Diseño de una plataforma móvil autónoma con procesamiento heterogéneo embebido

Carlos A. García

21 de mayo de 2002

Índice general

A todos los que
que me han provocado intelectualmente...

Agradecimientos

. . . a mis padres, porque la educación siempre fuese importante.

. . . a mis hermanos, por enseñarme el placer de la lectura.

. . . a Ricardo por Lem y el café, por conversaciones siempre interesantes y por toda su ayuda y paciencia.

Introducción

La visión de un robot moviéndose ya no produce admiración. La expresión "Inteligencia Artificial"evoca robots casi humanos: Hal 9000, R2D2 & C3PO... La opinión generalizada es que los robots actuales son capaces de realizar tareas casi humanas.¿Que es lo que hay de cierto?¿Cuales son las capacidades actuales de un robot *"state of the art"*?

El término "Strong AI"(Inteligencia Artificial Fuerte), fue acuñado por John Searle en su libro "Minds, Brains, and Programs "(1980):

". . . according to strong AI, the computer is not merely a tool in the study of the mind; rather, the appropriately programmed computer really is a mind, in the sense that computers given the right programs can be literally said to understand and have other cognitive states. In strong AI, because the programmed computer has cognitive states, the programs are not mere tools that enable us to test psychological explanations; rather, the programs are themselves the explanations."

Mi opinión particular es que no hay ningún impedimento metafísico que prohíba la existencia de tal inteligencia en un máquina algún día mas o menos cercano. La consciencia y la inteligencia aparecieron el algún tiempo lejano de la evolución natural, y de igual manera aparecerán en los cerebros mecánicos, creados por el hombre.

Cada año, cada mes, cada día, avanzamos un paso mas en este camino que nos llevará a ser dioses, a crear vida artificial. Algún día no estaremos solos nunca mas.

Objetivo del proyecto

Este proyecto se engloba dentro de uno mucho mayor, cuyo objetivo final es una mejora del conocimiento sobre la naturaleza de la autonomía y una mejora de los perfiles operacionales empleados en la especificación de sistemas de control.

Para ello se exige una arquitectura flexible para sistemas de alta autonomía.

Este proyecto se planteó como la creación de una plataforma adecuada para avanzar en arquitecturas modulares. Es una base para poder investigar alternativas al control tradicional de robots, que permitan sobre todo una mayor autonomía. ¿Que pasa cuando un robot se enfrenta a algo que sus creadores no tuvieron en cuenta? ¿Debe dejar su labor incompleta?¿O puede (podría) razonar acerca del suceso y encontrar un nuevo camino, no previsto, para seguir avanzando en su tarea?

Eso es lo que queremos conseguir, un robot, una planta química, que sepan reaccionar ante lo imprevisto ... como los seres humanos hacen continuamente.

Parte I

Fundamentos

Capítulo 1

Conocimiento del mundo externo

¿Como puede un robot moverse por un entorno altamente cambiante . . . ?

- Contacto
- Infrarrojos
- Ultrasonidos
- Vision artificial
- Laser

1.1. Contacto

Estos son los sensores mas sencillos que puede utilizar un robot. Son sencillos interruptores que detectan el contacto con otro objeto. Cuando chocan con un objeto se cierra un circuito (o, en algunos casos, se abre).Suelen ser empleados como medio de seguridad para evitar colisiones.

1.2. Infrarrojos

Los sensores de infrarrojos son usados para la detección de objetos y cálculo de distancias.

Las dos partes principales de un sensor de infrarrojos son el emisor y el receptor. El emisor es básicamente un diodo LED que emite luz infrarroja. La longitud de onda típica es de 880nm, lo que la hace invisible para el ojo humano. Los receptores pueden ser fotodiodos o fototransistores.

Además del emisor y el receptor es necesario un circuito que detecte una modulación de 40kHz, que es la frecuencia a la que emite el emisor. Limitando la recepción a señales con esta frecuencia permite eliminar los ruidos", o fuentes de luz no deseadas.

Como detector de objetos se utiliza el hecho de que la radiación infrarroja que refleja un objeto se incrementa al disminuir la distancia (es decir, un objeto cercano al emisor refleja una gran cantidad de la radiación que recibe).

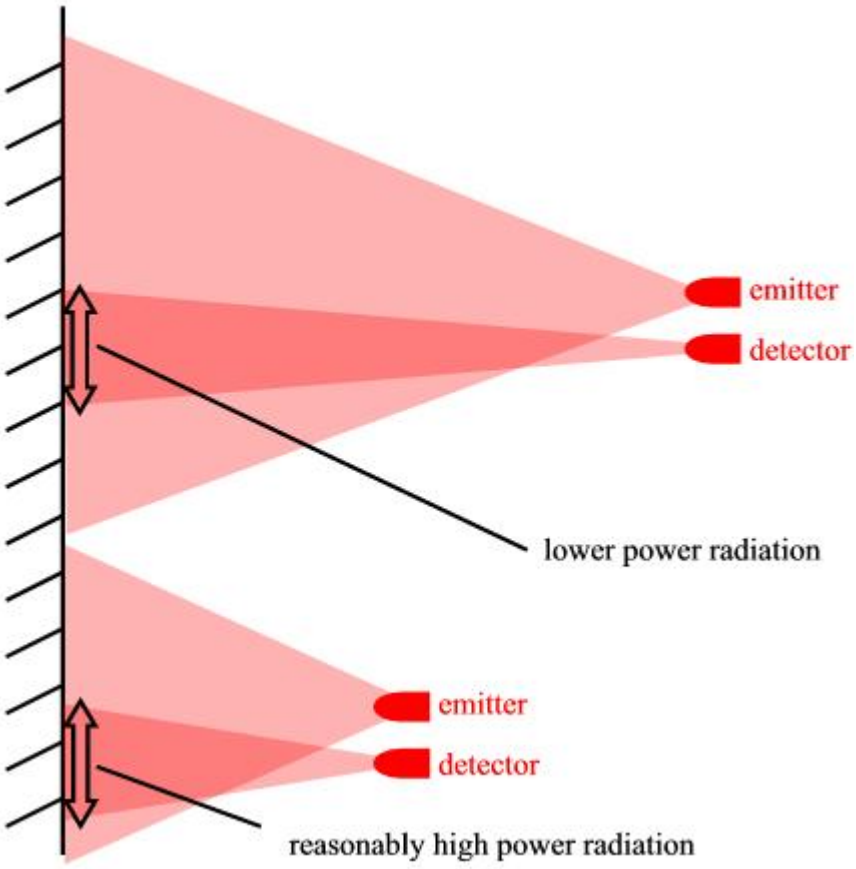


Figura 1.1: Detección de Distancias por Infrarrojos

Para calcular distancias con estos sensores, se emplea la técnica de paralaje, es decir, la medida del ángulo de reflexión entre una fuente de luz conocida y su rayo reflejado. Se emite un rayo de luz infrarroja, este se refleja en el objeto y vuelve al sensor. Cuanto mas cercano se encuentre el objeto, mayor será el ángulo de desplazamiento debido al paralaje. En este caso el sensor receptor es un array o fila de pequeños fotodetectores.

1.3. Ultrasonidos

Los ultrasonidos son usados para el cálculo de distancias y velocidades. Una señal de ultrasonidos de alta frecuencia es enviada por el emisor. La diferencia de tiempo entre la emisión de la señal y la recepción de su eco nos da la distancia del objeto. La velocidad puede ser calculada utilizando el efecto Doppler: el tiempo entre la señal emitida y el eco se incrementa o disminuye proporcionalmente en función de la velocidad del objeto.

Para calcular distancias se usa un transductor [1], que emite una corta señal de ultrasonidos de alta frecuencia (normalmente de 40kHz, mas allá del umbral de audición del oído humano). Estos ultrasonidos rebotan en un objeto, y el eco es recibido por otro transductor. Un circuito o microprocesador se encarga de medir el tiempo entre la emisión y la recepción y calcula la distancia. Puede usarse el mismo transductor para emitir y recibir la señal, pero en este caso es necesario esperar un cierto tiempo para que la membrana del transductor deje de vibrar después de la emisión antes de recibir el eco. Por esta razón los sistemas que emplean dos transductores distintos, uno para emitir y otro para recibir, pueden detectar objetos a distancias mas pequeñas.

1.4. Vision artificial

Varios dispositivos electrónicos pueden ser usados como .ºjos"para un robot:

- Fotoresistores: se trata normalmente de células de sulfato de cadmio(CdS, a menudo llamadas "fotocélulas"). Actúan como resistencias variables,

[1] un tipo especial de altavoz/micrófono que emite/recibe ultrasonidos

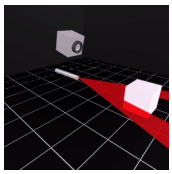


Figura 1.2: Detección de Distancias por Laser

su resistencia varía con la intensidad de la luz que reciben. Cuando no reciben luz se comportan como circuito abierto, Fáciles de usar, pero lentas.

- Fototransistores: básicamente transistores normales a los que se quita sus carcasas opacas de metal o plástico, sustituyéndose por otras transparentes. Muy rápidas, salida no lineal.
- Fotodiodos: al igual que con los fototransistores, son semiconductores a los que se cambia su carcasa normal por otra transparente.

1.5. Laser

Actualmente se encuentran en el mercado dispositivos láser que emiten un haz plano. Una cámara detecta los rayos reflejados y permite haver una reconstrucción de los objetos detectados, usando fórmulas geométricas simples.

Capítulo 2

Movimiento

2.1. Servos

Un es un dispositivo que tiene un eje que puede ser posicionado en una determinado ángulo mediante el envío de una señal codificada. Mientras esta señal siga llegando al servo este intentará mantener ese ángulo frente a posibles fuerzas que intenten cambiar su posición.

2.1.1. Control

Del servo salen tres cables, normalmente de color negro, rojo y amarillo. El cable negro es la referencia de tensión, y se conecta al negativo de la alimentación . El rojo es el positivo (entre 4 V y 6 V). El amarillo se usa para transmitir al servo la señal de control. Esta señal es un pulso de duración variable con un periodo de aproximadamente 20 ms (50 Hz) , aunque este valor no tiene por que ser exacto (servirá cualquier periodo entre 15 y 30 ms). La posición que el servo adoptará es función de la duración del pulso. Esta puede variar entre 1 ms y 2 ms; para 1 ms el servo se posicionará en un extremo, y para 2 ms en el extremo contrario. Un pulso de 1.5 ms hará que el servo permanezca en su posición central.

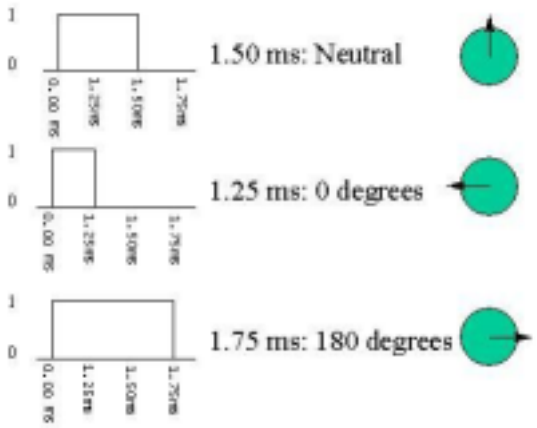


Figura 2.1: Control de un servo por PWM

2.2. Motores

Los motores usados normalmente en robótica son de corriente continua. Cuando se aplica una tensión continua el rotor gira de manera continua. Son reversibles, es decir, girarán en una u otra dirección en función del sentido de la corriente aplicada. Para cambiar el sentido de la corriente el método mas sencillo es usar un "puente en H". Se trata de un circuito electrónico con 4 transistores (normalmente bipolares) que permite el paso de corriente en dos direcciones, de manera que el motor pueda girar en los dos sentidos. Permite el uso de PWM para variar la velocidad. Comercialmente se puede encontrar el circuito integrado L293/L298

Algunas características a tener en cuenta:

- Los motores de corriente continua pueden ser usados con corrientes mayores o menores que las especificadas. Si el motor tiene una corriente especificada de 12 V, operarlo a 6 V es posible, pero con una reducción de la velocidad y el par motor. Inversamente, una tensión mayor hará que el motor gire mas rápido y con mayor par (pero se desgastará mucho antes)
- Los motores de corriente continua consumen una gran cantidad de corriente cuando se les obliga a detenerse, por ejemplo porque hay un obstáculo que detienes su movimiento. Es necesario tener esto en cuenta y diseñar toda la electrónica usada con el motor para soportar esta corriente máxima.
- La velocidad de rotación de un motor de corriente continua es demasiado rápida para ser usada directamente en un robot. Por ello es necesario la utilización de engranajes reductores que disminuyan la velocidad final, a la vez que aumentan el par motor.

La velocidad del motor es directamente proporcional al valor medio de tensión aplicada en sus terminales. Para variar la tension media se emplea un PWM [1](ver apéndice II)

[1]Pulse Width Modulation, o Modulación por ancho de pulso

Capítulo 3

Comunicaciones

La comunicación inalámbrica sirve para conectar el robot móvil al host sin necesidad de un canal físico, lo cual es necesario para dotarlo de movilidad suficiente.

Los requisitos exigidos son:

- Comunicación inalámbrica con un ancho de banda suficiente para permitir la transmisión de imágenes de video.
- Suficiente distancia en medios abiertos y cerrados.

