Departamento de Ingeniería de Sistemas y Automática

Diseño e implementación de un bus de campo para automatización de un helicóptero de aeromodelismo

Autor: D. Raúl Muñoz Monís
Tutor: D. Ismael Alcalá Torrego

Septiembre 2003

A mis padres,
y a Che.

1.Objetivo del proyecto.

1.1 Marco General

El marco general de este proyecto es la adaptación para el control de un helicóptero de aeromodelismo. El control de un helicóptero es complicado debido a sus no linealidades y por constituir sistemas multivariables acoplados en los que los grados de libertad superan el número de actuadores.

A bordo del helicóptero se trata de diseñar e implementar tarjetas electrónicas embarcadas basadas en uCs (microcontroladores) de ATMEL que permitan probar leyes de control avanzadas en sistemas de control de vuelo (FCS). Para ello habrá que tener siempre en cuenta las limitaciones que plantea el trabajar en un helicóptero de estas características y se buscará que las tarjetas sean pequeñas, ligeras, precisas, de bajo consumo y bajo coste.

Unidad de Medida Inercial (IMU)

Sistema de sensores inerciales compuesto por acelerómetros y giróscopos que proporcionan datos a un uC encargado de procesarlos para obtener una estimación de la actitud de la aeronave (*picth, yaw* y *roll*). Esta información se envía a través de un bus al procesador principal del sistema de control de vuelo.

Sistema de posicionamiento global (GPS)

Sistema de posicionamiento global utilizando un circuito integrado receptor GPS que proporciona la longitud, latitud a un uC encargado de procesar los datos y obtener una estimación de la posición del helicóptero. Esta información se envía a través de un bus al procesador principal del sistema de control de vuelo.

Unidad de comunicaciones a bordo.

El objetivo es realizar una tarjeta que envíe y reciba datos digitales de una estación PC en tierra. Los datos enviados serán las medidas de actitud y posición, que serán facilitadas por el FCS o directamente por el subsistema sensor. Los datos recibidos de la estación PC serán las señales de control para los servomecanismos, que enviará por el bus al FCS o directamente a los propios servomecanismos.

Sistema de control de propulsión

Sistema encargado de regular la velocidad del rotor actuando sobre el motor de combustión. Para ello se buscarán los sensores que permitan obtener medidas de la

velocidad de giro del rotor y se integrarán con un microcontrolador o un circuito integrado PID. Los datos de las consignas de velocidad de giro del rotor provienen de un bus conectado al FCS.

Altímetro

Sistema encargado de medir la distancia del helicóptero al suelo utilizando sensores redundantes (láser y ultrasonidos).

Protocolo de datos de la aeronave

Diseño de módulos para realizar el protocolo de comunicaciones entre los distintos sistemas de la aeronave utilizando en la medida de lo posible el estándar MIL-STD-1553B.

Sistema de control de vuelo (FCS)

Sistema basado en uC de 32 bits para del control de posición y actitud del helicóptero. Se encargará de recoger información de los sistemas sensores y enviar las señales de control a los servos de la aeronave.

Tarjeta de comunicaciones para el sistema en tierra.

Esta tarjeta se conecta a la estación PC (preferiblemente por USB o ETHERNET) y permite la transmisión y recepción de datos con el helicóptero de pruebas.

Entorno básico para simulación 3D de aeronaves

En este proyecto se va a desarrollar un entorno de simulación para las primeras fases de diseño de modelos dinámicos y sistemas de control de vuelo de aeronaves. Se pretende que simule los cambios en los modelos de forma rápida.

Entorno de control para la estación en tierra.

Desarrollo de funciones S para SIMULINK que permitan la implementación rápida de controladores utilizando técnicas de prototipaje rápido para PC con sistema operativo LINUX en tiempo real o WINDOWS.

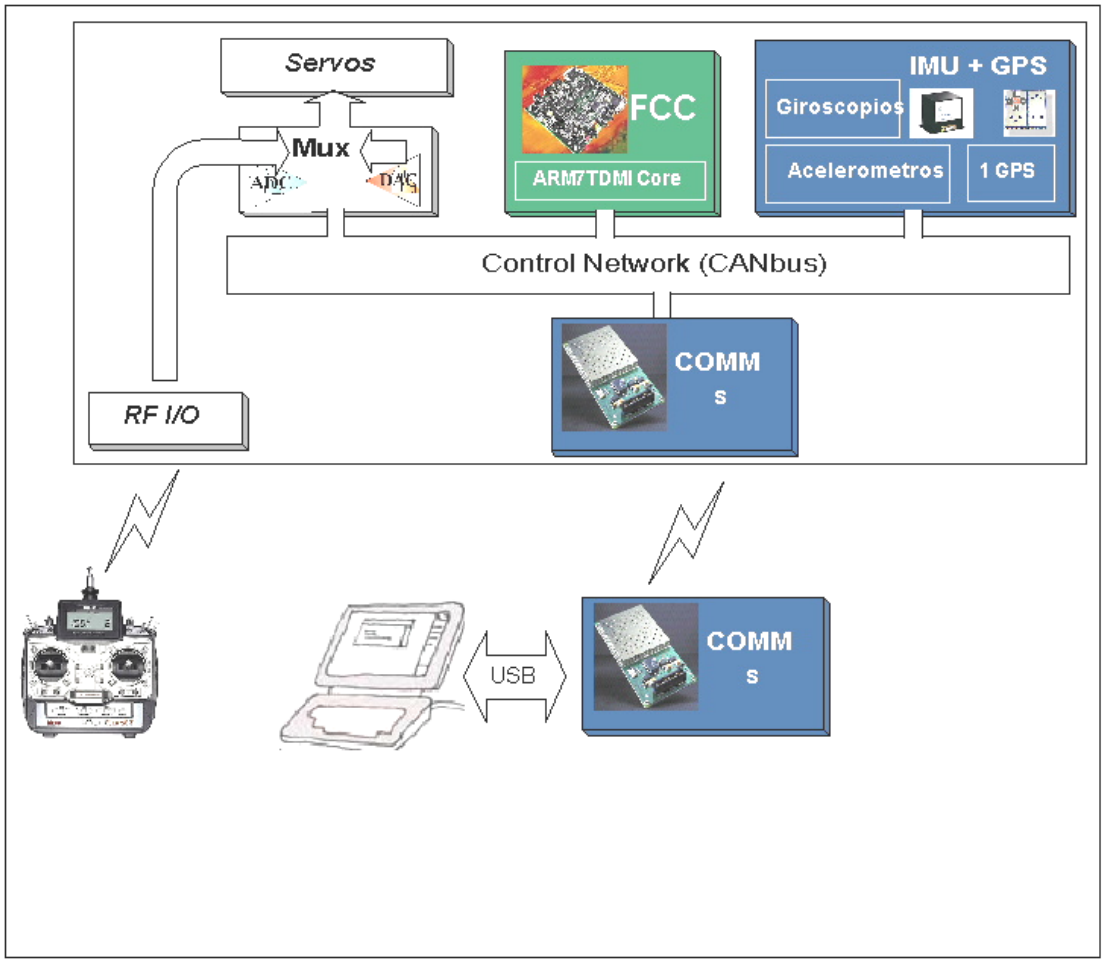


Figura 1.1-1 Diagrama de bloques general.

1.2 Objetivo particular de este proyecto y especificaciones.

El objetivo de este proyecto en cuestión dentro del marco general es la realización de la Unidad de Comunicaciones tanto en el lado del helicóptero como en el lado del PC. La idea es mantener los mecanismos habituales de radio control, y contar con un conmutador que seleccione entre las actuaciones dadas por el radio control manual y las actuaciones provenientes del FCS. El objetivo final es que los algoritmos del FCS se ejecuten en una tarjeta a bordo del helicóptero, pero inicialmente estos algoritmos se ejecutan en la estación PC en tierra, por este motivo es tan elevado el flujo de información a transmitir.

Figura 1.2-1

Detallando un poco mas a modo de bloques en el lado del helicóptero:

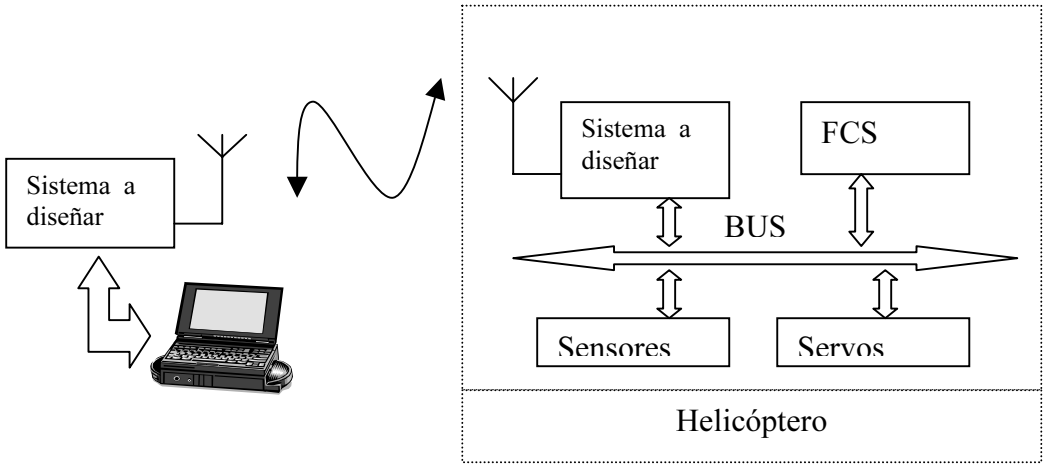


Figura 1.2-2

Lado Helicóptero

En el lado del helicóptero el objetivo es realizar una tarjeta que envíe y reciba datos digitales de una estación terrena. Los datos a enviar serán las medidas de actitud y posición, las cuales pueden ser facilitadas por el FCS, o directamente del subsistema sensor. Los datos recibidos de la estación PC serán las señales de control para los servomecanismos, que enviará por bus al FCS o directamente al subsistema actuador.

En cuanto a los requisitos de velocidad sería deseable un tiempo de muestreo de 1 ms para cada señal aunque esta especificación puede relajarse a 10 ms (o incluso 100 ms o 1s para determinadas señales).

Las especificaciones iniciales son las siguientes:

- Mínimo de 12 señales de 12 bits:

Señales en el canal descendente	Velocidad de giro del motor	Tmuestreo: 100 ms
	Paso Colectivo	Tmuestreo: 1 ms
	Paso Cíclico 1	Tmuestreo: 1 ms
	Paso Cíclico 2	Tmuestreo: 1 ms
	Señal de Cola	Tmuestreo: 1 ms
	Throttle	Tmuestreo: 1 ms
Señales en el canal ascendente	Actitud 1	Tmuestreo: 1 ms
	Actitud 2	Tmuestreo: 1 ms
	Actitud 3	Tmuestreo: 1 ms
	Posición x	Tmuestreo: 1 s
	Posición y	Tmuestreo: 1 s
	Posición z	Tmuestreo: 1 ms

Tabla 1.2 – 1: Especificaciones iniciales.

Unos primeros cálculos indican que velocidades de transmisión se van manejar:

- 12 señales x 12 bits/señal = 144 bits
- 144 bits + 30 % CRC etc = 200 bits aprox.
- 200 bits / Tmuestreo = 200 bits/m s = 200 Kbps = 25 Kbytes/s FDX

Dado el tipo de comunicación hay que tener ciertos aspectos en cuenta:

- Se trabaja con señales digitales que van a actuar sobre servomecanismos o van a ser resultados de medidas de sensores, por lo cual es necesaria una transparencia total del canal, a diferencia de las comunicaciones de voz, donde la señal si puede sufrir ciertas degradaciones que no afectan a la comunicación dado el tipo de está.

- Debe estudiarse si es admisible la perdida de alguna muestra o si hay señales menos críticas que puedan protegerse menos.

- Pueden usarse mecanismos de compresión pero deben ser sin perdida de información por el motivo anteriormente expuesto.

Lado PC en tierra

Las especificaciones son las mismas en cuanto a señales y velocidad necesaria, lo único que varia es que el interfaz es ahora con el PC en lugar del con el FCS, esta interfaz puede ser USB, puertos serie o paralelo, Ethernet, etc... siempre que se cumpla con las especificaciones deseables. Es recomendable la interfaz USB por su fuerte incorporación en los equipos PC y mejores prestaciones de velocidad.

Debido al que el FCS en sus fases previas de diseño se va a ejecutar sobre el PC y debe comunicarse con las distintas unidades a bordo vía el bus de campo se procede considerar un bus de campo en lado PC en tierra por lo que habrá un interfaz PC – Bus de campo y otro interfaz Bus de campo – Unidad de comunicaciones. Es decir se va tener un único bus de campo divido en dos segmentos que se comunican a través de las tarjetas de comunicaciones. Las actuaciones que genere el FCS destinadas a los servomecanismos serán idénticas independientemente de que el servomecanismo en cuestión este en el segmento del bus de campo en tierra o a bordo del helicóptero. Análogamente las medidas tomadas por los sensores llegarán al FCS este ubicado en un PC o en una tarjeta a bordo. Esto permite también depurar el funcionamiento de las diferentes tarjetas en tierra sin la necesidad de tener construida la tarjeta FCS, pues se simula el funcionamiento de ésta en un PC conectado a un segmento del bus CAN.

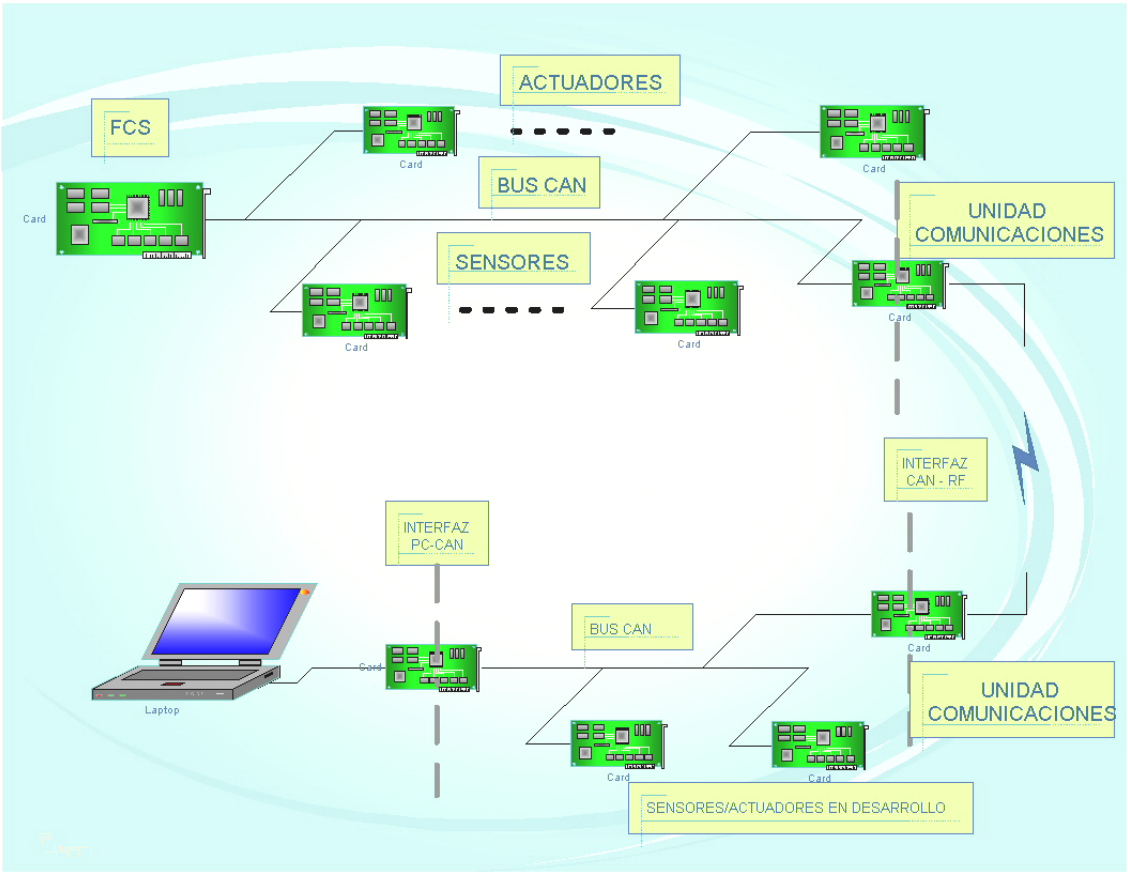


Figura 1.2 –3: Esquema Bus de campo e interfaces.

El **objetivo de este proyecto** es el **diseño e implementación** de las dos unidades de comunicaciones que actúan como interfaz entre el bus CAN y el medio radioeléctrico (**interfaz CAN – RF**). También es objetivo de este proyecto el **diseño** y **configuración** del **bus CAN.**

1.3 Estado Actual de la tecnología. Productos similares en el mercado.

Un producto existente en el mercado diseñado para cumplir la misma funcionalidad es el CANRF. CANRF es un controlador de área local de red inalámbrico de la compañía canadiense Automation Artisans que permite una tasa de transferencia de 20 Kbps (si trabaja en la banda de 868 MHz) o 10 Kbps (para la banda de 433 MHz). Estos anchos de banda son insuficientes para cumplir las especificaciones.

Figura 1.3: CANRF de Automation Artisans

1.4 Descripción de contenidos de esta memoria

En este punto se va a describir como se han estructurados los contenidos de esta memoria. En este primer cápitulo basicamente se ha descrito el contexto en el que se ubica este proyecto y su objetivo concreto dentro de este contexto.

En el capítulo 2 se procede a estudiar las posibles familias de tecnologías que se pueden aplicar para la consecución de los objetivos de este proyecto. Se hace un análisis de las posibles soluciones para resolver las necesidades de radiocomunicación, valorando su viabilidad y un análisis de los distintos sistemas de bus de campo. También se justifica la elección de la familia microcontroladores de Atmel AVR.

La configuración de la sección CAN se estudia en el capitulo 3, donde se describe el estándar CAN, la configuración de los dispositivos de este sistema de bus y como se ha estructrurado el sistema de bus para esta aplicación concreta.

En el capítulo 4 se describe el estándar Bluetooth que emplean los módulos de radiofrecuencia empleados para resolver las necesidades de radiocomunicación de este proyecto. También se describe como se han configurado estos módulos.

Las características del microcontrolador ATmega128 empleado en este proyecto son descritas en el capítulo 5, donde también se procede a describir los dispositivos internos de este microcontrolador empleados.

El diseño e implementación del hardware se detalla en el capítulo 6 donde se incluyen los esquemáticos de las tarjetas.

En el capítulo 7 se describe el software que se ejecuta en el microcontrolador. Se explica como se realizan las rutinas mas importantes y se adjunta el código completo.

Finalmente se hace una valoración de las prestaciones obtenidas y de posibles desarrollos futuros.

2. Análisis de las posibles soluciones en cada sección.

2.1 Sección de Radio Frecuencia

En esta sección del documento se analizan algunas de las diferentes soluciones de Radio Frecuencia (RF) que existen en el mercado, comentando sus características, y sus ventajas e inconvenientes a la hora de su empleo en el proyecto en cuestión.

Las diferentes familias de soluciones son las siguientes:

- Diseñar y construir toda una **tarjeta RF a partir de un chip RF** de los existentes en el mercado. Requiere el diseño de toda la circuitería anexa al chip y un diseño de pistas PCB muy cuidado por ser una tarjeta que trabaja a alta frecuencia. Realizar todo este diseño es muy complejo, de hecho existen empresas que se dedican a este menester, fabricando módulos de radiofrecuencia homologados que cumplan los requisitos de radiación de la legislación de radiofrecuencia.

- Diseñar y construir una tarjeta RF a partir de módulos **ISM en las bandas 433 y 868 MHz**. Son módulos que realizan una modulación sencilla normalmente FSK a partir de la secuencia de bits que se les entrega.. Tienen un ancho de banda que permiten velocidades binarias de unos 32 a 64 Kbps. Trabajan en semiduplex con un consecuente tiempo de conmutación que es bastante elevado.

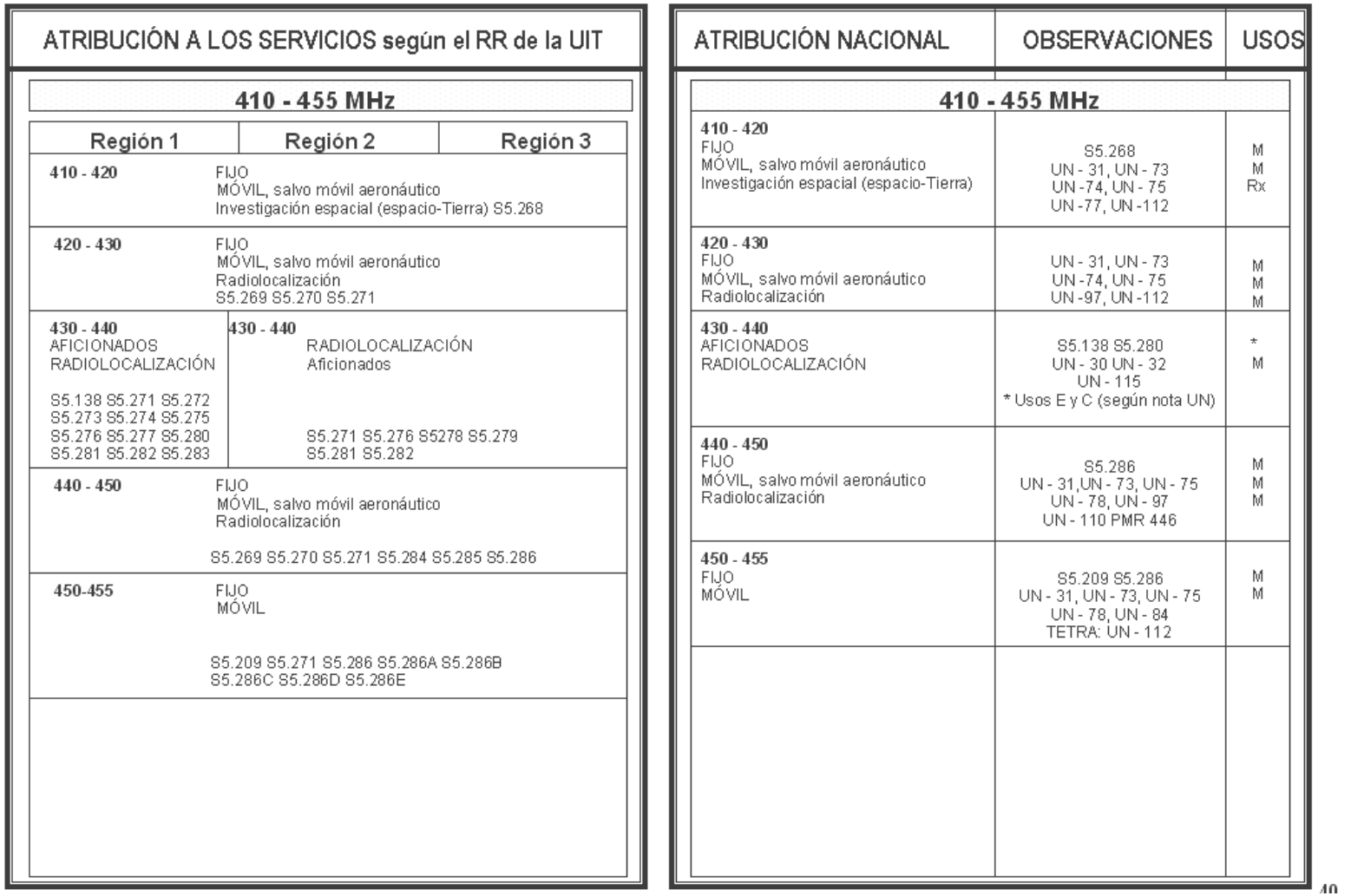
- Trabajar sobre tarjetas **ISM 2.4 Ghz,** que tienen un mayor ancho de banda y permiten un régimen binario mayor. Trabajan con alguna de las técnicas de espectro expandido FHSS o DSSS orientadas a las tecnologías inalámbricas que requieren un gran ancho de banda. En algunos de sus modos de trabajo presentan al usuario un entorno Full – Duplex (FDX).

2.1.1 ISM 433 MHz

2.1.1.1 Legislación Radiofrecuencia banda 433 MHz.

La banda de 433 MHz es una banda de uso común sin necesidad de licencia que tradicionalmente se ha empleado para distintas aplicaciones inalámbricas del tipo de telemandos. Es una de las bandas ISM (ICM aplicaciones industriales- científicas - médicas). La principal restricción es la limitación de potencia.

Para obtener la legislación detallada asociada a esta banda se acude al Cuadro Nacional de Atribución de Frecuencias (CNAF).



ATRIBUCIÓN A LOS SERVICIOS según el RR de la UIT		
410 - 455 MHz		
Región 1	**Región 2**	**Región 3**
410 - 420 FIJO MÓVIL, salvo móvil aeronáutico Investigación espacial (espacio-Tierra) S5.268		
420 - 430 FIJO MÓVIL, salvo móvil aeronáutico Radiolocalización S5.269 S5.270 S5.271		
430 - 440 AFICIONADOS RADIOLOCALIZACIÓN S5.138 S5.271 S5.272 S5.273 S5.274 S5.275 S5.276 S5.277 S5.280 S5.281 S5.282 S5.283	**430 - 440** RADIOLOCALIZACIÓN Aficionados S5.271 S5.276 S5278 S5.279 S5.281 S5.282	
440 - 450 FIJO MÓVIL, salvo móvil aeronáutico Radiolocalización S5.269 S5.270 S5.271 S5.284 S5.285 S5.286		
450-455 FIJO MÓVIL S5.209 S5.271 S5.286 S5.286A S5.286B S5.286C S5.286D S5.286E		

ATRIBUCIÓN NACIONAL	OBSERVACIONES	USOS
410 - 455 MHz		
410 - 420 FIJO MÓVIL, salvo móvil aeronáutico Investigación espacial (espacio-Tierra)	S5.268 UN - 31, UN - 73 UN -74, UN - 75 UN -77, UN -112	M M Rx
420 - 430 FIJO MÓVIL, salvo móvil aeronáutico Radiolocalización	UN - 31, UN - 73 UN -74, UN - 75 UN -97, UN -112	M M M
430 - 440 AFICIONADOS RADIOLOCALIZACIÓN	S5.138 S5.280 UN - 30 UN - 32 UN - 115 * Usos E y C (según nota UN)	* M
440 - 450 FIJO MÓVIL, salvo móvil aeronáutico Radiolocalización	S5.286 UN - 31,UN - 73, UN - 75 UN - 78, UN - 97 UN - 110 PMR 446	M M M
450 - 455 FIJO MÓVIL	S5.209 S5.286 UN - 31, UN - 73, UN - 75 UN - 78, UN - 84 TETRA: UN - 112	M M

Figura 2.1.1.1 – 1: CNAF Banda 433 MHz.

En el intervalo de 430 a 440 MHz existe una banda destinada a radioaficionados y radiolocalización. Los usos que se pueden hacer de esta banda son C (Común) , E (Especial) y M (Mixto). El uso mixto es una designación que comprende otros dos usos el P (Privativo) y el R (Reservado para uso por el Estado para la gestión a través de Administraciones Públicas o por concesión). En las notas de utilización nacional (UN) se concretan los detalles sobre el uso de la banda.

UN - 30

Frecuencias designadas para sistemas de radiocomunicaciones de telemando, telemedida y telealarmas con potencia máxima de salida igual o inferior a 500 mW y potencia radiada aparente (p.r.a.) igual o inferior a 100 mW, así como otros usos generales de baja potencia hasta 10 mW de potencia de equipo o p.r.a., dentro de la banda ICM de 433,050 a 434,790 MHz.

Con canalización de 25 kHz:

433,075 MHz	433,100 MHz	433,125 MHz	433,150 MHz
433,175 MHz	433,200 MHz	433,225 MHz	433,250 MHz
433,275 MHz	433,300 MHz	433,325 MHz	433,350 MHz

Para las mismas aplicaciones, con cualquier tipo de modulación y potencia de equipo o p.r.a. igual o inferior a 10 mW, también podrá ser utilizada la frecuencia central de la banda ICM, es decir 433,920 MHz.

En estas circunstancias, la utilización de las frecuencias indicadas se considera de uso común.

UN - 32

Banda de frecuencias 433,050 MHz a 434,790 MHz.

Esta banda está designada para aplicaciones industriales, científicas y médicas (ICM).

La utilización de estas frecuencias para dichas aplicaciones se considera de uso común.

Los servicios de radiocomunicaciones que funcionen en ella, deben aceptar la interferencia perjudicial resultante de estas aplicaciones.

Los equipos "ICM" que funcionen en esta banda de frecuencias, deberán cumplir los límites de radiaciones establecidos en el Real Decreto 444/1994 de 11 de marzo sobre requisitos de protección relativos a compatibilidad electromagnética

UN - 115

Bandas de frecuencias permitidas para aplicaciones no específicas de baja potencia (dispositivos de corto alcance).

Sin perjuicio de otras utilizaciones expresamente reconocidas en el CNAF, se destinan a estas aplicaciones las siguientes bandas de frecuencia:

13,553-13,567 MHz	2400-2483,5 MHz
40,660-40,700 MHz	24,0-24,25 GHz
433,050-434,790 MHz	61,0-61,5 GHz

Dichos dispositivos han de cumplir las especificaciones técnicas del correspondiente estándar EN 300 220, EN 300 330 o I-ETS 300 440, y ajustarse a las condiciones de uso y limites de potencia indicados en la Recomendación CEPT/ERC 70-03 Anexo 1 en los respectivos apartados.

Esta utilización se considera de uso común.

El Anexo 1 de la Recomendación CEPT/ERC 70-03 referido en la UN-115 es el que se adjunta en la figura 2.1.1.1 – 2 de la página siguiente.

En este anexo se indican la máxima potencia radiada, la duración del uso que se puede hacer y si es necesaria una determinada canalización. En base a estas características pueden hacerse distintos usos de esta banda.

Si no se emplea canalización se permiten 2 opciones:

- Potencia Radiada Efectiva 10 mW, con una carga de uso del espectro menor del 10%.
- Potencia Radiada Efectiva 1 mW y –13 dBm/KHz, con una carga de uso del espectro menor de hasta el 100%.

Si se trabaja con canalización esta será hasta 25 KHz dentro del rango de 434,040 a 434,790 MHz. Se permite una Potencia Radiada Efectiva 10 mW y una carga de uso hasta el 100%.

Se deben evitar señales de audio y voz en la banda completa (433,05 a 434,79 MHz).

Annex 1 Non-specific Short Range Devices

Scope of Annex

This annex covers frequency bands and regulatory as well as informative parameters recommended primarily for Telemetry, Telecommand, Alarms, Data in general and other similar applications. Video applications should only be used above 2.4 GHz. Audio and voice signals should be avoided in the band 433.050-434.790 MHz.

Regulatory parameters related to Annex 1

	Frequency Band	Power / Magnetic field	Duty cycle	Channel spacing	ERC Decision	Notes
a	6765 - 6795 kHz	42 dBuA/m at 10 m	No Restriction	No spacing	ERC DEC (01)01	
b	13.553 - 13.567 MHz	42 dBuA/m at 10 m	No Restriction	No spacing	ERC DEC (01)01	
c	26.957 - 27.283 MHz	42 dBuA/m at 10 m 10 mW e.r.p.	No Restriction	No spacing	ERC DEC (01)02	
d	40.660 - 40.700 MHz	10 mW e.r.p.	No Restriction	No spacing	ERC DEC (01)03	
e	433.050 - 434.790 MHz	10 mW e.r.p.	< 10 %	No spacing		Audio and voice signals should be avoided in the band 433.05-434.79 MHz
e1	433.050 - 434.790 MHz	1 mW e.r.p. -13 dBm/10 kHz	up to 100%	No spacing		Power density limited to -13 dBm/10 kHz for wideband channels Audio and voice signals should be avoided in the band 433.05-434.79 MHz
e2	434.040-434.790 MHz	10 mW e.r.p.	up to 100%	Up to 25 kHz		Audio and voice signals should be avoided in the band 433.05-434.79 MHz
f	868.000 - 868.600 MHz	25 mW e.r.p.	< 1.0 %	No spacing	ERC DEC (01)04	
g	868.700 - 869.200 MHz	25 mW e.r.p.	< 0.1 %	No spacing	ERC DEC (01)04	
h	869.300 - 869.400 MHz	10 mW e.r.p.	No Restriction	25 kHz		An appropriate access protocol should be used for example EN 301 391
i	869.400 - 869.650 MHz	500 mW e.r.p.	< 10 %	25 kHz	ERC DEC (01)04	The whole band may also be used as 1 channel for high speed data transmission
k	869.700 - 870.000 MHz	5 mW e.r.p.	up to 100%	No spacing	ERC DEC (01)04	
l	2400 - 2483.5 MHz	10 mW e.i.r.p.	No Restriction	No spacing	ERC DEC (01)05	
m	5725 - 5875 MHz	25 mW e.i.r.p.	No Restriction	No spacing	ERC DEC (01)06	
n	24.00 - 24.25 GHz	100 mW e.i.r.p.	No Restriction	No spacing		
o	61.0 - 61.5 GHz	100 mW e.i.r.p.	No Restriction	No spacing		
p	122 - 123 GHz	100 mW e.i.r.p.	No Restriction	No spacing		
q	244 - 246 GHz	100 mW e.i.r.p.	No Restriction	No spacing		
r	138.2 - 138.45 MHz	10 mW e.r.p.	< 1.0 %	No spacing		

Additional Information

Harmonised Standards

EN 300 220	subbands c) - k)
EN 300 330	subbands a) and b)
EN 300 440	subbands l) - m) and n)

Frequency issues

The bands in Annex 1 a - b - c - d - e - l - m - n - o - p and q are also designated for industrial, scientific and medical (ISM) applications as defined in ITU Radio Regulations.

To avoid interference between CT2 and SRD applications it is recommended that SRDs below 868.5 MHz should avoid using a dedicated frequency channel and instead use a technology that allows automatic channel selection of a free channel within the band.

The adjacent frequency band above 870 MHz has been designated for use by the high powered TETRA and other digital land mobile PMR/PAMR systems. Manufacturers should take this into account in the design of equipment and choice of power levels.

Technical parameters also referred to in the harmonised standard

No information

Figura 2.1.1.1 – 2: Anexo 1 Recomendación CEPT/ERC 70-03.

2.1.1.2 Dispositivos existentes en el mercado.

En este punto se van a presentar módulos de radiofrecuencia que existen en el mercado perteneciente a esta banda.

AUREL XTR-434

El módulo Aurel XTR-434 es un módulo híbrido diseñado para trabajar a una frecuencia central de 433,92 MHz. Esta fabricado por la compañía italiana Aurel muy conocida por sus diferentes módulos híbridos en revistas de electrónica. Existen 2 modelos en el mercado de este dispositivo uno que admite una tasa de datos de hasta 50 Kbps y otro que admite hasta 100Kbps (después se verá como realmente trabajaría a 50 Kbps por necesidad de duplicación de la información a transmitir). Cumplen las normativas europeas EN 300 220 y EN 301 489.

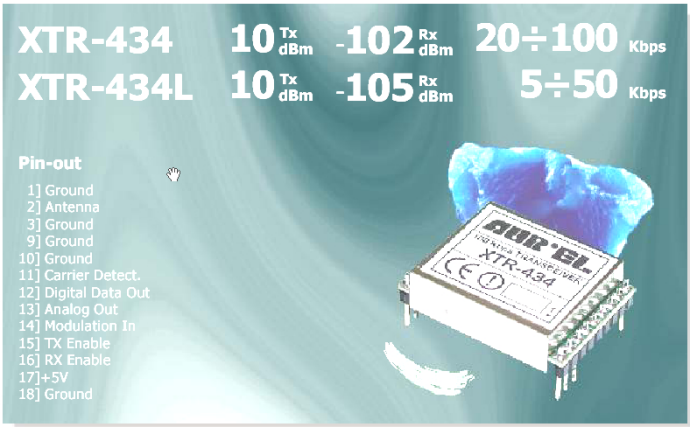


Figura 2.1.1.2 –1: Módulo Aurel XTR – 434

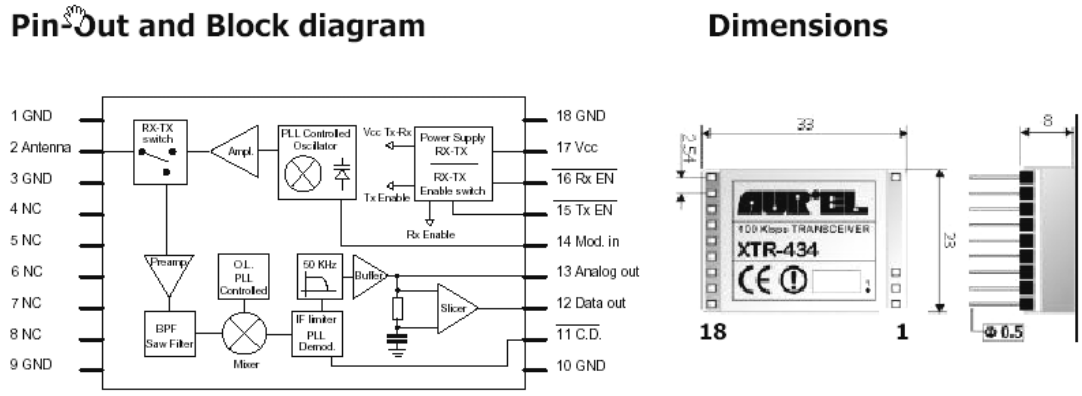


Figura 2.1.1.2 –2: Aurel XTR – 434

AUREL

XTR-434xxx
Instruction Manual

Technical features

Characteristics	Min	Typical	Max	Unity	Remarks
Voltage supply	4,5	5	5,5	Vdc	
Supply current [TX ON]	24	28	32	mA	
Supply current [RX ON]	10	11	12	mA	
Supply current [TX/RX OFF]			100	nA	
RX SECTION					
Reception frequency		433.92		MHz	
RF sensitivity [XTR-434]		-100	-102	dBm	See note 1
RF sensitivity [XTR-434L]		-103	-105	dBm	See note 1
IF passband		150		KHz	
Interferences rejection [at 5% band limits]		-80		dB	
RF spurious emissions in antenna		absent			See note 2
Output square wave [XTR-434]	10		50	KHz	
Output square wave [XTR-434L]	2.5		25	KHz	
Output low logic level		0,1		V	See note 4
Output high logic level		3.5		V	See note 4
Carrier Detect [CD] threshold	-96	-98		dBm	
TX SECTION					
Transmission frequency		433.92		MHz	
Modulation passband [XTR-434]	10		50	KHz	
Modulation passband [XTR-434L]	2.5		25	KHz	
FM deviation		±25		KHz	
TX output power			10	dBm	
Antenna impedance		50		Ω	
RX switch-on time		1		ms	
TX switch-on time		1		ms	
Working temperature	-20		+80	°C	
Working temperature [ETS 300 220]	-20		+55	°C	
Dimensions	33 x 23 x 8 mm				

Note1
[XTR-434]: test as Fig. 3 , RF IN −100 dBm, FM Deviation ± 25KHz and Modulation frequency 40 KHz.
[XTR-434L]: test as Fig. 3 , RF IN −103 dBm, FM Deviation ± 25KHz and Modulation frequency 20 KHz.
Note2: The R.F. emission measure has been obtained by direct connection of the spectrum analyser to XTR module pin 2.
Note3: Switch-on time is the time required by the device to acquire the declared characteristics, from the very moment the enable signal is applied.
Note4: Values obtained with a 10KΩ load applied.

Figura 2.1.1.2 –3: Características Técnicas XTR – 434

Las características más importantes son la frecuencia central, el ancho de anda, la velocidad física y la potencia de transmisión. La frecuencia central y al ancho de banda ubican en que zona del espectro se va trabajar, en este caso en la banda de 433 MHz. La velocidad física debe ser suficiente para soportar el caudal de datos que debe circular por el interfaz de radiofrecuencia. La potencia de transmisión suele medirse en términos de PIRE (Potencia Isótropa Radiada Equivalente) o PRA (Potencia Radiada Aparente) y debe ser inferior a lo que se establezca según las notas de utilización en la banda de frecuencia que se trabaje. A partir de estas características se puede comprobar que este módulo está dentro de lo permitido en la UN-30 en donde se hace referencia a aplicaciones ICM con frecuencia central de 433,92 MHz.

Este módulo emplea una modulación FM que se puede emplear para aplicaciones analógicas o digitales. Otra característica importante es que es

semiduplex, se transmite en un sentido y luego en el otro pero nunca simultáneamente. Asociado al hecho de trabajar en modo semiduplex se tiene un tiempo de conmutación que se trata del tiempo necesario que necesita el dispositivo para pasar de transmisión a recepción y viceversa. La gestión del nivel de enlace la debe realizar el microcontrolador *host*. Éste se debe encargar de formar una trama de protocolo a nivel de enlace. En esta trama se deben proveer mecanismos para detectar errores y corregirlos (o simplemente descartarlos), gestión del turno de acceso al canal, señalización de principio / fin de trama (caracteres especiales, longitud de trama), etc.

Un detalle de este módulo es que aunque se anuncia como capaz de trabajar a 100 Kbps luego en la hoja de características se advierte de la necesidad de compensar unos y ceros para que no se produzca la perdida de sincronización en la sección que se encarga de extraer el reloj (sobre todo a velocidades altas), los métodos sugeridos implican duplicar la información balanceando sobre el bit o sobre el byte. Luego realmente la velocidad de transmisión se divide por dos.

En cuanto a la interconexión física al sistema es necesario:

- Proveer la alimentación.
- Gestión de las señales de habilitación de recepción y de transmisión.
- Enviar / Recibir los bits de manera serie, si fuera viable enviarlos en formato serie se aprovecharía para configurar una solución de ‘reemplazamiento de cable’.
- Conexión del módulo a la antena. La antena puede ser una antena comercial o alguna realización impresa en PCB.

Radiometrix BIM2

Otro módulo de características muy parecidas es el BIM2 de Radiometrix.

Figura 2.1.1.2 – 4 a: BIM2 de Radiometrix.

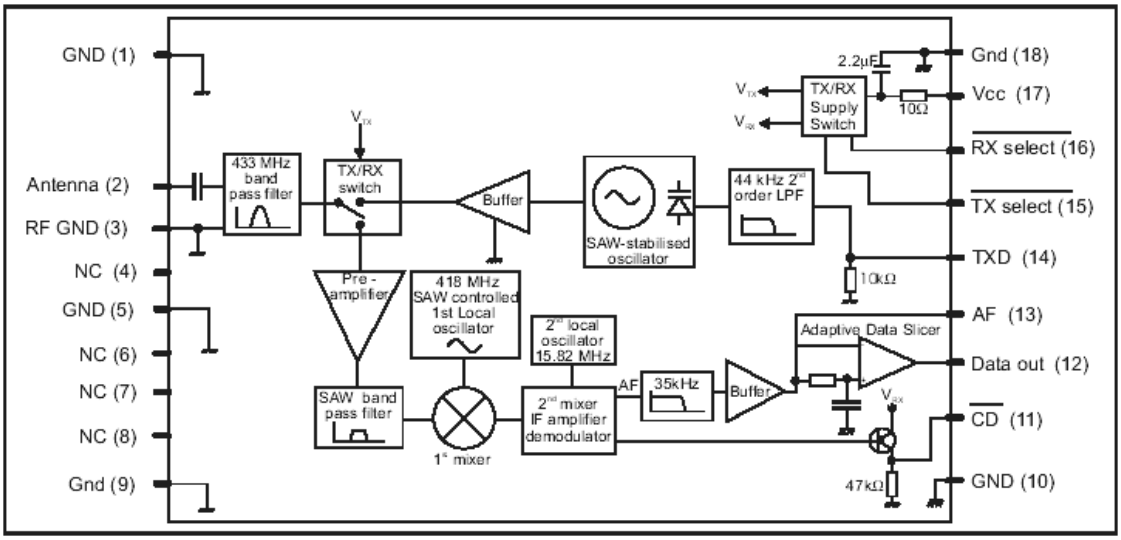


Figura 2.1.1.2 – 4 b: BIM2 de Radiometrix. bloques internos.

Ambos módulos (XTR – 434 y BIM2) funcionalmente presentan las mismas características, si bien existe una versión del BIM2 que alcanza una velocidad física de hasta 160 Kbps, aunque también se advierte de la necesidad de compensación de unos y ceros para el funcionamiento a elevadas velocidades. Este módulo presenta varias versiones que difieren en la tensión de alimentación (3.3V o 5V) y en la velocidad física del módulo (64Kbps, opción S,160 Kbps). Las características que ofrece el fabricante del módulo BIM2 son las de las figuras siguientes.

Electrical Performance	pin.	min.	typ.	max.	units.	notes.
DC Levels						
supply voltage, Vcc (std. version)	17	4.0	5	5.5	V	
supply voltage, Vcc (3V version)	17	3.0	3.3	4.0	V	
TX supply current, Vcc (std)	17	10	14	16	mA	5V supply
TX supply current, Vcc (3.3V)	17	6	8	10	mA	3.3V supply
RX supply current, Vcc (std)	17	12	18	21	mA	5V supply
RX supply current, Vcc (3.3V)	17	10	14	17	mA	3.3V supply
supply ripple allowed	17	-	-	20	mV_{pk-pk}	below1MHz
AF output DC level	13	1.0	1.25	1.5	V	
load capacitance on AF / Data	12,13	-	-	100	pF	
CD output load resistance	11	220	-	-	kW	
Interface levels						
data output high, 100mA source	12	-	Vcc-0.6	-	V	RXD high
data output low, 100mA sink	12	-	0.4	-	V	RXD low
TX & RX select, high (deselect)	15, 16	Vcc-0.5		Vcc	V	
low (select)	15, 16	0		0.5	V	
Internal select pull-ups	15, 16	-	10	-	Kw	
TXD, high	14	Vcc-0.5		Vcc	v	
low	14	0		0.5	v	

Figura 2.1.1.2 – 5: Características eléctricas del BIM2 .

RF Parameters	pin.	min.	typ.	max.	units.	notes.
Antenna pin impedance	2	-	50	-	w	TX or RX
RF centre frequency	-	-	433.92	-	MHz	
Transmitter						
RF power output, Vcc std	2	+7	+11	+12	dBm	5v
RF power output, Vcc 3.3V	2	+3	+6	+8	dBm	3.3V
Initial frequency accuracy	-	-50	0	+50	kHz	
Overall frequency accuracy	-	-100	0	+100	kHz	
FM deviation	-	20	30	40	kHz	
Modulation bandwidth	-	DC	-	32	kHz	
Modulation bandwidth	-	DC	-	80	KHz	160kbps
Modulation distortion	-	-	-	15	%	
Receiver						
RF sensitivity, 10db S/N, 5V	2, 13	-95	-101		dBm	
RF sensitivity, 10db S/N, 3.3V	2, 13	-91	-96		dBm	
RF sensitivity, 10db S/N, 5V	2,13		-94		dBm	160kbps
RF sensitivity, 1ppm BER 5V	2, 12	-87	-93		dBm	
RF sensitivity, 1ppm BER 3.3V	2, 12	-82	-88		dBm	
RF sensitivity, 1ppm BER 5V	2,12		-90		dBm	160kbps
CD threshold, Vcc = 5V	2, 11	-98	-104		dBm	
CD threshold, Vcc =3V	2,11	-92	-98		dBm	
Cd threshold, Vcc = 5V	2,11		-96		dBm	160kbps
IF bandwidth	-	-	500		kHz	
CD bandwidth	2, 11	-	400		kHz	
Ultimate (S+N)/N, -70dBm I/P	13	-	>40		dB	
Ultimate (S+N)/N, -70dBm I/p	13	-	30		dB	160kbps
maximum operating RF I/P	2	-	+10		dBm	
AF output level	13	-	400		mV	peak to peak
Initial frequency accuracy	-	-50	0	+50	kHz	CD centre

Figura 2.1.1.2 – 6: Características RF del BIM2 .
Potencia, sensibilidad y ancho de banda.

EMC Parameters	pin.	min.	typ.	max.	units.	notes.
Rejections: rejection figures are relative to a 15dB (S+N)/N wanted signal						
Co-channel rejection	2	-		-	dB	
Image rejection (f_{RF}-$2f_{IF}$)	2	-	64	-	dB	402.0MHz
Out of band rejection	2	-	>70	-	dB	DC to 2GHz
AM rejection	2	-	>30	-	dB	
Out of band blocking level	2	-	>-15	-	dBm	
Out of band IP_3	2	-	+1	-	dBm	
Radiations						
RX LO leakage, conducted	2	-	-60	-57	dBm	
RX LO leakage, radiated	-	-	-70	-	dBm	
TX 2nd harmonic	2	-	-42	-36	dBm	
TX harmonics >1GHz	2	-	-40	-30	dBm	
TX spectral bandwidth @-40dBc	2	-	-	250	kHz	worst case

Figura 2.1.1.2 – 7: Parámetros de compatibilidad electromagnética.

Baseband Transfer Performance	pin.	min.	typ.	max.	units.	notes.
TX RX						
Linear baseband BW @-3dB	13	0,08	-	34	kHz	TXD to AF
Linear baseband BW @-3db	13	0,08		80	kHz	TXD to AF 160kbps
Balance code bit rate	12	-	64	-	kbit/s	
Time between code transitions	14	15.6	-	1000	µs	
Time between code transitions	14	15.6	-	120	µs	S version
Time between code transitions	14	6.25	-	100	µs	160kbps
preamble duration	14	3	-	-	ms	01010101 pattern
preamble duration	14	1	-	-	ms	S version
link delay	14, 12	-	15	-	µs	TXD to RXD
Dynamic Timing						
Power up with signal present						
Power up to valid CD, t_{PU-CD}	11	-	0.7	1	ms	
Power up to stable AF, t_{PU-AF}	13	-	0.5	1	ms	
Power up to stable data, $t_{PU-data}$	12	-	3	5	ms	
Power up to stable data, $t_{PU-data}$	12	-	-	1	ms	1, S version
Power up to stable data, $t_{PU-data}$	12	-	-	0.8	ms	160kbps
Signal applied with supply on						
Signal to valid CD, t_{sig-CD}	11	-	0,25	0,5	ms	
Signal to stable data, $t_{sig-data}$	12	-	3	4	ms	
Signal to stable data, $t_{sig-data}$	12	-	-	1	ms	1, S version
Signal to stable data, $t_{sig-data}$	12	-	-	0.5	ms	160kbps
TX power up to full RF	2	-	100	-	µs	

Note 1: From 45% to 55% duty cycle

Figura 2.1.1.2 – 8: Características temporales.

2.1.1.3 Análisis de viabilidad de los dispositivos ISM en la banda de 433 MHz.

En este apartado se van a analizar las características de los módulos para estudiar si cumplen las especificaciones básicas.

Cobertura y diversidad.

