IMPLEMENTACIÓN DE SENSORES DE ULTRASONIDOS EN UN SISTEMA AUTÓNOMO DE TIEMPO REAL.

Mª Esther Gilaberte Sanz

18 de marzo de 2003

Índice general

Al final todo acaba funcionando.

Carlos A. García.

AGRADECIMIENTOS

Ante todo agradecer a mis padres todo su apoyo y paciencia porque sin ellos nada de todo esto habría sido posible.

Y GRACIAS también a todos aquellos (amigos, compañeros, profesores,...) que me han apoyado y ayudado durante todos estos años.

PROLOGO

Los sistemas autónomos son sistemas complejos difíciles de desarrollar. Integran múltiples sensores y actuadores, tienen muchos grados de libertad y deben reconciliar tareas de tiempo real con sistemas que no pueden cumplir con los tiempos de entrega.

Los robots autónomos son sistemas completos que operan dentro de entornos complejos sin la actuación directa del ser humano. Ellos procesan señales, establecen relacciones, toman decisiones en tiempo de ejecución y adaptan sus planes de actuación a las diferentes circunstancias externas. La información que reciben les hace autoprogramables, alterando sus programas en función de las condiciones de contorno. Estos sistemas incluyen ciertas técnicas de inteligencia artificial en sus comportamientos. En la mayoría de los casos, la información requerida es la posición, velocidad, el par, aceleración, fuerza, tamaños y formas de objetos y temperatura. La cuantificación de estas medidas se realiza a través de sensores mecánicos, ópticos, térmicos, eléctricos, ultrasónicos,...

Los robots móviles son una de las tecnologías que más interés ha despertado en la industria por cuanto su posible aplicación a una gran diversidad de tareas de forma cooperante con el ser humano. Conforme la tecnología se ha ido desarrollando ha crecido en importancia el concepto de "*Autonomía*" el cual sobrepasa el concepto de sistema automático. La autonomía es un requerimiento adicional importante. Para definir la autonomía, podemos recurrir a definiciones suministradas por el etólogo Smithers que dice: "La idea central del concepto de autonomía se identifica en la etimología del término: autos (propio) y nomos (ley o regla). Se aplicó por primera vez en la antigua Grecia para referirse a aquellas ciudades o estados que se regían por leyes propias en lugar de vivir acorde al poder de un gobierno externo. Es útil contrastar el concepto de autonomía con el de sistema automático... Los sistemas automáticos se autoregulan pero ellos no establecen las leyes que sus reguladores intentan satisfacer. Estas leyes les son suministradas o están

inmersas en su construcción. Los sistemas automáticos son capaces de conducirse a lo largo de un camino corrigiendo y compensando los efectos de las perturbaciones externas. Los sistemas autónomos son capaces de generar por ellos mismos las leyes y estrategias con las que regularan su comportamiento: se autogobiernan y se autoregulan. Determinan el camino a seguir y se conducen sobre él".

Esta definición recoge la cuestión esencial, para ser autónomo primero tiene que ser automático. Esto implica sentir el entorno y ejercer acciones sobre él de manera beneficiosa para el agente y las tareas que debe desarrollar. Pero la autonomía va más allá que el automatismo, porque se supone que la base de autoregulación se genera desde la propia capacidad del agente de componer y adaptar sus principios de comportamiento. Más aún, el proceso de construcción y adaptación es algo que tiene lugar mientras el agente opera en su entorno.

Para conseguir esto, el robot requiere una serie de capacidades que en gran medida se agrupan bajo el concepto de inteligencia.

Se puede considerar que la inteligencia se centra en la "*habilidad de un sistema de mantenerse a sí mismo mediante la creación y uso de representaciones*".

Muy estrechamente ligada a la autonomía está la capacidad de percibir el entorno y actuar sobre el mismo. Esta propiedad es la que permite al robot obtener datos sobre su entorno (sensación) por medio de unos sensores y elaborarlos para su utilización (percepción) por medio de procesos de fusión más o menos elaborados. Parece claro que a mayor capacidad de autonomía , mayor es la capacidad de percepción y actuación necesaria. En el proceso de percepción no sólo se ven involucrados los sensores, con el objeto de medir el tiempo de vuelo de una señal o una cantidad de luz. También es necesario percibir situaciones erróneas o peligrosas. Para esto además de la capacidad de sentir y percibir es necesaria una cierta inteligencia en forma de conocimiento, bien previo o aprendido. Por otro lado, la inteligencia, a su vez, requiere de una capacidad de percepción para poder evolucionar y adaptarse. Esto lleva a un bucle cerrado en el cual no hay un principio y un final. Es decir, no hay un antes ni un después. Es cada vez más admitido entre la comunidad investigadora que la capacidad de percepción de un robot construye o modifica su inteligencia computacional, la cual a su vez decide su actuación o interacción con todo lo que le rodea cerrándose todo el ciclo. Esta idea caería lejos de lo que en robótica aplicada a la industria podríamos definir como un control clásico de un robot, pero también es cierto que los

robots que hoy en día se utilizan en la industria, en el 90 % de las ocasiones lo hacen en entornos estructurados y poco dinámicos, por lo cual sus necesidades de percepción y por tanto la inteligencia desarrollada es mínima. Los grandes retos hoy por hoy de la robótica móvil se encuentran en :

- Capacidad de construcción de representaciones del entorno autónomas.
- Capacidad de trabajo en entornos cambiantes y dinámicos como los entornos al aire libre (outdoor).
- Avances en los sistemas de visión artificial.
- Capacidad de cooperar en grupos de robots.
- Capcidad de comunicación entre el robot y el hombre de forma natural.
- Avances en percepción y su relación con la inteligencia.

INTRODUCCIÓN

Este proyecto se engloba dentro de uno mucho más ambicioso cuyo fin es la investigación de las técnicas de control de sistemas autónomos. Se pretende crear un robot que reaccione ante lo imprevisto.

Como plataforma de desarrollo se parte de un modelo de radio control de Tamiya, tipo "Bigfoot", que aporta estabilidad, capacidad de carga y potencia motriz, y es de pequeño tamaño lo que le dota de movilidad en interiores.

Esta plataforma ha sufrido algunas modificaciones:

- Se le ha suprimido el control remoto.
- Se ha sustituido el servo de dirección por otro de par mayor, este cambio es necesario debido al aumento de peso que va a experimentar el coche al dotarlo de sensores, baterías, procesadores, circuitos auxiliares,...
- Se han cambiado los engranajes metálicos por otros que proporcionan una mayor durabilidad.
- Se le han eliminado los amortiguadores originales.

Además en la fase de diseño se tuvo en cuenta la distribución de todos estos componentes para facilitar el cambio y mantenimiento de los mismos.

Este sistema autónomo tiene un cerebro principal situado en un armario de control de la firma Honeywell, situado en el centro de cálculo del Departamento de Ingeniería de Sistemas y Automática(DISAM) de la Escuela Técnica Superior de Ingenieros Industriales de la Universidad Politécnica de Madrid. Este cerebro es un computador con procesamiento en paralelo y se comunica con el coche mediante una red inalámbrica 802.11b del IEEE.

Figura 1: Plataforma.

Dentro del coche se han incorporado dos procesadores:

- El procesador principal es una placa Wafer 5820, con factor de forma de 3,5 pulgadas, que ofrece un completo PC con procesador Pentium, ethernet y todos los puertos necesarios ocupando un espacio mínimo y con un consumo muy pequeño, además de la poca disipación de calor que produce, con el consecuente ahorro en cuanto a problemas de diseño se refiere. La red inalambrica comunica esta placa Wafer con el cerebro principal.
- Un procesador secundario que se encarga de gestionar la información procedente de los sensores y actuar en consecuencia, así como de controlar el servo instalado. Como procesador secundario se usa el aJ-80 dentro del entorno de desarrollo JStamp que tiene como principal característica su arquitectura nativa Java y su pequeño tamaño (1 x 2 pulgadas).

El objetivo de este proyecto será dotar de información al sistema mediante la implantación de sensores de ultrasonidos de BOSCH y programar el

procesador para que actúe en función de la información recibida, dotando así de autonomía al sistema.

Parte I

FUNDAMENTOS DEL SENSOR DE ULTRASONIDOS

Capítulo 1

FUNDAMENTOS DEL SENSOR DE ULTRASONIDOS

1.1. Introducción.

Los sensores de ultrasonido están siendo utilizados de forma creciente en los últimos 10 años por parte de los diseñadores de robots autónomos. Los principales motivos son la buena relación precio-cantidad de información proporcionada, que dichos sensores poseen. En la mayoría de las implementaciones, la distancia a la que se encuentran los objetos se determina mediante el calculo del tiempo de vuelo de la señal (Time of Fligth ó T.o.F.). Es posible, mediante el uso de varios de estos sensores, evitar obstáculos de forma rápida y sin interferencias entre los dispositivos.

1.2. Definición de sensor.

Los sensores van a ser en el robot la fuente de datos sobre los cambios tanto en el entorno (distancias a objetos, luz ambiental) como en sí mismo (nivel de las baterías...). Entorno a los sensores conviene antes de entrar en profundidad en los mismos definir una terminología:

- Transductor : aquel dispositivo que transforma una magnitud física de entrada (por ejemplo luz) en otra de salida (normalmente voltaje).
- Procesador de señal : dispositivo que realiza una cierta operación con una señal como un filtrado, amplificación etc.
- Sensor : estará compuesto de uno o más transductores y algún procesador de señal.

Esta sería una definición bajo un punto de vista de bloques. De forma funcional podemos decir que un sensor es un dispositivo que capta la magnitud de una variable física en un sistema físico o entorno. Cualquier dispositivo que es alterado por las variaciónes de una manitud física de una forma predecible y medible es un sensor para esa magnitud. Así, un sensor viene caracterizado por su función de transferencia, que relaciona el valor de la magnitud física con el valor que éste suministra en su salida.

Las características que definen un sensor son:

- **Accesibilidad.** Es la zona del entorno físico en la que las variaciones de la magnitud á medir afectan al sensor.

- **Dimensión.** Segun el valor sea escalar, vectorial, n-dimensional...
- **Rango de operación.** Loa sensores operan sólo en un determinado rango de valores de la magnitud a medir e incluso de otras magnitudes.
- **Datos.** Formato de los datos. Discretos o contínuos, procesamientolocal, ancho de banda, capacidades de compresión de la información...
- **Sensibilidad.** Especificaciones sobre exactitud y precisión.
- **Localización.** Local o remoto al lugar de procesamiento.
- **Inteligencia.** Se dice que un sensor es inteligente si tiene capacidades de procesamiento o decisión. El uso de sensores inteligentes permite establecer un compromiso entre computación y comunicación. Las capacidades de procesamiento local pueden reducir las necesidades de ancho de banda para una red de sensores.

En robótica móvil se usan una gran variedad de sensores que miden distintas magnitudes. De todos ellos aquellos que más se utilizan son los orientados a resolver uno de los problemas fundamentales de todo sistema autónomo: la determinación de la posición. Básicamente este problema se resuelve mediante medidas relativas de posición o mediante sistemas absolutos de posicionamiento. De todos ellos hay uno que destaca por ser utilizado en gran parte de las plataformas móviles, me estoy refiriendo a los sensores de ultrasonidos. Estos han sido utilizados desde los primeros robots dedicados a la investigación hasta las más recientes plataformas comerciales, con pocas variaciones en la tecnología. Junto con estos, los sensores basados en luz son otros de los sensores utilizados para determinación de distancias a objetos y por tanto para la construcción de mapas de entorno.

1.3. Conceptos necesarios o terminología necesaria

Entorno a los sensores de ultrasonidos conviene antes de entrar en profundidad en ellos definir algunos conceptos:

1.3.1. Zona muerta.

Los sensores de ultrasonidos tienen una zona muerta en la cual no pueden detectar exactamente el objeto u obstáculo. Esta es la distancia entre la membrana sensora y el mínimo rango de sensibilidad. Si el objeto está demasiado cercano, la señal ultrasónica puede chocar contra el objeto antes de que dicha señal haya dejado el transductor, por tanto, la información del eco devuelta al sensor es ignorada por el transductor, puesto que éste está todavía transmitiendo y no recibiendo. Si el objeto está demasiado cerca puede ocurrir otro problema, que el eco generado se refleje sobre la membrana sensora y viaje de nuevo hacia el objeto. Estos ecos múltiples pueden dar lugar a errores cuando el objeto está dentro de la zona muerta.

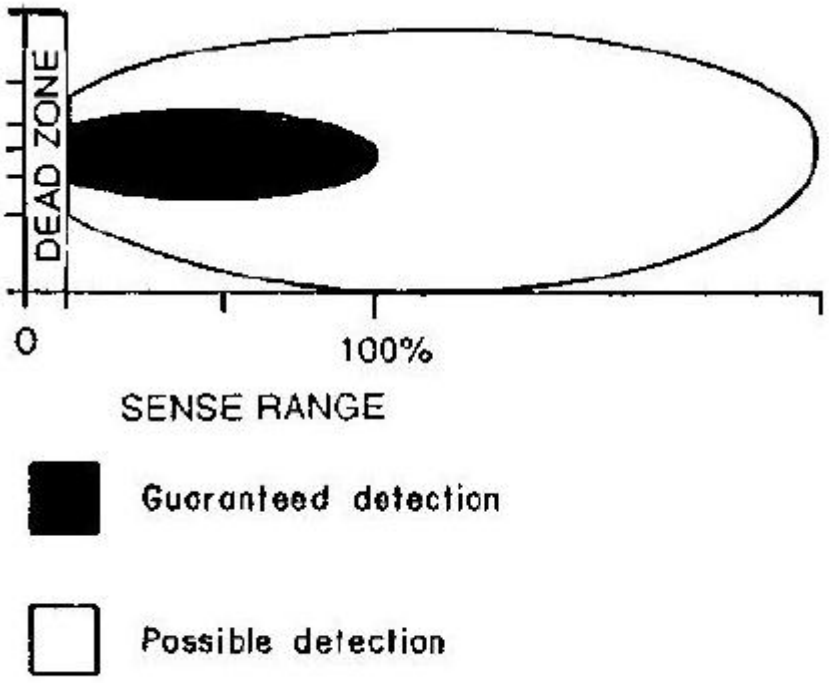


Figura 1.1: Zona muerta.

1.3.2. Máximo rango sensible.

El rango máximo en el que se puede detectar cada objeto y cada aplicación se determina mediante experimentación. En las figuras 1.1 y 1.3 se muestran las características de sensibilidad y las distancias sensibles típicas para el sensor de ultrasonidos.

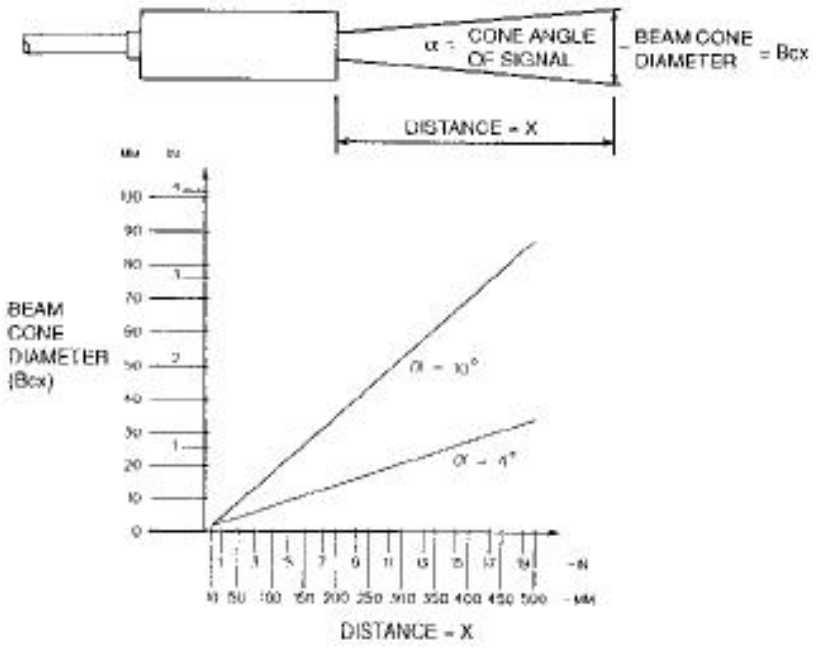


Figura 1.2: Cono de emisión.

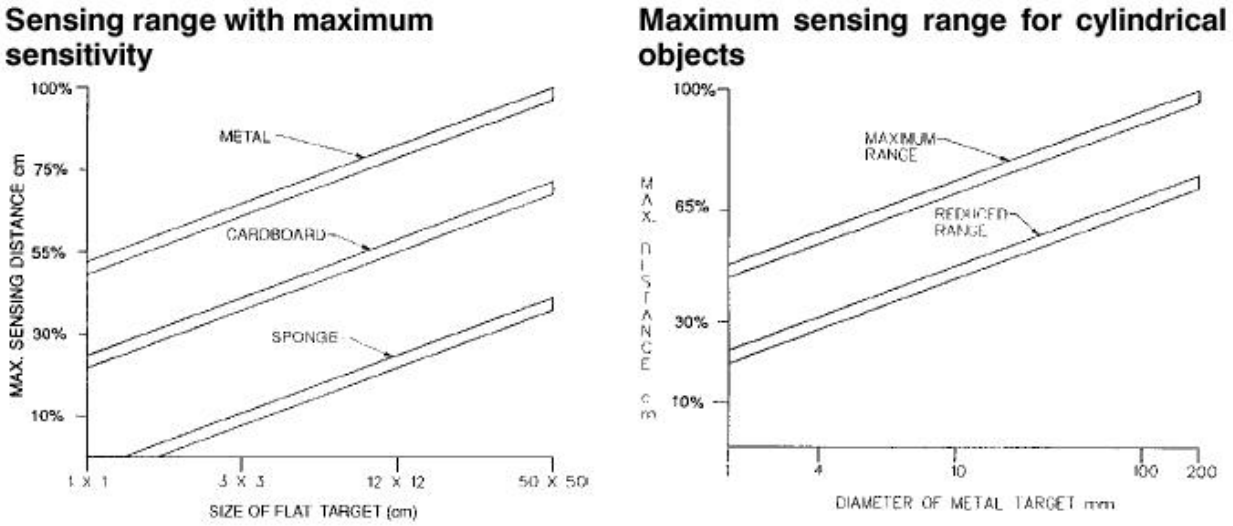


Figura 1.3: Comparaciones.

1.3.3. Ángulo de emisión.

El ángulo del cono de emisión está formado por los puntos del espacio en los que la señal del sensor es atenuada por lo menos 3dB. Fuera de este cono la señal de ultrasonidos existe pero es bastante débil. Este cono debe determinarse experimentalmente y dentro de él pueden detectarse los objetos.

1.3.4. Diámetro del cono de emisión.

El sensor de ultrasonidos emite un haz de sonido en forma de cono que elimina los lóbulos laterales. Es importante el tamaño del objeto respecto del tamaño de la zona que abarca el haz. Teóricamente, el objeto más pequeño detectable es aquel que mide la mitad de la longitud de onda de la señal del sensor de ultrasonidos. Para 215KHz, la longitud de onda de la señal es de 0.063", por lo que, bajo condiciones ideales, estos sensores son capaces de detectar objetos con un tamaño mínimo de 0.032". Normalmente los objetos son grandes, por lo que son detectados a varias distancias. para determinar

el área que abarca el sensor de ultrasonidos a una determinada distancia (diámetro del cono de emisión), se usa la fórmula:

$$Box - 2 * X * \tan\frac{\alpha}{2} \tag{1.1}$$

Donde Box es el diámetro del cono de emisión a la distancia X.
X es la distancia del objeto (obstáculo) al sensor.
α es el ángulo del cono de emisión.

1.3.5. Frecuencia de disparo.

La máxima frecuencia a la que un sensor es capaz de dispararse o pararse depende de varias variables, las más significativas son:

- el tamaño del objeto
- el material del que está hecho
- la distancia a la que se encuentra.

De este modo la máxima frecuencia para un objeto pequeño será menor que para un objeto grande. Los materiales que absorben de sonido altas (algodón, esponja,...) son más difíciles de detectar que el acero, el cristal o el plástico. De este modo, ellos tienen también una menor máxima de frecuencia cambiante.

La distancia del objeto al sensor es muy importante para determinar el máximo de frecuencia de disparo. El sensor manda una señal ultrasónica por el aire, la señal deja el sensor, viaja hasta el objeto, choca contra él y vuelve hasta el sensor como un eco.

1.3.6. Inclinación del haz de ultrasonidos.

Si un objeto liso es inclinado más de $\pm 3^o$ con respecta a la normal al eje del haz de emisión de la señal de ultrasonidos, parte de la señal es desviada del sensor y la distancia de detección disminuye. Sin embargo, para objetos pequeños situados cerca del sensor, la desviación respecto a la normal puede

aumentar hasta $\pm 8^o$. Si el objeto está inclinado más de $\pm 12^o$ respecto a la normal, toda la señal es desviada fuera del sensor y el sensor no responderá. La señal que choca contra un objeto de superficie rugosa (como un material granulado) se difunde y refleja en todas las direcciones y parte de la energía vuelve al sensor como un eco débil.

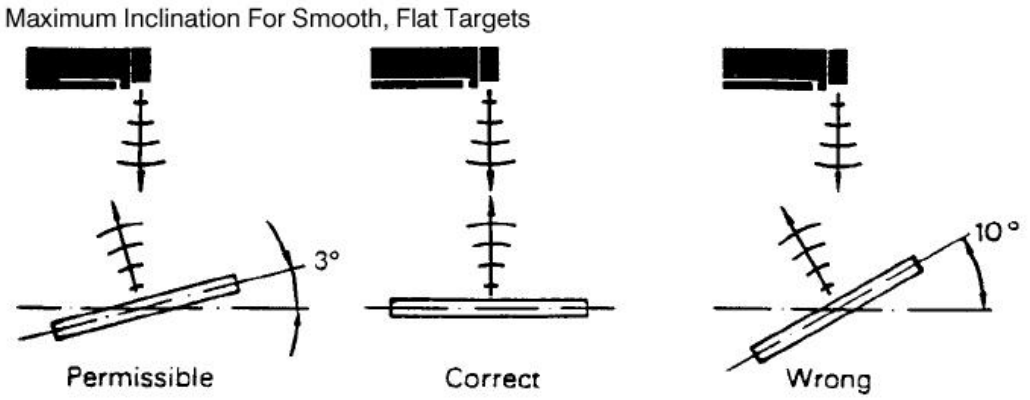


Figura 1.4: Dependencia del rango de inclinación.

1.4. Características físicas de la onda ultrasónica.

Las ondas de ultrasonido se han utilizado para la determinación de la distancia desde hace muchos años, así mismo se han empleado para la ubicación de un objeto en el espacio, en sistemas militares, de construcción, control industrial y robótica.

El funcionamiento básico del sensor de ultrasonidos se basa en la medida del tiempo transcurrido entre la emisión de un ultrasonido y la recepción del eco correspondiente al mismo. Conocido el tiempo, y siguiendo la velocidad de propagación del sonido en el aire la expresión

$$v = 331{,}6 + 0{,}6 * T \tag{1.2}$$

(T en o C y v en m/s), la distancia a la que se encuentra el objeto que ha devuelto el eco se calcula según la ecuación :

$$d = (331{,}6 + 0{,}6 * T) * \frac{t}{2} \tag{1.3}$$

Donde t representa el tiempo transcurrido entre la emisión y la recepción.

Este tiempo es dividido por 2 para calcular sólo el tiempo que tarda en llegar la onda al objeto. El empleo de ondas de ultrasonidos en lugar de ondas electromagnéticas se justifica por los siguientes puntos:

- Las ondas acústicas, al contrario que las electromagnéticas, requieren de un medio para transmitirse; como puede ser el aire.
- La velocidad de transmisión de las ondas ultrasónicas es mucho menor que la de las electromagnéticas (velocidad de la luz). Esta característica permite emplearlas para la medida de distancias pequeñas.
- La longitud de onda de un ultrasonido a 50KHz. es de 6.8mm:, medida que es mayor que la rugosidad de la mayoría de las superficies, lo cual permite que la reflexión que se produce en la mayoría de los objetos sea especular y no difusa.

La apertura del cono de emisión constituye en realidad una aproximación del lóbulo central de emisión, pues la expresión de la onda emitida es en realidad mas compleja, siguiendo un patrón como el mostrado en la figura [1.5]

1.5. Estudio de las restricciones de percepción: sensores de ultrasonidos

Los sensores de ultrasonidos no son ideales, es decir producen lecturas bastante aproximadas a la realidad, pero su repetibilidad es casi nula, como se puede observar en la figura, que muestra los mapas que se reconstruyen en tiempo real de un mismo entorno de navegación, en diferentes experimentos.

Los sensores de distancia por emisión de ultrasonidos permiten modelar el entorno en el que se mueven los robots móviles basándose en la forma en la que emiten y recogen las ondas de sonido.

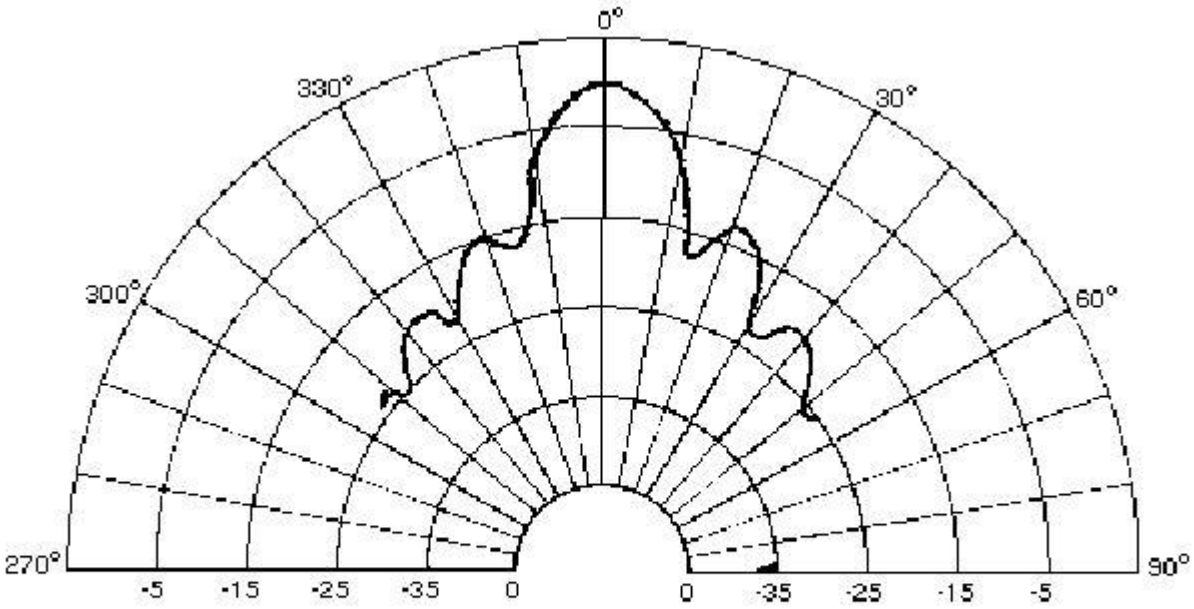


Figura 1.5: Energía emitida por la onda ultrasónica en todas las direcciones del entorno. Observar como el lóbulo central abarca un ángulo de $\approx 30^o$.

Existen básicamente tres posibles aproximaciones al problema de la medida de la distancia a un objeto:

- Los sensores basados en el tiempo de vuelo de un pulso de energía que viaja hacia un objeto en el que se refleja, para volver de nuevo al receptor.
- Los sensores basados en la medida del desplazamiento de fase, que necesitan una transmisión de señal contínua, en lugar de emitir pulsos.
- Los sensores basados en un radar de frecuencia modulada. Esta técnica está bastante relacionada con la anterior.

1.5.1. Sensores basados en el tiempo de vuelo

La mayor parte de los sensores usan esta técnica. El pulso de energía con el que se lleva a cabo la medida, puede provenir tanto de un generador de ultrasonidos, como de uno de energía óptica, o uno de radiofrecuencia.

Por lo tanto, debido a esto, los parámetros importantes para llevar a cabo la medición son la velocidad del sonido en el aire, y la velocidad de la luz. El tiempo medido es un representación de la distancia existente entre el dispositivo medidor, y el objeto que ha reflejado la onda.

La ventaja de estos sistemas, viene de su naturaleza directa de la medición, y del hecho de que un mismo dispositivo puede actuar tanto como generador, como receptor. La distancia al objetivo, puede obtenerse directamente de la salida del sensor, y no es necesario realizar ninguna suposición sobre si las superficies son planas, ni sobre la orientación existente entre estas y el sensor.

Sus desventajas, vienen del hecho de que no se puede llevar a cabo un esquema de triangulación, debido a la nula distancia existente entre emisor y receptor. en cualquier caso, estos sistemas mantienen una buena precisión, siempre y cuando el obstáculo que se analiza presente un eco fiable.

Las fuentes de errores en la medida, vienen fundamentalmente de la imprecisión de la medida del tiempo, de las variaciones en la velocidad de onda transmitida, de las deficiencias en el circuito temporizador, o por interacción de la onda con la superficie en la que debe reflejarse. A continuación se incluye una breve descripción de cada uno de estos problemas.

Velocidad de propagación de la onda: Para las aplicaciones de robótica móvil, las variaciones en la velocidad de propagación de las ondas electromagnéticas, puede ser totalmente ignorada. Este no es el caso sin embargo de los sistemas acústicos, donde la velocidad del sonido, está muy influenciada por la temperatura y en menor medida por la humedad.

Incertidumbre de la detección: Estas variaciones vienen motivadas por la distinta reflexión de las ondas en distintos tipos de objetos. Algunos las reflejan con una intensidad mayor que otros, lo que hace que los sistemas de detección respondan de forma más rápida ante los primeros, y por lo tanto hagan que esos objetos parezcan más próximos. Por este motivo, es muy importante la selección del valor umbral, a partir del cual se inicia la detección de la onda.

Incertidumbre en la medida del tiempo: Debido a la relativamente lenta velocidad del sonido en el aire, en comparación con la de la luz, los sistemas acústicos, tienen unos requisitos de velocidad muy inferiores a los de sus contrapartidas basadas en la luz. La velocidad a la que se transmiten las ondas electromagnéticas, obliga a imponer en los circuitos electrónicos unos requisitos de velocidad muy fuerte, requiriendo circuitería que responda en tiempos inferiores al nanosegundo. Para obtener precisiones de tan solo 1cm, se necesita una circuitería con temporizaciones de 3 nanosegundos. Conseguir esto resulta muy caro, y aparta estos sistemas de medida de las aplicaciones normales.

Interacción con las superficies: Cuando la onda se refleja en un objeto, solo una pequeña fracción de la señal vuelve al receptor. El resto de la energía se refleja en otras direcciones, o es obserbida por la propia superficie. La energía reflejada en distintas direcciones puede, a su vez, volver a reflejarse en más objetos, y llegar finalmente al receptor siguiendo una línea que no es recta, o volver reflejada a un receptor que no corresponde con el lugar desde el que fue emitida. A esto se lo conoce como habla cruzada.

1.5.2. Sensores basados en desplazamiento de fase.

En este caso no se emite un haz de onda ultrasónica, sino una señal de larga duracción hacia el objeto en cuestión. Una pequeña parte de la señal volverá al detector, siendo comparada con una señal de referencia igual a la emitida, pudiendo así medir la diferencia de fase entre ambas. El desplazamiento de fase puede expresarse en función de la distancia como:

$$\Phi = \frac{4 * \pi * d}{\lambda} \tag{1.4}$$

donde Φ es el desplazamiento de fase.
d es la distancia al objeto.
λ es la longitud de onda de la señal emitida.