3.1. Alternativas

3.1.1. Bluetooth

Bluetooth [1] es una tecnología de comunicación wireless de corta distancia. Fue diseñada para transmisión de voz y de datos. Ha sido recientemente introducida en el mercado, como medio de comunicación entre dispositivos portátiles.

Se transmite a través de radio, y usa un algoritmo de salto de frecuencias para evitar interferencias en una determinada frecuencia. Su distancia máxima es de 10 metros.

3.1.2. IrDa

IrDa: esta técnica de transmisión es la usada en controles remotos. Desventajas: es point-to-point, muy direccional y no atraviesa materiales opacos.

3.1.3. Wireless LAN

Wireless LAN: (WLan) tambien basado en el estándar 802.11, uso protocolo LAN. Permite la conexión inalámbrica entre redes de area local y ordenadores portátiles. Transmite a 2.5GHz, y es capaz de una velocidad de transmisión de 11Mbps.

[1] Este nombre viene de un jefe vikingo, Harald Gormsson, mas conocido como Bluetooth, rey de Noruega y Dinamarca en el siglo X

3.1.4. Ultra Wide Band

UWB: Ultra Wide Band, si esta nueva tecnología es aprobada por la FCC (existe la preocupación de que interfiera las actuales transmisiones de GPS, tráfico aéreo y televisión)podría conducir al desarrollo de una tecnología muy superior a las LAN inalámbricas del estándar 802.11b. UWB ofrece menor costo, un ancho de banda mayor y un mayor soporte al usuario que la 802.11b y Bluetooth. La señal de UWB no emplea portadora de señal, su consumo es bastante pequeño, en torno a los 50 a 70 mW.

3.2. El estándar 802.11

Este estándar define el protocolo para dos tipos de redes: ad-hoc y cliente-servidor.

Una red ad-hoc es una red sencilla donde la comunicación se establece entre múltiples estaciones en un área definida, sin la necesidad de usar un punto de acceso o servidor. Especifica la "etiqueta"que cada máquina debe observar para que todas tengan un acceso equivalente a la red. Proporciona métodos para arbitrar las solicitudes de acceso al medio, para asegurar que la cantidad de datos vertida al medio sea máxima para todos los usuarios.

El cliente-servidor usa un punto de acceso que controla el tiempo de transmisión para todas las estaciones, y permite que estaciones móviles puedan acceder a la red en distintos puntos. El punto de acceso controla el tráfico desde las máquinas móviles a la red cableada de la red cliente-servidor. Esto permite la coordinación de todas las estaciones y el manejo óptimo del tráfico de datos.

En 802.11b, los dispositivos se comunican a 11Mbps siempre que sea posible. Si la intensidad de la señal o las interferencias están estropeando los datos, los dispositivos cambiarán su velocidad a 5.5Mbps, a 2Mbps y finalmente a 1Mbps. Aunque esto puede significar una reducción de velocidad permite que la red siga funcionando sin errores.

Ventajas de 802.11b:

- Es rápido
- Sin errores

- Tiene un largo alcance (de 76 a 122m en áreas cerradas y unos 300m en áreas abiertas)
- Es sencillo de integrar en redes ya existentes.
- Es compatible con los dispositivos 802.11 DSSS originales.

Desventajas:

- Es caro
- Requiere un punto de acceso
- Puede ser complicado de configurar
- La velocidad puede fluctuar significativamente, dependiendo de interferencias electromagnéticas en la zona y de obstáculos, como por ejemplo personas que circulen entre emisor y receptor.

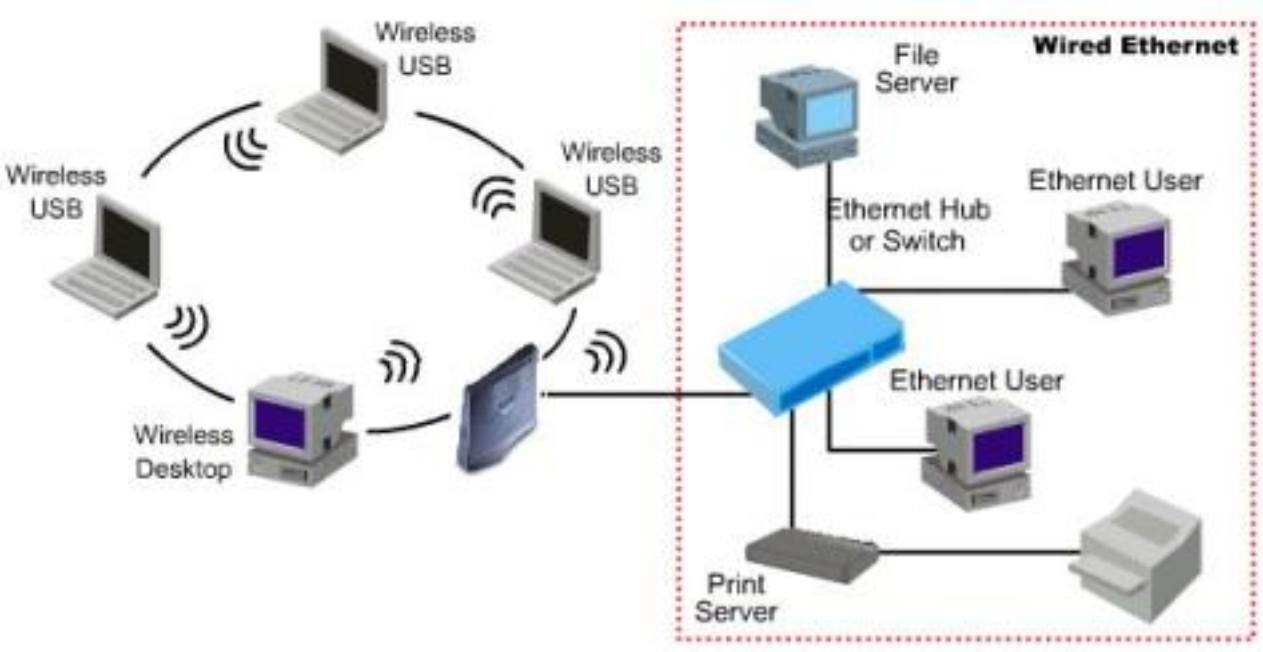


Figura 3.1: Red inalámbrica

Capítulo 4

Sistemas operativos

De el sistema operativo que se use en nuestra aplicación dependerán en gran medida las posibilidades de la misma. El sistema operativo controla y coordina el uso del hardware entre los diversos programas de aplicación. Proporciona los medios necesarios para usar adecuadamente los recursos del sistema, y proporciona un entorno dentro del cual los otros programas pueden realizar su función.

4.1. Tiempo Real

Un sistema de tiempo real puede ser definido como aquel sistema informático en el que la corrección del sistema no sólo depende de los resultados lógicos de los algoritmos, sino que también depende del momento en el que éstos se producen. El tiempo necesario no tiene porqué ser el más corto, sino el adecuado: el sistema tiene que asegurar el tiempo de respuesta.

No todos los sistema de tiempo real son iguales, no es lo mismo controlar el sistema de frenado ABS de un coche o la inyección de combustible en el motor de un avión, que la descompresión y visualización de un fichero mpeg. En el primer caso, la perdida de algún plazo de ejecución puede producir perdidas humanas o graves perdidas materiales; en el segundo caso, sencillamente se tiene una degradación de la calidad del sistema (la imagen se queda congelada o se pierde algún fotograma). A los primeros se les llama sistemas de tiempo real duro o estricto (hard real-time) y a los segundo sistemas de tiempo real blando (soft real-time).

Obsérvese que hemos definido un sistema "de tiempo realz no un sistema "en tiempo real". Un sistema "en tiempo real"es lo que normalmente se entiende por un sistema rápido, capaz de dar la impresión de realidad". Típicamente todas las simulaciones y juegos interactivos pretenden dar sensación de continuidad en el tiempo de forma que cuantas más imágenes generen en menos tiempo, mejor para sus intenciones.

Con estas definiciones en mente, vamos a diferenciar entre dos tipos de sistemas de tiempo real:

1. Sistemas de tiempo real blando
2. Sistemas de tiempo real estricto

1.- Sistemas de tiempo real blando

Este tipo de sistemas deben mantener casi siempre la temporización. Una pérdida de algún plazo de ejecución solamente puede causar una degradación en la calidad ofrecida por el sistema, pero no resulta excesivamente importante. De hecho este tipo de sistemas requieren un buen rendimiento en promedio. Así que estos sistemas necesitan realizar sus plazos de ejecución con frecuencia.

Un ejemplo típico es la visualización de un fichero de video : si el sistema pierde un plazo no es grave, sólo provocará algún salto de imagen que puede ser indetectable o a lo sumo incómodo para el usuario.

2.-Sistemas de tiempo real estricto

Se trata de sistemas cuyos plazos de ejecución no pueden perderse de ningún modo. Los sistemas de tiempo real estricto no pueden utilizar la mejora del rendimiento medio para compensar un mal rendimiento en el peor caso. La pérdida de un plazo de ejecución puede causar un error irrecuperable. Este tipo de sistemas debe garantizar todos los tiempos de respuesta. Las aplicaciones que cubren van desde el control a la supervisión de motores, robots, sistemas de adquisición de datos, plantas de procesado, sistemas de telecomunicación. . . entre otras máquinas e instrumentos donde la temporización sea un factor decisivo.

Como ejemplo pongamos una secuencia de lanzamiento de un cohete, donde los plazos límite deben ser respetados o si no el cohete puede explotar. Otro ejemplo sería una cadena de montaje, que necesita trabajar a intervalos de tiempo precisos o en caso contrario el resultado pueden ser cientos de productos defectuosos.

RT-Linux está especialmente diseñado para trabajar como un sistema de tiempo real estricto. Trata de reducir la complejidad para así reducir la impredecibilidad del sistema, como hacen otros sistemas de tiempo real como VxWorks, Linx, OS/9 y tantos otros. Pero es diferente a todos ellos porque no es un sistema específico. Por contra, está basado en Linux, lo que le aporta una compatibilidad mucho más variada con herramientas y programas ya existentes.

Por otro lado, desde el punto de vista de los sistemas de tiempo real blando, RT-Linux todavía no está preparado para ello. ¿Por qué? Porque los sistemas de tiempo real blando "serios"deben dar soporte QOS (Quality Of Service: Calidad De Servicio), de forma que aporten algún sistema de control

que permita regular la posibilidad de perder algunas iteraciones en las tareas de tiempo real blando. La clave podría estar en establecer distintos niveles de prioridad, como por ejemplo tratando separadamente las tareas de tiempo real estricto (de mayor prioridad) de las de tiempo real blando (de menor prioridad).

4.2. Linux y RT-Linux

Como hemos descrito antes, RT-Linux está especialmente diseñado para trabajar como un sistema de tiempo real estricto. Pero este sistema operativo es mucho más que eso: aprovecha que está basado en uno de los sistemas operativos mas populares y difundidos para construir algo entre un sistema de múltiples propósitos (como es Linux) y un sistema de tiempo real.

Realmente, RT-Linux provee soporte para programas y tareas de tiempo real, dejando el problema de ejecutar aplicaciones rápidas.[a]l Linux estándar. Pero de hecho estamos mezclando dos propiedades incompatibles en el mismo sistema operativo: operatividad de tiempo real y servicios POSIX (Portable Operating System Interface) estándares de Linux, como GUI, TCP/IP...

La idea principal a tener en mente cuando utilizamos RT-Linux es que tendremos pequeñas y simples tareas de tiempo real con sofisticadas tareas de Linux normal, conectadas ambas entre sí pero de forma que el planificador dé una preferencia prácticamente absoluta a la tarea RT. De hecho, RT-Linux nos da la posibilidad de interrumpir una seccion crítica de Linux para permitir ejecutarse a una tarea de tiempo real.

RT-Linux combina pues las características de un sistema de tiempo real con la ventaja de tener disponible una gran cantidad de funcionalidad de Linux. Ciertamente, Linux ofrece un buen número de herramientas y servicios:

- X-windows y soporte a entornos gráficos (TCL/TK, etc.) Redes de datos Compiladores y herramientas de programación utilidades GNU política GNU (disponibilidad de código fuente) Desarrollo rápido y facilidad de formación del usuario (mucha documentación) Amplio y actualizado soporte (¡en todo el mundo!)
- La principal aplicación de RT-Linux es como equipamiento de laboratorio, pero también ha demostrado trabajar bien en sistemas empotrados

como robots y dispositivos. En tareas de control y medida ofrece buenos resultados. Pero la mayor parte de aplicaciones que se han desarrollado hasta el momento bajo RT-Linux han sido sistemas de adquisición de datos.

4.3. Historia de RT-Linux

Hace pocos años RT-Linux fue desarrollado como parte de un proyecto en el Instituto de Minería y Tecnología de Nuevo Mexico. La primera versión se encontraba bastante limitada, pero su funcionalidad era bastante práctica y conseguía una temporización bastante precisa. Esta primera edición no disponía de funciones que dieran soporte a procesos de tiempo real, como por ejemplo semáforos, funciones de apoyo de temporización, paso de mensajes, soporte multiprocesador (SMP), etc.

Después, se han hecho algunas modificaciones importantes desde que se sacara la primera versión. Se incluyeron semáforos y otras funcionalidades para suministrar métodos de comunicación entre procesos (para comunicar procesos de tiempo real con los demás). En sucesivas versiones se fue consiguiendo mayor estabilidad, estableciéndose que las versiones 2.X donde la X fuese par serían las de probada estabilidad.