La gran pregunta y la de más difícil respuesta cuando se trabaja con dispositivos de radiofrecuencia es el alcance. Los transmisores son diseñados para generar la máxima potencia de radiación que les permita la regulación (10 mW = 10dBm para 433MHz) o el coste del diseño. Los receptores a su vez son diseñados para detectar

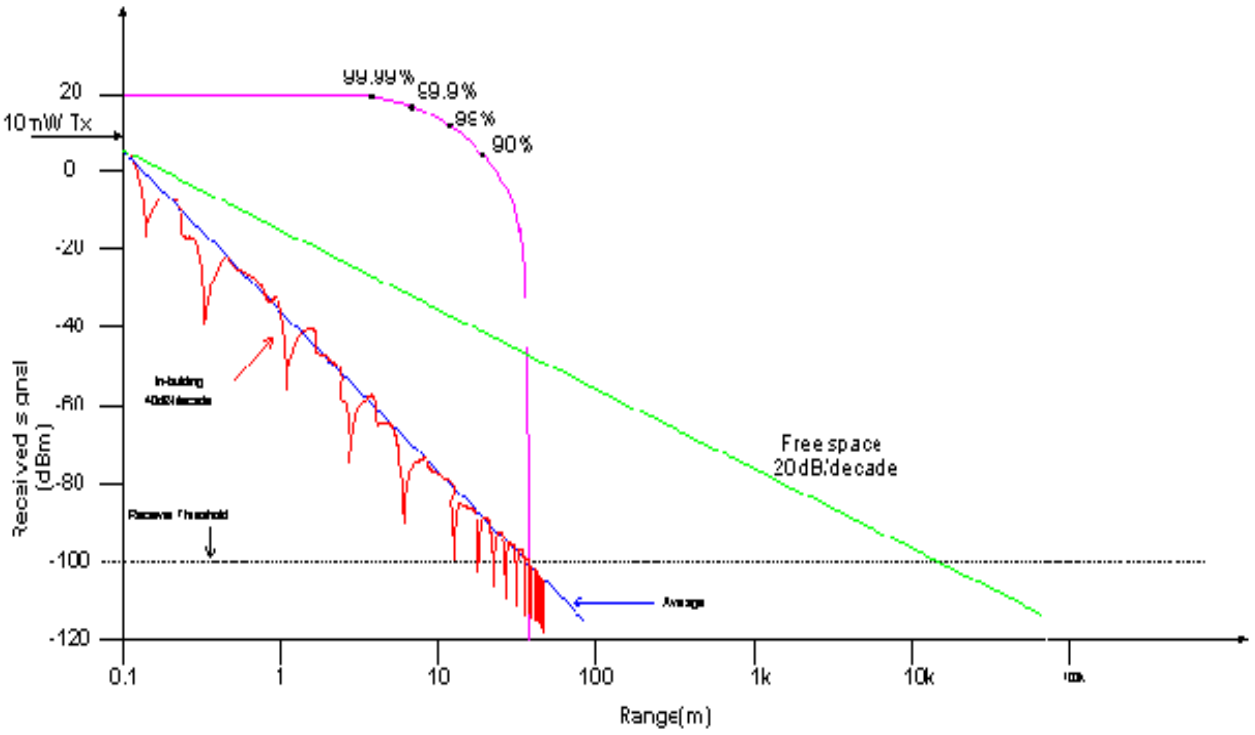
señales muy débiles, el valor umbral de detección se conoce como sensibilidad (valor típico –100 dBm). La diferencia entre la potencia radiada y la sensibilidad es lo que se conoce como la máxima perdida de trayecto admisible. En condiciones de espacio libre la atenuación del trayecto sólo depende de la distancia y fácilmente se podría calcular el alcance máximo. Asumiendo antenas ideales de ganancia 0dB la expresión para calcular el alcance viene dada por:

$$\text{Range} = \frac{\lambda}{4\pi}\sqrt{\frac{P_{tx}}{P_{rx}}}$$

$$= \frac{23.87 \times 10^{\frac{L}{20}}}{f}$$

Where R = range in meters
f = frequency in MHz
L = path loss in dB

Pero realmente no se está trabajando en esas condiciones ideales, existen perdidas extras. El simple hecho de estar sobre el suelo hace que se produzcan interferencias por reflexiones en el terreno, la presencia de obstáculos como edificios cercanos produce también interferencias extras. Estas interferencias se denominan interferencias multitrayecto y se pueden modelar como perdidas a añadir a la perdida básica de propagación. En el caso de interior de edificios esos efectos se aprecian todavía más.



Range curve for a 433MHz 10mW TX (unity gain antenna)

Figura 2.1.1.3 - 1: Alcance.

En la figura 2.1.1.3 – 1 se representa la señal recibida frente a la distancia, el alcance se determina cuando la señal recibida cae por debajo del umbral recepción. Se observa como considerando exclusivamente la perdida básica en un modelo de espacio libre el alcance supera los 10 km. Por el contrario cuando se considera un modelo mas real con obstáculos como el interior de un edificio se observa en la figura como se reduce el alcance notablemente y además existen puntos dentro del alcance donde existen bruscas caídas de la señal recibida (puntos ciegos o de visibilidad reducida). En estos puntos se produce una interferencia destructiva entre la propagación directa y las reflexiones en suelo y obstáculos. Se puede modelar el comportamiento por alguna distribución con una determinada media y que es susceptible de sufrir determinadas desviaciones, así en función de la distancia se tiene una probabilidad de que en algún momento se esté por debajo del umbral de recepción.

Un receptor ubicado aleatoriamente presenta:

- Una probabilidad del 10% de estar en un punto ciego con una atenuación superior a 10dB.
- Una probabilidad del 1% de estar en un punto ciego con una atenuación superior a 20dB.
- Una probabilidad del 0.1% de estar en un punto ciego con una atenuación superior a 30dB.

El efecto de la interferencia multitrayecto es de por sí bastante negativo pero puede empeorar pues el patrón de interferencia puede variar con el tiempo. También pueden existir otras fuentes de interferencias ajenas como otros sistemas en las cercanías, emisiones de equipos electrónicos (como el propio sistema donde se va a ubicar el módulo u ordenadores próximos)

Aunque en el caso de la aplicación concreta para el sistema de control de vuelo no se va a tener un entorno tan hostil como el interior de un edificio pero si se va a tener interferencia multitrayecto asociada a la reflexión en el suelo y a posibles obstáculos.

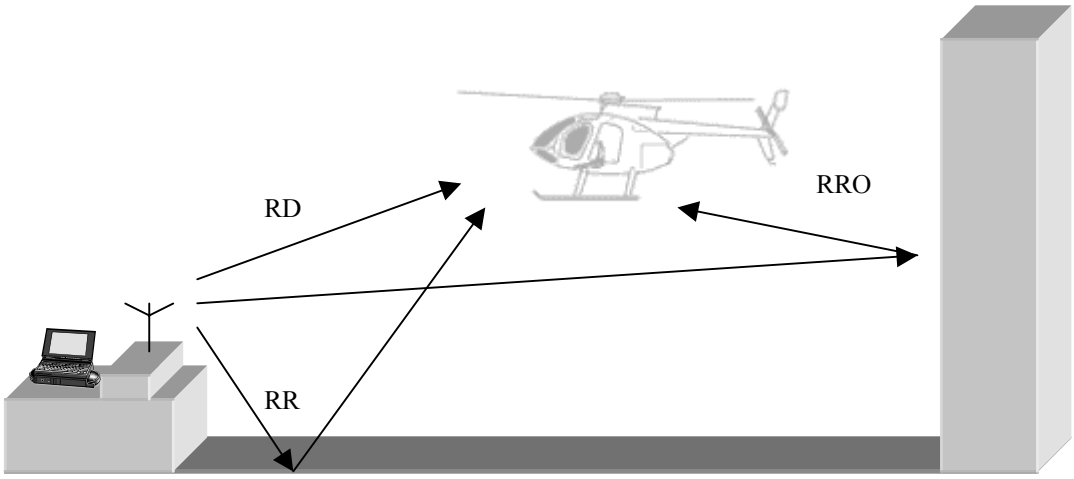


Figura 2.1.1.3 - 2: Rayo Directo, Rayo reflejado y Rayo Reflejado en Obstáculo.

Diversidad.

Para evitar los efectos de la interferencia multitrayecto se pueden emplear técnicas de diversidad.

Las técnicas de **diversidad espacial** consisten en duplicar las antenas o los equipos completos de recepción. Disponiendo las antenas los suficientemente alejadas para evitar el acoplamiento mutuo se puede conseguir que ambas no se hallen simultáneamente en el mismo nulo. El inconveniente de esta técnica es el coste de la duplicación de equipos y que no se adapta fácilmente a los módulos de 433 MHz vistos dada su constitución. Los módulos están diseñados para conectar una antena y habría que diseñar externamente al módulo una circuitería para disponer de dos antenas en recepción y de solo una en transmisión o contar con dos módulos actuando en recepción, y el microcontrolador *host* debería realizar un procesado de ambas recepciones y seleccionar un solo módulo para que transmita simultáneamente.

La diversidad temporal consiste en repetir el mensaje varias veces en periodos, aleatorios o de forma acordada con el dispositivo remoto. Esta técnica provee de una excelente protección frente a interferencias puntuales pero no mejora la fiabilidad si uno de los dispositivos de comunicación se halla permanentemente en un punto ciego. En el caso de este proyecto la ubicación del helicóptero varía en el tiempo con lo que lo diversidad temporal tiene el mismo efecto que la diversidad espacial. El inconveniente de esta técnica es que como mínimo se duplica el tiempo de transmisión para una determinada información con lo que al menos se está dividiendo por 2 la velocidad de transmisión.

La diversidad en frecuencia se basa en el hecho de que la posición de un punto ciego depende de la frecuencia, es decir, un punto que es ciego a una determinada frecuencia deja de serlo a una frecuencia lo suficientemente separada. En este caso el desplazamiento en frecuencia necesario es demasiado grande para el ancho de banda disponible. Además dada la constitución de los módulos no se puede variar la frecuencia central.

Compensación de símbolos.

Ambos módulos necesitan que en las tramas existan suficientes transiciones de 1 a 0 o viceversa para extraer correctamente los bits. De hecho en la especificaciones se da como dato que la media de alguno de los símbolos no puede superar el 70% para un intervalo de 2ms (en cualquier segmento de 2 ms el porcentaje de unos (o de ceros) no puede superar el 70%). Al principio de la trama es obligatorio un preámbulo formado por unos y ceros alternados, este tiempo de preámbulo se añade al tiempo de conmutación de transmisión a recepción y viceversa y tendrá efectos en la regulación de acceso al medio.

En la figura 2.1.3 – 3 a) se representa la tasa de error de paquete (trama) para unas tramas especiales muy susceptibles de error. Esta tasa de error se representa frente

a la relación de símbolos dentro de cada byte. Estos resultados provienen de un modelo anterior del BIM-2 denominado BIM-433-40 que permite una máxima velocidad de transmisión de 40 Kbps. En este experimento se comprueba como con el formato serie en el que no se compensan unos y ceros no puede alcanzar la máxima velocidad del módulo. Lo primero que se aprecia es que con señales cercanas al umbral de recepción la probabilidad de error es muy alta sobre todo a velocidades elevadas. Con una señal de recepción más aceptable se presenta una elevada probabilidad de error incluso a bajas velocidades cuando hay un desequilibrio en la proporción de unos y ceros. Por lo tanto se concluye que es necesario compensar la proporción de unos y ceros si se quiere trabajar a velocidades elevadas, pero por otro lado se tiene que para compensar unos y ceros se tiene que añadir redundancia. Se puede balancear sobre el bit enviado tras cada bit su complementario obteniéndose una codificación Manchester. También se puede balancear sobre el byte enviando el complemento a uno tras cada byte. Estos mecanismos implican duplicar la información a transmitir por lo que realmente nunca se podrá alcanzar la máxima velocidad de transmisión a efectos de la información útil. Por lo tanto la máxima velocidad útil que se podrá alcanzar será de 80 Kbps cuando el BIM2 trabaje a 160 Kbps de velocidad física.

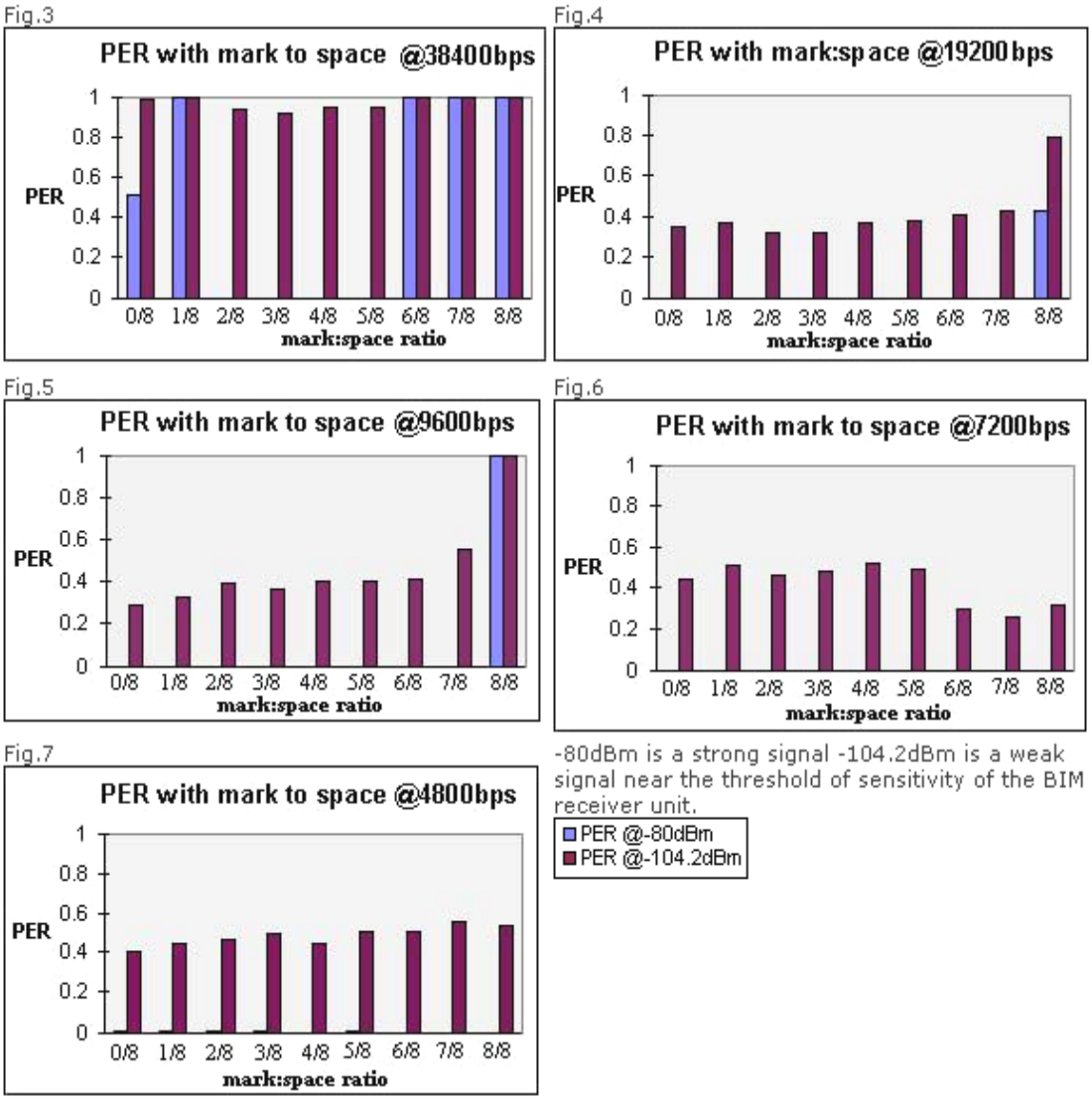


Figura 2.1.1.3 – 3 a): Compensación de símbolos.

El efecto negativo de una desproporción de símbolos es tan importante que si se introduce paridad se puede aumentar la probabilidad de error, produciendo una desensibilización del dispositivo.

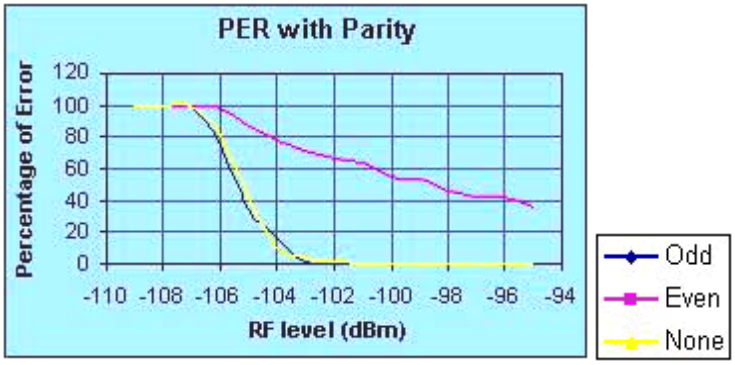


Figura 2.1.1.3 –3 b): Efecto de la paridad.

Acceso al medio y trama.

Dado que los dispositivos trabajan en modo semiduplex hay que regular el acceso al medio. Como se ha visto los dispositivos tienen un determinado tiempo de conmutación y por otro lado es obligatorio un preámbulo en el que se transmite una secuencia de unos y ceros alternos. Este preámbulo se puede englobar en el llamado tiempo de conmutación total.

Para el caso del XTR-434 el mínimo tiempo de conmutación total es de 2 ms (aunque sería aconsejable 3 o 4 ms). Con este tiempo de conmutación se puede asegurar de antemano que va ser imposible conseguir enviar datos cada milisegundo como se deseaba en las especificaciones inicialmente, por lo que si se trabaja con estos módulos se debe aumentar el tiempo de muestreo a un valor entorno a 10 ms.

La forma más sencilla para regular el acceso al medio es operar por turnos primero se produce una transmisión en el canal ascendente (Estación Base - Helicóptero) y posteriormente en el canal descendente (Helicóptero – Estación Base).

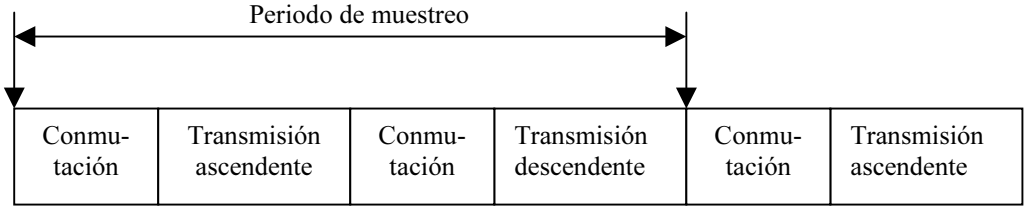


Figura 2.1.1.3 - 4: Acceso al medio.

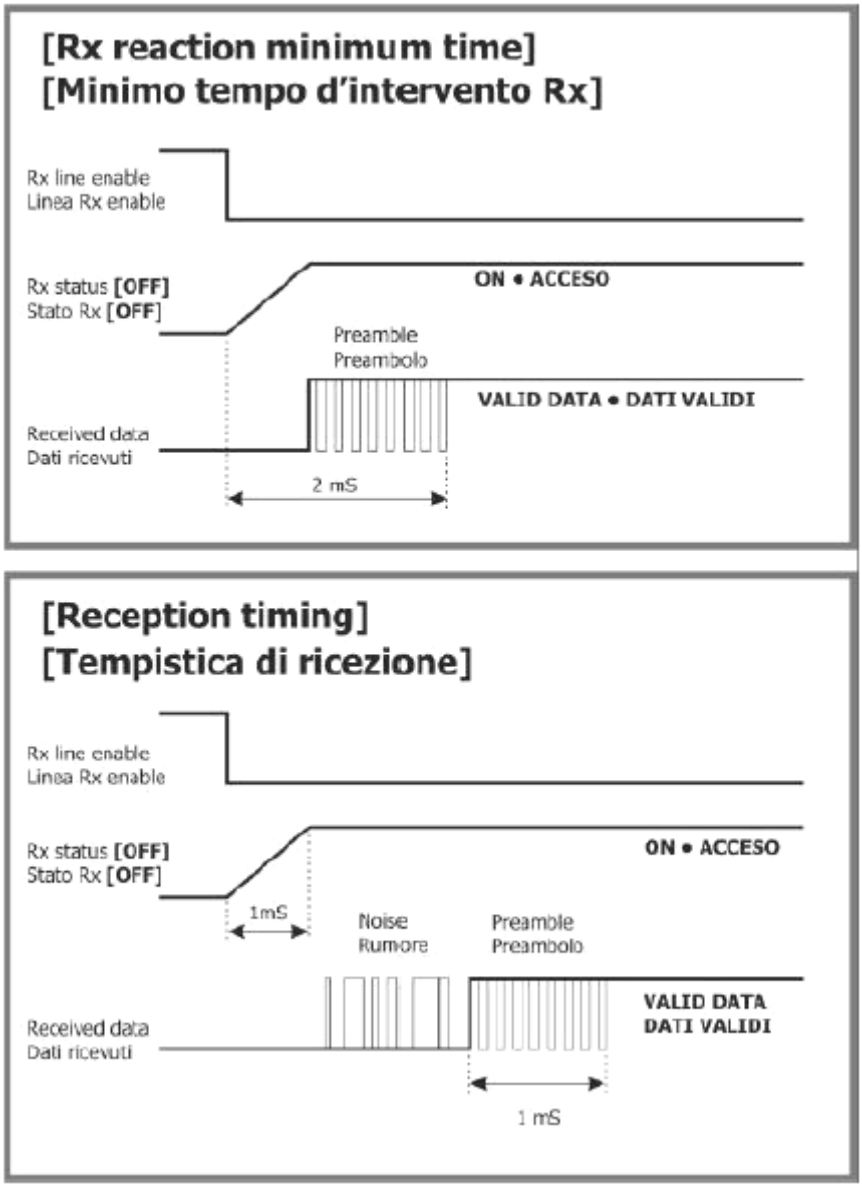


Figura 2.1.1.3 - 5: Tiempo de conmutación y preámbulo.

Para hacer los cálculos que puedan predecir el desempeño del sistema se va suponer que se forma una trama con una cabecera que tendría un byte para indicar el principio de trama, otro byte indicando la longitud de la trama y 2 bytes para el CRC. En cuanto a los datos en cada sentido de la comunicación se tienen 6 señales de 12 bits resultando 72 de bits de datos. Se puede optar por enviar siempre todos los datos o solo aquellos de interés (por ejemplo aquellos que hayan cambiado recientemente) en el segundo caso habría que añadir etiquetas que indicasen a que señal corresponde cada dato. Considerando que en cada transmisión se envían todos los datos resultan 104 bits, sin tener en cuenta la compensación de unos y ceros necesaria para reducir la probabilidad de error. Para estos cálculos iniciales se va a aproximar la longitud total de la trama compensada a 200 bits. En la tabla 2.1.1.3-1 se muestran resultados para distintas configuraciones de los módulos ISM 433 MHz.

Módulo		XTR-434 @20Kbps	XTR-434 @25Kbps	XTR-434 @50Kbps	BIM2-160 @64Kbps	XTR-434 @100Kbps	BIM2-160 @160Kbps
Tct (ms)=		2	2	2	3,5	2	3,5
bits totales =		200	200	200	200	200	200
Velocidad (Kbps)		20	25	50	64	100	160
Tdata (ms)		10	8	4	3,13	2	1,25
1 paquete info por trama	Tmuestreo=2Tdata+2Tct (ms)	24	20	12	13,3	8	9,5
	Tretraso=(2Tdata+2Tct)	24	20	12	13,3	8	9,5
2 paquete info por trama	Tmuestreo=(4Tdata+2Tct)/2 (ms)	22	18	10	9,75	6	6
	Tretraso=(4Tdata+2Tct)	44	36	20	19,5	12	12
3 paquete info por trama	Tmuestreo=(6Tdata+2Tct)/3 (ms)	21,3	17,3	9,33	8,58	5,3	4,833
	Tretraso=(6Tdata+2Tct)	64	52	28	25,8	16	14,5

Tabla 2.1.1.3-1

En la segunda fila se halla el tiempo de conmutación total que es la suma del tiempo de conmutación mas el tiempo de preámbulo (en este caso se ha escogido el mínimo posible que permite cada uno de los dispositivos, aunque se recomienda que el preámbulo sea mayor que el mínimo. El tiempo para transmitir la trama (que se ha llamado t_{data} se obtiene de dividir los bits a transmitir entre la velocidad física a la que este funcionando el dispositivo. Se observa como el tiempo de conmutación (incluyendo el preámbulo) es dominante frente al tiempo empleado para transmitir una trama cuando se trabaja a velocidades elevadas. Por este motivo se plantean soluciones donde se encadenan varias tramas de datos para transmitirlas en un mismo sentido. El problema de esta solución es que se esta aumentando el retardo que puede ser muy perjudicial para los procedimientos de control.

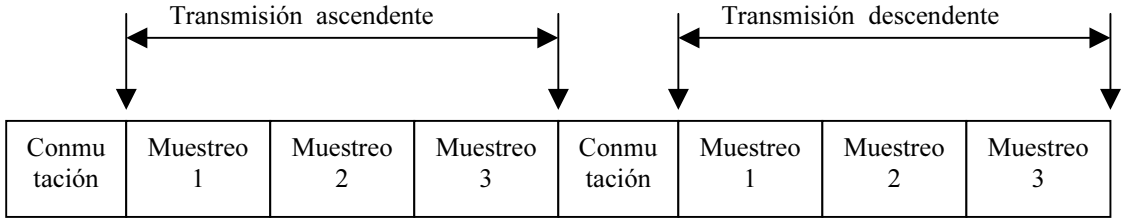


Figura 2.1.1.3 - 6: Reducción del efecto del tiempo de conmutación..

De la tabla se deduce que si se trabaja con un solo paquete de datos en cada ciclo canal ascendente/descendente interesa escoger el XTR - 434 a 100 Kbps. Por el

contrario si se puede permitir un retardo mayor y consecuentemente tramas mayores uniendo varios paquetes de datos interesa aprovechar la mayor velocidad física del BIM2-443-160Kbps. De todas formas se recuerda el hecho de que los propios fabricantes indican que los módulos se deben usar empleando técnicas de compensación de símbolos para que no se degrade la calidad de la comunicación. Estas técnicas implican el empleo de redundancia enviado un bit y su complementario o un byte y su complementario esto multiplica por 2 los tiempos de transmisión con lo que se tiene que realmente no se puede trabajar a la velocidad máxima que anuncia el fabricante sino a la mitad.

2.1.2 ISM 868 MHz

2.1.2.1 Legislación Radiofrecuencia banda 868 MHz.

La banda de 869 MHz es una banda de uso común sin necesidad de licencia de reciente liberación, las caracteríticas de esta banda son similares a la de la banda de 433 MHz. Es una de las bandas ISM (ICM aplicaciones industriales- científicas -médicas). La principal restricción es la limitación de potencia. Para obtener la legislación detallada asociada a esta banda se acude al Cuadro Nacional de Atribución de Frecuencias (CNAF).

ATRIBUCIÓN A LOS SERVICIOS según el RR de la UIT		
470 - 890 MHz		
Región 1	**Región 2**	**Región 3**
470 - 790 RADIODIFUSIÓN S5.149 S5.291A S5.294 S5.296 S5.300 S5.302 S5.304 S5.306 S5.311 S5.312	**470 - 512** RADIODIFUSIÓN Fijo Móvil S5.292 S5.293	**470 - 585** FIJO MÓVIL RADIODIFUSIÓN S5.291 S5.298
	512 - 608 RADIODIFUSIÓN S5.297	**585 - 610** FIJO MÓVIL RADIODIFUSIÓN RADIONAVEGACIÓN S5.149 S5.305 S5.306 S5.307
	608 - 614 RADIOASTRONOMÍA Móvil por satélite, salvo Móvil aeronáutico por satélite (Tierra-espacio)	
790 - 862 FIJO RADIODIFUSIÓN S5.312 S5.314 S5.315 S5.316 S5.319 S5.321	**614 - 806** RADIODIFUSIÓN Fijo Móvil S5.293 S5.309 S5.311	**610 - 890** FIJO MÓVIL S5.317A RADIODIFUSIÓN
862 - 890 FIJO MÓVIL, salvo móvil aeronáutico S5.317A RADIODIFUSIÓN S5.322 S5.319 S5.323	**806 - 890** FIJO MÓVIL S5.317A RADIODIFUSIÓN S5.317 S5.318	S5.149 S5.305 S5.306 S5.307 S5.311 S5.320

ATRIBUCIÓN NACIONAL	OBSERVACIONES	USOS
470 - 890 MHz		
470 - 790 RADIODIFUSIÓN Servicio móvil terrestre	S5.296 S5.302 ATRIBUCIÓN ADICIONAL A TÍTULO SECUNDARIO AL SERVICIO MÓVIL TERRESTRE PARA APLICACIONES AUXILIARES DE LA RADIODIFUSIÓN UN - 35	P
790 - 830 RADIODIFUSIÓN	UN - 35	P
830 - 862 FIJO MÓVIL, salvo móvil aeronáutico RADIODIFUSIÓN	S5.316 UN - 36 DVB-T	M M P
862 - 868 FIJO	UN - 111 UN - 118 MICRÓFONOS SIN HILOS	M
868 - 890 MÓVIL, salvo móvil aeronáutico	S5.322 ATRIBUCIÓN A SISTEMAS MÓVILES PARA DIVERSAS APLICACIONES UN - 40 UN - 41, UN - 73 UN - 39 TLEMANDO Y TELEMEDIDA UN - 104 CT1-E UN - 113 TETRA * Usos M y C (según notas UN)	*

Figura 2.1.2.1- 1: CNAF

La nota de utilización que se corresponde con la aplicación del proyecto es la UN-39:

UN - 39

Banda de frecuencias 868 a 870 MHz.

Esta banda se destina para aplicaciones de baja potencia y de datos en general conforme al siguiente cuadro de clasificación.

- Alarmas:
  868,600 - 868,700 MHz con 10 mW de potencia máxima y 25 kHz de canalización
  869,250 - 869,300 MHz con 10 mW de potencia máxima y 25 kHz de canalización
  869,650 - 869,700 MHz con 25 mW de potencia máxima y 25 kHz de canalización
- Dispositivos personales de alarma y emergencia: 869,200 - 869,250 MHz con 10 mW de potencia máxima y 25 kHz de canalización
- Dispositivos baja potencia de aplicaciones genéricas:
  868,000 - 868,600 MHz con 25 mW de potencia máxima
  868,700 - 869,200 MHz con 25 mW de potencia máxima
  869,300 - 869,400 MHz con 100 mW de potencia máxima y 25 kHz de canalización
  869,400 - 869,650 MHz con 500 mW de potencia máxima y 25 kHz de canalización
  869,700 - 870,000 MHz con 5 mW de potencia máxima

El resto de las características y condiciones de uso de esta banda se ajustarán a la Recomendación CEPT/ERC 70-03 Anexos 1 y 7.

El uso de estas frecuencias con las características indicadas se considera común cuando la potencia de los equipos sea inferior o igual a 100 mW medida tanto como salida de equipo o como potencia radiada aparente (p.r.a.).

Los títulos habilitantes existentes en estas frecuencias, habrán de ajustarse a las características indicadas en esta nota a mas tardar a la renovación de los mismos.

Los dispositivos que existen en el mercado y que se van a estudiar trabajan en la frecuencia central de 869.85 MHz, los que los ubica según la UN-39 dentro de los dispositivos de baja frecuencia para aplicaciones genéricas y concretamente en el intervalo de frecuencias de 869.7 MHz a 870 MHz. La potencia máxima que se permite es de 5 mW (aproximadamente 7 dBm) y no se tiene limitaciones de tiempo de uso, como se describe en el anexo 1 de la Recomendación CEPT/ERC 70-03 (Figura 2.1.1.1-2).

El interés de esta banda es que al ser de reciente liberación no existen todavía muchos sistemas trabajando en ella, aunque se debe de tener en cuenta que esta banda esta bastante cerca de la banda de trabajo de la telefonía móvil GSM, por lo que las emisiones de teléfonos móviles próximos podrían bloquear el sistema de comunicación.

2.1.2.2 Dispositivos existentes en el mercado.

Los dispositivos existentes en el mercado que se van analizar son el BIM3 y el XTR – 869. Son de las mismas empresas (AUREL y RadioMetrix) de los módulos ya vistos para la banda de 433 MHz. Los módulos vuelven a ser muy similares entre ellos, prácticamente tienen las mismas características y mismo modo de empleo.

AUREL XTR – 869

El módulo XTR – 869 de Aurel es idéntico físicamente a su predecesor a 433 MHz. La forma de emplear este módulo también es idéntica, se tienen dos líneas para seleccionar entre transmisión o recepción. También existe una salida analógica y una señal para avisar al controlador que existe una portadora entrante.

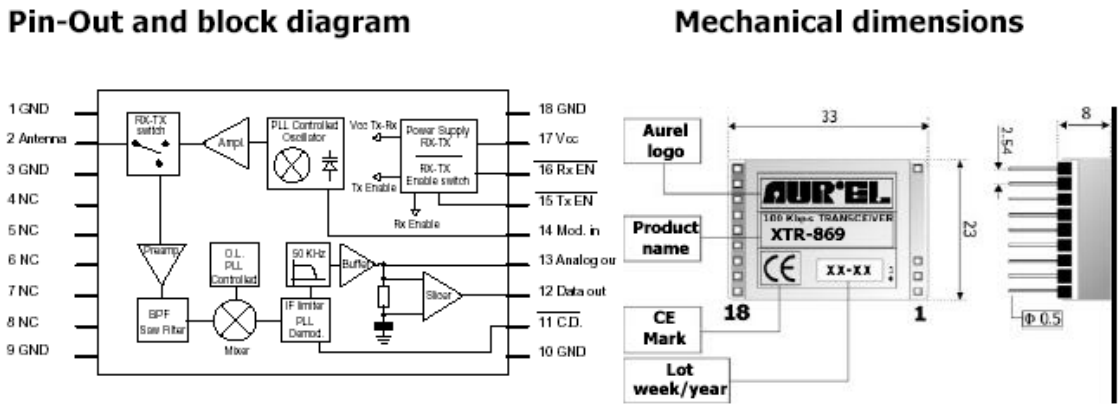


Figura 2.1.2.2 –2: Módulo Aurel XTR – 869.

En cuanto a las características en frecuencia se comprueba como trabaja en la frecuencia central de 869,85 MHz dentro del intervalo de 869.7 a 870 MHz. La potencia de salida es 7 dBm como máximo tal y como permite la legislación, aunque habría que consultar al fabricante si este nivel es considerando ya conectada una antena de las que él mismo distribuye, si no fuese así habría que reducir la potencia de salida con algún tipo de atenuador o disminuyendo la tensión de alimentación.

Characteristics	Min	Typical	Max	Unity	Remarks
Voltage supply	4.5	5	5.5	Vdc	
Absorbed current (TX ON)	24	28	32	mA	
Absorbed current (RX ON)	10	11	12	mA	
Absorbed current (TX/RX OFF)			100	nA	
RX Section					
Reception frequency		869.85		MHz	
RF sensitivity		-100	-102	dBm	See note 1
IF passband		150		KHz	
Interferences rejection [±5% band's extremities]		-80		dBm	
RF spurious emissions in antenna	**ETS 300 220 compliance**				See note 2
Output square wave			50	KHz	
Output low logic level		0,1		V	See note 4
Output high logic level		3,5		V	See note 4
Carrier Detect (CD) threshold	-96	-98		dBm	
RX Switch-on time		1		mS	
TX Section					
Transmission frequency		869.85		MHz	
Modulation passband			50	KHz	
FM deviation		±25		KHz	
TX output power			7	dBm	
Antenna impedance		50		Ω	
TX Switch-on time		1		mS	
Operating temperature	-20		+80	°C	
Working temperature [ETS 300 220]	-20		+55	°C	
Dimensions	33 x 23 x 8 mm				

Figura 2.1.2.2 –3: Características en frecuencia XTR - 869.

El tiempo de conmutación es de 1 milisegundo pero también la necesidad de un preámbulo de al menos 1 ms por lo que se tiene el mismo tiempo de conmutación total de 2 ms. En las especificaciones también se hace referencia a la necesidad de mantener la proporción de unos y ceros equilibrada, por lo que se tendrían que aplicar las mismas técnicas de compensación sobre el bit o sobre el byte.

RadioMetrix BIM3 – 869 – 64

El módulo BIM3 presenta la misma funcionalidad y características que versión previa a 433 MHz salvo por el hecho de que no existe una versión que trabaja a 160 Kbps.

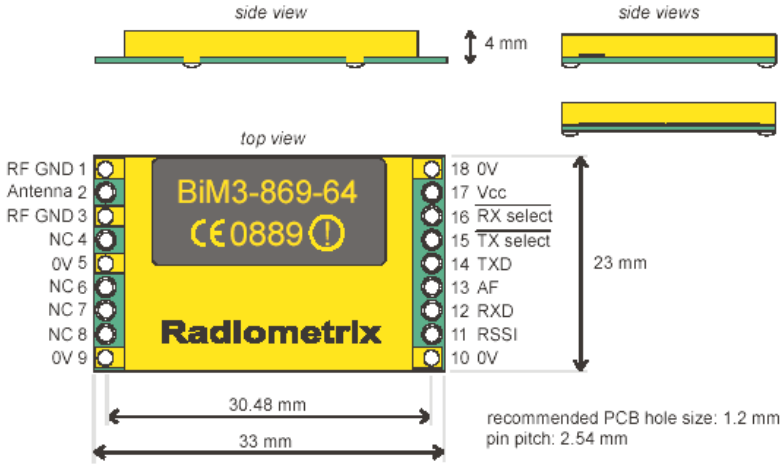


Figura 2.1.2.2 – 4: Módulo BIM3 – 869 - 64.

RF parameters	pin	min.	typ.	max.	units	notes
General						
Antenna pin impedance	2	-	50	-	Ω	Tx or Rx
RF centre frequency	-	-	869.85	-	MHz	EU version
RF centre frequency	-	-	914.50	-	MHz	US version
Transmitter						
RF power output (*all versions*)	2	-2	+1	+3	dBm	Vcc=5V/3.3V
Initial frequency accuracy	-	-25	0	+25	kHz	
Overall frequency accuracy	-	-50	0	+50	kHz	
FM deviation	-	30	40	50	kHz	
Modulation bandwidth	-	DC	-	35	kHz	
Modulation distortion	-	-	5	10	%	
TX harmonics	2	-	-	-40	dBm	
TX spectral bandwidth @-40dBc	2	-	-	250	kHz	worst case
Receiver						
RF sensitivity, 10dB S/N	2, 13	-100	-105	-	dBm	
RF sensitivity, 1ppm BER	2, 12	-92	-97	-	dBm	
RSSI threshold	2, 11	-	-115	-	dBm	
IF bandwidth	-	-	180	-	kHz	
AF bandwidth	13	-	35	-	kHz	
Image rejection (fRF-21.4MHz)	2	35	>40	-	dB	
RX LO leakage, conducted	2	-	-	-57	dBm	
RX LO leakage, radiated	-	-	-	-57	dBm	
Ultimate (S+N)/N	13	-	>40	-	dB	-70dBm RF
Initial frequency accuracy	-	-15	0	+15	kHz	

Figura 2.1.2.2 –5: Características en frecuencia BIM3 – 869 - 64.

En cuanto a las características en frecuencia se comprueba como este módulo trabaja en la frecuencia central de 868.85 MHz con un ancho de banda dentro de lo permitido. El módulo BIM3 tiene peores prestaciones en cuanto a velocidad, el tiempo de conmutación total es de 4ms incluyendo el preámbulo.

Baseband parameters	pin	min.	typ.	max.	units	notes
Tx –>Rx						
Linear baseband bandwidth	13	0.08	-	32	kHz	TXD to AF
Balanced code bit rate	12	-	-	64	kb/s	
Time between code transitions	14	15.6	-	1000	*µs*	
Averaged code mark:space	14	30	50	70	%	in any 2ms
Dynamic Timing						
Rx power up with signal present						
Power up to valid RSSI, tPU-RSSI	11	-	5	6	ms	
Power up to stable data, tPU-data	12	-	4	6	ms	
Signal applied with Rx on						
Signal to valid RSSI, tsig-RSSI	11	-	1.6	2	ms	
Signal to stable data, tsig-data	12	-	3	4	ms	
TX power up to full RF	2	-	1	1.5	ms	

Figura 2.1.2.2 – 6: Características temporización BIM3 - 869 - 64.

2.1.2.3 Análisis de viabilidad de los dispositivos ISM en la banda de 868 MHz.

El análisis que se realizó para la banda de 433 MHz conlleva resultados similares para esta banda. El alcance es similar y esta determinado por lo presencia de obstáculos. En este caso se preveen menos fuentes de interferencias de sistemas similares, pues como se ha comentado esta es una banda de reciente liberación para estas aplicaciones, aunque dada la cercanía a la banda de trabajo de GSM se podrían recibir interferencias de teléfonos móviles en las cercanías. Las antenas en esta banda requieren menores dimensiones que en la banda de 433 MHz, lo cual puede ser interesante.

En cuanto a la temporización y acceso al medio se obtienen resultados similares en los que el tiempo no útil de conmutación ya sea por conmutación física, de transmisión a recepción o viceversa, o la necesidad de un preámbulo para facilitar el sincronismo, hacen que se tenga que relajar la tasa de muestreo que se especificaba inicialmente de 1 ms a una tasa de unos 10 ms.

2.1.3 ISM 2.4 GHz

2.1.3.1 Legislación Radiofrecuencia banda 2.4 GHz.

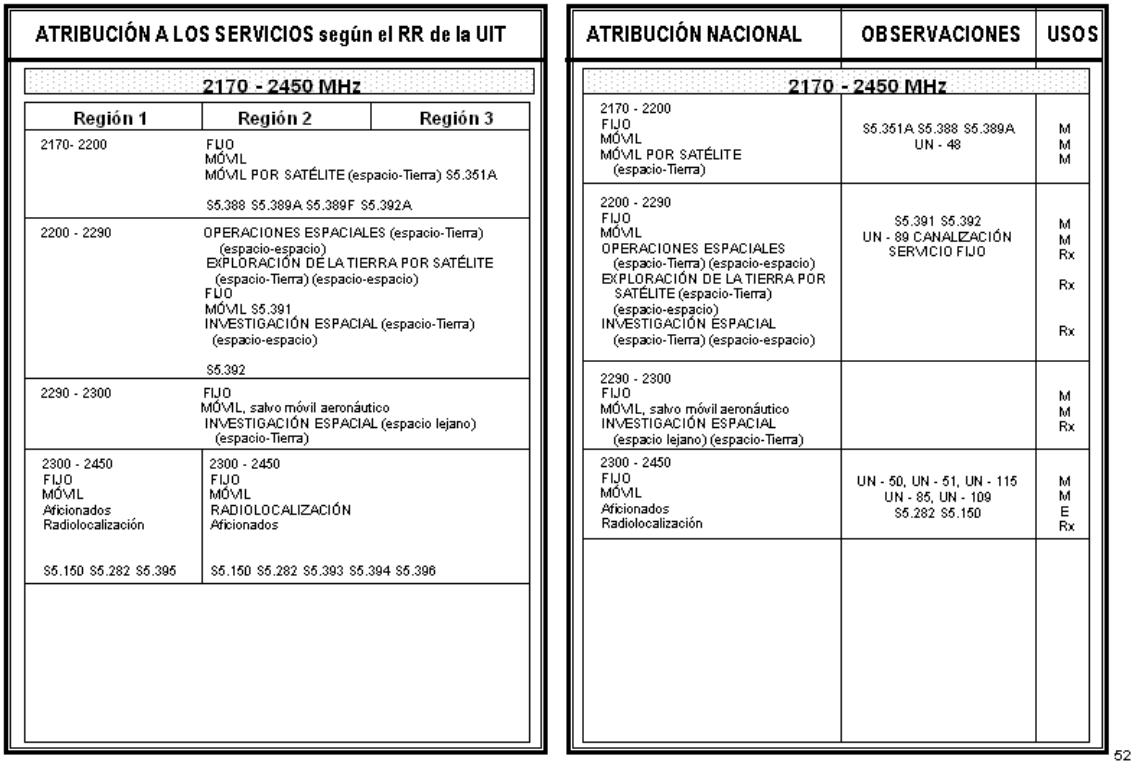
La legislación de esta banda ha cambiado en los últimos años. Antiguamente la normativa especificaba la siguiente disponibilidad de frecuencias para aplicaciones de redes de área local inalámbricas.

Canal	Frecuencia	USA	Canadá	Europa	España	Francia	Japón
1	2.412 MHz	X	X	X	-	-	-
2	2.417 MHz	X	X	X	-	-	-
3	2.422 MHz	X	X	X	-	-	-
4	2.427 MHz	X	X	X	-	-	-
5	2.432 MHz	X	X	X	-	-	-
6	2.437 MHz	X	X	X	-	-	-
7	2.442 MHz	X	X	X	-	-	-
8	2.447 MHz	X	X	X	-	-	-
9	2.452 MHz	X	X	X	-	-	-
10	2.457 MHz	X	X	X	X	X	-
11	2.462 MHz	X	X	X	X	X	-
12	2.467 MHz	-	-	X	-	X	-
13	2.472 MHz	-	-	X	-	X	-
14	2.484 MHz	-	-	-	-	-	X

Tabla de canales definidos en la banda ISM, y su regulación para los diferentes países.

Tabla 2.1.3.1 - 1

Con lo que se aprecia que en España quedaba muy limitado el número de canales. Esa normativa se ha actualizado como se desprende de la información del Ministerio de Ciencia y Tecnología. En el Cuadro Nacional de Atribución de Frecuencias (CNAF) aparecen las siguientes tablas.



ATRIBUCIÓN A LOS SERVICIOS según el RR de la UIT

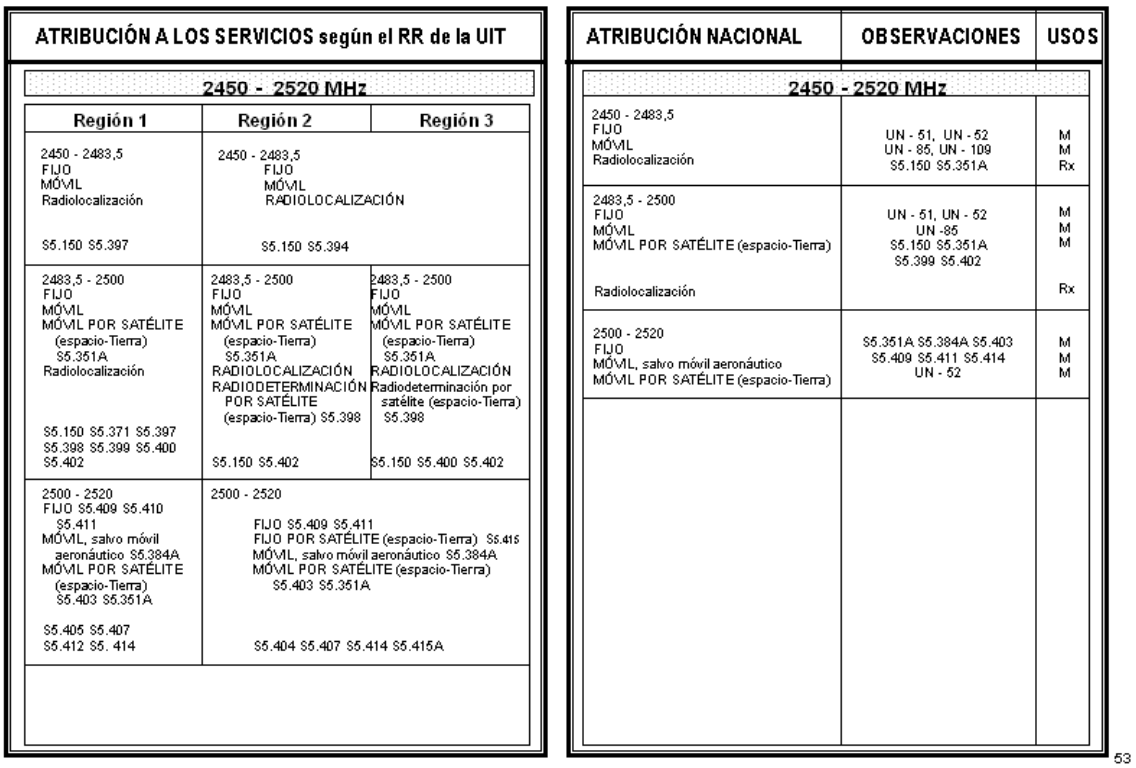
2170 - 2450 MHz

Región 1	Región 2	Región 3
2170- 2200	FIJO MÓVIL MÓVIL POR SATÉLITE (espacio-Tierra) S5.351A S5.388 S5.389A S5.389F S5.392A	
2200 - 2290	OPERACIONES ESPACIALES (espacio-Tierra) (espacio-espacio) EXPLORACIÓN DE LA TIERRA POR SATÉLITE (espacio-Tierra) (espacio-espacio) FIJO MÓVIL S5.391 INVESTIGACIÓN ESPACIAL (espacio-Tierra) (espacio-espacio) S5.392	
2290 - 2300	FIJO MÓVIL, salvo móvil aeronáutico INVESTIGACIÓN ESPACIAL (espacio lejano) (espacio-Tierra)	
2300 - 2450 FIJO MÓVIL Aficionados Radiolocalización S5.150 S5.282 S5.395	2300 - 2450 FIJO MÓVIL RADIOLOCALIZACIÓN Aficionados S5.150 S5.282 S5.393 S5.394 S5.396	

ATRIBUCIÓN NACIONAL	OBSERVACIONES	USOS
2170 - 2450 MHz		
2170 - 2200 FIJO MÓVIL MÓVIL POR SATÉLITE (espacio-Tierra)	S5.351A S5.388 S5.389A UN - 48	M M M
2200 - 2290 FIJO MÓVIL OPERACIONES ESPACIALES (espacio-Tierra) (espacio-espacio) EXPLORACIÓN DE LA TIERRA POR SATÉLITE (espacio-Tierra) (espacio-espacio) INVESTIGACIÓN ESPACIAL (espacio-Tierra) (espacio-espacio)	S5.391 S5.392 UN - 89 CANALIZACIÓN SERVICIO FIJO	M M Rx Rx Rx
2290 - 2300 FIJO MÓVIL, salvo móvil aeronáutico INVESTIGACIÓN ESPACIAL (espacio lejano) (espacio-Tierra)		M M Rx
2300 - 2450 FIJO MÓVIL Aficionados Radiolocalización	UN - 50, UN - 51, UN - 115 UN - 85, UN - 109 S5.282 S5.150	M M E Rx

52

Figura 2.1.3 – 2 a)



ATRIBUCIÓN A LOS SERVICIOS según el RR de la UIT

2450 - 2520 MHz

Región 1	Región 2	Región 3
2450 - 2483,5 FIJO MÓVIL Radiolocalización S5.150 S5.397	2450 - 2483,5 FIJO MÓVIL RADIOLOCALIZACIÓN S5.150 S5.394	
2483,5 - 2500 FIJO MÓVIL MÓVIL POR SATÉLITE (espacio-Tierra) S5.351A Radiolocalización S5.150 S5.371 S5.397 S5.398 S5.399 S5.400 S5.402	2483,5 - 2500 FIJO MÓVIL MÓVIL POR SATÉLITE (espacio-Tierra) S5.351A RADIOLOCALIZACIÓN RADIODETERMINACIÓN POR SATÉLITE (espacio-Tierra) S5.398 S5.150 S5.402	2483,5 - 2500 FIJO MÓVIL MÓVIL POR SATÉLITE (espacio-Tierra) S5.351A RADIOLOCALIZACIÓN Radiodeterminación por satélite (espacio-Tierra) S5.398 S5.150 S5.400 S5.402
2500 - 2520 FIJO S5.409 S5.410 S5.411 MÓVIL, salvo móvil aeronáutico S5.384A MÓVIL POR SATÉLITE (espacio-Tierra) S5.403 S5.351A S5.405 S5.407 S5.412 S5. 414	2500 - 2520 FIJO S5.409 S5.411 FIJO POR SATÉLITE (espacio-Tierra) S5.415 MÓVIL, salvo móvil aeronáutico S5.384A MÓVIL POR SATÉLITE (espacio-Tierra) S5.403 S5.351A S5.404 S5.407 S5.414 S5.415A	

ATRIBUCIÓN NACIONAL	OBSERVACIONES	USOS
2450 - 2520 MHz		
2450 - 2483,5 FIJO MÓVIL Radiolocalización	UN - 51, UN - 52 UN - 85, UN - 109 S5.150 S5.351A	M M Rx
2483,5 - 2500 FIJO MÓVIL MÓVIL POR SATÉLITE (espacio-Tierra) Radiolocalización	UN - 51, UN - 52 UN -85 S5.150 S5.351A S5.399 S5.402	M M M Rx
2500 - 2520 FIJO MÓVIL, salvo móvil aeronáutico MÓVIL POR SATÉLITE (espacio-Tierra)	S5.351A S5.384A S5.403 S5.409 S5.411 S5.414 UN - 52	M M M

53

Figura 2.1.3 – 2 b)

En estas tablas se especifican para cada rango de frecuencias las normas que se han de aplicar. De estos cuadros se deduce que las notas de utilización que afectan son la UN 51, UN 52, UN 85, UN 109 para el rango de los 2400 a 2500 MHz..

UN - 51

Bandas de frecuencias designadas para aplicaciones industriales, científicas, y médicas (ICM).

- 2400 a 2500 MHz (frecuencia central 2450 MHz)
- 5725 a 5875 MHz (frecuencia central 5800 MHz)
- 24,00 a 24,25 GHz (frecuencia central 24,125 GHz)
- 61,00 a 61,50 GHz (frecuencia central 61,250 GHz)

Los servicios de radiocomunicaciones que funcionen en las citadas bandas deberán aceptar la interferencia perjudicial resultante de estas aplicaciones.

Los equipos ICM que funcionen en estas bandas estarán sujetos a las medidas prácticas que adopte la Secretaría de Estado de Telecomunicaciones y para la Sociedad de la Información si fuera necesario para que las radiaciones fuera de banda de estos equipos sean mínimas, sin perjuicio de lo establecido en el Real Decreto 444/1994 de 11 de marzo sobre requisitos de protección relativos a compatibilidad electromagnética.

La utilización de estas frecuencias para las aplicaciones indicadas se considera uso común.

UN - 52

Banda de 2500 a 2700 MHz.

Se han realizado nuevas atribuciones del servicio móvil por satélite con categoría de servicio primario entre 2483,5 y 2520 MHz en sentido espacio-Tierra y entre 2670 y 2690 MHz en el sentido Tierra-espacio, por lo que las canalizaciones que han venido siendo utilizadas deben ser reordenadas y por ello solo se asignarán frecuencias en las mismas en casos excepcionales y con categoría de servicio secundario.