De esta expresión podemos obtener que :

$$d = \frac{\Phi * \lambda}{4 * \pi} = \frac{\Phi * v}{4 * \pi * f} \tag{1.5}$$

Donde f es la frecuencia de modulación de la señal. Para altas frecuencias, el desplazamiento de fase se puede medir multiplicando las dos señales (entrante y de referencia mediante un dispositivo electrónico y hallando la media sobre

dichos ciclos). Este proceso de integración puede ser costoso en tiempo, por lo que se hará difícil con una frecuencia de muestreo alta. Este proceso se expresa matemáticamente como

$$\lim_{T\to\infty} \frac{1}{T} * \int_o^T (\sin(\frac{2*\pi*v}{\lambda} + \frac{4*\pi*d}{\lambda}) * \sin(\frac{2*\pi*v}{\lambda}))$$

expresión que puede reducirse a:

$$\Phi = A * \cos(\frac{4*\pi*d}{\lambda}) \tag{1.6}$$

esta formulación para la obtención de la distancia puede presentar un problema de ambigüedad de intervalo cuando la distancia medida excede la longitud de onda de la señal modulada. Este problema reduce el atractivo de los sistemas de desplazamiento de fase, ya que se hace necesario mecanismos adicionales para eliminarla o bien asumir su posible aparición. Entre las posibles soluciones se encuentra limitar la distancia máxima medible a lo permitido por el intervalo de ambigüedad, realizando dos medidas a distintas frecuencias de modulación.

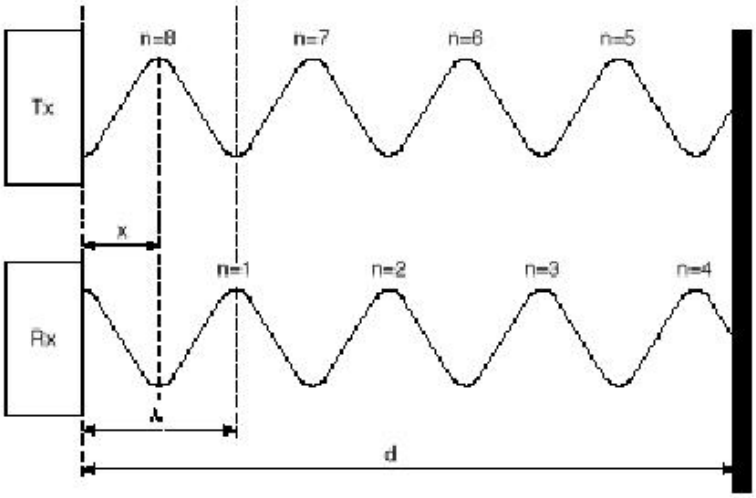


Figura 1.6: Efecto del desplazamiento de fase

1.6. El sistema ultrasónico.

El sistema se compone fundamentalmente de un transductor electrostático, y de una pequeña placa electrónica en la que está la circuitería encargada de emitir los pulsos, recibirlos, y procesarlos. Para determinar la distancia a un objeto, se mide de forma externa, el intervalo transcurrido entre la emisión y la recepción del pulso ultrasónico.

Con el uso de un reloj apropiado, se pueden obtener precisiones elevadas. Si se necesita una precisión muy alta, es necesario llevar a cabo correcciones, según la temperatura del aire.

1.6.1. Descripción general del sistema

La determinación de distancias por medio de un sistema de medición de ecos es un proceso muy simple. Un pulso corto de energía ultrasónica se genera de forma electrónica, se amplifica y se envía a un transductor. La señal viaja a través del medio 8 en general el aire), se refleja en un objeto, y vuelve al transductor. Esta señal se recibe, se amplifica, y se procesa por el sistema electrónico. El tiempo que ha tardado en viajar la señal se puede usar para determinar la distancia a la que está el objeto, al conocer la velocidad a la que se transmite el sonido en el aire.

El sistema de medida más simple se compone tan solo de dos módulos, el módulo electrónico, y el transductor. El transductor, de tipo electrostático o piezoeléctrico, se usa tanto para emitir el pulso, como para recibir el eco. El segundo módulo, contiene toda la circuitería necesaria para generar la señal que se transmitirá, enviarla al transductor, recibir el eco, y procesar la información obtenida de éste.

La distancia del transductor al objetivo, puede determinarse por un circuito adicional que conozca la velocidad del sonido en el aire, y el intervalo de tiempo transcurrido entre la emisión y la recepción de la señal.

Esta amplificación llevada a cabo en el receptor, es la parte más complicada del sistema, ya que debe ser de caracter exponencial (al igual que lo es la atenuación del sonido en el aire), y tiene que partir desde valores de amplificación pequeños hasta subir la intensidad de la señal recibida en varios ordenes de magnitud.

Para minimizar el peligro de que pequeños ruidos confundan al sistema y éste no crea recibir falsos ecos, el sistema dispone de un integrador, en el que se recibe la señal entrante. Solo cuando a la salida del integrador se alcance un nivel determinado se considera que se ha recibido un eco auténtico.

Las señales de salida, indican los instantes en los que se envió el pulso, y el instante en el que se recibe el eco, por lo que resulta sencillo llevar a cabo funciones de control, así como medir el tiempo transcurrido entre ambos eventos.

1.6.2. El transductor.

La parte más importante del sistema es el transductor electrostático. Está compuesto por una membrana muy fina, una lámina recubierta de oro para formar el electrodo negativo de un diafragma de vacío. El electrodo positivo, es una lámina recubierta de aluminio, que también sirve como estructura resonante para el diafragma.

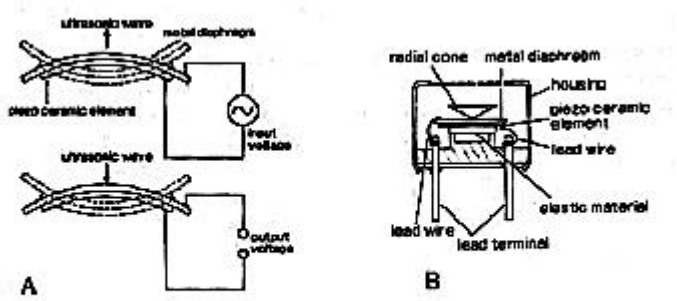


Figura 1.7: Transductor del sensor de ultrasonidos

La señal no se transmite de forma lineal en línea recta, a partir del transductor, sino que en lugar de esto se genera una señal que se extiende formando un cono.

1.6.3. El módulo electrónico.

La electrónica del sistema de sonar está reunida en un único módulo, usando circuitos integrados específicos para realizar las funciones digitales y analógicas. Estos dos circuitos integrados, están unidos junto con los componentes discretos necesarios, en una pequeña placa de tan solo 2cm de ancho y 4 de largo.

Cuando se activa el sistema, estos circuitos generan una serie de pulsos a frecuencias discretas distintas. La señal se amplifica en un transformador incluido en la placa. La componente contínua se mantiene en el transductor, por medio de un condensador de almacenamiento, mientras este actúa como micrófono para recibir el eco. El transductor se bloquea durante unos pocos microsegundos, para evitar la recepción del eco que se genera al salir la señal del propio transductor, y a partir de ese instante pasa a actuar como un micrófono.

Para compensar la pérdida de energía de la señal al recorrer mayores distancias, la ganancia del amplificador que trabaja en la recepción de la señal, se va incrementando en función del tiempo, además de disminuir en ancho de banda para disminuir los efectos del ruido.

Cuando una señal es recibida por el sistema, y si esta señal supera un nivel umbral mínimo, se activa una fuente de corriente, que va cargando un condensador hasta que alcance éste los 1.2V. En este instante, se considera que la señal recibida es el eco, y se genera una señal lógica.

Para medir el intervalo de tiempo transcurrido entre las dos señales hay que recurrir a un circuito externo. Al multiplicar este valor por la velocidad del sonido, y teniendo en cuenta que la señal se ha recorrido dos veces, dividiendo entre dos, se obtiene la distancia a la que se encuentra el objeto del transductor.

1.6.4. Medida del eco.

Mientras que el hecho de medir el eco recibido es relativamente simple, algunos de los procesos involucrados en él no lo son. La propagación de la energía acústica a través de un medio fluido es muy compleja, pero afortunadamente, a las frecuencias con las que se trabaja está bastante bien estudiada, y la mayor parte de los factores de atenuación, reflexión, refracción, etc. pueden ser simplificados de una forma bastante satisfactoria.

Los factores acústicos que más afectan al rendimiento del sistema de medida por sonar están relacionados con el rendimiento del transductor, su frecuencia de trabajo, y la distancia máxima a la que se desee llegar con las medidas.

Dejando a un lado los aspectos electrónicos, el aspecto que más interesa es la relación existente entre el alcance y la frecuencia de trabajo. Esta es debida a la atenuación que tiene cada frecuencia al viajar por el aire. El mecanismo de esta perdida es bastante complejo, dependiendo de factores tales como la temperatura, humedad, etc. El efecto de atenuación debido a la humedad es máximo, al trabajar con valores intermedios de la misma.

1.6.5. Resolución.

En general, con un sistema de medida de ecos ultrasónicos, se puede llegar a alcanzar una resolución del orden de la longitud de onda de la señal que se transmite.

Esto, por supuesto, depende del tipo de sistema de detección empleado. En estos sensores se usa una sencilla técnica de integración. Cuando la señal que se recibe, supera un umbral determinado, se activa una fuente de corriente que empieza la carga de un condensador. Existe un drenaje continuo de la carga de este condensador, para garantizar que pequeños picos de ruido consecutivos no vayan cargando el condensador, y lo lleven al valor umbral en el que se identifica la señal como positiva.

Usando una velocidad de carga de este condensador, diez veces superior a la velocidad de descarga, se consigue un esquema que puede alcanzar una resolución de aproximadamente unos 6mm, siempre y cuando el objetivo esté fijo respecto al sensor, y se cumplan algunas otras restricciones.

Si se necesita alcanzar realmente esta precisión máxima, es necesario conocer de forma precisa la velocidad del sonido en el aire. La velocidad del sonido en el aire, puede ser determinada de acuerdo con la expresión:

$$v = 331{,}4 * (\frac{T}{273})^{\frac{1}{2}} \tag{1.7}$$

donde T es la temperatura en grados Kelvin y se ve que existe una fuerte dependencia de la temperatura.

Si se trabaja dentro de un margen de temperaturas de entre -30ºC y 30ºC, entonces esta expresión puede aproximarse por:

$$v = 331,4 + 0{,}607 * t \tag{1.8}$$

donde t es la temperatura en grados centígrados.

Es fácil ver que la dependencia de la temperatura es muy fuerte, y por lo tanto si se desea obtener una precisión alta, es necesario llevar a cabo la compensación. En un entorno en el que la temperatura pueda oscilar en un rango de 60ºC, la variación de la velocidad del sonido llega a un 5 %. La correción de este valor puede llevarse a cabo bien por el sistema de medición, o bien por el procesador que vaya a analizar los datos obtenidos.

1.7. Funcionamiento del sensor.

El funcionamiento típico de un sensor de ultrasonidos viene dado por un ciclo de operación que sigue los siguientes pasos:

1. El circuito de control dispara el transductor, quedando a la espera de una señal que confirme el comienzo de la transmisión.
2. El circuito de recepción es blanqueado durante un tiempo, para evitar que ondas residuales de la transmisión puedan ser interpretadas como falsos ecos.
3. Las señales recibidas son amplificadas mediante un amplificador de ganancia variable, el cual compense la atenuación del medio en aquellas señales que han recorrido una mayor distancia.
4. Las señales recibidas que superen un determinado nivel son reconocidas como ecos, calculándose la distancia a la que se encuentra el objeto que lo ha provocado.

De los puntos anteriores se deduce que la distancia mínima a la que un sensor responderá, vendrá dada por el tiempo de blanqueo. Para el caso del sensor usado es de 2.38 ms sustituyendo este valor en la ecuación [1.2] se obtiene una distancia mínima de $\approx$ 20cm. Por otro lado, la distancia máxima vendrá dada por la atenuación de la onda ultrasónica en el medio en el que se propague (aire) y por la ganancia del amplificador que recoge los ecos. En

los sensores usados esta distancia es de 10.5cm. La señal emitida por cada sonar tiene una duración de 300μs., y consta de los siguientes pulsos: 8 a 60KHz..8 a 57KHz.,16 a 53KHz. y 24 a 50KHz.. El hecho de transmitir pulsos a varias frecuencias tiene como objetivo reducir las perturbaciones que pueda inducir un objeto a una determinada frecuencia en una señal ultrasónica. Emitiendo a varias frecuencias se asegura que aunque una frecuencia se vea perturbada, las otras no lo serán. En la figura 1.10 se muestra un cronograma del funcionamiento de este tipo de sonar. La apertura del cono de emisión de la onda ultrasónica se calcula a partir de la expresión:

$$\vartheta = \arcsin(\frac{0{,}61 * \lambda}{r}) \tag{1.9}$$

Donde λ es la longitud de onda, y r el radio del anillo exterior del sensor.

Los sensores de ultrasonidos tienen un transductor acústico que vibra a frecuencias ultrasónicas . Los pulsos son emitidos en haz cónico y apuntan a un objeto. Los pulsos se reflejan en el objeto y vuelven al sensor en forma de ecos. El instrumento de medida mide el tiempo de retraso entre cada emisión y el pulso de eco para determinar exactamente la distancia del sensor al objeto. Los sensores de ultrasonidos detectan todos los objetos sean del material que sean e independientemente del color que tengan. Detectan objetos claros, transparentes y brillantes tan fácilmente como los oscuros y opacos. Esta habilidad permite a los sensores de ultrasonidos detectar todo el rango de materiales, desde una botella de cristal transparente hasta neumáticos de goma negra. Si un material se cubre, el sensor puede detectar de manera exacta y repetidamente el material cubriente a pesar de lo brillante o claro que sea (no como ocurre con los sensores infrarrojos). Los sensores de ultrasonidos funcionan bien en ambientes toscos (humos, polvo, ruido).

1.8. Configuraciones.

1.8.1. Configuración de un solo sensor.

Se puede usar para detecar la presencia de un objeto o calcular su distancia respecto al sistema autónomo un sólo sensor de ultrasonidos rotatorio. El sensor se va girando pasos pequeños, tomando varias posiciones para la medición completa del entorno(360º). Como se mencionó anteriormente, la

intensidad de la señal va variando para diferentes ángulos respecto del centro del sensor y no supone que la detección del objeto se lleve a cabo mediante el lóbulo más sensible que es el central, puede producirse la medida por uno de los laterales. El problema en estas zonas es que la señal tiene la intensidad suficiente para activar el disparo por flanco y por tanto estimar una medida, pero no tiene la suficiente potencia como para hacerlo pronto, con lo cual se produce un incremento del tiempo en la estimación del vuelo. Como consecuencia de esto la estimación de la distancia es mayor. En resumen, el sensor de ultrasonidos tiene un cono de recepción ancho y puede sobrestimar distancias si el eco es débil.

Con el objeto de determinar las medidas que pertenezcan al lóbulo central (ya que son más fiables) y desestimar en la medida de lo posible las de los lóbulos laterales, se propone un modelo de sensor. Este modelo de interpretación de las medidas está basado en el concepto de "*Región of Constant Depth*" (RCD), que permite eliminar las estimaciones de distancia originadas por ecos débiles. El problema de este método es el tiempo de medida, puesto que se requieren varios minutos para una prueba de 360º, lo cual es impracticable para un modo de operación normal de un sistema autónomo.

Por tanto se puede adoptar una variante al método anterior. La base de este modelo se encuentra en la siguiente idea: Con la cantidad de información generada por un barrido de todas las posiciones posibles para los 360º, es imposible eliminar los ecos débiles. Si un objeto es detectado sólo por un eco, no es posible discernir si se trata de eco fuerte o débil. Pero si el mismo objeto se detecta en más de un eco, se puede incrementar la calidad de la información que se genera. Se puede decrementar las medidas sobrestimadas de ecos débiles tomando todos los mínimos de los ecos de la misma RCD. Por otro lado, la incertidumbre angular también puede ser decrementada haciendo que el objeto sea detectado en una zona donde los ecos de las lecturas se solapen. Un conjunto de ecos se considera que pertenecen al mismo objeto si son adyacentes en la lectura y además su estimación de la distancia al objeto no difiere más de un valor umbral de valor 3 cm. De esta forma se define una lectura como el agrupamiento de uno o más ecos adyacentes, y su valor de medida de distancia es la mínima de todos los ecos que la componen. Cabe resaltar que los ecos débiles se producen para ángulos mayores de 20º, siendo la diferencia entre dos ecos consecutivos de 18º, por tanto menor. Esto asegura que no suceda el que dos ecos consecutivos den una medida de distancia errónea. En la figura 1.8 se puede ver cual es el método de agrupamiento empleado y el cono efectivo que produce.

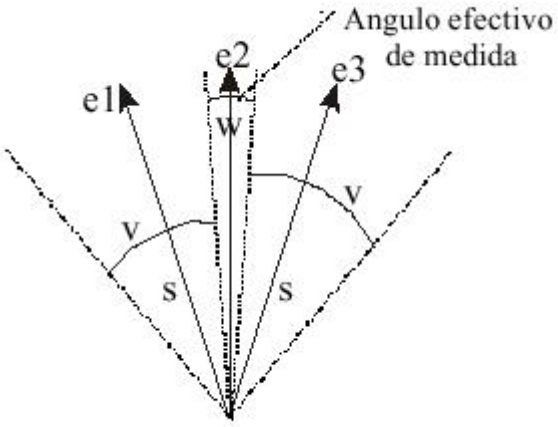


Figura 1.8: Agrupación de medidas.

1.8.2. Configuración de varios sensores.

Cuando dos o más sensores se montan cerca, es posible que existan interferencias acústicas. Si se evita la multiplexación de los sensores, solamente transmitirá uno cada vez. Pero si se sincronizan todos los sensores para que transmitan al mismo tiempo, se pueden conectar todos juntos.

Multiplexación de sensores.

Se pueden conectar varios sensores de ultrasonidos a un sólo amplificador. Ellos se pueden conectar en paralelo o en serie según se muestra en las figuras.

- Si se conectan en paralelo, todos los sensores se dispararán a la vez, en el mismo instante. El sensor más cercano al objeto proporcionará la distancia al amplificador. La información proveniente del resto de los sensores es ignorada. Como máximo se pueden conectar seis sensores en paralelo y se tienen que colocar diodos y una resistencia de pull-up como se muestra en la figura 1.9.

- Si se conectan en serie, solamente se dispara un sensor cada vez proporcionando la medida de la distancia. Este sensor se deshabilita y se habilita el siguiente. La multiplexación en serie se hace con alimentaciones externas al amplificador.Como puede verse en la figura 1.10.

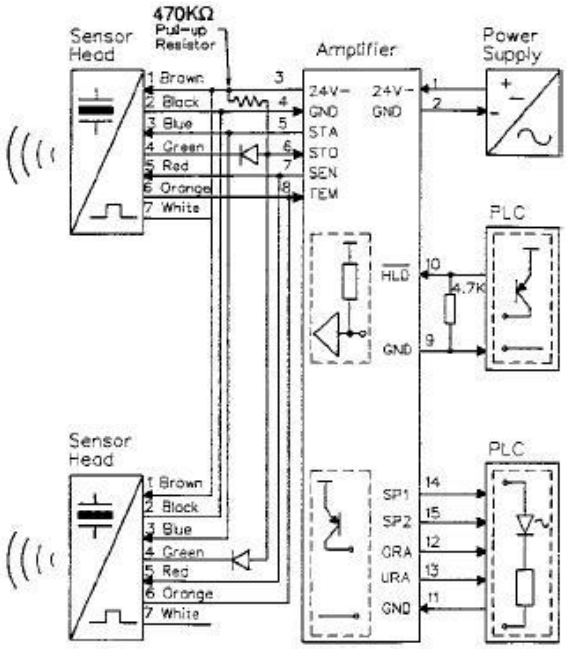


Figura 1.9: Conexión de sensores de ultrasonido en paralelo.

Todas las señales entre el amplificador y los sensores pueden conectarse en paralelo a excepción de la señal de START y la de STOP. El pulso de disparo es generado por el amplificador e inicia el ciclo del sensor. El pulso de parada es generado por el sensor e indica que un eco ha sido recibido. Estos pulsos pueden ser multiplexados desde un sensor a otro mediante relevos o demultiplexando mediante circuitos integrados.

Anillos de sensores.

La configuración en anillo es muy común en los robots móviles hoy en día. Generalmente se trata de un número de sensores dispuestos alrededor del robot formando un perímetro, con el objeto de detectar todo el entorno circundante sea cual sea la posición del robot. Las configuraciones pueden variar desde unos pocos sensores a 24. En la mayoría de ellos el transductor es de tipo POLAROID, o bien su mismo sistema de determinación del TOF. El problema principal que tienen que resolver estas configuraciones se encuentra en la eliminación de interferencias y ruidos entre los transductores que posee el robot y que están emitiendo señales periódicamente. Cuantos más sensores se usen, se reducirá la probabilidad de choque dada la alta capacidad de

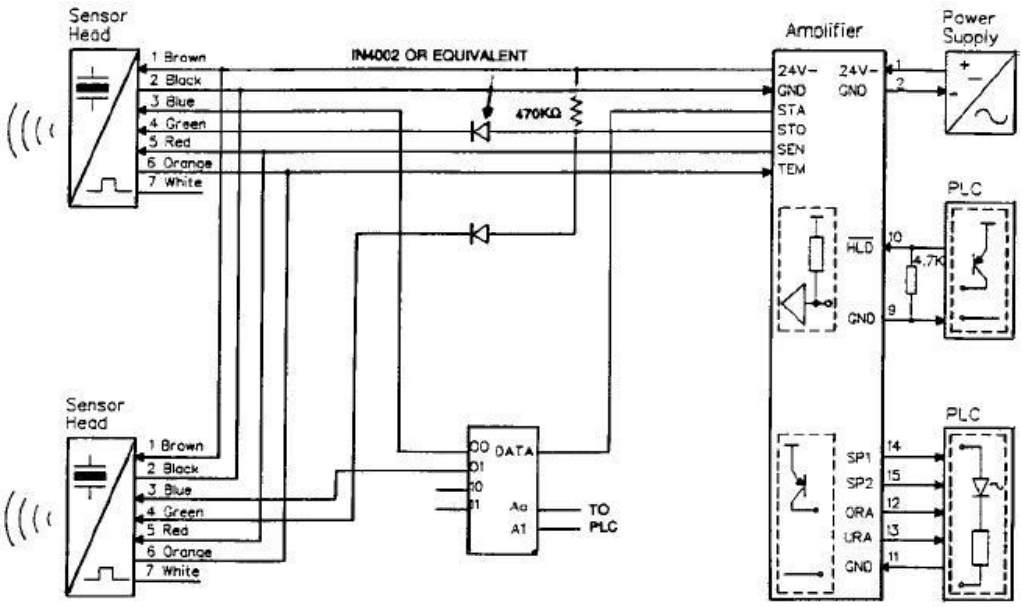


Figura 1.10: Conexión de sensores de ultrasonido en serie.

percibir información del entorno que posee el sistema autónomo, pero por otro lado, se incrementa la cantidad de ruido en el entorno de dos formas:

- Ruido por sensores de otro robot: Cuando en el mismo entorno tenemos varios robots operando, puede darse la circunstancia de que señales de un transductor de un robot, lleguen en forma de eco a otro robot distinto. Estas interferencias suelen ser discretas en el tiempo y ocurren a distancias por encima de los 20 metros.
- Ruido por sensores del mismo robot (interferencias): Este tipo de ruido se da cuando un sensor de un robot recibe como eco la señal que ha emitido otro transductor del mismo robot. Borestein definió "cámino crítico" como aquel camino que lleva a cabo una señal de ultrasonidos desde un sensor a otro u otros, diferente del que lo emitió. En la figura 1.11 se pueden ver dos casos de camino crítico.

En el primer caso se produce camino crítico directo ya que los sensores vecinos al que emite la señal recogen el eco. Pero el segundo caso es un camino crítico indirecto ya que el eco se recoge por sensores no vecinos a quien emite la señal debido a reflexiones en el entorno. Este último caso es el más complejo de resolver.

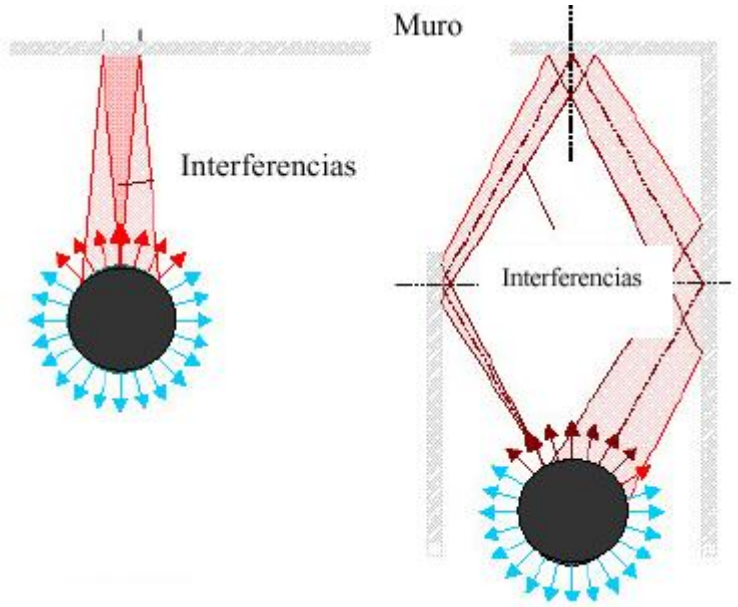


Figura 1.11: Ejemplos de posibles caminos de la señal que provocan interferencias en un anillo de ultrasonidos.

La primera aproximación para solucionar el problema de las interferencias consiste en la comparación de dos medidas consecutivas. En principio dos medidas consecutivas en ausencia de ruido deberían ser idénticas (no totalmente idénticas por la naturaleza discreta de la medida). De esta forma podemos rechazar medidas erróneas por causas externas mediante este mecanismo, sin más que limitar su diferencia a un factor T_δ.

Pero este método no es útil para solucionar el problemas de las interferencias entre sensores. Como se puede ver en la figura 2.6 dos medidas consecutivas debidas a una interferencia también se diferenciaran en poco en la mayoría de los casos, ya que el camino seguido por la señal en ambas ocasiones será el mismo.

Por eso se combina el método de comparación de medidas consecutivas con el de esperas alternadas. Estas esperas alternadas no son más que tiempos de espera introduccidos entre dos disparos consecutivos de un sensor con el objeto de desincronizar las interferencias en otros sensores y conseguir así que dos medidas consecutivas de sensor sean semejantes sólo si son generadas por el propio sensor y no por interferencias de otros. Por ello, se escogen para todo sensor dos tiempos de espera $T_{i,espera,a}$ y $T_{i,espera,b}$. La condición que deben

cumplir los tiempos seleccionados para los K sensores del sistema autónomo es que:

$$||T_{x,espera,a} - T_{y,espera,a}| - |T_{x,espera,b} - T_{y,espera,a}|| > 2T_{\delta} \tag{1.10}$$

Y también que:

$$||T_{x,espera,a} - T_{y,espera,b}| - |T_{x,espera,a} - T_{y,espera,a}|| > 2T_{\delta} \tag{1.11}$$

para permitir comenzar con cualquiera de los dos valores de espera.

Esto debe cumplirse $\forall x, y \in 1..k, x \neq y$.

Por últimó, la estrategia descrita anteriormente se mejora con un método alternado de disparo de los sensores. Esto tiene como objeto evitar las interferencias por camino directo. Del estudio del comportamiento de los sensores se dedujo que para este tipo de interferncia los sensores afectados son los tres vecinos próximos. Por este motivo los sensores se disparan en grupos de cuatro emitiendo sólo uno de ellos cada vez.

Finalmente el disparo de los sensores se produce de la siguiente forma:

- Los sensores 1 al 4 son disparados en los instantes 0,15,30 y 45 ms.
- Los siguientes grupos de cuatro sensores utilizan los mismos intervalos.
- A este periodo de disparo de cada sensor se le debe añadir un tiempo de espera con valores a y b que se suman de forma alternativa a los tiempos de disparo.

De esta forma un sensor se dispara cada 60 ms.

La selección real de tiempos para la prueba se llevó a cabo de acuerdo con los criterios matemáticos que debían cumplir, las restricciones del hardware y el tiempo mínimo que debía transcurrir en el disparo de un sensor de un grupo de cuatro (15 ms). Finalmente se simplifica la prueba con doce sensores justificando esto en la no necesidad de llevar a cabo medidas detrás del robot, sino solo en la parte delantera.

Las pruebas realizadas demostraron que con la metodología adoptada se producían entre 1 % y 2 % de errores, frente al 30 % que se producía en plataformas donde no se tenían en cuenta las interferencias.

Esta solucción fue aportada por Borestein, se llama, EERPUF (Error Eliminating Rapid Ultrasonic Firing), y da una idea de la complejidad y problemas que puede acarrear el usar un conjunto grande de sensores en un sistema autónomo, siempre que se quieran usar todos de forma simultánea.

Anillo de sensores para tratamiento geométrico.

Una opción alternativa a los anillos de sensores, son las configuraciones de varios sensores a distancias conocidas, con el objeto de reconocer alguna característica del entorno (planos, esquinas o vertices). La configuración hardware de este método sería la siguiente. Un sistema sensorial de 32 sensores de ultrasonidos con frecuencias que van desde los 40 kHz a los 200 kHz (también se puede hacer con un solo tipo de sensores). Cada sensor dispone de una targeta de electrónica propia. La señal de dichos sensores es manejada por un DSP TMS320C25, llevando a cabo una adquisición y procesamiento en paralelo de 5 medidas. Este módulo de procesamiento de la señal puede trabajar como esclavo en un sistema VME o directamente conectado a un PC mediante un puerto RS-232. La aplicación al reconocimiento geométrico del entorno se lleva a cabo mediante la introducción en una red neuronal del tiempo de vuelo de la señal estimado de cada uno de los tres transductores que forman un "*plane array*", devolviendo la red una estimación de la distancia y el ángulo que formaban con el sensor central.

1.8.3. Configuraciones para extracción de características.

El conjunto de arquitecturas sensoriales para ultrasonidos tienen en común su diseño orientado a detección de características del entorno. Son sensores formados por varios transductores y que operan como un todo en conjunto. De esta forma el sensor en sí ya no es un elemento aislado como se ha visto hasta ahora, sino un conjunto de dispositivos. El sensor más básico dentro de este grupo es un sensor para la localización y clasificación de características del entorno 2D (planos, esquinas y vértices), que tiene una extensión para 3D (al medir en tres dimensiones se pueden cometer menos errores al fusionar con otros sensores).

Estos sensores consiguen una exactitud alta por medio de un gran ancho de banda de la señal, una estimación óptima del instante de tiempo en que se detecta un eco y por último por el uso de plantillas para modelar la forma

de la señal de eco y una técnica de "*matched filtering* [1]"en su uso.

1.9. Incidencias del medio ambiente.

Temperatura.

La velocidad del sonido en el aire depende de la temperatura. Un sensor interno de temperatura puede adaptar la frecuencia del reloj del contador y la frecuencia de la portadora para ayudar a compensar las variaciones de la temperatura del aire. Sin embargo, las fluctuaciones grandes de temperatura dentro del camino que recorre la onda ultrasónica pueden causar dispersión y refracción de la señal de ultrasonidos, afectando negativamente a la exactitud y estabilidad de la medida. Si se quiere detectar un objeto caliente, habrá que experimentar posicionando el sensor y el objeto en un plano vertical y apuntar a la parte del objeto más fría. De esta manera, se pueden evitar las corrientes de aire caliente y lograr la operación satisfactoriamente.

Presión del aire.

Los cambios normales en la presión atmosférica del aire no tienen efectos sustanciales en la exactitud de la medida. Pero se aconseja no usar sensores de ultrasonidos con presiones de aire bajas o demasiado elevadas.

Humedad.

El efecto de la humedad en la medida es virtualmente insignificante, sólo cambia un 0.07 % para un cambio en la humedad relativa de 20 %. Sin embargo, la absorción del sonido aumenta con el aumento de la humedad. Así, la máxima distancia de medida es reducida muy poco.

Turbulencias en el aire.