A partir de la primera versión aparecida se hicieron otros sistemas de tiempo real. El DIAPM (Dipartimento di Ingegneria Aerospaziale, Politecnico di Milano) creó una versión propia en la búsqueda de un temporizador "bajo demanda"(one-shot) más eficiente, llamado RTAI (Real Time Application Interface). El temporizador periódico de RT-Linux era suficientemente bueno, y tanto éste como la temporización por demanda estaban basados en el chip 8254 que utiliza la arquitectura de bus ISA. Pero reprogramar el temporizador requiere un intervalo determinado así que la temporización bajo demanda resulta costosa. RTAI usa un sistema diferente, más eficiente, para crear temporizadores de este tipo: el CPU TSC (Time Stamp Clock). Pero el inconveniente es que se trata de una característica que sólo poseen los Pentium y los microprocesadores posteriores. Véase la página http://server.aero.polimi.it/projects/rtai para mayor información.

RT-Linux ha continuado su desarrollo más allá de la versión 2.0. En esta nueva versión se cambió sensiblemente la API para adaptarse al estándar POSIX, así que la anterior API queda obsoleta. De hecho, fue bastante difícil

acomodar RT-Linux en las estrictas especificaciones de POSIX.4 (renombradas a 1003.1b desde 1993). A consecuencia de estas modificaciones, la API de RTAI dejó de ser compatible con RT-Linux.

Junto a estas modificaciones, RT-Linux 2.0 incluyó soporte multiprocesador SMP (Symmetric Multi Processors). La estabilidad alcanzada por RT-Linux hace de él un sistema ampliamente usado a pesar de su juventud. Pero al importar código de la antigua API a la nueva API implementada aparecen algunos problemas a pesar de la posibilidad de usar una librería especialmente diseñada para ello.

Otras variaciones de RT-Linux fueron diseñadas como adaptaciones especiales de la versión original. MiniRTLinux (http://www.rtlinux.org/rtlinux.new/minirtl.html) es una edición pensada para arrancar RT-Linux desde un diskette. De hecho, no se necesita un disco duro para hacerla funcionar, así que puede ser fácilmente integrada en un sistema específico. Hay otras versiones en preparación están consiguiendo funcionar sobre otras plataformas, como estaciones de trabajo Alpha y PowerPC.

A destacar otro proyecto diferente para adaptar Linux a los sistemas de tiempo real: se llama KURT (Kansas University Real Time Linux). Su filosofía es muy diferente, porque se instala dentro del núcleo de Linux y modifica el planificador para incluir una nueva política de planificación, llamada SCHED-KURT. Es una especie de planificador cíclico, basado en una tabla elaborada previamente. Es más simple y eficiente que otros, pero cada nuevo proceso de tiempo real fuerza a una redefinición completa de todo el plan, y la tabla suele tener un tamaño considerable. Esta es una de las razones por las que KURT sólo trabaja con tareas de tiempo real blando. Por ello también incluye algunos mecanismos de control de la calidad de servicio (QOS: Quality Of Service) y ofrece además una amplia variedad de funciones Linux dentro de las tareas de tiempo real, contrariamente a las limitadas tareas de tiempo real de RT-Linux.

Un proyecto bastante reciente, proveniente de Timesys (www.timesys.com) y la Carnegie Mellon University ha sido desarrollado como parte de algunas herramientas comerciales como TimeWiz y TimeTrace. Es llamado TimeSys Linux/RT e integra una parte del kernel de Linux dentro del núcleo de RT-Linux con la intención de dotar a las tareas de tiempo real de mayor flexibilidad. Se ha usado una versión modificada de RTAI como kernel de tiempo real para este sistema operativo. No obstante, el núcleo ha sido preparado para permitir reserva de prioridad, esto es, deja a las tareas de tiempo real que puedan reservar tiempo de CPU que vayan a usar después, y de esta

forma suministrar una replanificación del sistema más fácil. A día de hoy, el coste de esta versión se halla entre 50 dólares US (Standard version) y 200 (Professional version). TimeSys también ha desarrollado otros productos como Java/RT y otras herramientas para sustemas de tiempo real.

En el campo de los sistemas empotrados, Kwon Seok Kuon ha desarrollado un RT-Linux especialmente diseñado para ser utilizado en una tarjeta especial llamada Csimm que utiliza una versión dedicada de Linux denominada Clinux. Modificó RT-Linux v0.9J para la versión Csimm del kernel Clinux (http://www.uclinux.org/simm/realtime).

Hoy en día, RT-Linux se encuentra en una nueva versión, la 3.0beta, y como puede suponerse no está aún comprobada con suficiente seguridad. Se le ha dotado de mayor compatibilidad POSIX y es capaz de trabajar con otras plataformas como estaciones de trabajo Alpha y PowerPC.

Para mayor información acerca de casi todas estas versiones, se puede consultar www.rtlinux.org y www.rtlinux.com.

Capítulo 5

CORBA

5.1. Precedentes.

La aparición de está muy ligada a la existencia de sistemas distribuidos. En un principio, con el mainframe, el sistema se controlaba de forma centralizada. Con la introducción de los PC´s se pudo descargar gran parte de la carga de trabajo que tenían que soportar los mainframes, apareciendo por lo tanto la arquitectura cliente/servidor. Así se sustituyeron estos "monstruos"por servidores UNIX mucho más económicos. Esto trajo consigo una mayor independencia de las secciones dentro una empresa para crear aplicaciones que se adaptasen mejor a sus necesidades y no depender tanto del control central.

En este tipo de sistema, los datos se hallan almacenados en el servidor, el interfaz de usuario en el cliente y la lógica de control en cualquiera de ellos. Tenía la desventaja de que al cambiar alguna parte del cliente, el nuevo programa debía ser distribuido a todos los usuarios.

Esta arquitectura evolucionó hacia la multicapa (multitier) cliente/servidor. Divide al servidor en varias capas, para poder aislar al cliente de los cambios producidos en la aplicación. De esta forma se podrán hacer cambios en la aplicación con poca probabilidad de que afecten al cliente. Además permite más flexibilidad en la aplicación al poder colocar los módulos del servidor en máquinas diferentes.

A continuación se pueden encuadrar los sistemas distribuidos. En estos sistemas todas las funciones de la aplicación se exponen como objetos, cada uno de los cuales puede usar los servicios proporcionados por otros objetos del mismo o de diferente sistema. Por lo tanto se oscurece la diferencia entre cliente y servidor: los clientes pueden crear objetos que actúen como servidores. Proporcionan una gran flexibilidad. Ésta se basa en que la interfaz (protocolo de comunicación entre dos módulos) de un módulo especifica a los demás los servicios que ofrece y cómo se usan. Mientras la interfaz no varíe, se podrán hacer todos los cambios que se deseen dentro del módulo. Se deben incluir servicios que permitan localizar a los diferentes módulos.

Llegados a este punto es donde tenemos que empezar a hablar de CORBA. En los sistemas distribuidos es muy importante la definición de la interfaz y ciertos servicios como la búsqueda de módulos, ... Lo que hace CORBA es proporcionar un estándar para poder definir estas interfaces entre módulos, así como algunas herramientas para facilitar la implementación de dichas interfaces en el lenguaje de programación escogido. Adicionalmente se

han incluido algunos servicios estándar accesibles a todas las aplicaciones en CORBA. Además CORBA proporciona todo el mecanismo que permite a los distintos módulos de una aplicación comunicarse entre sí.

Las dos principales características de CORBA es que es independiente tanto de la plataforma como del lenguaje de la aplicación. La independencia de plataforma significa que los objetos de CORBA se pueden utilizar en cualquier plataforma que tenga una aplicación CORBA ORB. La independencia de lenguaje se refiere a que los objetos CORBA y los clientes se pueden implementar en cualquier lenguaje de programación. Así, a un objeto CORBA no le hará falta saber el lenguaje en que ha sido escrito otro objeto con el que se esté comunicando.

Para diseñar aplicaciones distribuidas hay más opciones aparte de CORBA, tales como:

- Programación con sockets. De esta forma las aplicaciones se comunican entre sí a través de un çanal". Es la forma más directa de comunicar componentes de una aplicación. Se escriben o leen los datos del socket. Al ser una forma de programación de bajo nivel, no se introduce mucha sobrecarga a la aplicación, pero no es muy adecuada para manejar tipos de datos complejos, especialmente cuando los módulos de la aplicación se encuentran en diferentes tipos de máquinas o están implementados en diferentes lenguajes.

- Llamada a procedimiento remoto (RPC). El programador define una función que usa sockets para comunicarse con el servidor remoto que ejecuta la función y devuelve el resultado de nuevo mediante sockets. El RPC es un mecanismo bastante potente como para ser la base de muchas aplicaciones cliente/servidor.

- Entorno de computación distribuido de la OSF (DCE). Se trata de un conjunto de estándares iniciados por el Open Software Foundation (OSF). Nunca ha tenido demasiada aceptación.

- Modelo de objetos para componentes distribuidas de Microsoft (DCOM). Ofrece capacidades similares a las de CORBA. Es un modelo de objetos relativamente robusto. Su defecto es que está disponible casi exclusivamente en el entorno Windows aunque para aplicaciones específicas de Windows es una muy buena solución.

item Invocación de métodos remotos de Java (RMI). Su principal ventaja es que soporta pasar objetos por valor y su principal desventaja es que es una solución solamente para Java.

5.2. Breve historia de CORBA.

5.2.1. Introducción al Grupo de Administración de Objetos

(OMG).

Su misión es proporcionar un entorno de trabajo arquitectural común para aplicaciones orientadas a objetos basadas en especificaciones de interfaces ampliamente disponibles. Esto se logra con el establecimiento de la Arquitectura para el Manejo de Objetos (OMA) de la cual CORBA es sólo una parte. La OMA consta de la función ORB, servicios de objetos (serviciosCORBA), facilidades comunes (facilidadesCORBA), interfaces de dominio y objetos de aplicación. El papel de CORBA en el OMA es implementar la función ORB.

5.2.2. Las versiones de CORBA

La primera surgió en Diciembre de 1990 y fue seguida rápidamente por CORBA 1.1 que definía el lenguaje de definición de interfaces (IDL) así como los interfaces de programación de aplicaciones (API) para comunicar las aplicaciones con los ORBs. La versión de CORBA 2.0 (1994) ya especifica un protocolo estándar para la comunicación entre ORBs. Así surgió el protocolo inter-ORBs de Internet (IIOP) aplicable sólo a redes basadas en TCP/IP.

5.3. La arquitectura CORBA.

CORBA es una arquitectura orientada a objetos. Por eso exhibe muchas de las características comunes a otros sistemas orientados a objetos, incluyendo herencia de interfaces y polimorfismo. Lo que diferencia a CORBA y lo hace más interesante es que proporciona esta capacidad incluso cuando se utiliza con lenguajes no orientados a objetos como C y COBOL.

Los dos pilares fundamentales de la arquitectura CORBA son el ORB (Object Request Broker), que dirige la comunicación entre objetos CORBA, y el IDL (Interface Definition Language), que define las interfaces de los componentes de la aplicación sobre los que se construyen las aplicaciones CORBA.

5.3.1. El Object Request Broker (ORB)

El Object Request Broker es uno de los pilares fundamentales de CORBA. Un ORB es un componente software cuyo propósito es facilitar la comunicación entre objetos. Esto lo hace proporcionando una serie de facilidades, una de las cuales es localizar un objeto remoto dada una referencia a ese objeto y otra es la ordenación de los parámetros y valores de retorno a y de invocaciones a métodos remotos.

Su funcionamiento es el siguiente: cuando un módulo de una aplicación quiere usar un servicio proporcionado por otro módulo obtiene una referencia del objeto que provee ese servicio. Después de obtenerla, el módulo puede invocar métodos en ese objeto. La primera responsabilidad del ORB es resolver peticiones de referencias de objetos, permitiendo a los módulos de la aplicación establecer conexión entre ellos.

Otra de las responsabilidades del ORB es el marshaling y el unmarshaling. Después de que un módulo de la aplicación haya obtenido una referencia del objeto cuyos servicios quiere usar, ese módulo ya puede invocar métodos en ese objeto. Generalmente estos métodos necesitan parámetros como entrada y devuelven otros parámetros como salida. El ORB debe recibir los parámetros de entrada del módulo que llama al método y marshal estos parámetros. Esto quiere decir que el ORB traduce los parámetros a un formato (on-the-wire format) que puede ser transmitido por la red hasta el objeto remoto. El ORB también unmarshals los parámetros devueltos, convirtiéndolos a un formato que el módulo llamante entienda.

Todo esto se hace de forma transparente a la intervención del programador. Una aplicación cliente invocará un método remoto y recibirá los resultados como si el método fuese local.

Gracias a este proceso se consigue la independencia de plataforma, debido a que los parámetros se traducen en la transmisión a un formato independiente de la plataforma (el on-the-wire format forma parte de las especificaciones CORBA) y en recepción se convierten al formato específico de la

plataforma. Un cliente ejecutándose en un sistema Macintosh podrá invocar métodos de un servidor que se ejecute en un sistema UNIX. Además de la independencia del SO usado, las diferencias de hardware (como el ordenamiento de los bytes, endianness, la longitud de las palabras, etc.) son irrelevantes puesto que el ORB hace las conversiones necesarias automáticamente.