El abandono de esta banda por los usuarios de radioenlaces del servicio fijo deberá producirse a medida que vayan caducando las concesiones o vaya siendo necesario. En sustitución de dicha canalización se adopta la que se indica en la UN-90 y que la configuran las subbandas de frecuencias 2520 a 2593 MHz junto con 2597 a 2670 MHz para ser utilizada por el servicio fijo en enlaces de pequeña y mediana capacidad.

Las solicitudes de frecuencias que se reciban para las nuevas canalizaciones del servicio fijo solo podrán ser atendidas cuando el nivel de abandono de las antiguas canalizaciones lo permita.

En la banda entre 2450 y 2700 MHz se autorizan utilizaciones de enlaces móviles de televisión (ENG) con categoría de servicio secundario.

La CMR-2000 ha identificado, entre otras, la banda de frecuencias 2500-2690 MHz para futuras ampliaciones de los sistemas de tercera generación IMT-2000/UMTS.

UN - 85

Banda de frecuencias 2400 a 2483,5 MHz.

Estas frecuencias podrán ser utilizadas en redes de área local para la interconexión sin hilos entre ordenadores y/o terminales y dispositivos periféricos para aplicaciones en interior de edificios.

La potencia total será inferior a 100 mW (PIRE). Otras condiciones de uso han de ser conforme a la Recomendación CEPT/ERC 70-03 Anexo 3.

Esta utilización se considera de uso común.

Esta banda de frecuencias también podrá utilizarse para aplicaciones generales de baja potencia en recintos cerrados y exteriores de corto alcance.

La potencia radiada máxima será inferior a 100 mW.

Esta utilización se considera de uso común.

En ambos casos, las características radioeléctricas de estos equipos se ajustarán a las especificaciones ETSI ETS 300 328, ETS 300 440 o bien al estándar específico, si es el caso y en base a lo anterior deberá realizarse la correspondiente evaluación de la conformidad.

UN - 109

Frecuencias para enlaces de vídeo de corto alcance.

Se destinan las frecuencias 2421 MHz, 2449 MHz y 2477 MHz para su utilización, entre otras aplicaciones, en enlaces de vídeo de corto alcance para aplicaciones genéricas, tanto en interior de edificios como en exteriores, para alcances cortos en circuitos cerrados y equipos de potencia inferior a 500 mW con anchura de banda de emisión ajustada a la calidad de señal requerida.

Las instalaciones de este tipo deben de aceptar la interferencia perjudicial que pudiera resultar de aplicaciones ICM u otros usos de radiocomunicaciones en estas frecuencias.

Esta utilización se considera de uso común.

La nota de utilización UN-51 define de manera genérica la banda de 2400 a 2500 MHz además de otras, como una banda ICM (o ISM), sin especificar detalladamente como se puede usar. La UN-52 se refiere a la banda de 2483,5 MHz a

2520 MHz y los usos que se puede hacer de ella pero la banda que usan los dispositivos comerciales ISM es la banda de 2400 MHz a 2483,5 MHz cuya utilización se recoge en la UN-85 y se refiere a aplicaciones de redes de ordenadores y/o terminales y periféricos tanto para interior como exterior con una limitación de potencia de 100 mW (PIRE). La nota UN-109 es para aplicaciones de vídeo de corto alcance y no pertenece a las aplicaciones de este proyecto.

La recomendación que se menciona en la nota de utilización UN-85 es la siguiente, aunque básicamente viene a decir lo mismo.

Annex 3 **Wideband Data Transmission systems and HIPERLANs**

ERC/REC 70-03E
Page 34

Scope of Annex
This annex covers frequency bands and regulatory as well as informative parameters recommended for Wideband data transmission systems formerly known as (Radio Local Area Networks (RLANs)) within the band 2400-2483.5 MHz and for High Performance Radio Local Area Networks (HIPERLANs) within the bands 5150-5350 MHz, 5470-5725 MHz and 17.1-17.3 GHz.

Regulatory parameters related to Annex 3

	Frequency Band	Power		Duty cycle	Channel spacing	ERC Decision	Notes
a	2400 - 2483.5 MHz	100 mW	e.i.r.p.	No Restriction	No spacing	ERC DEC (01)07	For direct sequence spread spectrum, the maximum spectrum power density is limited to -20 dBW/1 MHz. For FHSS the maximum spectrum power density is limited to -10 dBW/100 kHz.
b	5150 - 5350 MHz	200 mW	Max mean		No spacing	ERC DEC (99)23	Indoor use only
c	5470 - 5725 MHz	1 W	Max mean	No Restriction	No spacing	ERC DEC (99)23	
d	17.1 - 17.3 GHz	100 mW	e.i.r.p.	No Restriction	No spacing		

Resumiendo, de la normativa vista en el Ministerio de Ciencia y Tecnología se desprende que la normativa a aplicar en España es la ERC 70 03, que es la que se aplica en Europa.

2.1.3.2 Técnicas de espectro ensanchado.

La ventaja de trabajar en esta banda es el ancho de banda disponible. Existen dos tipos técnicas, la banda estrecha y la banda ancha, también conocida espectro ensanchado, ésta ultima es la que más se utiliza.

En mayo de 1985, y tras cuatro años de estudios, el FCC (Federal Communications Comission), la agencia Federal del Gobierno de Estados Unidos encargada de regular y administrar en materia de telecomunicaciones, asignó las bandas IMS (Industrial, Scientific and Medical) 902-928 MHz, 2,400-2,4835 GHz, 5,725-5,850 GHz a las redes inalámbricas basadas en espectro ensanchado. Entre ellas, el IEEE 802.11 incluyo en su especificación las frecuencias en torno a 2,4 GHz que se habían

convertido ya en el punto de referencia a nivel mundial, la industria se había volcado en ella y está disponible a nivel mundial.

La tecnología de espectro ensanchado, utiliza **todo el ancho de banda disponible**, en lugar de utilizar una portadora para concentrar la energía a su alrededor. Tiene muchas características que le hacen sobresalir sobre otras tecnologías de radiofrecuencias (como la de banda estrecha, que utiliza microondas), ya que, por ejemplo, posee excelentes propiedades en cuanto a inmunidad a interferencias y a sus posibilidades de encriptación. Esta, como muchas otras tecnologías, proviene del sector militar.

Existen dos tipos de tecnología de espectro ensanchado:

- FHSS (Frequency Hopping Spread Spectrum)
- DSSS (Direct Sequence Spread Spectrum)

Espectro Ensanchado por Secuencia Directa (DSSS)

En esta técnica se genera un patrón de bits redundante (señal de chip) para cada uno de los bits que componen la señal. Cuanto mayor sea esta señal, mayor será la resistencia de la señal a las interferencias. El estándar IEEE 802.11 recomienda un tamaño de 11 bits, pero el optimo es de 100. En recepción es necesario realizar el proceso inverso para obtener la información original.

La secuencia de bits utilizada para modular los bits se conoce como secuencia de Barker (también llamado código de dispersión o *PseudoNoise*). Es una secuencia rápida diseñada para que aparezca aproximadamente la misma cantidad de 1 que de 0. Un ejemplo de esta secuencia es el siguiente:

+1 –1 +1 +1 –1 +1 +1 +1 –1 –1 –1 –1

Solo los receptores a los que el emisor haya enviado previamente la secuencia podrán recomponer la señal original. Además, al sustituir cada bit de datos a transmitir, por una secuencia de 11 bits equivalente, aunque parte de la señal de transmisión se vea afectada por interferencias, el receptor aún puede reconstruir fácilmente la información a partir de la señal recibida. Esta secuencia proporciona 10.4dB de aumento del proceso, el cual reúne los requisitos mínimos para las reglas fijadas por la FCC.

En la figura 2.1.3.2 - 1 se puede observar como se utiliza la secuencia de *Barker* para codificar la señal original a transmitir:

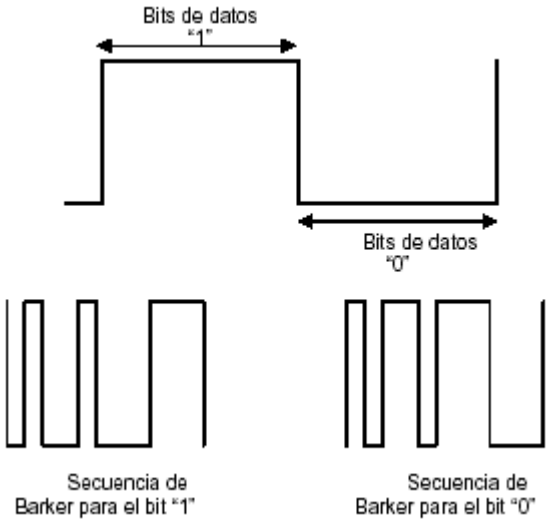


Figura 2.1.3.2 – 1: Codificación Barker.

Una vez aplicada la señal de chip, el estándar IEEE 802.11 ha definido dos tipos de modulación para la técnica de espectro ensanchado por secuencia directa (DSSS), la modulación ***DBPSK*** (Differential Binary Phase Shift Keying) y la modulación ***DQPSK*** (Differential Quadrature Phase Shift Keying), que proporcionan una velocidad de transferencia de 1 y 2 Mbps respectivamente.

El IEEE revisó posteriormente este estándar, y en esta revisión, conocida como 802.11b, además de otras mejoras en seguridad, aumentó esta velocidad hasta los 11Mbps, lo que incrementa notablemente el rendimiento de este tipo de redes. En otra posterior revisión se ha aumentado la velocidad a 22 Mbps.

En el caso de Estados Unidos y Europa la tecnología DSSS utiliza un rango de frecuencias que va desde los 2,4 GHz hasta los 2,4835 GHz, lo que permite tener un ancho de banda total de 83,5 MHz. Este ancho de banda se subdivide en canales de 5 MHz, lo que hace un total de 14 canales independientes. Cada país esta autorizado a utilizar un subconjunto de estos canales.

En configuraciones donde existan mas de una celda, estas pueden operar simultáneamente y sin interferencias siempre y cuando la diferencia entre las frecuencias centrales de las distintas celdas sea de al menos 30 MHz, lo que reduce a tres el número de canales independientes y funcionando simultáneamente en el ancho de banda total de 83,5 MHz. Esta independencia entre canales nos permite aumentar la capacidad del sistema de forma lineal.

Espectro ensanchado por salto de frecuencia (FHSS)

La tecnología de espectro ensanchado por salto en frecuencia (FHSS) consiste en transmitir una parte de la información en una **determinada frecuencia durante un intervalo de tiempo** llamada *dwell time* e inferior a 400 ms. Pasado este tiempo se cambia la frecuencia de emisión y se sigue transmitiendo a otra frecuencia. De esta manera cada tramo de información se va transmitiendo en una frecuencia distinta durante un **intervalo muy corto de tiempo**.

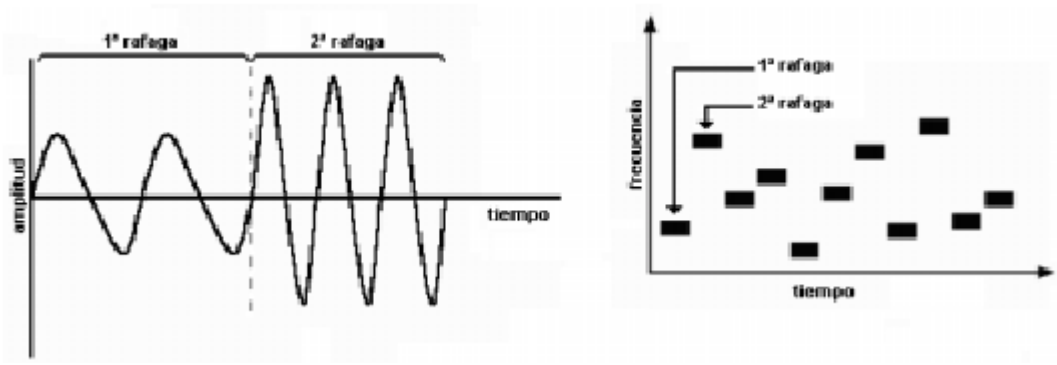


Figura 2.1.3.2 – 2: Codificación por salto en frecuencia.

El orden en los saltos en frecuencia se determina según una secuencia pseudoaleatoria almacenada en unas tablas, y que tanto el emisor y el receptor deben conocer.

Si se mantiene la sincronización en los saltos de frecuencias se consigue que, aunque en le tiempo se cambie de canal físico, a nivel lógico se mantiene un solo canal por el que se realiza la comunicación.

Esta técnica también utiliza la banda de los 2.4GHz, la cual organiza en 79 canales con un ancho de banda de 1MHz cada uno. El número de saltos por segundo es regulado por cada país, así, por ejemplo, Estados Unidos fija una tasa mínima de saltas de 2.5 por segundo.

El estándar IEEE 802.11 define la modulación aplicable en este caso. Se utiliza la modulación en frecuencia ***FSK*** (Frequency Shift Keying), con una velocidad de 1Mbps ampliable a 2Mbps. En la revisión del estándar 802.11b, esta velocidad también ha aumentado a 11Mbps.

La técnica FHSS seria equivalente a una multiplexación en frecuencia

Limite inferior	Límite superior	Rango regulatorio	Área geográfica
2.402 GHz	2.480 GHz	2.400-2.4835 GHz	América del Norte
2.402 GHz	2.480 GHz	2.400-2.4835 GHz	Europa
2.473 GHz	2.495 GHz	2.471-2.497 GHz	Japón

Figura 2.1.3.2 – 3: Rango de frecuencias en FHSS.

Canal	Valor	Canal	Valor	Canal	Valor
2	2.402	28	2.428	54	2.454
3	2.403	29	2.429	55	2.455
4	2.404	30	2.430	56	2.456
5	2.405	31	2.431	57	2.457
6	2.406	32	2.432	58	2.458
7	2.407	33	2.433	59	2.459
8	2.408	34	2.434	60	2.460
9	2.409	35	2.435	61	2.461
10	2.410	36	2.436	62	2.462
11	2.411	37	2.437	63	2.463
12	2.412	38	2.438	64	2.464
13	2.413	39	2.439	65	2.465
14	2.414	40	2.440	66	2.466
15	2.415	41	2.441	67	2.467
16	2.416	42	2.442	68	2.468
17	2.417	43	2.443	69	2.469
18	2.418	44	2.444	70	2.470
19	2.419	45	2.445	71	2.471
20	2.420	46	2.446	72	2.472
21	2.421	47	2.447	73	2.473
22	2.422	48	2.448	74	2.474
23	2.423	49	2.449	75	2.475
24	2.424	50	2.450	76	2.476
25	2.425	51	2.451	77	2.477
26	2.426	52	2.452	78	2.478
27	2.427	53	2.453	79	2.479
				80	2.480

Figura 2.1.3.2 – 4: Canales FHSS.

2.1.3.3 Aerocomm

La compañía Aerocomm presenta una gran variedad de módulos que trabajan en la banda ISM 2,4 GHz.

Figura 2.1.3.3 – 1: Aerocomm AC4424

Elección del módulo Aerocomm mas adecuado para el proyecto.

En este apartado se analizan las diferentes características de los módulos Aerocomm para ver cual de ellos se adapta mejor a las características del proyecto. Los factores que hay que tener en cuenta son los siguientes:

- Rango de frecuencias y Potencia de transmisión tal que el conjunto módulo antena cumpla los requisitos de la normativa española y que permita un máximo alcance conforme a esta.

- Velocidad de transmisión los mas elevada posible y que permita un desempeño total o throughput admisible (es decir considerando cabeceras de protocolo, tiempos de conmutación si el sistema es semiduplex, etc.)

- Consumo aceptable.

- Rango de temperaturas.

- Tamaño y peso.

Aspectos radioeléctricos.

La normativa española nos permite trabajar en el rango de 2400 a 2483 Mhz con una PIRE (o eirp) máxima de 100mW.

Los módulos de Aerocomm suelen ser de tres tipos ACXX24-10, ACXX24-100, ACXX24-200, el 10, 100, y 200 es la PIRE que se consigue con una antena 3dBi, por tanto se tomará un modulo 100 con cuidado de no tomar una antena con ganancia mayor que 3 dBi.

Además se configura el módulo de tal forma que el conjunto de canales que se maneje con FHSS este dentro de lo que nos permite la legislación (2400-2483 MHz).

El modelo AC1524 presenta las variantes que aparecen en la figura 2.1.3.3 – 2. Se observa como no interesa pues no esta diseñado para trabajar con los 100mW de PIRE que buscamos. Este modelo es antiguo, de antes de que cambiara la normativa en España, y por eso se excluía su uso en España y Francia.

Part Number	US/FCC	CAN/IC	EUR/EN**	Portable	Mobile	Fixed
AC1524C-3A	X	X	X	X	X	X
AC1524C-10	X	X	X	X	X	X
AC1524C-150	X	X			X-30cm*	X-30cm*

* See RF Exposure warning on next page
** Does not include France and Spain
Note: The product approvals above are with antennas specified below.

Agency Identification Numbers

Part Number	US/FCC	CAN/IC	EUR/EN
AC1524C-3A	KQL-LX2400	CAN2268391182A	X
AC1524C-10	KQL-LX2400-10	CAN2268391182A	X
AC1524C-150	KQL-LX2400-150	CAN2268391182A	

RADIO	
Frequency Band	2.402 – 2.478 GHz
Radio Type	Frequency Hopping Spread Spectrum
Output Power (conducted, no antenna)	AC1524C-3A, 2.5mW typical AC1524C-10, 11.0mW typical AC1524C-150, 147.9mW typical
Effective Isotropic Radiated Power (EIRP with S191FL-5-RMM-2450S antenna, except with -3A)	AC1524C-3A, 3.1mW typical AC1524C-10, 25.1mW typical AC1524C-150, 288.4mW typical
Voltage	5V nominal ±2%, ±50mV ripple
Sensitivity	-90dBm typical
Range	AC1524C-3A, Indoors to 100 ft., Outdoors to 500 ft. AC1524C-10, Indoors to 300 ft., Outdoors to 3000 ft. AC1524C-150, Indoors to 500 ft., Outdoors to 10000 ft.
PLL Lock Time	500 µs typical

Figura 2.1.3.3 – 2: Características Aerocomm AC1524

El modelo AC3124 no presenta la versión a 100mW.

RADIO	
Frequency Band	2.402 – 2.478 GHz
Radio Type	Frequency Hopping Spread Spectrum
Output Power (conducted, no antenna)	AC3124-10, 10mW typical AC3124-200, 200mW typical
Effective Isotropic Radiated Power (EIRP with S191FL-5-RMM-2450S antenna, except with -10A)	AC3124-10, 25.1mW typical AC3124-200, 363.1mW typical
Voltage	5V nominal ±2%, ±50mV ripple
Sensitivity	-90dBm typical
Range	AC3124-10, Indoors to 300 ft., Outdoors to 3000 ft. AC3124-200, Indoors to 500 ft., Outdoors to 10000 ft.
PLL Lock Time	500 µs typical
ENVIRONMENTAL	

Figura 2.1.3.3 – 3: Características Aerocomm AC3124

El modelo 4424 si presenta la versión a 100mW se aprecia que es más reciente y ya está al tanto de la nueva normativa española.

Agency Approval Overview

Part Number	US/FCC	CAN/IC	EUR/EN	Portable	Mobile	Fixed
AC4424-10	X	X	X	X	X	X
AC4424-100	X	X	X		X-32cm*	X-32cm*
AC4424-200	X	X			X-32cm*	X-32cm*

* See RF Exposure warning on next page
Note: The product approvals above are with antennas specified below.

Agency Identification Numbers

Part Number	US/FCC	CAN/IC	EUR/EN
AC4424-10	KQL-PKLR2400	CAN2268391158A	X
AC4424-100	X	X	X
AC4424-200	KQL-PKLR2400-200	CAN2268391180A	

RADIO	
Frequency Band	US/Canada: 2.402 – 2.478 GHz France: 2.448 – 2.457 GHz
Radio Type	Frequency Hopping Spread Spectrum
Output Power (conducted, no antenna)	AC4424-10, 10mW typical AC4424-100, 50mW typical AC4424-200, 200mW typical
Effective Isotropic Radiated Power (EIRP with 3dBi gain antenna)	AC4424-10, 20mW typical AC4424-100, 100mW typical AC4424-200, 400mW typical
Voltage	5V nominal ±2%, ±50mV ripple
Sensitivity	-90dBm typical
Range (based on 3dBi gain antenna)	AC4424-10, Indoors to 300 ft., Outdoors to 3000 ft. AC4424-100, Indoors to 400 ft., Outdoors to 6000 ft. AC4424-200, Indoors to 500 ft., Outdoors to 10000 ft.

Figura 2.1.3.3 – 4: Características Aerocomm AC4424

El modelo AC5124 no tiene la versión a 100mW:

GENERAL	
Bus Interface	Serial (TTL Level Asynchronous) through 40 pin mini connector. AMP P/N 177986-1
Serial Interface Data Rate	Programmable to 882 Kbps. PC rates to 115.2 Kbps
Compliance	Certifiable under:
AC5124C-10	US (FCC 15.247); Canada (IC); Europe (EN)
AC5124C-200	US (FCC 15.247); Canada (IC)

Figura 2.1.3.3 – 5 a: Características Aerocomm AC5124

TRANSCEIVER	
Frequency Band	2.402 – 2.478 GHz
Transceiver Type	Frequency Hopping Spread Spectrum
Output Power	
AC5124C-10	10mW
AC5124C-200	200mW
Input Voltage	5V nominal +2%, + 50mV ripple
Sensitivity	-90dBm
RF Data Rate	882 Kbps
Range	Can be extended with directional antenna
AC5124C-10	Indoors up to 300 ft., Outdoors up to 3,000 ft.
AC5124C-200	Indoors up to 500 ft., Outdoors up to 10,000 ft.
Synchronization Time	Average = 750ms; Maximum = 1.5s

Figura 2.1.3.3 – 5 b: Características Aerocomm AC5124

En resumen, **en cuanto a potencia la mejor opción es el AC4424-100**, más sencilla a la hora de diseño. Está dentro de la normativa con la máxima potencia que se permite con antenas normales.

Existe la posibilidad de usar opciones con menor potencia a costa de reducir el rango o emplear antenas más directivas, que reducen la cobertura en ciertas regiones del espacio.

El kit de desarrollo asociado al AC4424X-100 se denomina SDK-4424C-100 100:

SDK-4424C-100 100: Includes:
(2) AC4424C-100 transceivers,
(2) RS232 Serial Adapter Boards,
(2) 6Vdc unregulated power supplies,
(2) Serial cables,
(2) S151FL-5-RMM-2450S dipole antennas with 5” pigtail and MMCX connector,
configuration/testing software, Integration engineering support

Nota: la X indica el rango de temperaturas C comercial, I Industrial

AC4424C-100 100: AC4424C with 50mW output power, interface data rates to 288Kbps, MMCX antenna connector, 0°C to 60°C.

AC4424I-100 100: AC4424I with 50mW output power, interface data rates to 288Kbps, MMCX antenna connector, -40°C to 80°C

En cuanto al **rango frecuencia** todos trabajan entre 2402 y 2478 MHz con lo que estamos dentro del rango permitido por la normativa española. Estos transceptores tienen este rango dividido en 64 canales a su vez repartidos en 4 conjuntos de 16. Se puede seleccionar que canales se van a utilizar.

En cuanto a la **antena** que trae el kit es la S151FL-5-RMM-2450S, está da 5 dBi de ganancia con lo que teóricamente se estaría por encima de los 100mW de PIRE, en la tabla 2.1.3.3 – 1 se muestran antenas aprobadas por el fabricante del módulo.

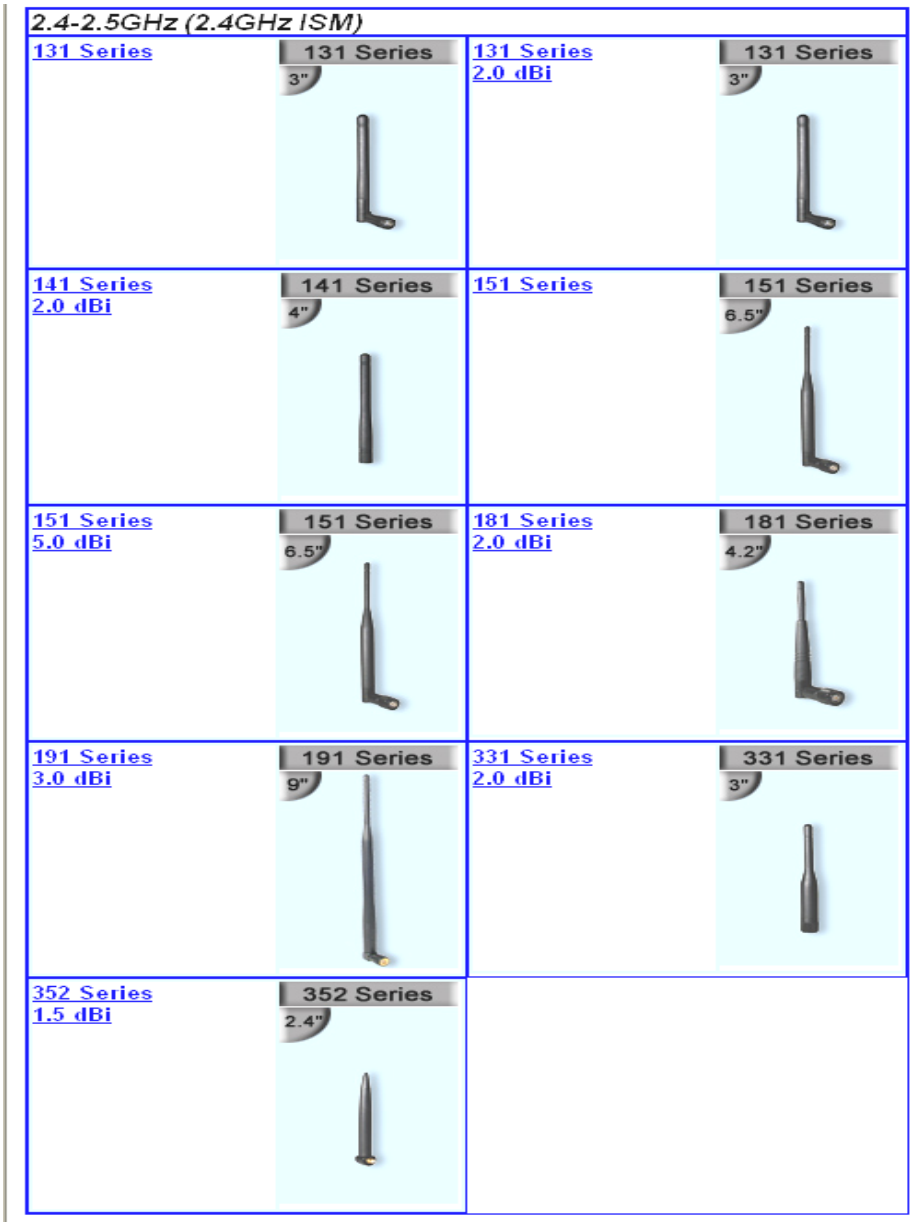


Figura 2.1.3.3 – 6: Posibles antenas del fabricante Nearson.

Approved Antenna List

Item	Part Number	Mfg.	Type	Gain (dBi)	AC4424X-10	AC4424X-100	AC4424X-200
1	WCP-2400-MMCX	Centurion	1/4 Wave Dipole	2	PMF	MF	MF
2	WCR-2400-SMRP	Centurion	1/4 Wave Dipole	2	PMF		
3	MFB24008RPN	Maxrad	Omni	8	MF		
4	BMMG24000MSMARP12'	Maxrad	Omni-Mobile	1	MF		
5	BMMG24005MSMARP12'	Maxrad	Omni-Mobile	5	MF		
6	MP24013TMSMARP12	Maxrad	Panel	13	MF		
7	MUF24005M174MSMARP12	Maxrad	Omni-Mobile	5	MF		
8	MC2400	Maxrad	Patch	2.5		MF	MF
9	NZH2400-MMCX (External)	AeroComm	Microstrip	1	PMF	MF	MF
10	NZH2400-I (Integral)	AeroComm	Microstrip	1	PMF	MF	MF
11	S131CL-5-RMM-2450S	Nearson	1/4 Wave Dipole	2		MF	MF
12	S181FL-5-RMM-2450S	Nearson	1/4 Wave Dipole	2	PMF	MF	MF
13	S191FL-5-RMM-2450S	Nearson	5/8 Wave Dipole	3	PMF	MF	MF
14	S151FL-5-RMM-2450S	Nearson	Omni	5	MF	MF	MF
15	MLPV1700	Maxrad	Omni-Mobile	4	MF	MF	MF

P=Portable, M=Mobile, F=Fixed/Basestation

Tabla 2.1.3.3 – 1: Antenas aprobadas por Aerocomm..

La serie 191 del fabricante Nearson tiene una ganancia de 3dBi pero son muy grandes 9” (9” =20 cm), sería aconsejable alguna otra con menos ganancia pero de un tamaño menor como la 131 (3”=7,62 cm) que da 2 dBi y que tiene como referencia S131CL-5-RMM-2450S.

Throughput

Este otro importantísimo aspecto. Hay distintas tasas que aparecen en las hojas de características. Salta a la vista 882Kbps en el modelo AC5124 (Figura 2.1.3.3 – 5)

Pero como se ha visto esta versión (AC5124) no tiene disponible el modelo de 100 mW de PIRE. Otro aspecto es que un microcontrolador típico con un reloj de unos 8 MHz, tiene una UART que soporta hasta 115Kbps, por lo que tampoco tiene mucho sentido ir mas allá de estas velocidades si no se va a trabajar con microcontroladores más potentes que trabajen a mayor velocidad.

La versión AC4424-100 permite una velocidad en el interfaz RS-232 de hasta 288Kbps como se aprecia en la figura 2.1.3.3 – 7.

GENERAL						
Interface	20 pin mini-connector					
Serial Interface Data Rate	PC baud rates from 110 bps to 288,000 bps					
Power Consumption (typical)		Duty Cycle (TX=Transmit; RX=Receive)				
		10%TX	50%TX	100%TX	100%RX	Pwr-Down
	AC4424-10:	90mA	115mA	140mA	85mA	15mA
	AC4424-100:	100mA	160mA	235mA	85mA	15mA
	AC4424-200:	115mA	235mA	385mA	85mA	15mA
Channels (used to create independent networks)	4 channel sets consisting of 16 channels each					
Security	One byte System ID					

Figura 2.1.3.3 – 7: Consumo y velocidad interfaz para AC4424.

Más importante que la velocidad de transmisión es el esquema con que se trabaja, dúplex, semiduplex, tiempos de conmutación, etc. En las hojas de especificaciones del AC4424 se dan estos valores de throughput:

Table 4 – Maximum Overall System Throughputs

RF Mode	Interface Baud Rate	Duplex	FEC	Direction	Throughput (bps)
Stream	192k	Half	Disabled	One way	192k
Stream	192k	Half	Enabled	One way	64k
Acknowledge	115,200	Half	Disabled	One way	80k
Acknowledge	115,200	Full	Disabled	Both ways	40k

Tabla 2.1.3.3 – 2: Throughput AC4424.

Hay que decir que este dispositivo realmente no trabaja en dúplex, solo reparte frecuencias para los dos sentidos de transmisión, con lo que se evitan colisiones en una misma frecuencia.

En el caso de full-dúplex a 115,2Kbps en el interfaz de sistema se tienen 40 Kbps para cada sentido de la comunicación con lo que se pueden transmitir 400 bits cada ms que viene a ser 4 veces mas de lo que se necesita. Por lo que desde el punto de vista del throughput la versión AC4424 también es válida.

Consumo

El consumo del AC4424 está dentro de lo permitido según se aprecia en la Figura 2.1.3.3 – 7.

En la aplicación se tiene un consumo de 160 mA. La batería será aproximadamente 1800 mAH. La cuarta parte sería 450 mAH. Suponiendo que todo el conjunto AC4424 más microcontrolador consume 200mA se tiene 450mAH/200mA=2.25 Horas de duración.

Rango de temperatura

Como se ha visto los módulos están disponibles en 2 rangos de temperatura comercial e industrial.

Dimensiones físicas

A continuación se aprecian las dimensiones físicas del módulo en pulgadas, que se resumen en 1.65” de ancho x 2.65” de largo x 0.20” de espesor.

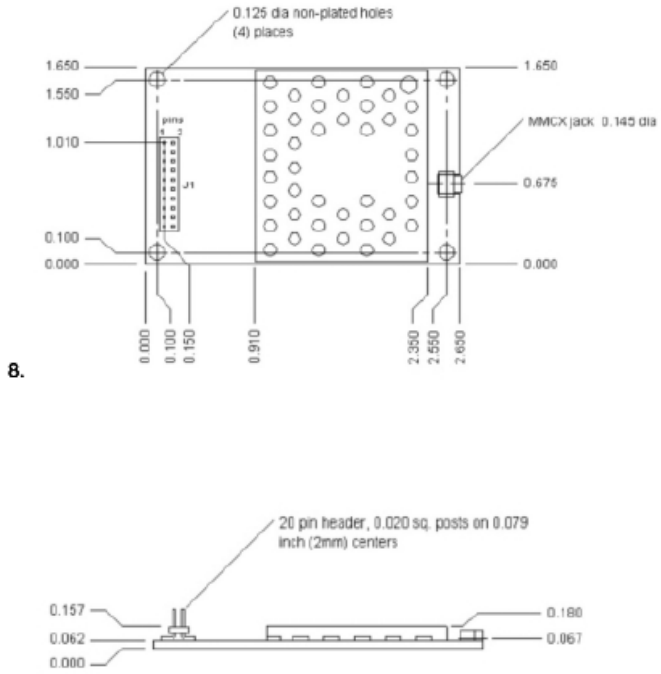


Figura 2.1.3.3 – 8: Dimensiones físicas AC4424.

2.1.3.4 B2400

El módulo B2400 es un radio modem en la banda ISM 2,4 GHz diseñado por la compañía RFSolutions. En este apartado se va a realizar un análisis de las características de este módulo similar al que se ha realizado para los módulos de Aerocomm.

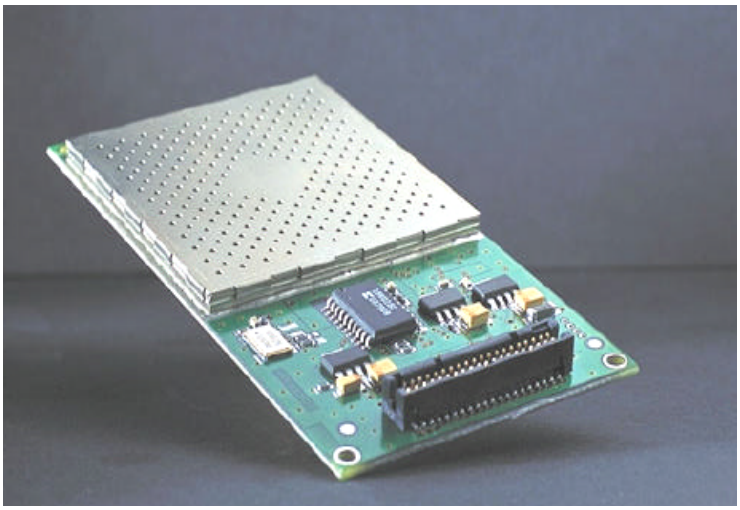


Figura 2.1.3.4 – 1: Módulo B2400 de RFSolutions.

Aspectos Radioeléctricos

La normativa española nos permite trabajar en el rango de 2400 a 2483 Mhz con una PIRE (o eirp) máxima de 100mW (20 dBm). El módulo B2400 cumple esta normativa pues trabaja en ese dicho rango de frecuencias y con una potencia máxima de 15.9 dBm.

El tipo de modulación es QPSK y emplea la técnica de espectro expandido DSSS.

Throughput

En el modo punto a punto seguro (en el que el modulo se encarga de las retransmisiones en caso de error) que presenta este módulo se permite un throughput máximo en full - duplex de 230 Kbps que cumple las especificaciones mas restrictivas del proyecto, esto es para el interfaz serie, pero este dispositivo también admite el interfaz paralelo a modo de periférico del microcontrolador (que es el que se empleara seguramente pues es difícil conseguir estas elevadas tasas de transmisión serie con los AVR mas básicos) y que debe admitir mayor velocidad.

Consumo

La tensión nominal de alimentación del módulo es de 3.6 V y el consumo nominal es de 300 mA. que esta dentro de lo admisible si bien es mucho más elevado que el de los módulos Aerocomm. También hay que tener en cuenta que en el arranque del modulo se necesitan 800 mA. a la hora del diseño.

Rango de temperatura

El rango de temperatura que presenta este módulo es pequeño de 10º a 50º C.

Dimensiones físicas

El modulo es compacto y de dimensiones reducidas, como se aprecia en la figura dada por el fabricante:

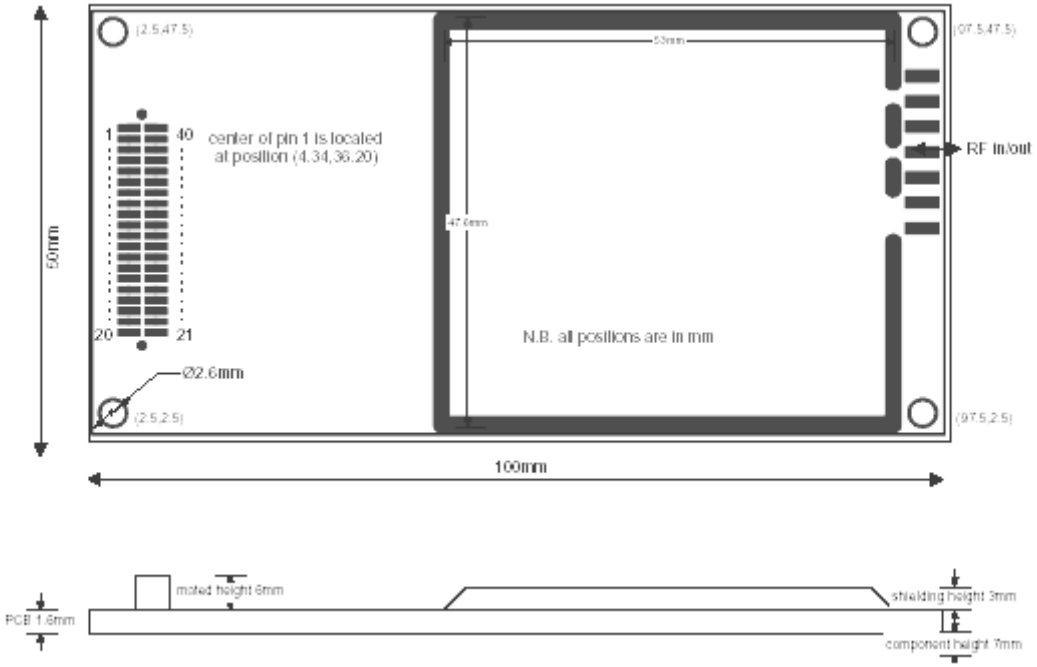


Figura 2.1.3.4 – 2: Dimensiones físicas B2400.

2.1.3.5 BRM01

El módulo BRM01 del fabricante RFSolutions es un módulo basado en el estándar Bluetooth. Trabaja en la banda de 2.4 GHz empleando FHSS. Permite dos modos de funcionamiento uno básico en el que el sólo se configura y actúa como pasarela transparente y otro modo que permite mejores prestaciones en el que se tiene mas acceso a distintas propiedades de la comunicación. En la Figura 2.1.3.5 – 1 se aprecia su reducido tamaño, y también se observa que ya viene incluida la antena.

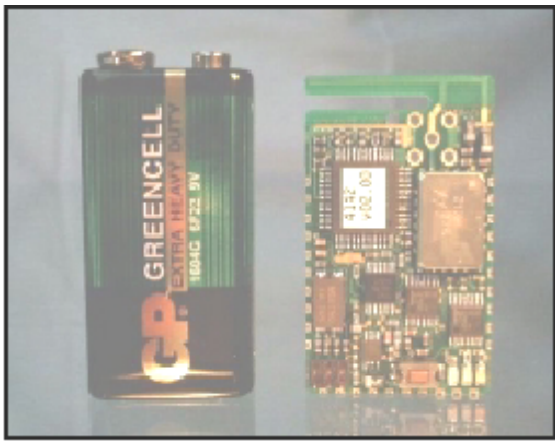


Figura 2.1.3.5 – 1: Módulo BRM01 de RFSolutions

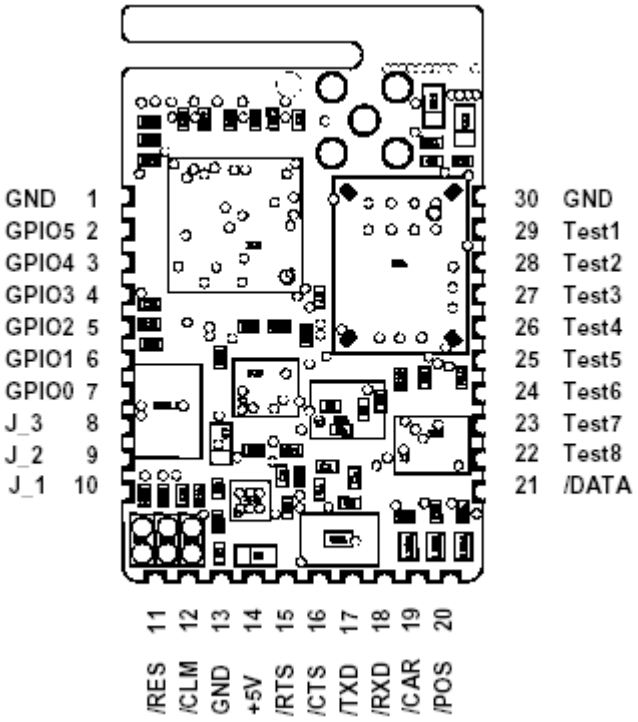


Figura 2.1.3.5 – 2: Interfaz físico del módulo BRM01.

Aspectos Radioeléctricos.

El módulo trabaja en la banda ISM 2.4 GHz, dentro del rango de 2400 MHz a 2480 MHz. que está dentro del rango que permite la legislación. En cuanto a la potencia de salida está justo en el límite 100 mW (20 dBm). El ancho de banda total de 79 MHz se divide en 79 canales de 1 MHz. Se emplea la técnica de espectro expandido FHSS realizándose 1600 saltos por segundo, lo que permite una alta inmunidad ante interferencias causadas por multitrayecto u otros sistemas. Concretamente estas características cumplen exigencias de la clase 1 del estándar Bluetooth.

Throughput

En las hojas de datos no se dan datos de throughput aunque si la velocidad física del interfaz serie que es 230 Kbps.

Consumo

El consumo es similar al del módulo B2400 RFSolutions unos 250 mA., que es bastante elevado aunque está dentro lo admisible.

Dimensiones físicas

El módulo es compacto y de dimensiones reducidas y además incluye la antena integrada en el PCB del propio módulo.

Technical Specifications

Physical	
Size:	45mm x 28mm x 3mm with internal antenna 40mm x 28mm x 3mm without internal antenna
Connections:	Direct to PCB via solder wells on 2.54mm pitch (SMT) Optional SMA, SMB, SMC or MCX RF connector
Radio	
Maximum output power:	100mW (+20dBm)
RF Frequency range:	2.400Ghz to 2.480Ghz
RF Channels:	79, 1MHz channel spacing
Frequency Hopping:	1600 hops/sec (625uS dwell time)
Over Air Data Rate:	1Mb/s
Achievable Range:	200m Line-of-sight using internal antenna
Interfacing	
Power Supply:	Voltage: 4V to 5.5V, Supply Current: <250mA peak (@ max TX o/p power), 20mA Idle
Serial Interface:	CMOS logic or optional RS232 levels
Control Lines:	CMOS compatible
Antenna:	Integrated F type or approved external (via optional SMA RF connector)

Figura 2.1.3.5 – 3: Características del módulo BRM01.

2.1.3.6 F2M01C1

El módulo F2M01C1 de la compañía sueca Free2move es otro módulo de clase 1 basado en el estándar Bluetooth. Este módulo viene encapsulado y presenta un interfaz serie RS - 232 con un conector DB9.

- No need for external drivers
- External power can be supplied via D-SUB connector or via DC connector
- Implements the Bluetooth Serial Port Profile and Generic Access Profile
- Interoperability with PDA, laptops etc.
- Nominal 100m range
- Configurable for use of different baud rates and serial settings

Figura 2.1.3.6 –1: Módulo F2M01C1 de Free2Move.

Aspectos radioeléctricos

Al igual que módulo BRM01 este módulo es un módulo que cumple el estándar Bluetooth, trabaja en la banda ISM de 2.4 GHz, dentro del rango de 2400Mhz a 2483 MHz, la potencia es de 16 dBm aproximadamente que es menor que el limite superior de 20 dBm que marca la legislación española al respecto. Un dato interesante que adjunta el fabricante las hojas de datos técnicos es el patrón de radiación que nos da una idea la direccionalidad que presentan estos dispositivos en el que se aprecia como es presenta un comportamiento mas o menos homogéneo en todas direcciones y no presenta picos abruptos en ninguna dirección.

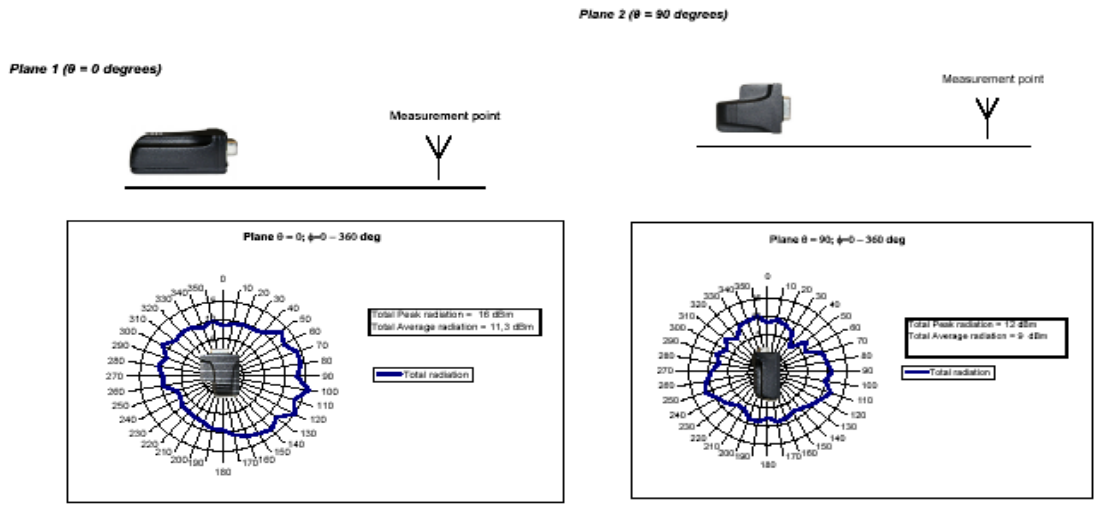


Figura 2.1.3.6 – 2: Patrón de radiación

Consumo

El consumo es inferior el del módulo BRM01 como es previsible pues la potencia radiada es inferior.

Role	Mode	Average (mA)	Max (mA)
Endpoint (slave)	Data transfer 115,200 bit/s	61	164
Endpoint (slave)	Data transfer 9,600 bit/s	46	166
Endpoint (slave)	No data transfer, established connection	45	166
Endpoint (slave)	No connection	69	70
Connecting (master)	Data transfer 115,200 bit/s	55	164
Connecting (master)	Data transfer 9,600 bit/s	30	166
Connecting (master)	No data transfer, established connection	29	166
Connecting (master)	No connection	85	168

Figura 2.1.3.6 – 3: Consumo del módulo Free2Move.

Rango de temperaturas

El rango de temperaturas que permite este dispositivo es de 0º a 70º. Este es un rango comercial pero admisible para esta aplicación pues no es previsible el trabajo en entornos extremos.

Dimensiones Físicas

Las dimensiones físicas son reducidas y también incluyen la antena, concretamente son 47x33x19 mm.

Otros aspectos

El módulo trae un SW para facilitar la configuración del dispositivo. Esto junto con el hecho de ser fácilmente insertado extraído en la tarjeta a través del conector DB9 permite reutilizar este dispositivo fácilmente para otros posibles proyectos.

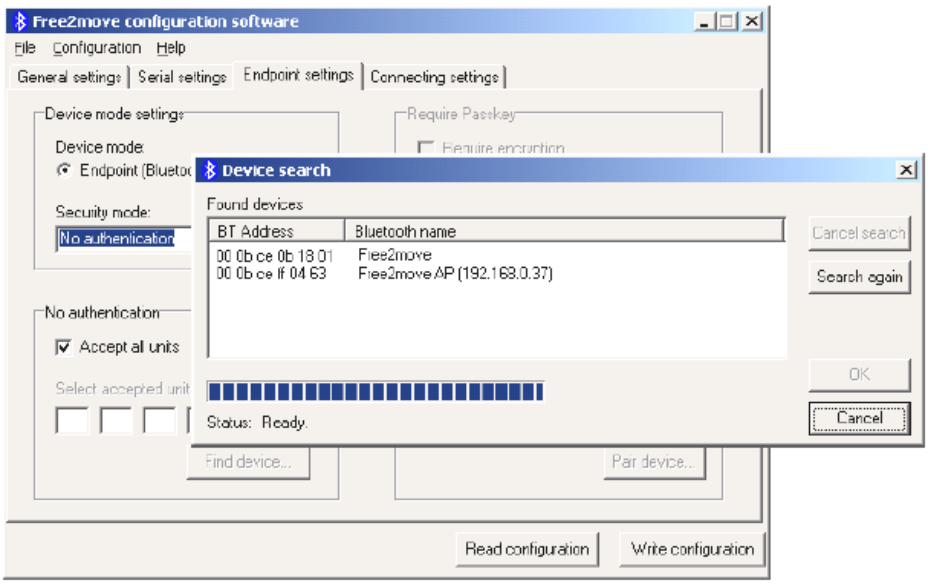


Figura 2.1.3.7 – 4: Software de configuración.

2.1.3.7 WLAN

Otra posible solución es usar tarjetas de red inalámbricas. Estas tarjetas suelen presentar un interfaz PCI o PCMCIA para la conexión a ordenadores personales, pero conectar este tipo de tarjetas a un sistema basado en un microcontrolador *host* es bastante complicado. En la figura 2.1.3.7 –1 se muestra un kit de desarrollo que se parecería a la tarjeta a implementar, en el caso de este kit de desarrollo se tiene un módulo wireless con interfaz PCMCIA, un controlador Realtek de Ethernet y un microcontrolador. Como se estudia mas adelante el bus de campo que se va a emplear es el estándar CAN por lo que no interesa este kit desarrollo, teniendo en cuenta el elevado coste del kit de desarrollo.

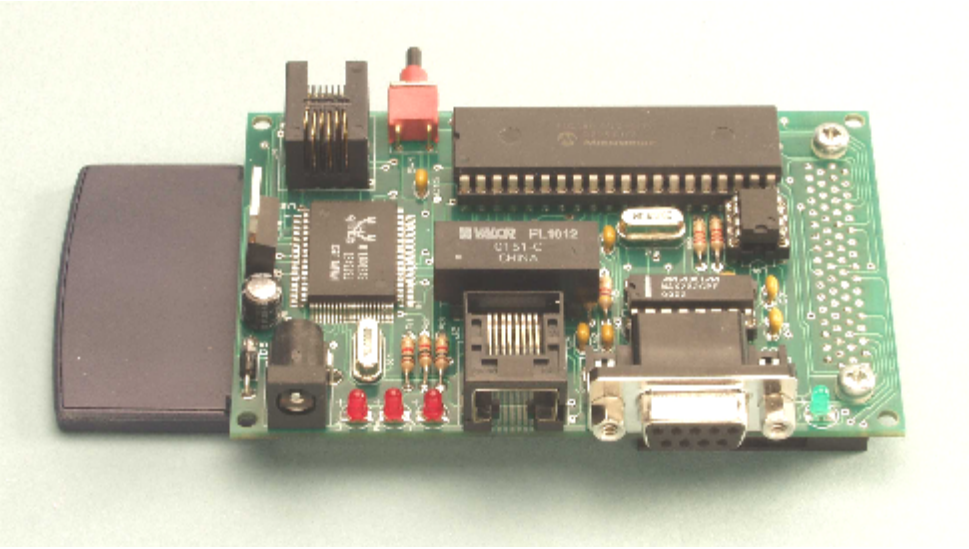


Figura 2.1.3.7 -1

2.1.3.8 Justificación de la elección del módulo de radio.

En este punto se valoran los diferentes módulos de radiofrecuencia vistos. Los módulos en las bandas ISM 433 y ISM 868 de Aurel y RadioMetrix no cumplen las especificaciones de velocidad, sería válidos solo si rebajasen las especificaciones del sistema en cuanto a tiempo de muestreo notablemente, por encima 10 ms.

