proceso de compilación de módulos. Para compilar un módulo para User-mode Linux, se escribiría solamente.

```
uml@uml:~/hello$ ARCH=um make
make -C /lib/modules/2.6.16.18/build M=/home/uml/hello modules
make[1]: Entering directory `/home/n/uml/linux-2.6.16.18'
  CC [M]  /home/uml/hello/hello1.o
  Building modules, stage 2.
  MODPOST
  CC      /home/uml/hello/hello1.mod.o
  LD [M]  /home/uml/hello/hello1.ko
make[1]: Leaving directory `/home/n/uml/linux-2.6.16.18'
```

Ahora podemos insertar el módulo en el kernel.

```
# insmod hello1.ko
Hola, mundo
```

Al borrar el módulo, se llama la función especificada en la macro module_exit. En este caso, es la función hello_exit.

```
# rmmod hello1
Adios, mundo cruel
```

Módulo simple con un driver de caracter

Los drivers en el kernel son accedidos por medio de un archivo especial. Estos archivos se pueden crear con el comando mknod.

```
mknod [OPTION]... NAME TYPE [MAJOR MINOR]
# mknod prueba char 1 1
# ls -l prueba
crw-r--r-- 1 root root 1, 1 2006-05-31 05:55 prueba
# rm prueba
```

En el archivo Documentation/devices.txt se encuentra un listado de los dispositivos que ya tienen un nombre asignado en Linux. Al hacer un driver existen dos opciones: tomar un número mayor y menor no asignado, o usar uno dinámico. Cuando se usa asignación dinámica, se puede leer los números asignados de `/proc/devices`.

```
$ cat /proc/devices
```

Character devices:

```
  1 mem
  2 pty
  3 ttyp
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 14 sound
 29 fb
116 alsa
128 ptm
136 pts
180 usb
189 usb_device
216 rfcomm
226 drm
```

```
$ ls -l /dev/console
crw------- 1 root root 5, 1 2006-05-31 04:54 /dev/console
$ ls -l /dev/input/mice
crw-rw---- 1 root root 13, 63 2006-05-31 12:52 /dev/input/mice
```

Sintaxis de inicialización "C Tagged Structures"

En el código del kernel es normal encontrar incializaciones de estructuras que no asignan un valor a todos los campos de la estructura. Estas inicializaciones se hacen usando la siguiente sintaxis:

```
#include <stdio.h>

struct prueba
{
 int a;
 int b;
};

int main(int argc, char *argv[])
{

  struct prueba x =
  {
    .a = 1,
  };

  printf("x.a = %d\n", x.a);

  return 0;
}
```

Las inicializaciones de este estilo permiten inicializar los miembros de la estructura sin seguir un orden en particular, y sin que sea necesario inicializar todos los campos. El siguiente es el código de ejemplo de un driver de caracter de lecto-escritura

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <asm/uaccess.h> /* get_user and put_user */
```

```
#define SUCCESS 0
#define DEVICE_FILE_NAME "char_dev"
#define DEVICE_NAME "char_dev"
#define MAJOR_NUM 100
#define BUF_LEN 80

MODULE_LICENSE("Dual BSD/GPL");


static atomic_t Device_Open = ATOMIC_INIT(1);
static char Message[BUF_LEN]; /* Buffer */
static char *Message_Ptr;     /* ¿Qué tan lejos llegó la lectura? */

static int device_open(struct inode *inode, struct file *file)
{

#ifdef DEBUG
  printk(KERN_INFO "device_open(%p)\n", file);
#endif

  if (!atomic_dec_and_test (&Device_Open))
  {
    atomic_inc(&Device_Open);
    return -EBUSY; /* ya está abierto */
  }

  Message_Ptr = Message;
  try_module_get(THIS_MODULE);

  return SUCCESS;
}

static int device_release(struct inode *inode, struct file *file)
{
#ifdef DEBUG
  printk(KERN_INFO "device_release(%p,%p)\n", inode, file);
#endif


  atomic_inc(&Device_Open); /* marca el dispositivo como no-abierto */

  module_put(THIS_MODULE);
  return SUCCESS;
}

static ssize_t device_read(struct file *file,    /* ver include/linux/fs.h   */
                           char __user * buffer, /* buffer a llenar con datos */
```