En resumen, las responsabilidades del ORB son:

- Dada una referencia a un objeto por un cliente, el ORB localiza la correspondiente implementación del objeto (el servidor).
- Cuando el servidor está localizado, el ORB asegura que el servidor está preparado para recibir la petición.
- El ORB del lado del cliente acepta los parámetros del método que se está invocando y marshals los parámetros a la red.
- El ORB del lado del servidor unmarshals los parámetros de la red y se los entrega al servidor.
- Los parámetros de retorno, si existen, se marshal/unmarshal del mismo modo.
- La mayor ventaja que ofrece el ORB es el tratamiento de los datos independientemente de la plataforma.

5.3.2. Lenguaje de definición de interfaces (IDL)

La otra pieza fundamental de la arquitectura CORBA es el uso del lenguaje de definición de interfaz (IDL) que especifica interfaces entre objetos CORBA. Es un lenguaje estándar.

Los interfaces descritos en IDL se pueden ajustar a cualquier lenguaje de programación lo que hace que las aplicaciones y los componentes CORBA sean independientes del lenguaje utilizado para implementarlos. El pliego de condiciones IDL es el responsable de asegurar que los datos son intercambiados correctamente entre lenguajes diferentes. Por ejemplo, el tipo long en IDL es un entero de 32 bits con signo, que corresponde a un long de C++ o a un int de Java.

Esta independencia del lenguaje se consigue gracias al language mapping, que es un pliego de condiciones que empareja las construcciones en IDL

con las de un lenguaje de programación particular. Por ejemplo en el C++ mapping el "interface"de IDL corresponde a una çlass"de C++. La OMG ha definido un número de language mapping estándar para muchos lenguajes, tales como C, C++, COBOL, Java, Smalltalk, etc. Mappings para otros lenguajes existen, pero o no son estándar o están en proceso de serlo.

Otra característica importante de IDL es que no se trata de un lenguaje de implementación (no se pueden escribir aplicaciones en IDL), su único propósito es definir interfaces. Comparando con C++, se podría decir que las definiciones IDL son análogas a los ficheros de cabecera para las clases (que no contienen la implementación de la clase, sino que más bien definen su interfaz). Con respecto a Java sería parecido a las definiciones de los interfaces Java.

CORBA no dicta el uso de un lenguaje en particular, por lo que deja a los diseñadores de la aplicación la elección del lenguaje que más se ajuste a sus necesidades. Incluso se pueden utilizar varios lenguajes para implementar cada parte de la aplicación. Por ejemplo, el cliente puede estar escrito en Java que asegura que pueda funcionar en cualquier máquina, mientras que el servidor se puede implementar en C++ para obtener unas buenas prestaciones.

5.3.3. El modelo de comunicaciones de CORBA

Vamos a intentar entender el papel de CORBA en una red de ordenadores. Típicamente ésta consiste en sistemas que están físicamente conectados. Esta capa física proporciona el medio a través del cual tiene lugar la comunicación. Más allá de la capa física se encuentra la capa de transporte, que incluye los protocolos responsables de mover los paquetes de datos desde un punto hacia otro. CORBA es neutral respecto estos protocolos de red, es independiente del protocolo utilizado y podría funcionar con cualquiera.

El estándar CORBA especifica lo que se conoce como el Protocolo General Inter-ORB (GIOP) que, en un nivel alto, establece un estándar para la comunicación entre varios ORB´s de CORBA. GIOP es tan sólo un protocolo general, por lo que el estándar CORBA también especifica otros protocolos que detallan GIOP para usar un protocolo de transporte particular (como TCP/IP o DCE). El utilizado para redes TCP/IP se denomina Internet Inter-ORB Protocol (IIOP). Los fabricantes deben implementar este protocolo para ser considerados conformes a CORBA. Este requerimiento

ayuda a asegurar la interoperabilidad entre productos CORBA de diferentes fabricantes, aunque cada uno puede tener además sus propios protocolos.

Un ORB puede soportar cualquier protocolo, pero siempre debe que incluir el IIOP (se negocia el protocolo a usar al establecerse la conexión entre los ORB´s). De todas formas, los ORB´s de CORBA normalmente se comunican usando el IIOP debido, en parte, a que este protocolo es el correspondiente al protocolo TCP/IP, que es el usado en Internet. En el argot CORBA/IIOP las referencias a objeto se pasan a denominar Referencias a Objetos Interoperables o IOR´s.

Resumiendo, las aplicaciones CORBA se construyen encima de los protocolos derivados de GIOP (como el IIOP). Estos protocolos están, a su vez, encima de los protocolos de transporte (TCP/IP, DCE). Las aplicaciones CORBA no están limitadas a usar sólo uno de estos protocolos, sino que se puede usar un puente para interconectar aplicaciones situadas en redes que trabajen con diferentes protocolos. Se puede ver que, más que suplantar los protocolos de transporte, la arquitectura CORBA crea otra capa (la capa del protocolo Inter-ORB) que utiliza estos protocolos como soporte. Esta es otra de las claves de la interoperabilidad entre las aplicaciones CORBA ya que no se dicta el uso de un protocolo de transporte particular.

5.3.4. El modelo de objetos de CORBA

En CORBA, todas las comunicaciones entre objetos se hacen a través de referencias de objeto (explicadas posteriormente). Además, la visibilidad de los objetos se obtiene únicamente pasando referencias a esos objetos. Los objetos no pueden ser pasados por valor (al menos en la especificación actual de CORBA). En otras palabras, los objetos remotos en CORBA permanecen remotos, no hay actualmente ninguna forma de mover o copiar un objeto de un sitio a otro.

Todas las arquitecturas orientadas a objetos ofrecen un modelo de objetos, que describe cómo se representan los objetos en el sistema. Tres de las principales diferencias entre el modelo de objetos de CORBA y los modelos tradicionales radican en la forma semi-transparente de distribuir los objetos en CORBA, el tratamiento de las referencias a objetos y el uso de los llamados adaptadores de objetos (como el BOA -Basic Object Adapter-).

Para un cliente CORBA, una llamada a un método remoto es exactamente igual a una llamada a un método local. La naturaleza distribuida de los

objetos en CORBA es transparente a los usuarios de dichos objetos. Debido a que los objetos se encuentran distribuidos hay más posibilidades de que se produzca un fallo, por lo que CORBA debe ofrecer la forma de manejar estas situaciones. Lo resuelve ofreciendo un conjunto de excepciones de sistema, que pueden ser lanzadas por cualquier método remoto.

En una aplicación distribuida, hay dos métodos posibles de obtener acceso a un objeto de otro proceso. Uno se conoce como "paso por referencia". Se explica muy bien mediante un ejemplo. Supongamos dos procesos, A y B. El primer proceso (A) pasa una referencia a un objeto suyo al segundo proceso (B). Cuando el proceso B invoca un método en ese objeto, el método lo ejecuta el proceso A puesto que él posee el objeto. El proceso B sólo tiene visibilidad del objeto (a través de la referencia al objeto) y por lo tanto sólo puede pedir al proceso A que ejecute los métodos en su lugar. Es decir, pasar un objeto por referencia significa que un proceso concede visibilidad de uno de sus objetos a otro proceso mientras retiene la propiedad del objeto. Las operaciones en el objeto a través de la referencia al objeto las procesa el propio objeto.

El segundo método de pasar un objeto se denomina "paso por valor". El estado actual del objeto se pasa al módulo que lo pide donde se creará una nueva copia del objeto. Siguiendo con el ejemplo de antes, cuando el proceso B invoque métodos en el objeto se ejecutarán en dicho proceso (en la copia) en vez de en el proceso A, donde el objeto original reside. Por lo tanto el estado del objeto original no cambia.

Como se dijo antes, en el modelo de objetos de CORBA todos los objetos se pasan por referencia. Al pasar un objeto por valor es necesario asegurarse de que el módulo que recibe el objeto tiene implementaciones para los métodos soportados por dicho objeto. En cambio esto no es necesario cuando los objetos son pasados por referencia.

Sin embargo, hay algunos problemas asociados a pasar los objetos por referencia. Todas las llamadas a los métodos serán remotas, por lo que si un componente invoca repetidas veces métodos en un mismo objeto remoto se producirá una gran cantidad de sobrecarga en la comunicación entre los dos componentes (sería más eficiente pasar el objeto por valor, de forma que el componente pueda manipular el objeto localmente).

El estándar CORBA define una serie de adaptadores de objetos (object adapters) cuyo primer propósito es conectar la implementación de un objeto con su ORB. La OMG proporciona tres adaptadores de objetos como

muestra: el BOA (Basic Object Adapter), el Library Object Adapter y el Object-Oriented Database Adapter, estos dos últimos útiles para acceder a objetos almacenados de forma persistente. El BOA suministra a los objetos CORBA un conjunto común de métodos para acceder a las funciones del ORB. Estas funciones van desde la autentificación del usuario hasta la activación de objetos. El BOA es, en definitiva, el interfaz entre los objetos CORBA y el ORB.

Una característica importante del BOA es su capacidad de activar y desactivar objetos. Provee cuatro tipos de políticas de activación, que indican cómo se deben inicializar los componentes de la aplicación (cómo se va a acceder al servidor):

- Shared server (servidor compartido), en la que un único servidor es compartido entre múltiples objetos.
- Unshared server (servidor no compartido), un servidor contiene sólo un objeto.
- Server-per-method (servidor por método), que automáticamente arranca un servidor cuando se invoca un método en un objeto, y termina el servidor cuando el método regresa.
- Persistent server (servidor persistente), en la que el servidor es arrancado manualmente (por un usuario, un demonio del sistema, ...).

  Esta variedad de políticas de activación permite elegir el tipo de comportamiento que más se adapte a un determinado servidor. Por ejemplo, un servidor que tarde mucho en arrancar trabajaría mejor como persistent server; por el contrario un servidor que arranque rápidamente bajo demanda podría trabajar bien como server-per-method.

5.3.5. Clientes y servidores CORBA

Tradicionalmente, en una aplicación cliente/servidor, el servidor es el componente que provee un servicio a otros componentes de la aplicación, mientras que un cliente es un componente que utiliza servicios proporcionados por un servidor. En CORBA, igualmente, un componente es considerado un servidor si contiene objetos CORBA cuyos servicios están accesibles para otros objetos. De la misma forma, un componente es considerado un cliente si accede a servicios de otros objetos CORBA. Pero cuando consideramos una

invocación a un método remoto los papeles de cliente y servidor se pueden invertir temporalmente, ya que un objeto CORBA puede participar en múltiples interacciones simultáneamente.

Si un componente crea un objeto y proporciona a otros componentes visibilidad a ese objeto (permite a otros componentes obtener referencias a ese objeto), ese componente actúa como un servidor para ese objeto; cualquier petición hecha a ese objeto por otro componente será procesada por el componente que creó el objeto. Ser un servidor CORBA significa que el componente (el servidor) ejecuta métodos en un objeto particular en nombre de otros componentes (los clientes).

Pero frecuentemente un módulo de una aplicación puede proporcionar servicios a otros módulos mientras a su vez accede a servicios de otros módulos. En este caso el módulo está actuando como cliente y servidor a la vez. De hecho, dos módulos pueden actuar simultáneamente como clientes y servidores uno del otro.

Consideremos el siguiente escenario: un primer módulo (A) recibe una referencia a un objeto creado por un segundo módulo (B) e invoca un método en ese objeto. Aquí el módulo A actúa como cliente y el módulo B como servidor. Ahora supongamos que como un parámetro del método invocado se debe pasar una referencia a un objeto. Si el módulo B ahora invoca algún método en ese objeto, para esta invocación en particular el módulo A actúa como servidor mientras que el módulo B actúa como cliente. Los dos módulos no han cambiado sus papeles globales en la aplicación, pero han invertido temporalmente sus funciones como cliente y servidor. Por eso los términos cliente y servidor dependen del contexto del método invocado y de en qué módulo resida el objeto del método.

Aunque un módulo de una aplicación puede funcionar tanto como cliente como servidor, es usual etiquetar al módulo como uno de los dos (no ambos a la vez). Si la función global de un módulo es usar los servicios proporcionados por otro módulo, y sólo proporciona objetos como argumentos a los métodos de módulo invocado, nos podemos referir al primer módulo como cliente y al segundo como servidor. Los métodos invocados de esta forma se denominan client callback methods (un callbak es un método implementado por un cliente e invocado por un servidor). Son realmente importantes en CORBA debido a su carencia de pasar objetos por valor.

5.3.6. Stubs y skeletons

Después de crear las definiciones de la interfaz de un módulo usando IDL se ejecutan los ficheros IDL resultantes en un compilador IDL. Éste genera lo que se conoce como client stubs y server skeletons (no es necesario escribirlos). Sirven como una especie de "pegamento"que conecta las especificaciones de la interfaz IDL independientes del lenguaje con la implementación en un código de un lenguaje específico.

Un client stub es una pequeña parte de código que hace que la interfaz de un servidor CORBA esté disponible para un cliente, y se compila conjuntamente con la parte cliente de la aplicación, es decir, los client stubs de cada interfaz se proporcionan para incluirlos en los clientes que usan ese interfaz. El client stub de un interfaz determinado proporciona una maqueta de la implementación de cada uno de los métodos de esa interfaz. Los métodos del client stub simplemente se comunican con el ORB para marshal y unmarshal parámetros.