Las corrientes de aire, turbulencias, y capas de distinta densidad causan refracción de la onda de sonido. Un eco puede ser producido y la señal debilitada o desviada y por tanto el eco no es recibido. El máximo rango sensible, la precisión de medida y la estabilidad de medida pueden deteriorarse bajo estas circunstancias.

[1]Matched filtering es el proceso de estimación del tiempo de vuelo que se lleva a cabo buscando la máxima correlación entre un eco recibido y un conjunto de plantillas previamente almacenadas

1.10. Errores de medida con ultrasonidos.

La medida de distancias con ultrasonidos está sometida a diferentes fuentes de error, lo mas comunes son los detallados a continuación.

1. Errores de origen natural

    - La velocidad de propagación del sonido en el aire depende de la temperatura. (Ver ecuación [1.2])

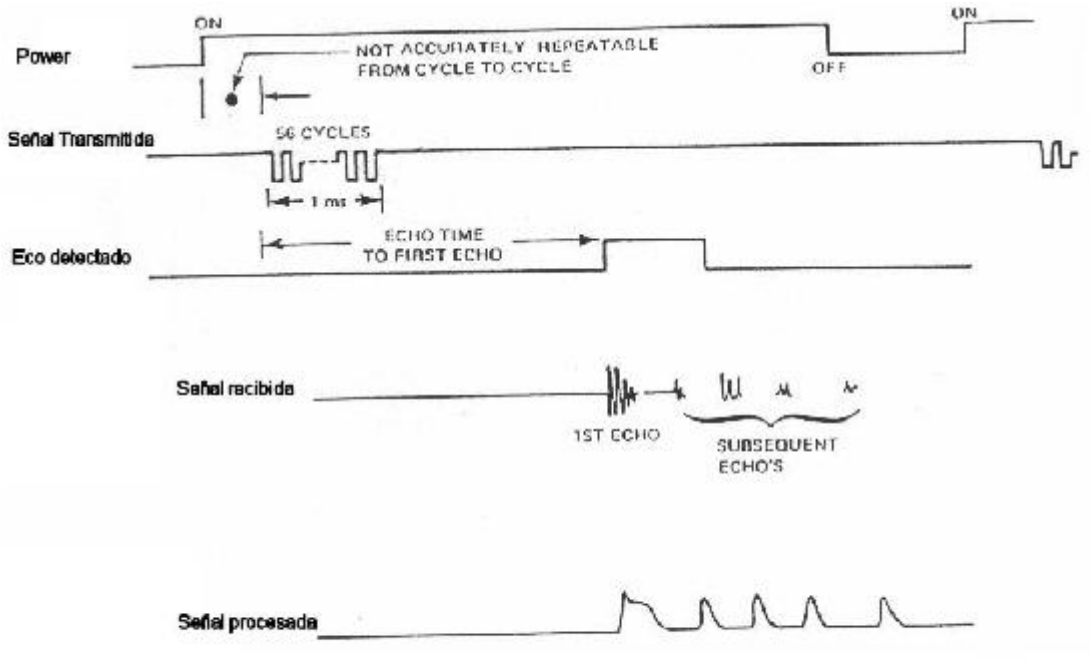


Figura 1.12: Cronograma del proceso de emisión y recepción de una onda ultrasónica mediante el sensor de ultrasonidos. Se aprecia como se emiten varios pulsos y son recibidos de igual manera varios ecos.

    - Según sea la rugosidad de los objetos, estos serán reflectantes o difractantes. Aunque este efecto se corrige en parte con la emisión de un pulso de varias frecuencias, existen objetos que pueden atrapar la onda, devolviendo una distancia mayor de la que en realidad se encuentra.

- El patrón de emisión de la onda no es completamente cónico (Ver figura 1.5), si no que existen unos lóbulos laterales de emisión que pueden provocar un eco.

2. Errores de origen electrónico

   - El amplificador de ganancia variable que contrarresta el efecto de la atenuación en el aire es aproximado por 16 niveles de amplificación. Los puntos de discontinuidad en esta gráfica provocan un error en la corrección.
   - El nivel que determina si un eco es reconocido como tal se realiza mediante la carga de un condensador. Los ecos de objetos cercanos cargan este condensador en tres periodos, sin embargo los ecos de objetos mas lejanos necesitan mas periodos, por lo que aparentarán encontrarse a una distancia mayor.

3. Errores debidos a la posición relativa Sonar-Objeto Según sea la posición relativa de los sónares frente a un objeto, se pueden producir diferentes efectos que pueden hacer que un objeto que esté mas cercano aparezca como mas lejano o viceversa, que sea inapreciable... Las secuencia de gráficos de la figura 1.13 muestra alguno de los errores más comunes.

1.11. Aplicaciones.

1.11.1. Introducción.

Los sensores de ultrasonidos proporcionan al mercado un método de detección eficaz a bajo coste con unas propiedades únicas que no poseen otras tecnologías de detección. Usando un gran variedad de transductores ultrasónicos y varios rangos distintos de frecuencia, un sensor de ultrasonidos puede ser diseñado para resolver muchos problemas de aplicación que no se pueden hacer por su coste elevado o simplemente porque no pueden resolverse mediante otros sensores.

Gran rango de detección: En la detección industrial mediante sensores, muchísimas aplicaciones requieren detección a larga distancia. Los sensores de ultrasonidos detectan a distancias superiores a los cuarenta pies, mientras que los sensores inductivos no lo pueden hacer.

Area de detección ancha: Mientras que algunos sensores fotoeléctricos pueden detectar a largas distancias, carecen de la habilidad de detectar sobre áreas grandes sin usar un número elevado de sensores. La ventaja de los sensores de ultrasonidos es que pueden cubrir tanto áreas anchas como estrechas.

Permiten detectar todo tipo de materiales: Sólamente los sensores de ultrasonidos están capacitados para detectar todo tipo de materiales, sea cual sea su composición. El material detectado puede ser claro, sólido, líquido, poroso, blando, madera, y de cualquier color porque todos son detectados.

Miden distancias sin necesidad de contacto: Se mide el tiempo que tarda el sonido desde que deja el transductor hasta que vuelve a él, por lo que la medida de la distancia es sencilla y exacta con un margen de error de 0.05 %.

1.11.2. Ventajas.

Las ventajas de estos sensores con respecto a otros son:

- Miden y detectan distancias a objetos en movimiento.

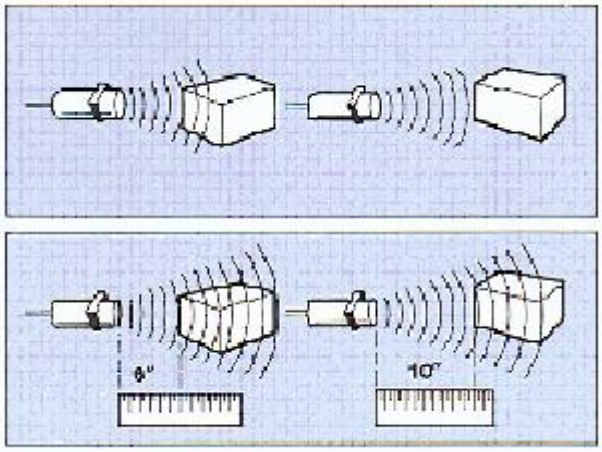
- Es independiente del tipo de material, superficie y color del objeto a detectar.
- Detectan objetos pequeños a distancias grandes.
- Son resistentes frente a perturbaciones externas tales como vibraciones, radiacciones infrarrojas, ruido ambiente y radiación EMI.
- No son afectados por el polvo, suciedad o ambientes humedos.
- No es necesario que haya contacto entre el objeto a detectar y el sensor.

1.11.3. Diferencia entre detección de proximidad y medida del rango.

El sensor de ultrasonidos se puede usar para:

La detección de proximidad: Un objeto pasando por algún sitio dentro del rango de alcance presente del sensor puede ser detectado y generar una señal de salida. El punto de detección es independiente del tamaño del objeto, de su material o del grado de reflexión.Las aplicaciones de detección de proximidad incluyen la detección de la presencia o ausencia de personas y objetos de interés. El sensor detectará el objeto que esté dentro del cono de emisión y reflexión de la señal de ultrasonidos. La detección de objetos situados a grandes distancias requiere que el objeto sea grande o esté orientado de modo que la señal que refleja llegue al ultrasonido.

La medida de la distancia: La distancia precisa a un objeto en movimiento del sensor se mide vía el intervalo de tiempo entre la señal ultrasónica transmitida y la recepción de la reflejada. El ejemplo muestra la detección de un objeto que está a seis pulgadas de distancia del sensor y al rato, al moverse, está a 10 pulgadas. La distancia cambiante se calcula contínuamente.



1.11.4. Campos de aplicación.

Máquinas constructoras: Los sensores de ultrasonidos son usados para el control del movimiento, el control del nivel o para dimensionar o detectar proximidad en máquinas que se usan en construcción. Estas son comunes en aplicaciones dentro de la industria de transformación, impresión, del caucho, de los metales, la textil y otras industrias de manufactura.

Automatización: Los sensores de ultrasonidos reducen los costes de automatización proporcionando métodos de control de tamaños y detección de posición o proximidad de objetos en los procesos de producción. La información proporcionada por los sensores es usada para:

- Aceptar o rechazar objetos basandose en el tamaño, posición o nivel de llenado.
- Para tomar decisiones sobre el camino que deben seguir los paquetes basandose en el tamaño o posición.
- Controlar el flujo de líquidos, sólidos o materiales granulados.
- Indicar cuando un objeto está cercano o en la posición debida.
- Determinar tolerancias.
- Proporcionar una señal de alarma cuando los objetos no están en la posición debida, están a punto de llenarse o vaciarse.

- Para indicar la terminación de un proceso.

Control de procesos: Las aplicaciones comunes incluyen medida del nivel de materiales en un tanque o lata, o controlar la cantidad de material dispersado desde un contenedor. Se puede dar cuenta de la medida del nivel de un tanque a una computadora mediante un red de datos. Las alarmas pueden dispararse debido a un nivel bajo, un determinado nivel establecido, un nivel elevado u otras condiciones.

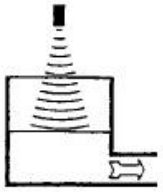
Información y diversión: se puede detecar a gente que se aproxima a una cabina, o se puede controlar su distancia para establecer una determinada respuesta.También se puede usar la detección de personas y objetos en determinados juegos.

Derroche de agua: Se pueden medir los niveles de depósitos de agua para controlarlos. El flujo de agua de un canal abierto se mide para dar informes y establecer un control.

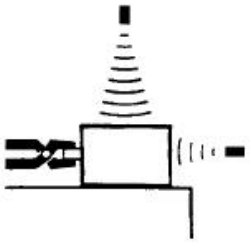
1.11.5. Aplicaciones típicas.

- Para medida del diámetro de un arrollamiento, para control de enrollado o desenrrollado, o para medida de la tensión de una tela. Esto es usado en: La fabricación de papel o tela, la fabricación de gomas o neumáticos, el procesado del acero.

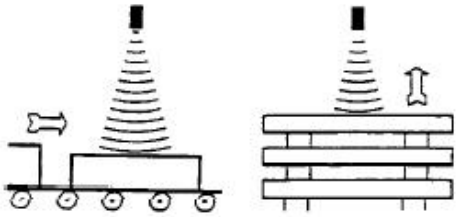
- Para el control de nivel de tanques (tanto de líquidos como de granulados), o para el control del nivel de llenado de botellas o latas. Usado en la industria alimenticia, química o de plásticos.



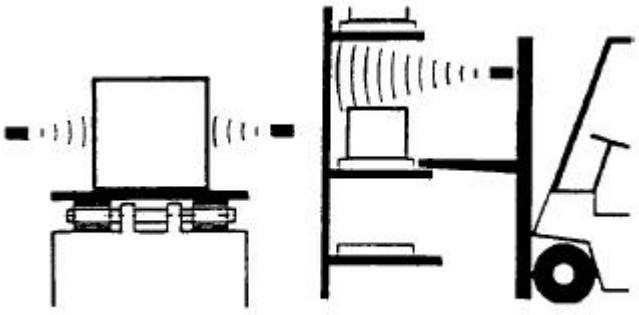
- Para medida de distancia, para el posicionamiento de piezas de trabajo en robots, y para medida de tamaño (dimensionamiento). Se usa en la industria de la automatización, el trabajo con metales y en equipos de ensamblaje.



- Para medida de tamaño o anchura o para el empaquetamiento. Se usa para el manejo de materiales o para trabajar con metales.



- Para la detección de presencia o ausencia, o para detección de partes claras o de cristal o vidrio. Se usa en la industria alimenticia, manejo de materiales o para equipos de ensamblaje.



- Para control de movimiento, detección de personas, seguridad y para muchas otras cosas.

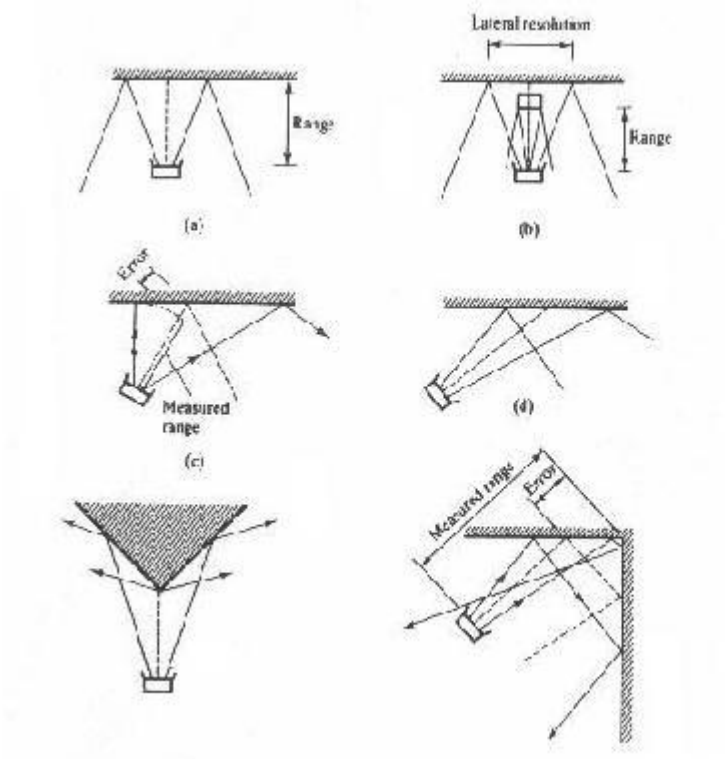


Figura 1.13: Posibles errores debido a la disposición relativa entre el sonar y el objeto. La figura **(a)** muestra una medida de distancia correcta. En la figura **(b)** se muestra el problema de no poder determinar el ancho del objeto que refleja la onda, debido a la apertura del cono. En la figura **(c)** se obtiene una distancia mas corta de la que en realidad existe al objeto, debido a que el eco producido se debe al extremo del cono, no a la parte central. En la figura **(d)** se muestra un objeto inapreciable, ya que los ecos no regresan al punto de emisión. La figura **(e)** muestra el mismo problema para el caso de una esquina. En la figura **(f)** la medida obtenida es mayor de la real debido a los falsos ecos producidos por sucesivos rebotes.

Parte II

Sistemas empotrados.

Capítulo 2

Sistemas empotrados.

2.1. Introducción.La era Post-PC.

Desde ya hace un tiempo se han comenzado a oir noticias relativas al inminente inicio de una nueva era, la era Post-Pc. Tecnológicamente hablando, es posible que ya nos econtremos en esta "nueva era", y posiblemente lo estemos desde hace bastante.

La computerización de dispositivos se encuentra en un estado muy avanzado si tenemos en cuenta que según la IDC (International Data Corporation) se fabrican 2000 millones de microprocesadores cada año y que el 95 % de los mismos va destinado a equipos electrónicos en general y no a ordenadores personales (PC), que se lleva el 5 % restante (Artículo de Rick Lehrbaum publicado en la revista Linux Jornal de agosto del 2000 sobre sistemas empotrados). Lamentablemente, estos dispositivos electrónicos generales, como automóviles, lavadoras, equipos de aire acondicionado, etc. No cuentan con un sistema de interconexión capaz de producir una sinergia, tan drásticamente plasmada en el mundo del PC como lo es la red de redes: Internet.

La era Post-Pc, demanda cinco claves, cinco pilares sobre los que se sustenta, por lo que un punto de estudio o discusión importante, se centra en estimar si estos cinco pilares se encuentran es un estado de madurez óptimo como para soportar el peso de una tecnología global, una nueva era tecnológica.

2.2. Los cinco pilares del crecimiento de los sistemas empotrados.

Internet

Pocos comentarios que no se hayan dicho ya caben en estas líneas.

La red de redes ha sido un revulsivo que ha puesto el concepto de conectividad por delante de muchos otros, demostrando las ventajas de un mundo global, donde el acceso, tanto a personas como sistemas (equipos) se ha transformado en un motor de crecimiento de la humanidad.

Redes de área local (LAN) inalámbricas

El aumento inevitable de los equipos con acceso a red esta demandado no solamente una arquitectura y calidad de red cada vez más potente y fiable

sino además, el crecimiento de soluciones inalámbricas con el fin de evitar el problema físico del cableado.

GNU/Linux empotrado (sistemas abiertos empotrados)

El sistema operativo GNU/Linux y su motor, la licencia GPL se han transformado en una plataforma ideal para el desarrollo de sistemas empotrados. Las versiones de GNU/Linux orientadas a ejecutarse en SOCs (System On Chip), no han tardado en aflorar, siendo el avance en este campo uno de los mayores. El que GNU/Linux no sea dependiente de la arquitectura debido a la multitud de adaptaciones realizadas, su alto grado de escalabilidad, modularidad y flexibilidad, así como su mayor ventaja, su arquitectura abierta, han sido los principales culpables de este avance.

Tipos de interfaz

La multitud de sistemas computarizados que poco a poco comienzan a convivir con nosotros, nos obliga al aprendizaje de multitud de interfaces siendo muchos de ellos una barrera en el uso y adaptación a los equipos.

Poco a poco la tecnología está logrando vencer estas barreras generando interfaces no solamente más sencillas sino unificándolas bajo un mismo aspecto: Interfaces WEB. Pequeños servidores web comienzan a ser un clásico dentro de los sistemas empotrados actuales.

Por otra parte, si bien estos interfaces han facilitado mucho las cosas, todavía queda bastante por avanzar siendo uno de los objetivos a perseguir el reconocimiento del habla como interfaz universal.

Sistemas en un chip

Las companías diseñadoras y fabricantes de circuitos integrados desde siempre han participado en la carrera por obtener dispositivos más potentes, basando esta potencia en una aumento de la velocidad (que trajo consigo la consiguiente disminución del tamaño), así como en el aumento de los recursos internos del dispositivo.

Respecto a esto último, el avance ha sido vertiginoso. En los años 80 un simple controlador de disquetera contenía más del doble de los componentes que hoy en día tiene un ordenador. Todas las decisiones arquitecturales como RISC vs CISC, Hardvard vs Princeton, Firmware vs Wired, etc, se fueron concentrando gracias al avance de la VLSI (Very Large Scale Integration), hasta llegar a ser decisiones internas a un solo chip, ya que en el mismo, es

posible encontrar recursos tales como CPU, memorias, periféricos de comunicaciones, control, potencia, interfaz, etc.

Es decir, los SOC, son una realidad de nuestros días y los llamados microcontroladores tienen gran culpa de ello. De todas maneras, el avance de los SOC, tiene ya una frontera, siendo esta no funcional, sino arquitectural.

Muchos de los diseños que se realizan no se pueden resolver con un SOC, ya que por motivos de optimización de recursos (redundantes o sobrantes) es necesario diseñar en base a circuitos integrados aislados.

Los SOC, con el fin de abordar el problema del diseño de sistemas empotrados se han caracterizado por ofrecer soluciones con una mayor capacidad de interfaz en detrimento de la capacidad de procesador o cálculo.

2.3. ¿Para qué sirven?

- Un sistema embebido está diseñado para realizar funciones que pueden ser peligrosas, repetitivas o que requieran de tiempos de respuesta imposibles de alcanzar para los seres humanos.
- Existen sistemas embebidos para realizar los más variados tipos de aplicaciones usando una amplia gama de tecnologías diferentes.

2.4. Sistemas empotrados.

En primer lugar, resulta significativo el hecho de que cada vez es mayor el número y variedad de sistemas de tiempo real que toman la forma de sistemas empotrados en todo tipo de sistemas de ingeniería y de consumo. El abaratamiento del hardware ha hecho posible la extensión de los sistemas de tiempo real empotrados, que hasta hace poco se limitaban a sistemas de alto coste y alta tecnología, a los productos de uso cotidiano. Así, los sistemas de tiempo real ya no se encuentran únicamente en sistemas de control de aviones o cohetes, control de fábricas o sistemas de telecomunicación complejos, sino que también forman parte, a menudo invisible, de productos como teléfonos móviles, reproductores de discos DVD, televisores e incluso afeitadoras o aspiradoras. Varios estudios de mercado estiman en unos 5000 millones de dólares el mercado de los sistemas empotrados en el 2001, con

una tendencia creciente. Otros estudios cifran el mercado de sistemas empotrados conectados a Internet en el año 2005 en unos 6500 millones de dólares. Los mercados con una tendencia al alza más significativa, según estos mismos estudios son, los de consumo doméstico y telecomunicaciones avanzadas. Es de suponer que esta tendencia continuará durante los próximos 25 años, y que los sistemas empotrados de tiempo real serán cada vez más comunes y permitirán construir sistemas cada vez más autónomos.

Por otro lado, todo hace pensar que en los sistemas del futuro convergerán una serie de tecnologías, cuya evolución hay que tener en cuenta si queremos tener al menos un atisbo de cuales serán sus características más importantes. Algunos de estos campos son:

Comunicaciones.

Los medios de comunicación para sistemas de tiempo real tienen que cumplir un requisito fundamental: es necesario poder predecir la duración máxima del tiempo de transmisión de los mensajes. Esta realidad sólo se ha manifestado recientemente y ha dado lugar a la aparición en los últimos años de protocolos de comunicación con tiempo de transmisión determinista. Aunque el motivo principal de estos desarrollos ha sido la necesidad de implementar sistemas de tiempo real distribuidos en la industria del automóvil, se ha extendido rápidamente a otros campos de aplicación. Es de esperar que esta tendencia continúe y que en un futuro próximo se generalicen los protocolos de comunicación con comportamiento temporal previsible, junto un mayor ancho de banda y tolerancia de fallos. El desarrollo de protocolos de comunicación por radio, además, facilitará previsiblemente el desarrollo de sistemas empotrados portátiles.

Ingeniería de software.

El abaratamiento del hardware y el incremento de la densidad de integración facilita la realización de sistemas de tiempo real cada vez más complejos. Es de preveer que esta tendencia continúe en un futuro próximo, lo que hace necesario disponer de métodos y herramientas adecuados para desarrollar sistemas de tiempo real complejos, teniendo en cuenta además que la dificultad de desarrollar este tipo de sistemas es mayor que en el caso de sistemas convencionales. Esto se debe a que hay que considerar los requisitos temporales y de fiabilidad, ya que los fallos de estos sistemas pueden causar cuantiosos daños materiales, e incluso en ocasiones la pérdida de vidas humanas.

Por este motivo, resulta preocupante la visión, ingénua pero frecuente,

de que el desarrollo de sistemas empotrados se simplifica utilizando sin más los métodos de desarrollo de software que han dado buenos resultados en otros campos de aplicación, o que hacer sistemas de tiempo real consiste únicamente en hacer sistemas más rápidos o más eficientes.

Para poder desarrollar este tipo de sistemas de forma que satisfagan los requisitos de determinismo temporal y fiabilidad que les son inherentes manteniendo unos costes razonables, es absolutamente necesario disponer de técnicas de ingeniería de software más avanzadas. En este sentido, se dispone de un buen bagaje de técnicas de diseño y análisis desarrolladas en su mayor parte en los últimos diez años, que se puede utilizar de manera efectiva para conseguir estos objetivos. No obstante resulta preocupante el hecho de que en algunos desarrollos recientes de lenguajes de programación o de sistemas operativos se hayan ignorado casi completamente los avances tecnológicos realizados en los últimos años en este campo.

Inteligencia artificial.

La complejidad de las aplicaciones de sistemas de tiempo real del futuro hace pensar que será necesario emplear técnicas de inteligencia artificial para adaptar su comportamiento a los cambios del entorno, aprender y emplear técnicas de toma de decisiones, y para aumentar su autonomía y prestaciones. Estas técnicas ya se utilizan en muchos tipos de sistemas informáticos, pero la satisfacción de requisitos temporales presenta dificultades adicionales que no siempre se sabe resolver de manera adecuada. El desarrollo de técnicas que permitan compatibilizar los métodos de la inteligencia artificial con un comportamiento temporal previsible dará todavía trabajo a la próxima generación de investigadores.

Sistemas distribuidos.

Es previsible que la mayoría de los sistemas de tiempo real del futuro sean distribuidos. La necesidad de utilizar diversos computadores estará motivada por la necesidad de disponer de una gran potencia de cómputo, por la distribución física de estos sistemas y por la necesidad de hacer que toleren una mayor cantidad y variedad de fallos. Problemas como la sincronización de los relojes de los diversos procesadores, el determinismo temporal y la distribución dinámica de la carga son algunos aspectos que deberán ser solucionados de forma adecuada.

Tolerancia de fallos.

La necesidad de disponer de sistemas fiables y que puedan funcionar de

manera contínua es cada vez más frecuente, y es previsible que se acreciente en el futuro próximo. Para tal fin, las técnicas de tolerancia de fallos, tanto de hardware como de software, deberán tener en cuenta los requisitos temporales, de forma que la recuperación y el tratamiento de los fallos de hardware y de software no impliquen una degradación inadmisible del comportamiento temporal del sistema.

2.5. Aplicaciones de los Sistemas de Tiempo Real dentro de 25 años.

La previsión consiste en que los sistemas de tiempo real serán una tecnología de base para desarrollar sistemas de control empotrados con una serie de características comunes. Serán autónomos, en el sentido que tendrán prefijadas sus funciones y las realizarán sin intervención humana directa. Para ser capaces de realizar esta funciones, será imprescindible disponer de sistemas sensoriales avanzados que les permitan conocer el estado de los elementos del entorno, según sea necesario. Dispondrán de capacidades de comunicación inalámbricas, lo que les permitirá comunicarse con los humanos, entre ellos y con centros avanzados de control. Tendrán capacidades de aprendizaje y de adaptación a entornos cambiantes.

Los dispositivos generales descritos tendrán una aplicación específica en distintos campos, que se analizan a continuación con algo más de detalle.

Transporte.

Uno de los campos de aplicación más importantes de los sistemas de tiempo real ha sido tradicionalmente el control de los medios de transporte como aviones, trenes, etc. Una de las tendencias más interesantes en relacción con este tipo de sistemas es la de conseguir un funcionamiento autónomo de los mismos. Esto ya se ha conseguido en algunos trenes con recorridos sencillos y velocidad limitada, como los que circulan en algunos aeropuertos o en algunas líneas de metro. Obviamente, la complejidad es mayor en el caso de otros medios de transporte como los aviones, trenes o automóviles. Los avances que se están produciendo en sistemas de navegación, sensores e inteligencia artificial permiten esperar un progreso notable en aplicaciones como el desprgue, aterrizaje y navegación de avione, o en el funcionamiento automático de los sistemas de control y circulaci´n de trenes. En cuanto a los automóviles, en la actualidad se están investigando métodos de navegación que permitan

conseguir formas limitadas de conducción automática en autopistas. Para lograr este comportamiento, los sistemas empotrados en loscoches deberán de disponer de sensores inteligentes y muy eficaces para detectar cualquier desviación del sistema, con objeto e corregirla a tiempo. Los vehículos estarán interconectados entre sí y con centros de control. Esto permitirá avisar con tiempo de maniobras y adaptar el comportamiento de los vehículos a las circunstancias del tráfico.

Fabricación automática.

La utilización de las tecnologías descritas en la industria permite esperar el desarrollo de fábricas automáticas. Un conjunto de robots y máquinas especiales se encargarán de fabricar los productos con mínima intervención de operadores. La capacidad de adaptación y reconfiguración de estos componentes podría permitir cambiar las características del producto fabricado en un tiempo muy reducido. De esta manera aumentará la flexibilidad de esta instalaciones y será posible el desarrollo de productos a la carta.

Los componentes de una fábrica podrían estar conectados a los centros de planificación mundiales de la compañía, lo que permitiría ajustar la cantidad y el tipo de productos a fabricar.

Aplicaciones domésticas.

La configuración de los hogares es previsible que cambie en este lapsdo de tiempo. Actualmente se trabaja en redes de comunicación para el hogar, con el objetivo de interconectar los dispositivos. En algunos casos, esta meta se puese conseguir en un espacio de tiempo pequeño. No parece descabellado pensar en la conexión de cámaras de seguridad o de vigilancia del cuarto de los niños que se conectan a una red de área local y que permiten desde el televisor digital observar lo que captan. De la misma manera parece factible interconectar todos los elementos de la casa, de forma que se muestre un mensaje de aviso en la televisión cuando tremine la lvadora o cuando el frigorífico solicite la revisión de las 1000 horas. La aplicación de robots autónomos en el hogar podría resultar en obviar toda esa serie de labores tediosas y necesarias que hay que realizar a diario en cualquier hogar.

CONCLUSIÓN.

Teniendo en cuenta la velocidad con que los avances tecnológicos han cambiado nuestras vidas durante los últimos años, es dificil predecir cómo será este cambio en los próximos 25 años.

Parte III

Fundamentos del procesador ajile80 y JStamp

Capítulo 3

Java y el tiempo real.

3.1. Introducción.

El mercado de los sistemas empotrados y de tiempo real ha estallado en la era "POST-PC", especialmente gracias a internet. Los sistemas empotrados de tiempo real están llegando a ser comunes en mercados como el de las telecomunicaciones, la automatización industrial, control de hogares, sistemas automóviles e instrumentación médica. El contenido software de estos aparatos está evolucionando y creciendo,poniendo especial interés y esfuerzo en el desarrollo de recursos puesto que son muy escasos. El desarrollador de sistemas de tiempo real empotrados se encuentra también con un entorno de trabajo extremadamente heterogéneo, con una gran cantidad de procesadores, sistemas operativos y tipos de periféricos. De este modo, los ingenieros están cada vez más interesados y estudian más las tecnologías Java, para proporcionar un entorno de desarrollo para sistemas empotrados de tiempo real más productivo y portable. Estas tecnologías incluyen el lenguaje de programación orientado a objetos Java, las Java Virtual Machines, y la gran cantidad de librerías de clases de ejecución.

3.2. La plataforma JAVA.

La plataforma Java debutó en 1995 como un entorno de trabajo y de lenguaje server, aunque fue originalmente desarrollado como un lenguaje de programación orientado a objetos para sistemas empotrados. La plataforma neutral, el modelo de objetos simplificado, las fuertes nociones de seguridad y garantía, y el soporte multihilos de la plataforma Java proporcionan muchas ventajas para una nueva generación de sistemas empotrados de tiempo real que funcionan a través de la red.

3.2.1. El lenguaje de programación Java.

El lenguaje de programación orientado a objetos Java, con su sintaxis similar al lenguaje C y su modelo de objetos simplificado, es más fácil de dominar que C++, y ha demostrado ser un 25 % - 40 % más efectivo. La inexistencia en Java de los punteros y el direccionamiento automático de memoria son puntos frecuentemente citados como factores clave en el progreso del software en cuanto a productividad y robusted se refiere. El lenguaje

Java también soporta multihilos y la palabra sincronización proporciona un significado particular y elegante de ejecución mutua y exclusión.

3.2.2. La máquina virtual Java.

La ejecución de un programa Java es soportada por una máquina virtual (JVM) estandar donde los programas son compilados. Este diseño permite a Java ser portable puesto que los programas Java pueden ejecutarse en cualquier sistema que soporte una JVM. Además, el aparato donde se tiene implementado el sistema está aislado de la propia aplicación por la JVM, se este modo se aumenta la seguridad y garantía del código.