Los módulos de Aerocomm y el módulo B2400 si las cumplen pero son muy caros de por sí y además se venden en kits de desarrollo, lo cual hace prohibitivo su precio.

Los módulos BRM01 y F2M01C1 cumplen aunque no con demasiada holgura las especificaciones de velocidad aunque siguen siendo caros (unos 200 € cada unidad). Inicialmente se escogió como opción el módulo BRM01 pero cuando llegó este módulo fue imposible trabajar con él pues no funcionaba correctamente, además el soporte postventa fue escaso por no decir nulo, pues se enviaron los módulos (argumentaban

que se habían desconfigurado) pero a la vuelta seguían sin funcionar. Todo esto además se extendió durante varios meses pues su respuesta antes los diferentes correos electrónicos en los que se demandaba apoyo o información técnica era tardía y vaga. Por este motivo se optó por solicitar el módulo F2M01C1 de Free2Move pese a que este es ligeramente inferior en prestaciones de velocidad y potencia.

2.2 Sección Bus de Campo

En este apartado se presentan distintas opciones para la realización del bus de campo, aunque la decisión sobre la solución a tomar corresponde en gran medida al marco general del proyecto.

2.2.1 Ethernet

En esta solución se realizarían pequeñas tarjetas de red como periféricos de los microcontroladores que llevan a cabo las funciones de control de los sistemas sensores y/o actuadores. Las características típicas de esta tecnología son las mismas que las de las redes locales convencionales. El empleo de esta tecnología se está extendiendo notablemente en el ámbito de las aplicaciones de control industrial. Las redes basadas en Ethernet han cambiado desde su uso tradicional en niveles más altos de comunicación en la planta, bajando hacia el nivel de campo. Hoy en día aproximadamente un 96% de todos las redes informáticas están basadas en Ethernet. El efecto ha sido dramático, una tarjeta de Ethernet por ejemplo cuesta 15 € y se puede encontrar en cualquier tienda de informática, mientras una tarjeta para un controlador industrial puede costar fácilmente 100 veces más, y solamente la venden unos cuantos suministradores, y a veces solo uno.

Características	**Valor Ethernet**	**Valores IEEE 802.3**					
		10Base5	**10Base2**	**1Base5**	**10BaseT**	**100BaseTX**	**100BaseT4**
Velocidad de los datos (Mbps)	10	10	10	1	10	100	10
Longitud máxima del segmento (m)	500	500	185	250	100 con pares trenzados no blindados	100 UTP o STP	1800
Soporte	Coaxial de 50-ohmios (grueso)	Coaxial de 50-ohmios (grueso)	Coaxial de 50-ohmios (fino)	Pares trenzados sin blindaje	Pares trenzados sin blindaje	Categoría 5 Pares trenzados blindados o sin blindaje	Coaxial de 75-ohmios
Topología	Bus	Bus	Bus	Estrella	Estrella	Estrella	Bus

Tabla 2.2.1 – 1: Características Ethernet y 802.3

Como se puede apreciar en la figura 2.2.1 –1 la trama de Ethernet está orientada a mensajes con mucha información, hasta 1500 octetos de datos, también la cabecera es extensa y ocupa 26 bytes. Esto es excesivo puede cada tarjeta del sistema que se desea implementar envía o recibe un dato por mensaje con lo cual los datos serían como mucho cuatro bytes dos para el dato y uno para etiquetar la señal y otro para posibles labores de control. De esta manera para una trama de 30 bytes en total 4 bytes serían de información útil y 26 de cabecera (sin contar el relleno para tener la mínima longitud de datos que son 46 bytes) con lo que se aprecia claramente una reducción del caudal útil, aunque en parte se compensaría con la elevada velocidad física de Ethernet. El protocolo Ethernet es mas apropiado para enviar tramas con mas información como por ejemplo toda la información de un sistema de campo hacia un sistema Scada.

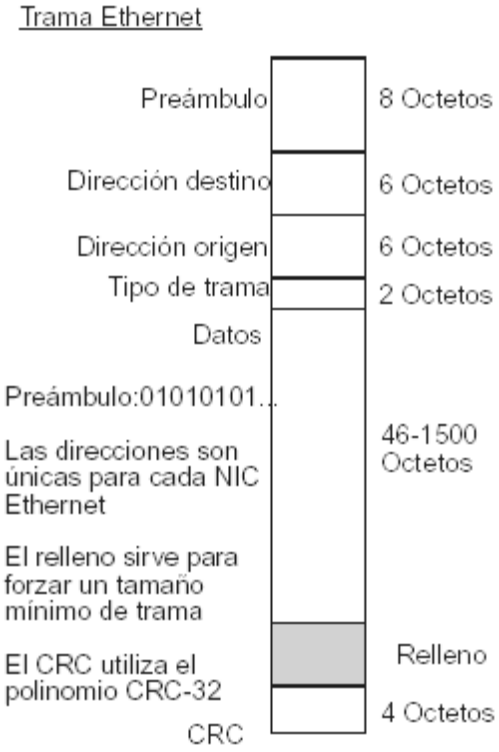


Figura 2.2.1 –1: Trama Ethernet.

Por otro lado las implementaciones más sencillas y baratas se basan en el empleo de sistemas 10/100 Base T /TX que se basan en topologías de en estrella sobre par trenzado en las que es necesario el uso de concentradores o 'hubs' para realizar la conmutación, por lo que habría que disponer de uno de estos concentradores o pensar en una topología en anillo tipo 'token ring' que supondría que los controladores *host* de cada tarjeta asociada a un sensor o actuador deberían ser potentes para manejar este flujo de información y harían falta microcontroladores mas potentes.

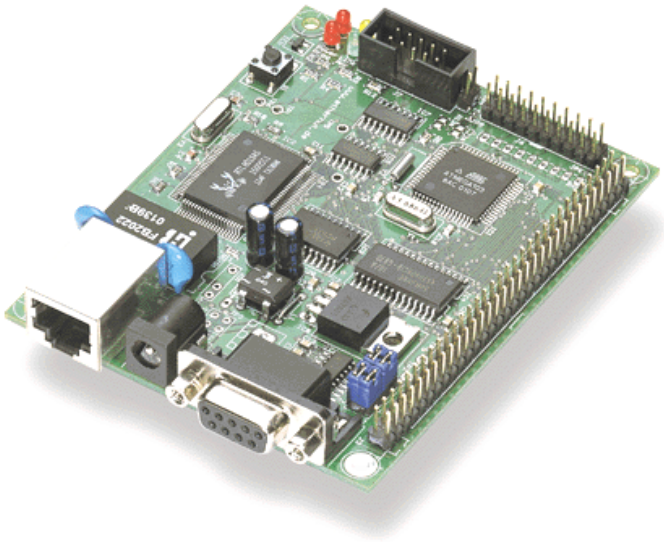


Figura 2.2.1 – 2: Kit Ethernet

El kit de la figura 2.2.1 – 2 es un servidor web y da una idea de cómo sería la tarjeta a realizar. Junto al conector RJ-45 se aprecia el circuito integrado de aislamiento eléctrico y el controlador de red Realtek muy empleado en tarjetas de red convencionales.

2.2.2 Estándar de aviónica MIL-STD-1553B

2.2.2.1 Origen.

MIL-STD-1553B es un estándar militar que define las características eléctricas y el protocolo de acceso a un bus de datos.

En 1968 la Society of Automotive Engineers (SAE), un cuerpo técnico formado por miembros de la industria y del ámbito militar estadounidense, estableció un subcomité para definir un bus de datos serie que cumpliese con las necesidades la comunidad de aviónica militar. Este comité se conocía como A2-K y desarrolló su primer borrador del documento en 1970. Tres años de revisiones por parte del gobierno y ejercito estadounidense llevaron a la liberación de MIL-STD-1553 (USAF) en Agosto de 1973. El primer empleo del estándar fue en el caza de combate F-16.

Se realizaron posteriores cambios y mejoras y una versión tri-servicio, MIL-STD-1553A se liberó en 1975. Los primeros usuarios de la versión A del estándar fueron el F-16 y el helicóptero AH-64A Apache. Con la experiencia real, se descubrió pronto que se necesitaban mas especificaciones y capacidades adicionales. La SAE empleó tres años de esfuerzos concentrados para producir MIL-STD-1553B, que se liberó en 1978. En este punto el gobierno estadounidense decidió 'congelar' el estándar en el nivel B para permitir a los fabricantes de componentes el desarrollo de productos y

permitir a la industria ganar experiencia real antes de determinar el siguiente conjunto de cambios a realizar.

Actualmente el estándar MIL-STD-1553 sigue en el nivel B, sin embargo se han introducido cambios reflejados en las Notas 1 y 2 para aplicaciones más restrictivas. La Nota 1 está pensada para aplicaciones de la Fuerza Aérea aunque una gran parte de la industria encontró que la nota 1 era muy restrictiva y limitaba las capacidades de aplicación del estándar. La SAE empleó otros tres años para desarrollar la Nota 2 que fue liberada en 1986 y que reemplaza la Nota 1. La Nota 2 reemplaza a la Nota 1 y describe con mas detalle las opciones dentro del estándar, no restringe opciones de uso pero si dice como se debe usar si se quiere implementar.

2.2.2.2 Aplicaciones

El estándar se ha aplicado a satélites, carga útil dentro de la lanzadera espacial incluso en la ISS (Estación Espacial Internacional). Aunque sus aplicaciones militares son las más numerosas y de mayor alcance. Se ha empleado transporte pesado, aviones cisterna, bombarderos, cazas tácticos y helicópteros. También se emplea en mísiles y en algunos casos se usa como interfaz primario entre la aeronave y el misil. La marina estadounidense ha empleado este bus tanto aplicaciones sobre como bajo la superficie. La armada estadounidense además de emplearlo en sus helicópteros también lo ha aplicado en tanques. Entre las aplicaciones comerciales se encuentra el metro urbano y líneas de producción.

El estándar ha sido aceptado por la OTAN y distintos gobiernos como el Reino Unido. La amplia aceptación y aplicación de MIL-STD-1553B ha fomentado el desarrollo de otros esfuerzos de estandarización como MIL-STD-1773 que es una versión para fibra óptica.

2.2.2.3 Características.

Las características del bus se recogen en la tabla 2.2.2.3 – 1. se observa como estas características se adecuan mas a las necesidades de un bus de instrumentación. La velocidad de trabajo es de 1MHz. Existen distintos formatos de mensajes según las necesidades de la comunicación pueden ser mensajes de datos de controlador a terminal o viceversa o incluso entre terminales, mensajes de control del sistema, y mensajes de multidifusión.

Data Rate	1 MHz
Word Length	20 bits
Data Bits / Word	16 bits
Message Length	Maximum of 32 data words
Transmission Technique	Half-duplex
Operation	Asynchronous
Encoding	Manchester II bi-phase
Protocol	Command/response
Bus Control	Single or Multiple
Fault Tolerance	Typically Dual Redundant, second bus in "Hot Backup" status
Message Formats	Controller to terminal Terminal to controller Terminal to terminal Broadcast System control
Number of Remote Terminals	Maximum of 31
Terminal Types	Remote terminal Bus controller Bus monitor
Transmission Media	Twisted shielded pair
Coupling	Transformer and direct

Tabla 2.2.2.3 – 1: Características bus MIL-STD-1553B.

Cable Type	Twisted Shielded Pair
Capacitance	30.0 pF/ft. max, wire to wire
Characteristic Impedance	70.0 to 85.0 Ohms at 1 MHz
Cable Attenuation	1.5 dB/100 ft. max, at 1MHz
Cable Twists	4 Twists per ft., minimum
Shield Coverage	90% min (Notice 2)
Cable Termination	Cable impedance (+/- 2%)
Direct Coupled Stub Length	Maximum of 1 foot
XFMR Coupled Stub Length	Maximum of 20 feet

Tabla 2.2.2.3 – 2: Características del medio de transmisión.

Se dispone de diferentes formatos de mensajes según la función. Los mensajes se componen de palabras de 20 bits con 16 bits de información, los tres primeros bits se usan para sincronismo y el último es el bit de paridad. Las palabras pueden ser de control, estado o datos, según el uso que hagan de los 16 bits de carga útil (ver figura 2.2.2.3 –1).

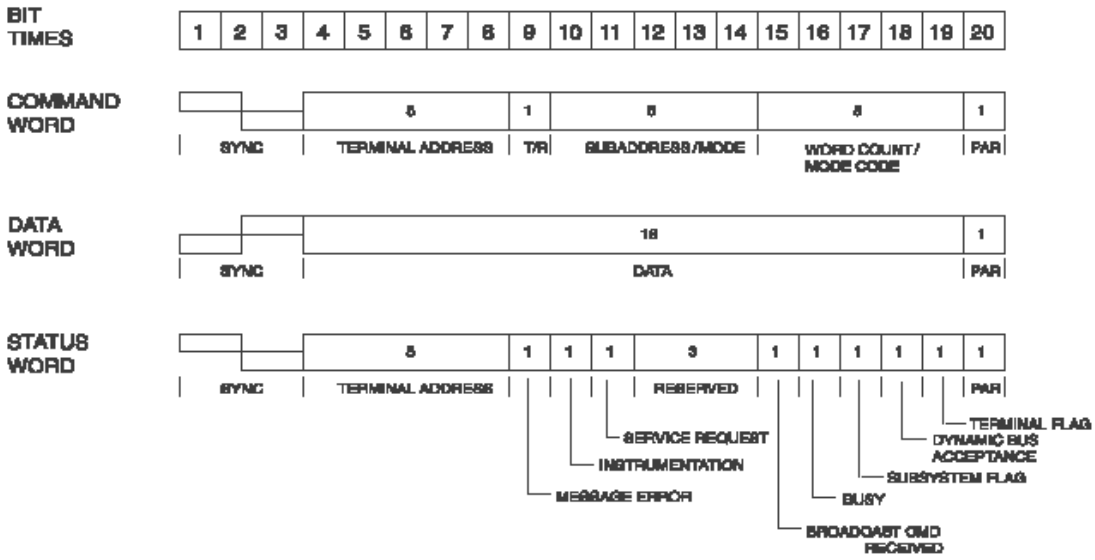


Figura 2.2.2.3 – 1: Tipos de palabras.

Para construir los mensajes se necesitan 1 ó 2 palabras de control, las palabras de datos y una posible palabra de estado. Por lo tanto suponiendo una palabra de control, una de datos y otra de estado se tiene una carga solo aproximadamente 1/3 del mensaje es información útil aunque este valor es mayor del que se obtenía para Ethernet que era aproximadamente 4/30 del mensaje (4 bytes de datos de 30 bytes para el mensaje total).

Otro aspecto a tener en cuenta es el cableado del bus que es similar y al que se emplea para cablear cables coaxiales. Se emplean acopladores, segmentos en *stub*, impedancias de terminación, el tamaño de estos elementos es similar al de los empleados en cableado de redes coaxiales, que puede ser excesivo para los elementos a interconectar en el helicóptero de radio control.

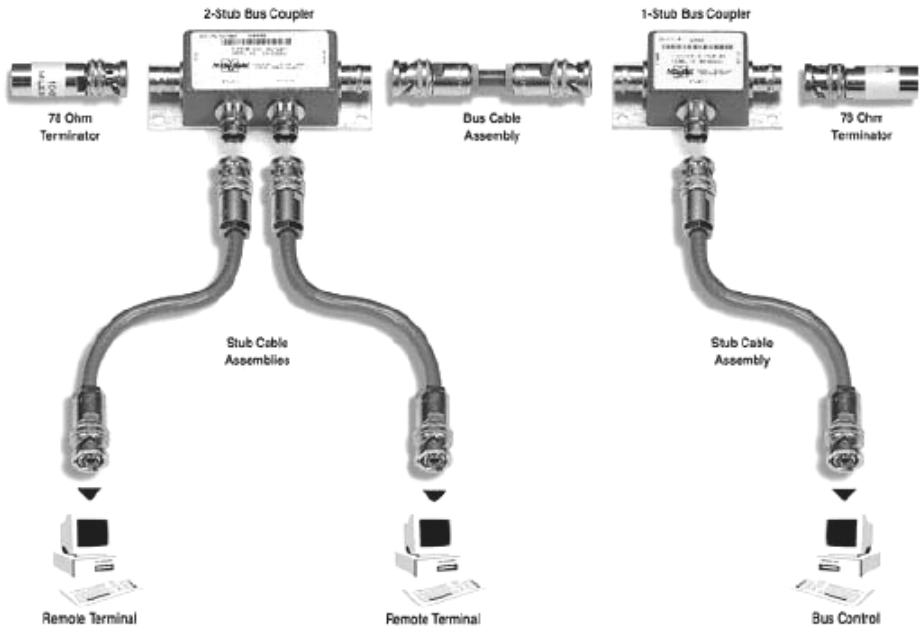


Figura 2.2.2.3 – 2: Cableado del Bus.

Por último hay que considerar que este estándar no está tan difundido como Ethernet o el estándar CAN.

2.2.3 Estándar CAN.

El estándar CAN (Controller Area Network) se presentó en la SAE en 1986. Actualmente casi todos los automóviles tipo turismo manufacturados en Europa vienen equipados con al menos una red CAN. También se emplea en otros medio de transporte desde trenes a barcos y aplicaciones de control de la industria. El estándar CAN es uno de los más extendidos, en 1999 se vendieron 60 millones de controladores reales y en el 2000 mas de 100 millones.

El bus CAN es un sistema de bus serie diferencial sobre 2 hilos. El medio físico consiste en un par trenzado apantallado o no. El bus CAN opera en entornos eléctricamente ruidosos con un alto grado de integridad de los datos. El bus CAN permite una velocidad de 1 Mbit/segundo sobre distancias cortas (menores de 40 m) y bajas velocidades 5Kbps sobre distancias largas de hasta 10 Km. El controlador CAN implementa físicamente mecanismos de detección y manejo de errores.

Se llega a la conclusión que la solución más interesante es la basada en el estándar CAN pues ofrece mejores resultados para tramas cortas en las que sólo se quiere un dato por mensaje que el estándar Ethernet y mas barato debido a su mayor extensión de mercado que el estándar MIL-STD-1553B.

Figura 2.2.3 – 1: Trama a nivel físico del bus CAN.

En cuanto a la estructura de la trama se tiene un bit de comienzo de trama (SOF), un identificador que se emplea para labores de direccionamiento, bit RTR (solicitud de respuesta), bit IDE(tipo de trama), 2 bits reservados, campo DLC (codifica la longitud de los siguientes datos hasta 8 bytes), los datos, el CRC, el bit ACK y el campo EOF, y un bit para separar tramas consecutivas. Como el identificador se puede aprovechar para codificar la etiqueta del dato se tiene como cabecera extra 1 byte formado por SOF, RTR, IDE, DLC y los bits reservados, 2 bytes de CRC y ACK, EOF e IFS se pueden aproximar por otro byte. En total se tienen 4 bytes de cabecera y se estiman 2 bytes de datos para codificar la señal mas el identificador que se puede aproximar por otro byte. De esta manera se tiene una trama de 7 bytes en total de los cuales 3 bytes se usan para carga útil resulta una mejor relación que en el caso del estándar Ethernet y del estándar MIL-STD-1553B.

Se llega a la conclusión que la solución más interesante es la basada en el estándar CAN pues ofrece mejores resultados para tramas cortas en las que solo se quiere un dato por mensaje que el estándar Ethernet y mas barato debido a su mayor extensión de mercado que el estándar MIL-STD-1553B.

En el capítulo 3 se hace una introducción a este estándar y se analízale funcionamiento dentro del marco del proyecto.

2.3 Microcontrolador.

En cuanto al microcontrolador usado se plantea en las especificaciones del proyecto general el empleo de microcontroladores Atmel AVR para las diferentes tarjetas que componen el montaje final. Si bien esto no es obligatorio si es recomendable para facilitar la labor de tutorización de cada proyecto y reducir el gasto en herramientas de desarrollo de software y hardware. Aunque las especificaciones de cada uno de los proyectos son distintas, la amplia gama de los microcontroladores AVR ofrecen cobertura para los posibles requerimientos de cada una de las tarjetas.

[Show all]	Flash,kB	EEPROM,B	SRAM,B	XRAM interface	I/O	Interrupts	Interrupts Ext.	PinChange	TWI (I2C)	SPI	UART	USART	USI	Timers 8bit	Timers 16bit	PWM,ch	WDT	RTC	AnaComp	ADC 10bit,ch	Int. Osc.	BOD	JTAG prog.	ISP	Self-prog.	Volts,V	Speed,MHz	PDIP,pins	MLF,pins
ATtiny26	2	128	128		16	11	1	Y					Y	2		2	Y		Y	11	Y	Y		Y		4.5-5.5	0-16	20	32
ATtiny12L	1	64			6	5	1	Y						1			Y		Y		Y	Y		Y		2.7-5.5	0-4	8	
ATtiny12	1	64			6	5	1	Y						1			Y		Y		Y	Y		Y		4-5.5	0-8	8	
ATtiny11L	1				6	4	1	Y						1			Y		Y		Y					2.7-5.5	0-2	8	
ATtiny11	1				6	4	1	Y						1			Y		Y		Y					4-5.5	0-6	8	
ATmega8515L	8	512	512	Y	35	16	3			Y		1		1	1	4	Y		Y		Y	Y		Y	Y	2.7-5.5	0-8	40	44
ATmega8515	8	512	512	Y	35	16	3			Y		1		1	1	4	Y		Y		Y	Y		Y	Y	4.5-5.5	0-16	40	44
ATmega8L	8	512	1024		23	18	2		Y	Y		1		2	1	3	Y	Y	Y	8	Y	Y		Y	Y	2.7-5.5	0-8	28	32
ATmega8	8	512	1024		23	18	2		Y	Y		1		2	1	3	Y	Y	Y	8	Y	Y		Y	Y	4.5-5.5	0-16	28	32
ATmega32L	32	1024	2048		32	20	3		Y	Y		1		2	1	4	Y	Y	Y	8	Y	Y	Y	Y	Y	2.7-5.5	0-8	40	44
ATmega32	32	1024	2048		32	20	3		Y	Y		1		2	1	4	Y	Y	Y	8	Y	Y	Y	Y	Y	4.5-5.5	0-16	40	44
ATmega16L	16	512	1024		32	20	2		Y	Y		1		2	1	4	Y	Y	Y	8	Y	Y	Y	Y	Y	2.7-5.5	0-8	40	44
ATmega16	16	512	1024		32	20	2		Y	Y		1		2	1	4	Y	Y	Y	8	Y	Y	Y	Y	Y	4.5-5.5	0-16	40	44
ATmega128L	128	4096	4096	Y	53	29	8		Y	Y		2		2	2	6+2	Y	Y	Y	8	Y	Y	Y	Y	Y	2.7-5.5	0-8		
ATmega128	128	4096	4096	Y	53	29	8		Y	Y		2		2	2	6+2	Y	Y	Y	8	Y	Y	Y	Y	Y	4.5-5.5	0-16		
AT90S8535	8	512	512		32	16	2			Y	1			2	1	3	Y	Y	Y	8				Y		4-6	0-8	40	
AT90LS8535	8	512	512		32	16	2			Y	1			2	1	3	Y	Y	Y	8				Y		2.7-6.0	0-4	40	
AT90C8534	8	512	256		7	6	2							1	1					6						3.3-6	0-1.5		
AT90S8515-8	8	512	512	Y	32	12	2			Y	1			1	1	2	Y		Y					Y		4-6	0-8	40	
AT90S8515-4	8	512	512	Y	32	12	2			Y	1			1	1	2	Y		Y					Y		2.7-6.0	0-4	40	
AT90LS4433	4	256	128		20	13	2			Y	1			1	1	1	Y		Y	6		Y		Y		2.7-6.0	0-4	28	
AT90S4433	4	256	128		20	13	2			Y	1			1	1	1	Y		Y	6		Y		Y		4-6	0-8	28	
AT90LS2343	2	128	128		5	2	1							1			Y				Y			Y		2.7-6.0	0-4	8	
AT90S2343	2	128	128		5	2	1							1			Y				Y			Y		4-6	0-10	8	
AT90S2323	2	128	128		3	2	1							1			Y							Y		4-6	0-10	20	
AT90LS2323	2	128	128		3	2	1							1			Y							Y		2.7-6.0	0-4	8	
AT90S2313-4	2	128	128		15	10	2				1			1	1	1	Y		Y					Y		2.7-6.0	0-4	20	
AT90S2313-10	2	128	128		15	10	2				1			1	1	1	Y		Y					Y		4-6	0-10	20	
AT90S1200-12	1	64			15	3	1							1			Y		Y		Y			Y		4-6	0-12	20	
AT90S1200-4	1	64			15	3	1							1			Y		Y		Y			Y		2.7-6.0	0-4	20	

Figura 2.3 – 1: Algunos de los microcontroladores AVR.

Los microcontroladores AVR son de tecnología CMOS de 8 bits con bajo consumo basado en la arquitectura RISC. Permiten un poderoso juego de instrucciones de un ciclo, la mayoría de las cuales se ejecutan en un solo ciclo de reloj, por este motivo se consigue una capacidad de procesamiento cercana a 1MIPS por MHz permitiendo al diseñador de un sistema optimizar el consumo gracias a la gran velocidad de procesamiento del microcontrolador. La familia AVR utiliza el concepto de la arquitectura Harvard con buses y memorias separadas para los datos y para el programa. Este concepto permite que las instrucciones sencillas sean ejecutadas en un ciclo de reloj. La memoria de programa es Flash.

Según cada modelo incorporan diferentes tamaños de memoria flash, RAM y EEPROM. También es posible usar RAM externa usando un bus de datos / direcciones multiplexado más las señales típicas de control en modo Intel (ALE,RD,WR,CS).La memoria flash permite programar los dispositivos incluso una vez montados en la tarjeta final (ISP - In System Programming) sin necesidad de proveer niveles especiales de tensión. La memoria EEPROM permite el almacenamiento de datos que se quieran conservar incluso cuando se apague el dispositivo, por ejemplo para guardar configuraciones.

La tecnología AVR combina un numeroso juego de instrucciones con 32 registros de propósito general. Los 32 registros están conectados directamente a la unidad aritmético lógica (ALU), permitiendo que dos registros independientes sean accesibles mediante una instrucción simple ejecutada en un solo ciclo de instrucción. La arquitectura resultante es muy eficiente permitiendo una capacidad de procesamiento más de 10 veces superior a los microcontroladores CISC convencionales.

Las frecuencias de trabajo de la familia AVR permiten hasta 16 MHz. Suele haber dos versiones para cada modelo. La versión L trabaja con una tensión de alimentación 2.7 V y permite menores velocidades de trabajo típicamente hasta 4 u 8 MHz, la versión superior trabaja con alimentación dentro del rango 4.0 – 5.5 V y trabaja hasta 8 o 16 MHz.

Caracteríticas típicas de la familia AVR:

- Timer / contadores de 8 y 16 bits flexibles con modos de comparación.
- Interrupciones internas y externas.
- UART serie programable.
- USART serie programable.
- Puerto serie SPI.
- RTC.
- TWI(I2C).
- Líneas de entrada /salida digitales configurables por SW.
- Watchdog Timer programable con oscilador interno.
- Detector de Brown-Out.
- Comparador Analógico.
- Conversor Analógico Digital.
- Interfaz JTAG.

El modo “Idle Mode” detiene la CPU mientras permite que la SRAM, los contadores / Timer, el puerto SPI y el sistema de interrupciones continúe funcionando. El modo de bajo consumo guarda el contenido de los registros pero detiene el oscilador, deshabilitando todas las funciones del chip hasta que se produzca una interrupción o un reset.

La memoria Flash on-chip permite que la memoria de programa del chip sea reprogramada en el propio sistema a través del interfaz SPI o mediante un programador de memoria convencional. Combinando una tecnología RISC de 8 bits con una CPU con memoria Flash, los microcontroladores AVR de Atmel proporcionan una alta flexibilidad en los diseños a bajo coste aportando una solución muy efectiva para la mayoría de las aplicaciones de control.

La familia AVR está apoyada por un completo juego de programas y sistemas de desarrollo incluyendo: compiladores C, ensambladores, simuladores, emuladores en circuito, y kits de evaluación. En concreto se va a trabajar con el entorno de trabajo *ImageCraft Compiler for Atmel AVR Microcontrollers*.

3. Configuración de la sección CAN

3.1 Introducción al estándar CAN

Se va a emplear un bus de campo tipo CAN muy extendido en aplicaciones de automoción. CAN es un estándar ISO (ISO 11898) para comunicaciones serie que fue desarrollado por BOSCH en 1980 para aplicaciones de automoción. Actualmente se ha extendido el empleo del estándar CAN a automatización industrial. Aún así en este punto se procede a una introducción genérica al estándar CAN (Controller Area Network) y posteriormente se detallará la configuración concreta utilizada en este proyecto.

El tipo de bus de campo a emplear en este proyecto viene fijado por las especificaciones del marco general de este proyecto. El principal requisito que se le exige al bus es que sea capaz de soportar el régimen binario que introduzca el sistema. El máximo típico que permite el bus CAN es de 1 Mbit/s

El estándar CAN incluye:

- La capa física.
- La capa de enlace:
    - Formatos de mensajes.
    - Reglas de arbitraje para el acceso al bus.
    - Métodos para detección de errores y confinamiento de errores.

Algunos aspectos destacables del estándar CAN:

- El medio físico de transmisión es muy simple un par de cable trenzado o incluso con un solo hilo. No presenta inconvenientes para otros medios como ópticos o vía radio.
- CAN es un estándar consolidado y existen numerosos productos y herramientas.
- El protocolo CAN es implementado vía hardware lo que combina la detección y confinamiento de errores con una elevada velocidad de transmisión. El controlador CAN solo admite mensajes de nodos con direcciones predeterminadas liberando al microcontrolador de procesar mensajes que no le interesan para el desempeño de sus funciones.
- Se basa en un bus serie asíncrono, por lo que no hay necesidad de transferir señal de reloj alguna.

- No necesita de ningún nodo central que desempeñe funciones de direccionamiento. Cada mensaje lleva un identificador que identifica contenidos y prioridad. Cuanto menor sea este identificador mayor prioridad presenta este mensaje.

- Presenta un arbitraje de acceso por CSMA y detección de colisión.

- Orientado a multi-master y broadcast.

- **Excelente relación prestaciones / precio**.

Modelo de Referencia OSI.

El estándar CAN comprende las capas 1 y 2 del modelo OSI, es decir la capa física y la capa de nivel de enlace. Algunas extensiones del estándar incluyen funciones de las capas superiores.

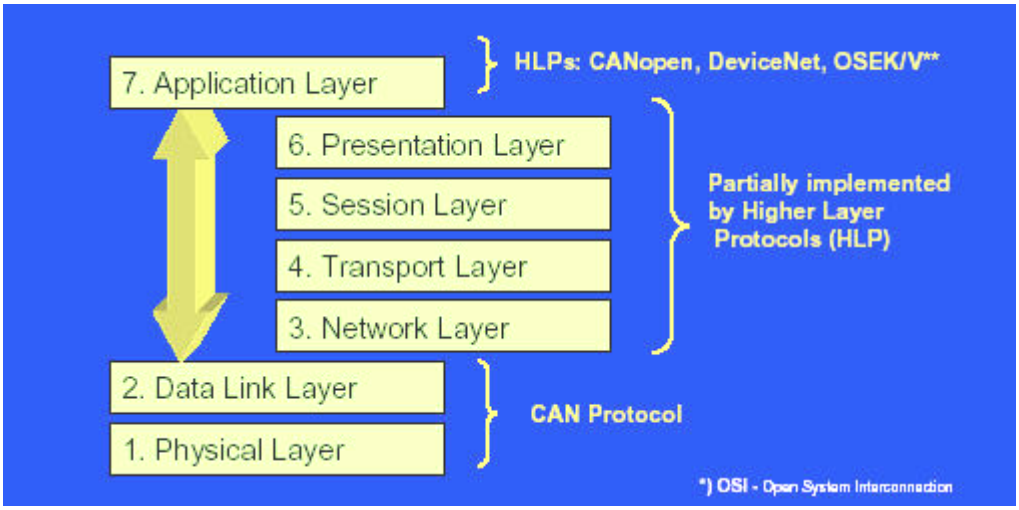


Figura 3.1 –1: Modelo OSI.

Formato de Tramas.

Las tramas de información que presenta el protocolo CAN por encima de la capa de enlace (normalmente directamente el nivel de aplicación) son las CAN - V2.0A y CAN - V2.0B de la figura siguiente. El microcontrolador *host* introduce el identificador (y su tipo, normal o extendido) , la longitud de los datos y los propios datos. El resto de campos los introduce el propio controlador CAN. La diferencia entre CAN - V2.0A y CAN - V2.0B es el tamaño del identificador 11 bits en CAN - V2.0A y 29 en CAN - V2.0B y consecuentemente el máximo número de participantes en el bus.

En el nivel de enlace el identificador se introduce en el campo arbitraje. Se añaden bits de control propios del nivel de enlace (CTRL.), bits de redundancia cíclica (CRC), bit de confirmación (ACK), delimitadores (del), bits indicadores de final de trama (EOF) y bits para el inter-espaciado de tramas (IFS).

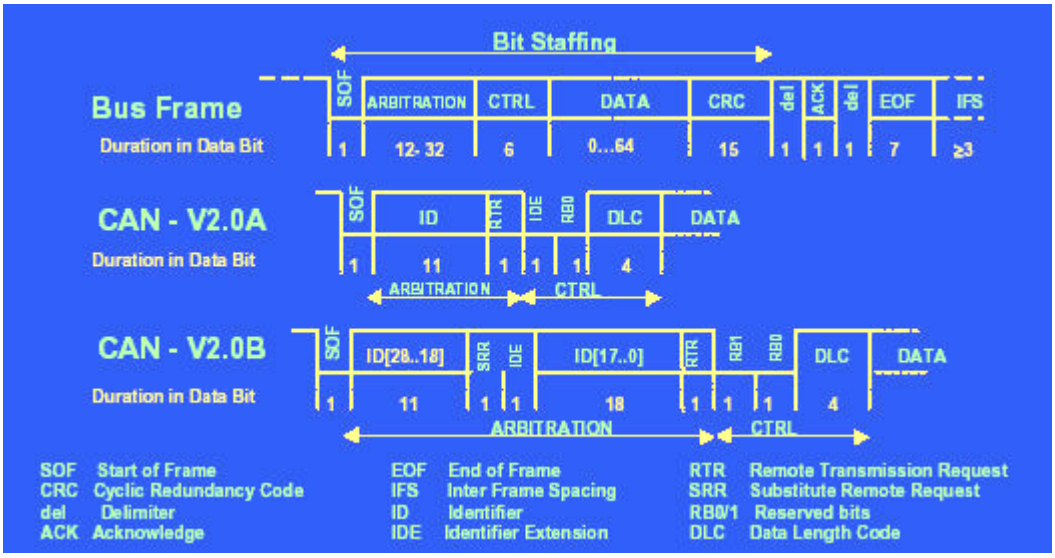


Figura 3.1 – 2 a) Trama física y formatos CAN 2.0 A y B.

Existen otros tipos de tramas como las remotas, error y sobrecarga. Las tramas remotas son la respuesta a tramas de datos con el bit RTR activado.

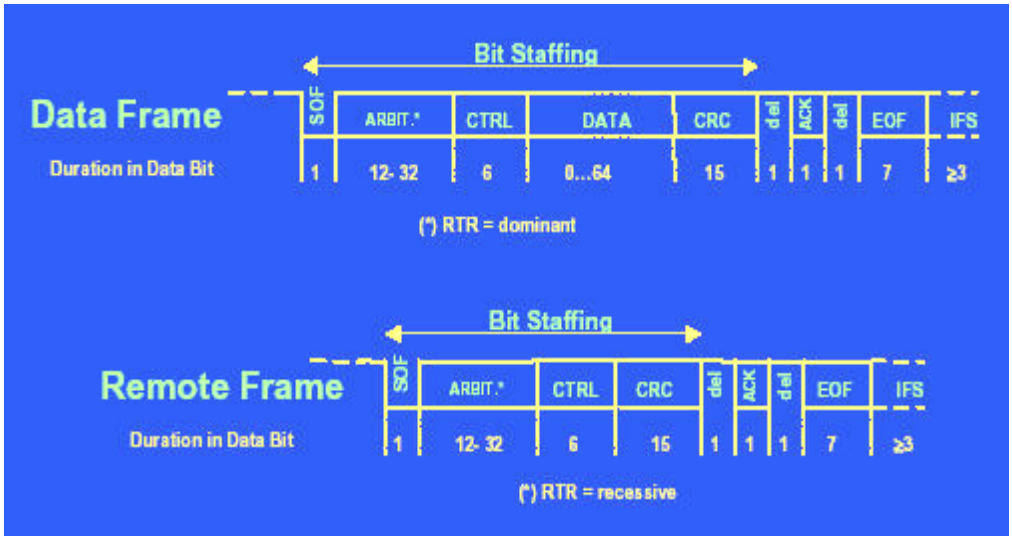


Figura 3.1 – 2 a) trama de datos y asentimiento remoto

Las tramas de error son emitidas por cualquier nodo que detecte una violación del formato de la trama o de la codificación de bit.

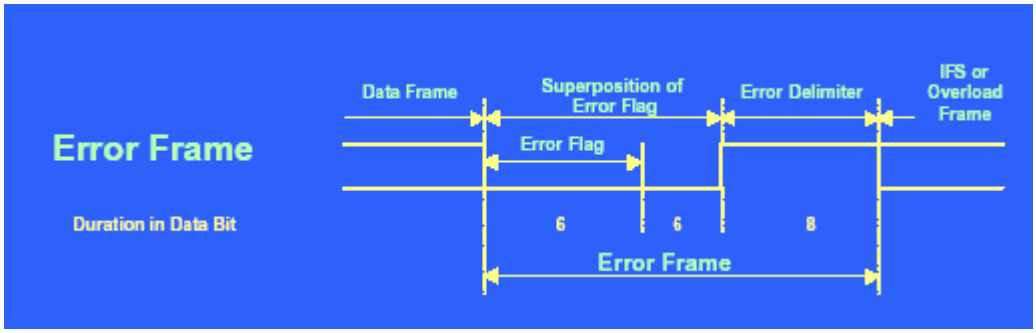


Figura 3.1 –2 c) Trama de error

Las tramas de sobrecarga son emitidas por cualquier nodo que sufra un desbordamiento.

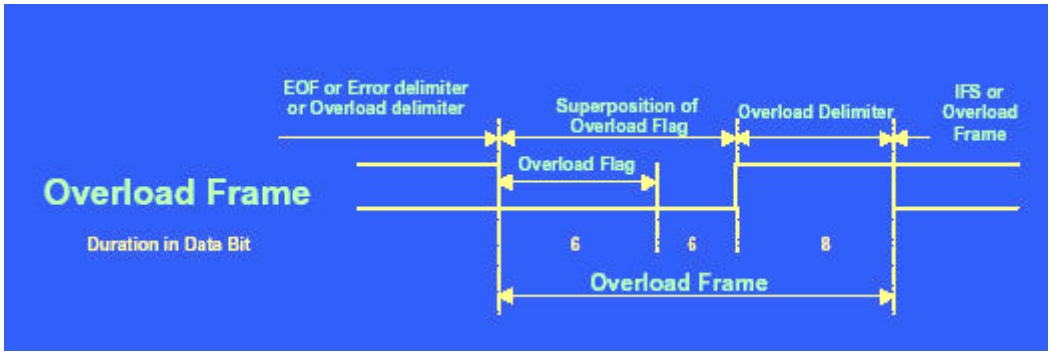


Figura 3.1 –2 d) Trama de sobrecarga

Niveles lógicos del bus CAN.

En la capa física se suele renombrar a los niveles lógicos como dominante y recesivo:

- "1" (nivel alto de tensión) pasa a denominarse estado recesivo.
- "0" (nivel bajo de tensión) pasa a denominarse estado dominante.

Esta denominación se basa en que si dos nodos escriben a la vez en el bus lo que se lee en el bus es el nivel de tensión bajo por eso pasa a llamarse dominante. Esta propiedad es la que se utiliza para priorizar los mensajes en el acceso al bus como se estudiará posteriormente.

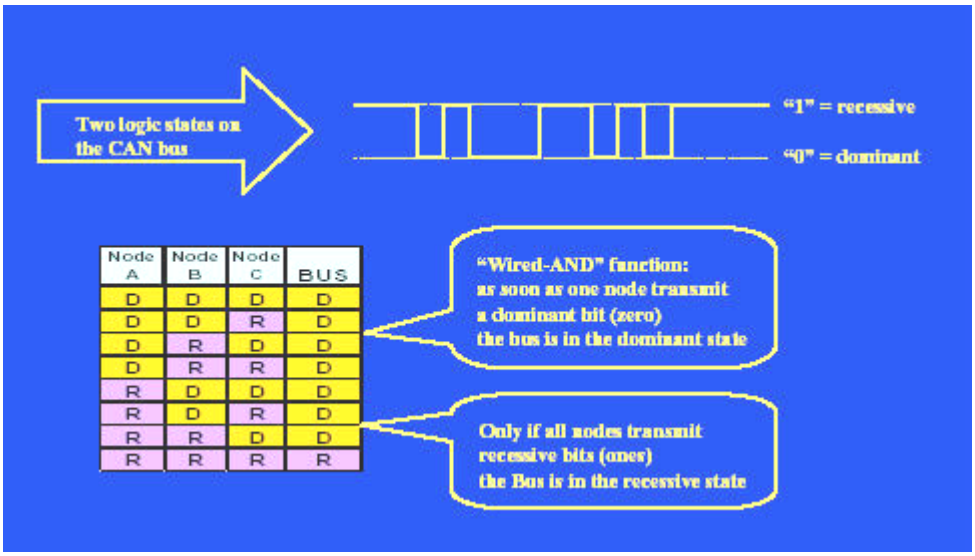


Figura 3.1 –3: Estados dominantes y recesivos

Arbitraje de acceso al Bus CAN.

El acceso al bus CAN se regula CSMA/CD, es decir antes de iniciar se escucha la línea para detectar que este libre, aún así puede ser que dos nodos comiencen a transmitir a la vez. Mientras los nodos escriben en el bus leen simultáneamente el estado del mismo para detectar colisiones. En cuanto un nodo detecta estado dominante en línea cuando el esperaba leer uno recesivo (que era el que estaba transmitiendo) cesa la transmisión. En la parte inicial de la trama va el identificador por lo que el nodo con identificador menor gana el acceso al bus y con la ventaja de no tener que comenzar a retransmitir de nuevo la trama.

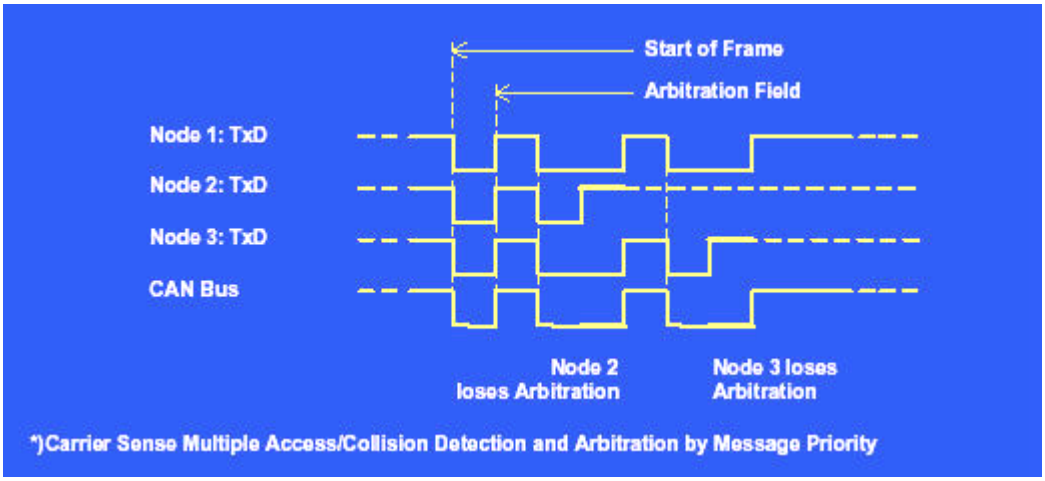


Figura 3.1 – 4: Arbitraje de acceso al bus

Codificación de Bit y sincronización.

El estándar CAN emplea codificación 'Non Return to Zero' (NRZ). Como esta codificación esta codificación no introduce suficientes flancos para mantener la sincronización de reloj se introducen bits de relleno cada 5 bits consecutivos que mantengan la misma polaridad. Este bit de relleno tiene polaridad inversa a la de los 5 anteriores, introduciendo por tanto un flanco que facilita la sincronización.

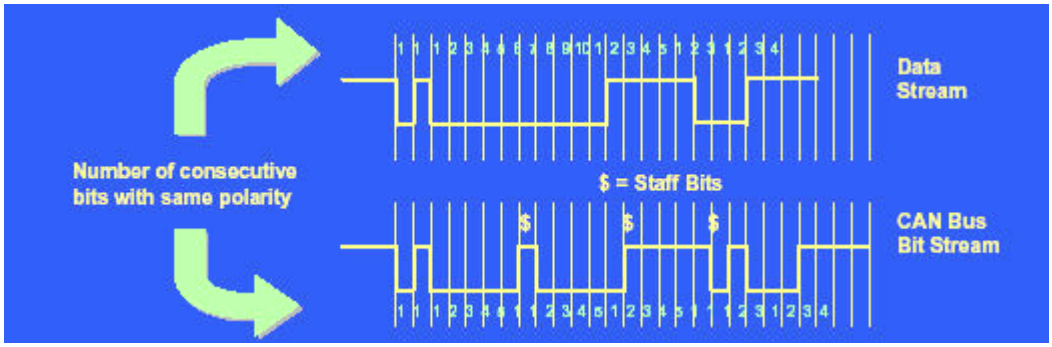


Figura 3.1 –5: Bits de relleno.

La sincronización 'dura' se produce en el bit SOF (Start Of Frame) que es el primer bit de la trama y que siempre es un estado dominante, y posteriormente se producen resincronizaciones en cada transición de estado recesivo a dominante.

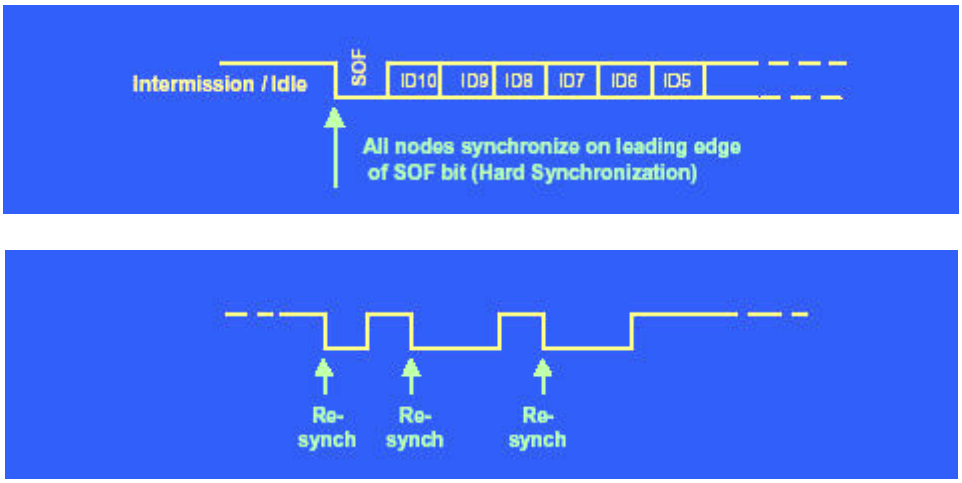


Figura 3.1 – 6: Resincronización

Constitución de un bit CAN.

El periodo de bit está dividido en 'cuantos', para poder configurar la duración del propio periodo de bit , el punto de muestreo dentro del periodo de bit y alguna otra propiedad como una guarda para compensar retardos de propagación de nodos lejanos. Así, todos estos parámetros se configuran a partir del reloj del controlador CAN, a través de estos 'cuantos'.

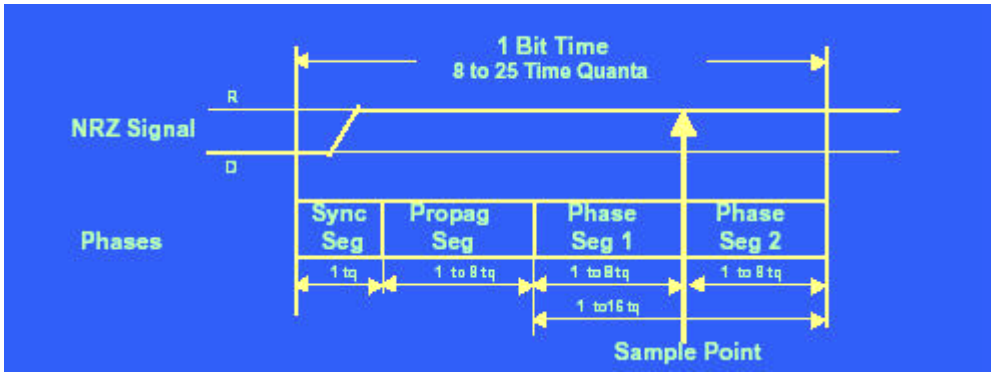


Figura 3.1 – 7: Constitución interna de un periodo de bit.

Un periodo de bit se compone de 8 a 25 'Time Quantum' (TQ), estos TQ están agrupados en 4 segmentos:

- **Sincronismo:** Este TQ es obligatorio.

- **Propagación:** Es segmento consta de 1 a 8 TQ y en función de la topología de la red se usa para compensar retrasos de nodos lejanos. En los controladores SJA1000 de Philips este segmento se introduce en el siguiente segmento Fase 1.

- **Fase 1 y Fase 2:** Se utilizan para fijar el punto de muestreo dentro del periodo de bit.

La velocidad máxima a nivel físico que permite el bus CAN es 1 Mbit/s, por tanto el periodo mínimo es de 1 us. A la hora de fijar la velocidad del bus el principal factor es la distancia máxima entre nodos pues el retardo es el principal enemigo del mecanismo de arbitraje de acceso al bus.

Detección y Confinamiento de errores.

Por confinamiento de errores se entiende la capacidad que tiene un controlador CAN para avisar que esta detectando errores e incluso desconectarse para que ese nodo no introduzca errores al resto del bus. Existen determinados contadores de errores(para transmisión y recepción) y se definen 3 estados:

- **Error Activo:** cuando se ha pasado un determinado umbral en los contadores de errores se entra en este estado. Se mantiene todavía toda la funcionalidad del controlador.

- **Error Pasivo:** cuando se pasa un determinado umbral superior al de Error Activo. En este estado si se limita la funcionalidad y este nodo deja de emitir tramas de error.

- **Bus Off:** En este estado el controlador se aísla del bus para evitar introducir errores en el bus.

Se puede habilitar que el controlador CAN al acceder a alguno de estos estados provoque una interrupción para que el microcontrolador actúe en consecuencia.

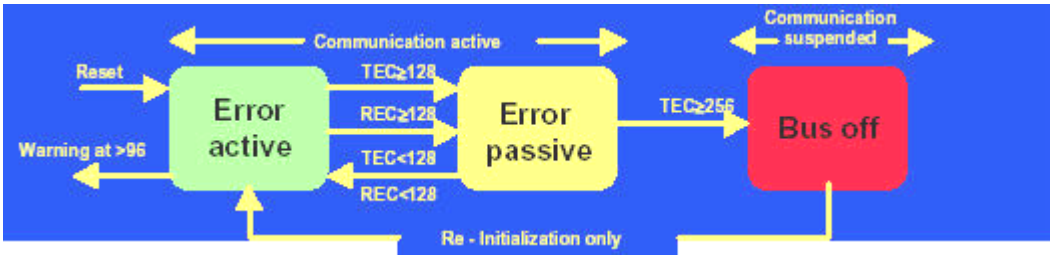


Figura 3.1 – 8: Estados de error

Para la detección de errores se calcula el CRC de la trama desde el bit SOF hasta el último bit de datos excluyendo los bits de relleno y se compara con el CRC recibido. Si coinciden se envía un reconocimiento positivo, si no es así se devuelve una trama de error. Al detectar un error se incrementa el contador asociado (REC y/o TEC) una determinada cantidad que depende del tipo de error (según en que bit de la trama se detectó el error).

En lo que se refiere a errores no detectados cabe decir que depende de los siguientes factores:

- Número total de nodos.

- Disposición física de la red.

- Perturbaciones EMI.

Pero también hay que resaltar que debido al empleo de los bits CRC la probabilidad de no detectar un error es ínfima, por **ejemplo para un automóvil que se emplee 2000 horas al año y un sistema configurado a 500 Kbps con 25 % de carga se produce un error no detectado cada 1000 años.**

Filtrado de aceptación

El filtrado de aceptación es una función que ofrece el controlador CAN. Consiste en programar el controlador CAN para que solo admita como mensajes entrantes aquellos cuyo identificador corresponda a los mensajes destinados al nodo en cuestión. Esta función es muy útil porque reduce la carga de trabajo del microcontrolador *host*.