Un server skeleton es una pequeña parte de código que proporciona la .ªrmazón"sobre la cual se construye el código de implementación del servidor para un determinado interfaz. Para cada método de una interfaz el compilador IDL genera un método vacío en el server skeleton. El programador debe entonces proporcionar una implementación para cada uno de estos métodos.

5.3.7. Servicios CORBA y facilidades CORBA

El OMA, visto anteriormente y del que CORBA es una parte, también provee capacidades adicionales en la forma de servicios CORBA y facilidades CORBA que proporcionan servicios y facilidades horizontales (generalmente útiles en todas las industrias) y verticales (diseñadas para industrias específicas). Estas capacidades incluyen manejo de eventos, obtención de licencia, seguridad, nombramiento, manejo de la interfaz del usuario, intercambio de datos y muchos más. Las interfaces para usar estas capacidades están estandarizadas por el OMG, y además están especificadas en IDL, lo que quiere decir que las aplicaciones pueden usar estos servicios tal como usan cualquier otro objeto CORBA.

Parte II

Diseño

Capítulo 6

Diseño general

Como plataforma de desarrollo se decidió adquirir un coche o camión de pequeño tamaño para permitir la movilidad en interiores, que a su vez fuese estable y con gran capacidad de carga, en previsión de futuras ampliaciones (principalmente sensores).

Se escogió un modelo de radio control de Tamiya, tipo "Bigfoot", por su estabilidad, capacidad de carga y potencia motriz. Aunque este es evidentemente poco mas que un juguete, sirve bien como base después de algunas pequeñas modificaciones mecánicas, de las cuales las mas importantes fueron el cambio del servo de dirección por otro con un mayor par (necesario debido al aumento de peso) y engranajes metálicos que le otorgan una mayor durabilidad, y la supresión de los amortiguadores originales.

A lo largo de toda la fase de diseño se ha tenido especial cuidado en la facilidad de cambio de componentes. Las baterías pueden reemplazarse fácilmente para permitir un mayor tiempo de funcionamiento.

Como cerebro principal se decidió usar un Beowulf, es decir, un computador con procesamiento paralelo, montado en un armario de control de la firma Honeywell, que actualmente se encuentra en el aula Centro de Cálculo de DISAM. Desafortunadamente no se consiguió tenerlo disponible a fecha de entrega del presente proyecto, por lo que se ha empleado uno de los PCs de dicho Centro de Cálculo.

En cuanto al procesamiento a bordo, los requisitos especificaban el uso de al menos dos procesadores heterogéneos, es decir, de características distintas. La decisión final fue usar 2 computadores: uno mas potente para la toma de decisiones, comunicación con el host...otro mas pequeño que se encarga de el control de los servos y motores (PWM), liberando de estas tareas al procesador principal.

Como procesador principal se eligió una placa Wafer 5820, con factor de forma de 3.5 pulgadas. En tan escaso tamaño disponemos de un completo PC,

Figura 6.1: Plataforma móvil

con procesador Pentium, ethernet, todos los puertos necesarios (incluyendo 4 puertos serie y USB) etc, con un consumo muy pequeño y poca disipación de calor, lo que ahorra no pocos problemas de diseño.

Como procesador secundario se decidió usar un nuevo modelo de procesador con arquitectura nativa Java, el JStamp .

La comunicación "interna"entre los dos procesadores se lleva a cabo a través de un cable serie RS-232, al menos hasta que se pueda usar un bus CAN. Esta previsto que el JStamp esté preparado para usar este bus en un futuro cercano. La comunicación serie tiene la ventaja de una fácil programación, pero es muy limitada en cuanto ancho de banda y velocidad.

Un objetivo secundario de este proyecto es el uso de tecnologías Real Time",tiempo real.

Para este proyecto se preseleccionaron dos sistemas operativos en tiempo real: RTLinux y VxWorks. Finalmente se optó por el primero debido a las dificultades de la interfaz de usuario y dificultad de programación que ofrece VxWorks.

Capítulo 7

Comunicación inalámbrica

Requisitos:

- RG5 :El host y el robot móvil deben estar conectados inalámbricamente.
- RG5.1: La distancia operativa de transmisión debe ser mayor de 20 metros.
- RG5.2: La comunicación inalámbrica será tal que permita la transmisión de imágenes de video desde el robot, así como la de los diversos sensores.

Se ha dedicado mucho tiempo al estudio de las diversas alternativas para disponer de una buena comunicación inalámbrica. Finalmente se ha decidido usar una wireless PC-Card WL110 de Compaq . Esta tarjeta PCMCIA necesita de un adaptador para conectarla al bus PC104 del Wafer 5820.

Conjuntamente con las tarjetas PCMCIA se adquirió un Access Point de Compaq. Este permite comunicar la red inalámbrica formada por las tarjetas PCMCIA con la red LAN del laboratorio.

Las pruebas realizadas demuestran que si bien la distancia operativa es aceptable, la velocidad de transmisión no alcanza los niveles esperados.

La distancia máxima a la que se ha podido mantener una comunicación entre la tarjeta y el access point es de aproximadamente 50 metros, a través de los gruesos muros exteriores del laboratorio.

Respecto a la velocidad de transmisión, si bien la máxima especificada es de 11 Mbps, en laboratorio y bajo condiciones idóneas, solo hemos obtenido una velocidad de 5 Mbps, con picos de 8 Mbps.

Una de las principales razones de la elección de esta tarjeta ha sido la garantía de que existen drivers para Linux, sistema operativo que será empleado en el Wafer 5820.

Figura 7.1: Compaq PC-Card

Figura 7.2: Compaq Access Point

Capítulo 8

Procesadores

Requisitos:

- RG1:El host será un computador con una potencia suficiente para recibir y procesar toda la información recibida del robot, y ordenar una serie de acciones encaminadas a lograr el objetivo deseado.
- RG3: El robot debe tener varios procesadores heterogéneos.
- RG3.1: El procesamiento principal se llevará a cabo en el Wafer 5820, bajo sistema operativo Linux en tiempo real, mientras que un procesador JStamp será el encargado de manejar los sistemas de navegación (motores y sensores).

El procesador principal, encargado del razonamiento y la comunicación con el host, es una placa Wafer 5820, con procesador GXLV 233 MHz, 128Mb SDRAM.

El esta diseñado para aplicaciones en espacios reducidos, con un tamaño estándar de 3 pulgadas y media (el tamaño de un disco duro). Ofrece todas las funciones de un computador AT industrial en una sola placa. Esta equipado con un procesador GXLV/GX1 de bajo consumo, compatible Pentium II, lo que resulta ideal para una aplicación en la que la fuente de alimentación es un paquete de baterías. No requiere ventilador.

El Wafer dispone de un bus PC-104. Este bus es similar al standard de un PC de sobremesa, pero tiene un tamaño mucho mas reducido que un bus PCI normal, lo que permite su utilización en aplicaciones empotradas donde el espacio es un requerimiento básico. Otra de sus características es que permite la conexión en vertical de distintos módulos, sin necesidad de emplear ningún tipo de cable, lo que permite añadir nuevos módulos a la placa base sin problemas, como por ejemplo CPU´s, puertos de entrada salida, controladores de vídeo, etc.

El procesador secundario, encargado de las tareas mas "mecánicas" (¿asimilable a la función del cerebelo en el cuerpo humano?[1]) es el nuevo procesador

[1]El cerebelo desempeña un papel regulador en la coordinación de la actividad muscular, el mantenimiento del tono muscular y la conservación del equilibrio. Precisa estar informado constantemente de lo que se debe hacer para coordinar la actividad muscular *(movimiento)*de manera satisfactoria. A tal fin recibe información procedente de las diferentes partes del organismo. Por un lado, la corteza cerebral *(procesador principal)* le envía una serie de fibras *(cable serie)* que posibilitan la cooperación entre ambas estructuras. Por otro lado, recibe información procedente de los músculos y articulaciones*(sensores)*, que le señalan de modo continuo su posición.

Figura 8.1: JStamp

JStamp, cuya principal característica es tener Java como lenguaje nativo. En un principio se consideró la idea de usar un microcontrolador ëstándarcomo el Motorola 68HC11, pero el JStamp dispone de una capacidad de procesamiento y memoria mucho mayor.

8.1. Programación del JStamp

La tarea principal del JStamp es el control de los motores, servos y sensores del robot. Recibe las órdenes del Wafer por via puerto serie, a la vez que le comunica el estado de los sensores.

Se ha creado una programación basada en multithreading, que permite:

- un uso mas efectivo de la CPU
- ejecutar varias tareas al mismo tiempo, como por ejemplo controlar los motores a la vez que transmite información.
- el fallo en uno de los threads no impide que el resto de funciones siga ejecutándose.

Las principales clases son:

- SerialComms: esta clase se encarga de la comunicación serie con el Wafer. Inicializa el puerto serie del JStamp (initializeSerialPort), transmite (sendMessage) y recibe (receive)información.
- ServoControl: este thread controla el servo motor por medio de PWM. El método setReloadRegisterValue() especifica el período de la señal.

Capítulo 9

Motores

Se usan dos canales, uno para la dirección y otro para los motores.

Para la dirección se usa un servomotor , que actúa simultáneamente sobre los dos ejes del robot, proporcionando una buena capacidad de maniobra.

Especificaciones:

Torque:

4.8VDC: 88.8oz-in. (6.37 kg-cm)

6.0VDC: 111oz-in. (7.96 kg-cm)

Speed @ 60 Degrees:

4.8VDC: 0.11 seconds

6.0VDC: 0.10 seconds

Bearing Type:

Dual Ball Bearing

Case Size:

1.59"x 0.79"x 1.48"(40.3 x 20 x 37.5 mm)

Weight: 1.94oz (55 g)

Wire Length: 11.25"(Including plug)

Para la tracción se emplean dos motores de corriente continua, de 12V de tensión nominal. Ambos dan potencia al mismo eje, con lo que se pueden considerar mecánicamente unidos.

Los motores usados en el robot son de corriente continua. Para hacer que se muevan a una velocidad determinada hay que suministrarles una tension media de alimentación adecuada, que puede ser de hasta 12V. Un procesador no puede dar tanta tensión, por lo que se necesita un componente intermedio entre el procesador y el motor. Este componente es un ESC (*Electronic Speed Controller*) que se ocupa de controlar la cantidad de energía que pasa de las baterías a los motores, de acuerdo con las órdenes que recibe del procesador. Esto se logra çortando"(*chopping*)el voltaje que pasa de las baterías a los motores. Cuantas mas veces por segundo se realize este çorte", mas fluido será el movimiento del motor.

Para la elección del ESC hay que tener en cuenta que a bajas velocidades está sometido a una gran carga. Al 100 % de velocidad el controlador se limita a dejar pasar toda la potencia de las baterías al motor. Pero a bajas velocidades debe reducir la cantidad de corriente que llega al motor, lo que genera una gran cantidad de calor en el ESC (de aquí el gran tamaño de los disipadores de todos los ESC de cierta potencia.)

El ESC recibe una señal (*Pulse Width Modulation*, o modulación por ancho del pulso) del JStamp. El JStamp debe ocuparse de generar al menos dos PWM, así como escuchar a los distintos sensores. Para que pueda generar estas señales y al mismo tiempo estar continuamente atento a los sensores, se ha realizado una programación por medio de "threads.º hilos, que permite que varias tareas se ejecuten permanentemente al mismo tiempo. También permite que el programa continue en el caso de que alguna de las tareas (por ejemplo, debido al mal funcionamiento de alguno de los sensores).

Figura 9.1: Servo Futaba S9450

Figura 9.2: ESC

Capítulo 10

Potencia

Listado de consumos:

- procesadores
  - Wafer 5820: 5V@2A 12V@0.1A
  - JStamp:5V@100mA
- sensores
- cámaras: 12v@200mA (max.)
- comunicación inalámbrica: Wireless Ethernet PC Card (Orinoco):5V@280ma (max.)

La primera consideración en relación con la alimentación de potencia, es separar las alimentaciones para la electrónica y para el movimiento. Motores y servomotores introducen ruidos en la alimentación, además de requerir grandes picos de corriente intermitentes que podrían afectar a los componentes electrónicos. Por ello se usan dos paquetes de baterías: las baterías originales del modelo se usarán para los motores, mientras que otro paquete comprado a posteriori dará potencia a la electrónica del robot.

Para que el robot sea autónomo estamos limitados a escoger como fuente de potencia baterías recargables. Se han escogido baterías recargables Ni-Cd de alta capacidad.

Se trata de pilas cilíndricas de níquel cadmio de alta capacidad, con terminales de soldadura en la parte superior e inferior para su montaje en lotes de pilas.

- Altas capacidades para tiempos de descarga ampliados Capacidad de realizar más de 700 ciclos completos de carga/descarga Altas corrientes de descarga continuas Baja resistencia interna
- Estas pilas se pueden usar para producir lotes de baterías de alta potencia y poco peso, ideales para equipos portátiles.
- Para cargar, hay que utilizar un cargador de corriente constante.

El paquete se realizó con 12 pilas en serie, lo que proporciona una tensión de salida de 15V, adecuada como entrada del convertidor de tensión.

Dado que se prevé un consumo considerable por parte, principalmente, de los dos procesadores, módulo de comunicación y cámaras de video, se han preparado dos paquetes de baterías iguales, para poder realizar cambios rápidos in situ que permitan un mayor tiempo de uso del robot.