El conjunto de instrucciones de la JVM está basado en una arquitectura de 32 bits con un número de instrucciones únicas, así como invocación de métodos virtuales con detección de bloqueo. En un entorno de ejecucción Java típico, las aplicaciones portables o applets compilados en código de JVM son interpretadas por una implementación software JVM, Just-In-Time (JIT) compilando al código máquina nativo o directamente son ejecutados por un microprocesador Java.

Un microprocesador Java es más eficiente desde el punto de vista del espacio y el tiempo para ejecutar el código Java. Los interpretes de código son lentos y requieren memoria para almacenar el código del interprete. JIT´s requiere áreas de memoria que varíen de tamaño para soportan la traducción de código y la ejecución en un entorno de JIT sufre desde efectos de pérdida de memoria caché si se transfiere el control hasta bloqueos de código no traducido. Sin embargo, generar código como un procesador no es una tarea fácil por eso el conjunto de instrucciones de la JVM es complejo, más incluso que el conjunto de instrucciones de los computadores tradicionales como son la familia x86 de Intel y la familia 680x0 de Motorola.

3.2.3. Aplicaciones de Java en objetos móbiles.

La "J2ME Connected, limited Device Configuration (CLDC)" especifica un subconjunto de Java 2 para ocupar los recursos de memoria (menos de los 512 KB disponibles para el entorno Java) con baja velocidad y conectividad intermitente con la red, tales como teléfonos móbiles y asistentes personales digitales. El "Connected Device Configuration (CDC)" es tomado como

punto de referencia para muchos aparatos, tales como set-top boxes o aplicaciones de red "plug-in-the-wall". Junto con estas configuraciones estandar, un numeroso grupo de perfiles de mercado verticales se pueden definir a partir de J2ME. Un perfil del J2ME particularmente importante es el "Mobile Information Device Profile (MIDP)", que entre otras características importantes posee la de definir un conjunto mínimo de API´s gráficas para aparatos de información como por ejemplo los teléfonos móbiles, los asistentes digitales personales, etc. La arquitectura software de una aplicación J2ME típica es la mostrada en la figura 3.1:

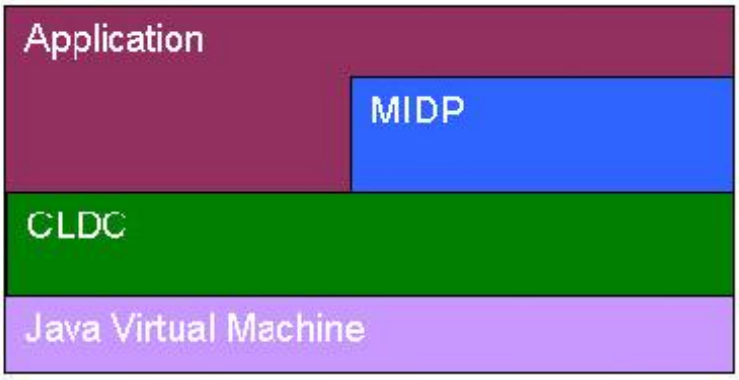


Figura 3.1: Arquitectura Software del J2ME.

3.2.4. Conclusión.

El entorno Java, con su plataforma neutral, su modelo de objetos simplificado, sus fuertes nociones de seguridad así como su soporte de hilos múltiples,

proporcionan muchas ventajas para una nueva generación de sistemas empotrados y de tiempo real. Sin embargo, el gran tamaño, el comportamiento no determinista y la pobre representación de la primera generación de implementaciones Java en sistemas empotrados han influido negativamente en la aceptaciónde Java en el mundo de los sistemas empotrados y de tiempo real. aJile Systems ha desarrollado una implementación hardware de bajo coste de una máquina virtual Java que hace una realidad el uso de Java en sistemas empotrados y de tiempo real. El hardware de aJile proporciona un soporte directo para el conjunto de instrucciones de la JVM y el modelo de hilos (threads), eliminando la necesidad de un interprete de Java o compilador "'Just-In-Time", así como el tradicional sistema operativo de tiempo real (RTOS).

Esta tecnología hardware de aJile soporta múltiples JVM ejecutándose al mismo tiempo en la CPU, aumentando la seguridad, garantizando el espacio en memoria y porción de tiempo para las distintas aplicaciones Java. Esto combinado con la segunda edicción de Java (J2ME) y unas herramientas de construcción, hacen de esta tecnología una plataforma eficiente para el desarrollo de aplicaciones empotradas de tiempo real desarrolladas enteramente en Java.

3.3. 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 real" y no un sis-

tema "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).

3.3.1. Origen de los sistemas de tiempo real.

La primera propuesta para que un computador trabajara en tiempo real, como un sistema de control, fue publicada en 1950. Esta propuesta mostró un computador con bucles feeback y feed-forward. Fue supuesto que serían usados elementos de computación analógicos pero los digitales no fueron excluidos. El primer computador digital desarrollado específicamente para control en tiempo real fue para operaciones aéreas. Por 1954 un computador digital Digitrac fue usado con éxito para proporcionar un vuelo automático y para sistemas de control de armas.

En 1960 empezaron a emerger los sistemas operativos en tiempo real para aplicaciones industriales. También aparecieron los compiladores de proceso fortran. La llegada del microprocesador en los 70´s y el aumento de la velocidad de la memoria empezaron a forzar el aumento de atención en los problemas de la escritura correcta y el control fiable por computador del software.

3.3.2. Definición de sistemas de tiempo real.

La definición de sistemas de tiempo real parece variar entre los distintos grupos, tales como: vendedores, pranticantes, académicos, desarrolladores, etc. Algunas definiciones son las siguientes:

- Según el diccionario de computación Oxford:" *Un sistema en el cual, el tiempo en el que se obtiene la respuesta es significativo. Esto normalmente es, porque la entrada corresponde con algún movimiento de mundo físico, y la salida tiene que estar relacionada con ese movimiento. El retraso entre la entrada y la salida debe ser suficientemente pequeño para que sea aceptable."* Esta definición cubre diferentes tipos de sistemas, desde estaciones de trabajo que se ejecutan bajo sistemas operativos UNIX donde el usuario espera obtener una respuesta dentro de pocos segundos, hasta sistemas de control de aviones donde se debe obtener una respuesta en un tiempo determinado y cualquier fallo o retraso puede causar la pérdida del control y la posibilidad de la muerte de los pasajeros.
- La definición del Journal of Systems and Control Engineering: "*Los sistemas de tiempo real son aquellos que deben producir respuestas correctas dentro de un tiempo límite definido. Si las respuestas sobrepasan este tiempo es posible que se den degradaciones y funcionamientos extraños en los resultados."*
- Otra definición alternativa es: "*Los sistemas de tiempo real leen las entradas de una planta (sistema físico a controlar, por ejemplo: un robot, un proceso automatizado de una empresa, una cámara digital, etc.) y mandan señales de control a la planta en tiempos determinados para las consideraciones operativas de la planta, no a los tiempos limitado por las capacidades del sistema computador.*"

3.3.3. Clasificación.

Los sistemas de tiempo real y los computadores empotrados se interconectan con el entorno en el que trabajan mediante un amplio rango de interfaces de periféricos que reciben y envían estímulos. Los procesos externos operan en sus propias escalas de tiempo y al computador se le exige que opere en tiempo real si las operaciones de los procesos externos se llevan al computador.

Se debe definir la sincronización entre los procesos externos y las acciones internas del computador. Esto se puede hacer por:

- El periodo de tiempo en el que se le dice al sistemaque responda que se base en el reloj("clock based"), y entonces las operaciones llevadas a realizar al computador se realizan de acuerdo a un tiempo planificado.
- Acciones, que tienen que realizarse no en un tiempo determinado o intervalo establecido, pero en respuesta de algún evento en cuyo caso es llamado a ser "event based". Por ejemplo, apagar un congelador cuando su temperatura baje de una mínima establecida y encenderlo cuando la temperatura supere el máximo establecido. Los sistemas basados en eventos("event based") emplean normalmente interrupciones para informar al computador de la acción que es requerida. Algunos sistemas usan el *polling*, que consiste en preguntar periódicamente a los sensores si alguna acción es requerida.
- Un método interactivo ("interactive based") que consiste en definir de una manera menos fuerte el parentesco entre las acciones del computador y el sistema. El requisito es que se complete en el ordenador un conjunto de operaciones dentro un tiempo predeterminado. Un cajero automático tiene una sincronización interactiva puesto que requiere la respuesta interactiva del usuario que debe introducir su clave y operaciones que desea realizar para obtener información sobre el estado de su cuenta o realizar una transacción, y esto debe de hacerlo dentro un determinado tiempo límite(si no se obtiene ninguna respuesta por parte del usuario en 20 segundos, se le devolverá la tarjeta).

3.3.4. Las limitaciones de tiempo en los sistemas de tiempo real.

Los sistemas de tiempo real se pueden dividir en dos categorías principales según sus limitaciones de tiempo.

- *Hard real-time*:
- *Soft real-time*:

3.4. Real-Time Java.

Los sistemas de tiempo real se encuentran en aplicaciones de sistemas empotrados y en otras aplicaciones que requieren un determinado comportamiento en el tiempo.

3.4.1. Introducción.

Algunas de las características de la especificación del lenguaje Java, particularmente el comportamiento no determinista del recolector de basura y el manejo de hilos, han impedido la adopción de Java en la industria de los sistemas de tiempo real y aplicaciones software. Esta limitación fue entendida tanto por la industria como por las instituciones académicas, por eso se organizó un grupo de trabajo dentro del Instituto Nacional de Estandares y Tecnología (NIST) para discutir y proponer un proyecto: "Real-Time Specification for Java (RTSJ)". Este requerimiento llevo a la formación de un grupo experto dentro de "Java Community Process(JCP)" para realizar el proyecto de la especificación de Java para tiempo real.

3.4.2. La debilidad de Java para las aplicaciones de tiempo real.

Había algunos puntos en la especificación del lenguaje Java que frenaron su adopción dentro de la industria del control de tiempo real. Los principales problemas fueron:

- El controlador de eliminación de basura de la memoria dinámica podía interrumpir la ejecución de aplicaciones por intervalos de tiempo impredecibles. Esto era debido al comportamiento no determinista del recolector de basura, que podía elegir donde borrar los objetos de la memoria dinámica y en un orden arbitrario.
- La planificación de hilos. Como la JVM usa el planificador del servidor del sistema operativo, el sistema operativo debe ser capaz de planificar en tiempo real, pero esto no lo cumplen todos los servidores donde está disponible la JVM.

Es vital en las aplicaciones de tiempo real tener un planificador de eventos totalmente determinista, no que lo haga de manera indeterminada. El riesgo asociado a estos dos aspectos superaba las otras ventajas de Java.

3.4.3. Principios a seguir.

El "Real-Time Java Expert Group (RTJEG)" estableció un conjunto de principios a seguir por los diseñadores de la RTSJ. Estos son:

- RTSJ no debe incluir ningún requerimiento que limite su implementación a una plataforma, versión o entorno Java en particular.
- Cualquier modificación, no debe, bajo ninguna circunstancia, impedir la ejecución del software que no esté escrito correctamente en tiempo real, en cualquier implementación de una JVM.
- Incluso cuando la dificultad en la especificación de los parámetros de tiempo real para una plataforma independiente sea reconocida, los principios de W.O.R.A(Write Once Run All) deben ser perseguidos en la mayor medida posible.
- La especificación debe también incluir las características actuales de los sistemas de tiempo real y permitir la inclusión de otras más avanzadas en el futuro.
- La principal característica que debe perseguirse es la previsión incluso en el costo de una actuación de propósito general.
- Cualquier modificación no debe incluir una extensión del lenguaje que conduzca a un requerimiento de modificación del compilador y de aquí a un aumento de la probabilidad de descargas frecuentes.

3.4.4. Diseño e implementación.

Hay siete áreas funcionales en las que Real-Time Java presenta intensificaciones: planificación y despacho de hilos (thread Scheduling and dispatching), manejo de memoria, sincronización y reparto de recursos, manejo de eventos asíncronos, transferencia de control asíncrono, terminación de hilos y acceso a la memoria física.

- **La planificación y despacho de hilos**: Un esquema fijo y previsor con una prioridad de orden de entrada tipo FIFO(First Input, First Output) se define como base de la politica de planificación. Además, un conjunto de de características y una API son definidas para soportar una variedad de otras políticas y mecanismos de planificación.

- **Manejo de memoria**: Tanto las ventajas como los inconvenientes del sistema de recolección de basura son conocidos. Por lo tanto, dos métodos simultánmeos son usados para cubrir los inconvenientes : se introduce un nuevo régimen de memoria sin recolección de basura y también se dispone de interfaces estandar para instalar recolectores de basura de tiempo real.

- **Sincronización y reparto de recursos**: La emulación de máxima prioridad y los protocolos de herencia de prioridad son establecidos para obligar las inversiones de prioridad en los controles usados para implementar keywords sincronizadas. Por defecto se proporciona la herencia de prioridades con una prioridad máxima opcional.

- **Manejo de eventos asíncrono**: Los objetos de sucesos asíncronos son añadidos para representar eventos asíncronos que se espera que ocurran. Estos objetos son entonces mapeados por controladores de eventos asíncronos para que puedan ejecutarse siempre que ocurra un suceso. Estos controladores son ejecutados como hilos de tiempo real. Es fácil programar un sistema estructurado de conducción de eventos, tanto que, cada suceso es atendido por un hilo creado por ese suceso particular y planifica los atributos para cada suceso ayundando así al planificador. El tiempo desde que ocurre un suceso hasta que éste es atendido es un overhead en la sensibilidad de tiempo real. La creación de hilos es lenta. Es un servicio de colocación de recursos, y los programadores de tiempo real evitan la colocación de recursos cuando están interesados en el tiempo.

- **Transferencia de control asíncrona**: La semántica del método de interrupción (interrupt()) se expande permitiendo que ocurra en cualquier lugar del código en vez de sólamente en ciertos bloques de llamadas. Por tanto, se añade una excepción de interrupción asíncrona para especificar donde puede un método recibir esta excepción. El control de transferencia asíncrono es un mecanismo que permite a un hilo lanzar una excepción en otro hilo.

- **Terminación de hilos**: Se define un medio para permitir la terminación ordenada de un hilo. Usa el mecanismo de sucesos asíncronos

para ejecutar el método *interrupt()* del hilo. La nueva definición semántica del hilo permite la correcta terminación del hilo.

- **Acceso a la memoria física**: Se define un nueva área de memoria que puede ser mapeada en una dirección física fija. Esto permite la comunicación directa entre hilos y periféricos en Java de tiempo real.

The Real-Time Specification for Java (RTSJ) holds the distinction of being the first Java Specification Request (JSR 1) of the 171 submitted to the Java Community Process (JCP) so far. But order of submission doesn't imply order of completion; although many other specifications have passed through the JCP, the RTSJ was just finalized in November 2001. Sun is not providing a reference implementation of the specification – that task was delegated to TimeSys. RTSJ includes such features as real-time threads, asynchronous events, interruptible nonblocking I/O (input/output), access to physical memory, scheduling, and timers. One of many Sun attempts to address embedded applications, RTSJ joins the ranks of aborted efforts such as EmbeddedJava, PersonalJava, and PicoJava as well as successful efforts such as J2ME (Java 2 Platform, Micro Edition). Whether RTSJ survives and where it will fit in with other Sun offerings remains to be seen. Past experiences in this area suggest that Sun has a tough hill to climb to succeed with RTSJ. Although many companies offer real-time solutions for Java – aJile Systems, esmertec, NewMonics, and Zucotto Wireless, for example – none of these vendors support RTSJ in their products. In fact, most embedded VM and hardware vendors seem to focus most of their efforts on J2ME and have no plans to immediately jump on the real-time Java bandwagon. Since these companies already support some real-time capabilities, they see no compelling reason to immediately support the new RTSJ. Many are considering RTSJ support for the future (in the next 12-18 months), but most are waiting to see market demand before committing to this new API. The exception is aJile Systems – aJile participated in the RTSJ expert group and is currently working on an RTSJ implementation for its aJ-80 and aJ-100 chips.

3.4.5. La "Real Time Specification for Java".

La naturaleza dinámica del entorno de ejecución Java es una de las más potentes dentro de los entornos de trabajo tradiccionales y el servidor mundial. Sin embargo el no determinismo introducido por la recolección y eliminación de basura, la resolucción de las clases en tiempo de ejecucción, etc,

es un verdadero problema para los desarrolladores de tiempo real. Por eso, el grupo de expertos de Real-Time for Java (del que aJile es miembro) fue formado en Marzo de 1999 bajo el "Java Community Process" para crear la especificación de Java para tiempo real (RTSJ). La RTSJ proporciona intensificaciones para la especificación del lenguaje Java y la especificación de la JVM en siete áreas clave: programación y despacho de hilos (thread Scheduling and dispatching), manejo de memoria, sincronización y reparto de recursos, manejo de eventos asíncronos, transferencia de control asíncrono, terminación de hilos y acceso a la memoria física. Como ejemplo de estas intensificaciones, la RTSJ define nuevos tipos de área de memoria separados del área de memoria dinámica de Java, incluyendo la "*InmortalMemory*" y la "*ScopedMemory*". Los hilos de Real-Time pueden crear objetos en estas áreas de memoria de la misma manera que en otras (con el operador "new"), pero desde que estos objetos no residen en el área de memoria dinámica de Java, el acceso a los objetos no sufre de las pausas debidas al no determinismo por recolección de basura.

Capítulo 4

Fundamentos del procesador ajile80 y JStamp

4.1. Introducción a Java.

Java es un lenguaje que necesita ser interpretado por una máquina virtual local para poder ser ejecutado en un computador. Esto es lo que hace a Java una máquina y plataforma independiente. Por eso, cuando se tiene la máquina virtual específica para la computadora que se está usando, el código Java será interpretado por esa máquina virtual y el programa se ejecutará sin ningún tipo de problema.

Para cualquier programa dado, el compilador basicamente toma el programa escrito en lenguaje de alto nivel y lo traduce en código ensamblador. Este código en lenguaje ensamblador es proporcionado al ensamblador que lo transforma rápidamente en código de bytes y será ejecutado por el hardware. El código de bytes es básicamente el lenguaje de 0´s y 1´s que constituye el único nivel de comunicación que entiende el hardware de una máquina.

El hecho de que exista un paso intermedio como es la máquina virtual tiene sus ventajas y sus inconvenientes. Las ventajas son que incluye una máquina y una plataforma independiente del computador en el que se use el programa. La mayor desventaja es el factor velocidad. En programas grandes donde el código Java es traducido vía la máquina virtual, la caída de velocidad es significativa. Este paso podría eliminarse si el procesador fuera capaz de interpretar el código Java directamente. Pero esto no es práctico porque entonces se necesitarían distintas máquinas virtuales para distintas aplicaciones.

En el JStamp, el código de bits usado por el procesador ajile, es el código de bits producido por el compilador de Java en un archivo .class, que es traducido más tarde en código binario por el programa JemBuilder y cargado en la placa JStamp usando otro programa llamado Charade. Esto significa que el código Java es directamente interpretado por el procesador... en este caso, el microprocesador aj-80.

4.2. ¿ Qué es JStamp ?

JStamp es un nuevo producto de Systronix, en simples términos, es un computador. Como la mayoría de los computadores, tiene una CPU, RAM, ROM, alimentación, y un bloque de entrada/salida. Aparte de esto, JStamp es distinto que los demás computadores. La mayor diferencia es obviamente el

tamaño (1 pulgada por 2 pulgadas), pero quizas su diferencia más significativa es que se programa enteramente en Java. Este no debe ser confundido con los sistemas que obligan a programar en Java, aunque se ejecuta en el nivel nativo. En JStamp, Java es el nivel nativo.

El hecho de que Java sea el nivel nativo hace que JStamp sea muy rápido, no existe una capa interprete entre el código Java escrito y el procesador. Esto permite al JStamp ser muy pequeño. No es necesario la existencia de memoria extra para almacenar el interprete de Java. Esto significa que en JStamp se ejecuta **real Java** (JStamp ejecuta el código byte de Java). No existe tiempo de compilación, ni de generación, ni de ejecución o cualquier tiempo de traducción del código Java en código ensamblador de la máquina.Esto lleva al Slogan del JStamp, que dice: **JStamp is real fast, real small,real Java.**

Figura 4.1: El Módulo del JStamp.

La **CPU** del JStamp es el procesador aJ-80 de aJile Systems. El aJ-80 se programa enteramente en Java. El código de la máquina virtual de Java estandar es el conjunto de instrucciones nativo. El aJ-80 puede trabajar a una frecuencia máxima de operación de 80MHz. Es posible configurar el JStamp para funcionar a distintas velocidades del reloj, tan lento como 7.3728MHz o tan rápido como la máxima frecuencia de operación (80MHz). Debido a que el material usado en el JStamp tiene una frecuencia de 7.372MHz, la máxima velocidad del reloj interno será de 73.72MHz. La potencia de uso del JStamp es proporcional a esta velocidad y la mínima potencia que consuma dependerá de lo lento que se pueda programar los contadores, esto es una característica a tener en cuenta frente a la duración de las baterías usadas.

El **aJ-80** tiene un número de entradas/salidas configurables. Este incluye: dos UARTs, un controlador de interfaces perifericas en serie (SPI), tres con-

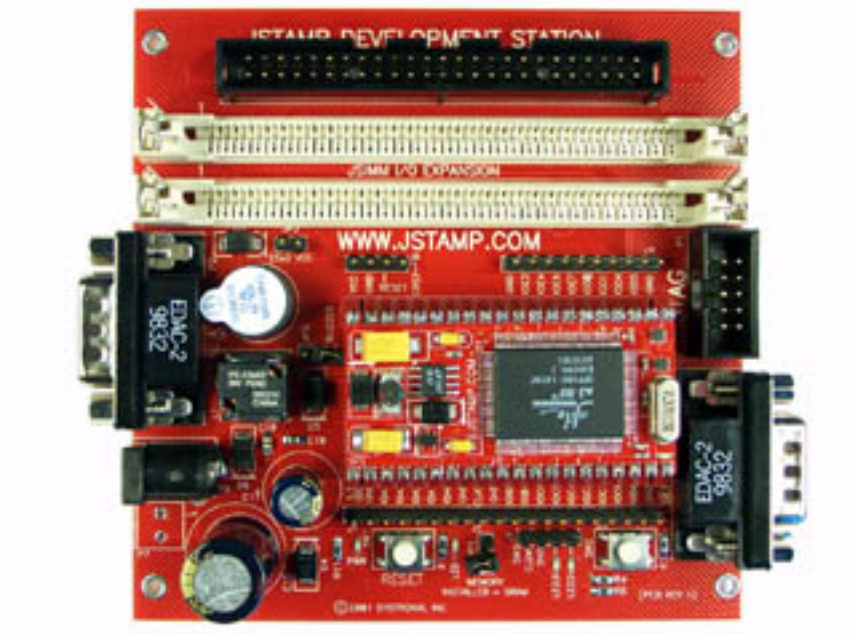


Figura 4.2: La estación de desarrollo JStamp.

tadores de 16 bits muy versátiles, y puertos de entrada/salida de propósito general.

El aJ-80 también incluye una interfaz de texto estandar conocido como la interfaz JTAG. Esta es usada para cargar y depurar programas en el JStamp. Es rápido y potente proporcionando la habilidad de depurar el programa mientras se ejecuta en un circuito. El aJ-80 hace del JStamp su propio dueño en un circuito emulador.

El JStamp tiene 512 Kbytes de memoria **RAM** estática. Esta RAM no tiene batería propia y su contenido se pierde cada vez que se retira la alimentación. Debido a esto, los programas no tienen que ser cargados en la RAM excepto durante los procesos de desarrollo.

El JStamp tiene 512 Kbytes de memoria **ROM** flash. Otra versión del JStamp, la variante llamada JStamp-Plus, contiene 2MB de memoria ROM flash. Esta memoria extra es la única diferencia entre el JStamp y el JStamp-Plus. Normalmente se carga el programa en la memoria ROM flash. También se usa para almacenar ficheros del sistema local.

La fuente de alimentación y el JStamp son un regulador de tensión cam-

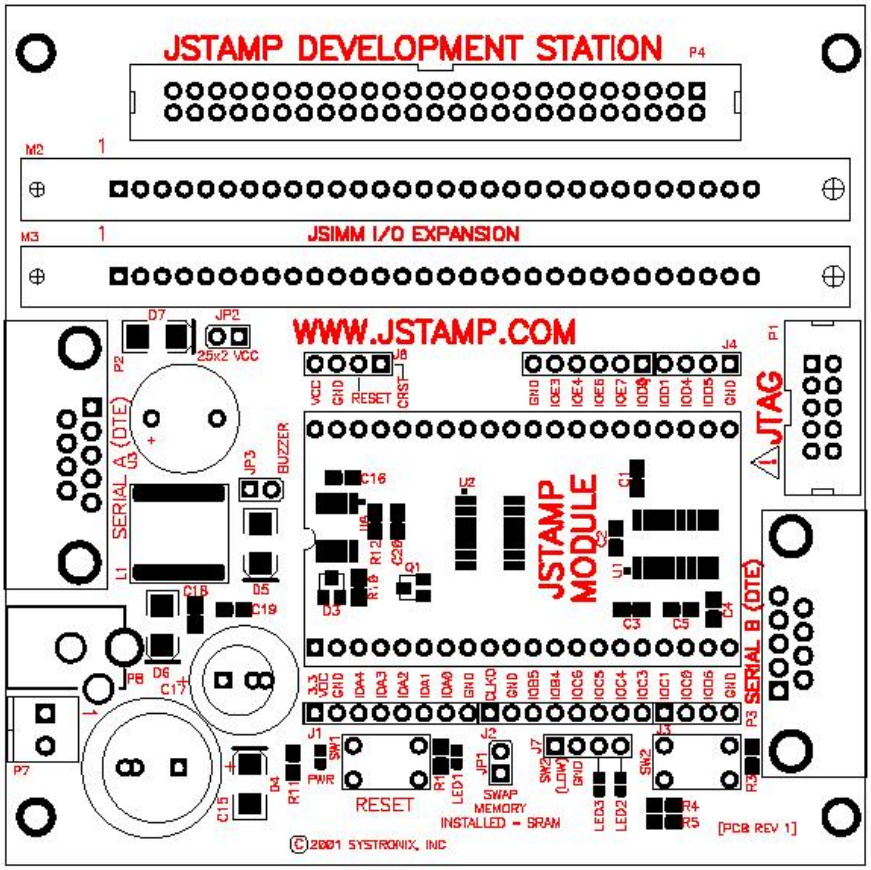


Figura 4.3: Esquema de la estación de desarrollo JStamp.

biante. La alimentación del JStamp puede variar entre 5 y 14 voltios de contínua. El regulador de tensión cambiante genera 3.3 voltios para el circuito de la placa de desarrollo y proporciona 100 miliamperios de 3.3 voltios de tensión continua para ser usados por el circuito de salida.

Casi todos los 40 pines del JStamp pueden ser usados como entrada/salida. Estos pines de entrada/salida son conectados con las patillas de entrada/salida del aJ-80. El módulo JStamp tiene también un LED. En la primera versión del JStamp, este LED era solamente un indicador de alimentación. En las versiones posteriores, se puede controlar este LED via software, aunque por defecto se comporte como un indicador de alimentación.

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 él 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".

JStamp puede ejecutar más de tres millones de bytes de código Java por segundo a 74MHZ. Además puede funcionar durante 24 horas con una batería de 9V.

4.3. Software de desarrollo necesario para el JStamp.

El desarrollo software para el JStamp implica escribir programas en Java. Al contrario que con la mayoría de PC´s y estaciones de desarrollo, el JStamp no tiene teclado ni ningún otro periférico. Esto implica la necesidad de otro computador para escribir los programas en Java y posteriormente cargarlos en él.

Para cargar los programas previamente escritos usando Java en otro computador o estación de trabajo, en el JStamp hay que seguir un proceso que consta de dos pasos. Puesto que el JStamp usa como microprocesador el aJ-80, el proceso a seguir es el mismo que se sigue con cualquier computador que use un procesador aJile. Para ello se necesitan dos herramientas que son proporcionadas por aJile: JEMBuilder y Charade. Estas dos son unas herramientas muy sofisticadas que están en contínuo proceso de integración y mejora.

4.3.1. JEMBuilder

Las librerías de los procesadores aJile no soportan la carga y linkado dinámicos. Los programas cargados en los procesadores aJile, y por tanto en el JStamp, deben por lo tanto, tener todas sus clases linkadas previamente antes de ser ejecutados. Esto es lo que hace el JEMBuilder, es su principal tarea. Aunque el JEMBuilder se usa también para configurar varias opciones del hardware dentro del aJ-80, como la de donde se carga el programa y cómo se va a reiniciar el sistema. El JStamp se puede configurar como se desee mediante esta herramienta. Lo que se obtiene del JEMBuilder es un

fichero binario apropiado para ser cargado en el JStamp, varios otros ficheros útiles para la depuración y el guión que debe seguir el Charade para cargar el fichero en el JStamp.

4.3.2. Charade

La herramienta Charade es usada para transferir el fichero binario generado por el JEMBuilder al JStamp. También es usada para una depuración simbólica y a muy bajo nivel del código que se está ejecutando en el JStamp. Charade se comunica con el sistema aJile mediante un cable especial conectado entre un puerto paralelo del sistema que se esté usando para el desarrollo del programa y el puerto JTAG del JStamp. A este cable se le llama, cable JTAG.

4.4. El procesador aJ-80.

4.4.1. Introducción.

El aJ-80 es un procesador de bajo consumo basado en tecnología Java. Ha sido diseñado especialmente para portar código Java para aplicaciones en el mercado de sistemas empotrados.

Al contrario que los microprocesadores tradicionales que deben consumir potencia de computación para convertir el código de instrucciones Java en el lenguaje nativo del procesador, la línea de procesadores aJile, basada en tecnología Java, opera directamente desde el código Java. Además, las primitivas de threads de Java (como: wait, yield, notify, monitorenter, monitorexit, etc.) son implementadas como extensión del código del aJ-80, eliminando la necesidad de un sistema tradicional que opere en tiempo real (RTOS). Usando el código Java como conjunto nativo de instrucciones, e incorporando las primitivas de threads de Java, el aJ-80 requiere menos memoria y menos tiempo de ejecución en computación que un microprocesador convencional que ejecute una aplicación en código Java.

Esto muestra la ilimitada extensión de la tecnología Java en computación, desde el área de trabajo a aparatos remotos y móbiles, ejecutándose en todos ellos las mismas aplicaciones, lo que puede ser eficientemente actualizado y realizado su mantenimiento desde una única fuente.

El aJ-80 hace posible una de las primeras metas del tiempo real 100 %, los procesadores basados en Java, es decir, sin necesidad de interprete. Esta solución aumenta las oportunidades para los desarrolladores de software para disfrutar de las ventajas del entorno Java y hace posible que las aplicaciones de tiempo real basadas en tecnología Java se ejecuten en gran cantidad de productos de bajo coste y bajo consumo como: teléfonos móbiles, PDA´s, juegos...

El aj-80, al igual que su hermano el aJ-100, presentan como atractivo especial una máquina de procesamiento Java de 32 bits que implementa una "Java Virtual Machine"de tiempo real. El procesador también integra 48 Kbytes de memoria RAM on-chip, y un controlador de memoria, dual UARTs, tres timer/counters de 16 bits y veintidos entradas/salidas de propósito general.

Los procesadores aJile basados en tecnología Java soportan la versión actual de "'Real -Time Specification for Java (JSR-001)"desarrollada por la "Java Community Process".

El aJ-80 ha sido incorporado dentro de un módulo de tecnología Java en tiempo real, el **JStamp**. Ha sido desarrollado por Systronix e integra un procesador aJ-80 con 512 Kbytes de memoria SRAM externa y 512 Kbytes de memoria Flash externa en una placa mínima de 1×2 pulgadas.