3.2 Configuración del Bus CAN basado en el controlador SJA1000

Para llevar a cabo la implementación de la sección CAN se emplea el integrado controlador CAN SJA1000 de Philips junto con el transceptor PCA82C250. Este controlador se ha elegido por ser uno de los mas extendidos y por su bajo costo.

3.2.1 Estructura de un nodo del Bus CAN.

La estructura de un nodo CAN se basa en un microcontrolador *host* que tiene como periféricos una serie de sensores y actuadores mas el conjunto formado por el controlador y el transceptor CAN. El controlador CAN es visto por parte del microcontrolador como un periférico mas del tipo MMI (Memory Mapped Interface).

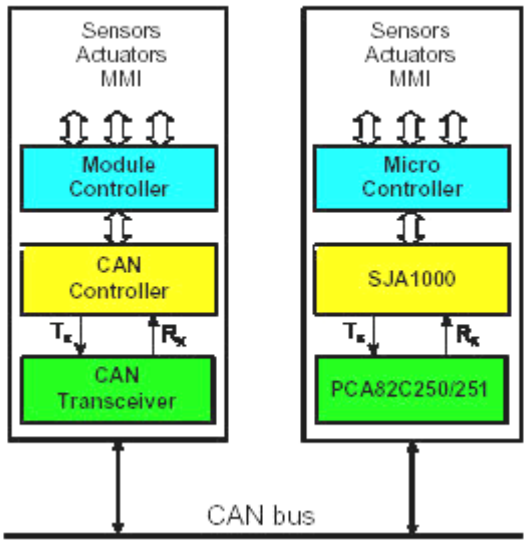


Figura 3.2.1 -1

El **transceptor CAN** convierte los niveles lógicos que recibe del controlador en niveles físicos del bus y viceversa.

El **controlador CAN** implementa el protocolo CAN definido en la especificación del fabricante que es la del estándar más algunos pequeños complementos.

En este punto se van a concretar los aspectos generales, vistos anteriormente, pero particularizados sobre el circuito integrado SJA1000. Este circuito está diseñado para ser compatible con un modelo anterior, el circuito integrado PCA82C200 que trabaja en el modo **BasicCAN**, pero además presenta un nuevo modo mejorado llamado **PeliCAN.** Como todos los nodos que van a funcionar sobre la red son de nueva creación y no hay de necesidad de mantener la compatibilidad con dispositivos PCA82C200 se va a trabajar en modo PeliCAN ya que presenta mejores prestaciones que BasicCAN.

3.2.2 Estructura Interna del controlador SJA1000

En la figura 3.1.2.1 –2 se describen los distintos bloques funcionales dentro del SJA1000.

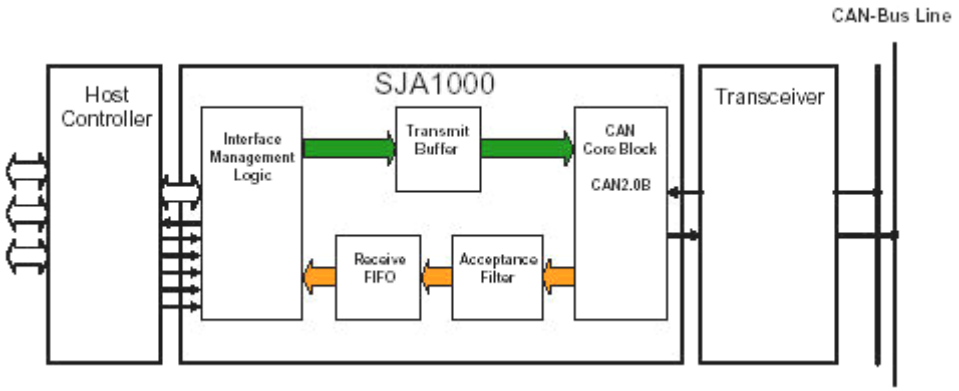


Figura 3.2.2 – 1: Controlador SJA1000

- El **bloque núcleo del CAN** (**CAN Core Block**) controla la transmisión y recepción de tramas CAN de acuerdo a la especificación CAN.

- El bloque de **lógica de gestión del interfaz** (**Interface Management Logic IML**) se encarga de gestionar la comunicación con el controlador *host*. Esta comunicación esta diseñada para trabajar en modo registro o memoria, es decir, para el microcontrolador el SJA1000 se presenta como un dispositivo mapeado en la memoria del microcontrolador. Cada acceso se realiza vía bus direcciones/datos multiplexado y las señales típicas de control (ALE, /CS, /RD, /WR).

- El **Buffer de Transmisión** es capaz de almacenar un mensaje completo (normal o extendido). Cuando una transmisión es iniciada por el controlador *host* la lógica de gestión del interfaz fuerza al núcleo CAN a tomar el contenido del buffer de transmisión y transmitirlo.

- El **filtro de aceptación** se programa para seleccionar que mensajes recibidos deben ser pasados al controlador *host*. Existe una máscara que selecciona los bits del identificador a tener en cuenta para proceder a la aceptación o no del mensaje en función a otro registro que tiene el código de aceptación. Esto se detallará en un punto posterior.

- La **cola de recepción** (**Receive FIFO**) almacena los mensajes que han sido validados por el filtro de aceptación.

En la figura 3.1.2-3 se representa un modelo más detallado de la estructura interna del controlador CAN con otros bloques:

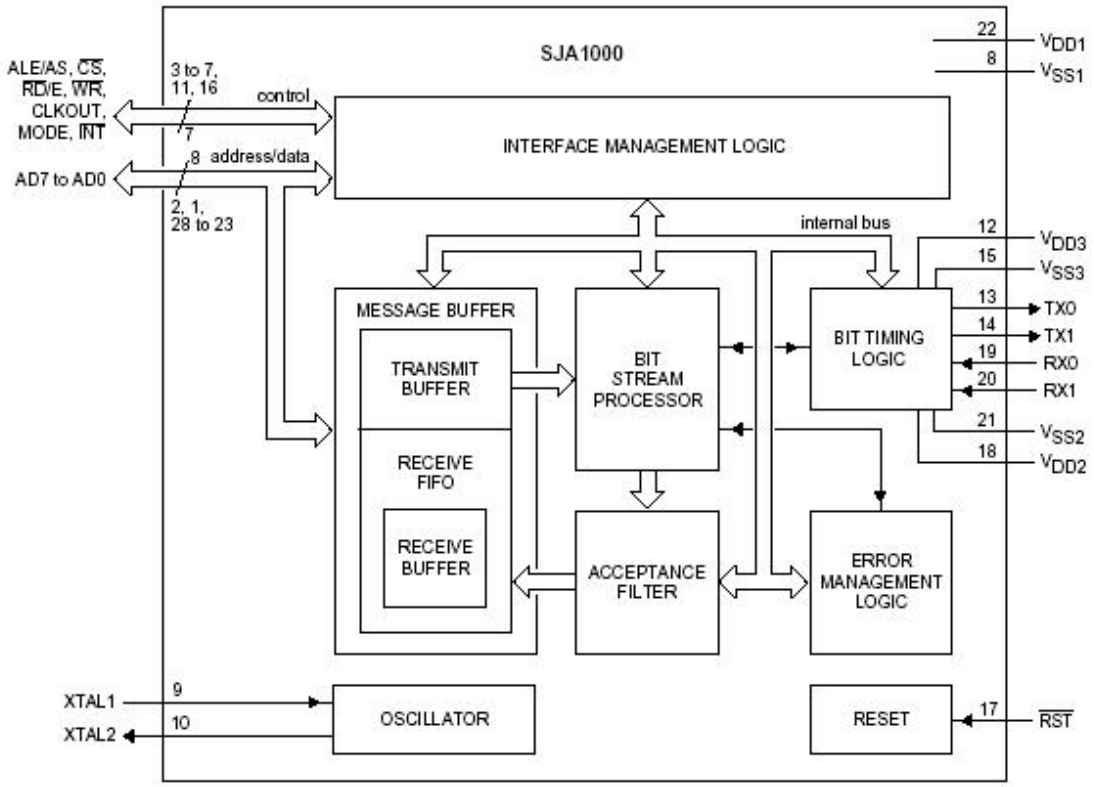


Figura 3.2.2-2: Constitución interna del controlador SJA1000.

- El Procesador de Flujo de Bits (**Bit Stream Processor BSP**) es un secuenciador que controla el flujo de datos entre el buffer de transmisión, la cola de recepción y el bus CAN. También se encarga de la detección de errores, arbitraje, manejo de bits de relleno, y manejo de errores en el bus CAN.

- La Lógica de Temporización de Bit (**Bit Timing Logic BTL**) monitoriza la línea del bus CAN y maneja cuestiones relacionadas con esta como la sincronización del nodo al flujo de bits en el bus. Se

produce una sincronización en la transición de estado recesivo a dominante que hay en el primer bit de la trama (sincronización 'dura') y se resincroniza durante posteriores transiciones similares dentro del mensaje (sincronización ' blanda').

- La Lógica de Gestión de Errores (**Error Management Logic EML**) se encarga de gestión de los errores de los que es informada por parte del BSP. Devuelve información sobre las estadísticas de error al BSP y la IML. También se encarga del confinamiento.

3.2.3 Comparación BasicCAN – PeliCAN

Como se ha mencionado anteriormente el controlador SJA1000 presenta dos modos de trabajo, BasicCAN (compatible con controlador PCA82C200 predecesor del SJA1000) y PeliCAN que presenta nuevas y mejores prestaciones. En este punto se van a detallar esas prestaciones.

El Controlador SJA1000 está diseñado para soportar la especificación completa del protocolo CAN 2.0B lo que significa que permite una tolerancia del oscilador mayor y el procesado de tramas extendidas. En el modo BasicCAN sólo se permite la transmisión / recepción de tramas normales o estándar (con identificador de 11 bits). En este modo si se detectan tramas extendidas (identificador de 29 bits) en el bus son permitidas e incluso confirmadas pero se pasan los datos al microcontrolador *host*.

El modo PeliCAN aparece con un mapa de registros reorganizado respecto al del modo BasicCAN, y presenta las siguientes mejoras en cuanto a prestaciones:

- Recepción y transmisión de tramas normales y extendidas.
- Cola de recepción de 64 bytes, mientras que en BasicCAN sólo se pueden almacenar dos mensajes sin que se provoque desbordamiento.
- Filtro sencillo / doble de aceptación, lo que significa que se pueden usar dos filtros para las funciones de aceptación.
- Se permite el acceso a los contadores y demás registros de errores para el uso del microcontrolador *host*.
- Límite de advertencia de error programable.
- Registro de almacenamiento del código del último error.
- Interrupciones de error para cada tipo de error del bus.
- Interrupción de perdida de arbitraje con información detallada de la posición de bit en la que se ha perdido el acceso al bus en detrimento de una trama de mayor prioridad.
- Transmisión única. Para no reenviar una trama cuando hubo error, es decir , configurar un rechazo simple.
- Posibilidad de modo de solo escucha.
- Posibilidad de deshabilitar la salida de reloj para reducir el consumo.

3.2.4 Configuración básica SJA1000. Registros internos.

La configuración del controlador CAN viene determinada por registros internos del controlador que se presentan al controlador *host* como una memoria en la que cada posición corresponde a un determinado registro. El controlador CAN presenta dos estados o modos de trabajo, el modo operativo (o normal) y el modo reset. El modo reset es un estado interno del controlador CAN en el que se puede acceder a determinados registros que no son accesibles en el modo de trabajo normal.

CAN ADDRESS	OPERATING MODE				RESET MODE	
	READ		WRITE		READ	WRITE
0	mode		mode		mode	mode
1	(00H)		command		(00H)	command
2	status		–		status	–
3	interrupt		–		interrupt	–
4	interrupt enable		interrupt enable		interrupt enable	interrupt enable
5	reserved (00H)		–		reserved (00H)	–
6	bus timing 0		–		bus timing 0	bus timing 0
7	bus timing 1		–		bus timing 1	bus timing 1
8	output control		–		output control	output control
9	test		test; note 2		test	test; note 2
10	reserved (00H)		–		reserved (00H)	–
11	arbitration lost capture		–		arbitration lost capture	–
12	error code capture		–		error code capture	–
13	error warning limit		–		error warning limit	error warning limit
14	RX error counter		–		RX error counter	RX error counter
15	TX error counter		–		TX error counter	TX error counter
16	RX frame information SFF; note 3	RX frame information EFF; note 4	TX frame information SFF; note 3	TX frame information EFF; note 4	acceptance code 0	acceptance code 0
17	RX identifier 1	RX identifier 1	TX identifier 1	TX identifier 1	acceptance code 1	acceptance code 1
18	RX identifier 2	RX identifier 2	TX identifier 2	TX identifier 2	acceptance code 2	acceptance code 2
19	RX data 1	RX identifier 3	TX data 1	TX identifier 3	acceptance code 3	acceptance code 3
20	RX data 2	RX identifier 4	TX data 2	TX identifier 4	acceptance mask 0	acceptance mask 0
21	RX data 3	RX data 1	TX data 3	TX data 1	acceptance mask 1	acceptance mask 1
22	RX data 4	RX data 2	TX data 4	TX data 2	acceptance mask 2	acceptance mask 2
23	RX data 5	RX data 3	TX data 5	TX data 3	acceptance mask 3	acceptance mask 3
24	RX data 6	RX data 4	TX data 6	TX data 4	reserved (00H)	–
25	RX data 7	RX data 5	TX data 7	TX data 5	reserved (00H)	–
26	RX data 8	RX data 6	TX data 8	TX data 6	reserved (00H)	–

Figura 3.2.4 –1: Registros internos del controlador CAN 0:26

CAN ADDRESS	OPERATING MODE				RESET MODE	
	READ		WRITE		READ	WRITE
27	(FIFO RAM); note 5	RX data 7	–	TX data 7	reserved (00H)	–
28	(FIFO RAM); note 5	RX data 8	–	TX data 8	reserved (00H)	–
29	RX message counter		–		RX message counter	–
30	RX buffer start address		–		RX buffer start address	RX buffer start address
31	clock divider		clock divider; note 6		clock divider	clock divider
32	internal RAM address 0 (FIFO)		–		internal RAM address 0	internal RAM address 0
33	internal RAM address 1 (FIFO)		–		internal RAM address 1	internal RAM address 1
↓	↓		↓		↓	↓
95	internal RAM address 63 (FIFO)		–		internal RAM address 63	internal RAM address 63
96	internal RAM address 64 (TX buffer)		–		internal RAM address 64	internal RAM address 64
↓	↓		↓		↓	↓
108	internal RAM address 76 (TX buffer)		–		internal RAM address 76	internal RAM address 76
109	internal RAM address 77 (free)		–		internal RAM address 77	internal RAM address 77
110	internal RAM address 78 (free)		–		internal RAM address 78	internal RAM address 78
111	internal RAM address 79 (free)		–		internal RAM address 79	internal RAM address 79
112	(00H)		–		(00H)	–
↓	↓		↓		↓	↓
127	(00H)		–		(00H)	–

Figura 3.2.4 -2: Registros internos del controlador CAN 27:127

Los registros de configuración se hallan en las primeras 32 posiciones, de la posición 32 a la 96 se encuentra ubicada la cola de recepción (RXFIFO), de la 97 a la 108 el buffer de transmisión y de ahí hasta el final (posición 127) posiciones vacías o rellenas con 0 no utilizables. En las primeras 32 posiciones aparecen posiciones de transmisión y recepción, estas posiciones son las que se usan para enviar y recibir mensajes a / del microcontrolador *host*.

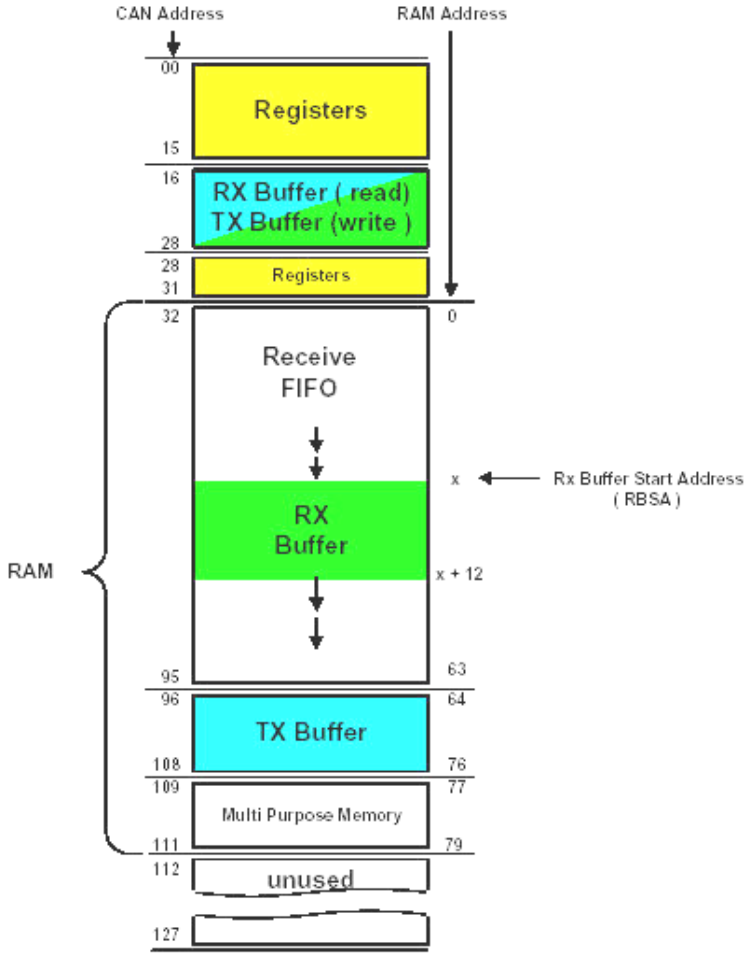


Figura 3.2.4 –3: Mapa de memoria del controlador CAN

Existen dos tipos de configuración, una principal que se hace normalmente al arrancar el controlador CAN y otra secundaria que permite en cada mensaje opciones diferentes.

Los registros de configuración se pueden programar según su uso, así se pueden definir 5 grupos:

- Registros para la selección del modo de operación.
- Registros para el ajuste de la comunicación.
- Registros básicos para la comunicación.
- Registros para la gestión y análisis de errores.
- Registros para la inserción / extracción de mensajes en /del controlador.

Type of Usage	Register Name (Symbol)	Register Address: PeliCAN mode	Register Address: BasicCAN mode	Functionality
elements for selecting different operation modes	Mode (MOD)	0	—	Sleep-, Acceptance Filter-, Self Test-, Listen Only- and Reset-Mode selection
	Control (CR)	—	0	Reset Mode selection in BasicCAN mode
	Command (CMR)	—	1	Sleep mode command in BasicCAN mode
	Clock Divider (CDR)	31	31	set-up of clock signal at CLKOUT (pin 7) selection of PeliCAN Mode, Comparator Bypass Mode, TX1 (pin 14) Output Mode
elements for setting up the CAN communication	Acceptance Code, (ACR) Mask (AMR)	16-19 20-23	4, 5	selection of bit patterns for Acceptance Filtering
	Bus Timing 0 (BTR0) 1 (BTR1)	6 7	6 7	set-up of Bit Timing Parameters
	Output Control (OCR)	8	8	selection of Output Driver properties

Figura 3.2.4 –4

Type of Usage	Register Name (Symbol)	Register Address: PeliCAN mode	Register Address: BasicCAN mode	Functionality
basic elements for the CAN communication	Command (CMR)	1	1	commands for Self Reception, Clear Data Overrun, Release Receive Buffer, Abort Transmission and Transmission Request
	Status (SR)	2	2	status of message buffers, status of CAN Core Block
	Interrupt (IR)	3	3	CAN Interrupt flags
	Interrupt Enable (IER)	4	—	enable/disable of interrupt events in PeliCAN mode
	Control (CR)	—	0	enable/disable of interrupt events in BasicCAN mode
elements for a comprehensive error detection and analysing	Arbitration Lost Capture (ALC)	11	—	shows bit position, where arbitration was lost
	Error Code Capture (ECC)	12	—	shows last error type and location
	Error Warning Limit (EWLR)	13	—	selection of threshold for generating an Error Warning Interrupt
	RX Error Counter (RXERR)	14	—	reflects the current value of the Receive Error Counter
	TX Error Counter (TXERR)	14, 15	—	reflects the current value of the Transmit Error Counter
	Rx Message Counter (RMC)	29	—	number of messages in the Receive FIFO
	Rx Buffer Start Addr. (RBSA)	30	—	shows the current internal RAM address of the message available in the Receive Buffer
message buffers	Transmit Buffer (TXBUF)	16-28	10-19	
	Receive Buffer (RXBUF)	16-28	20-29	

Figura 3.2.4 -5

A continuación se va a comentar las diferentes funciones que tienen los registros de configuración.

Registro de Modo (MOD)

Este es el registro que ocupa la posición 0 del mapa de memoria del controlador CAN. Los bits de este registro controlan ciertos aspectos del funcionamiento.

El bit más importante es **MOD0** (se suele representar por **RM**) que permite entrar en el modo reset o configuración. En este modo se permiten operaciones especiales de configuración que no son accesibles desde el modo operativo o normal. El procedimiento para acceder al modo reset se detalla en la sección SW, pero se resume en que al poner el bit RM a '1' lo se está haciendo es una solicitud de acceso al modo reset, al que se podrá entrar una vez que este bit vuelva a valer '0'.

El bit **MOD1** (**LOM** – Listen Only Mode) sirve para entrar en el modo de solo escucha en el que el controlador CAN inhibe la escritura en el bus, como sucede en el estado de error pasivo.

El bit **MOD2** (**STM** – Self Test Mode) permite entrar en el modo de auto testado, se usa sobre todo en fases iniciales de diseño y para comprobar el funcionamiento del nodo sin necesidad del resto del bus. En este modo es posible la autoescucha del mensaje transmitido si solicita con el comando SRR, más que la autoescucha lo que permite en este modo es que se reciba el mensaje que este mismo nodo está transmitiendo. Esta acción en modo normal no se permite pues realmente los mensajes que emite un nodo no están destinados a él mismo y sólo podrían dar lugar a errores.

El bit **MOD3** (**AFM** – Acceptance Filter Mode) permite seleccionar entre un filtro de aceptación de 32 bits o 2 filtros de 16 bits. Este aspecto se comenta mas detalladamente en una sección posterior, pero se adelanta aquí que se va a trabajar con doble filtro.

El bit **MOD4** (**SM** – Sleep Mode) permite acceder al modo sleep de menor consumo.

El resto de bits de este registro **MOD5** – **MOD7** están reservados y no usan. Se leen siempre como '0' cada uno de ellos.

BIT	SYMBOL	NAME	VALUE	FUNCTION
MOD.7	–	–	–	reserved
MOD.6	–	–	–	reserved
MOD.5	–	–	–	reserved
MOD.4	SM	Sleep Mode; note 1	1	sleep; the CAN controller enters sleep mode if no CAN interrupt is pending and if there is no bus activity
			0	wake-up; the CAN controller wakes up if sleeping
MOD.3	AFM	Acceptance Filter Mode; note 2	1	single; the single acceptance filter option is enabled (one filter with the length of 32 bit is active)
			0	dual; the dual acceptance filter option is enabled (two filters, each with the length of 16 bit are active)
MOD.2	STM	Self Test Mode; note 2	1	self test; in this mode a full node test is possible without any other active node on the bus using the self reception request command; the CAN controller will perform a successful transmission, even if there is no acknowledge received
			0	normal; an acknowledge is required for successful transmission

BIT	SYMBOL	NAME	VALUE	FUNCTION
MOD.1	LOM	Listen Only Mode; notes 2 and 3	1	listen only; in this mode the CAN controller would give no acknowledge to the CAN-bus, even if a message is received successfully; the error counters are stopped at the current value
			0	normal
MOD.0	RM	Reset Mode; note 4	1	reset; detection of a set reset mode bit results in aborting the current transmission/reception of a message and entering the reset mode
			0	normal; on the '1-to-0' transition of the reset mode bit, the CAN controller returns to the operating mode

Figura 3.2.4 - 6: Registro de Modo

Registro de Comando (CMR)

El empleo principal de este registro es ordenar al controlador CAN que inicie la transmisión de un mensaje previamente introducido en el buffer de transmisión. Hay diferentes modos de transmisión solicitan poniendo a '1' el correspondiente bit:

- **CMR0** (**TR** – Transmission Request): ordena una transmisión normal con asentimiento.

- **CMR4** (**SRR** – Self Reception Request): ordena una transmisión con autorecepción,

- **CMR0 y CMR1** a '1' es una combinación especial que ordena una transmisión única (single shot transmission), es decir, si se detecta que la transmisión fue errónea no se retransmite el mensaje.

Otras funciones de este registro vienen definidas por los bits:

- **CMR1** (**AT** – Abort Transmission): ordena abortar una transmisión si todavía no ha comenzado ésta.

- **CMR2** (**RRB** – Release Receive Buffer): este bit se emplea cuando se gestiona la cola de recepción desde el microcontrolador *host*, normalmente se deja este trabajo al controlador CAN que se encarga de ofrecer el mensaje al *host* en las posiciones 17 a 28, liberando de trabajo *host*.

BIT	SYMBOL	NAME	VALUE	FUNCTION
CMR.7	–	reserved	–	–
CMR.6	–	reserved	–	–
CMR.5	–	reserved	–	–
CMR.4	SRR	Self Reception Request; notes 1 and 2	1	present; a message shall be transmitted and received simultaneously
			0	– (absent)
CMR.3	CDO	Clear Data Overrun; note 3	1	clear; the data overrun status bit is cleared
			0	– (no action)
CMR.2	RRB	Release Receive Buffer; note 4	1	released; the receive buffer, representing the message memory space in the RXFIFO is released
			0	– (no action)
CMR.1	AT	Abort Transmission; notes 5 and 2	1	present; if not already in progress, a pending transmission request is cancelled
			0	– (absent)
CMR.0	TR	Transmission Request; notes 6 and 2	1	present; a message shall be transmitted
			0	– (absent)

Figura 3.2.4 - 7: Registro de Comando

Registro de Estado (SR)

El registro de estado refleja el estado del controlador CAN y se presenta al *host* como una posición de memoria de solo lectura. Se suele acceder a este registro antes de iniciar alguna acción para comprobar la viabilidad de ésta, por ejemplo cuando se trabaja con transmisión / recepción con espera activa se chequea el estado de los bits TBS / RBS antes de acceder al buffer correspondiente.

En cuanto al estado de la transmisión / recepción se tiene los siguientes bits de estado:

- **SR0** (**RBS** – Receive Buffer Status) Indica si se puede extraer un mensaje del buffer de recepción.

- **SR2** (**TBS** – Transmit Buffer Status) Indica si se puede introducir un mensaje en el buffer de transmisión.

- **SR3** (**TCS** – Transmission Complete Status) Indica si se ha completado correctamente la última transmisión solicitada.

- **SR4** (**RS** – Receive Status) Indica si hay una recepción en curso.
- **SR5** (**TS** – Transmit Status) Indica si hay una transmisión den curso.
- **SR1** (**DOS** – Data Overrun Status) Indica que el último mensaje entrante se ha perdido porque no cabe en la cola de recepción.

BIT	SYMBOL	NAME	VALUE	FUNCTION
SR.7	BS	Bus Status; note 1	1	bus-off; the CAN controller is not involved in bus activities
			0	bus-on; the CAN controller is involved in bus activities
SR.6	ES	Error Status; note 2	1	error; at least one of the error counters has reached or exceeded the CPU warning limit defined by the Error Warning Limit Register (EWLR)
			0	ok; both error counters are below the warning limit
SR.5	TS	Transmit Status; note 3	1	transmit; the CAN controller is transmitting a message
			0	idle
SR.4	RS	Receive Status; note 3	1	receive; the CAN controller is receiving a message
			0	idle
SR.3	TCS	Transmission Complete Status; note 4	1	complete; last requested transmission has been successfully completed
			0	incomplete; previously requested transmission is not yet completed
SR.2	TBS	Transmit Buffer Status; note 5	1	released; the CPU may write a message into the transmit buffer
			0	locked; the CPU cannot access the transmit buffer; a message is either waiting for transmission or is in the process of being transmitted
SR.1	DOS	Data Overrun Status; note 6	1	overrun; a message was lost because there was not enough space for that message in the RXFIFO
			0	absent; no data overrun has occurred since the last clear data overrun command was given
SR.0	RBS	Receive Buffer Status; note 7	1	full; one or more complete messages are available in the RXFIFO
			0	empty; no message is available

Figura 3.2.4 - 8: Registro de Estado.

Registro de interrupciones y de habilitación de interrupciones.

Con estos dos registros se configuran las interrupciones que puede generar el controlador CAN. Estas interrupciones son un mecanismo de comunicación para indicar al microcontrolador *host* determinados eventos. Los eventos básicos son la recepción de un mensaje y la posibilidad de transmitir un nuevo mensaje, otros eventos son condiciones de error. El **registro de interrupciones IR** actúa a modo de registro de

estado, el microcontrolador *host* recibe la señal de interrupción a través de una determinada línea física que es puesta a nivel bajo por el controlador CAN, en la rutina de atención correspondiente lo primero que hace el microcontrolador es leer el valor de **IR** para determinar que tipo de evento provocó la interrupción. Al leer este registro se borran los bits *flags* que se habían activado.

Por otro lado se puede configurar en el controlador CAN que eventos solicitan una interrupción al microcontrolador *host*, para esto se emplea el **registro de habilitación de interrupciones IER**.

Las interrupciones más usadas son las de transmisión y recepción. Otro grupo de interrupciones son las que se refieren a errores:

- Advertencia de error (**EI**).
- Desbordamiento del buffer (**DOI**)
- Error Pasivo (**EPI**).
- Perdida de arbitraje (**ALI**)
- Error en el Bus (**BEI**)

BIT	SYMBOL	NAME	VALUE	FUNCTION
IR.7	BEI	Bus Error Interrupt	1	set; this bit is set when the CAN controller detects an error on the CAN-bus and the BEIE bit is set within the interrupt enable register
			0	reset
IR.6	ALI	Arbitration Lost Interrupt	1	set; this bit is set when the CAN controller lost the arbitration and becomes a receiver and the ALIE bit is set within the interrupt enable register
			0	reset
IR.5	EPI	Error Passive Interrupt	1	set; this bit is set whenever the CAN controller has reached the error passive status (at least one error counter exceeds the protocol-defined level of 127) or if the CAN controller is in the error passive status and enters the error active status again and the EPIE bit is set within the interrupt enable register
			0	reset
IR.4	WUI	Wake-Up Interrupt; note 1	1	set; this bit is set when the CAN controller is sleeping and bus activity is detected and the WUIE bit is set within the interrupt enable register
			0	reset
IR.3	DOI	Data Overrun Interrupt	1	set; this bit is set on a '0-to-1' transition of the data overrun status bit and the DOIE bit is set within the interrupt enable register
			0	reset
IR.2	EI	Error Warning Interrupt	1	set; this bit is set on every change (set and clear) of either the error status or bus status bits and the EIE bit is set within the interrupt enable register
			0	reset
IR.1	TI	Transmit Interrupt	1	set; this bit is set whenever the transmit buffer status changes from '0-to-1' (released) and the TIE bit is set within the interrupt enable register
			0	reset
IR.0	RI	Receive Interrupt; note 2	1	set; this bit is set while the receive FIFO is not empty and the RIE bit is set within the interrupt enable register
			0	reset; no more message is available within the RXFIFO

Figura 3.2.4 - 9: Registro de Interrupciones.

BIT	SYMBOL	NAME	VALUE	FUNCTION
IER.7	BEIE	Bus Error Interrupt Enable	1	enabled; if an bus error has been detected, the CAN controller requests the respective interrupt
			0	disabled
IER.6	ALIE	Arbitration Lost Interrupt Enable	1	enabled; if the CAN controller has lost arbitration, the respective interrupt is requested
			0	disabled
IER.5	EPIE	Error Passive Interrupt Enable	1	enabled; if the error status of the CAN controller changes from error active to error passive or vice versa, the respective interrupt is requested
			0	disabled
IER.4	WUIE	Wake-Up Interrupt Enable	1	enabled; if the sleeping CAN controller wakes up, the respective interrupt is requested
			0	disabled
IER.3	DOIE	Data Overrun Interrupt Enable	1	enabled; if the data overrun status bit is set (see status register; Table 14), the CAN controller requests the respective interrupt
			0	disabled
IER.2	EIE	Error Warning Interrupt Enable	1	enabled; if the error or bus status change (see status register; Table 14), the CAN controller requests the respective interrupt
			0	disabled
IER.1	TIE	Transmit Interrupt Enable	1	enabled; when a message has been successfully transmitted or the transmit buffer is accessible again (e.g. after an abort transmission command), the CAN controller requests the respective interrupt
			0	disabled
IER.0	RIE	Receive Interrupt Enable; note 1	1	enabled; when the receive buffer status is 'full' the CAN controller requests the respective interrupt
			0	disabled

Figura 3.2.4 - 10: Registro de Habilitación de Interrupciones.

3.2.5 Configuración Funcional del Bus CAN para el helicóptero.

3.2.5.1 Esquema de Comunicación y Direccionamiento.

En este punto se procede a definir aspectos de bus CAN, es decir como se va a emplear en esta implementación concreta. Antes se recuerdan las especificaciones:

- 12 señales
- señales de 8 a 12 bits como máximo
- máxima tasa de muestreo por señal de 1ms
- básicamente la comunicación consiste en:
  - los sensores envían medidas a sistema de control (FCS)
  - el sistema de control (FCS) envía ordenes a los actuadores.

- aunque también los sensores podrían recibir alguna orden de tipo configuración (por ejemplo modificar alguna propiedad como la tasa de muestreo, la resolución, ...) sería interesante dejar algún bit para definir tramas de configuración u otro tipo de tramas.

En este sistema la comunicación es típicamente maestro-esclavo, entre sensores y actuadores (esclavos) y el sistema de control (maestro) salvo por la existencia de las unidades de comunicaciones (objetivo especifico de este proyecto dentro del marco general). Estas unidades de comunicaciones en cada segmento de bus CAN captan todo el tráfico de ese segmento y lo envían vía radio al otro segmento (se podrían también configurar para filtrar mensajes que no sea necesario enviar al otro segmento). El objetivo ideal es que estas unidades de comunicaciones a la hora de escribir en el bus CAN lo hiciesen como si fueran el propio elemento remoto que originó el mensaje. Las unidades de comunicación deben extraer mensajes dirigidos a ellas. También puede ser útil implementar mecanismos de multidifusión, es decir que haya una dirección especial que interpreten todos los nodos como suya.

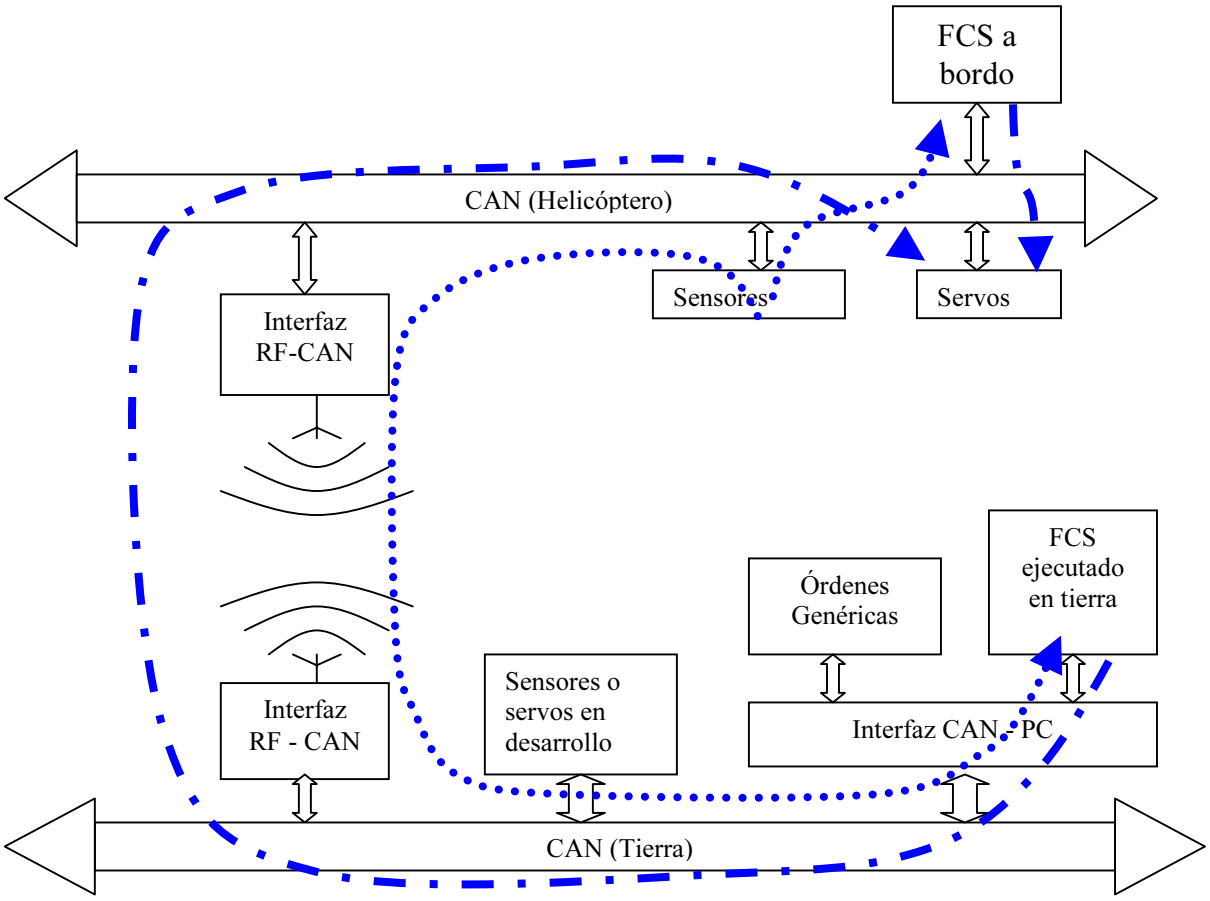


Figura 3.2.5.1 – 1: Sentido del tráfico de mensajes.

Las características de esta comunicación sugieren un método de direccionamiento tal que en el campo dirección de la trama sólo aparecen las direcciones de los esclavos. En el caso de los sensores el maestro recibe la medida y la identificación del sensor. Cuando el maestro quiere enviar una orden de actuación a un servomecanismo en la trama aparece la dirección del servomecanismo. Este método es fácilmente implementable usando las facilidades que ofrece el bus CAN, concretamente el filtrado de aceptación. Como se ha visto el controlador CAN dispone de un código y una máscara de aceptación. La máscara decide que bits del identificador de la trama entrante se tienen en cuenta para comparar con el código de aceptación, de esta manera el controlador CAN libera al microcontrolador *host* de procesar mensajes no dirigidos a él. Además este mecanismo de enmascaramiento permite que un mismo nodo responda varias direcciones, este hecho junto con la posibilidad que tiene el *host* de acceder a la dirección del mensaje entrante permite que las direcciones se puedan asociar a las etiquetas de señal reduciendo así la cantidad de información a introducir el bus, y permitiendo que en un mismo nodo se implemente varios sensores.

Para direccionar las 12 señales bastaría con 4 bits con los que se podrían abarcar hasta 16 señales. Se van a tomar 5 bits con los que se pueden direccionar 32 señales. Además se pueden agrupar señales ubicadas en el mismo nodo de forma que una parte de la dirección indique el nodo y otra la señal dentro del nodo. Como solo se tienen 32 posibles direcciones se va configurar el controlador CAN en el formato de trama normal o estándar que presenta 11 bits para el direccionamiento, pues los 29 bits del formato extendido serían excesivos y únicamente aumentarían la carga del bus.

Trabajando en modo normal o estándar se tiene un identificador de 11 bits, con lo que quedan libres 6 bits, que se pueden aprovechar para transmitir información dado que el *host* puede acceder a la dirección del mensaje y estos bits se pueden ensmascarar para que no se tengan en cuenta en el direccionamiento. De estos 6 bits se toman 4 bits para transmitir datos y los otros 2 se dejan para posibles funciones como tramas de configuración de nodos. Con los 4 bits de datos mas un byte de datos se tienen los 12 bits de tamaño máximo que puede presentar una señal. En la Figura 3.1.2.4.1–2 se presenta como se ha reorganizado la información dentro de la trama para una mejor adaptación a las necesidades concretas de este proyecto. La segunda fila representa la trama donde el *host* introduce la dirección (ID10 – ID0), la longitud de los datos en bytes (DLC), y los datos. En la tercera fila se expone como se aprovecha parte del identificador para enviar los bits D11-D8, el bit CMD para posibles funciones de configuración de una señal, un bit reservado para futuras utilidades RES, el campo DLC se pone un 1 pues solo se va a enviar un byte de datos.

Byte Cabecera 1								Byte Cabecera 2								Bytes datos		
ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR	DLC3	DLC2	DLC1	DLC0	DT7	DT6	….
ID4	ID3	ID2	ID1	ID0	D11	D10	D9	D8	CMD	RES		0	0	0	1	D7	D6	…

Figura 3.2.4 – 12: Empaquetado de identificador y dato.

El microcontrolador *host* se encarga de disponer la información en los registros de transmisión del controlador CAN a partir del identificador de 5 bits y el dato. Si el dato es de 8 bits se utiliza solo el byte de datos de la trama CAN. Si el dato es de mas de 8 bits se ubican estos bits en **D11-D8**. En recepción se procede a recuperar los datos. A partir del identificador el microcontrolador *host* determina con que señal está tratando y sabe como proceder si extrae solo el byte de datos si debe conformar un entero de 12 bits a partir **D11-D8** y el byte de datos (**D7-D0**).

Para la multidifusión de un mensaje que puedan leer todos los nodos, se puede reservar una dirección especial, y aprovechar el doble filtro de aceptación del controlador CAN. En uno de los filtros solo se permite la entrada de mensajes con la dirección del nodo, y en el otro solo la dirección **multicast** (por ejemplo para ordenar un reset de todos los dispositivos). De esta manera se interrumpe en lo mínimo posible a cada uno de los microcontroladores *host* de sistema.

3.2.5.2 Filtrado de aceptación

Una vez que se ha explicado como se va proceder al direccionamiento de las señales en el bus se configura el sistema de filtrado de aceptación de manera que se adapte lo mejor posible al objetivo deseado.

El sistema de filtrado se basa en un registro de código de aceptación (ACR) y en otro de mascara de aceptación (AMR).

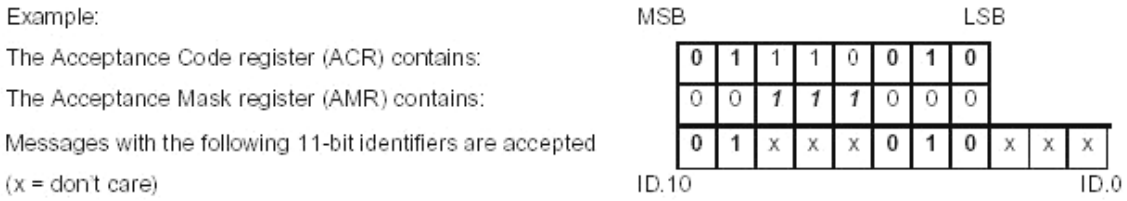


Figura 3.2.5.2 - 1

El registro de máscara decide que bits del identificador del mensaje entrante son relevantes para la aceptación del mensaje, los bits puestos a ‘1’ en AMR no se tienen en cuentan.

En el modo PeliCAN los registros AMR y ACR tienen 4 bytes de ancho y pueden dividirse en cada uno en dos para configurar dos filtros de aceptación (filtrado dual).

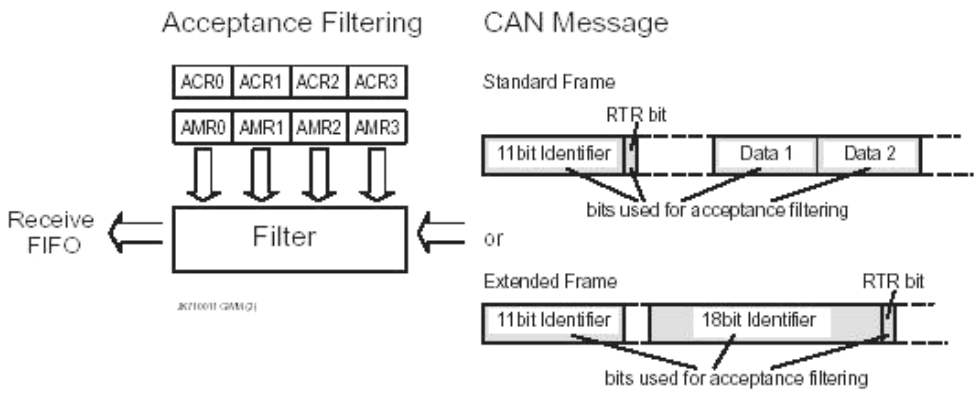


Figura 3.2.5.2 - 2: Filtrado simple en modo PeliCAN.

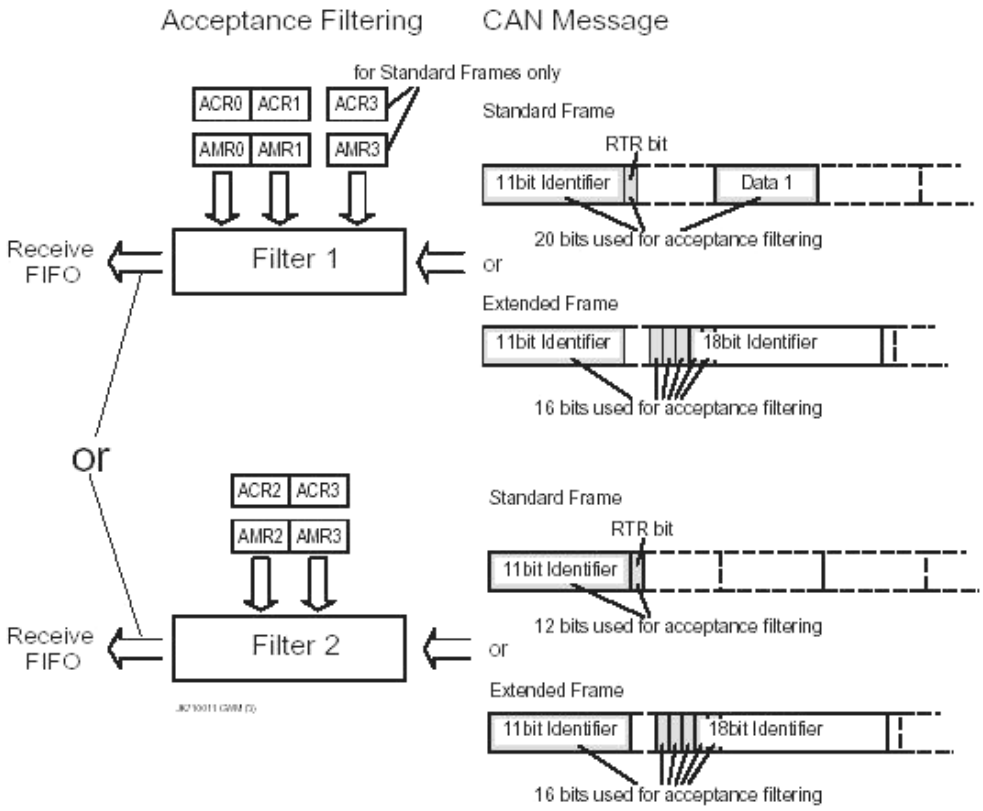


Figura 3.2.5.2 - 3: Filtrado dual en modo PeliCAN.

Como en esta aplicación se va trabajar con identificadores de 11 bits interesa mas el filtro doble pues permite mas versatilidad, por ejemplo, para que un nodo reconozca varias direcciones cuando estas no son compatibles con único filtro o por ejemplo para crear una dirección especial de multidifusión. Para seleccionar del modo de filtrado dual se procede a activar el correspondiente bit (AFM) en el registro de modo. En este modo de trabajo para tramas estándar los filtros que se obtienen son:

- Filtro 1: Filtro de 20 bits (8 bits ACR0/AMR0, 8 bits ACR1/ AMR1, 4 bits LSB ACR3/AMR3). Al aplicar este filtro se están considerando (si la mascara AMR lo permite) los 11 bits del identificador, el bit RTR y el primer byte de datos.

- Filtro 2: Filtro de 12 bits (8 bits de ACR2/AMR2, 4 bits MSB de ACR3/AMR3) que se aplica sobre el identificador de 11 bits y el bit RTR.

En el caso de tramas extendidas se configurarían dos filtros de 16 bits cada uno, el filtro 1 estaría compuesto por ACR0/AMR0 y ACR1/AMR1, y el filtro 2 ACR2/AMR2 ACR3/AMR3.

Para las unidades de comunicación el sistema de filtrado se configura en principio para una aceptación de todos los mensajes en el segmento de bus en el que están ubicadas, pues el cometido principal de estas unidades es hacer llegar vía radio todos estos mensajes al otro segmento del bus. Si existe algún mensaje que no sea necesario enviar al otro segmento se puede configurar el filtro de aceptación en consecuencia, o simplemente configurar el microcontrolador de unidad de comunicaciones para que no pase este mensaje a la sección de radio. Es preferible que todo el filtrado se haga dentro del controlador CAN, pues en otro caso se complican las rutinas de atención del microcontrolador las cuales se desea que sean sencillas.

3.2.5.3 Recepción

Existen dos formas típicas de recepción:

- El microcontrolador *host* se halla en estado de espera activa chequeando el valor del registro de estado del controlador CAN., cuando cambia la bandera de estado asociada a una nueva recepción RBS se procede a leer el mensaje entrante del buffer de recepción.

- Se configura el controlador CAN de manera que genere una interrupción cada vez que se halle disponible un mensaje para su extracción. Lógicamente esta forma de operación libera de trabajo al microcontrolador *host*. Se ha optado por esta opción para la implementación de las unidades de comunicación.

Existe una cola de recepción (RXFIFO) y un buffer de recepción, el buffer de recepción presenta ordenadamente al microcontrolador *host* los mensajes recibidos por el controlador CAN en un espacio fijo de registros del controlador CAN. La cola de

recepción es accesible desde el microcontrolador *host* si que quiere gestionar externamente dicha cola. No se ha hecho uso de esta gestión externa , pues es mas sencillo que el controlador CAN se encargue estas funciones.

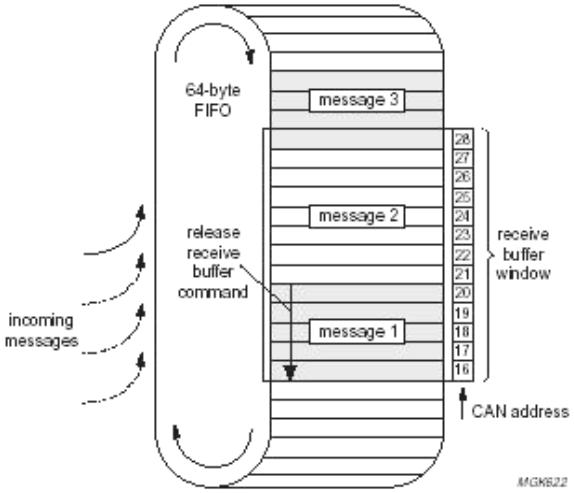


Figura 3.2.5.3 – 1: Cola circular de recepción.

3.2.5.4 Transmisión

La configuración se hace cada vez que se quiere transmitir un mensaje y consiste en escribir los bytes Cabecera 1, Cabecera 2 , el byte de datos y un byte de información sobre la trama a transmitir en los registros del controlador CAN. En los bytes Cabecera 1 y 2 solo hay que poner el identificador de 11 bits, la longitud de los datos y el byte RTR se introducen a través del registro de información sobre la trama. En este registro también se indica el tipo de trama a transmitir en el bit FF (Frame Format).

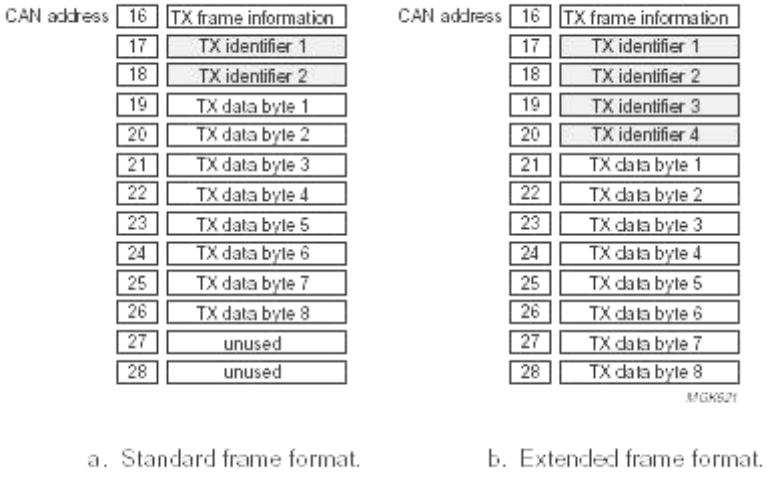


Figura 3.2.5.4 –1: Buffer de Transmisión
a) Trama estándar
b) Trama extendida

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FF(1)	RTR(2)	X(3)	X(3)	DLC.3(4)	DLC.2(4)	DLC.1(4)	DLC.0(4)

Notes

1. Frame format.
2. Remote transmission request.
3. Don't care; recommended to be compatible to receive buffer (0) in case of using the self reception facility (self test).
4. Data length code bit.