```
                           size_t length,        /* longitud del buffer  */
                           loff_t * offset)
{
  int bytes_read = 0; /* bytes escritos en el buffer */

#ifdef DEBUG
  printk(KERN_INFO "device_read(%p,%p,%d)\n", file, buffer, length);
#endif

  if (*Message_Ptr == 0) /* retornar EOF */
    return 0;

  while (length && *Message_Ptr)
  {
    put_user(*(Message_Ptr++), buffer++); /* escribe al buffer del usuario */
    length--;
    bytes_read++;
  }

#ifdef DEBUG
  printk(KERN_INFO "Lei %d bytes, quedan %d\n", bytes_read, length);
#endif

  return bytes_read; /* bytes escritos al buffer */
}

static ssize_t
device_write(struct file *file,
             const char __user * buffer, size_t length, loff_t * offset)
{
  int i;

#ifdef DEBUG
  printk(KERN_INFO "device_write(%p,%p,%d)", file, buffer, length);
#endif

  for (i = 0; i < length && i < BUF_LEN; i++)
    get_user(Message[i], buffer + i);

  Message_Ptr = Message;

  return i;
}

struct file_operations Fops = {
  .read = device_read,
  .write = device_write,
```

```
  .open = device_open,
  .release = device_release,
};

int init_module()
{
  int ret_val;

  ret_val = register_chrdev(MAJOR_NUM, DEVICE_NAME, &Fops);

  if (ret_val < 0)
  {
    printk(KERN_ALERT "El registro del dispositivo falló (%d)\n", ret_val);
    return ret_val;
  }

  printk(KERN_INFO "Se registró el dispositivo. El major device number es %d.\n",
MAJOR_NUM);
  printk(KERN_INFO "mknod %s c %d 0\n", DEVICE_FILE_NAME, MAJOR_NUM);

return 0;
}

void cleanup_module()
{
  int ret;

  ret = unregister_chrdev(MAJOR_NUM, DEVICE_NAME);

  if (ret < 0)
    printk(KERN_ALERT "Error: unregister_chrdev: %d\n", ret);
}
# insmod chardev.ko

# dmesg | tail
```

Se registró el dispositivo. El major device number es 100.

```
mknod char_dev c 100 0

# mknod /dev/char_dev c 100 0

# ls -lh /dev/char_dev
crw-r--r-- 1 root root 100, 0 2006-05-31 07:21 /dev/char_dev

# chmod a+w /dev/char_dev
```

Usando python podemos hacer pruebas. La idea es ver como van saliendo las líneas de depuración a medida que se hacen las llamadas.

```
$ python

>>> f = open("/dev/char_dev", "r")
>>> f.close()
>>> f = open("/dev/char_dev", "w")
>>> f.write("Hola\n")
>>> f.close()
>>> f = open("/dev/char_dev", "r")
>>> print f.readline()
Hola