En resumen, el aJ-80 es un procesador revolucionario, proporciona a los diseñadores de sistemas un procesador de 32 bits a un bajo coste, con un bajo consumo y ejecuta Java como conjunto de instrucciones nativo. Esto, junto con la SRAM y memoria Flash , el suministro de alimentación y otro circuitería adiccional confinado en un pequeño espacio (JStamp) y fácil de usar, aumentan el rango de aplicación de Java para diseños empotrados.

4.4.2. La arquitectura Java para sistemas Java empotrados de tiempo real eficientes.

aJile Systems fue formado para satisfacer la necesidad del desarrollo de aplicaciones orientadas a objetos para sistemas empotrados de tiempo real de bajo consumo. La arquitectura Java de aJile fue diseñada con un enfoque hacia sistemas empotrados y de este modo maximizar la cantidad de datos de ejecucción que puedan estar en la memoria ROM y eliminar la modificación de la corriente de instrucciones de ejecución (quickizing).

Figura 4.4: aJ-80.

La CPU de aJile soporta directamente el modelo de hilos de Java en su hardware, producciendo el cambio extremadamente rápido entre hilos. La arquitectura aJile también define un conjunto de instrucciones extendido por la intrefaz hardware física y otras tareas de programación de sistemas. Esta extensión de instrucciones no está disponible para el código bajado dinámicamente y que es poco fiable.

4.4.3. The JEM2 direct execution Java microprocesor core.

El aJile System JEM2 es un microprocesador de bajo consumo(1MW/MHz) de segunda generación que se ejecuta directamente en Java. Su diseño compacto y su bajo consumo le hacen muy conveniente como microcontrolador en áreas de aplicación tales como las telecomunicaciones, automatismos y vehículos industriales. JEM2 soporta buses de datos externos de 8, 16 y 32 bits y proporciona una inerfaz de test estandar IEEE 1149.1 (JTAG). Con el JEM2, los desarrolladores de sistemas empotrados y de tiempo real pueden

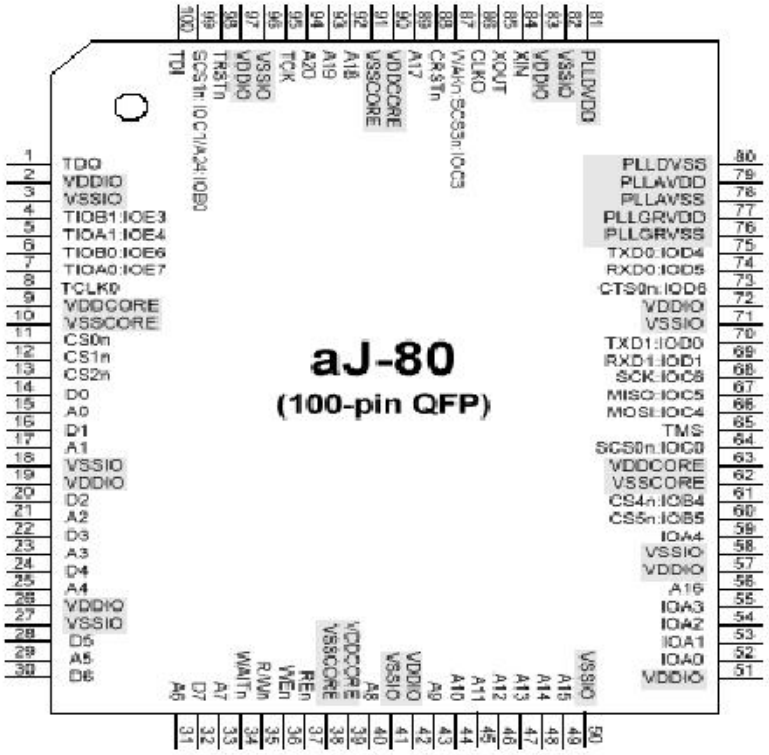


Figura 4.5: Asignación de pines en el aJ-80.

usar el lenguaje orientado a objetos Java, con sus probadas ventajas en productividad, para crear aplicaciones que sean tan eficientes en espacio y tiempo como si estuvieran escritas en lenguaje C para otras plataformas microcontroladoras. Además, el JEM2 es 20 veces más eficiente desde el punto de vista del consumo por unidad de ejecución Java que otras implementaciones empotradas Java.

La arquitectura del J2ME es la mostrada en la figura 4.6:

El JEM2 core implementa todo el conjunto de instrucciones de la JVM en el silicio. Los únicos dos códigos que atrapan inmediatamente al software son *multianewarray* y *athrow*. Obviamente, operaciones como carga de clases son manejadas en software, pero se ha obtenido una solución, la ejecución de código como *invokevirtual* es hecha como una única instrucción JEM, incluyendo la detección de bloqueos.

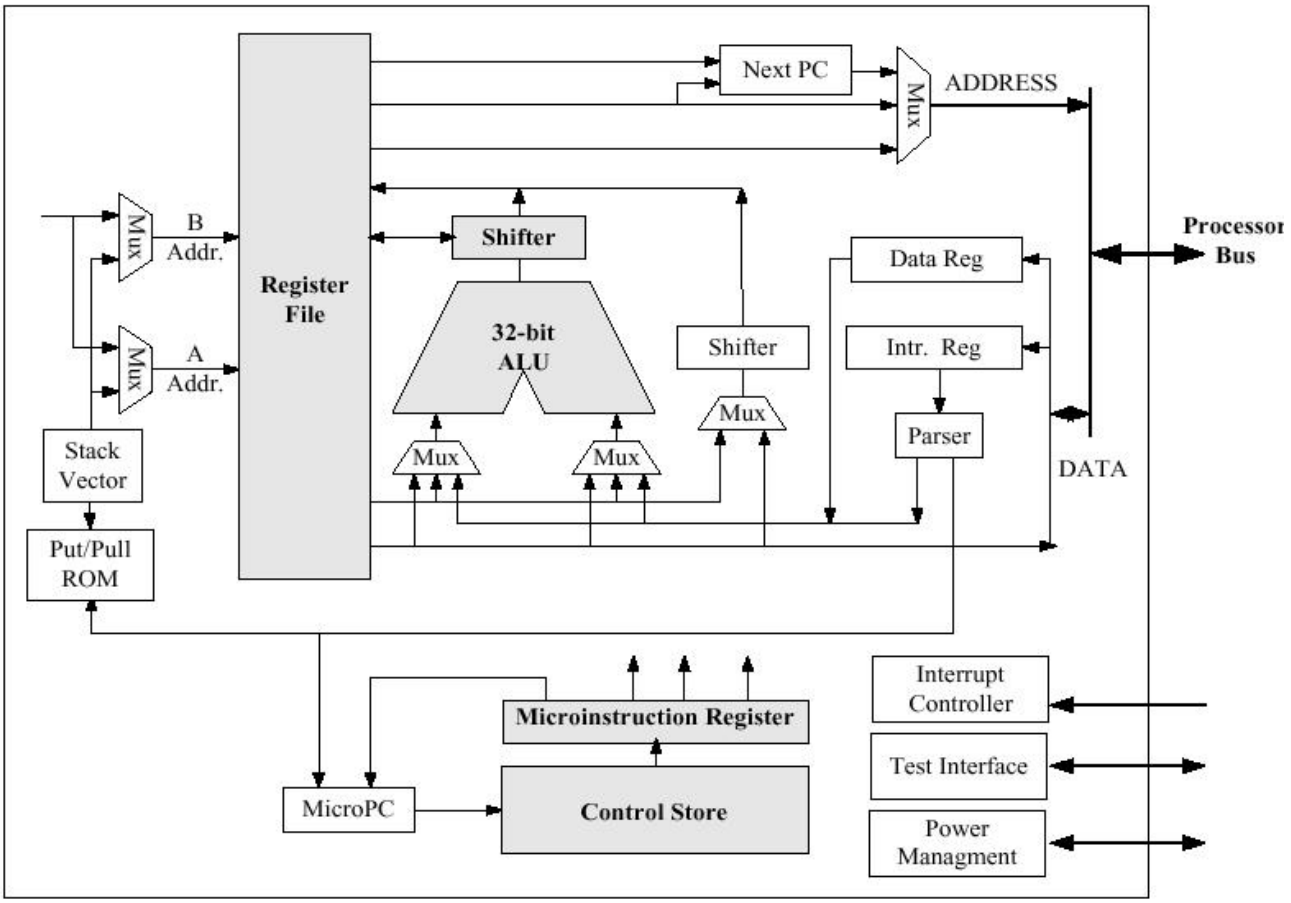


Figura 4.6: La arquitectura del J2ME core.

4.4.4. Soporte hardware para hilos de Java en tiempo real.

Una de las características especiales de la arquitectura aJile es el soporte hardware para hilos Java en tiempo real. El uso de control concurrente está profundamente arraigado en la especificación de la máquina virtual Java (JVM). Las operaciones elementales como la llamada a los métodos de las instrucciones requiere la adquisición de un bloqueo si el método del objeto es declarado como *synchronized*. De este modo, la CPU del ajile implementa las rutinas de sincronización y programación básicas de los hilos en el microcódigo.Esto significa, por ejemplo, que la primitiva *yield()* de la *java.lang.Thread* es una sola instrucción. Con esto obtenemos varios beneficios, el primero,

que la CPU del aJile no requiere el kernel de un sistema operativo de tiempo real (RTOS), lo que implica un ahorro de memoria. Además el uso de hilos múltiples es extremadamente rápido en la CPU del aJile. Por ejemplo, el tiempo requerido para ejecutar un *yield()* y reanudar un hilo diferente es aproximadamente de 500 nanosegundos en una CPU de aJile de 100MHz. Adicionalmente, el hardware de aJile soporta la consulta periódica de hilos y también implemente la inversión de prioridades para el control de estos.

4.4.5. Un entorno de ejecución Java escrito enteramente en Java.

Una característica especial única del entorno de ejecucción de aJile es que este procesador es programado enteramente en lenguaje Java, incluye los sustitutos para la mayoría de los métodos nativos del sistema de ejecución Java. Esto se puede hacer gracias a la herramienta de generación de aplicaciones JEMBuilder, que proporciona la sustitución de ciertas llamadas a métodos con código aJile.

Con este adelanto, no es necesario el uso de un ensamblador, y los desarrolladores pueden crear implementaciones Java con las clases del aJile en un entorno de simulación. El resultado es que todo el código ejecutable está escrito en Java, de este modo es más sencillo el mantenimiento y comprobación.

4.4.6. Manejo de memoria.

La especificación de la máquina virtual de Java incluye varias instrucciones para el manejo de memoria. Sin embargo, el entorno de desarrollo Java no tiene la primitiva *free*, además el reclamo de memoria es automático. Por tanto, el entorno aJile, implementa instrucciones de localización de memoria e implementa un recolector de basura muy simple en su software.

El recolector de basura es implementado como un hilo y usa la sincronización de Java para asegurarse que lo que recoge para eliminar no produce errores. El hilo del recolector de basura es muy rápido, dura menos de un microsegundo para el aj-100 a 100MHz.

4.4.7. Manejo de interrupciones y bloqueo.

La naturaleza virtual de la especificación de la JVM implica que los mecanismos de bajo nivel como son las interrupciones y el bloqueo de la ejecución están especificados. Por eso, los arquitectos del aJile han proporcionado una implementación propia. En relacción con el tema de escribir todo el código software en Java, las interrupciones y bloqueos son manejados por métodos estáticos, que se ejecutan en una pila ejecutora o supervisora. El modo ejecutor del procesador puede ser pensado como un solo hilo de máxima prioridad que tiene su propia pila y zona de memoria, y que estos métodos son los controladores de los distintos bloqueos e interrupciones.

Como se puede esperar, los controladores de máscara de las interrupciones pueden ser apropiados de antemano por la ocurrencia de otra interrupción de prioridad mayor, si las interrupciónes se habilitan en el código puesto que por defecto están deshabilitadas. Cuando un controlador de prioridad más alta es invocado, una nueva pila se ejecuta. Cuando haya acabado, el controlador devuelve el control al programa.

4.4.8. Personalización del conjunto de instrucciones.

La CPU del aJile permite generalmente escribir el control de almacenamiento de datos, permitiendo así crear un conjunto de instrucciones personalizado para cada aplicación particular. Las nuevas instrucciones pueden aumentar la rapidez de actuación de los algoritmos usados más frecuentemente en esa aplicación. El poder de las instrucciones personalizadas se refleja en el cambio de instrucciones de hilos del J2ME. Por ejemplo: el resultado de una instrucción *yield()* en el paso de un hilo a otro es de un microsegundo mientras que en un RTOS tradicional escrito en lenguaje de alto nivel puede tomar varios milisegundos. Como con otros códigos extendidos, el hecho de que el usuario defina las instrucciones a seguir se puede hacer gracias al JEMBuilder llamando a la sustitución de métodos estáticos. No se requieren cambios en el compilador para esto.

4.4.9. Varias máquinas virtuales de Java concurrentes.

La arquitectura del aJile proporciona un soporte hardware para la ejecución simultánea de varias máquinas virtuales independientes en una apli-

cación programando el reparto de tiempo de la memoria determinista con protección total de la memoria. Dentro de estos intervalos de ejecución seguros y espacio de memoria, cada entorno de aplicación puede desarrollar sus propias reglas de atención a hilos y manejo de memoria sin que esto perjudique al resto de las aplicaciones. Adicionalmente, cada máquina virtual tiene su propio modo de ejecutarse, esto permite a la JVM asignar sus propias interrupciones que no se atenderan mientras que la JVM esté suspendida.

4.4.10. Multiple JVM Manager (MJM).

Las características especiales de las multiples JVM del aJ-80 permiten llevar a cabo dos aplicaciones Java independientes que se ejecutan de una programación y reparto de tiempo totalmente determinado y con protección de la memoria. Dentro de su espacio de memoria limitado, cada entorno JVM puede emplear sus propia política de manejo de hilos múltiples y utilización de memoria sin la intervención erronea de otras aplicaciones o hilos.

El controlador de multiples JVM (MJM) proporciona recursos de tiempo e interrupciones lógicas para asegurar que otras JVM que se estén ejecutando en ese momento no interfieran en las aplicaciones que esté realizando una JVM determinada. El controlador de JVM proporciona un contador para controlar la porción de tiempo que le corresponde a cada JVM. También se tiene un contador individual para cada JVM (en total hay cuatro) para controlar los hilos dentro de esa JVM. El MJM es el representado en la figura 4.7:

4.4.11. aJ-80: Un microcontrolador de Java para sistemas empotrados de tiempo real.

El aJ-80 de aJile Systems es un microcontrolador de Java para sistemas empotrados de tiempo real basado en el JEM2 core que integra el código de ejecución de la máquina virtual de Java, las primitivas de hilos para tiempo real de Java, y el soporte necesario para el uso de varias JVM, junto con periféricos comunes de los sistemas empotrados.

La arquitectura del chip aJ-80 es la mostrada en la figura 4.8:

Esto permite a los desarrolladores disfrutar de las ventajas y beneficios del entorno Java en un paquete muy compacto y de bajo consumo con el

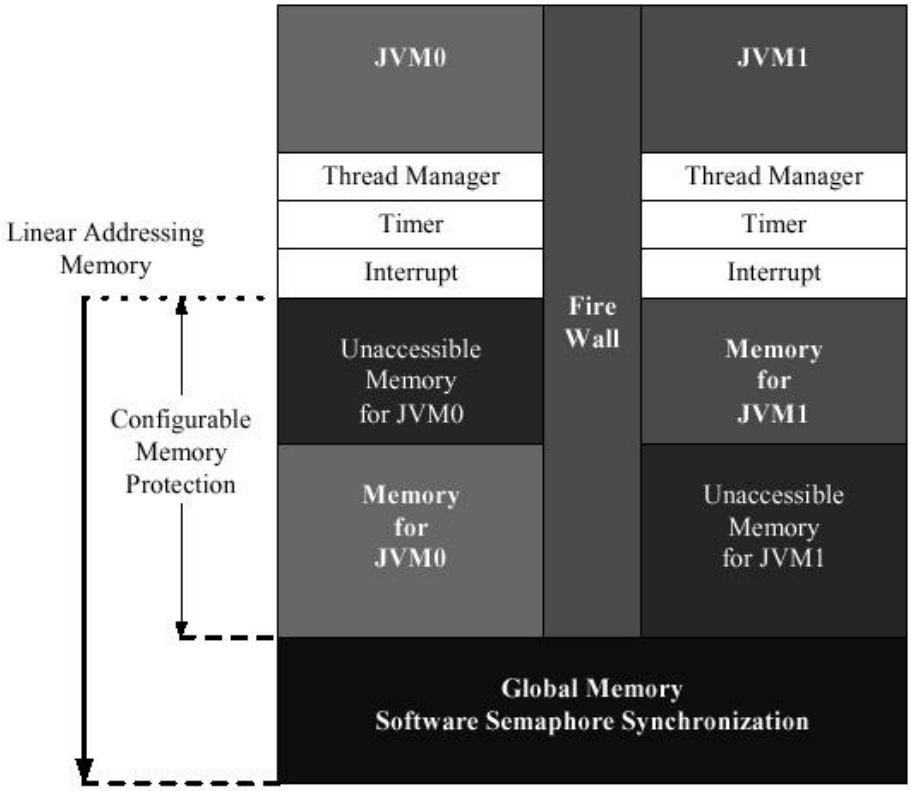


Figura 4.7: Multiple JVM Manager (MJM).

cumplimiento del procesamiento necesario para los sistemas empotrados. Con un "Java core" de bajo consumo de 32 bits, memoria on-chip y un conjunto de periféricos, esta plataforma microcontroladora de Java en tiempo real, es idónea para pequeños aparatos móbiles, aparatos de consumo, aplicaciones automotrices y controladores industriales de red.

Proporcionando un "System on-chip", el aJ-80 permite a los desarrolladores proporcionar fácilmente la funcionalidad de los sistemas empotrados Java a sus productos. Esto viene acompañado del paquete de ejecución J2ME CLDC, incluyendo los drivers necesarios para todos los periféricos del aJ-80 y los drivers externos más comunes como por ejemplo: Ethernet, memoria FLASH, y controladores de los displays LCD.

La arquitectura del system-on chip(SOC) del aJ-80 es la mostrada en la figura 4.9. El aJ-80 usa una arquitectura de bus dual: el bus del procesador y el bus de los perífericos. Para minimizar la carga del bus y el consumo

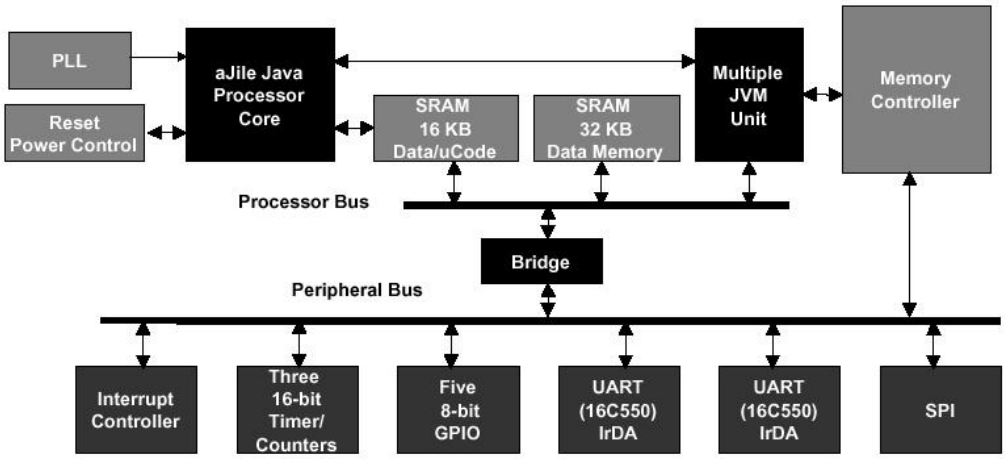


Figura 4.8: Bloque de control del aJ-80.

asociado, el bus del procesador es limitado a aquellos periféricos que requieren un ancho de banda elevado (CPU, memoria, interfaz externa). El bus de los periféricos proporciona acceso a los périféricos on-chip y es aislado del bus del procesador gracias al puente de periféricos. La interfaz externa del bus genera la dirección, datos, señales de control para conectar directamente con la mayoría de periféricos.

Memoria interna.

El aJ-80 proporciona 48 Kbytes de memoria interna que no tiene la necesidad de esperar, se carga directamente. Los 32 Kbytes de RAM son normalmente usados para almacenar la pila de procesamiento del JEM2. Los 16 Kbytes restantes son usados para implementar el kernel de Real-Time, extensiones de código e instrucciones personalizadas.

La interfaz externa de Bus (EBI).

La interfaz externa de bus (EBI) genera las señales de control de acceso a la memoria externa y los periféricos. La EBI puede acceder directamente a 256 Mbytes de memoria externa. Además se puede extender este espacio de memoria mediante líneas de direcciones. La EBI proporciona 8 selectores de

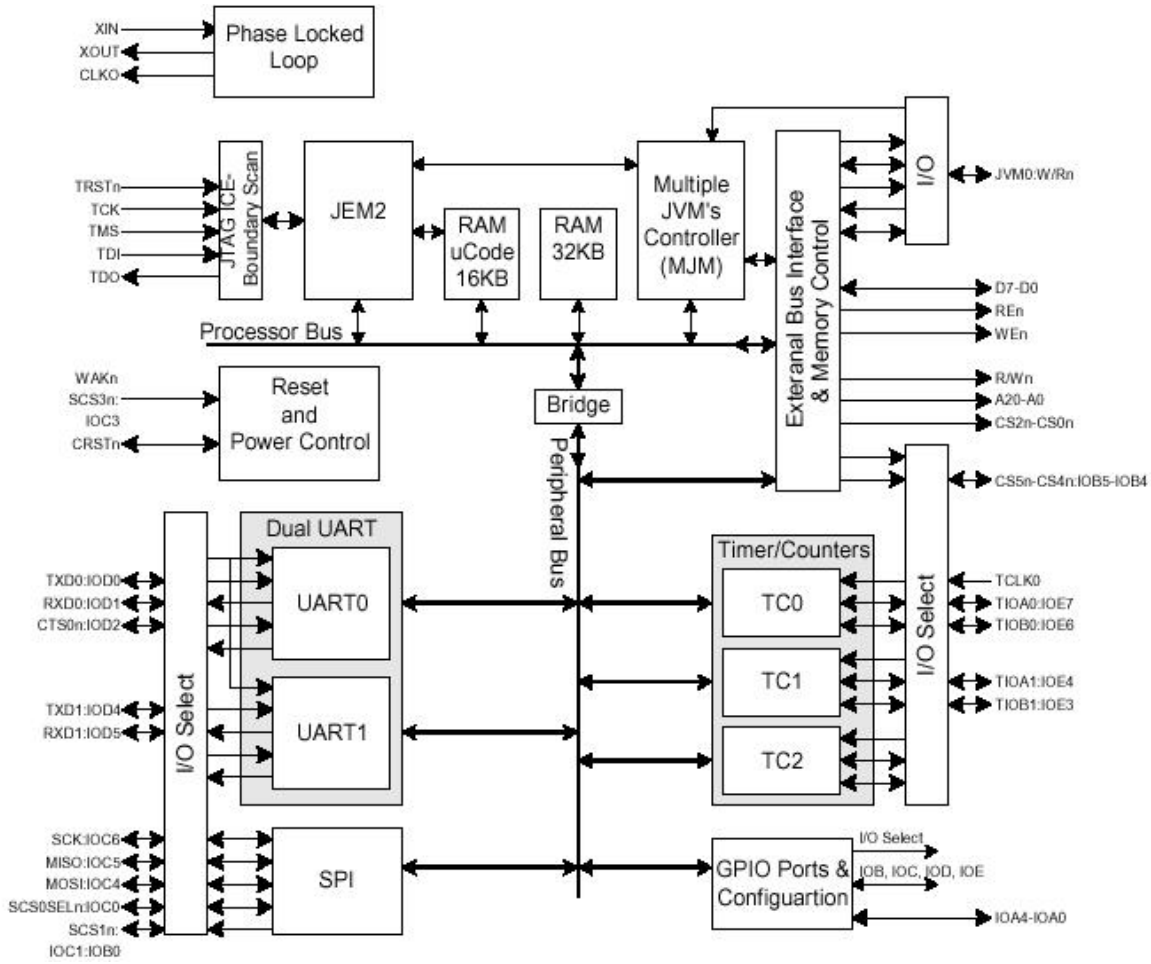


Figura 4.9: Arquitectura del aJ-80.

chip. La EBI puede ser configurada para soportar periféricos de 8, 16 y hasta 32 bits de memoria. Las señales de control de memoria son proporcionadas para habilitar conexiones directas con la memoria externa y la memoria mapeada de los periféricos de entrada/salida. Las transacciones son controladas con el generador de estados de espera con una señal externa de espera proporcionada para facilitar el acceso a los periféricos lentos. La EBI se muestra en la figura 4.10:

- **Bus Interface Timeing.**

  El aJ-80 proporciona una interfaz flexible y segura para interactuar con la memoria externa y los periféricos. El aJ-80 proporciona las opera-

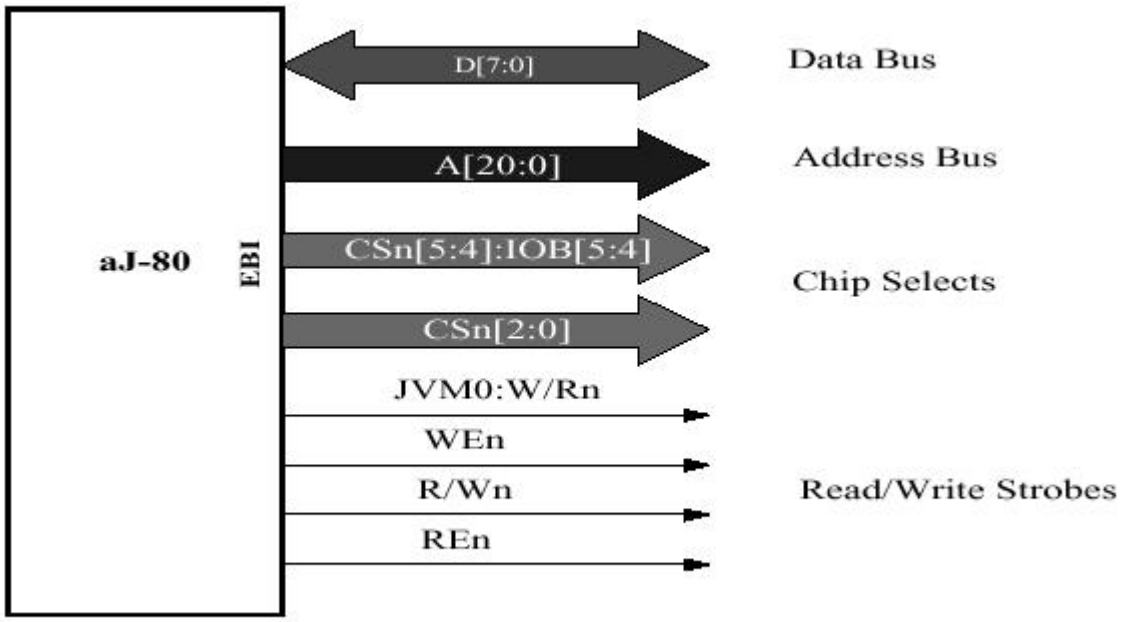


Figura 4.10: External Bus interface.

ciones de tiempo necesarias para acceder a los periféricos externos. Cada salida del selector de chip tiene asociada un registro de configuración para especificar el tiempo de establecimiento, el tiempo de trabajo, los estados de espera y el ancho de memoria. Los registros de configuración son cargados como parte del proceso de inicialización del reset. Una señal de espera externa se proporciona para extender la transferencia externa a periféricos lentos o buses globales. La transferencia de datos fundamental es la mostrada en la figura 4.11 :

- **Tiempos de transferencia extendidos.**

  La interfaz de bus del aJ-80 permite que las transacciones puedan ser prolongadas gracias a la señal externa de espera de transaciones (WAITn). Los ciclos prolongados son útiles para acceder a recursos que tienen tiempos de respuesta variables o periféricos lentos. En la figura 4.12 se muestra la WAITn:

- **Acceso a periféricos con interfaces ISA.**

Muchos periféricos tienen interfaces que soportan mediante un bus ISA. Para soportar periféricos con este tipo de interfaz el aJ-80 habilita el CS4n y CS5n para operar como señales MEMRN y MEMWN (o IORN y IOWN). Un acceso a un periférico mrdiante un Bus ISA se ilustra en la figura 4.13:

4.4.12. Timer/Counter (TC).

El aJ-80 incluye tres timer/counters 16 bits (TC) que pueden realizar un rango amplio de funciones. Las funciones de los timer/counters incluyen medida de frecuencia, contador de eventos, medida de intervalo, contabilizar retrasos, y modulacion de anchura de pulso. Las caracteristicas principales de los timer/counters de proposito general son:

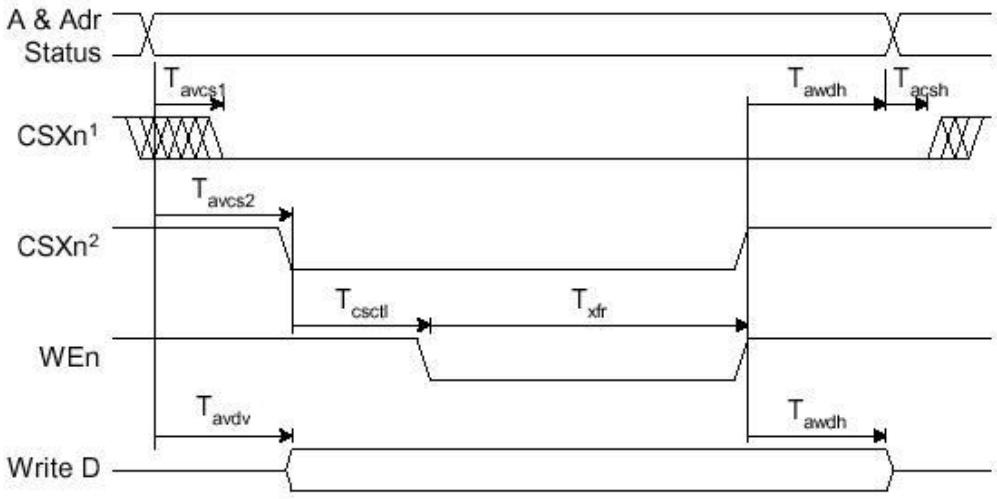
- Tres contadores de 16 bits.
- Encadenamiento de los tres contadores para conseguir un rango más amplio de resoluciones.
- Reloj externo, control de disparo y de puerta.
- Dos moduladores de ancho de pulso y dos módulos wave-form.
- Un generador flexible de interrupciones.
- Un preescalador de 16 bits.

El timer/counter Programable (TC) comprende un prescaler de 16-bits y tres timer/counters muy versátiles de 16 bits como se puede ver en la figura 4.15. Se pueden usar como fuentes para el TC dos fuentes de reloj , la entrada del reloj interno y el reloj externo (TCLK0). El prescaler divide el reloj de entrada seleccionado por PRL+1, donde $0 \leq$ PRL≤ 65535, y proporciona la habilitación usada por los timers. Los timers pueden ser empleados como un intervalo o como un contador cíclico.

4.5. Acerca de Systronix Inc.

Systronics es una empresa que lleva algo más de 15 años creando sistemas de control de tiempo real empotrados de pequeño tamaño para mercados tan

distintos como reproductores DNA o equipos de test para vuelos militares. Durante los últimos años ha estado metida de lleno en la industria emergente de los sistemas empotrados que usan tecnología Java. Está desarrollando un numeroso grupo de productos sobre los controladores de tiempo real aJ-80 y aJ-100.



Transferencia de escritura en el aJ-80.