Para transformar la potencia de las baterías en tensiones utilizables se usa un convertidor CC/CC de 30W, con salida triple de +12V / -12V / +5V

Estas salidas son llevadas a una placa de conexión con conectores para:

- ESC
- servo
- Wafer
- JStamp

Figura 10.1: Pilas

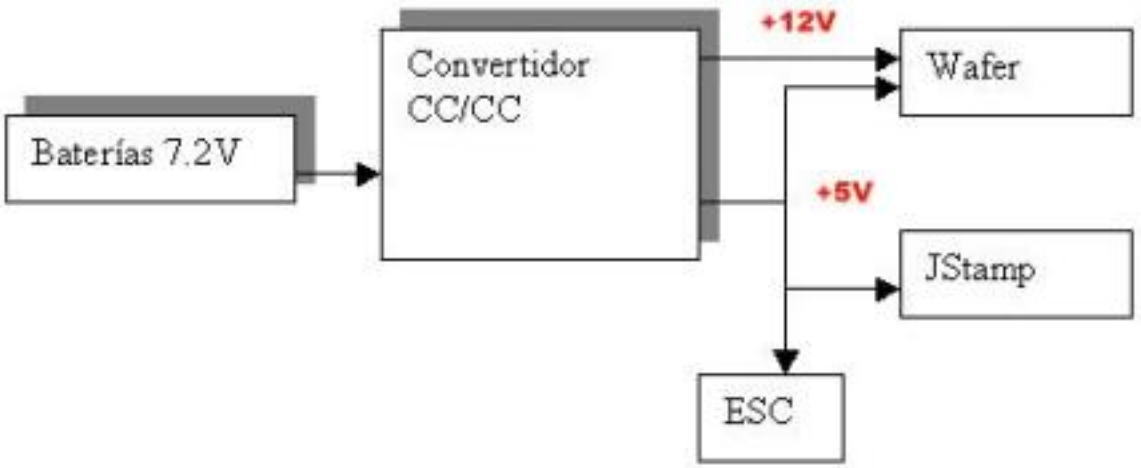


Figura 10.2: Elementos de potencia

Capítulo 11

Conexionado

Conexiones del JStamp:

- puerto IOE6: salida de control para el servo de dirección. El puerto de entrada/salida E6 sirve como salida del PWM
- puerto IOE7: salida de control para el ESC. El puerto de entrada/salida E7 sirve como salida del PWM
- pin 40: el JStamp admite una entrada de voltaje no regulado de entre 5-14 VDC. Tambien puede alimentarse con una entrada de tensión regulada de 3.3V por su pin 1. Mucho cuidado de no alimentar las dos entradas a la vez, pues esto podría causar daños irreparables.

11.1. Interconexión

Se ha usado una placa PCB para dos fines: instalar el convertidor de tensión y para poder conectar las salidas de los pines del JStamp de una manera sencilla. Esta placa se encuentra en el interior de la caja que contiene los procesadores.

Por una parte se encuentra el convertidor CC/CC. A su entrada se conecta el paquete de baterías de 15V, y a su salida obtenemos +5V, tierra y +12V. Estas salidas son luego llevadas a la entrada de ambos procesadores, así como a los conectores de 3 pines del servo y el motor.

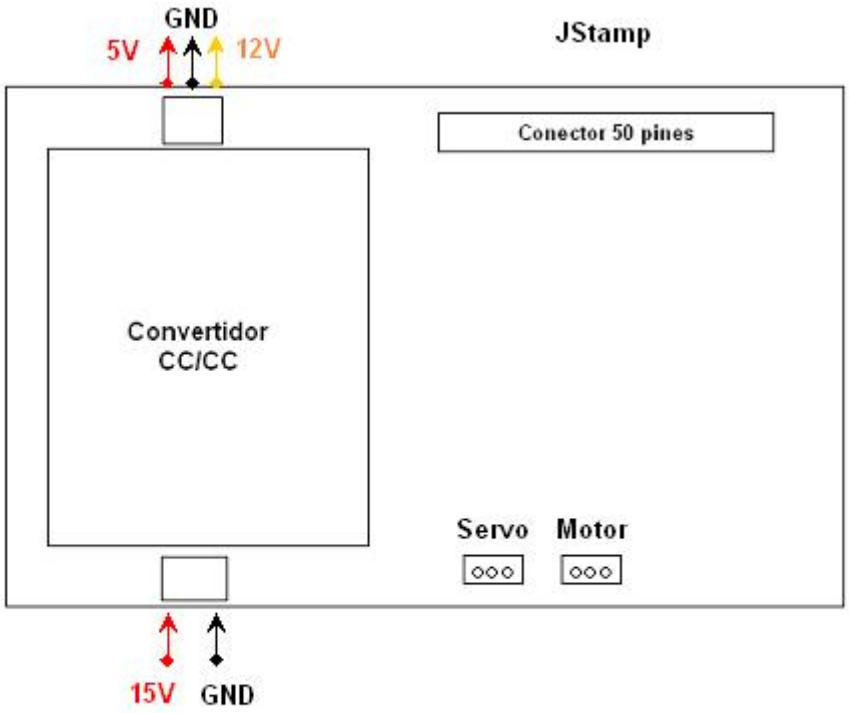


Figura 11.1: Placa de conexión

Un cable plano de 50 pines es el encargado de llevar la salida de los pines del JStamp a la placa de interconexión, de forma que sea mas sencilla su utilización.

Parte III

Apéndices

Apéndice A

Análisis de Requisitos

- RG1- El sistema se compondrá de un host estático y de un robot móvil, comunicados inalámbricamente. La tarea del host será el procesamiento de la información recibida por los sensores del robot y la toma de decisiones estratégicas.
  - RG1.1-El host será un computador con una potencia suficiente para recibir y procesar toda la información recibida d el robot, y ordenar una serie de acciones
  - RG1.2- La comunicación inalámbrica será tal que permita la transmisión de imágenes de video desde el robot, así como la de los diversos sensores.
- RG2- CORBA será la tecnología elegida para disponer de una interfaz de programación independiente de las plataformas usadas.
- RG3- El robot debe tener varios procesadores heterogéneos.
  - RG3.1- El procesamiento principal se llevará a cabo en el Wafer 5820, bajo sistema operativo Linux en tiempo real, mientras que un procesador JStamp será el encargado de manejar los sistemas de navegación (motores y sensores).
- RG4- El robot debe disponer de cámaras de video, cuya información será transmitida al host para su procesamiento.
  - RG4.1- El propósito de las cámaras será, en principio, el de teleoperar el robot desde el host, así como de poner a prueba las comunicaciones inalámbricas
- RG5- El robot debe ser teleoperable desde el host.
  - RG5.1- Se debe crear una interfaz de usuario bajo S.O. Windows que permita el control del robot por el usuario así como un acceso a sus subsistemas y sensores.
- RG6- Si se produce un fallo en las comunicaciones host-robot, este último debe contar con autonomía suficiente

Apéndice B

Variadores de velocidad y PWM

Objetivo: controlar los motores y servomotores del vehículo con un microprocesador.

Problema: la corriente que un procesador (tal como el Wafer 5820 o el JStamp) es del orden de los 24 mA, mientras que un motor de corriente continua de los habitualmente usados en radio control necesita tensiones de entre 6 - 12 V e intensidades que en el arranque pueden ser de hasta 2A.

Los variadores de velocidad reciben una señal análoga a la que recibe un servo. Si la duración del pulso es de 1.5 ms, el variador ordena al motor que permanezca parado. Un pulso mas largo hará que el motor gire en una dirección, mientras que un pulso mas corto hará que gire en dirección opuesta. De esta manera podemos controlar tanto la velocidad como la dirección de giro con solo tres cables.

Internamente un ESC realiza las siguientes operaciones:

- mide el ancho del pulso recibido.
- traduce estos pulsos en una señal analógica de pequeño voltaje.
- abre y cierra la alimentación del motor muy rápidamente.

Existen dos tipos de ESC en el mercado: los mas sencillos solo permiten el giro del motor en un sentido, mientras que los segundos permiten cambiar el sentido de giro. Evidentemente nosotros necesitamos un ESC reversible que permita al robot moverse libremente por entornos altamente variables, donde puede ser necesario cambios bruscos de dirección .

Figura B.1: ESC

REVERSIBLE MODELS		REACTOR REVERSIBLE		610-HRV
SPEC / FEATURE	**SUPER ROOSTER**	**REACTOR**	**ROOSTER**	**610-HRV**
Part Number	#1860	#1810	#1850	#1800
List Price	$220.00	$173.00	$139.00	discontinued
Input Voltage (cells)	6 to 10	6 & 7	6 & 7	6 to 10
Case Size (in)	1.63 x 2.02 x 1.22	1.63 x 1.42 x 0.69	1.63 x 2.02 x 1.22	1.63 x 1.42 x 0.6
Case Size (cm)	4.14 x 5.13 x 3.10	4.14 x 3.61 x 1.75	4.14 x 5.13 x 3.10	4.14 x 3.61 x 1.7
Weight (ounces/grams)	4.0 / 113.4	1.81 / 51.3	3.0 / 85.0	1.81 / 51.3
On-Resistance* (ohms)	0.0020	0.0040	0.0180	0.0065
Transistor Type	HYPERFET III	HYPERFET III	MEGAFET	HYPERFET II
Motor Limit	No Limit (single/dual motors in series); 15 turns (dual motors in parallel)	12 turns (at 6 cells)	15 turns (at 6 cells)	12 turns (at 6 cells)

One-Touch Set-Up	Yes	Yes	Yes	No
Drive Frequency (Hz)	1250	1000	1250	2500
Brake Frequency (Hz)	1250	1000	1250	2500
Discrete Steps	64: 32 Forward, 32 Reverse	512: 256 Forward, 256 Reverse	64: 32 Forward, 32 Reverse	128: 64 Forward, 64 Reverse
Rated Fwd. Current* (Amps)	320	160	100	120
Rated Rev. Current* (Amps)	160	80	100	60
Braking Current* (Amps)	160	80	100	60
B.E.C. (volts / amps)	6.0 / 3.0	5.0 / 0.5	5.7 / 0.5	5.0 / 0.5
Wire Size (gauge)	14	16	16	16
Adjustment Knob(s)	None	None	None	Neutral, High Speed, Reverse Delay
Polar Drive Circuitry	Yes	Yes	Yes	No
Radio Priority Circuitry	Yes	Yes	Yes	No
Digital Anti-Glitch Circuitry	Yes	Yes	Yes	No
Reverse Voltage Protection	No	No	No	No
Thermal Protection	Yes: Dual-Level	Yes	Yes: Dual-Level	Yes

Reverse Disable	Yes	Yes	Yes	No
Smart Braking Circuitry	Yes	Yes	Yes	No
Heat Sinks	Included	Included	Factory Installed	Included
Brake Light Circuitry	Yes	No	Yes	No
Brake Light Kit	Optional	N/A	Optional	N/A
Battery Plug Installed	None	Tamiya	Tamiya	Tamiya
Motor Plug Installed	None	Bullet-Style	Bullet-Style	Bullet-Style
REVERSIBLE MODELS	SUPER ROOSTER	REACTOR	ROOSTER	610-HRV