Figura 3.2.5.4 – 2: Registro de configuración de trama.

Una vez que se han escrito el registro que configura la trama y la propia trama para iniciar la transmisión solo hay que escribir determinados bits en el registro de comando (CMR). Escribiendo un '1' en el bit CMR0 se inicia la transmisión. Si se escribe '11' en los bits CMR1 y CMR0 se inicia una transmisión única , es decir si hay error no se retransmite. Esto es útil para evitar la trasmisión de datos obsoletos.

En el caso en cuestión se ha programado una interrupción (interrupción de transmisión) para que el controlador CAN avise al microcontrolador *host* de que el controlador CAN está listo para iniciar una nueva transmisión. Esta interrupción se habilitará cada vez que el microcontrolador *host* introduzca un dato en su buffer de transmisión, y se mantendrá activada mientras este buffer tenga mensajes para transmitir. Si la rutina de transmisión detecta que al enviar el mensaje se vacía el buffer de transmisión del microcontrolador deshabilita la interrupción de transmisión para no provocar una interrupción artificial.

En la rutina de atención se procede como se ha explicado al principio de este apartado, se introduce en el buffer de transmisión el identificador y los datos y en el registro de configuración de la trama el tipo de trama y la longitud de los datos.

Finalmente se ordena la transmisión activando el correspondiente bit en el registro de comandos.

3.2.5.5 Gestión de estados error.

Como ya se ha visto en función del estado de los contadores de errores el controlador CAN puede hallarse en uno de los tres siguientes estados de error:

- Error Activo: Cuando ambos contadores se hallan en el intervalo de 0 a 127. Se permite el funcionamiento normal del dispositivo.
- Error Pasivo: Cuando alguno de los dos contadores de error entra en intervalo de 128 a 255. Se limita en parte el funcionamiento del dispositivo.
- Bus-Off: Cuando alguno de los contadores sobrepasa 255. El dispositivo se aísla del bus el mismo, además entra en modo reset y el microcontrolador *host* debe solicitar volver al modo normal de trabajo.

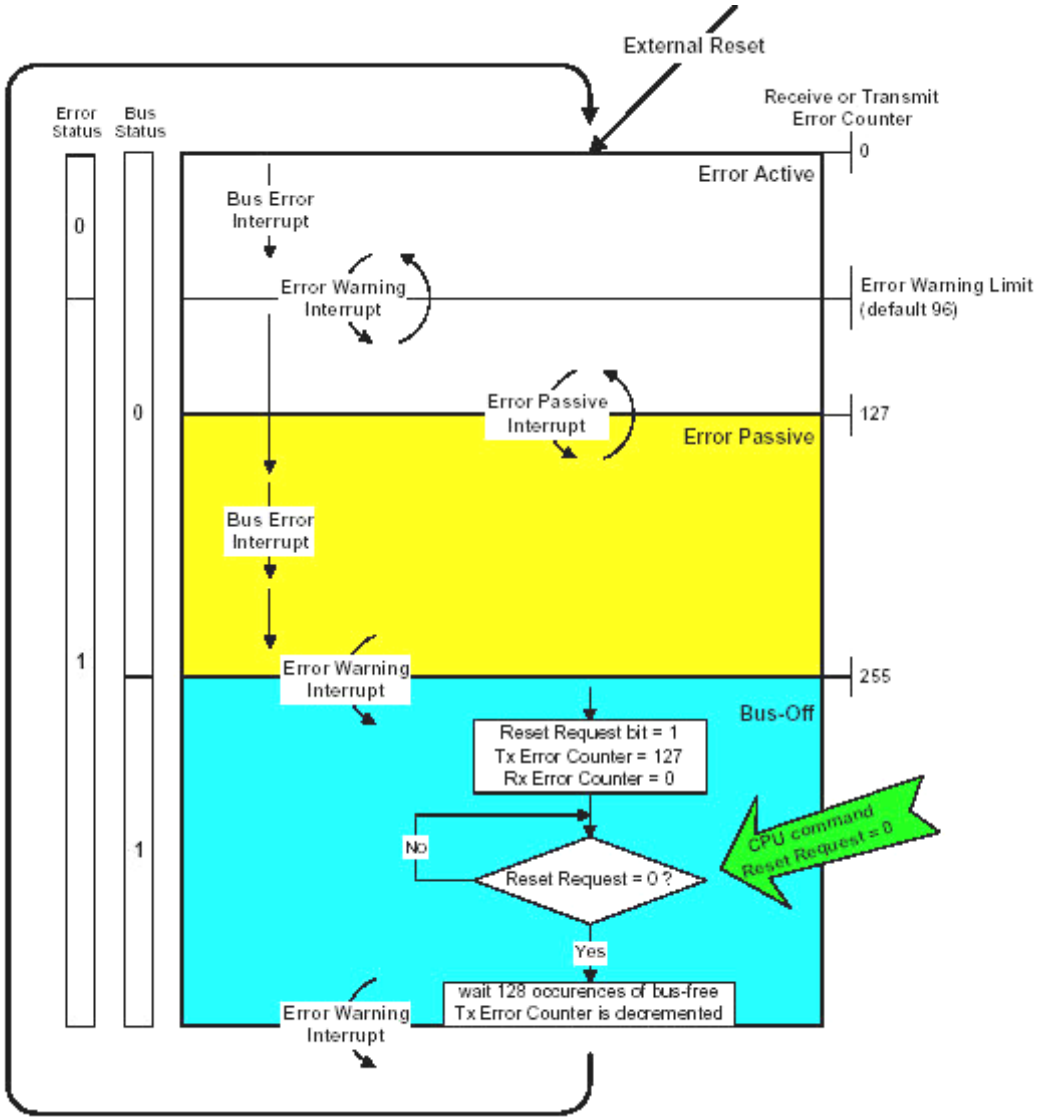


Figura 3.2.5.5 – 1: Estados de error en el controlador SJA1000.

Se puede llegar a análisis muy detallados del error pues el controlador CAN ofrece al microcontrolador *host* mucha información al respecto, en el registro Captura de Código de Error (ECC). Cuando ocurre un error de bus se genera la correspondiente interrupción y el procesador de flujo de bits (BSP) captura la posición del bit erróneo dentro del registro ECC. También el registro ECC indica si el error se produjo en transmisión o recepción.

BIT	SYMBOL	NAME	VALUE	FUNCTION
ECC.7(1)	ERRC1	Error Code 1	–	–
ECC.6(1)	ERRC0	Error Code 0	–	–
ECC.5	DIR	Direction	1	RX; error occurred during reception
			0	TX; error occurred during transmission
ECC.4(2)	SEG4	Segment 4	–	–
ECC.3(2)	SEG3	Segment 3	–	–
ECC.2(2)	SEG2	Segment 2	–	–
ECC.1(2)	SEG1	Segment 1	–	–
ECC.0(2)	SEG0	Segment 0	–	–

Figura 3.2.5.5 – 2: Registro de captura de código de error.

BIT ECC.7	BIT ECC.6	FUNCTION
0	0	bit error
0	1	form error
1	0	stuff error
1	1	other type of error

Figura 3.2.5.5 – 3: Código de error.

BIT ECC.4	BIT ECC.3	BIT ECC.2	BIT ECC.1	BIT ECC.0	FUNCTION
0	0	0	1	1	start of frame
0	0	0	1	0	ID.28 to ID.21
0	0	1	1	0	ID.20 to ID.18
0	0	1	0	0	bit SRTR
0	0	1	0	1	bit IDE
0	0	1	1	1	ID.17 to ID.13
0	1	1	1	1	ID.12 to ID.5
0	1	1	1	0	ID.4 to ID.0
0	1	1	0	0	bit RTR
0	1	1	0	1	reserved bit 1
0	1	0	0	1	reserved bit 0
0	1	0	1	1	data length code
0	1	0	1	0	data field
0	1	0	0	0	CRC sequence
1	1	0	0	0	CRC delimiter
1	1	0	0	1	acknowledge slot
1	1	0	1	1	acknowledge delimiter
1	1	0	1	0	end of frame
1	0	0	1	0	intermission
1	0	0	0	1	active error flag
1	0	1	1	0	passive error flag
1	0	0	1	1	tolerate dominant bits
1	0	1	1	1	error delimiter
1	1	1	0	0	overload flag

Figura 3.2.5.5 – 4: Segmento de la trama erróneo..

Para gestionar los estados de error el controlador CAN dispone de una serie de interrupciones para que en función del estado de error el microcontrolador pueda resolver la situación. Existe una interrupción cuando alguno los contadores (TXERR o RXERR) sobrepasa un limite programable (EWLR que por defecto vale 96), existe otra interrupción cuando se entra en el estado de error pasivo (EPI) y por último la interrupción que avisa de que se ha llegado al estado de Bus-Off (BEI). La forma mas sencilla de proceder con estas interrupciones provocar una reiniciación del controlador CAN, siempre y cuando no se produzcan errores continuos, pues en este caso hay que buscar la causa concreta (fallo en la terminación del bus, sobrecarga, necesidad de mecanismo de sincronización).

3.2.6 Configuración Nivel Físico del Bus CAN.

En este apartado se procede a explicar como se configuran determinados registros del controlador CAN que están orientados a fijar aspectos como la velocidad física del bus o el interfaz con la línea física.

Configuración de la temporización de bit.

Para el controlador CAN se ha usado un cristal de 8 MHz, el empleo de este cristal se debe a que se busca como objetivo la máxima velocidad física del bus CAN que es de 1 Mbit/s. La configuración física de las señales de reloj en el sistema se estudia en la descripción hardware del sistema. En la estructura de un periodo de bit para el controlador SJA1000 de Philips viene detallada en la figura 3.1.2.5.1-1. El periodo de bit consta de 3 segmentos, segmento de sicronización, segmento 1 (previo al punto de muestreo) y segmento 2 (guarda post muestreo). Ademas el periodo de bit se divide en pequeños cuantos (de duración **TQ** de tal manera que los 3 segmentos anteriores son múltiplos de estos cuantos.

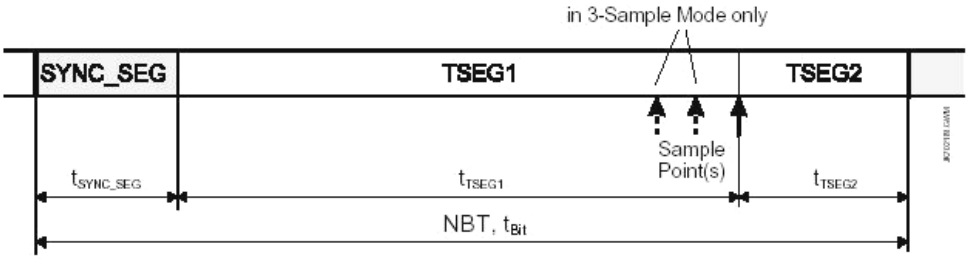
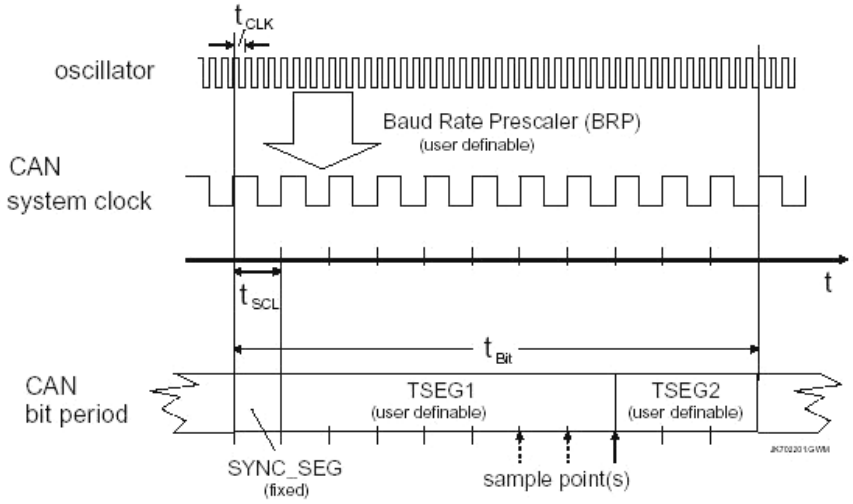


Figura 3.2.6 - 1



Note: t_{SCL} is the time duration of one Time Quantum (TQ)

Figura 3.2.6 -2

Para configurar estos segmentos se hace uso de los registros de control de temporización de bit (**BTR** Bit Timing control Register) BTR0 y BTR1.

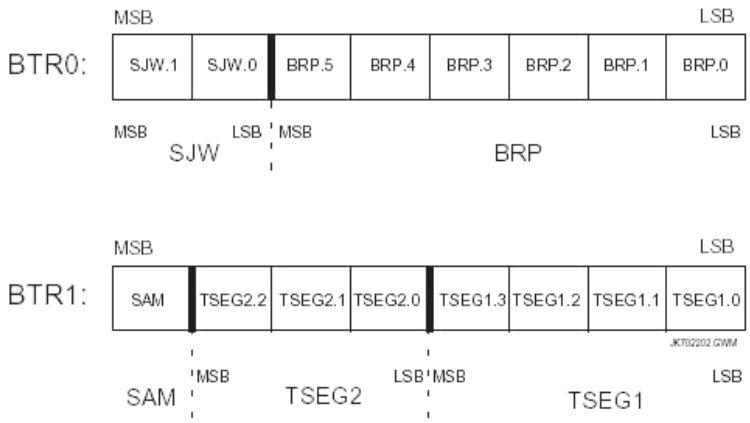


Figura 3.2.6 -3: Bits de los registros de control de temporización.

BRP: **Baud Rate Prescaler** **Register BTR0** **Range: 1....64**

$$BRP = 32\ BRP.5 + 16\ BRP.4 + 8\ BRP.3 + 4\ BRP.3 + 2\ BRP.1 + BRP.0 + 1$$

SAM: **Sample Mode** **Register BTR1**

SAM = 0: 1-Sample Mode

SAM = 1: 3-Sample Mode

SJW: **Synchronization Jump Width** **Register BTR0** **Range: 1....4**

$$SJW = \frac{t_{SJW}}{t_{SCL}} = 2\ SJW.1 + SJW.0 + 1$$

SYNC_SEG: **Synchronization Segment** **fixed** **Value: 1**

$$SYNC\_SEG = \frac{t_{SYNC\_SEG}}{t_{SCL}} = 1$$

TSEG1: **Bit Time Segment 1** **Register BTR1** **Range: 1....16**

$$TSEG1 = \frac{t_{TSEG1}}{t_{SCL}} = 8\ TSEG1.3 + 4\ TSEG1.2 + 2\ TSEG1.1 + TSEG1.0 + 1$$

TSEG2: **Bit Time Segment 2** **Register BTR1** **Range: 1....8**

$$TSEG2 = \frac{t_{TSEG2}}{t_{SCL}} = 4\ TSEG2.2 + 2\ TSEG2.1 + TSEG2.0 + 1$$

NOTE: TSEG2 must be chosen ≥2 in 1-Sample Mode and ≥3 in 3-Sample Mode.

NBT: **Nominal Bit Time** **Range: 3....25**

$$NBT = \frac{t_{Bit}}{t_{SCL}} = SYNC\_SEG + TSEG1 + TSEG2$$

Figura 3.2.6 - 4: Significado de los bits de BTR0 y BTR1.

A partir de las definiciones anteriores y de las ecuaciones:

$$f_{Bit} = \frac{1}{t_{Bit}}$$

$$t_{Bit} = t_{SYNC\_SEG} + t_{TSEG1} + t_{TSEG2}$$

$$t_{SCL} = BRP \cdot 2t_{CLK} = \frac{2\ BRP}{f_{CLK}}$$

Se concluye que para obtener un periodo de bit de 1 us a partir de un reloj de 8MHz, el programa a diseñar tiene que configurar el controlador CAN de la siguiente manera:

- BRP=1
- NBT=8
- SYNC_SEG=1
- TSEG1=4
- TSEG2=3

4. Configuración de la sección de Radio Frecuencia

4.1 Introducción a la tecnología Bluetooth.

La interconexión de dispositivos electrónicos supone un problema en el que se ven implicados distintos aspectos tales como, creación de infraestructuras, imposibilidad de utilización de cables, velocidad de transmisión, etc. Recientemente se ha desarrollado las especificaciones de un sistema radio con el soporte de las empresas líderes en el campo de la electrónica como Ericsson, Nokia, IBM, Toshiba, Intel, etc. Los productos desarrollados permitirán la interconexión sin cables entre sistemas estándares, impresoras, ordenadores, teléfonos móviles, headsets, módems, etc a muy bajo coste con precios estimados del orden de 5 €. Su introducción masiva se pronosticaba para finales de 2001, principios de 2002, pero los costes y el nivel de desarrollo de la tecnología actuales auguran todavía un cierto retardo en la proliferación de productos Bluetooth respecto a las previsiones más optimistas.

Por otro lado, esta tecnología permite la creación de nuevos productos o la adaptación de sistemas actuales disminuyendo costes de instalación e infraestructuras.

4.1.1 Aspectos Básicos de la Tecnología Bluetooth

Bluetooth es un sistema estandarizado que resuelve el problema de acceso de los últimos metros. La estandarización contempla tanto la definición del enlace radio como la arquitectura software que soporta los protocolos necesarios para poder ofrecer distintos servicios. La normativa también contempla diversos perfiles de utilización que definen la interoperatividad entre dispositivos que soporten el mismo tipo de aplicaciones. De este modo se asegura una compatibilidad entre dispositivos y entre los servicios basados en transmisión radio Bluetooth. Por consiguiente, Bluetooth resulta una tecnología apropiada para facilitar la conectividad inalámbrica entre equipos (susceptibles de ser conectados) en dos tipos básicos de escenarios:

- Aplicaciones dentro de las telecomunicaciones
- Aplicaciones dentro de los PC y otro equipo electrónico o entre equipos electrónicos
- Un terminal Bluetooth presenta las siguientes propiedades:
    - Bajo volumen, peso y consumo eléctrico (uno/dos chips)
    - Bajo coste: Precio objetivo 5 €
    - No requiere licencia de Radio (trabaja en la banda ISM aceptada mundialmente de 2.4-2.5 GHz)
    - Robusto a interferencias (por ejemplo hornos microondas y otros terminales Bluetooth)
    - Cada terminal tiene una identidad exclusiva
    - Alcance de 10 m con 1 Mbit/s (nominal) y 0 dBm de potencia.
    - Alcance de 100 m con 20dBm de potencia.

- Conexión de hasta 7 terminales de forma simultánea.
- Arquitectura software de interconexión basada en protocolos existentes. (OBEX, PPP, TCP-IP, Puerto Serie, etc)
- Interoperatividad estandarizada

4.1.2 Arquitectura de protocolos de la tecnología Bluetooth

Tal como se ha comentado anteriormente la arquitectura Bluetooth se basa tanto en la definición del enlace radio como en la torre de protocolos que utiliza dicho enlace y proporciona a las aplicaciones toda una serie de facilidades y servicios de interconexión. A continuación se describen los aspectos básicos de este sistema que deben tenerse en cuenta en el diseño de una solución Bluetooth.

La arquitectura completa de la torre de protocolos de un sistema Bluetooth y su utilización en los diferentes modos de uso contemplados en las especificaciones Bluetooth 1.1 (transferencia de ficheros, transmisión de audio, Acceso LAN, etc.) se muestra en la Figura 4.1.2-1

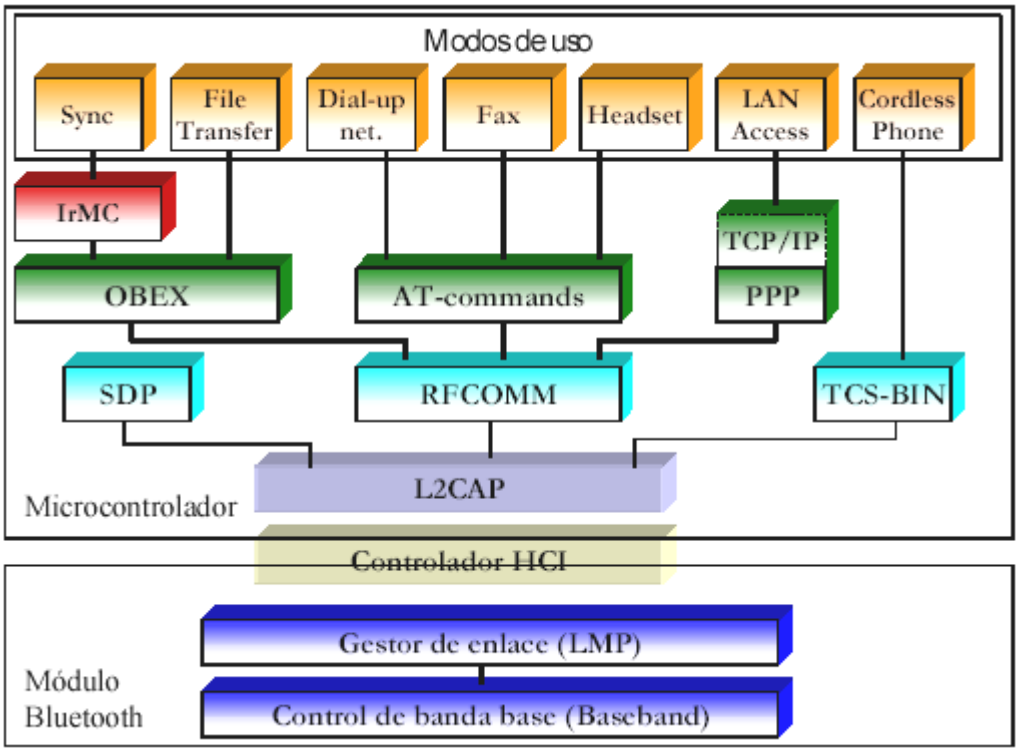


Figura 4.1.2 -1: Componentes y protocolos especificados en Bluetooth y modelos de uso.

La capa de control de banda base es la encargada de gestionar la transmisión en la interfaz radio (formato de paquetes, gestión del salto de frecuencia, etc.). Por encima de dicha capa, el gestor de enlace (LMP) permite la creación y configuración de las conexiones radio con otros dispositivos. Tal y como se indica en la Figura 1, una configuración hardware típica de los módulos Bluetooth actuales consiste en ubicar la capa de control de banda base y la capa LMP en un único chipset, deniminado en la figura como modulo Bluetooth. La comunicación entre dichas capas y las capas superiores de la torre de protocolos se realiza mediante la interfaz HCI (Host Controller Interface). La normativa actual especifica la realización de dicha interfaz a través de una UART, RS-232 o USB. El resto de capas superiores y la propia aplicación podrían ubicarse en un microprocesador externo, ya sea un ordenador personal o bien un terminal de control desarrollado específicamente a tal efecto. Particularmente la capa L2CAP permite la multiplexación de las capas superiores y gestiona la segmentación y re-ensamblado de los paquetes. RFCOMM permite emular una interfaz serie del tipo RS-232. La capa SDP permite que los dispositivos Bluetooth descubran de forma automática los servicios que se encuentran disponibles en otros dispositivos Bluetooth de su entorno radioeléctrico. La capa TCS proporciona servicios de telefonía. El resto de capas representadas en la figura no son exclusivas de Bluetooth pero se consideran en la definición de los perfiles de uso de forma que una aplicación compatible con un determinado perfil, por ejemplo un Headset, debe incluirlos.

4.1.3 Análisis de la tecnología bluetooth.

En este aparatado se comentan las características de Bluetooth para comprender mejor su funcionamiento y capacidades.

Conectividad Ad-Hoc.

La mayor parte de los sistemas de radio actuales se basan en la arquitectura celular. Los móviles se conectan a través de estaciones base para acceder a la red móvil. Las estaciones base administran el tráfico a través de los canales de control. No es posible la comunicación directa entre dos unidades. En los sistemas ***AD HOC*** no hay estaciones base o terminales distinguibles. No existe una estructura cableada que soporte la conectividad de los terminales (no existe un control central). En los sistemas ***AD HOC*** no existen operadores ni ningún sistema de coordinación entre los distintos enlaces que podrían establecerse en una misma área. Un típico sistema ***AD HOC*** es el ***walky-talky***, utilizado por militares, policías, etc. ***Bluetooth*** es el primer sistema que utiliza la conectividad ***AD HOC*** a gran escala y disponible para el gran público.

Inmunidad a la interferencia.

Como ya se introdujo en un apartado anterior para el ensanchamiento del espectro existen dos alternativas: DSSS o FHSS.

DSSS presenta el problema cercano-lejano (near-far problem) que en este caso se relaciona con transmisiones externas al estándar. El problema cercano-lejano consiste en que un transmisor ajeno o con otro código en las cercanías del receptor puede dominar sobre un transmisor lejano con el mismo código del receptor, bloqueando al receptor. DSSS es costoso de implementar. Por otro lado, es sumamente probable que la interferencia presente sea de espectro limitado, lo que hace muy posible espacios del espectro sin interferencia.

En FHSS el espectro se divide en varios canales. Durante una conexión, los transceptores saltan de una frecuencia a otra de manera pseudo aleatoria y de acuerdo a un patrón preestablecido. El ancho de banda instantáneo es pequeño, pero se logra una dispersión de la señal a través del tiempo, con lo cual el problema cercano-lejano puede provocar que haya error en algunas ráfagas pero no en todas.

Definición del canal.

En Bluetooth, se utiliza un esquema FH/TDD. El canal se divide en slots consecutivos de 625 ms. Cada slot utiliza una frecuencia de salto distinta En cada slot se transmite un paquete. Slots consecutivos se utilizan para transmitir y recibir (TDD).

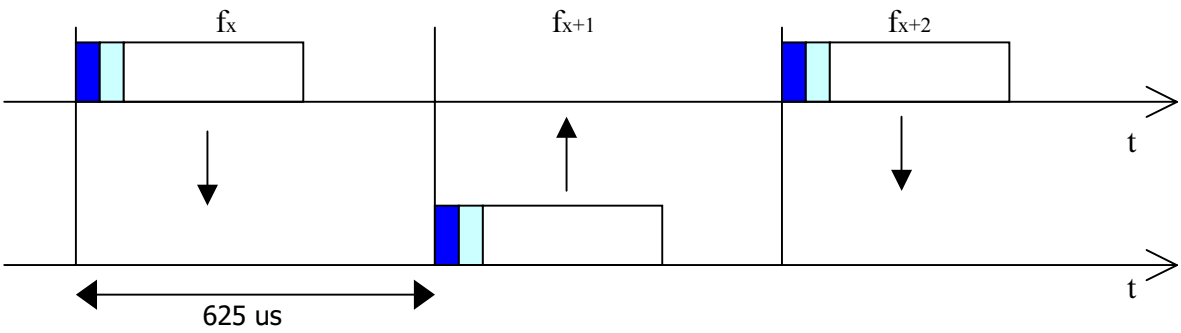


Figura 4.1.3 – 1: Slots de tiempo.

Dos o más unidades que comparten un canal forman una ***Picored,*** este canal FH está determinado por la secuencia de saltos y la fase de esta secuencia. En cada ***Picored*** una unidad actúa como maestro y el resto como esclavos. El tamaño máximo de unidades en una ***Picored*** es 8. La secuencia de saltos está determinada por la identidad del maestro y la fase por el reloj de este.

Para conseguir el reloj maestro en el esclavo, cada unidad debe agregar un ***offset*** a su reloj nativo. De esta forma cada unidad a partir de la identidad del maestro y el ***offset*** respectivo podrá seleccionar adecuadamente la secuencia de salto y permanecerá sincronizada con el resto.

Modulación.

El ancho de banda disponible es de 79 MHz por lo que se dispone de 79 canales de salto. ***Bluetooth*** utiliza FSK con pulsos Gaussianos y un índice de modulación nominal k=0.3. Con esto se logra una tasa de transmisión cercana a 1 Mbps. La elección de este esquema radica en su robustez y simplicidad de implementación (demodulación no coherente).

Control de acceso al medio.

El maestro controla el tráfico y el acceso a la red y establece un control centralizado; solo es posible la comunicación entre maestro y esclavo. Para que no se produzcan colisiones el maestro utiliza ***polling***. En cada slot el maestro decide quien transmite. Este esquema se realiza mediante una conexión ***de a pares***. El maestro transmite un paquete a un esclavo el cuál debe responder en el slot siguiente y no otro.

Se definen 2 tipos de enlace:

- Enlace sincrónico orientado a la conexión (SCO)
- Enlace asincrónico no orientado a la conexión (ACL)

El enlace SCO soporta conexiones simétricas punto a punto y con conmutación de circuitos. Típicamente es usado para voz. Para establecer este tipo de conexión se reservan dos slots consecutivos cada un cierto período fijo. La reserva se realiza cuando se establece la conexión entre el maestro y el esclavo.

Los enlaces ACL soportan conexiones simétricas o asimétricas, punto a multipunto y con conmutación de paquetes. Por defecto, cuando se establece la Picored la unidad maestra establece una conexión ACL con las unidades esclavas. La conexión SCO debe establecerse explícitamente después de que se ha creado la Picored.

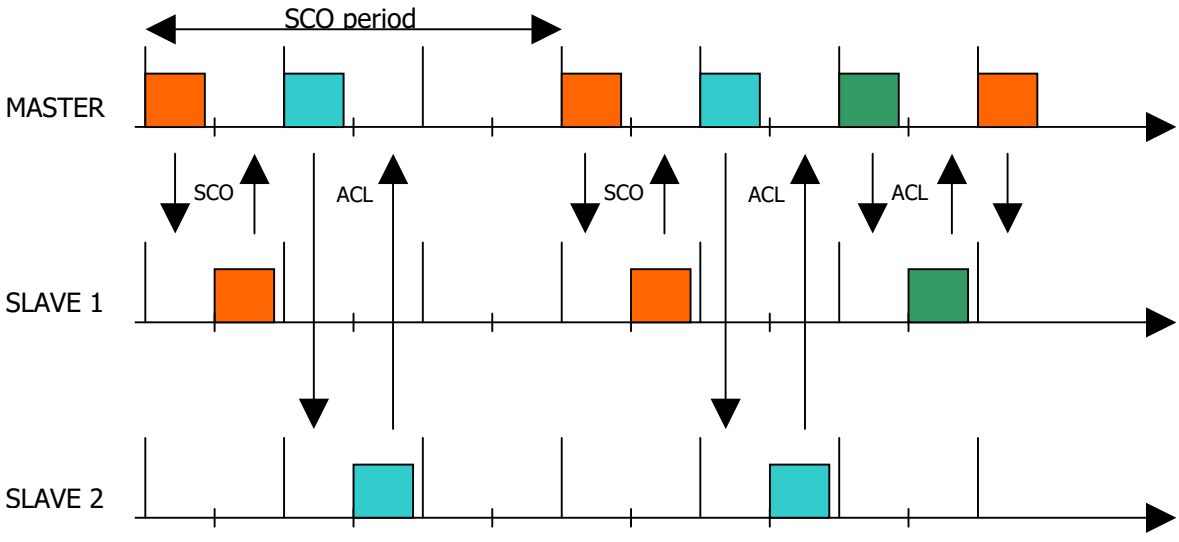


Figura 4.1.3 – 2: Enlaces SCO y ACL.

Definición de paquetes.

En cada slot se intercambia un paquete entre el maestro y alguna de las unidades esclavo. Los paquetes tienen formato fijo. Cada paquete comienza con un código de acceso de 72 bits que se deriva de la identidad del maestro y es único para ese canal.

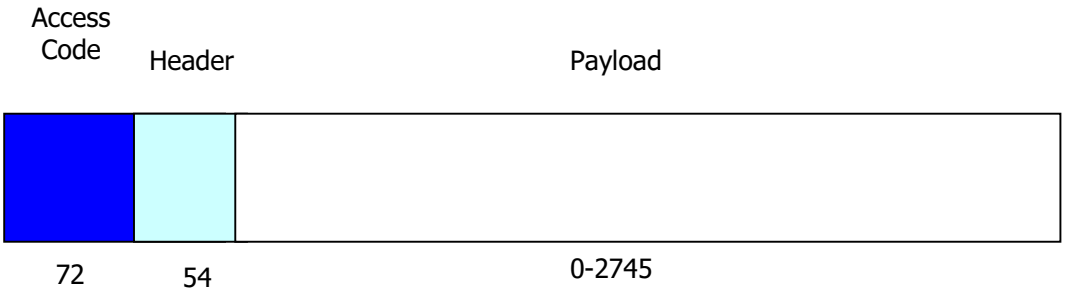


Figura 4.1.3 – 3: Trama de slot.

El código de acceso es seguido de un header. Este contiene importante información de control tal como la dirección del esclavo, tipo de paquete, control de flujo y bits para el ARQ (Automatic Retransmission Query). El ARQ se basa en un sistema Stop-and-Wait con un período de espera de un slot. Es decir el éxito o fracaso de la transmisión se indica en el campo ARQN del paquete de vuelta

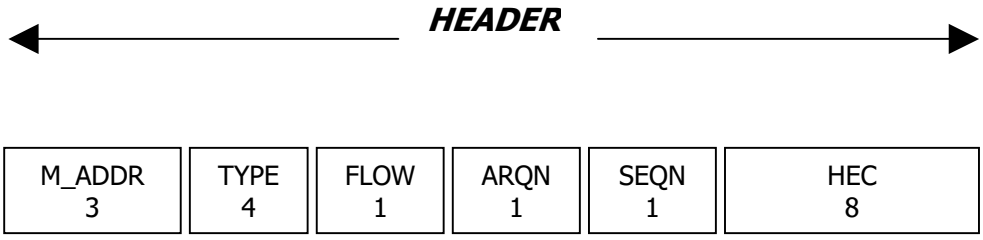


Figura 4.1.3 – 4: Header.

Basados en la información del bit ARQN, el transmisor decide si envía un nuevo paquete o retransmite el anterior. El receptor tiene 220 [ms] entre la transmisión del último bit y el envío de la respuesta. En ese tiempo debe verificar la validez del paquete El campo SEQN permite distinguir si el paquete es nuevo o es una retransmisión.

Este sistema de ARQ es bastante simple pero se adecua perfectamente a la aplicación, debido a que los tiempos de propagación y procesamiento son bastante pequeños, además permite reducir el header. El header esta protegido con un FEC 1/3, el cual se basa en repetir 3 veces cada bit.

Los paquetes ***pueden o no contener payload.*** El tamaño de este puede variar entre 0 y 2745 bits. Para alcanzar payloads mayores a 280 bits, se utiliza formato multislot. Un paquete puede ocupar 1, 3 o 5 slots, dependiendo del tamaño del payload. Durante la transmisión de un paquete se mantiene la misma frecuencia.

Los tipos de paquetes se dividen entre paquetes de control y paquetes de información. Los paquetes de control son de 4 tipos:

- ID: Paquete de identificación. Consiste solo en el código de acceso
- NULL: Consiste solo en el código de acceso y el header. Sirve para enviar información de control
- POLL: Similar al anterior; usado por el maestro para forzar al esclavo a responder
- FHS: Paquete de sincronización. Sirve para intercambiar información de identidad e información de reloj.

Los 12 códigos de paquete restantes sirven para definir el tipo de servicio que se entrega (sincrónico o asincrónico) y el tamaño en slots del paquete. El payload puede o no ser protegido con FEC (1/3 o 2/3). Considerando transmisión sin FEC se puede lograr una máxima tasa asimétrica de 723.2 Kbps con un enlace de retorno de 57.6 Kbps.

Las máximas tasas de transmisión promedio en Kbps que se pueden alcanzar se recogen en la siguiente tabla donde:

- DMx: Paquetes de largo x slots, con FEC
- DHx: Paquetes de largo x slots, sin protección

Type	symmetric	asymmetric	
DM1	108.8	108.8	108.8
DH1	172.8	172.8	172.8
DM3	256.0	384.0	54.4
DH3	384.0	576.0	86.4
DM5	286.7	477.8	36.3
DH5	432.8	721.0	57.6

Tabla 4.1.3 – 1: Máximas tasas de transmisión promedio.

Fiabilidad del sistema.

La tasa de saltos en FH es bastante alta (1600 saltos/seg). Si se pierde un paquete, se pierde solo una pequeña fracción de la información. Los paquetes pueden ser protegidos con FEC y el esquema ARQ permite retransmitir la información perdida con un mínimo retardo. La voz transmitida en enlace SCO nunca se retransmite. En cambio se utiliza un esquema de codificación robusto basado en modulación CVSD (Continuos Variable Slop Delta Modulation).

4.2 Configuración del módulo Free2Move.

Para proceder a la configuración del módulo se ha heho uso del software que suministra el fabricante. Para el correcto funcionamiento del módulo y la obtención de las mejores prestacioens se procede a configurar el el interfaz serie de este módulo para que se pueda comunicar correctamente con el microcontrolador del sistema y ciertos aspectos de la comunicación Bluetooth. Para configurar el módulo se le debe proveer de una alimentación correcta de 4 – 5.0 V a traves del conector de alimentación o del pin numero 9 del conector DB9 y conectarlo al puerto serie de un PC. Al ejecutar el software de configuración aparece la ventana de la figura 4.2. – 1. en la que se debe seleccionar el puertoserie del Pc al que se ha conectado el módulo Free2Move.

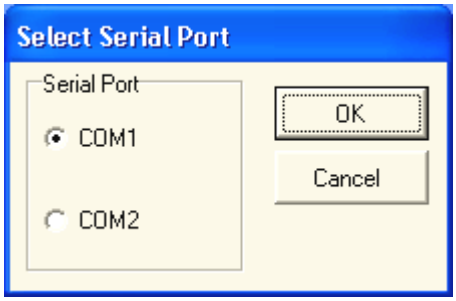


Figura 4.2 –1: Selección de puerto serie.

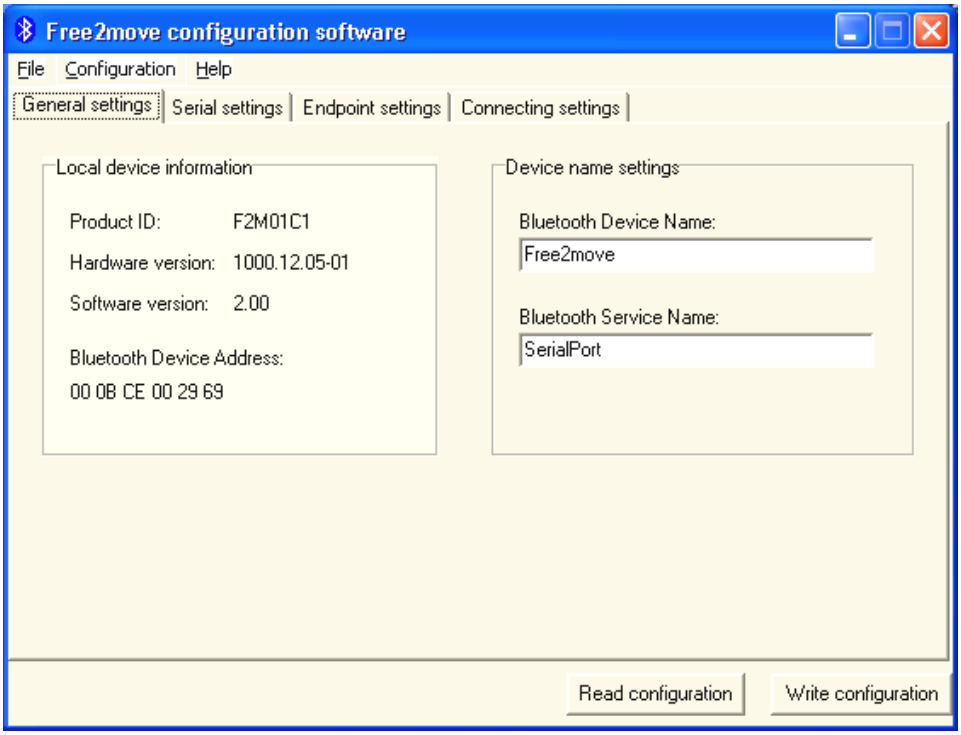


Figura 4.2 – 2: General Settings.

Tras elegir el puerto serie al que se tiene conectado el módulo el programa lee la configuración del módulo y aparece la ventana de la figura 4.2 – 2. En esta ventana hay

diferentes pestañas. En la pestaña *General Settings* se muestra la dirección física del dispositivo (Bluetooth Device Address). Esta dirección es única y no se puede cambiar. También aparecen dos nombres que si son accesibles para ser modificados uno es un nombre para el dispositivo y otro para el servicio. No hace falta modificar estos valores.

En la pestaña de ajustes del interfaz serie, se configura la velocidad, paridad, bits de parada y el control de flujo. Se toman las opciones marcadas en la figura 4.2 – 3.

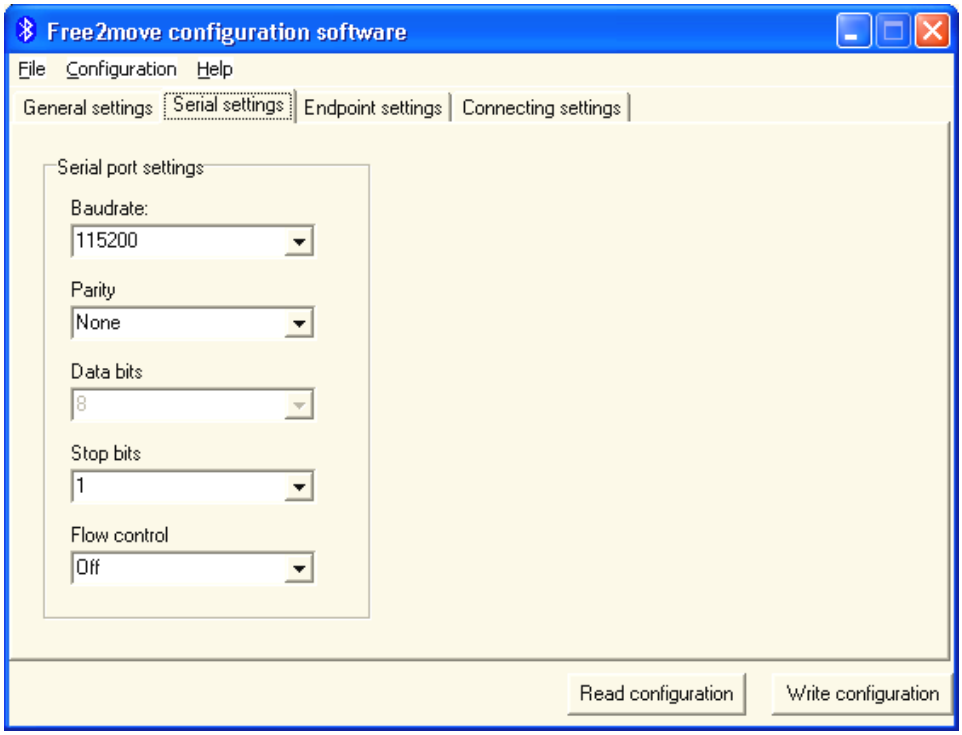


Figura 4.2 – 3: Serial Settings.

Como se vió en la sección sobre la topología de red de Bluetooth uno de los módulos de la picored debe actuar como maestro para la coordinación de la comunicación. Por lo tanto uno los dos módulos se ha de configurar como maestro o principal y el otro como esclavo. Para este tipo de situación el software ofrece dos modos de configuración uno normal en el que se le dice a cada módulo la dirección del otro y un modo emparejado (*paired devices*). La diferencia es el grado de seguridad y encriptación de mensajes. En este caso no son necesarias fuertes restricciones de seguridad por lo que se toma el modo normal indicando la dirección física del otro dispositivo Bluetooth y sin *passkey*. En las figura 4.2 – 4 y 4.2 –5 se aprecia como se han configurado los módulos, uno como maestro y el otro como esclavo, indicándole a cada uno la dirección del otro módulo.

Hay que tener en cuenta que el módulo que funcione como maestro tiene un consumo mayor, por lo que irá ubicado en la tarjeta del sistema que tenga menores restricciones de consumo.

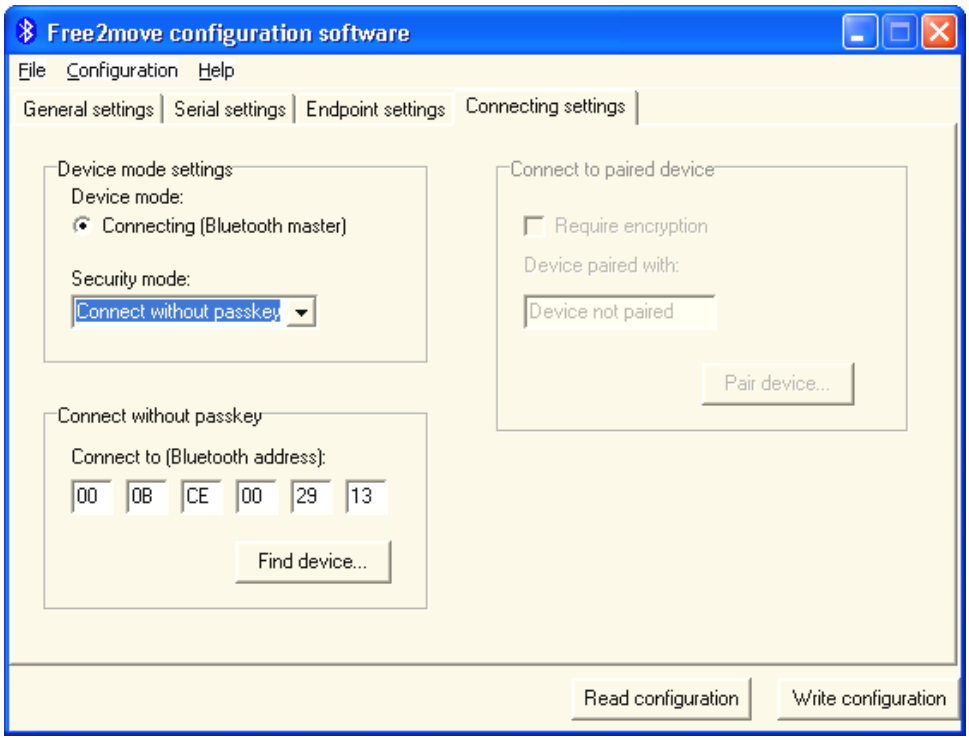


Figura 4.2 – 4: Configuración módulo maestro.

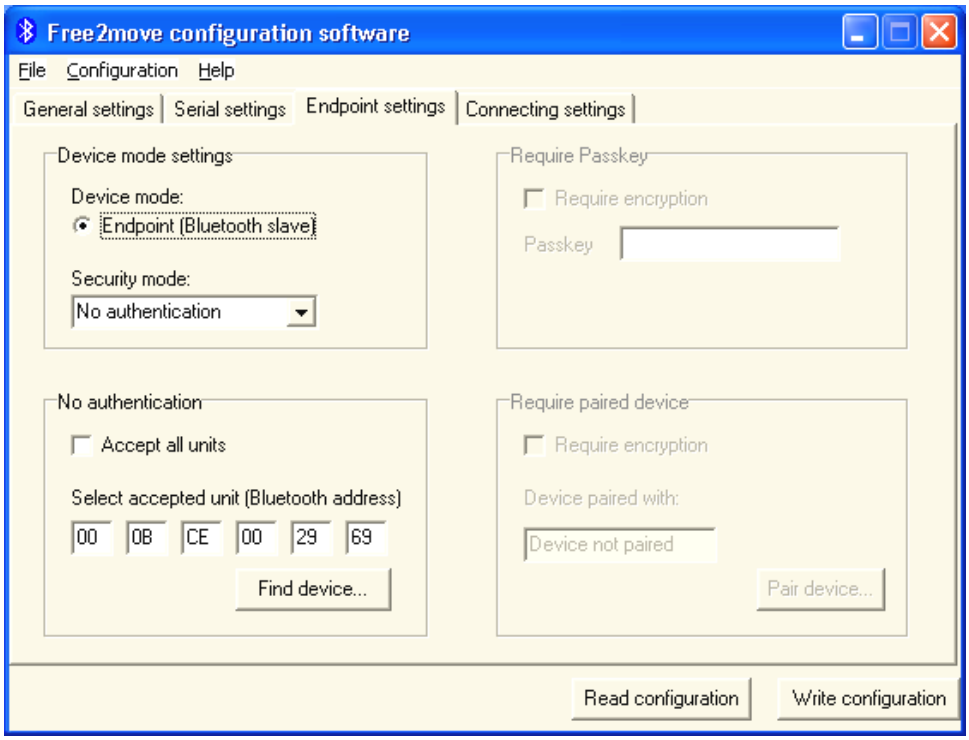


Figura 4.2 – 4: Configuración módulo esclavo.

5. Características del Microcontrolador.

Entre la amplia variedad de microcontroladores AVR se ha seleccionado el modelo ATmega128 pues es uno de los mas potentes dentro de su gama y viene con todos los dispositivos y periféricos integrados. Los principales factores que influyeron en la elección de este microcontrolador fueron su velocidad, amplia memoria de programa y RAM internas, y la presencia de 2 puertos serie UART.

La velocidad es importante pues el puerto serie debe trabajar con tasas de transmisión altas y los modelos mas simples de AVR solo pueden trabajar hasta 8 MHz. La velocidad del puerto serie se obtiene a partir del reloj del sistema y para lograr una tasa de transmisión alta y exacta se necesita un reloj con una frecuencia alta.

En cuanto a la memoria es importante tener bastante RAM pues se prevé que se va a trabajar con búferes de transmisión/ recepción para almacenar la información que tiene que pasar del modulo de radio al bus CAN y viceversa.

Este dispositivos presenta 2 puertos serie (USART), uno de ellos se va utilizar para comunicarse con el módulo de radiofrecuencia y el otro para funciones de depuración. El empleo de este segundo puerto serie facilita la depuración de programas pues se puede ver el comportamiento del sistema ante ordenes mientras esta en ejecución de manera sencilla usando algún programa de terminal serie sobre PC como HyperTerminal.

Hubiera sido válida alguna versión mas reducida la gama ATmega pero se disponía de 2 unidades de este microcontrolador con lo que se ahorra el coste de microcontroladores.

5.1 Descripción del microcontrolador AVR ATmega128.

En este punto se va a proceder a detallar un poco más las características concretas del microcontrolador ATmega128.

Características

Las características que definen a este dispositivo son:

- Avanzada arquitectura RISC
  - 133 potentes instrucciones, la mayoría de las cuales se ejecutan en un único ciclo de reloj
  - Registros de propósito general

  - Registros para el control de periféricos
  - Trabajo en modo estático.
  - Hasta 16 MIPS trabajando a 16 MHz
  - Multiplicador por 2 interno.

- Memoria de programa no volátil y RAM internas
  - 128K Bytes de memoria Flash reprogramable en el sistema final (ISP)
    - Vida útil: 10,000 ciclos escritura/lectura
    - Sección de arranque Bootload opcional con bits de cerrojo independientes.
  - 4K Bytes EEPROM
    - Vida útil: 100,000 ciclos escritura/lectura
  - 4K Bytes de memoria RAM estática interna
    - Hasta 64K Bytes de espacio para memoria externa
  - Protección de programa para seguridad del SW.
  - Interfaz SPI para programación en sistema final (ISP)

- Interfaz para depuración JTAG
  - Soporte para depuración interna
  - Programación de memorias (Flash, EEPROM), fusibles y cerrojos a través del interfaz JTAG

- Caracteríticas de los periféricos
  - 2 Temporizadores/Contadores de 8 bits con programación y modo de comparación independientes
  - 2 Temporizadores/Contadores de 16 bits con programación, modo de comparación y modo de captura independientes

- Contadores de tiempo real (Real Time Clock)

- 2 canales de modulación por ancho de pulso (PWM) de 8 bits

- 6 canales de modulación por ancho de pulso (PWM) con resolución programable de 2 a 16 bits

- Modulador por comparación de salida

- Conversor analógico digital (ADC) de 8 canales y 10 bits.
  - 8 Canales simples
  - 7 Canales diferenciales
  - 2 Canales diferenciales con ganancia programable 1x, 10x, or 200x

- Interfaz TWI (Two-wire Serial Interface) orientado a byte

- Dos unidades USART programables.