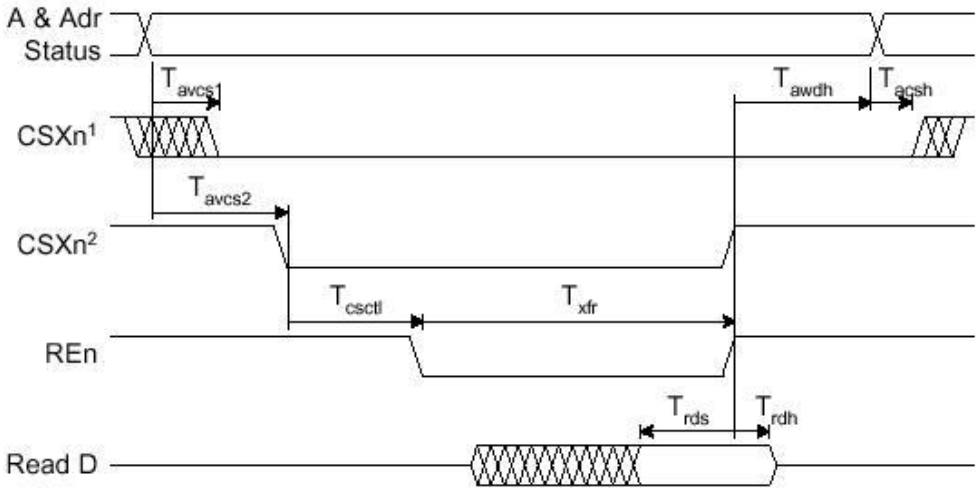


Figura 4.11: Transferencia de datos (lectura y escritura)

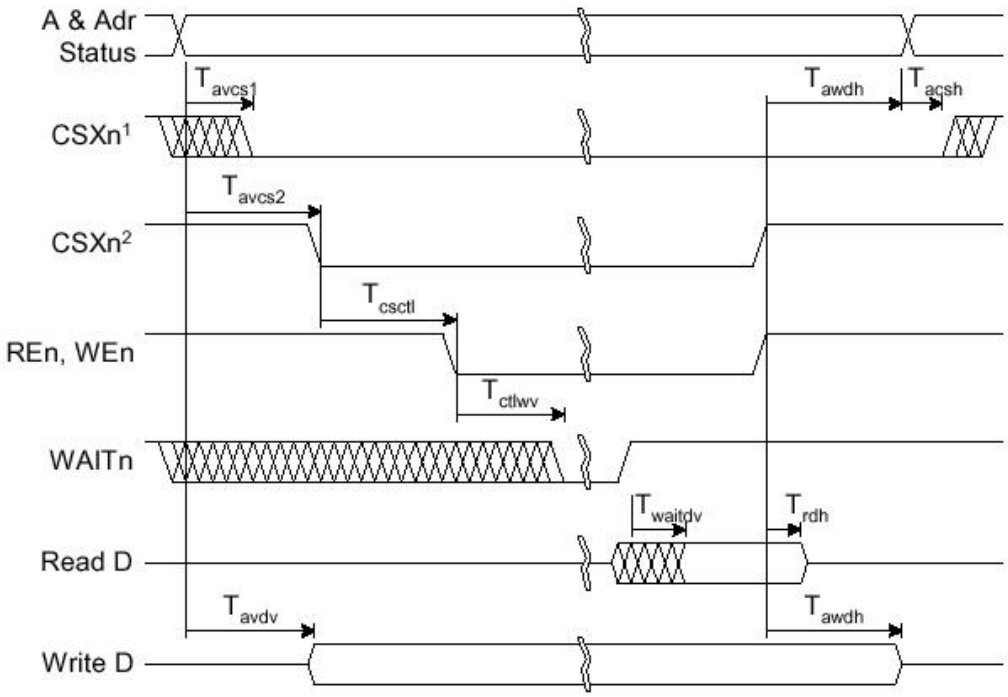


Figura 4.12: Extended Bus Transaction (WAITn).

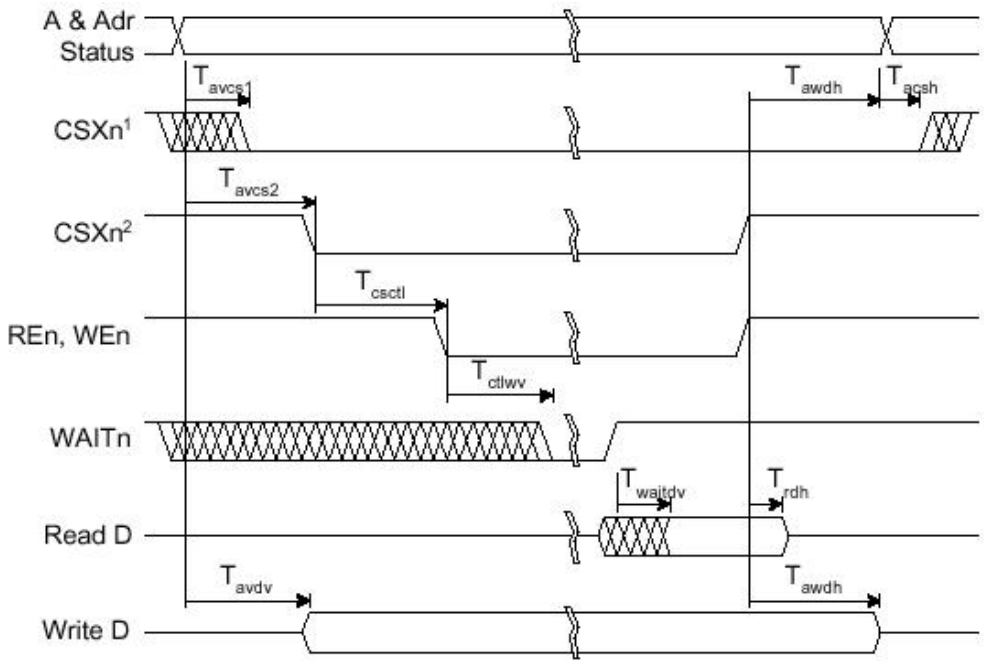


Figura 4.13: ISA-Oriented Peripheral Accesses.

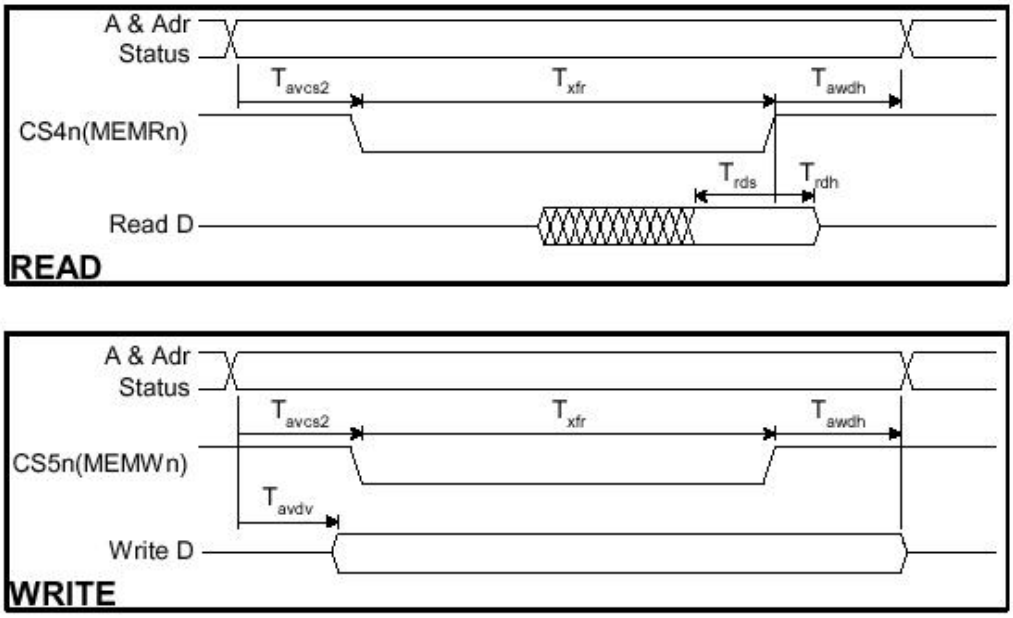


Figura 4.14: ISA-Oriented Peripheral Accesses2.

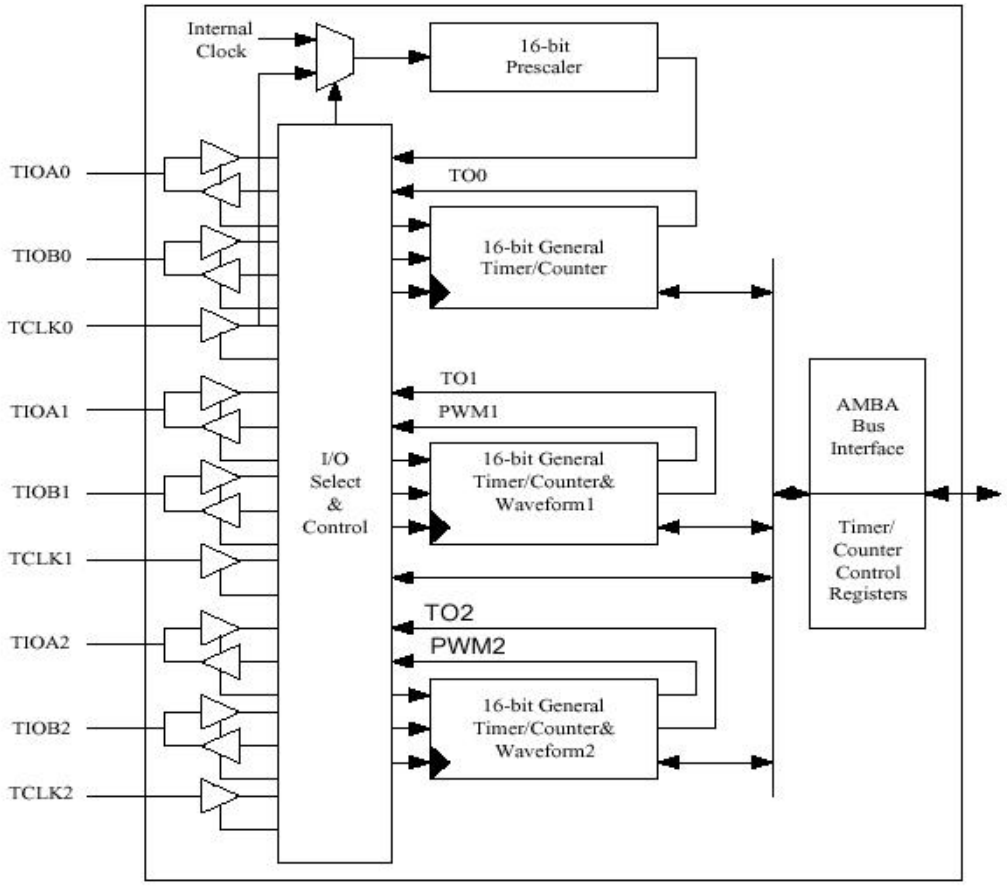


Figura 4.15: Diagrama de bloques del Timer / Counter.

Parte IV

Diseño.

Capítulo 5

Diseño.

5.1. Requisitos.

- Implantación de los sensores de ultrasonidos de BOSCH facilitados en un sistema autónomo de tiempo real.
- Usar como procesador para gestión de la información proporcionada por los ultrasonidos el aJ-80 de Systronix, y la estación de desarrollo JStamp.

5.2. Descripción de la tecnología usada.

El ultrasonidos usado es de control manual, es decir, se debe generar externamente el pulso de transmisión y medir el tiempo que tarda en recibirse la señal de eco. El pulso de disparo se genera mediante un programa escrito en Java y ejecutado por el JStamp, dicho pulso debe tener una duracción mínima de 300μs como puede verse en la figura 5.1. Una vez introduccido

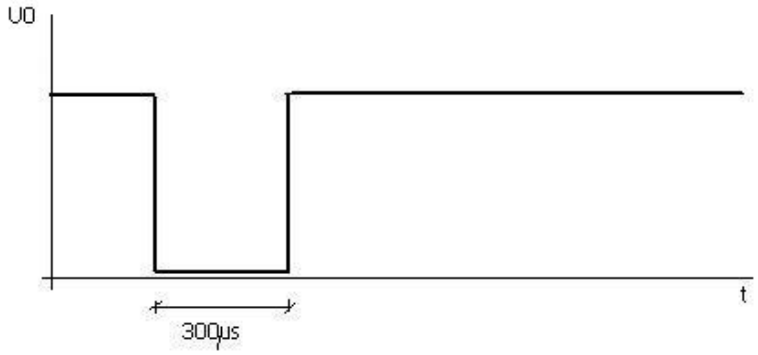


Figura 5.1: Pulso de disparo

el pulso de disparo en el sensor y tras unos microsegundos, se produce la ráfaga ultrasónica, es decir, las señales ultrasónicas que emite el sensor para detectar si hay o no obstáculos.

Tras unos microsegundos que tarda el sensor en estabilizar su membrana,

empieza a funcionar como receptor esperando el eco produccido al chocar la ráfaga ultrasónica en un objeto.

Se disparará el ultrasonido de nuevo después de un tiempo de rearme necesario para que se estabilice de nuevo la membrana. Debido a que este sensor tiene un único módulo que funciona como emisor y receptor, el tiempo de rearme es mayor.

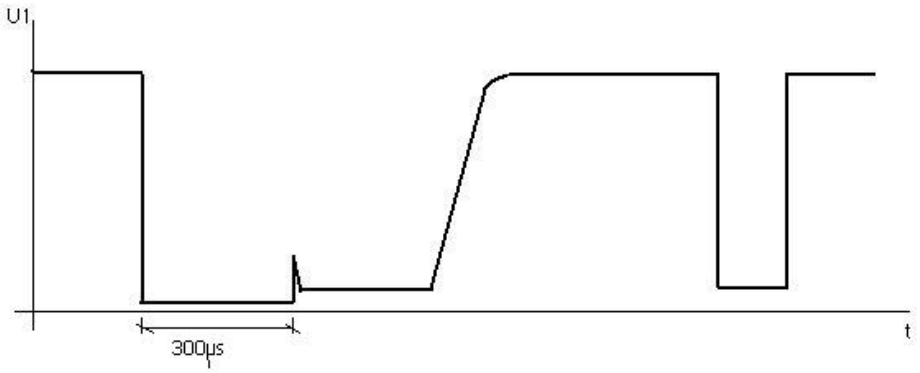


Figura 5.2: Salida del sensor.

5.3. Conexionado.

El conector hembra de USS 3.X es el mostrado en la figura 5.3

El pin 1 es el pin de entrada y salida del ultrasonido, por lo que para conectarlo al JStamp, habra que hacer una construcción auxiliar con un diodo en la entrada. El diodo empleado es el 1N4148.

El pin 2 es el pin que se conecta a tierra.

El pin 3 es la salida analógica.

El pin 4 es el pin que se conecta a la alimentación, a una fuente de tensión de 8V de contínua.

De este modo la conexión queda como se puede ver en la figura 5.4.

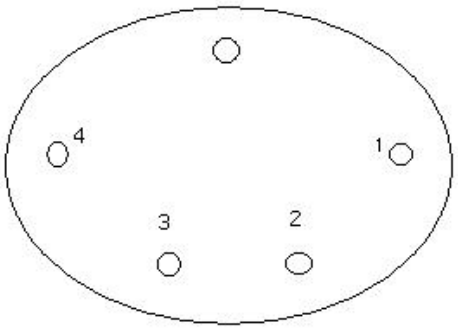


Figura 5.3: Conector Hembra de USS 3.X

5.3.1. Conexiones en el JStamp.

- El puerto **IOE3** se usa como salida del pulso de disparo que se genera mediante un programa Java. Por tanto se unirá con en pin 1 del conector hembra del ultrasonidos 3.X mediante un cable y usando un diodo 1N4148 como se mostró en la figura 5.4. Además es necesario un pequeño circuito auxiliar puesto que el JStamp proporciona unas salidas de 3,3V y para el disparo del sensor son necesarios 8V.
- El puerto **IOE4** se usa como entrada de la señal de eco que recibe el ultrasonidos. Por tanto se unirá con el pin 1 del conector hembra mediante un cable.

5.3.2. Circuito auxiliar.

Para poder conectar la salida del JStamp que produce el pulso de disparo (El puerto **IOE3** con la entrada del sensor de ultrasonidos, es necesario adaptar la tensión puesto que el JStamp produce salidas de 3,3V y el ultrasonidos necesita 8V. Para ello, se dispone de un regulador de tensión MC7808C, que proporciona una salida de 8V y 1A.

Dicho regulador necesita dos condensadores para su conexión en el circuito como se muestra en la figura 5.6.

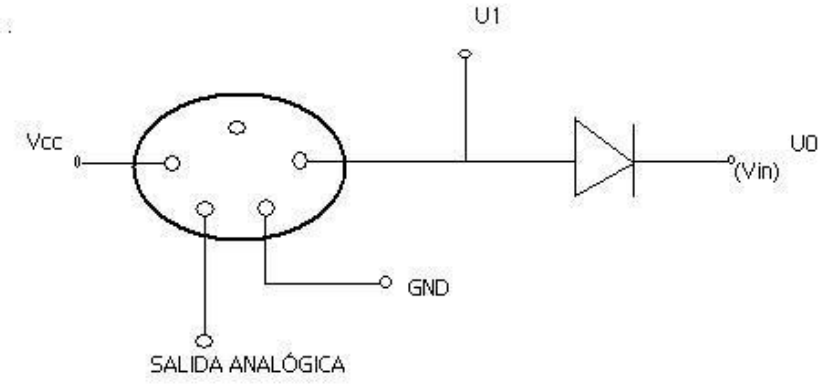


Figura 5.4: Conexiones en el ultrasonidos.

5.3.3. Alimentación.

La alimentación usada para el JStamp y por tanto para los sensores será distinta de la usada para el servo puesto que éste introduce ruidos en la alimentación y requiere grandes picos de corriente intermitentes que podrían afectar a los componentes electrónicos.

Puesto que estamos hablando de un sistema autónomo, la elección se limita a baterías recargables como fuente de potencia. Por tanto se van a usar baterías recargables de Ni-Cd de alta capacidad. Son unas pilas cilíndricas con terminales de soldadura en la parte superior e inferior para su montaje en lotes.

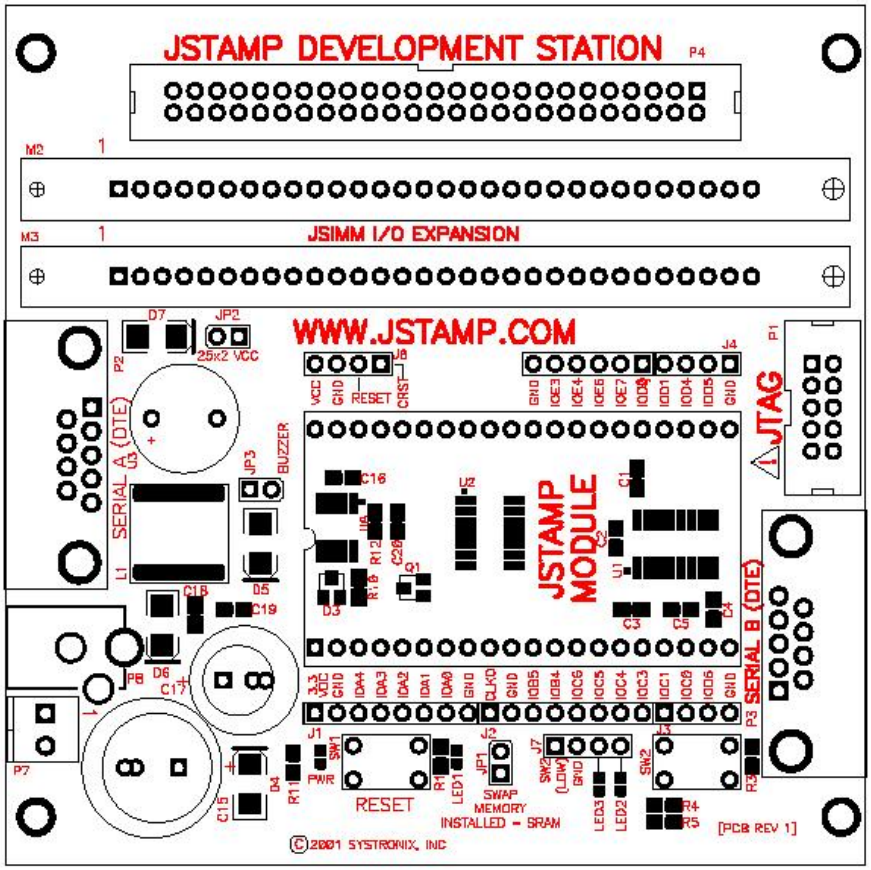


Figura 5.5: Pines del JStamp.

Estas pilas proporcionan altas capacidades para tiempos de descarga ampliados y tienen la capacidad de realizar más de 700 ciclos completos de carga/descarga. Además soportan altas corrientes de descarga continuas y presentan baja resistencia interna.

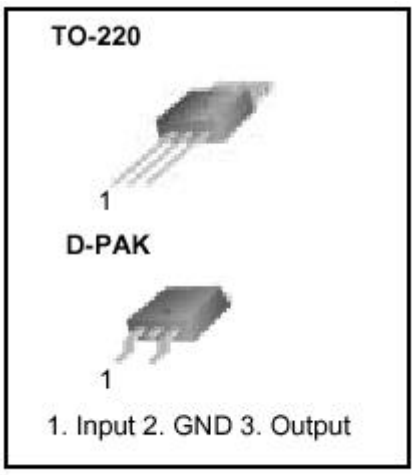


Figura 5.6: Regulador de tensión MC7808C.

Se ha realizado un paquete compuesto de 12 pilas en serie lo que proporciona 15V de tensión. Para transformar la potencia de las baterías en tensiones utilizables, se usa un convertidor de tensión CC/CC de 30W, con salida triple de +12V /-12V /+5V. Estas tres salidas fueron llevadas a una placa de conexión con conectores para: ESC, Servo, Wafer, JStamp.

5.4. Programación.

Tanto para generar el pulso de disparo como para medir el eco de respuesta se usa un contador, el timer1, que se configura con una entrada (line_A) y una salida (line_B) para que corra libremente, se recargue automáticamente. La salida es la encargada de disparar el ultrasonidos y la entrada espera la señal de eco del ultrasonidos.

En este programa el timer1 realiza una acción cada 33ms (aproximadamente), esto permite al JStamp guardar el resultado sin prisas ni problemas.

5.4.1. Generación del pulso de disparo.

Hay que establecer la granularidad del contador, como se precisa un pulso de disparo de 300μs, tendremos que determinar el valor de la granularidad

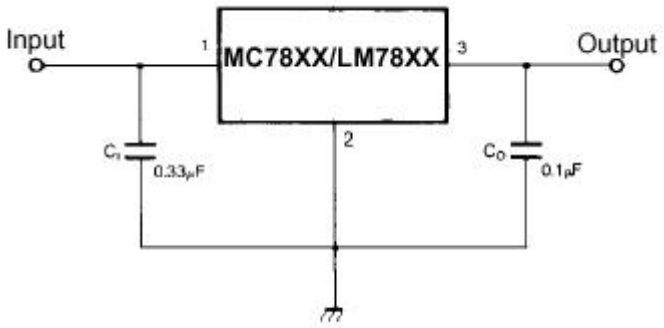


Figura 5.7: Circuito del regulador de tensión.

en función de este valor. El *prescaler* del aJ-80 puede tomar cualquier valor comprendido entre 2 y 65535, este valor divide la frecuencia del reloj tomado como fuente dandonos así la granularidad. Las fuentes posibles para el reloj del contador son dos:

- Una fuente externa.
- El reloj interno del procesador.

He escogido el reloj interno, cuya frecuencia para el JStamp es de 73,728 MHz. Como el *Internal peripheral Clock* (este es el reloj escogido como fuente del prescaler) tiene una frecuencia igual a la mitad de la del reloj fuente, se tiene una frecuencia de 36,864 MHz, que proporciona un periodo de 27,127 ns, es decir, proporciona una granularidad de 27,127 ns. Como se quiere una granularidad de 300 μs se deberá introducir en el prescaler un valor de 11059,2, por lo tanto, introduciré un valor de 11060.

Establecida la granularidad se crea un pulso de diparo como el mostrado en la figura 5.1.

5.4.2. Lectura del eco de respuesta.

Una vez se ha enviado el pulso de disparo y tras unos microsegundos, el sensor funciona como receptor esperando una señal de eco. Para que esto sea

posible se habilitan las interrupciones para que el sensor esté a la escucha y en el momento que llegue una señal se interrumpa el timer.

En el momento que llega la señal de eco ocurren dos cosas:

- Se almacena el valor que tiene el timer en ese momento en un registro del procesador (en el *Sample Value Register*). Para que se realice esta acción no es necesario incluir ninguna línea de código, se hace de manera automática, por lo que ocurre instantáneamente, la única limitación de tiempo es la granularidad del timer.
- Se dispara un evento programado para que se pueda leer ese valor almacenado en el registro. Como esto ocurre a partir de unas instrucciones de software programadas, no es instantáneo, lleva un tiempo, pero esto no es ningún problema siempre y cuando no se vuelva a disparar el sensor hasta que se haya obtenido este valor. Por eso no se vuelve a disparar el ultrasonidos hasta que no ha pasado un tiempo suficiente (33ms en este caso).

La interrupción que se habilita es la del puerto **IOE4**, de manera que cuando se reciba por él un flanco de bajada se ejecute la rutina de la interrupción programada. Esta rutina recupera el valor almacenado en el *Sample Value Register* y el número de veces que se ha recargado el contador para así poder sacar el tiempo de vuelo con operaciones (de esto se encarga el programa principal).

5.4.3. Calculo de la distancia.

La distancia a la que se encuentra el obstáculo del robot se calcula a partir de los datos obtenidos en la rutina de interrupción (el valor del contador en ese momento y el número de veces que se había recargado) con un sencillo cálculo consistente en restar el valor obtenido del contador al valor con el que se había recargado éste, el resultado de esta operración se suma al resultante de multiplicar el valor de recarga del contador por el número de veces que se ha recargado, obteniendo así el tiempo de vuelo de la señal.

A partir de este valor de tiempo de vuelo, se calcula la distancia a la que se encuentra el objeto con la siguiente fórmula:

$$distancia = (340 * tiempodevuelo) \tag{5.1}$$

siendo 340 la mitad de la velocidad del sonido en el aire en cm/sg .

5.4.4. Comportamiento del robot.

Una vez calculado el valor de la distancia al objeto u obstáculo, se opera en consecuencia. Para ello se ha escrito otro programa Java que determine la velocidad a la que puede ir el robot y la desviación de la trayectoria de éste, es decir, el giro del servo, para esquivar el obstáculo. Todo esto es posible gracias a la capacidad de Java de ejecutar hilos múltiples.

5.5. Descripción de la configuración adoptada.

En el apartado 1.8 del presente proyecto se expusieron las distintas alternativas de configuración posibles para implantar los sensores de ultrasonidos de manera eficiente en el robot de tiempo real objeto del proyecto. Se expusieron mostrando sus ventajas e inconvenientes de cada una de ellas, por lo que no me extenderé más en este apartado.

Teniendo en cuenta dichas ventajas e inconvenientes y teniendo en cuenta los objetivos del proyecto se llegó a la conclusión de adoptar la configuración en anillo de sensores usando el método de medidas consecutivas combinado con el de esperas alternadas (ambos expuestos en dicho apartado 1.8) para solucionar los problemas de ruidos e interferencias con otros sensores.

5.6. Programación en JBuider8.

Para este proyecto he elegido como entorno de programación Java el JBuilder, cuando comencé su desarrollo usé la versión JBuilder 5 y actualmente estoy usando la JBuilder 8, aunque se puede usar cualquier otro paquete de programación comercial.

Los pasos a seguir desde que se empieza a programar en Java hasta que se ejecuta en el JStamp son los siguientes:

1. Crear y compilar el programa con JBuilder
    - Crear proyecto (.jpx o .jpr)

- En "Propiedades de proyecto -¿Vias de acceso -¿Bibliotecas necesaria" añadir aJile CLDC y J2ME CLDC (necesarias puesto que necesitamos la especificación de tiempo real.)
- Compilar

Así se obtiene un fichero **.class** a partir del **.java** .

2. Usar JEMBuilder para tomar las clases compiladas y crear el byte-code para el JStamp.

   - Crear proyecto.
   - Elegir directorio donde se creará éste.
   - Elegir configuración flash de memoria en el siguiente paso (JStampFlashConfiguration) y como runtime :Runtime_CLDL . (recordar poner o quitar el jumper JP1 en la placa de desarrollo)
   - Crear una nueva JVM: por defecto JVM0
   - Nombre de la clase que contiene el "main"de esa JVM: en este caso "Ultrasonidos"
   - Indicar el Classpath a seguir para encontrar la clase anterior: por ejemplo .../blink/class
   - Available drivers: elegir los necesarios, en este caso TIMER y PORT E.

   Con estos pasos se genera el fichero **.bin** que entiende el aJ-80.

3. Usar Charade para bajar el programa al JStamp.

   Con este programa se baja el fichero **.class** generado por el JEMBuilder al procesador para que pueda ser ejecutado.

   Hay que tener en cuenta el elegir como "Device" el aJ-80 port 378 (por defecto escoge el aJ-100) cuando el Jtag esta conectado en el puerto COM1 del PC.

   Eligiendo archivo y dentro de este load, se elige el archivo **.bin** y se carga el el JStamp.

   Para ejecutarlo, se elige archive-¿execute y se elige el archivo **.soad**. Posteriormente se selecciona Run y se ejecuta.

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.

Parte V

Apéndices

Apéndice A

Abreviaturas de Java.

AIE:

Asynchronous Interrupt Exception.

ATC:

Asynchronous Transfer Control.

CLDC:

Connected Limited Device Configuration.

DUART:

Dual Universal Asynchronous Serial Port.

EBI:

External Bus Interface.

GPIO:

General Port Input/Output.

IDE:

Integrated Development Enviroment.

JAM:

Java Applications Manager.

JCP:

Java Community Process.

JIT:

Just-In-Time.

JSR:

Java Specification Request.

JVM:

Java Virtual Machine.

J2ME:

Java 2 Micro Edition.

MIDP:

Mobile Information Device Profile.

MJM:

Multiple Java Virtual Machine Manager.

NIST:

National Institude of Standards and Technology.

PLL:

Phase Locked Loop.

PRL:

Prescaler.

PWM:

Pulse Widw modulator.

RMI:

Remote Method Invocation.

RTJEG:

Real-Time Java Expert Group.

RTJG:

Real-Time for Java Expert Group.

RTJVM:

Real-Time Java Virtual Machines.

RTOS:

Real-Time Operating System.

SDE:

Software Development Enviroment.

SOC:

System-On-Chip.

SPI:

Serial Peripheral Interface.

TC:

Timer/Counter.

UART:

Universal Asynchronous Serial Port.

WAITn:

External transation wait signal.

Apéndice B

Datos del procesador aJ-80.

JEM2 32-bit Direct Execution Java Processor Core

- Native JVM bytecode
- Extended bytecodes for I/O and threading support
- IEEE-754 floating-point arithmetic
- Writeable control store supports custom extended bytecodes

Native Java Threading Support

- Hard real-time, multi-threading kernel in hardware
- Threading operations are atomic including true Java synchronization
- Built-in deterministic scheduling queues
- Directly supports the Real Time Specification for Java (RTSJ)
- Thread to thread yield in less than 1µsec
- Eliminates traditional RTOS layer

Multiple JVM Manager (MJM™)

- Support two independent JVM's
- Brick wall time and space protection
- Support external memory protection

Internal 48KB RAM

- 32KB dedicated data memory
- 16KB microcode memory

Memory Controller

- 8-bit interface
- Six chip selects to support ROM, Flash, SRAM, and peripheral devices

Dual 16550 compatible UARTs

- 128-byte FIFO on Rx and Tx
- Support IrDA physical layer protocol

Three 16-bit Timers/Counters

- Flexible count control and counter I/O
- Pulse Width Modulation (PWM)
- Waveform measurement

Serial Peripheral Interface (SPI)

- Master/Slave operation
- Four external chip selects
- Programmable transfer length

General Purpose I/O Ports

- Twenty-two I/O pins
- I/O programmable on a per-bit basis
- Flexible interrupt generation

Phase Locked Loop (PLL) and Power Management

- Transparent CPU power down when the "run queue" is empty
- Individual peripherals can be deactivated when not in use
- Global clock disable with external wake-up pin

IEEE 1149.1 (JTAG) Interface

- Boundary scan
- Low-level debugger interface
- JPDA Java Debugger Interface

Designed for ultra-low-power operation

- Less than 1mW/MHz power consumption
- Fully static operation up to 66 MHz
- Implemented in 3.3V and 0.25µm CMOS process
- Core operates at 2.5V

Housed in 100-lead QFP package

System Development Support

The aJ-80 processor, bundled with Sun's Java 2 Micro Edition (J2ME) Connected Limited Device Configuration (CLDC) Java runtime system, optimizing application builder, debugging tools and evaluation board provides a complete solution for implementing real-time networked embedded Java applications. Using commercial Java IDEs, application developers can create standalone real-time Java applications totally in Java with the performance and memory efficiency of systems programmed in C and assembly.