*Transistor's rating at 25 degrees Celsius junction temperature

Figura B.2: Especificaciones del ESC

Apéndice C

Wafer 5820

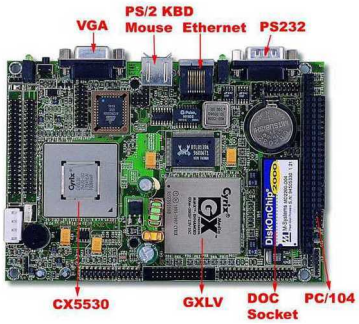


Figura C.1: Wafer 5820

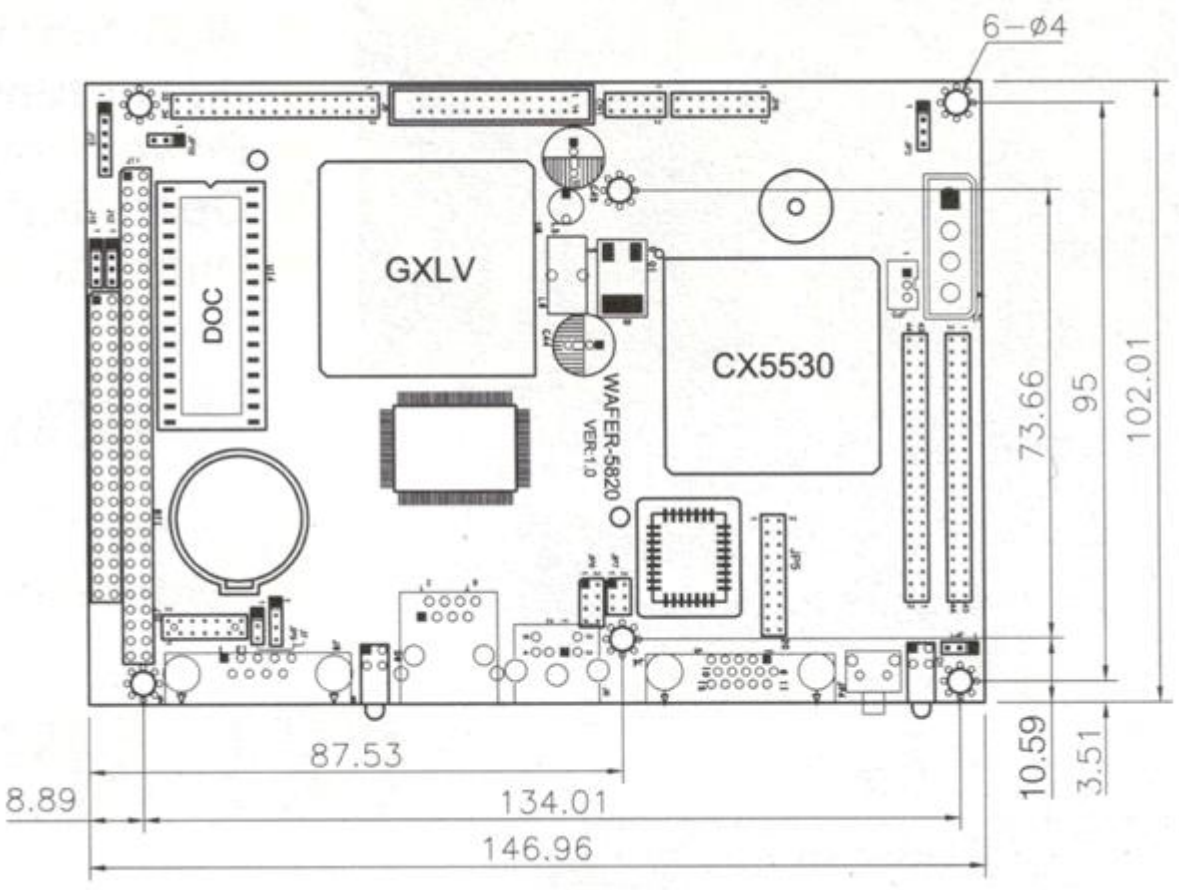


Figura C.2: Dimensiones del Wafer 5820

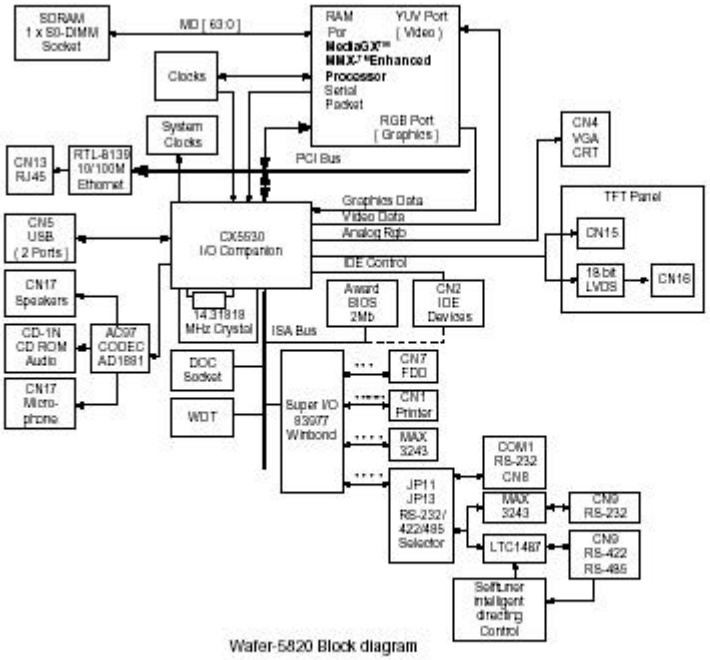


Figura C.3: Diagrama de bloques

Apéndice D

JStamp

JStamp es un procesador con una característica distintiva: Java es el nivel nativo. Esto significa que no existe un interprete entre el código Java escrito para el y el silicio.

El AJ-80 soporta la ejecución de múltiples aplicaciones a través de "múltiples JVM´s". Cada aplicación se ejecuta de manera determinística , en su propio hilo de ejecución y con sus propios "event handlers".

What is JStamp? In the simplest terms, JStamp is a computer. Like most other computers, it has a CPU, RAM, ROM, power supply, and I/O. Aside from that, JStamp is unlike most other computers. The most obvious difference is its size (1 inch by 2 inches), but perhaps its most significant difference is that it is programmed entirely in Java. This is not to be confused with systems that merely restrict one to programming in Java, yet execute proprietary opcodes at the native level. On JStamp, Java is the native level. That point bears repeating: On JStamp, Java is the native level.

The fact that Java is the native level makes the JStamp very fast; there is no Java interpreter layer between your Java code and the silicon. It allows JStamp to be very small; there is no need for extra memory to store a Java interpreter. It means that JStamp runs real Java; the JStamp runs actual Java byte codes. There is no compile-time, build-time, runtime, or at-any-other-time translation of Java byte codes into proprietary opcodes. This leads to the slogan that JStamp is real fast, real small, real Java.

- CPU The CPU on JStamp is the aJ-80 from aJile Systems. The aJ-80 is programmed entirely in Java. The standard Java Virtual Machine byte codes are its native instruction set. The aJ-80 is rated for a maximum operating frequency of 80 MHz. It is possible to configure JStamp to run at many clock speeds; as slow as 7.372 MHz or far faster than the aJ-80's maximum rated frequency of 80 MHz. Due to the 7.372 MHz crystal used on JStamp, the highest in-spec clock frequency is 73.72 MHz. JStamp's power use is proportional to its speed; the slower you run it, the less power is uses. This is very useful for battery powered systems.

  The aJ-80 contains a number of built-in I/O peripherals. These include two UARTs, a serial peripheral interface (SPI) controller, three very versatile 16-bit timer/counters, as well as general-purpose I/O ports.

  The aJ-80 also includes an industry standard test interface known as a JTAG interface. The JTAG interface is used for loading and debugging

programs on JStamp. It is both fast and powerful, providing the ability to debug your program while it is running in-circuit. Effectively, it makes the JStamp its own in circuit emulator. (Believe it or not, the 'J' in JTAG does not stand for Java!)

- RAM The JStamp contains 512 Kbytes of static RAM. This RAM is not battery backed and its contents are lost when power is removed. Because of this, programs are not normally stored in RAM except during the development process.

- ROM The JStamp contains 512 Kbytes of flash ROM. A different version of JStamp, called the JStamp-Plus, contains 2 MB of flash ROM. The extra flash ROM is the only difference between the JStamp and the JStamp-Plus . For normal deployment, programs are typically stored in flash ROM. The flash ROM is also used to store a local file-system, if any.

- Power Supply The power supply and JStamp is a switching voltage regulator. The raw supply voltage to JStamp can be between 5 and 14 volts unregulated DC. The switching voltage regulator generates 3.3 volts (regulated) for the on-board circuitry and provides up to 100 milliamps of regulated 3.3 volts DC for use by off-module circuitry.

- I/O Almost all of JStamp's 40 pins are used for I/O. The I/O pins are connected to the aJ-80's built-in I/O peripherals. The JStamp module also contains an LED. On the first version of JStamp, this LED was simply a power indicator. On subsequent versions, this LED can be controlled via software, but defaults to being a power indicator.

JStamp™

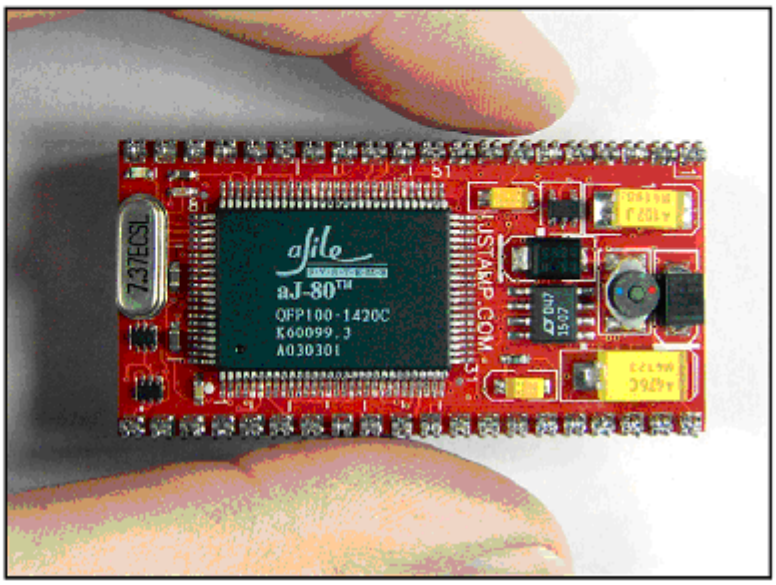
SYSTRONIX®

Real-Time Native Java™ Module

Have your cake and eat it too -- all the benefits of Java -- with FAST native execution and deterministic real-time capability! This is the Java processor you've been waiting for, and it's available *now*.

At last you can stop *dreaming* about embedded Java and start *using* it.

Here, JStamp is shown 2X larger than its actual size of 1.0 x 2.0 inches.



JStamp - the *power* of Java plus the *speed* of native execution.

JStamp - small, low power, low cost, and easy to use.

JStamp - J2ME CLDC plus RTSJ support.

Imagine what a native Java embedded control system with real time capability could do for you -- all the benefits of Java technology with native execution speed, plus Real Time Java support!

JStamp includes 512 KBytes of SRAM and 512 KBytes of Flash, power converter, dual UARTs, SPI, IrDA support, JTAG programming/debug port, heartbeat LED and more.

JStamp can run off a 9V battery for over 24 hours!

JStamp is under $100 in quantity. Development systems are $300.

- aJile aJ-80™ Real-Time Java Processor
- 512 KBytes SRAM, 512 KBytes Flash (JSTamp+ has 2 MBytes Flash). (8-bit wide memory data path).
- Executes Java code natively with a 32-bit internal architecture.
- Dual UARTs (TTL output level), includes IrDA support.
- SPI interface for easy I/O expansion
- Low power 3.3V system, accepts 5-14 VDC and provides 100 mA of 3.3V for your use off-module.
- Power and heartbeat LED
- JStamp development kits include a JTAG pod and software, JStamp, JSimm proto board, and a JStamp development station board. (Software requires Windows 98/NT/2000 and a PC parallel port).

WWW.JSTAMP.COM
FOR LATEST INFORMATION
AND SECURE ON-LINE ORDERING

Native Java Execution with Real Time Java Support!

JStamp uses the powerful new aJ-80 native Java processor from aJile Systems (www.ajile.com). This provides fast and efficient Java instruction execution, plus a small memory footprint - 2 to 3 times denser than code for 32-bit RISC machines.

The aJ-80 includes a microprogrammed real-time Java thread manager so that a typical RTOS is not needed. It includes Java threading primitives implemented as atomic instructions, a priority-based preemptive scheduler, extremely fast context switching and interrupt response.

The aJ-80 supports multiple application execution through "multiple JVMs". Applications execute in a deterministic, time-sliced schedule. Each application has its own thread management and event handlers.

JStamp Makes Embedded Java Hardware Easy

JStamp includes the aJ-80, memory, crystal, power converter, reset logic, and all other necessary circuitry. The high-speed system memory bus is isolated from JStamp I/O pins. All you have to provide is power and generic integrated circuit socket strips, 20 contacts, 100-mil centers, 1.00 inches wide. JStamp is similar to a standard DIP40 IC, only 0.400 inches wider. JStamp plugs into standard solderless breadboards and any solder type breadboard with an array of holes on 100-mil centers. If you prefer, DIP40 sockets 1.00 inches wide will be available from Systronix in prototype and production quantities.

All the fine-pitch, high speed circuit layout is done for you. The external I/Os of JStamp have no difficult design considerations. Just plug in JStamp and go!

Timers and Counters

The aJ-80 has multiple timers and counters, including PWM output.

Memory

JStamp is ready for serious work with 512 KBytes of SRAM and 512 KBytes of Flash (2 MBytes on JStamp+). Code can be executed from flash or SRAM.

User I/O Pins

JStamp has five 24- mA I/O pins plus seventeen 8-mA pins (some of which also serve the UART and SPI functions).

I/O Expansion & Networking

JStamp includes a SPI interface for easy peripheral and I/O expansion. The JStamp development station adds hardware support for RS-232 serial I/O, and - very soon- plug-on JSimm modules for Dallas 1-Wire network, CAN network, IrDA, RF, graphic LCD, analog I/O, power relays, and more. OF course, many current simmstick modules are JSimm compatible.

Easy JTAG Programming and Debugging

JStamp uses a JTAG interface for programming and debugging with aJile's JEM Builder and Charade development tools. JTAG pods are available from Systronix and Xilinx. Software requires Windows 98/NT/2000 and a PC parallel port.

Development Tools and Examples

The JStamp development system includes JStamp, JStamp development station board, JSimm prototyping board, aJile development tools, JTAG programming adapter, and example programs. Note: you must purchase at least one JStamp development system in order to program and use JStamp. Thereafter, you can purchase additional JStamps.

How do I order?

You can order in our secure on-line store at www.jstamp.com. The JStamp web site will always have the most current product information.

SYSTRONIX®

555 South 300 East #21, Salt Lake City, Utah, USA 84111
Tel:+1-801-534-1017 Fax:+1-801-534-1019 www.systronix.com

TECHNICAL DETAILS

Processor aJ-80, 32-bit internal core, ALU, and memory. Direct JVM bytecode execution - no interpreter or JIT compiler.

Memory 8-bit wide data path to 512 KBytes of 70 nsec SRAM and 512 KBytes of 90 nsec flash (2 MBytes in JStamp+).

Power Unregulated 5(4.75V min)-14 VDC. Current requirement is 30 to 100 mA @ 5 volts (min to max), depending on the processor speed, which is adjustable in firmware.