>>> f.close()
```

Módulo para un driver de caracter

A diferencia del módulo simple, este módulo:

Hecho:

- Usa un major number dinámico.
- Recibe parámetros cuando es insertado, y en un parámetro se puede especificar el major number.
- Usa memoria dinámica.
- Funciona llseek
- Soporta llamadas ioctl.
- Tiene un programa en el espacio del usuario para su configuración (ioctl).

Falta por hacer:

- Tiene una entrada en el proc filesystem con estadística

Módulo para un driver de con lectura bloqueante y no bloqueante

El módulo de ejemplo anterior siempre retorna algo al leer. Ahora trabajaremos haciendo que las funciones de lectura y escritura implementen la siguiente semántica estándar:

- Si un proceso llama a read pero no hay datos disponibles El proceso debe bloquearse. El proceso es despertado tan pronto llegue un dato, y los datos se retornan al proceso que hizo la llamada, incluso si hay menos datos que los que se solicitaron en la llamada.
- Si un proceso llama a write y no hay espacio en el buffer El proceso debe bloquearse, y debe quedar en una cola de espera diferente a la que se usa para las lecturas. Cuando algunos datos ya se han escrito por el hardware de salida y hay espacio en el buffer de salida, el proceso es despertado y la llamada a write tiene éxito, aunque la escritura puede ser parcial.

En este driver ya no aparece definida la llamada seek. Para que todas las llamadas a seek fallen, se ha llamado la función nonseekable_open en el método open, y se ha referenciado no_llseek (disponible en linux/fs.h) en la estructura file_operations del driver.

Algunas reglas para tener en cuenta:

- Nunca dormir si no se está seguro de que otro proceso nos despertará
- No dormir adquiriendo semáforos que impidan que nos despierten luego

El código está en este repositorio. Tiene muchos comentarios. Si lee alguna parte y cree que falta un comentario, con gusto lo adiciono.

Módulo para capturar una interrupción

Este módulo está en desarrollo

Comenzamos con el acceso a hardware. En muchas arquitecturas, el espacio de direcciones y el de los puertos de I/O es el mismo. No obstante, debido a que algunas arquitecturas implementan espacios de direcciones separados, Linux tiene funciones especiales para leer y escribir en puertos, incluso en las arquitecturas que no los tienen.

En la arquitectura ARM el espacio de memoria e I/O es el mismo.

Al programar es importante hacer que el compilador no optimice ciertas instrucciones, y para ello Linux provee las siguientes macros que garantizan que las lecturas o escrituras que están antes de la función se completen.

- void barrier(void); (en /linux/kernel.h)
- void rmb(void); (en /asm/system.h) – read memory barrier
- void wmb(void); (en /asm/system.h) – write memory barrier
- void mb(void); (en /asm/system.h) – memory barrier (más lenta que rmb y wmb).

También se recomienda usar variables con la palabra clave volátil donde haga falta para prevenir optimizaciones del compilador.
Para solicitar el acceso exclusivo a una región de I/O, se usan las funciones request_region y release_region.
Ver el módulo en http://svn.arhuaco.org/svn/src/linux/examples/modules/parallel/.

Para escribir y leer de estos puertos, Linux define en /asm/io.h las siguientes funciones:

- unsigned inb(unsigned port) : 8 bits
- unsigned inw(unsigned port) : 16 bits
- unsigned inl(unsigned port) : 32 bits
- unsigned outb(unsigned char byte, unsigned port)
- unsigned outw(unsigned short word, unsigned port)
- unsigned outl(unsigned long word, unsigned port)

En algunas arquitecturas (S390) sólo se permiten operaciones de I/O de 8 bits.
Para un ejemplo de acceso a estos puertos en espacio de usuario, ver count.c.
Note que las funciones que usan words y longs pueden cambiar el orden de los bytes dependiendo de la plataforma.

También se pueden leer strings de puertos, y para ello están las siguientes funciones:

- unsigned insb(unsigned port, void *addr, unsigned long count)
- unsigned outsb(unsigned port, void *addr, unsigned long count)

También hay funciones para leer strings de longs y de words, y estas no cambian el orden de los bytes.

Entradas en el /proc
Ver:

- /proc/interrupts
- /proc/ioports

Restricciones de un manejador de interrupción:

- No puede transferir datos del usuario

- No se puede dormir
- No puede obtener un semáforo
- Si aparta memoria, debe hacerlo con GPF_ATOMIC
- No pueden llamar a schedule

Work queues

En interrupciones, hay que retornar lo más rápido posible. En el kernel 2.6 se ha implementado una nueva interfaz llamada work queue.

Ver:

http://www.linuxjournal.com/article/6916

Y un ejemplo en:

/arch/um/drivers/port_kern.c y en /arch/um/drivers/net_kern.c.

LABORATORIO 17

Direct FB

Se trata de un toolkit gráfico, que nos permitirá lanzar aplicaciones gráficas bastante vistosas sin cargar demasiado al sistema.

Compilando las librerías requeridas

Bajar y compilar zlib 1.2.3 (http://www.zlib.net/):