An aJ-80 based system can be configured to execute in real-time and/or dynamic environments to support a wide range of applications. The dynamic runtime supports the CLDC Mobile Information Device Profile (MIDP) to allow Java "MIDlets" to be downloaded and executed dynamically in separate JVM environments. Multiple "MIDlets" can be run simultaneously under the control of the Java Applications Manager (JAM) that maintains memory and execution time allocations. aJ-80 can also be configured to execute both real-time and dynamic applications in deterministic time sliced schedule.

The primary components of the development and runtime environments are summarized as follows:

Optimizing Linker/Application Builder

- GUI based application build configuration and control tool - JEM Builder
- Utilizes standard JVM class files generated by commercial Java IDEs
- Statically resolves class files and eliminates unused methods and fields
- Performs bytecode optimizations
- Performs method substitutions (method invokes replaced by extended bytecodes)
- Builds boot tables, class initialization code, and assigns interrupt and trap handlers
- Configures JVM's and memory layout

Java Runtime System

- Java run-time environment based on a J2ME CLDC
- Includes networking classes, storage classes, and Java communications API
- Dynamic runtime includes the JAM (class loader, verifier, scheduler) and GC components
- Device drivers for integrated peripherals and generic physical device interfacing in Java

Application Debugging Tools

- Host-target communications via an IEEE 1149 (JTAG) interface using a simple inexpensive IEEE-1284 cable
- Host-based full featured low-level debugger - Charade (Target level debugger threads and routines are not required)
- Host-based JDI provided to interface to commercial JDI compliant source-level debuggers

Apéndice C

Planos del procesador aJ-80.

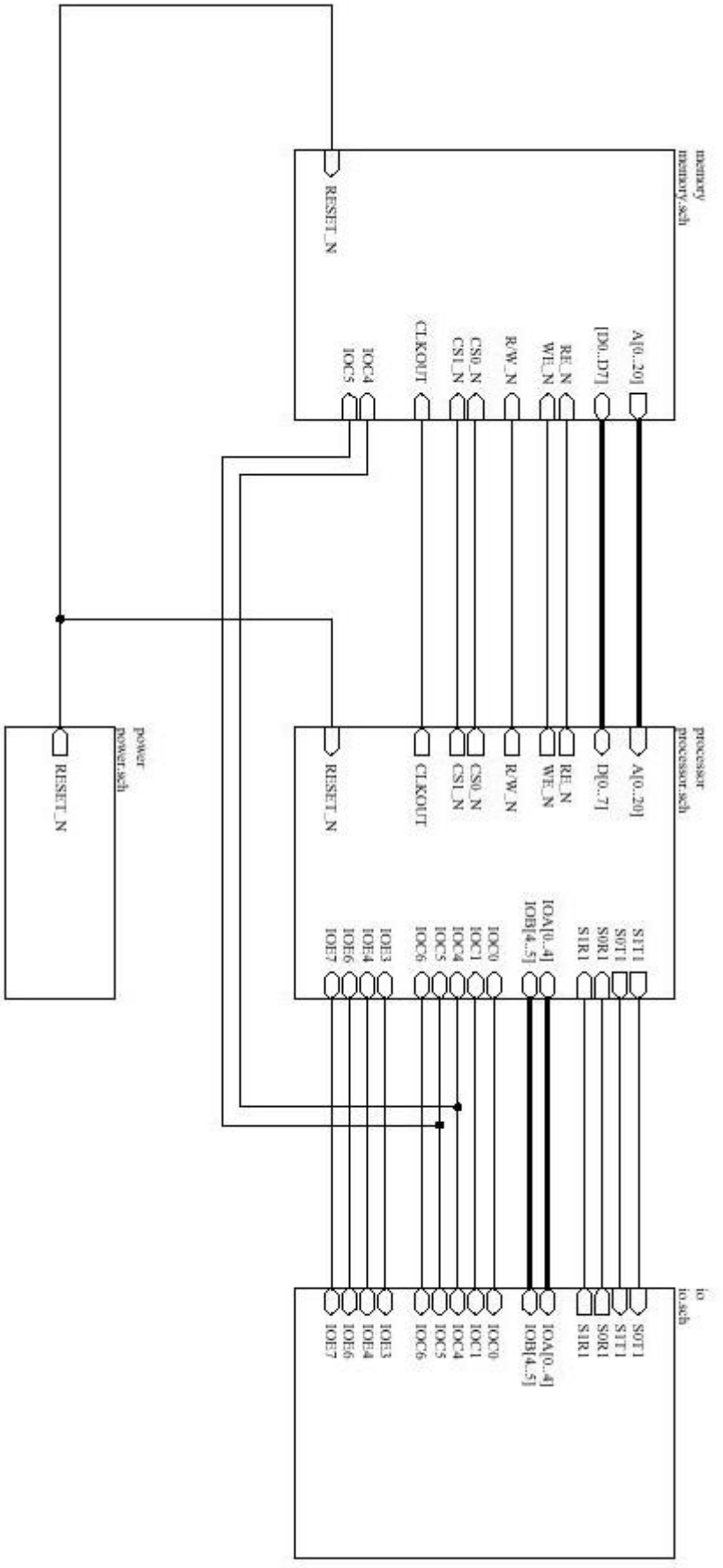


Figura C.1: aJ-80 Test Board.

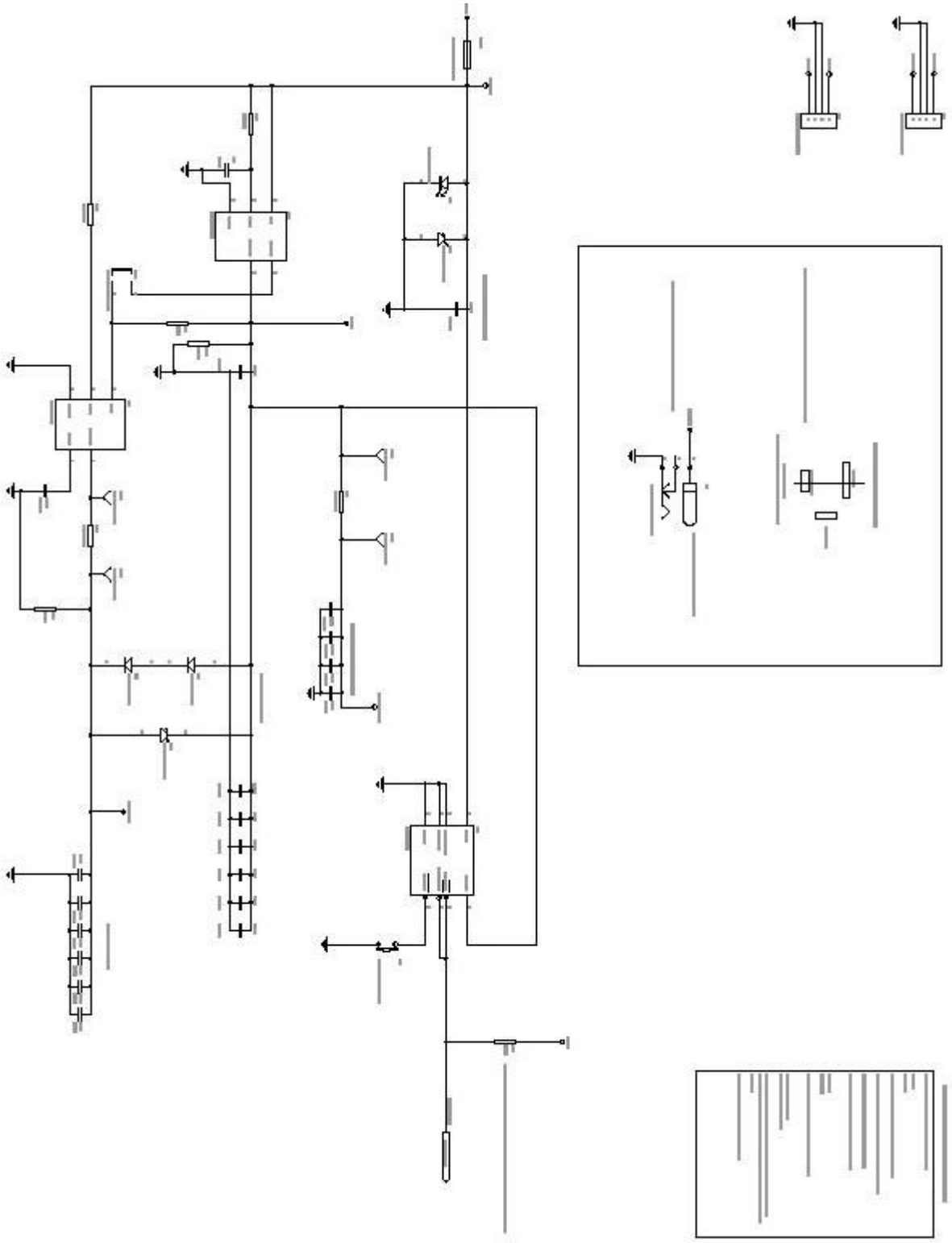
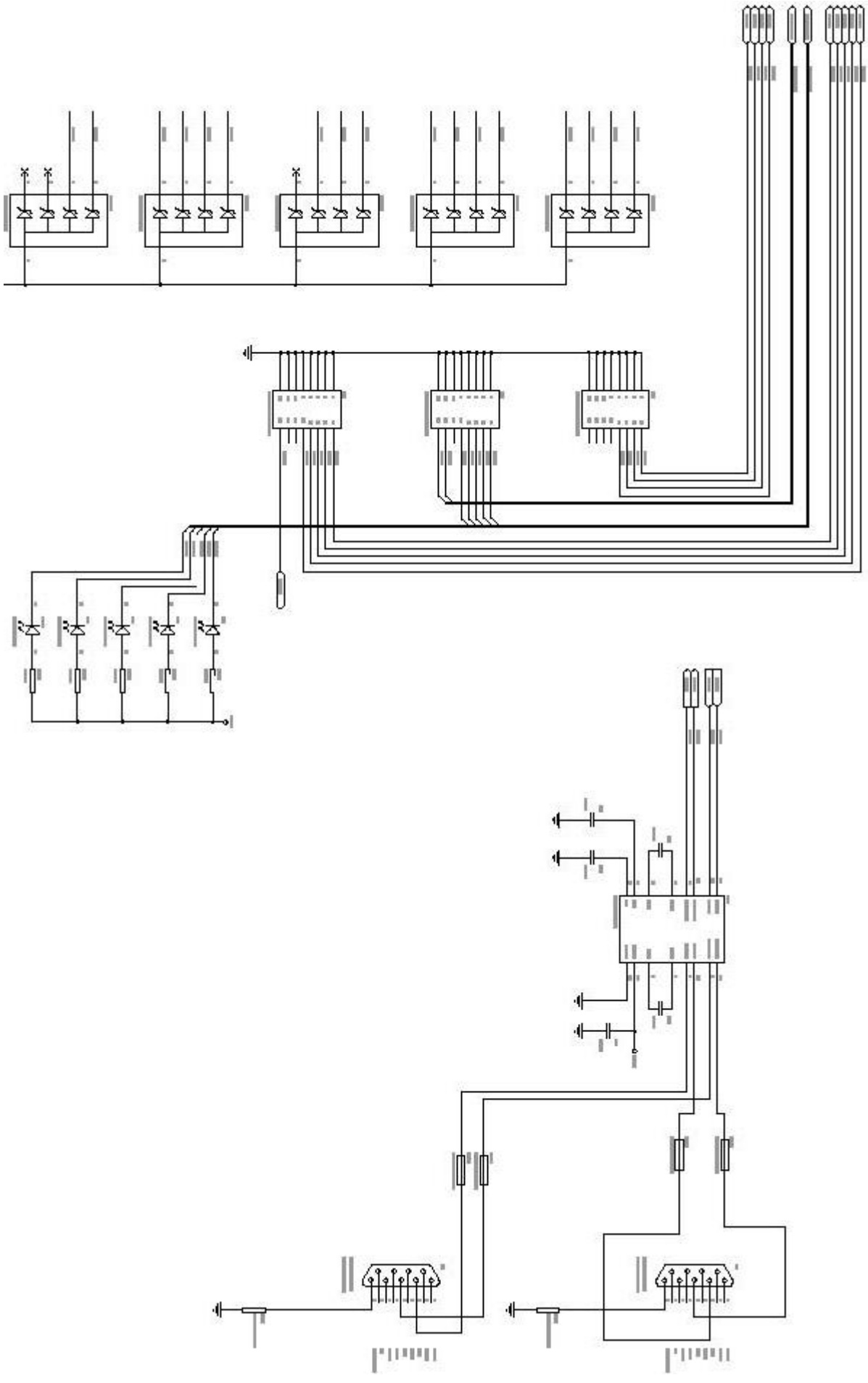


Figura C.2: I/O.



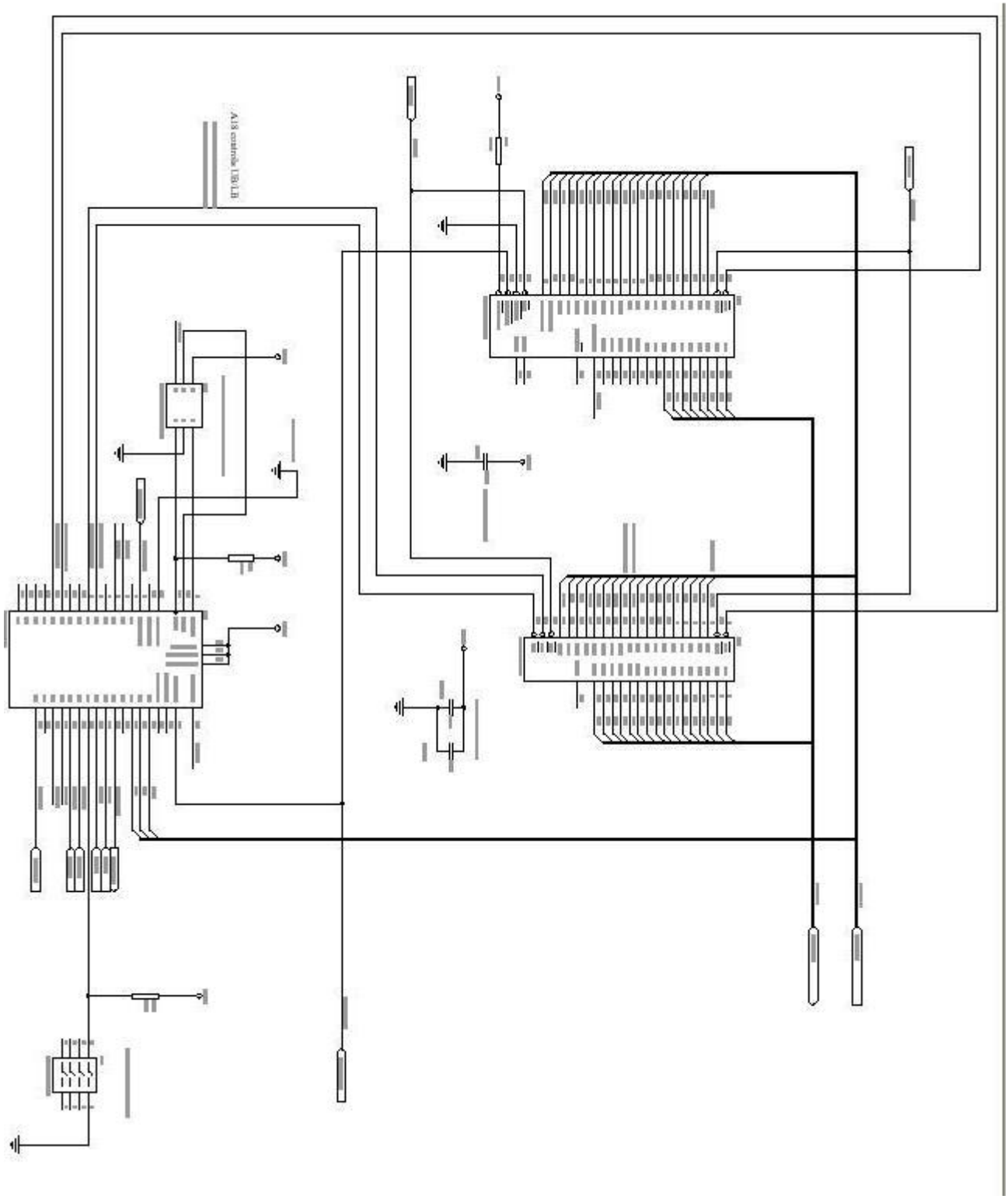
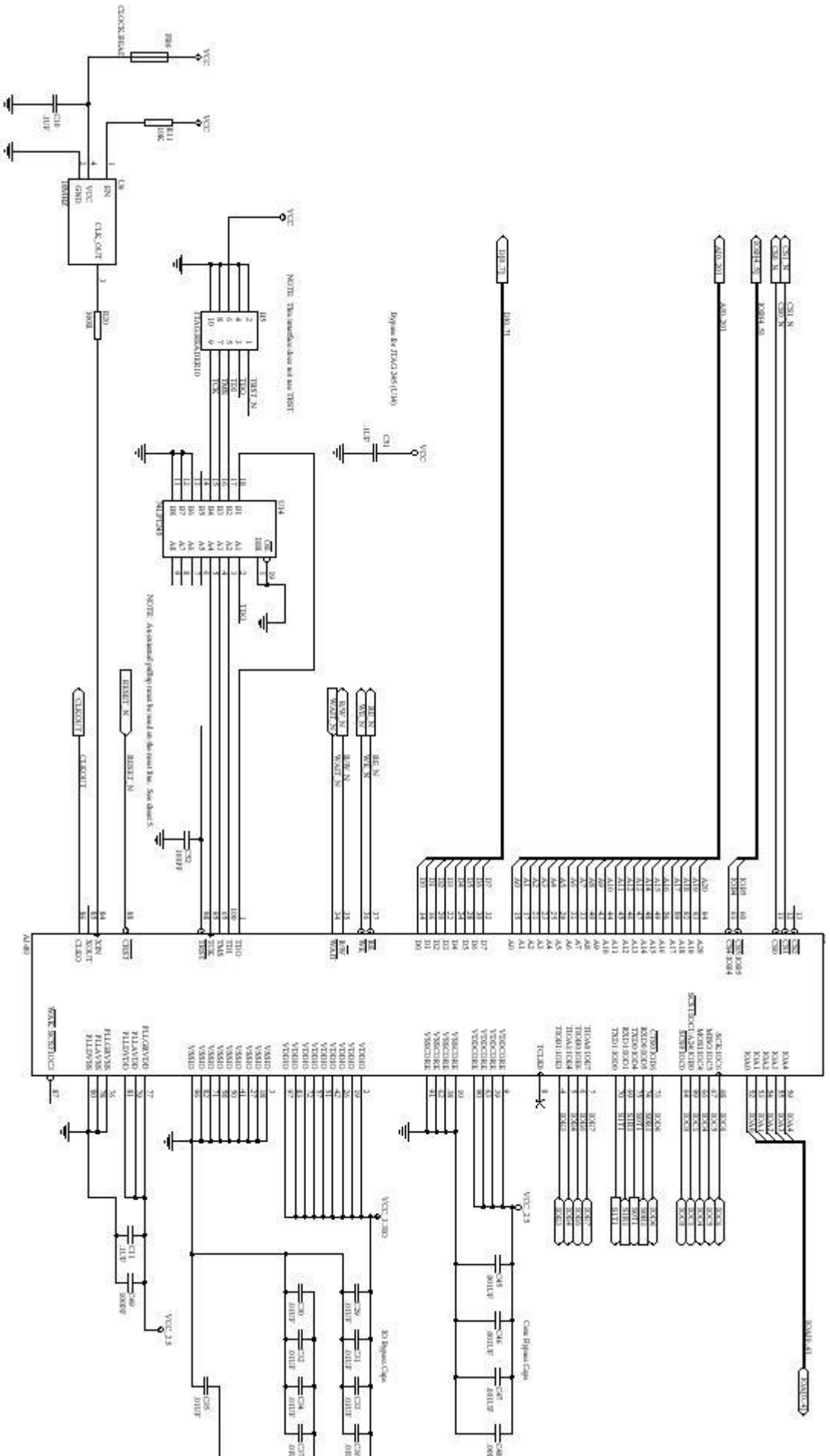


Figura C.4: Memory



Apéndice D

Datos del JStamp.

JStamp™

SYSTRONIX®

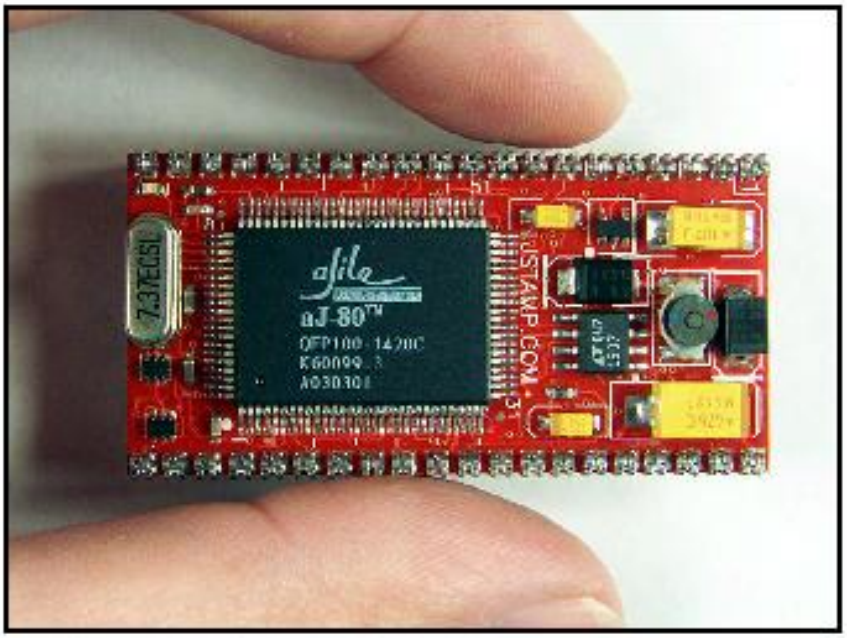
Real-Time Native Java™ Module

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 can execute more than 3 million Java byte codes per second at 74 MHz.



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 32-bit Windows 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 separate 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 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). All I/O pins are TTL-level compatible (thresholds of 0.8 and 2.0 volts) and are 5-volt tolerant. They are compatible with 3V and 5V TTL and 3V CMOS logic. Like all other 3V systems, they are not compatible with 5V CMOS which has a 2.5V threshold.

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. A variety of plug-on modules for Dallas 1-Wire, CAN network, IrDA, RF, graphic LCD, analog I/O, power relays, and more are available or under development. Many current SimmStick modules are JSimm compatible.

Easy JTAG Programming and Debugging

JStamp uses a JTAG interface for programming and debugging with aJile's JEMBuilder and Charade development tools. JTAG pods are available from Systronix and Xilinx. Software requires 32-bit Windows 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.

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. 7.3728 MHz crystal, with Phase Locked Loop control of integer multiples from 1X to 10X of the crystal value. A simple RAM program at 74 MHz consisting of a loop with a logical test and two I/O writes compiles to 7 Java byte codes and executes at 460 KHz, which is over 3 million Java byte codes per second.

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) to 14 VDC on the raw power contact. Or, provide regulated 3.3 VDC +/- 10% on the 3.3V contact. Power requirement is 45-300 mW typical, executing from flash, depending on the supply voltage and processor speed. Lowest power consumption is achieved by providing 3.3 VDC at pin 1.

At 74 MHz typical current is: 57mA @3.3V, 61mA @5V, 37mA @9V, 29mA @12V. At 7.37 MHz typical current is: 13 mA @3.3V, 20 mA @5V, 12 mA @9V, 9 mA @12V.

A 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 at 7.4 MHz.

Serial I/O Dual UARTs at up to 115 Kbaud, 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 & Weight 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. Weight 10.2 grams (0.36 ounce).

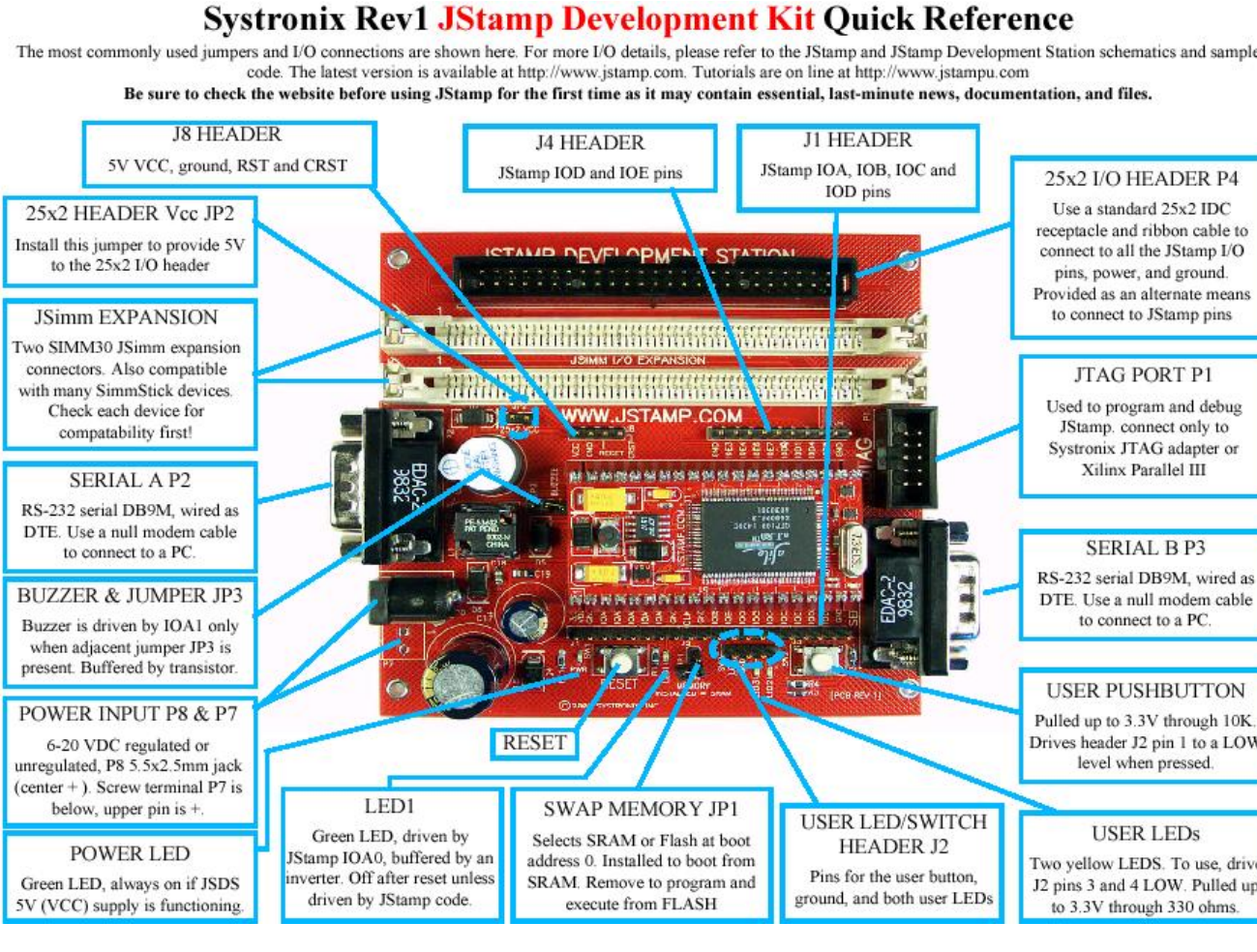
Environmental Commercial temperature 0 to 70 deg C.

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

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


Systronix Rev1 JStamp Development Kit Quick Reference
The most commonly used jumpers and I/O connections are shown here. For more I/O details, please refer to the JStamp and JStamp Development Station schematics and sample code. The latest version is available at http://www.jstamp.com. Tutorials are on line at http://www.jstampu.com
Be sure to check the website before using JStamp for the first time as it may contain essential, last-minute news, documentation, and files.
J8 HEADER
5V VCC, ground, RST and CRST
J4 HEADER
JStamp IOD and IOE pins
J1 HEADER
JStamp IOA, IOB, IOC and IOD pins
25x2 I/O HEADER P4
Use a standard 25x2 IDC receptacle and ribbon cable to connect to all the JStamp I/O pins, power, and ground. Provided as an alternate means to connect to JStamp pins
25x2 HEADER Vcc JP2
Install this jumper to provide 5V to the 25x2 I/O header
JSimm EXPANSION
Two SIMM30 JSimm expansion connectors. Also compatible with many SimmStick devices. Check each device for compatability first!
JTAG PORT P1
Used to program and debug JStamp. connect only to Systronix JTAG adapter or Xilinx Parallel III
SERIAL A P2
RS-232 serial DB9M, wired as DTE. Use a null modem cable to connect to a PC.
SERIAL B P3
RS-232 serial DB9M, wired as DTE. Use a null modem cable to connect to a PC.
BUZZER & JUMPER JP3
Buzzer is driven by IOA1 only when adjacent jumper JP3 is present. Buffered by transistor.
USER PUSHBUTTON
Pulled up to 3.3V through 10K. Drives header J2 pin 1 to a LOW level when pressed.
POWER INPUT P8 & P7
6-20 VDC regulated or unregulated, P8 5.5x2.5mm jack (center +). Screw terminal P7 is below, upper pin is +.
RESET
LED1
Green LED, driven by JStamp IOA0, buffered by an inverter. Off after reset unless driven by JStamp code.
SWAP MEMORY JP1
Selects SRAM or Flash at boot address 0. Installed to boot from SRAM. Remove to program and execute from FLASH
USER LED/SWITCH HEADER J2
Pins for the user button, ground, and both user LEDs
USER LEDs
Two yellow LEDS. To use, drive J2 pins 3 and 4 LOW. Pulled up to 3.3V through 330 ohms.
POWER LED
Green LED, always on if JSDS 5V (VCC) supply is functioning.
WWW.JSTAMP.COM
JSIMM I/O EXPANSION

Install aJile/Systronix SaJe tools from CDROM - install JemBuilder and Charade from the CDROM provided with your development kit. Check the website at http://www.jstamp.com for the latest configuration files, tips or software.

SWAP Memory Jumper - important! - JP1 selects whether JStamp boots up from SRAM (jumper installed) or flash (jumper not installed). You must build (in JemBuilder) for either SRAM or flash, and set the jumper to match. For slightly faster program downloading during development, load into SRAM. (Programs in SRAM are lost when power is removed.) Remove this jumper to enable programming FLASH memory, which persists without system power.

Be sure to build, link, and connect for the correct device - you must specify "JStamp Configuration" (either RAM or Flash, as appropriate) in JemBuilder, and "aJ80" or "JStamp" in Charade in order to correctly build and download programs to JStamp.

JTAG Port - used for loading programs and debugging. Use only the Systronix JTAG adapter and 5x2 100-mil cable, or the Xilinx Parallel III cable. Refer to the schematics for the 5x2 header pinout. Pin 1 is in the lower right corner of the header in the adjacent photograph.