- Interfaz SPI Maestro/Esclavo

- Watchdog programable con temporizador generado a partir de oscilador interno

- Comparador analógico interno

- Otras características especiales:
  - Circuito de interno de reset sobre alimentación estable
  - Monitorización de alimentación
  - Oscilador interno RC calibrado
  - Interrupciones externas e internas
  - Modos de bajo consumo: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
  - Frecuencia de reloj seleccionable por SW
  - Modo de compatibilidad con Atmega103
  - Deshabilitación global de resistencias pull-up

- Líneas de entrada/salida y encapsulados
  - 53 líneas de entrada/salidas programables por SW
  - Encapsulados 64-lead TQFP y 64-pad MLF

- Alimentación
  - 2.7 - 5.5V ATmega128L
  - 4.5 - 5.5V ATmega128

- Velocidad
  - 0 - 8 MHz for ATmega128L
  - 0 - 16 MHz for ATmega128

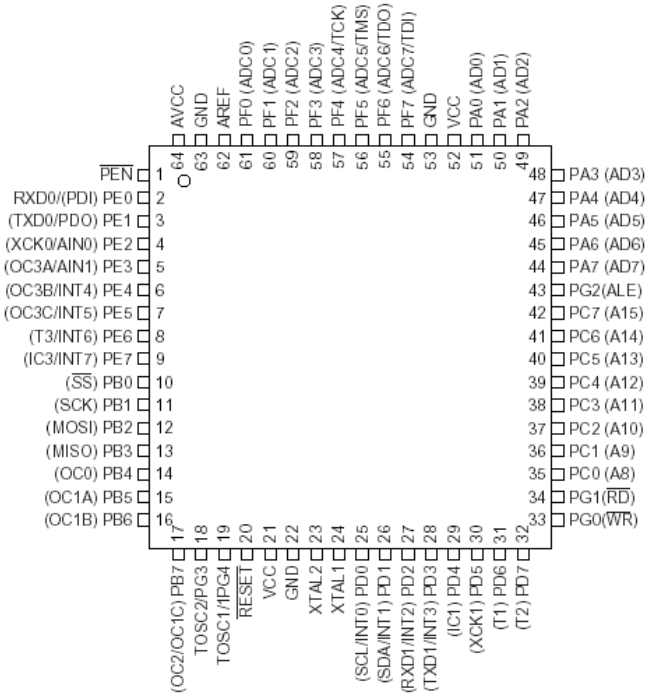


Figura 5.1 – 1: Pines del uC ATmega128

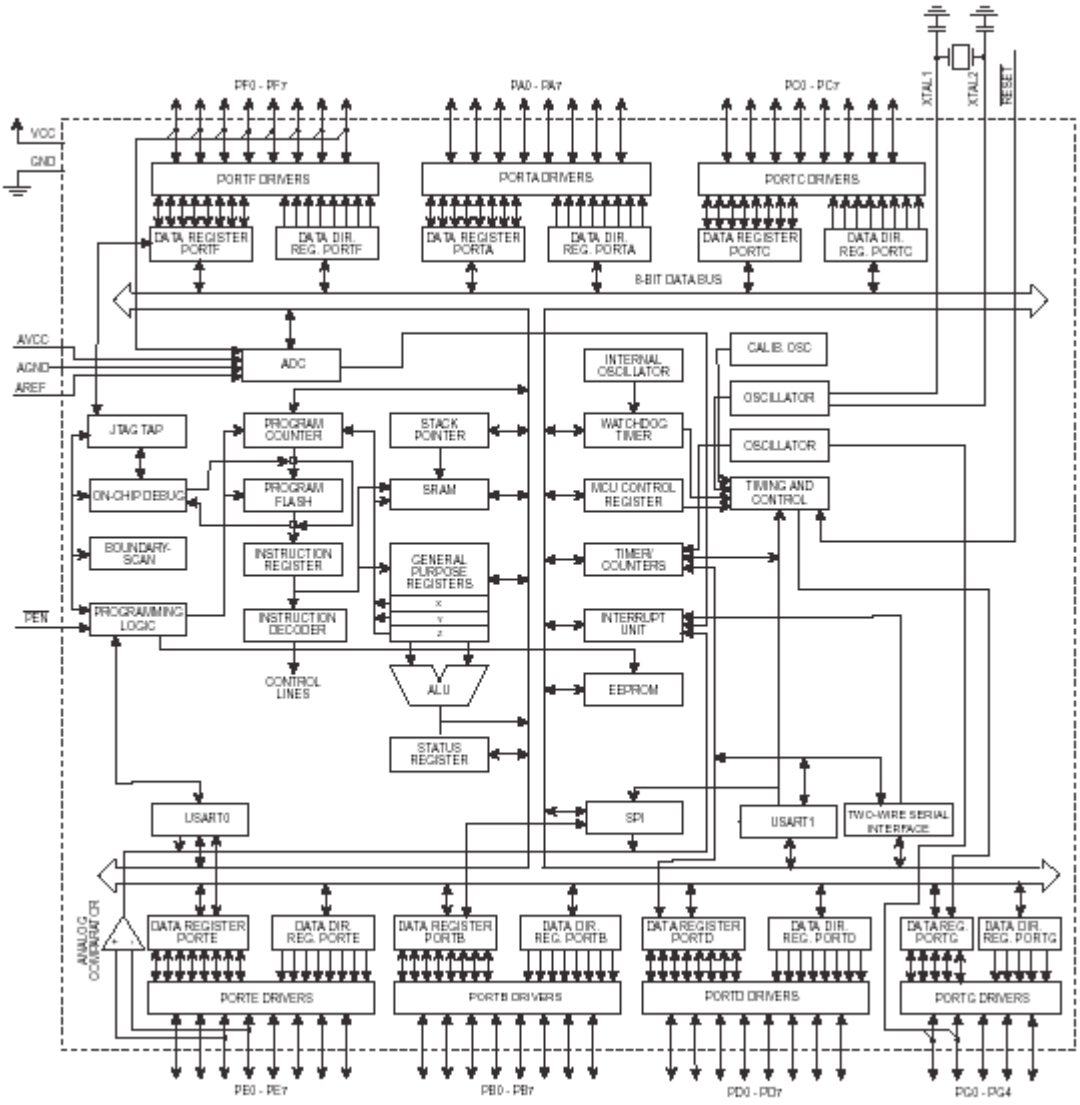


Figura 5.1 – 2: Diagramas de bloques internos

Descripción de pines.

Los pines que presenta el Atmega128 se pueden dividir en dos grupos, los asociados a puertos y los pines para configuración física del dispositivo (VCC, GND, RESET, PEN, XTAL1, XTAL2). Por otro lado los pines asociados a puertos tienen dos posibles modos de funcionamiento como línea de entrada/salida digital programable por SW o como pin de un determinado periférico interno del microcontrolador. Esto se aprecia en las figuras 5.1 – 1 y 5.1 – 2.

.

Puerto A (PA7 – PA0)

El puerto A es un puerto de 8-bit bidireccional configurable como entrada o como salida. Los pines del puerto pueden ir provistos de resistencias internas pull-up (seleccionables para cada bit). Cada pin del puerto A como salida puede absorber

suficiente intensidad controlar directamente cargas tipo LED. El puerto A sirve como bus de direcciones y datos de entrada/salida cuando se usa una SRAM externa. Los pines del puerto A se hallan en tri-estado durante la condición de reset.

Puerto B (PB7 – PB0)

El puerto B presenta las mismas características que el puerto A en cuanto a líneas de entrada / salida digitales. Las otras funcionalidades alternativas que presenta en cada pin son:

- **PB7 OC2/OC1C** (Output Compare and PWM Output for Timer/Counter2 or Output Compare and PWM Output C for Timer/Counter1)
- **PB6 OC1B** (Output Compare and PWM Output B for Timer/Counter1)
- **PB5 OC1A** (Output Compare and PWM Output A for Timer/Counter1)
- **PB4 OC0** (Output Compare and PWM Output for Timer/Counter0)
- **PB3 MISO** (SPI Bus Master Input/Slave Output)
- **PB2 MOSI** (SPI Bus Master Output/Slave Input)
- **PB1 SCK** (SPI Bus Serial Clock)
- **PB0 SS** (SPI Slave Select input)

Puerto C (PC7 – PC0)

El puerto C presenta las mismas características que los puertos anteriores para las líneas de entrada/salida digitales. La funcionalidad alternativa que presenta es actuar como byte alto del bus de direcciones en el acceso a una RAM externa.

Puerto D (PD7 – PD0)

El puerto D presenta las mismas características que los puertos anteriores para las líneas de entrada/salida digitales. La funcionalidad alternativa de los pines de este puerto es:

- **PD7 T2** (Timer/Counter2 Clock Input)
- **PD6 T1** (Timer/Counter1 Clock Input)
- **PD5 XCK1** (USART1 External Clock Input/Output)
- **PD4 IC1** (Timer/Counter1 Input Capture Trigger)
- **PD3 INT3/TXD1** (External Interrupt3 Input or UART1 Transmit Pin)
- **PD2 INT2/RXD1** (External Interrupt2 Input or UART1 Receive Pin)
- **PD1 INT1/SDA** (External Interrupt1 Input or TWI Serial DAta)
- **PD0 INT0/SCL** (External Interrupt0 Input or TWI Serial CLock)

Puerto E (PE7 – PE0)

El puerto E presenta las mismas características que los puertos anteriores para las líneas de entrada/salida digitales. La funcionalidad alternativa de los pines de este puerto es:

- **PE7 INT7/IC3** (External Interrupt 7 Input or Timer/Counter3 Input Capture Trigger)
- **PE6 INT6/ T3** (External Interrupt 6 Input or Timer/Counter3 Clock Input)

- **PE5 INT5/OC3C** (External Interrupt 5 Input or Output Compare and PWM Output C for Timer/Counter3)
- **PE4 INT4/OC3B** (External Interrupt4 Input or Output Compare and PWM Output B for Timer/Counter3)
- **PE3 AIN1/OC3A** (Analog Comparator Negative Input or Output Compare and PWM Output A for Timer/Counter3)
- **PE2 AIN0/XCK0** (Analog Comparator Positive Input or USART0 external clock input/output)
- **PE1 PDO/TXD0** (Programming Data Output or UART0 Transmit Pin)
- **PE0 PDI/RXD0** (Programming Data Input or UART0 Receive Pin)

Puerto F (PF7 – PF0)
El puerto F presenta las mismas características que los puertos anteriores para las líneas de entrada/salida digitales. La funcionalidad alternativa de los pines de este puerto es como líneas de entrada para el conversor analógico digital o señales del interfaz de depuración JTAG. Se recomienda que durante el proceso de conversión A/D ningún pin de este puerto configurado como salida cambie de valor pues esto perturba la medida.

- **PF7 ADC7/TDI** (ADC input channel 7 or JTAG Test Data Input)
- **PF6 ADC6/TDO** (ADC input channel 6 or JTAG Test Data Output)
- **PF5 ADC5/TMS** (ADC input channel 5 or JTAG Test Mode Select)
- **PF4 ADC4/TCK** (ADC input channel 4 or JTAG Test ClocK)
- **PF3 ADC3** (ADC input channel 3)
- **PF2 ADC2** (ADC input channel 2)
- **PF1 ADC1** (ADC input channel 1)
- **PF0 ADC0** (ADC input channel 0)

Puerto G (PG4 – PG0)
El puerto G presenta las mismas características que los puertos anteriores para las líneas de entrada/salida digitales. Presenta dos funcionalidades alternativas, por un lado las líneas de control de acceso a la memoria externa y por otro dos pines para poder conectar un cristal para aplicaciones donde se necesite RTC.

- **PG4 TOSC1** (RTC Oscillator Timer/Counter0)
- **PG3 TOSC2** (RTC Oscillator Timer/Counter0)
- **PG2 ALE** (Address Latch Enable to external memory)
- **PG1 RD** (Read strobe to external memory)
- **PG0 WR** (Write strobe to external memory)

VCC
Voltaje de alimentación.

GND
Tierra.

RESET
Entrada de reset. Una señal por nivel bajo durante dos ciclos máquina en este pin

mientras el oscilador está funcionando y el microcontrolador se reseteará.

XTAL1
Entrada del oscilador y entrada al circuito que opera con el reloj interno.

XTAL2
Salida del oscilador.

AVCC
Tensión de alimentación para el puerto F y el conversor analógico digital. Debe ser conectado externamente a VCC, incluso si no es usado el ADC. Si usa al ADC se debe conectar a VCC a traves de un filtro paso bajo.

AREF
Referencia de tensión analógica para el ADC.

PEN (Programming ENable)
Habilitación de la programación a través del SPI. Este pin debe estar a nivel bajo durante un reset para iniciar la programación SPI.
.

Arquitectura

Para maximizar el rendimiento, paralelismo y ejecución la familia AVR emplea una arquitectura Harvard con memorias y buses distintos para programa y datos. Las instrucciones de la memoria de programa son ejecutadas con un nivel simple de entrelazado o solapamiento (pipelining). Mientras una instrucción esta siendo ejecutada la siguiente esta siendo tomada de la memoria de programa y preparada para su ejecución. Este concepto permite que se ejecute una instrucción cada ciclo de reloj pese a que cada instrucción necesita de dos ciclos de reloj.

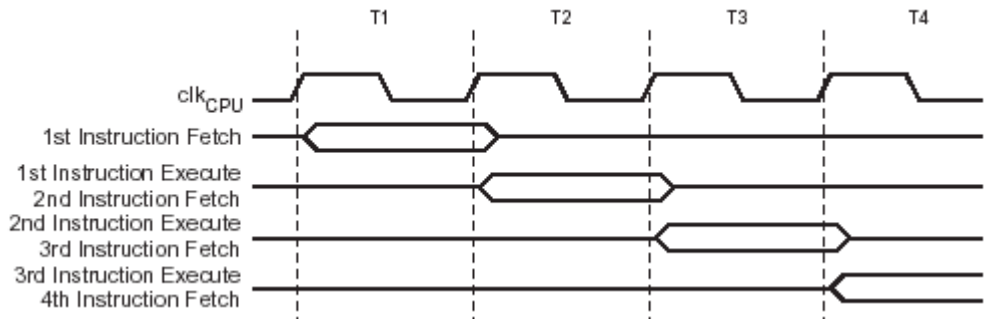


Figura 5.1- 3: Ciclos “extrae – ejecuta”

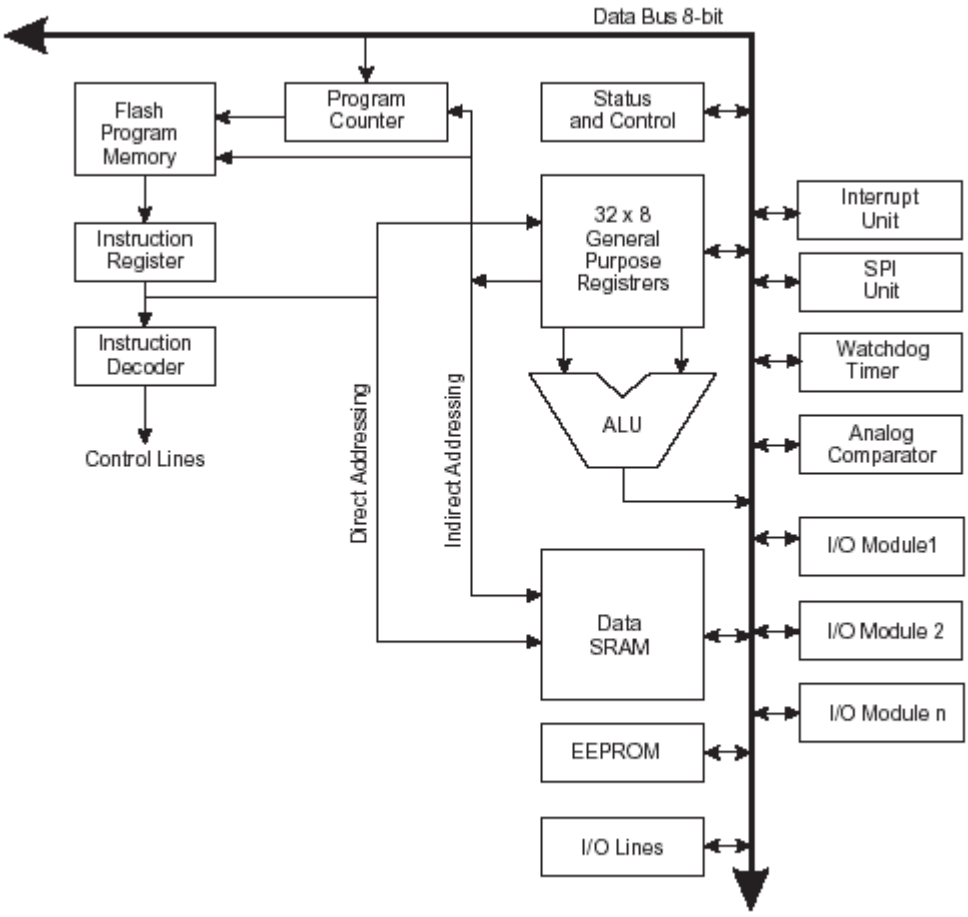


Figura 5.1- 4: Arquitectura interna

El Atmega128 posee 32 registros de trabajo de propósito general con un tiempo de acceso de un ciclo de reloj. Esto significa que en tan solo un ciclo de reloj, una operación de la ALU (unidad aritmético lógica) es ejecutada. Seis de los 32 registros de trabajo pueden usarse como 3 registros de 16 bits que sirven como punteros de direccionamiento indirecto para direccionar el espacio de datos. Estos registros con funciones especiales son el registro X, el Y y el Z.

La ALU soporta operaciones aritméticas y lógicas entre registros o entre un registro y una constante. Las operaciones simples con registros son también ejecutadas por la ALU. Los registros de trabajo están colocados en las 32 primeras posiciones de la memoria de datos ($00 - $1F), permitiendo su acceso como si fueran lugares de memoria convencionales. El espacio de memoria contiene 64 direcciones para las funciones propias de la CPU como son los registros de Control, los Contadores/Timer, conversores A/D, y otras funciones. Estos registros están colocados a continuación de los de trabajo, de la dirección $20 a la $5F. Particularmente el ATmega 128 tiene un espacio extendido de entradas/salidas de $60 a $FF.

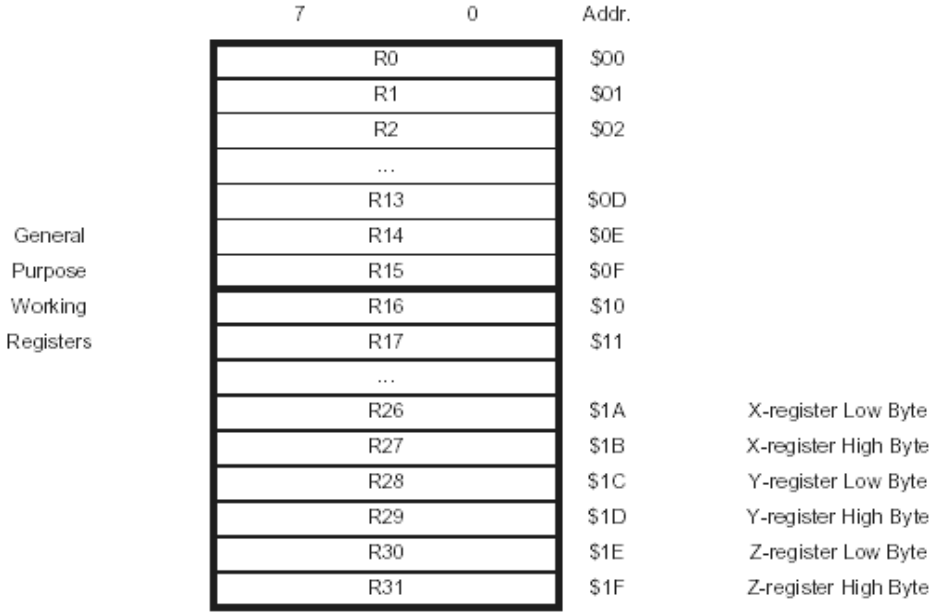


Figura 5.1 – 5: Registros de propósito general.

Con las instrucciones de salto incondicional y con la de llamada, se puede acceder directamente a toda la memoria. Las instrucciones son de 16 o 32 bits, por este motivo los 128 Kbytes de memoria de Flash están estructurados en 64 Kwords de 2 bytes por word. La memoria Flash está dividida en memoria de programa de aplicación y memoria de arranque 'boot'.

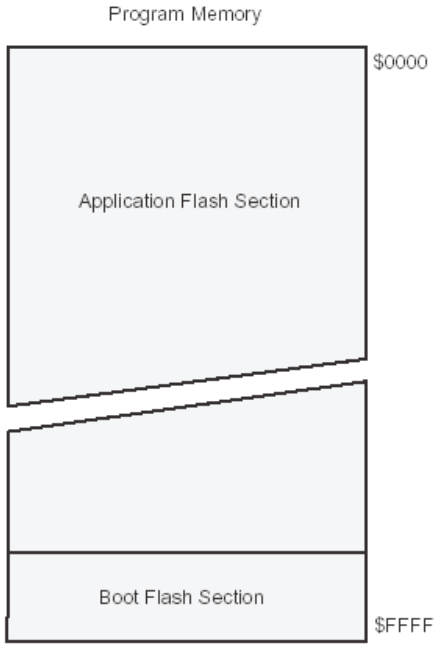


Figura 5.1 – 6: Memoria de programa.

La memoria SRAM es fácilmente accesible a través de los cinco modos de direccionamiento que permite la arquitectura AVR. El espacio de memoria en la arquitectura AVR está formado por mapas lineales y regulares. En el caso del ATmega128 presenta dos posibles mapas de memoria, según si se va a configurar el ATmega128 en modo normal o en modo de compatibilidad con ATmega163 (que es un modelo anterior y que tiene menos prestaciones). La diferencia es que en el modo de compatibilidad con ATmega163 se anulan ciertos periféricos del ATmega128 que no están presentes en el modelo ATmega163 por lo que desaparece el espacio extendido de direcciones E/S.

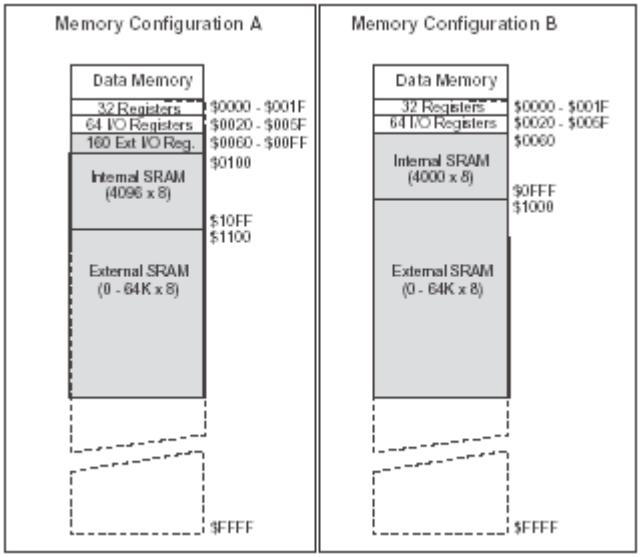


Figura 5.1 – 5: Memoria SRAM

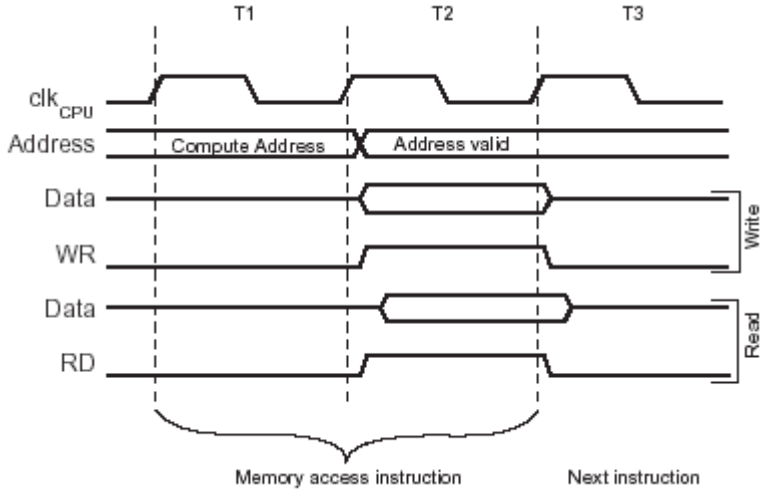


Figura 5.1 – 6: Tiempo de acceso a SRAM interna.

La memoria EEPROM consta de 4 Kbytes y está organizada como un espacio de datos separado.

Acceso a memoria externa.

El interfaz de memoria externa es muy importante en este proyecto pese a que el ATmega tiene memoria RAM interna de sobra para cumplir las especificaciones. Esto se debe a que el controlador CAN es accesible como si fuera una memoria externa a la que se le entrega una dirección y unas señales de control para obtener el byte almacenado en esa dirección.

El interfaz consta de las siguientes señales:

- AD7:0: Bus de direcciones/datos multiplexado
- A15:8: Byte alto del bus de direcciones (se puede configurar solo una parte)
- ALE: Address latch enable.
- RD: Read strobe.
- WR: Write strobe.

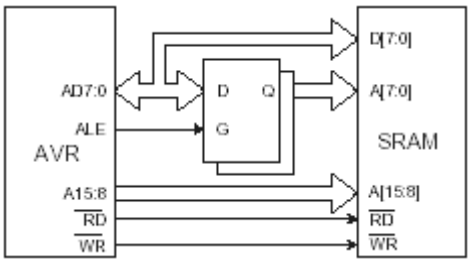


Figura 5.1 - 7: Acceso a memoria externa.

El controlador CAN de Philips que se va emplear tiene incluido el latch de direcciones. Y se puede configurar con microcontroladores tipo Intel como los AVR o del tipo Motorola. Como solo se va a tener un periférico de este tipo la señal Chip Select del controlador CAN se puede dejar habilitada de manera permanente. En el microcontrolador hay que configurar el espacio donde va a ir ubicada esa memoria y los ciclos de espera si son necesarios.

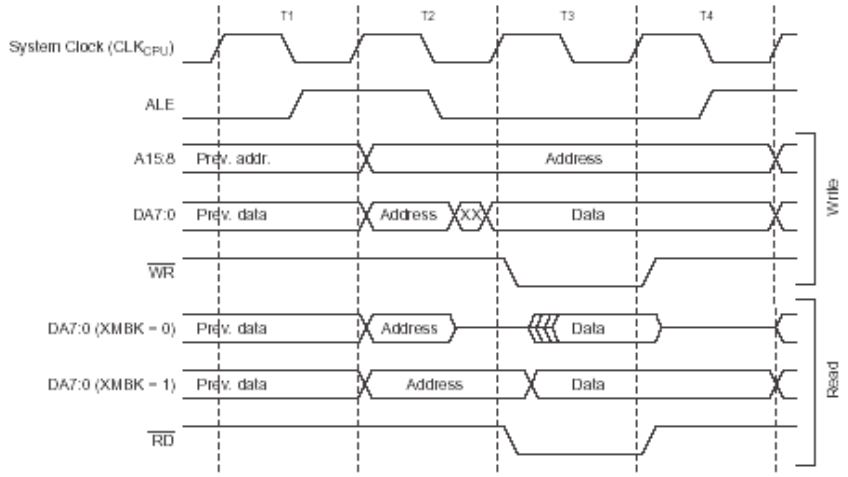


Figura 5.1 –8: Acceso a memoria externa sin ciclo de espera.

La configuración de este interfaz se realiza en los registros MCUCR que es un registro de control general y en los registros XMCRA y xMCRB que son específicos de la memoria.

Bit	7	6	5	4	3	2	1	0	
	SRE	SRW10	SE	SM1	SM0	SM2	IVSEL	IVCE	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	–	SRL2	SRL1	SRL0	SRW01	SRW00	SRW11	–	XMCRA
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	XMBK	–	–	–	–	XMM2	XMM1	XMM0	XMCRB
Read/Write	R/W	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 5.1 –9: Registros para la configuración de la memoria externa.

El bit 7 de MCUCR SER es el habilitador del interfaz de memoria externa y desactiva los pines del puerto A como E/S digitales para pasar a la funcionalidad de bus de datos/direcciones multiplexado.

Los bits SLR2:0 seleccionan como se puede dividir la memoria para tener espacios con distintos ciclos de espera.

SRL2	SRL1	SRL0	Sector Limits
0	0	0	Lower sector = N/A Upper sector = 0x1100 - 0xFFFF
0	0	1	Lower sector = 0x1100 - 0x1FFF Upper sector = 0x2000 - 0xFFFF
0	1	0	Lower sector = 0x1100 - 0x3FFF Upper sector = 0x4000 - 0xFFFF
0	1	1	Lower sector = 0x1100 - 0x5FFF Upper sector = 0x6000 - 0xFFFF
1	0	0	Lower sector = 0x1100 - 0x7FFF Upper sector = 0x8000 - 0xFFFF
1	0	1	Lower sector = 0x1100 - 0x9FFF Upper sector = 0xA000 - 0xFFFF
1	1	0	Lower sector = 0x1100 - 0xBFFF Upper sector = 0xC000 - 0xFFFF
1	1	1	Lower sector = 0x1100 - 0xDFFF Upper sector = 0xE000 - 0xFFFF

Figura 5.1 – 10: Segmentos posibles para distintos ciclos de espera.

Con los bits SRWn1:0 se selecciona el número de ciclos de espera para cada uno de los segmentos (n=0 para el segmento bajo y n=1 para el segmento alto).

SRWn1	SRWn0	Wait States
0	0	No wait-states
0	1	Wait one cycle during read/write strobe
1	0	Wait two cycles during read/write strobe
1	1	Wait two cycles during read/write and wait one cycle before driving out new address

Figura 5.1 – 11: Ciclos de espera.

En el registro de control XMCRB se configuran que bits del byte alto del bus de direcciones se necesitan para direccionar la memoria externa, liberando así el resto para E/S digitales.

XMM2	XMM1	XMM0	# Bits for External Memory Address	Released Port Pins
0	0	0	8 (Full 60 KB space)	None
0	0	1	7	PC7
0	1	0	6	PC7 - PC6
0	1	1	5	PC7 - PC5
1	0	0	4	PC7 - PC4
1	0	1	3	PC7 - PC3
1	1	0	2	PC7 - PC2
1	1	1	No Address high bits	Full Port C

Figura 5.1 – 12: Selección de bits de direccionamiento del byte alto.

Registro de estado.

El registro de estado almacena información concerniente a las ultimas operaciones realizadas. Esta información se emplea para alterar el flujo del programa. La información de este registro se modifica cada vez que se termina alguna operación en la unidad aritmético lógica.

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 5.1 – 13: Registro de estado

Todos los bits de este registro son de escritura o lectura. El valor inicial de este

registro es 0x00. La descripción de los bits es la siguiente:

- **Bit 7-I: Global Interrupt Enable**

El bit 7 permite habilitar las interrupciones cuando está a 1. El control individual de las interrupciones es configurado en los registros asociados a interrupciones externas (EICR), interrupciones del temporizador (TIMSK), y bits de habilitación de interrupciones en cada uno de los periféricos. Si este bit está a 0 todas las interrupciones están deshabilitadas independientemente del estado de los registros y bits anteriormente citados. Este bit es puesto a 0 vía hardware en el momento en el que se produce una interrupción y es puesto de nuevo a uno tras la instrucción RETI que sirve para producir el retorno de interrupción.

- **Bit 6-T: Bit Copy Storage**

Las instrucciones para copiar bits BLD (Bit LoaD) y BST (Bit STore) utilizan el bit T como fuente o como destino para un bit. Un bit de un registro del archivo de registros puede ser guardado en el bit T mediante la intrucción BST, y un bit en T puede ser copiado en un bit de un registro del archivo de registros mediante la instrucción BLD.

- **Bit 5-H: Half Carry Flag**

Este flag indica que se ha producido un half carry en algunas operaciones aritméticas. Ver la descripción del set de instrucciones para una información detallada.

- **Bit 4-S: Sign Bit, S = N XOR V**

El bit-S realiza una operación exclusiva entre los bits N y el bit V del registro estado. Ver la decripción del set de instrucciones para una información detallada.

- **Bit 3-V: Two´s Complement Overflow Flag**

El flag V se pone a 1 al producirse overflow al realizar operaciones aritméticas de complento a dos. Ver la descripción del set de instrucciones para una información detallada.

- **Bit 2-N: Negative Flag**

El flag N indica un resultado negativo tras realizar una operación aritmética o una lógica. Ver la descripción del set de instrucciones para una información detallada.

- **Bit 1-Z: Zero Flag**

El flag Z indica un resultado de 0 tras realizar una operación aritmética o una lógica. Ver la descripción del set de instrucciones para una información detallada.

- **Bit 0-C: Carry Flag**

Este flag indica un carry tras realizar una operación aritmética o una lógica. Ver el set de instrucciones para una información detallada.

Puntero de la pila

El Stack Pointer (puntero de la pila) de la tecnología AVR es de 16 bits y está construido sobre dos registros de 8 bits. Dado que el microcontrolador ATmega128 soporta 64 kB de memoria externa SRAM, los 16 bits son utilizados:

Bit	15	14	13	12	11	10	9	8	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Figura 5.1 – 14: Registro Stack Pointer

Este puntero debe señalar posiciones de memoria SRAM por encima de 0x60. La pila crece hacia direcciones de memoria bajas conforme se introducen datos en esta.

5.2 Dispositivos y configuración.

En esta se sección se van a describir los diferentes periféricos y componentes del microcontrolador que se van a usar en este proyecto.

5.2.1 Puertos.

Cada puerto del microcontrolador tiene una serie de pines que se pueden configurar como E/S digitales. Para configurar cada pin del puerto es necesario utilizar tres registros (**PORTx**, **DDRx** y **PINx**) donde x es la letra de cada uno de los puertos **(A,B,C,D,E,F** y **G).**

Mediante el registro **DDRx** puede configurarse cada pin del puerto x individualmente como entrada o como salida. Para configurar un pin como salida hay que escribir un cero y para configurarlo como entrada un uno. Del mismo modo, todos los pines del puerto tienen resistencias de pull-up seleccionables individualmente. Cuando un pin está configurado como entrada y su valor en el registro **PORTx** es uno la resistencia de pull-up queda activada. Para desconectar la resistencia de pull-up debe ponerse un cero en el bit correspondiente del registro PORTx o configurarse el pin como una salida.

DDxn	PORTxn	PUD (in SFIOR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Figura 5.2.1 – 1: Configuración de E/S digitales.

En el caso del puerto A se tiene que cuando el puerto queda configurado como entrada el valor de la entrada es guardado en el registro PINA. Del mismo modo cuando es configurado como salida y se quiere enviar un dato a través de ella, éste debe ser puesto en el registro PORTx y no en el PINx (sólo para entradas).

PORTA – Port A Data Register.

7	6	5	4	3	2	1	0
PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

DDRA – Port A Data Direction Register.

7	6	5	4	3	2	1	0
DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

PINA – Port A Input Pins Address

7	6	5	4	3	2	1	0
PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
R	R	R	R	R	R	R	R

Figura 5.2.1 – 2: Registros asociados a E/S digitales.

Hay que recordar que los pines de cada puerto tienen otras funciones según el periférico que tengan asociado y que no se pueden usar a la vez ambas funcionalidades.

5.2.2 USARTs.

El puerto serie es muy importante es esta aplicación pues es que se comunica con el módulo de radio frecuencia y se busca la mayor velocidad posible en esta conexión. Es tal la importancia de este dispositivo que es este dispositivo el que marca cual va a ser la frecuencia de reloj del sistema.

El microcontrolador ATmega128 dispone de dos unidades USART independientes que pueden trabajar cada una como un puerto serie asíncrono o síncrono.

Las principales características de estos dispositivos son:

- Operación full-duplex, recepción y transmisión independientes.
- Operación síncrona o asíncrona.
- Funcionamiento como maestro o esclavo en modo síncrono.
- Generador de tasas de transmisión de elevada precisión.
- Tramas flexibles de 5,6,7,8 o 9 bits de datos y 1 o 2 bits de parada.
- Generación de paridad par o impar y comprobador de paridad hardware.
- Detección de perdida de datos.
- Filtrado de ruido incluyendo detección de falso bit de inicio y un filtro digital paso bajo.
- Tres interrupciones independientes en los eventos fin de transmisión, registro de datos de transmisión vacío y recepción completa.
- Modo para comunicación multiprocesador.
- Modo de doble velocidad asíncrona (permite mayor velocidad reduciendo el numero de muestras por bit).

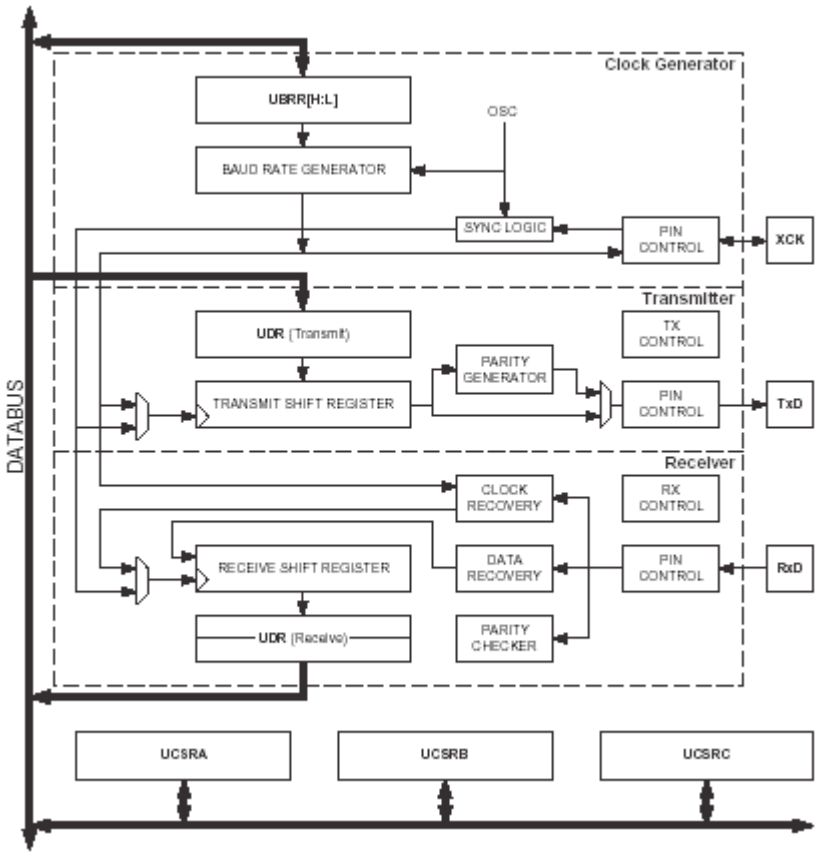


Figura 5.2.2 – 1: Diagrama de bloques del dispositivo USART

Síntesis de la tasa de transmisión.

La tasa de transmisión de la UART viene determinada por el reloj del sistema y el valor de un registro llamado UBBR (Usart Baud Rate Register). El reloj del sistema actúa sobre un contador que es comparado con el valor del registro UBBR para generar una señal de reloj que define la tasa de transmisión de la UART. En recepción se dispone de una unidad para recuperar la señal de reloj del dato entrante y sincronizar el reloj interno a las tramas entrantes.

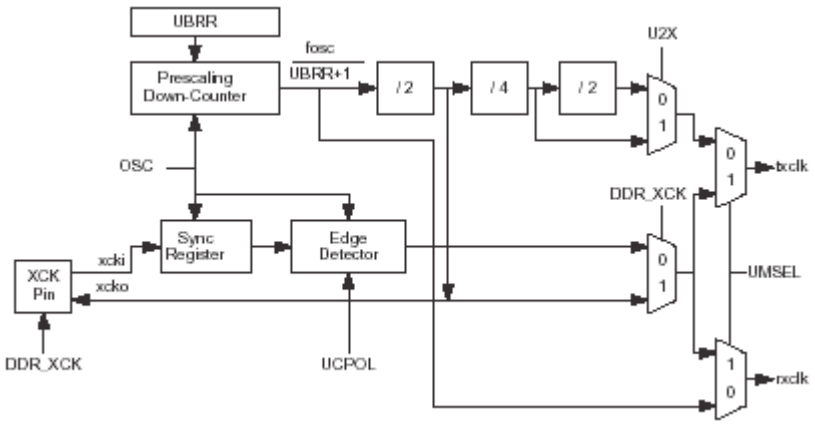


Figura 5.2.2 - 2: Generación de tasa de transmisión

El valor que hay que introducir en el registro UBRR se determina según la siguiente tabla dada por el fabricante. Dado que UBBR es un numero entero en un registro de 16 bits es posible que la tasa obtenida para no sea exactamente la buscada, esto introduce un error.

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps).

BAUD Baud rate (in bits per second, bps)

f_{OSC} System Oscillator clock frequency

UBRR Contents of the UBRRH and UBRRL Registers, (0 - 4095)

Tabla 5.2.2 –1: Relación entre Baud Rate y UBBR.

El rango de funcionamiento del receptor depende de la similitud entre la tasa recibida y la tasa internamente generada. Si el transmisor envía muy rápido o muy lento, o la tasa de interna del receptor presenta una elevada desviación respecto a la nominal el receptor no será capaz de sincronizar la tramas al bit de inicio de trama. Las siguientes expresiones se emplean para calcular la relación entre la tasa de datos entrante y la tasa de datos interna.

$$R_{slow} = \frac{(D+1)S}{S-1+D\cdot S+S_F} \qquad R_{fast} = \frac{(D+2)S}{(D+1)S+S_M}$$

D Sum of character size and parity size (D = 5 to 10-bit)

S Samples per bit. S = 16 for Normal Speed mode and S = 8 for Double Speed mode.

S_F First sample number used for majority voting. S_F = 8 for Normal Speed and S_F = 4 for Double Speed mode.

S_M Middle sample number used for majority voting. S_M = 9 for Normal Speed and S_M = 5 for Double Speed mode.

R_{slow} is the ratio of the slowest incoming data rate that can be accepted in relation to the receiver baud rate. R_{fast} is the ratio of the fastest incoming data rate that can be accepted in relation to the receiver baud rate.

Figura 5.2.2 - 3: Relación entre la tasa de datos entrante y la tasa de datos interna.

El fabricante sugiere que el error no supere un determinado valor porcentual, que para el caso de 8 bits sin paridad es de un 2.0%.

Table 75. Recommended Maximum Receiver Baud Rate Error for Normal Speed Mode (U2X = 0)

D # (Data+Parity Bit)	R_{slow} %	R_{fast} %	Max Total Error %	Recommended Max Receiver Error %
5	93,20	106,67	+6.67/-6.8	± 3.0
6	94,12	105,79	+5.79/-5.88	± 2.5
7	94,81	105,11	+5.11/-5.19	± 2.0
8	95,36	104,58	+4.58/-4.54	± 2.0
9	95,81	104,14	+4.14/-4.19	± 1.5
10	96,17	103,78 %	+3.78/-3.83	± 1.5

Table 76. Recommended Maximum Receiver Baud Rate Error for Double Speed Mode (U2X = 1)

D # (Data+Parity Bit)	R_{slow} %	R_{fast} %	Max Total Error %	Recommended Max Receiver Error %
5	94,12	105,66	+5.66/-5.88	± 2.5
6	94,92	104,92	+4.92/-5.08	± 2.0
7	95,52	104,35	+4.35/-4.48	± 1.5
8	96,00	103,90	+3.90/-4.00	± 1.5
9	96,39	103,53	+3.53/-3.61	± 1.5
10	96,70	103,23	+3.23/-3.30	± 1.0

Tabla 5.2.2 -2: Error máximo recomendado en el receptor

El fabricante entrega tablas con los cálculos para obtener las tasas de datos típicas a partir de las frecuencias mas comunes de cristales en el mercado. De estas tabla es de donde se deduce que es preferible trabajar con cristales cuyas frecuencias no sean múltiplos de 1MHz, pues no se consiguen con errores bajos. En este caso se aprecia claramente como es preferible trabajar con cristales de 11.0592 MHz o 14.7456 MHz que con uno 16 MHz. Se busca un cristal que tengo un error para las tasas de datos que se van a emplear previsiblemente que son 115200 bps o 230400 bps, en el caso del cristal de 16 MHz es justo en esas tasas donde el error en la generación de la tasa de datos es mayor alcanzando hasta 7.8 %.

$$\text{Error}[\%] = \left(\frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}} - 1\right) \bullet 100\%$$

Baud Rate (bps)	f_{osc} = 8.0000 MHz				f_{osc} = 11.0592 MHz				f_{osc} = 14.7456 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	–	–	2	-7.8%	1	-7.8%	3	-7.8%
1M	–	–	0	0.0%	–	–	–	–	0	-7.8%	1	-7.8%
Max (1)	0.5 Mbps		1 Mbps		691.2 kbps		1.3824 Mbps		921.6 kbps		1.8432 Mbps	

1. UBRR = 0, Error = 0.0%

Baud Rate (bps)	f_{osc} = 16.0000 MHz				f_{osc} = 18.4320 MHz				f_{osc} = 20.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	–	–	4	-7.8%	–	–	4	0.0%
1M	0	0.0%	1	0.0%	–	–	–	–	–	–	–	–
Max (1)	1 Mbps		2 Mbps		1.152 Mbps		2.304 Mbps		1.25 Mbps		2.5 Mbps	

.

Tabla 5.2.2 -3: Error en la tasa de datos para distintas configuraciones UBBR – Fclk

Registros asociados al puerto serie

UDR – Uart Data Register

Este registro es permite introducir caracteres para que sean transmitidos y extraer los caracteres recibidos. Actúa como interfaz pues internamente existen búferes separados para transmisión y recepción.

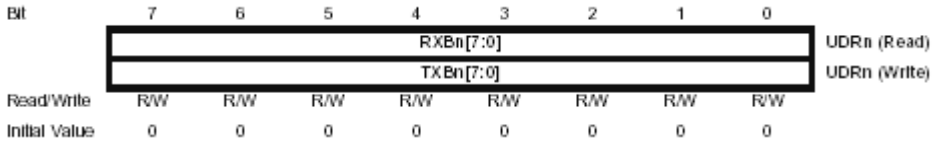


Figura 5.2.2 - 4: Registro de datos UDR

UCSRA – Uart Control and Status Register A

Este registro contiene flags que indican el estado del dispositivo y bits de configuración para el control.

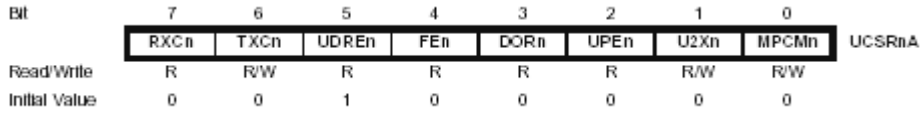


Figura 5.2.2 - 5: Registro de control UCSRA

Los siguientes bits indican el estado del dispositivo en funcionamiento normal sin errores, reportando información en cuanto a si se ha recibido un nuevo carácter, se ha terminado de transmitir y si esta vacío en registro de datos.

- **Bit 7 – RXCn: USART Receive Complete**
- **Bit 6 – TXCn: USART Transmit Complete**
- **Bit 5 – UDREn: USART Data Register Empty**

Los bits 4,3 y 2 avisan de condiciones de error. El bit 4 advierte que error en la trama. El bit 3 de que se ha perdido un dato porque ha sido sobrescrito por otro carácter entrante. El bit 2 informa de error en la paridad si esta comprobación esta habilitada por el bit UPM.

- **Bit 4 – FEn: Frame Error**
- **Bit 3 – DORn: Data OverRun**
- **Bit 2 – UPEn: Parity Error**

El bit 1 selecciona el número de muestras empleando en recepción para el procesamiento de los caracteres recibidos.

- **Bit 1 – U2Xn: Double the USART Transmission Speed**

El bit 0 es para habilitar el modo de comunicación multiprocesador.

- **Bit 0 – MPCMn: Multi-Processor Communication Mode**

UCSRB – Uart Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 5.2.2 - 6: Registro de control UCSRB

- **Bit 7 – RXCIEn: RX Complete Interrupt Enable**
- **Bit 6 – TXCIE: TX Complete Interrupt Enable**
- **Bit 5 – UDRIEn: USART Data Register Empty Interrupt Enable**

Los bits 7, 6 y 5 son bits habilitadores de las posibles interrupciones que puede generar la UART en los eventos recepción completa, transmisión completa y registro de datos vacío. Esta habilitación está supeditada a que el bit de habilitación global de interrupciones en el registro SREG esté habilitado.

- **Bit 4 – RXENn: Receiver Enable**
- **Bit 3 – TXENn: Transmitter Enable**

Los bits 4 y 3 habilitan separadamente las funcionalidades de recepción y transmisión de la UART.

- **Bit 2 – UCSZn2: Character Size**

El bit 2 se usa junto con los bits UCSZn1:0 del registro UCRSC para definir el número de bits que componen el carácter.

- **Bit 1 – RXB8n: Receive Data Bit**
- **Bit 0 – TXB8n: Transmit Data Bit 8**

Los bits 1 y 0 son una extensión del registro UDR cuando se emplean caracteres de 9 bits.

UCSRC – Uart Control and Status Register C

Bit	7	6	5	4	3	2	1	0	
	–	UMSELn	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

Figura 5.2.2 - 7: Registro de control UCSRC

El bit 7 está reservado para aplicaciones futuras, y debe escribirse como cero en los accesos de escritura al registro UCSRC

- **Bit 6 – UMSELn: USART Mode Select**

  El bit 6 selecciona el modo de trabajo sincrono/asíncrono de la USART

UMSELn	Mode
0	Asynchronous Operation
1	Synchronous Operation

Tabla 5.2.2 - 4: Configuración de modo síncrono/asíncrono.

- **Bit 5:4 – UPMn1:0: Parity Mode**

  Los bits 5 y 4 configuran si se emplea paridad y que tipo de paridad. El transmisor genera automáticamente la paridad, y el receptor calcula la paridad del carácter recibido y la compara según el valor delos bits UPM1:0.

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	(Reserved)
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Tabla 5.2.2 - 5: Configuración de paridad.

• Bit 3 – **USBSn**: Stop Bit Select

El bit 3 indica si se usan uno o dos bits de parada.

USBSn	Stop Bit(s)
0	1-bit
1	2-bits

Tabla 5.2.2 - 6: Configuración de bit de parada.

- **Bit 2:1 – UCSZn1:0: Character Size**

Los bits 2 y 1 definen junto con el bit UCSZ2 de UCSRB el número de bits que conforman el carácter

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
UCSZn2	UCSZn1	UCSZn0	Character Size
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Tabla 5.2.2 - 7: Configuración de número de bits.

- **Bit 0 – UCPOLn: Clock Polarity**

El bit 0 solo se utiliza en el modo síncrono para indicar cuando se produce el cambio de la entrada y salida respecto al reloj.

UBRR – Uart Baud Rate Register

Como ya se ha visto en este registro se escribe el valor que se necesite para obtener la tasa de datos deseada. Es un registro de formado por 2 bytes UBRRH y UBRRL, pero solo se usan los 12 bits menos significativos.

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Figura 5.2.2 – 8: Registro de configuración de tasa de datos.

5.2.3 Interrupciones.

El microcontrolador ATmega128 posee diversas fuentes de interrupción. Todas las interrupciones tienen un vector de interrupción propio en el espacio de memoria del programa y de unos bits que deben ser puestos a uno conjuntamente con el bit I del registro estado para que las interrupciones puedan producirse.

La ejecución de la respuesta a una interrupción es un proceso que tiene una duración mínima de 4 ciclos de reloj. Después de estos 4 ciclos de reloj el programa ejecuta el código de programa correspondiente a la interrupción producida. Durante este período de 4 ciclos de reloj, el Contador de Programa (2 bytes) es colocado en el Stack, y el Stack Pointer es decrementado en dos unidades. Tras producirse una interrupción la instrucción que se ejecutará en el vector correspondiente será una de salto RJMP a la rutina de código correspondiente y esta es una instrucción que tarda dos ciclos de reloj. Si una interrupción es producida mientras se está ejecutando una instrucción multiciclo se esperará a que la instrucción termine de ser ejecutada antes de atender a la interrupción.

El retorno de una rutina de interrupción dura 4 ciclos de reloj. Durante estos 4 ciclos de reloj el Contador de Programa (2 bytes) es nuevamente cargado con su valor del Stack y el Stack Pointer es incrementado en dos unidades. Cuando el microcontrolador sale de la ejecución del código de interrupción, vuelve al programa principal y ejecuta exactamente la instrucción que se encontraba antes de que se produjera la interrupción.