```
./configure
make
make install
```

Bajar y compilar libpng 1.2.20 (http://www.libpng.org):

```
./configure
make
make install
```

Bajar y compilar libjpeg 6b (http://www.ijg.org/):

```
./configure
make
make installlib
```

Bajar y compilar freetype 2.3.5 (http://freetype.org):

```
./configure
make
make install
```

Compilando DirectFB

Bajar DirectFB 1.0.1 de http://directfb.org. extraer,las fuentes y cometar la línea 1570 en systems/fbdev/fbdev.c:

```
//if (dfb_fbdev_compatible_format( var, 0, 5, 6, 5, 0, 11, 5, 0 ))
return DSPF_RGB16;
```

Luego:

```
./configure disablex11
withgfxdrivers=
none \
withinputdrivers=
keyboard,linuxinput,ps2mouse
make
make install
```

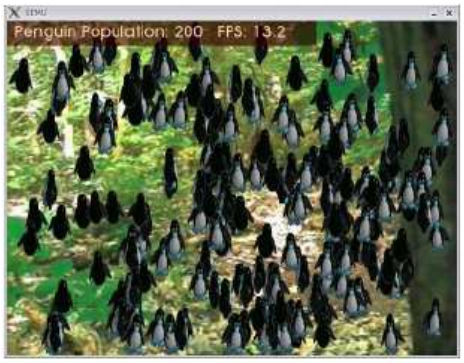
Luego necesitamos configurar la siguiente variable:

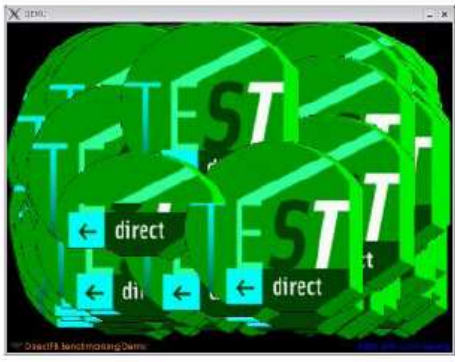
```
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
```

Compilando los ejemplos DirectFB

Bajar DirectFB-examples 1.0.0 (o posteriores) desde http://directfb.org. Configurar y compilar:

```
./configure
make
make install
```

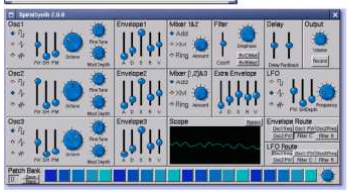

Penguin Population: 200 FPS: 13.2


direct
direct

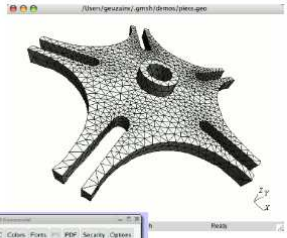
PRÁCTICAS DESATENDIDAS PROPUESTAS

- Montar un Servidor Samba
- Instalar FLTK y Aplicaciones: flPhoto, Gmsh, Spiral Synth, HTMLDOC

flPhoto



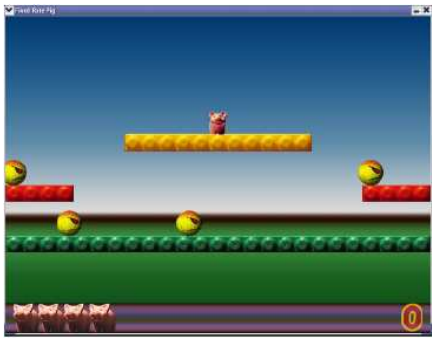
Spiral Synth



Gmsh

HTMLDOC

- Instalar SDL y probar el juego Pig, e instalar MPEG Menu System Version 2



- Instalar y configurar el navegador Dillo
- Lanzar Video streaming en un PC
- Montar un IDS
- Realizar una auditoría de Seguridad a un sistema instalando herramientas varias, como sniffers, escáner de puertos, IDSs, etc.
- Instalar y configurar una JAVA Virtual Machine
- Usar las GPIO's de la CS-E9302 y crear algún tipo de diseño hardware controlado mediante código C.

La Imaginación es el Limite…!!!