Serial A RS232, a DB9M wired as DTE. A straight-through serial cable will connect to a DCE device such as a modem or LCD. A null modem cable is needed to connect to a PC.

Serial B RS232, a DB9M wired as DTE. A straight-through serial cable will connect to a DCE device such as a modem or LCD. A null modem cable is needed to connect to a PC.

Power Supply is 6-20 volts, DC unregulated or regulated. The input jack is 5.5 x 2.5 mm, center positive or negative. The JStamp Development Station uses an efficient, wide input range switching supply to generate 5 VDC. (JStamp has an onboard 3.3V converter which uses this 5V as its input.) Supply current decreases nearly linearly as supply voltage increases due to the power conversion in the regulator. Recommended power includes the Systronix 1A 12 VDC cube. The regulator can supply at least 500 mA for additional components or expansion cards.

JSimm and SimmStick® Expansion Two SIMM30 connectors are provid JSimm is compatible with many SimmStick® devices. Check each specific SimmStick® device for compatability first!

JStamp I/O Headers Header pins (0.025 inches square on 0.100 inch cent are provided for all JStamp I/O pins. These I/O pins are 3.3V supply, 5V-tolerant, TTL-level compatible. They will drive 5V TTL I/O directly (but w not properly drive 5V CMOS devices). *Some I/O pins have multiple functio please refer to the JStamp I/O pinout description before selecting pins to connect to your own devices.* Note that the absolute maximum voltage on a tolerant raw JStamp input or I/O pin is 6 VDC. Maximum voltage on a JSta output pin is 5 VDC. Higher voltages can permanently damage JStamp.

25x2 I/O Header A standard 25x2 polarized header (0.025 inches square p on 0.100 inch centers) is provided as an alternate means of connecting to JStamp I/O pins. Keep this cable short, and provide your own static or surge protection as needed. Applying an excessive voltage or static to this header permanently damage JStamp pins. JP2 when present provides 5VDC from t Development Station to the 25x2 header pin 49, through schottky diode D7. The diode prevents power on the 25x2 header pin from driving the JSDS.

Power LED Driven by JStamp's pin IOA0. When IOA0 is high, the JStamp heartbeat LED is lit and so is the Development Station Power LED.

Buzzer Driven by JStamp's pin IOA1. When IOA1 is high, (during and afte reset, for example) the buzzer is off. Remove JP3 if you do not want to driv the buzzer.

JStamp™ by SYSTRONIX®

JStamp, JSimm, and 25x2 Header Pin Numbering and Description (also see important notes which follow)

JStamp # (note 1,4)	JSimm # (note 5)	25x2 #	Name	I/O (note 6)	Description JSDS=JStamp Development Station
1	none	none	3.3V	I/O	JStamp's power converter provides output of 3.3V @ 100 mA on this pin for your use, when you power JStamp's VRAW input (pin 40). Or you can drive this pin with regulated 3.3 VDC +/- 5%. (JSimm does not have a 3.3V signal, for SimmStick compatability). (note 7)
can be 40	7	49	5.0V	-	5V can be used as JStamp power input Vraw (pin 40).
2, 8, 10, 20, 21, 30, 39	9	2,4,6...50	GND	-	All even 25x2 header pins are GND (note 3)
3	19	39	IOA4	I/O	24 mA sink/source I/O pin
4	18	41	IOA3	I/O	24 mA sink/source I/O pin
5	17	43	IOA2	I/O	24 mA sink/source I/O pin
6	16	45	IOA1	I/O	24 mA sink/source I/O pin. Drives the JSDS buzzer through an inverter and transistor, if JSDS JP3 is present.
7	15	47	IOA0	I/O	24 mA sink/source I/O pin. Drives the JStamp heartbeat LED through a transistor on JStamp. Also drives the JSDS LED1 through an HCT04 inverter on the JSDS board.
9	6	3	CLKO	O	aJ-80 Clkout signal, a programmable divider output.
11	11	35	IOB5	I/O	8 mA sink/source I/O pin
12	10	37	IOB4	I/O	8 mA sink/source I/O pin
13	22	23	IOC6	I/O	8 mA sink/source I/O pin. Also SPI Transfer Clock.
14	21	25	IOC5/FA1	I/O	8 mA sink/source I/O pin. Also functions as SPI MISO when in SPI master mode and MOSI when a SPI slave. Also used as flash address 1 only when actually programming flash.
15	20	27	IOC4/FA0	I/O	8 mA sink/source I/O pin. Also functions as SPI MOSI when in SPI master mode and MISO when a SPI slave. Also used as flash address 0 only when actually programming flash.
16	3	29	IOC3	I/O	8 mA sink/source I/O pin, or SPI Slave Chip Select 3
17	2	31	IOC1	I/O	8 mA sink/source I/O pin, or SPI Slave Chip Select 1
18	1	33	IOC0	I/O	8 mA sink/source I/O pin, or SPI Slave Chip Select 0 or slave mode select (when JStamp is an SPI slave).
19	29	13	IOD6	I/O	8 mA sink/source I/O pin
22	12	15	IOD5/RXDA	I/O	8 mA sink/source I/O pin, or UARTA RXD (TTL- not RS232 level - note 8)
23	13	17	IOD4/TXDA	I/O	8 mA sink/source I/O pin, or UARTA TXD (TTL- not RS232 level - note 8)
24	28	19	IOD1/RXDB	I/O	8 mA sink/source I/O pin, or UARTB RXD (TTL- not RS232 level - note 8)
25	27	21	IOD0/TXDB	I/O	8 mA sink/source I/O pin, or UARTB TXD (TTL- not RS232 level - note 8)
26	26	5	IOE7	I/O	8 mA sink/source I/O pin
27	25	7	IOE6	I/O	8 mA sink/source I/O pin
28	24	9	IOE4	I/O	8 mA sink/source I/O pin
29	23	11	IOE3	I/O	8 mA sink/source I/O pin
31	none	none	SWAP_MEM(L)	I	Leave floating high to select and/or program flash memory at memory location 0 (this is the typical state of this pin). Pull this pin low to place SRAM at location 0 (normally used only in development).
32	none	none	JTAG_TDO	O	JTAG Test Data Output (note 2)
33	none	none	JTAG_TDI	I	JTAG Test Data Input (note 2)
34	none	none	JTAG_TMS	I	JTAG Test Mode Select (note 2)
35	none	none	JTAG_TCK	I	JTAG Test Clock input (note 2)
36	8	1	CRST(L)	I/O	Open-drain reset to/from JStamp. Use this signal to reset JStamp from external logic or to reset your external logic when JStamp resets.
37	5, 14, 30	none	N/C	-	Do not connect to this pin. May be used in a future JStamp version.
38	none	none	RESET_PB(L)	I	Input to JStamp from a reset pushbutton. Circuitry on JStamp debounces this. Use this signal to reset JStamp from a switch.
40	4	none	VRAW	-	Power JStamp with 5-14 VDC on this pin if you do not provide regulated 3.3 VDC on JStamp pin 1.

Pinout Notes

Note 1: JStamp I/O Pin Voltages and Logic Thresholds - JStamp GPIO pins are 3.3V max Voh, compatible with TTL levels, and are 5V I/O tolerant. They interface with no additional circuitry to 3.3V and 5V TTL-level devices. They will not drive 5V CMOS outputs directly.

Note 2: JTAG pins must be connected only to a Systronix JTAG adapter or Xilinx Parallel III programming adapter. Any other connection voids your warranty and may damage JStamp.

Note 3: Ground - JStamp grounds are all connected together, so in a minimal system you only need to connect at least one to your system.

Note 4: GPIO Pin Function - Each GPIO pin may be individually configured as input or an output. Every GPIO pin may also be configured to generate a CPU interrupt. Interrupt flexibility is provided by allowing interrupts to be triggered on a rising edge, falling edge, either edge, high level, or low level. To minimize pin-count most of the GPIO signals are shared with other I/O signals of the aJ-80. On a reset the shared signals are configured as GPIO inputs. Operation of the shared signals is controlled with the I/O configuration registers.

Note 5: JSimm pins - JStamp uses some GPIO pins for specific purposes when using the JSimm interface.

Note 6: I/O Direction is from the viewpoint of JStamp. I.e., an input is an input to JStamp.

Note 7: JStamp power can be either regulated 3.3 VDC +/- 5%, or unregulated 4.5-14 VDC +/- 5%. If you power JStamp's VRAW input (pin 40) then JStamp's 3.3V I/O (pin 1) provides output of 3.3V @ 100 mA for your use. If you provide regulated 3.3 VDC on pin 1, do not connect VRAW. *Under no circumstances should power be applied to both Pin 1 and Pin 40.* When JStamp is plugged into the Development Station, it receives 5VDC on pin 40.

JStamp Documentation and Resources

aJ-80 and aJ-100 - both aJile controllers share the same 32-bit CMOS core differ only in pinout (the aJ-100 is a larger package so brings out more pins). The aJ-80 has an 8-bit external data bus - the aJ-100 has 32-bits. They share same technical reference, Java edition and profile support, and so forth.

AJ-100 Reference Manual - available online at http://www.ajile.com/aj100.htm as a PDF document. This is the definitive source for information about inner workings of the aJile controllers.

On-line Support Groups: there is a JStamp Yahoo user group at http://groups.yahoo.com/group/jstamp as well as other third party support an information groups. Links to those may be found at http://www.jstamp.com

Java, J2ME, CLDC, and RealTime Java Information: J2ME and CLDC information and packages are available online from Sun at http://www.sun.com/software/communitysource/j2me/cldc/download.html a there are also links to Java resources at http://www.jstampu.com

JStamp in Robotics: www.jcx.systronix.com

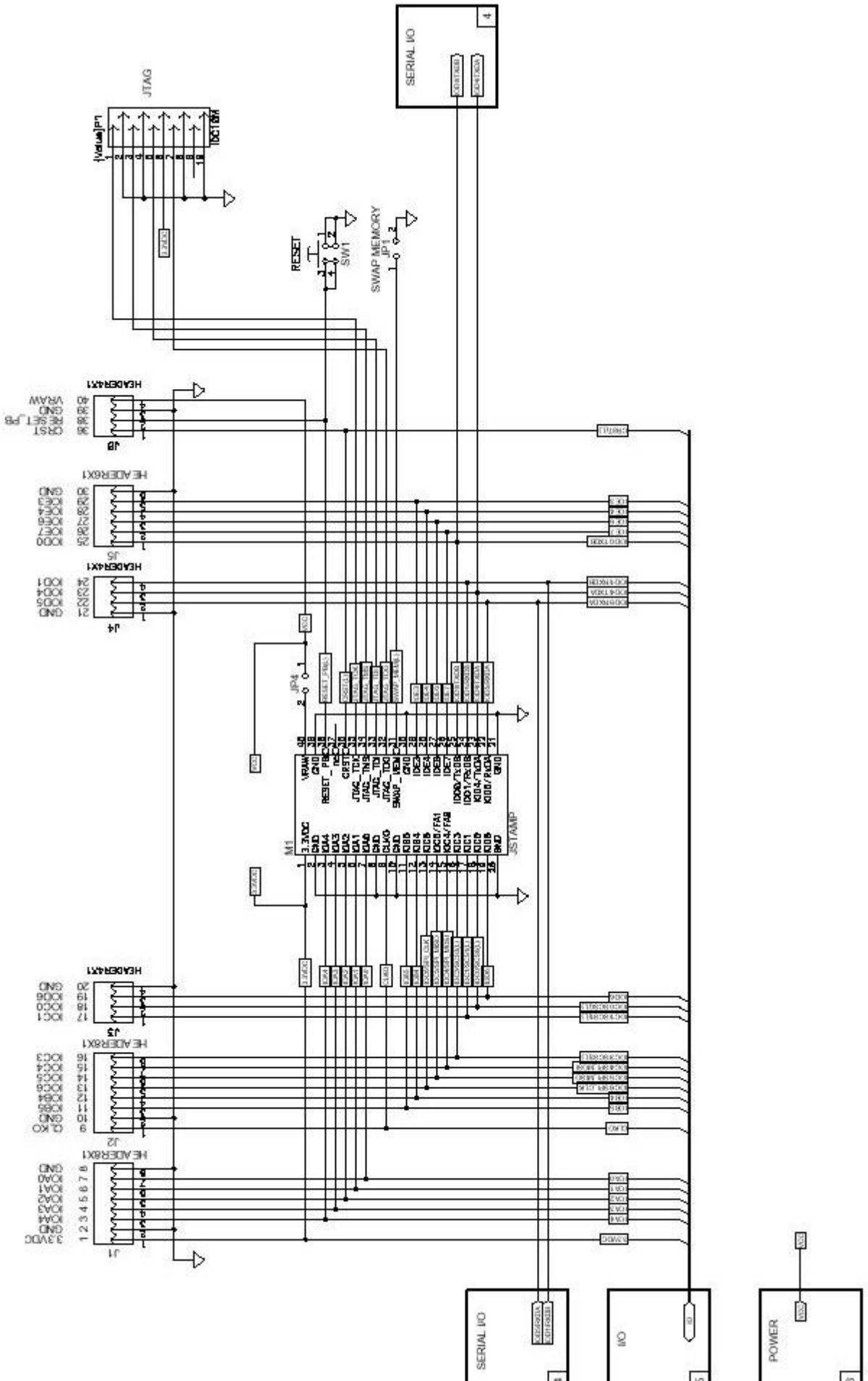
JCX is a JStamp-based platform with LEGO®-compatible inputs and output (also usable with other common robotic sensors and actuators). App notes ar on line here - for example, driving a sonar rangefinder from JStamp.

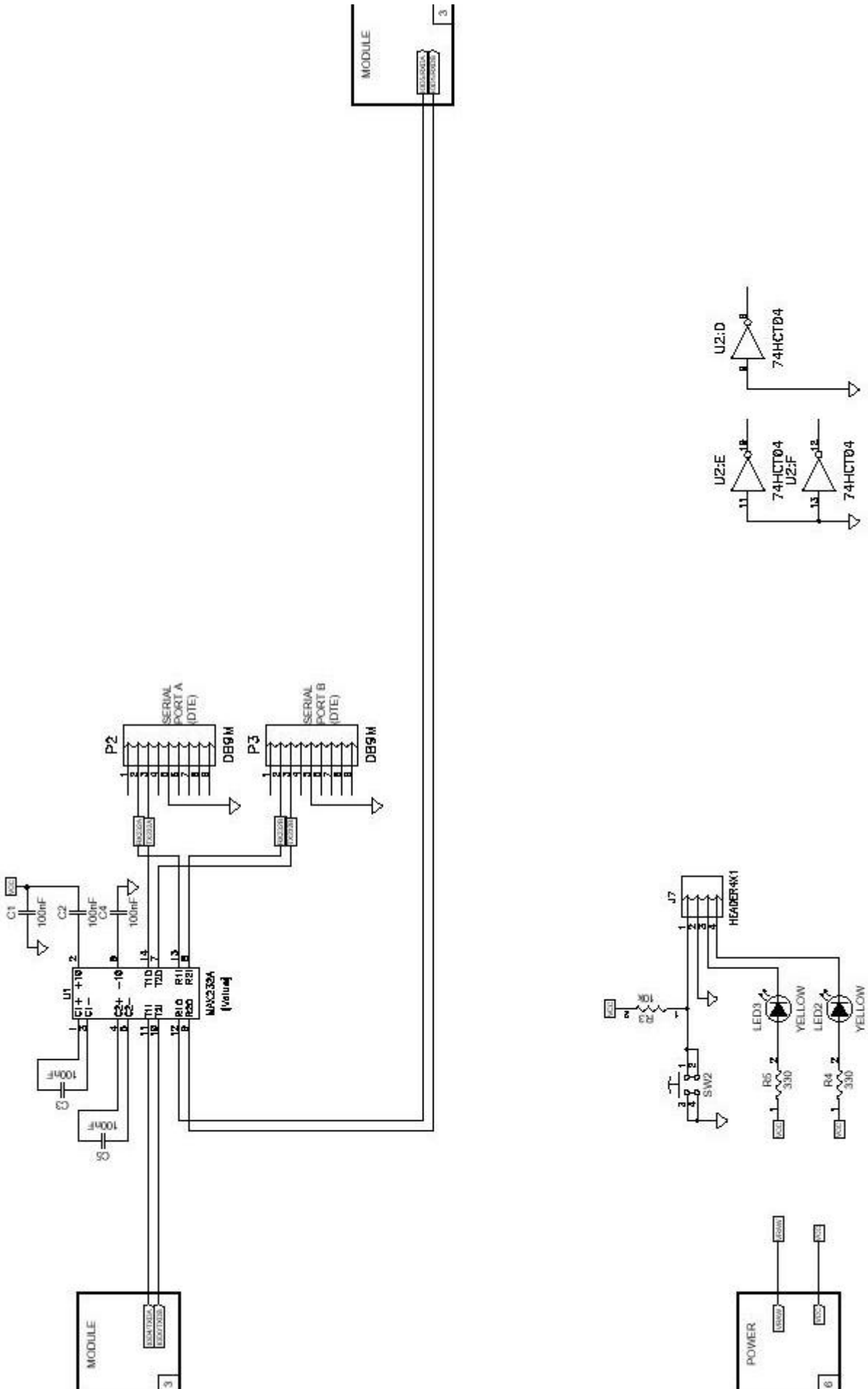
Tutorial & Examples, Educational Use: www.jstampu.com

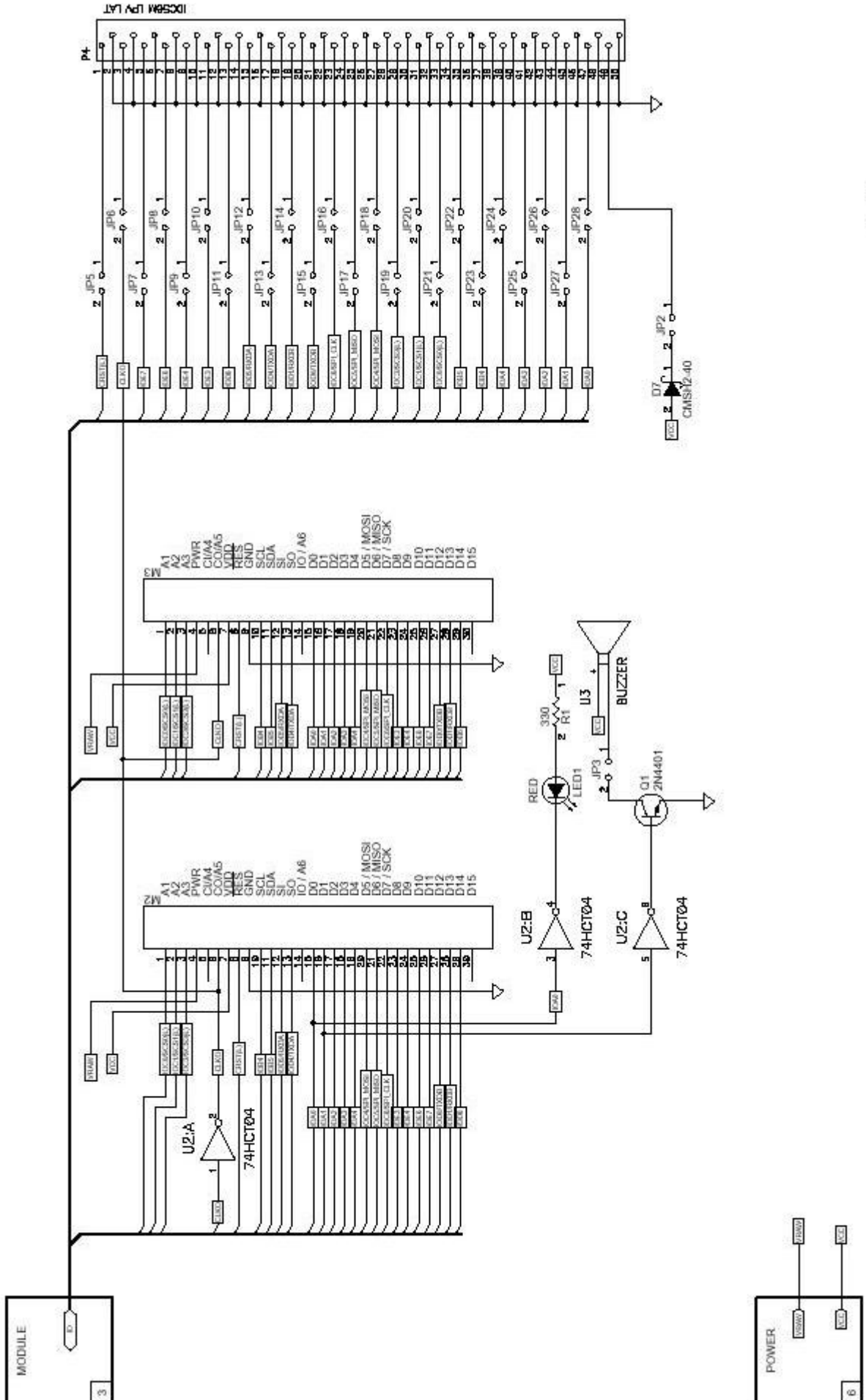
An ongoing tutorial and sample programs are available on our web site at www.jstampu.com. There are also links to use of JStamp and related produc in education, particularly university and college programs.

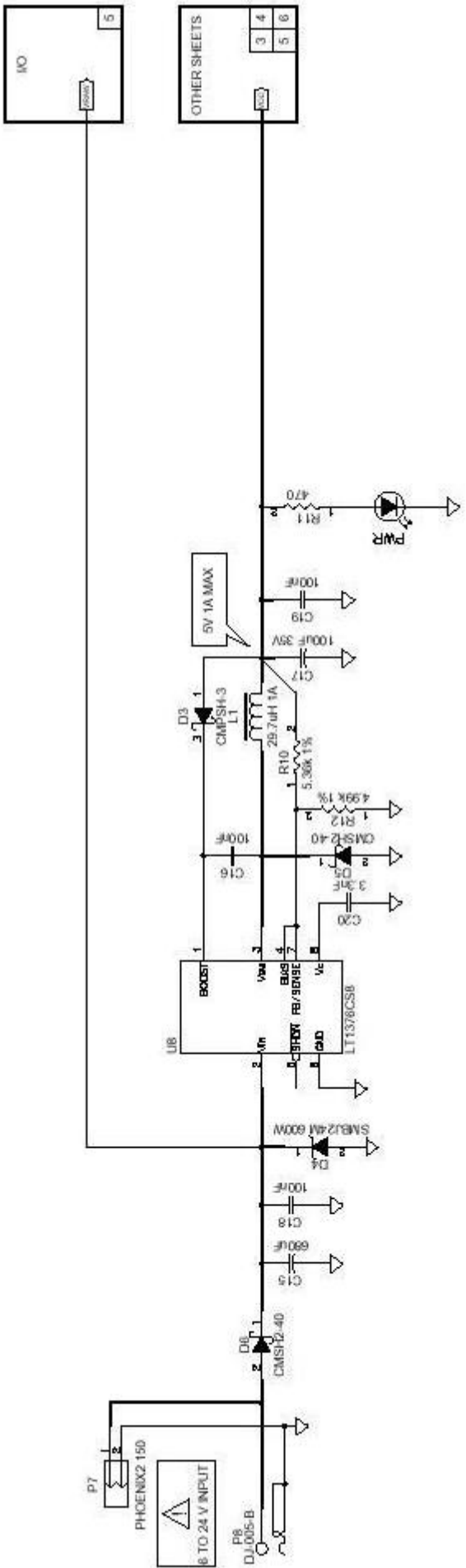
Apéndice E

Planos del JStamp.


SERIAL I/O
JTAG
RESET
SW1
SWAP MEMORY
JP1
JP4
HEADER4X1
HEADER6X1
HEADER8X1
M1
JSTAMP
SERIAL I/O
I/O
POWER

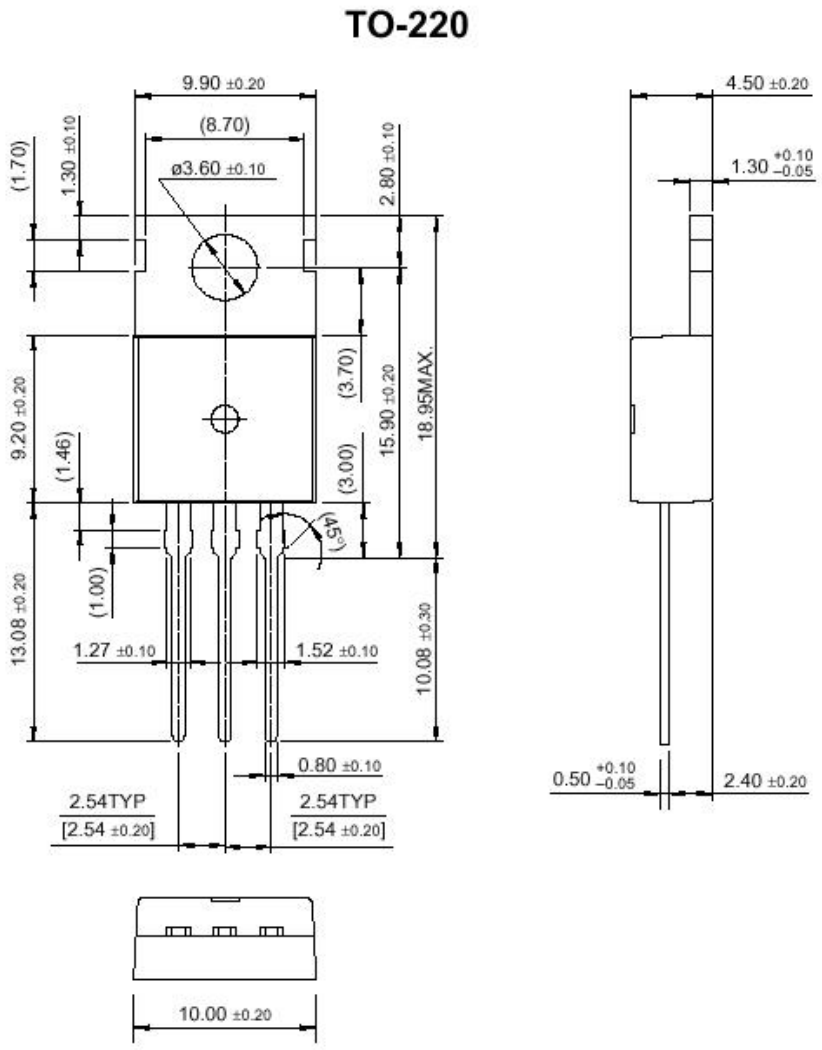

MODULE
3
U2:D
74HCT04
U2:E
74HCT04
U2:F
74HCT04
P2
SERIAL PORT A (DTE)
DB9M
P3
SERIAL PORT B (DTE)
DB9M
C1
100nF
C2
100nF
C4
100nF
U1
MAX232A
C3
100nF
C5
100nF
J7
HEADER4X1
R5
330
R4
330
LED3
YELLOW
LED2
YELLOW
SW2
MODULE
3
POWER
6


P4
JP5
JP6
JP7
JP8
JP9
JP10
JP11
JP12
JP13
JP14
JP15
JP16
JP17
JP18
JP19
JP20
JP21
JP22
JP23
JP24
JP25
JP26
JP27
JP28
JP2
D7
CMSH2-40
M3
M2
U2:A
U2:B
U2:C
74HCT04
U3
BUZZER
330
R1
RED
LED1
JP3
Q1
2N4401
MODULE
POWER


I/O
5
OTHER SHEETS
3
4
5
6
470
R11
PWR
5V 1A MAX
100nF
C19
100uF 35V
C17
D3
CMPSH-3
L1
29.7uH 1A
R10
5.36K 1%
R12
4.99K 1%
CMSH2-40
D5
100nF
C16
3.3nF
C20
U8
BOOST
VSW
BIAS
RT/SYNC
SHDN
GND
VC
LT1376CS8
SMBJ24A 600W
D4
100nF
C18
680uF
C15
D6
CMSH2-40
P7
PHOENIX2 150
6 TO 24 V INPUT
P8
DJ-005-B

Apéndice F

Regulador de tensión MC7808C.


TO-220
9.90 ±0.20
(8.70)
ø3.60 ±0.10
(1.70)
1.30 ±0.10
2.80 ±0.10
(3.70)
15.90 ±0.20
18.95MAX.
9.20 ±0.20
(1.46)
(3.00)
45°
(1.00)
13.08 ±0.20
1.27 ±0.10
1.52 ±0.10
10.08 ±0.30
0.80 ±0.10
2.54TYP
[2.54 ±0.20]
2.54TYP
[2.54 ±0.20]
10.00 ±0.20
4.50 ±0.20
1.30 +0.10 −0.05
0.50 +0.10 −0.05
2.40 ±0.20

ELECTRICAL CHARACTERISTICS (V_{in} = 14 V, I_O = 500 mA, T_J = T_{low} to T_{high} [Note 1], unless otherwise noted.)

Characteristic	Symbol	MC7808C			Unit
		Min	Typ	Max	
Output Voltage (T_J = 25°C)	V_O	7.7	8.0	8.3	Vdc
Output Voltage (5.0 mA ≤ I_O ≤ 1.0 A, P_D ≤ 15 W) 10.5 Vdc ≤ V_{in} ≤ 23 Vdc	V_O	7.6	8.0	8.4	Vdc
Line Regulation, T_J = 25°C, (Note 2) 10.5 Vdc ≤ V_{in} ≤ 25 Vdc 11 Vdc ≤ V_{in} ≤ 17 Vdc	Reg_{line}	 – –	 6.0 1.7	 32 16	mV
Load Regulation, T_J = 25°C (Note 2) 5.0 mA ≤ I_O ≤ 1.5 A	Reg_{load}	–	1.4	35	mV
Quiescent Current	I_B	–	3.3	8.0	mA
Quiescent Current Change 10.5 Vdc ≤ V_{in} ≤ 25 Vdc 5.0 mA ≤ I_O ≤ 1.0 A	ΔI_B	 – –	 – –	 1.0 0.5	mA
Ripple Rejection 11.5 Vdc ≤ V_{in} ≤ 18 Vdc, f = 120 Hz	RR	56	62	–	dB
Dropout Voltage (I_O = 1.0 A, T_J = 25°C)	$V_I - V_O$	–	2.0	–	Vdc
Output Noise Voltage (T_A = 25°C) 10 Hz ≤ f ≤ 100 kHz	V_n	–	10	–	μV/V_O
Output Resistance f = 1.0 kHz	r_O	–	0.9	–	mΩ
Short Circuit Current Limit (T_A = 25°C) V_{in} = 35 Vdc	I_{SC}	–	0.2	–	A
Peak Output Current (T_J = 25°C)	I_{max}	–	2.2	–	A
Average Temperature Coefficient of Output Voltage	TCV_O	–	–0.4	–	mV/°C

NOTES: 1. T_{low} = –40°C for MC78XXAC, C T_{high} = +125°C for MC78XXAC, C

2. Load and line regulation are specified at constant junction temperature. Changes in V_O due to heating effects must be taken into account separately. Pulse testing with low duty cycle is used.

Índice de figuras

Índice alfabético

Parte VI

Bibliografía.

Bibliografía.

[1] "Error eliminating rapid ultrasonic firing for mobile robot obstacle avoidance" Johan Borestein, Y.Koren. IEEE Transactions on robotics and automation, Vol11, Nº 1.

[2] "reflections on modelling a sonar range sensor" Gregory Dudek. McGill Research Centre for Intelligents Machines. McGill University of Montreal.

[3] "The Real-Time specification for Java". www.rtj.org

[4] "Direct sonar sensing for mobile robot navigation" J.J Leonard, H.F. Durrant-Whyte. Kluwer Academic Publishers.

[5] "aJile Systems: Low Power Direct-Execution Java Microprocesro for Real-Time and Networked Embedded Applications" David S. Hardin

[6] "The Java programing language" Ken Arnold, James Gosgling, Dvid Holmes. Addison Wesley,2000.

[7] "Mobile information device profile" Version 1.0. JSR-37 Expert Group, Java 2 Platform, ME.

[8] "The Java Community Process" Sun Microsystems, Inc