Efficient switching regulator powers the JStamp and provides 3.3V @ 100 mA (max) for use off-JStamp. Recommended power source is the Systronix 6VDC 1A power cube, or a 5 VDC, 5% regulated supply from the host socket. Operates for 24 hours off a standard 9V alkaline battery.

Serial I/O Dual UARTs similar to 16550, with IrDA support. TTL levels at the JStamp pins. If not used as serial I/O, these are generic I/O pins.

SPI MISO, MOSI, CLK and three chip selects.

Easy Programming Instructions and tutorials on line at www.jstamp.com.

Size 1.00 x 2.00 inches with 40 leads on 0.100 inch centers. Leads are .010 x .020 inches and fit standard leaded IC socket strips. 1.00 x 2.00 inch DIP40 JStamp sockets will be available from Systronix.

Environmental Commercial temperature 0 to 70 deg C.

Support & Warranty Friendly technical support. One year warranty against defects (processor is warranteed separately by aJile Systems).

PRELIMINARY DATA
SUBJECT TO CHANGE WITHOUT NOTICE

PLEASE NOTE THAT INITIAL JSTAMP PRODUCTION SCHEDULE AND PRICING MAY BE SUBJECT TO COMPONENT AVAILABILITY AND MARKET CONDITIONS

Prices:

Visit www.jstamp.com for current prices and special offers. JStamp is under $100 in moderate quantities. JStamp+ (2 MBytes flash) is about $150. Special offers and development bundles are available. Options include DIN rail mounting, JSimm modules, cables and enclosures.

WWW.JSTAMP.COM
FOR INFORMATION & ORDERS

revised 2001 Apr 06 bdb

Apéndice E

Creación de programas para el JStamp

Para este proyecto he elegido como entorno de programación Java el JBuilder 5, aunque se puede usar cualquier otro paquete de programación comercial.

1. Crear y compilar el programa con JBuilder
   - Crear proyecto (.jpx o .jpr)
   - En "Propiedades de proyecto -¿Vias de acceso -¿Bibliotecas necesarias.ªñadir aJile CLDC y J2ME CLDC
   - Compilar
2. Usar JEMBuilder para tomar las clases compiladas y crear el byte-code para el JStamp.
   - Crear proyecto
   - Elegir directorio donde se creará
   - Elegir configuración, dependiendo si queremos trabajar con la RAM o grabar en flash. (recordar poner o quitar el jumper JP1 en la placa de desarrollo)
   - Nuevo JVM: por defecto JVM0
   - Nombre de la clase que contiene el "main": por ejemplo "blink"
   - Classpath: por ejemplo .../blink/class
   - Available drivers: elegir los necesarios
3. Usar Charade para bajar el programa al JStamp.

   Cuidado al elegir el "device"; el apropiado es .ªj80 (port 378)çuando el Jtag esta conectado en el puerto COM1 del PC.

Recordar que el jumper JP1 en la JStamp Development Station selecciona donde se almacenará el programa. Si el jumper esta instalado se cargará en la memoria RAM, y por tanto se perderá cuando se desconecte la alimentación. Si el jumper no esta instalado el programa se cargará en memoria FLASH.

Apéndice F

Wireless Ethernet

SPECIFICATIONS	
Frequency Band	2400-2483.5 MHz
Number of Selectable Sub-channels	Worldwide Certified (FCC/ETS/JP/FR), 11 channels
Modulation Technique	Direct Sequence Spread Spectrum (CCK, DQPSK, DBPSK)
Spreading	11-chip Barker sequence
Bit Error Rate	Better than 10^{-5}
Media Access Protocol	CSMA/CA (Collision Avoidance) with ACK
Interface	PC Card, PCI
Size / Viewable Image Size (diagonal)	15 in / 13.8 in
OS Support	Windows 95/98/ME/NT4 (SP4)/2000/Windows CE/Pocket PC
Dimensions	117.8 mm x 53.95 mm x 8.7 mm (PC card)
Speed Options	11, 5.5, 2 and 1 Mbps; Automatic Rate Selection (ARS)
Output Power	15dBm
Power Consumption (5 volt power supply)	Doze mode - 9mA Receive mode - 185 mA Transmit mode - 285 mA
Temperature Range (operational)	0-55° C 95% max. Humidity (non condensing)
Standards	IEEE 802.11b
Regulations	US: FCC (47 CFR) Part 15C, Section 15.247 Canada: ISC RSS139 Europe/APA: ETS 300-328, CE Marked Japan: MPT Radio Regulations

Speed Options	11 Mbps	5.5 Mbps	2 Mbps	1 Mbps
Range in meters(feet) **Open Office**	160 m (525ft)	270 m (885 ft)	400 m (1300 ft)	550 m (1750 ft)
Semi Open Office	50 m (165 ft)	70 m (230 ft)	90 m (300 ft)	115 m (375 ft)
Closed Office	25 m (80 ft)	35 m (115 ft)	40 m (130 ft)	50 m (165 ft)
Receiver Sensitivity	-82	-87	-91	-94
Delay Spread (at FER of < 1%)	65ns	225ns	400ns	500ns

SPECIFICATIONS	
Features	Integrated 11 Mbps radio Supports -- Power Over Ethernet 128 bit key security using RC4 encryption 10 Mbps Ethernet IEEE 802.11b (Wi-Fi) compliant Spanning Tree Algorithm IEEE 802.1D Transparent Bridging Selective protocol filtering Access Control Table and Radius Authentication DHCP and BOOTP Multi-channel support Roaming support
Management	Compaq AP Manager Software SNMP MIB ii, 802.3, 802.1D, and 802.11 MIB compliant Windows based user interface TRAPS: power up, authentication, link up/down Site Survey Tools Remote Link Test
LEDs	Power Ethernet LAN Activity Wireless LAN Activity
Interface	Ethernet 802.3 -10Vase-T (RJ 45 Connector)
Dimensions	145 mm x 175 mm x 70 mm (includes table mount bracket) 130 mm x 175 mm x 45 mm (includes wall mount bracket)
Weight	0.50 kg
Power Supply	External wall plug-in unit: Auto sensing 100/240 VAC 47-63 Hz, 9v/1.1 Ready for - Power Over Ethernet
Temperature Range (operational)	0° to +40° C Operating humidity max 90% (no condensing allowed)

Speed Options	**11 Mbps**	**5.5 Mbps**	**2 Mbps**	**1 Mbps**
Range in meters(feet) **Open Office**	160 m (525ft)	270 m (885 ft)	400 m (1300 ft)	550 m (1750 ft)
Semi Open Office	50 m (165 ft)	70 m (230 ft)	90 m (300 ft)	115 m (375 ft)
Closed Office	25 m (80 ft)	35 m (115 ft)	40 m (130 ft)	50 m (165 ft)
Receiver Sensitivity	-82	-87	-91	-94
Delay Spread (at FER of < 1%)	65ns	225ns	400ns	500ns

QuickSpecs

Compaq WL110 Wireless PC Card

MODELS

WL110 Wireless PC Card

191808-B21

OVERVIEW

The Compaq WL110 Wireless PC Card can be used worldwide for the Enterprise, ISP, Public Access or SMB markets using any of the access point infrastructure products. The card provides high-speed wireless networking and secure access to network resources and the Internet. The WL110 PC card is for use in notebooks, desktops and the iPAQ Pocket PC. Benefits include:

- The same high-performance connectivity as wired systems, with the added freedom to roam around your building or campus
- Enhanced security with 128-bit Wired Equivalent Privacy (WEP) encryption
- WiFi certified and compatible with IEEE 802.11b compliant PC Cards
- Compaq Client Manager Software

COMPATIBILITY

Compaq Notebook

iPAQ Pocket PC

SERVICE AND SUPPORT

WL110 Wireless PC Card has a three-year limited warranty or the remainder of the warranty of the Compaq product in which it is installed. Technical support is available seven days a week, 24 hours a day, by phone as well as online support forms. Certain restrictions and exclusions apply.

SPECIFICATIONS

Frequency Band	2400-2483.5 MHz
Number of Selectable Sub-channels	Worldwide Certified (FCC/ETSI/JP/FR), 11 channels
Modulation Technique	Direct Sequence Spread Spectrum (CCK, DQPSK, DBPSK)
Spreading	11-chip Barker sequence
Bit Error Rate	Better than 10^{-5}
Media Access Protocol	CSMA/CA (Collision Avoidance) with ACK
Interface	PC Card, PCI
OS Support	Windows 95/98/MEINT4 (SP4)/2000; Windows CE/Pocket PC
Dimensions	4.6 in x 2.12 in x 0.34 in/117.8 mm x 53.95 mm x 8.7 mm (PC card)
Speed Options	11, 5.5, 2 and 1 Mbit; Automatic Rate Selection (ARS)
Output Power	15dBm
Power Consumption (5 volt power supply)	Doze mode – 9mA Receive mode – 185 mA Transmit mode – 285 mA
Temperature Range (operational)	32° to 130° F/0° to 55° C 95% maximum Humidity (non condensing)
Standards	IEEE 802.11b
Regulations	US: FCC (47 CFR) Part 15C, Section 15.247 Canada: ISC RSS139 Europe/APA: ETS 300-328, CE Marked Japan: MPT Radio Regulations

QuickSpecs Compaq WL110 Wireless PC Card

SPECIFICATIONS *(continued)*

Speed Options	11 Mb/s	5.5 Mb/s	2 Mb/s	1 Mb/s
Range in meters (feet) **Open Office**	160 m (525 ft)	270 m (885 ft)	400 m (1300 ft)	550 m (1750 ft)
Semi Open Office	50 m (165 ft)	70 m (230 ft)	90 m (300 ft)	115 m (375 ft)
Closed Office	25 m (80 ft)	35 m (115 ft)	40 m (130 ft)	50 m (165 ft)
Receiver Sensitivity	-82	-87	-91	-94
Delay Spread (at FER of < 8%)	65ns	225ns	400ns	500ns

Compaq PCs use genuine Microsoft® Windows®

http://www.microsoft.com/piracy/howtotell

Microsoft is a trademark of Microsoft Corporation in the U.S. and other countries.

QuickSpecs

Compaq WL410 Wireless SMB Access Point

MODELS

WL410 Wireless SMB Access Point

191811-001

OVERVIEW

The Compaq WL410 is a full featured wireless access point that acts as a bridge between the wired Ethernet backbone and your wireless clients. The small form factor of the WL410 Access Point makes it easy to locate either on a flat surface or easily mounted on a wall. Benefits of the WL410:

- Secure for placement in public areas with a Kensington locking system
- Comes with an integrated antenna, but also has a jack for adding an external antenna for enhanced performance
- Supports both 110/240 VAC power from local AC outlet or Power Over Ethernet
- Enhanced security with 128-bit Wired Equivalent Privacy (WEP) encryption

COMPATIBILITY

Compaq Desktop

Compaq Notebook

iPAQ Pocket PC

SERVICE AND SUPPORT

WL410 Wireless SMB Access Point has a one-year limited warranty. Technical support is available seven days a week, 24 hours a day, by phone as well as online support forms. Certain restrictions and exclusions apply.

SPECIFICATIONS

Features	Integrated 11 Mb/s radio
	Supports Power Over Ethernet
	128 bit key security using RC4 encryption
	10 Mb/s Ethernet
	IEEE 802.11b (Wi-Fi) compliant
	Spanning Tree Algorithm
	IEEE 802.1D Transparent Bridging
	Selective protocol filtering
	Access Control Table and Radius Authentication
	DHCP and BOOTP
	Multi-channel support
	Roaming support
Management	Compaq AP Manager Software
	SNMP MIB ii, 802.3, 802.1D, and 802.11 MIB compliant
	Windows based user interface
	TRAPS: power up, authentication, link up/down
	Site Survey Tools
	Remote Link Test

QuickSpecs

Compaq WL410 Wireless SMB Access Point

SPECIFICATIONS *(continued)*

LEDs		Power Ethernet LAN Activity Wireless LAN Activity		
Interface		Ethernet 802.3 -10Base-T (RJ 45 Connector)		
Dimensions		5.7 in x 6.9 in x 2.8 in/145 mm x 175 mm x 70 mm (includes table mount bracket) 5.1 in x 6.9 in x 1.8 in/130 mm x 175 mm x 45 mm (includes wall mount bracket)		
Weight		1.1 lbs/0.50 kg		
Power Supply		External wall plug-in unit: Auto sensing 100/240 VAC 47-63 Hz, 9V/1.1A Ready for -Power Over Ethernet		
Temperature Range (operational)		32° to 104° F/0° to +40° C Operating humidity maximum 90% (no condensing allowed)		
Speed Options	**11 Mb/s**	**5.5 Mb/s**	**2 Mb/s**	**1 Mb/s**
Range in meters (feet) Open Office	160 m (525 ft)	270 m (885 ft)	400 m (1300 ft)	550 m (1750 ft)
Semi Open Office	50 m (165 ft)	70 m (230 ft)	90 m (300 ft)	115 m (375 ft)
Closed Office	25 m (80 ft)	35 m (115 ft)	40 m (130 ft)	50 m (165 ft)
Receiver Sensitivity	-82	-87	-91	-94
Delay Spread (at FER of < 1%)	65ns	225ns	400ns	500ns

Compaq PCs use genuine Microsoft® Windows®
http://www.microsoft.com/piracy/howtotell

Microsoft is a trademark of Microsoft Corporation in the U.S. and other countries.

Índice de figuras

Índice alfabético