Vector No.	Program Address[2]	Source	Interrupt Definition
1	$0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	TIMER2 COMP	Timer/Counter2 Compare Match
11	$0014	TIMER2 OVF	Timer/Counter2 Overflow
12	$0016	TIMER1 CAPT	Timer/Counter1 Capture Event
13	$0018	TIMER1 COMPA	Timer/Counter1 Compare Match A
14	$001A	TIMER1 COMPB	Timer/Counter1 Compare Match B
15	$001C	TIMER1 OVF	Timer/Counter1 Overflow
16	$001E	TIMER0 COMP	Timer/Counter0 Compare Match
17	$0020	TIMER0 OVF	Timer/Counter0 Overflow
18	$0022	SPI, STC	SPI Serial Transfer Complete
19	$0024	USART0, RX	USART0, Rx Complete
20	$0026	USART0, UDRE	USART0 Data Register Empty
21	$0028	USART0, TX	USART0, Tx Complete
22	$002A	ADC	ADC Conversion Complete
23	$002C	EE READY	EEPROM Ready
24	$002E	ANALOG COMP	Analog Comparator
25	$0030[3]	TIMER1 COMPC	Timer/Counter1 Compare Match C
26	$0032[3]	TIMER3 CAPT	Timer/Counter3 Capture Event
27	$0034[3]	TIMER3 COMPA	Timer/Counter3 Compare Match A
28	$0036[3]	TIMER3 COMPB	Timer/Counter3 Compare Match B
29	$0038[3]	TIMER3 COMPC	Timer/Counter3 Compare Match C
30	$003A[3]	TIMER3 OVF	Timer/Counter3 Overflow

Tabla 5.2.3 – 1: Vectores de interrupción

Las direcciones más bajas del espacio de memoria de programa son automáticamente definidas para el vector de Reset y para los vectores de interrupción (a no ser que se trabaje en modo bootload). La lista completa de vectores se muestra en la siguiente tabla. Esta lista también determina los niveles de prioridad de las diferentes interrupciones. La mayor prioridad la tiene el vector de RESET y a continuación el vector INT0 y así sucesivamente.

En este punto se va describir como se configuran las interrupciones externas con los registros EICRA y EICRB. Las interrupciones asociadas a los temporizadores se ven en el siguiente punto y las asociadas a la UART se vieron en el anterior.

Interrupciones Externas

Se dispone de 8 líneas de interrupción externa en los pines **INT0:INT7**. Un detalle interesante es que estas líneas se pueden configurar también como salidas con lo que se tiene un procedimiento para provocar interrupciones SW. Los eventos que pueden generar una interrupción externa son un nivel bajo, un flanco de subida, un flanco de bajada o ambos. Esto se configura con los bits **ISC (Interrupt Source Capture)** de los registros **EICRA** y **EICRB** para cada una de las líneas.

Bit	7	6	5	4	3	2	1	0	
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 5.2.3 – 1: Registros de configuración de interrupciones externas.

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Reserved
1	0	The falling edge of INTn generates asynchronously an interrupt request.
1	1	The rising edge of INTn generates asynchronously an interrupt request.

Note: 1. n = 3, 2, 1or 0.

Symbol	Parameter	Condition	Min	Typ	Max	Units
t_{INT}	Minimum pulse width for asynchronous external interrupt			50		ns

Tabla 5.2.3 – 2: Configuración de las fuentes de interrupción asíncronas.

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Any logical change on INTn generates an interrupt request
1	0	The falling edge between two samples of INTn generates an interrupt request.
1	1	The rising edge between two samples of INTn generates an interrupt request.

Note: 1. n = 7, 6, 5 or 4.

Tabla 5.2.3 – 3: Configuración de las fuentes de interrupción externa muestreadas.

Para ensmascarar cada una de estas interrupciones se dispone del registro **EIMSK (External Interrupt MaSK register),** con cuyos bits se pueden habilitar/deshabilitar cada línea de interrupción.

Bit	7	6	5	4	3	2	1	0	
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	IINT0	EIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 5.2.3 – 2: Registro de enmascaramiento de interrupciones externas.

Existe un registro de estado en el que se refleja el estado de las interrupciones externas, **EIFR (External Interrupt Flag Register).**

Bit	7	6	5	4	3	2	1	0	
	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	IINTF0	EIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 5.2.3 – 3: Registro de estado de interrupciones externas.

5.2.4 Temporizadores/Contadores.

El ATmega128 presenta dos Temporizadores/Contadores, dos de 8 bits y dos de 16 bits. Cada temporizador tiene asignado un preescaler individual y pueden ser usado como Timer según la base de tiempos del oscilador interno o como contadores sincronizándose con eventos externos.

Temporizador/Contador de 8 bits

El Timer/Counter de 8 bits puede seleccionar su base de tiempos del reloj interno, del reloj interno con preescaler, a partir de un cristal RTC conectado a los pines

TOC1 y **TOC2** (Timer 0) o de la entrada T2 (Timer 2). Además puede ser activado o detenido y configurado según el registro de control Timer/Counter – **TCCR**. El flag que señaliza el desbordamiento del Timer se encuentra en el registro **TIFR**. Existe la posibilidad de que se produzca una interrupción cuando este Timer se desborda y hay que configurarlo en el registro **TIMSK**.

Cuando el Timer/Counter actúa según un evento externo, la señal externa es sicronizada con la frecuencia del oscilador de la CPU. Por ello para que la señal externa pueda ser contabilizada hay que señalar que el tiempo mínimo entre señal y señal debe ser el de un ciclo de reloj de la CPU.

Asociado a cada temporizador/contador existe uno o varios módulos **OC (Output Compare)** para generar señales de salida de tipo reloj o modulaciones por duración de pulso **PWM (Pulse Width Modulation).**

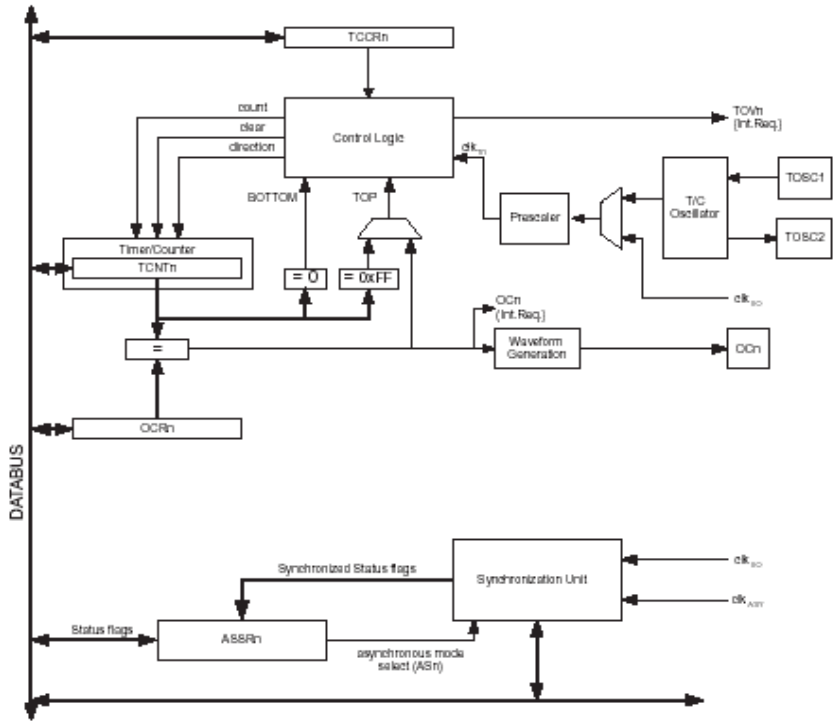


Figura 5.2.4 –1: Constitución interna del Timer/Counter 0.

El Timer0 funciona como un contador ascendente/descendente que va desde el valor 0x00 hasta el valor 0xff. Cuando pasa del valor 0xff de nuevo al 0x00, o viceversa, es cuando se habla de que el Timer se ha desbordado (overflow). En el registro **TCNT0** de Escritura/Lectura es en el que se puede introducir el valor inicial con el que se quiere que el Temporizador realice su cuenta. Así se puede configurar fácilmente para obtener interrupciones periódicas (llamadas **Timer OVerflow**) que sean la base de un reloj o el aviso para realizar determinadas operaciones.

Si se programa el **OCR** este se compara con **TCNT0** generando marcas de tiempo para actuar sobre el módulo generador de onda. También se genera una interrupción (si es habilitada asociada a este evento).

Los registro empleados en la configuración del Timer 0 son:

TCCR0 – Timer Counter Control Register

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 5.2.4 –2: Registro de configuración del Timer/Counter 0.

- **Bit 7 – FOC0: Force Output Compare**

Este bit no se emplea para modo PWM (debe ponerse a cero). Este bit puesto a uno fuerza un evento Output Compare en las condiciones marcadas por el resto de bits de configuración.

- **Bit 6, 3 – WGM01:0: Waveform Generation Mode**

Estos bits controlan el modo de operación del temporizador/contador.

Mode	WGM01[1] (CTC0)	WGM00[1] (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0 at	TOV0 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

Tabla 5.2.4 –1: Modo de trabajo

- **Bit 5:4 – COM01:0: Compare Match Output Mode**

Estos bits controlan el comportamiento del modulo Output compare.

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

Tabla 5.2.4 –2: Configuración de modo OC.

- **Bit 2:0 – CS02:0: Clock Select**

Estos tres bits selecciona la fuente de reloj que actúa sobre el contador a partir del reloj del sitema. La otra posible fuente de reloj es a partir del cristal RTC en los pines TOSC1 y TOSC2 que se configura con el registro ASSR.

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk_{T0S}/(No prescaling)
0	1	0	clk_{T0S}/8 (From prescaler)
0	1	1	clk_{T0S}/32 (From prescaler)
1	0	0	clk_{T0S}/64 (From prescaler)
1	0	1	clk_{T0S}/128 (From prescaler)
1	1	0	clk_{T0S}/256 (From prescaler)
1	1	1	clk_{T0S}/1024 (From prescaler)

Tabla 5.2.4 –3: Selección de prescaler.

TIMSK – Timer Interrupt MaSK register

Este registro contiene los bits que habilitan las distintas fuentes de interrupciones asociadas a los temporizadores/contadores. Los bits asociados al temporizador 0 son los bits 1 y 0 que se usan para habilitar las interrupciones asociadas al evento Output Capture y al evento Timer Overflow.

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 5.2.4 –3: Registro habilitación interrupciones asociadas al timer.

TIFR – Timer Interrupt Flag Register

Este registro presenta los *flags* asociados el estado de los temporizadores /contadores. Los bits que informan del estado del temporizador 0 son los bits 1 y 0, que informan si se ha producido un Output Capture o un Timer Overflow.

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 5.2.4 –3: Registro habilitación interrupciones asociadas al timer.

El temporizador 2 presenta un funcionamiento similar al del temporizador 0, aunque no admite la señal de reloj externa del cristal RTC, si no una señal a partir de la entrada T2.

Temporizador/Contador de 16 bits

El microcontrolador ATmega128 dispone de dos temporizadores/contadores de 16 bits. Cada uno puede seleccionar un preescaler independient.. Su funcionamiento

puede ser habilitado o interrumpido mediante la configuración que se realiza en los registros de control **TCCRA**, **TCCRB**, **TCCRC**. Presentan el modo de funcionamiento Input Capture. Con estos registros también puede seleccionarse los distintos modos de funcionamiento que puede tener este dispositivo como el de comparación de datos y de eventos, PWM, overflow, etc.

Cada uno de estos temporizadores es capaz de generar 5 interrupciones:

- **TOn** Timer OverFlow
- **ICn** Input Capture
- **OCnA** Output Capture A
- **OCnB** Output Capture B
- **OCnC** Output Capture C

El esquema de trabajo con estos temporizadores es similar a los de 8 bits, aunque con diferentes registros dado el amplio numero de interrupciones que maneja cada uno de los temporizadores. En los registros TCCRA, TCCRB y TCCRC se procede a la configuración. Se va a comentar conjuntamente TCCRA y TCCRB pues contienen bits que forman un conjunto.

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30	TCCR3A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 5.2.4 –4: Registro de configuración TCCRA

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30	TCCR3B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 5.2.4 –5: Registro de configuración TCCRB

Los bits **COMnA1:0**, **COMnB1:0**, **COMnC1:0** configuran que acción se realiza sobre la salida (de forma similar a lo visto en los temporizadores de 8 bits) en función del modo de trabajo que se haya seleccionado en **WGMn3:0** que está repartido entre **TCCRA** y **TCCRB.** Con **WGMn3:0** se selecciona entre distintos modos de trabajo, normal, CTC y PWM y sus variantes.

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation[1]	TOP	Update of OCRnX at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	TOP	TOP
15	1	1	1	1	Fast PWM	OCRnA	TOP	TOP

Tabla 5.2.4 –4: Selección del modo de trabajo.

Con los bits **CSn2:0** se selecciona la fuente de reloj que actúa sobre el contador **TCNTn,** la tabla es similar aunque añade como fuente las entradas en los pines Tn, lo cual no es posible en temporizador 0 de 8 bits (aunque sí en el 2 que difiere un poco del 0)

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Tabla 5.2.4 – 5: Selección fuente de reloj.

El bit **ICNCn** (Input Capture Noise Canceler) activa el cancelador de ruido, que filtra la señal de entrada en el pin ICP. El bit **ICESn** (Input Capture Edge Select) selecciona el flanco activo para la captura del evento IC.

El registro TCCRC contiene los bits que fuerzan los eventos Output Capture al igual que los vistos en los temporizadores de 8 bits.

Bit	7	6	5	4	3	2	1	0	
	FOC1A	FOC1B	FOC1C	–	–	–	–	–	TCCR1C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	FOC3A	FOC3B	FOC3C	–	–	–	–	–	TCCR3C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Figura 5.2.4 –5: Registro de configuración TCCRC

5.2.5 WatchDog interno.

El Watchdog es un Timer que actúa con su propio oscilador integrado en el propio chip cuya frecuencia es de 1 MHz si la alimentación del microcontrolador es de 5 Voltios. Controlando el prescaler del Watchdog este puede tardar en desbordarse desde 16.3ms a 2100ms. La instrucción WDR pone a 0 el valor del perro guardián. Cuando el Watchdog se desborda, y además está activado el bit correspondiente del registro WDTCR, se produce un RESET de modo que el programa salta al vector $000.

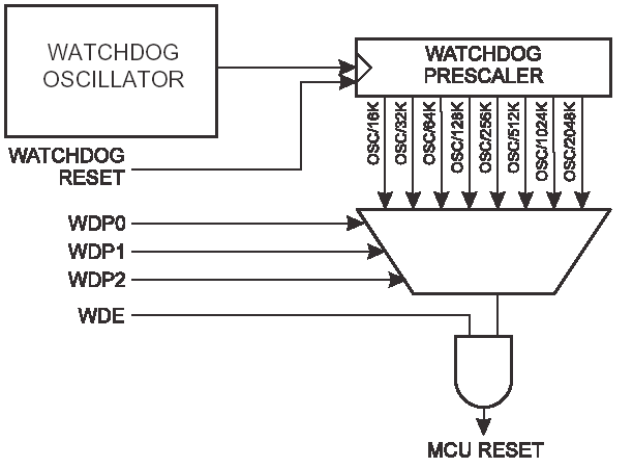


Figura 5.2.5 –1: Esquema del WatchDog.

El registro de configuración del Watchdog consta de un prescaler para configurar el periodo de supervisión **WDP2:0**, y de unos bits para la habilitación. El bit **WDE** habilita el Watchdog, pero para deshabilitarlo en funcionamiento hay que utilizar adecuadamente el bit **WDCE**.

Bit	7	6	5	4	3	2	1	0	
	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0	WDTCR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 5.2.5 –1: Registro de configuración del WatchDog.

WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 3.0V	Typical Time-out at V_{CC} = 5.0V
0	0	0	16K (16,384)	17.1 ms	16.3 ms
0	0	1	32K (32,768)	34.3 ms	32.5 ms
0	1	0	64K (65,536)	68.5 ms	65 ms
0	1	1	128K (131,072)	0.14 s	0.13 s
1	0	0	256K (262,144)	0.27 s	0.26 s
1	0	1	512K (524,288)	0.55 s	0.52 s
1	1	0	1,024K (1,048,576)	1.1 s	1.0 s
1	1	1	2,048K (2,097,152)	2.2 s	2.1 s

Tabla 5.2.5 –1: Temporización del WatchDog.

6. Realización Hardware.

En este capítulo se procede a describir la realización física de la tarjeta de comunicaciones objetivo de este proyecto.

6.1 Esquema Modular.

6.1.1 Sección CAN.

En este apartado se explican aspectos relacionados con el diseño físico del subsistema o sección CAN. En algún momento se hace referencia a ciertos registros internos del controlador CAN que se han de configurar para obtener la funcionalidad hardware deseada.

El diseño que se ha realizado es el sugerido por el fabricante del controlador SJA1000 (Philips).

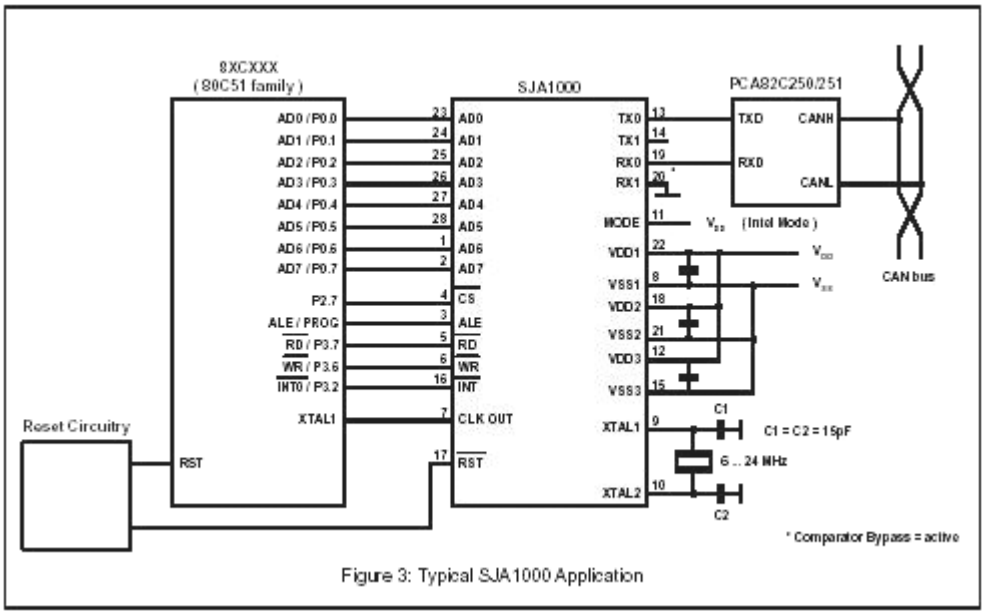


Figura 6.1.1. –1: Aplicación típica del SJA1000.

El controlador SJA1000 es compatible con los microcontroladores de la familia Intel y los de la familia Motorola. Para seleccionar compatibilidad con el modo Intel (que es el que tienen los microcontroladores Atmel) se pone a Vcc el pin **MODE** del SJA1000.

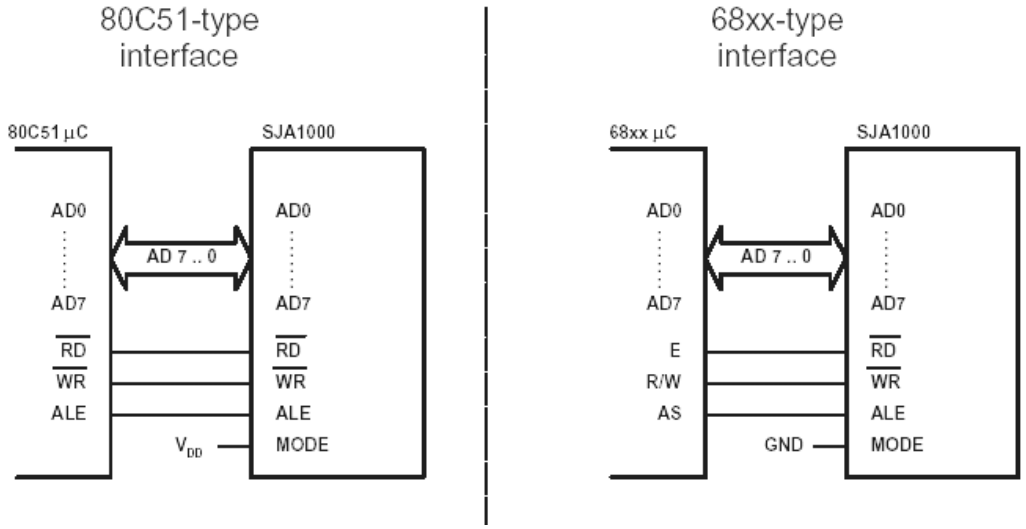


Figura 6.1.1 –2: Selección del tipo de interfaz.

El controlador CAN se presenta al microcontrolador como una memoria externa a la que se accede a través de un bus de direcciones/datos 'latcheado' junto con las líneas de control **ALE**, **RD**, **WR** y **CS**. Como no se va a disponer de ningún otro periférico MMI (mapeado en memoria) o memoria externa se puede poner la señal CS del controlador CAN a nivel bajo para dejarlo siempre habilitado. El controlador tiene una línea de solicitud de interrupción **INT**. Esta línea es puesta a nivel bajo por parte del controlador CAN para solicitar la atención del microcontrolador *host* ante un determinado evento.

El controlador CAN ofrece una señal de reloj (**CLKOUT**) que se puede aplicar al microcontrolador *host*. Esta señal es una señal de reloj escalable a partir de la frecuencia del cristal del controlador CAN. Este escalado se configura en los bits CD2,CD1 y CD0 del registro divisor de reloj CDR.

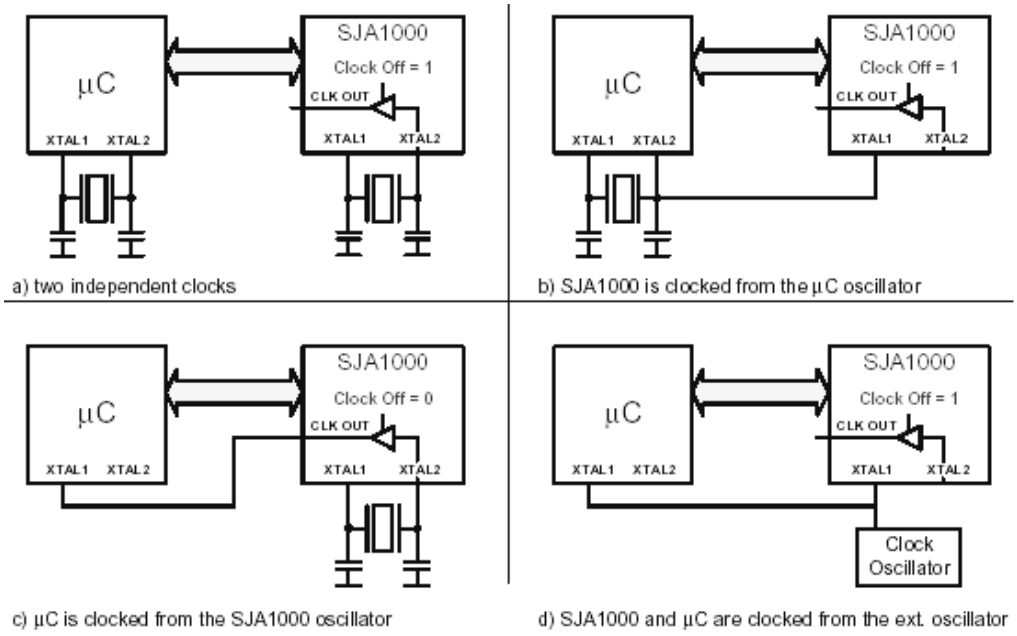


Figura 6.1.1. –3: Estrategias de reloj.

En este caso no se va utilizar CLKOUT pues las señales de reloj del microcontrolador y del controlador CAN se diseñaron para propósitos diferentes. El cristal de 8 MHz elegido para el controlador CAN esta pensado para configurar fácilmente la máxima velocidad física del bus Can de 1Mbit/s, mientras que el reloj del microcontrolador de 11,0592 MHz fue escogido para permitir velocidades altas de transmisión de la UART interna de este dispositivo con bajas desviaciones respecto a la velocidad nominal deseada. Como no se va a emplear la señal CLKOUT se desactiva para disminuir el consumo, esto se hace con el bit clockoff del registro CDR.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
CAN mode	CBP	RXINTEN	(0)(1)	clock off	CD.2	CD.1	CD.0

Figura 6.1.1 –4: Registro CDR.

El controlodaor CAN no se conecta directamente a la línea del bus CAN si no que necesita de una circuitería intermedia. Se va a emplear el circuito integrado transceptor PCA82C250 pues mejora las prestaciones frente a implementaciones discretas. Al emplear un transceptor externo es recomendable 'puentear' el comparador interno del controlador CAN (que esta dispuesto por motivos de compatibilidad con el integrado PCA82C200 que es una versión previa del controlador CAN SJA1000), esto se hace poniendo a '1' el bit CBP (Comparator By-Pass) del registro CDR.

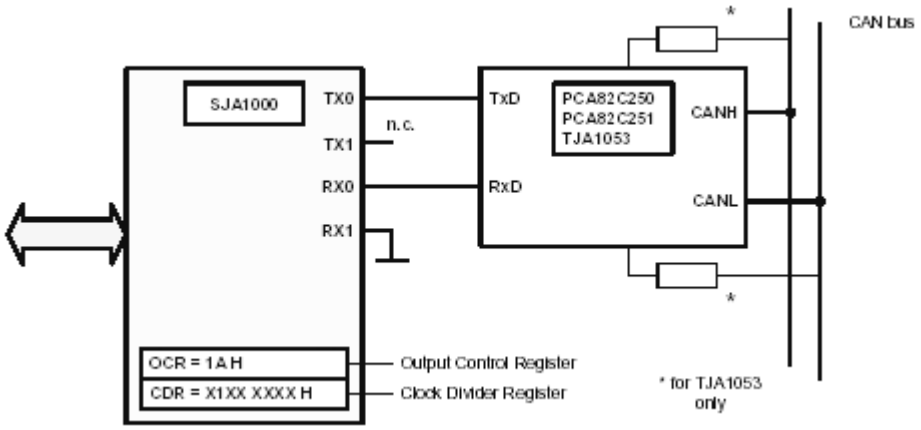


Figura 6.1.1 – 5: Conexión con el transceptor de línea.

Como último aspecto de este apartado se recuerda la importancia de emplear resistencias de terminación de línea de 124 ohmios. Estas resistencias se conectan entre las líneas CANH y CANL en los extremos del bus. Cada tarjeta lleva una clema doble con lo que se dispone un conector para que esa tarjeta acceda al bus y otro para conectar otro segmento de bus o la resistencia de terminación si está tarjeta está ubicada en un extremo del bus.

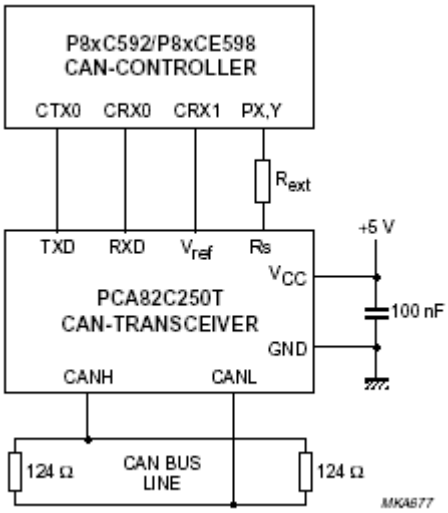


Figura 6.1.1 – 6: Resistencias de terminación de línea.

Existe la posilidad de intercalar optoacopladores entre el controlador CAN y el transceptor para aislar galvánicamente de la línea aunque en esta aplicación no es necesario.

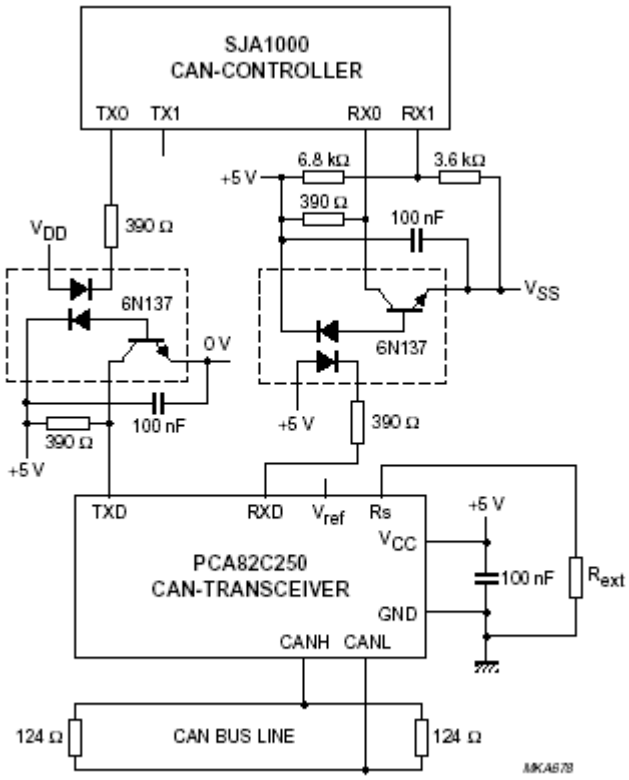


Figura 6.1.1 – 7: Resistencias de terminación de línea y aislamiento galvánico.

La sección del esquemático resultante para la sección CAN que se ha empleado es la siguiente:

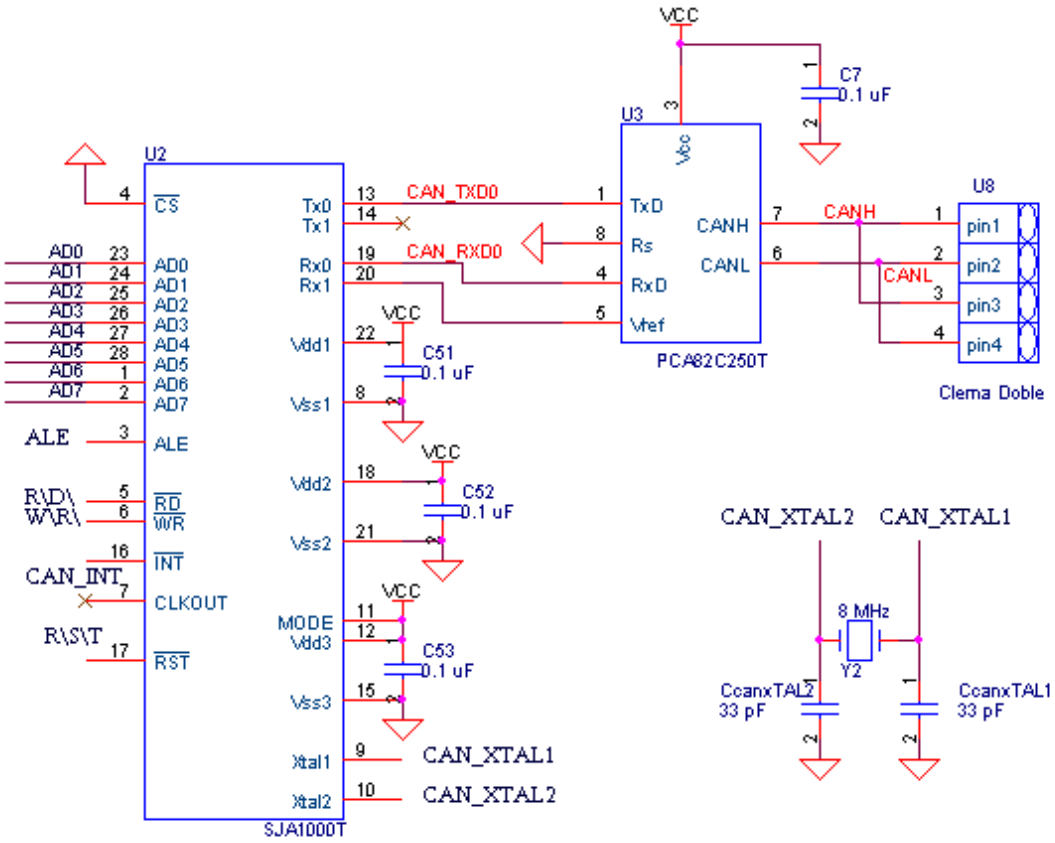


Figura 6.1.1 – 8: Sección CAN del esquemático.

Las líneas **AD0:7** se conectan al puerto A del microcontrolador que está configurado para trabajar como bus de direcciones/datos 'latcheado'. Las líneas de control de acceso a la memoria **ALE**, **RD** y **WR** se conectan también directamente a sus homónimas en el microcontrolador. La línea **CS** se conecta directamente a tierra pues este es el unico dispositivo mapeado en memoria SRAM externa. La línea de **RESET** es actuada por la circuitería de RESET que se estudia en un punto posterior de este capítulo.. La línea de interrupción se conecta a la línea de interrupción 0 del microcontrolador que debe estar configurada para que se provoque la interrupción cuando esta línea se halle a nivel bajo. Las líneas de alimentación **Vdd** y **Vss** se conectan a las lineas **Vcc** y **GND** del sistema y se intenta ubicar lo mas cerca posible del controlador CAN los condensadores de desacoplo.

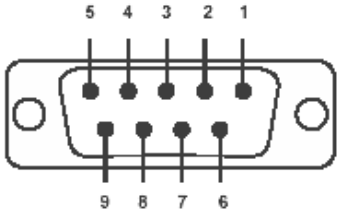
6.1.2 Sección RF.

La sección en el caso del empleo del módulo Free2Move es básicamente un interfaz RS-232 más una serie de señalizaciones para indicar visualmente el estado de la comunicación. En el caso del módulo BRM se tiene también un interfaz serie pero este

módulo provee de ciertas señales para indicar el estado de la comunicación y dispone de una serie de entradas para ser configurado.

En el caso del módulo Free2Move hay que tener en cuenta que presenta un interfaz serie RS-232 del tipo DCE en el que se utiliza la señal 9 para proveer la alimentación al dispositivo **VCC**. Las otras líneas de este interfaz que se emplean son **GND**, **TXD**, **RXD**, **RTS**, y **CTS**. Las líneas **DSR** y **DTR** no se emplean y el módulo las cortocircuita internamente. Las líneas RTS y CTS se emplean si se configura el control de flujo hardware.

Pin	RS-232 signal	Direction
1	CD	Not connected
2	TxD	Output
3	RxD	Input
4	DSR	Not connected
5	GND	
6	DTR Connected to pin 1 and 4	Not connected
7	CTS	Input
8	RTS	Output
9	Vcc Power	Input 4-5V



Serial Port Plug connector (female)

Figura 6.1.2 – 1: Interfaz del módulo Free2Move.

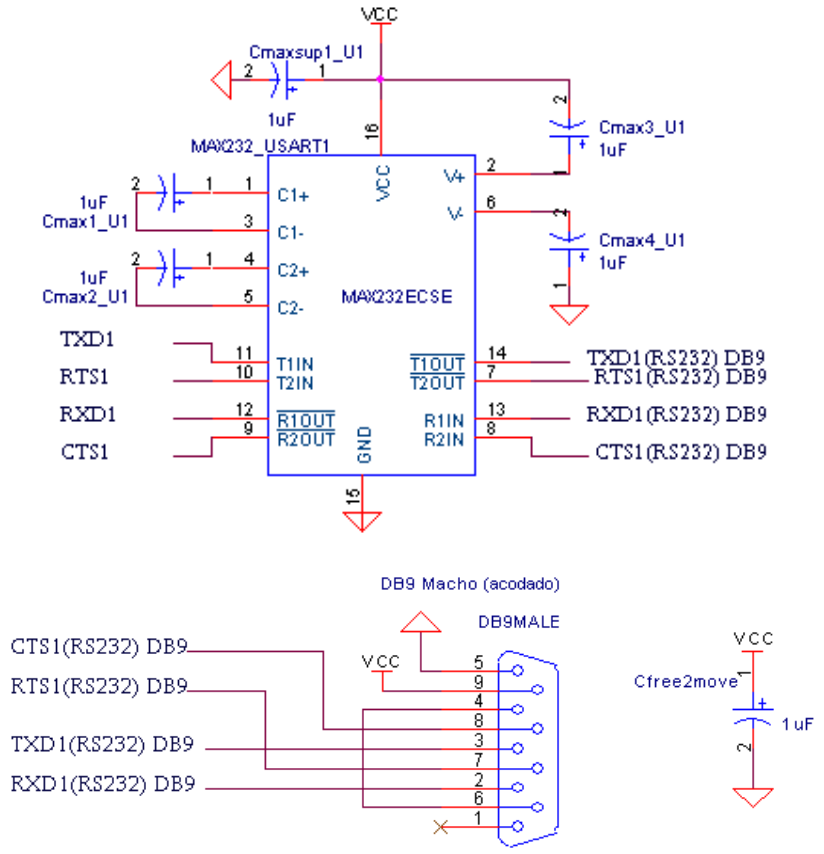


Figura 6.1.2 – 2: Adaptación TTL – RS232 para el módulo Free2Move.

Las líneas **TXD1** y **RXD1** se conectan al puerto serie del microcontrolador **USART1**. Este puerto serie del microcontrolador no incorpora el control de flujo hardware por lo que se habilitan en el microcontrolador una entrada digital llamada **CTS1** y una salida digital llamada **RTS1** para realizar las funciones de control de flujo si es necesario. Las señales del circuito MAX232 con niveles RS232 actúan o provienen del conector DB9 macho. El módulo Free2Move se inserta en este conector. Hay que tener en cuenta que el microcontrolador actúa como un dispositivo **DTE** mientras que que el módulo Free2Move presenta una interfaz tipo DCE por lo que la salida de transmisión **TXD1(RS232) DB9** del conversor de niveles (pin 3 del conector DB9 macho) debe conectarse con la entrada **RXD del módulo** (pin 3 del conector DB9 hembra del módulo Free2Move). Realmente el pin 3 del conector DB9 hembra en la norma RS-232 se llama TXD pero el fabricante le ha cambiado el nombre y la funcionalidad para evitar confusiones y el empleo de un cable cruzado (el módulo hace internamente el cruce de señales y así admite la conexión directa al puerto serie de un PC). Este argumento es idéntico para las otras señales y la mejor forma de no equivocarse es comprobar que la salida de un dispositivo va a la entrada del otro y viceversa. Se ha añadido un condensador para estabilizar la tensión de alimentación que se ubica lo mas cerca posible del conector DB9 macho.

ATmega		MAX232		DB9 Macho (DTE)		DB9Hembra Free2Move (DCE)
TXD1 (28)	→	T1IN / T1OUT	→	TXD (3)	→	RXD (3)
RXD1 (27)	←	R1OUT / R1IN	←	RXD (2)	←	TXD (2)
RTS1 (29)	→	T2IN / T2OUT	→	RTS (7)	→	CTS (7)
CTS1 (30)	←	R2OUT / R2IN	←	CTS (8)	←	RTS (8)

Tabla 6.1.1 – 1: Sentido de la comunicación

Para la señalización del estado del dispositivo se consideran los siguientes indicadores luminosos DATA_ACTIVITY y CONNECTION (que aparece como J1 en el esquemático porque este pin en el microcontrolador esta compartido para funcionar como J1 si se trabaja con el módulo BRM01). DATA_ACTIVITY indica si se están transmitiendo y/o recibiendo datos y CONECCTION indica si módulo Free2Move está listo para trabajar (se ha iniciado correctamente la conexión). Estas señales se configuran como salidas en el microcontrolador.

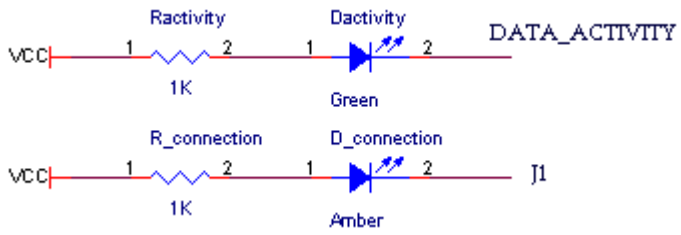


Figura 6.1.2 – 3: Leds indicadores del estado de la comunicación.

Existen conectores para poder comunicar esta tarjeta con el módulo BRM01 para lo cual se considera el siguiente esquemático:

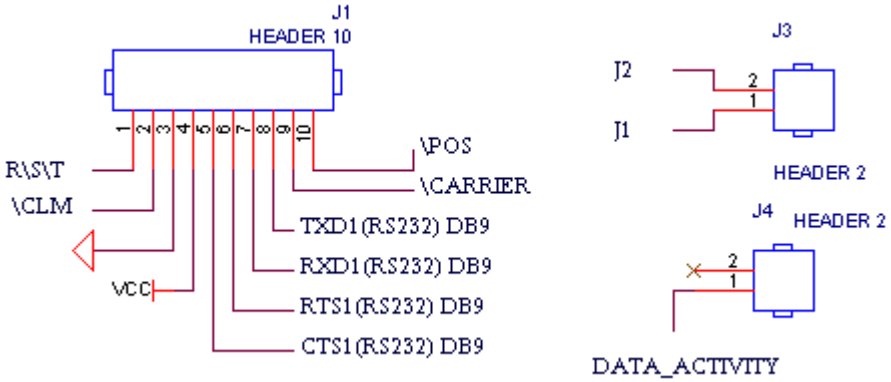


Figura 6.1.2 – 4: Conectores para conexión de módulo BRM01.

Las señales RS232 provienen del circuito MAX232 y la señal **RESET** de la circuiteria que desempeña esta funcionalidad. Las señales **\POS**, **\CARRIER** y **\DATA_ACTIVITY** son salidas del módulo BRM01 e indican el estado del dispositivo (si ha arrancado bien), la presencia de portadora y si se está transmitiendo y/o recibiendo datos. Las señales **J1** y **J2** son entradas del módulo BRM01 para seleccionar el modo 'enhaced' y básico respectivamente.

6.1.3 Interfaz Puerto Serie Depuración y programación SPI.

Estos interfaces se describen conjuntamente en este punto debido a que comparten los pines de acceso al microcontrolador como se aprecia en la figura 6.1.4 –1 del siguiente punto. La programación del microcontrolador se realiza a través del interfaz SPI (Serial Programming Interface). En el caso del ATmega128 existe una peculiaridad y es que la funcionalidad de escritura/lectura de programa de la memoria flash se realiza través de los pines **PDI/PDO** y no de los pines **MOSI/MISO** como suele ser habitual en otros microcontroladores de la familia AVR. Ambas funcionalidades, USART0 y SPI, no interfieren entre sí pues la programación SPI se realiza durante la fase de RESET en la que el sistema está inactivo en su modo de funcionamiento normal. Para compartir los pines simplemente se conectan a ambos interfaces. Las señales **RXD0/PDI** y **TXD0/PDO** son del microcontrolador y se bifurcan para ser conectadas al adaptador de niveles TTL – RS232 y al conector del programador SPI.

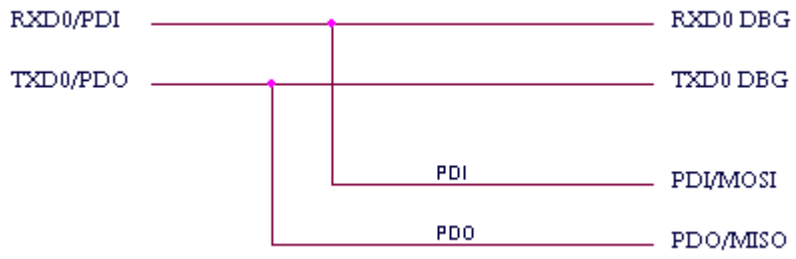


Figura 6.1.3 –1: Conexión a los pines del microcontrolador.

El interfaz SPI simplemente consta de un conector para comunicarse con el programador flash AVR ISP. La señal **PDI/MOSI** se emplea para enviar el programa desde el programador hasta el microcontrolador mientras que **PDO/MISO** se usa para leer el contenido de la memoria de programa. Este tipo de transferencias son síncronas para lo que se emplea la señal de reloj **SCK**. La señal **RESET** se conecta al RESET del sistema y actúa sobre el RESET del microcontrolador para que este entre en el modo de programación.

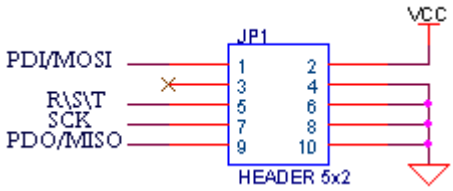


Figura 6.1.3 –2: Conector del programador ISP.

El dispositivo USART0 se emplea para conectar el sistema al puerto serie de un PC y facilitar la depuración del SW bajo desarrollo. Para comunicarse con el puerto serie de un PC hay que adaptar los niveles a la norma RS232 por lo que se emplea un adaptador de niveles basado en un circuito integrado MAX232 similar al empleado en el interfaz de acceso al módulo Free2Move. En este caso se tiene una comunicación entre dos dispositivos DTE por lo que o bien se emplea un cable serie cruzado o se cruzan las señales en algún determinado punto del esquemático siendo esta última la mejor opción pues es más fácil encontrar en el mercado el cable extensor del puerto serie DB9 que el cable cruzado.

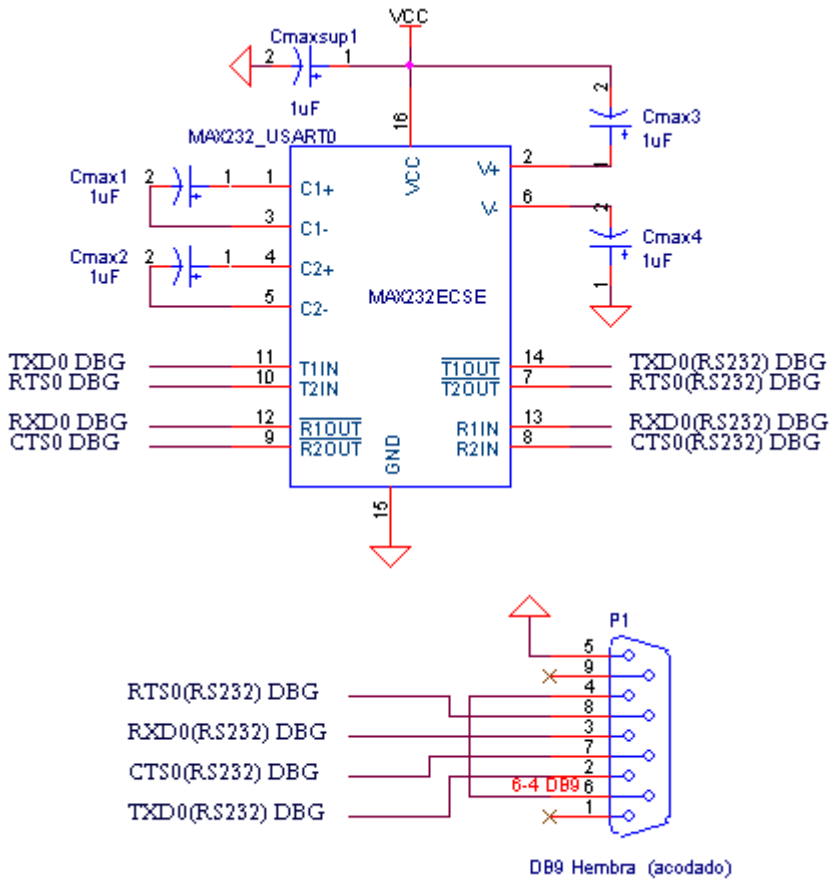


Figura 6.1.3 –3: Adaptación TTL – RS232 para el puerto serie de depuración.

En este caso se ha hecho el cruce de líneas entre el MAX232 y el conector DB9 hembra de manera que el PC ve el sistema como un DCE.

ATmega (DTE)		MAX232		DB9Hembra (DCE)		DB9 Macho (DTE) Puerto Serie PC
TXD1 (28)	→	T1IN / T1OUT	→	TXD (2)	→	RXD (2)
RXD1 (27)	←	R1OUT / R1IN	←	RXD (3)	←	TXD (2)
RTS1 (29)	→	T2IN / T2OUT	→	RTS (8)	→	CTS (7)
CTS1 (30)	←	R2OUT / R2IN	←	CTS (7)	←	RTS (8)

Tabla 6.1.3 –1: Adaptación TTL – RS232 para el puerto serie de depuración.

6.1.4 Sección Microcontrolador.

El microcontrolador es el elemento más importante del sistema pues se encarga de controlar los diferentes interfaces. En la figura 6.1.4 –1 se aprecian las señales que comunican con los diversos interfaces. La señal de **RESET** proviene de la circuitería de RESET que se describe en un punto posterior. El cristal que se emplea es de 11.0592 MHz. Es necesario emplear condensadores de desacoplo para regular la tensión de

alimentación del dispositivo. Estos condensadores se disponen cerca del microcontrolador para que actúen de manera eficiente.

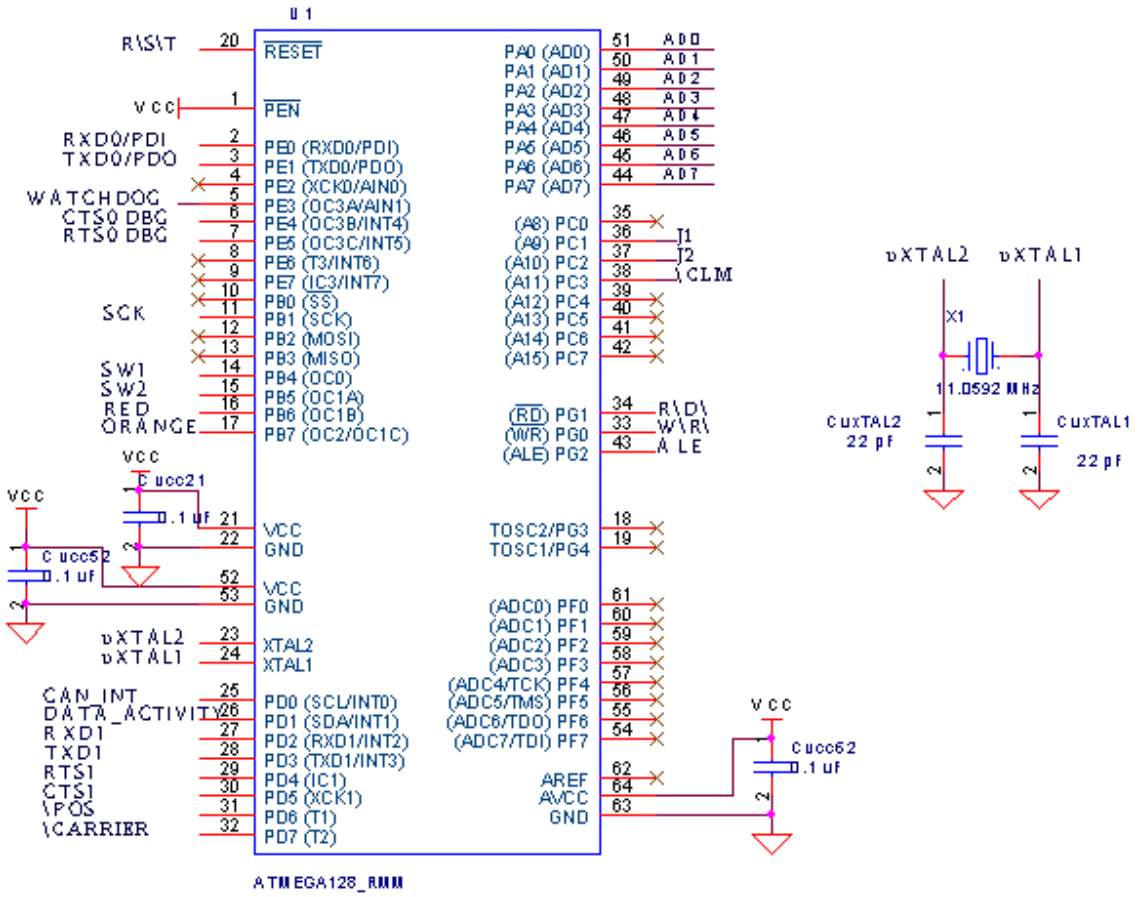


Figura 6.1.4 – 1: Microcontrolador.

6.1.5 Alimentación.

El circuito de alimentación se basa en un regulador BA17805FP que soporta hasta 500 mA de intensidad nominal.

El componente que más consume con diferencia es el módulo de radiofrecuencia que puede absorber hasta 168 mA. El consumo del microcontrolador aumenta con la frecuencia. El fabricante entrega la curva de la figura 6.1.5-2 para estimar el consumo del microcontrolador. Se ha tomado el valor asociado a una frecuencia de 11 MHz que es aproximadamente 22.5 mA y como valor de pico se ha estimado 50 mA. El consumo asociado a las redes que alimentan los diodos led se ha estimado considerando que estas redes constan de una resistencia de 1 KΩ y están sometidas a 5V cuando se encienden los diodos al poner a nivel bajo la salida del microcontrolador asociada a ese led. De esta manera despreciando la resistencia interna del diodo se tiene I_{led} = 5V/ 1 KΩ = 5 mA. Para el consumo medio se ha estimado que estos no están todo el tiempo encendidos si no sólo la mitad. Como se tienen 5 leds se considera un consumo total máximo de 25 mA y un consumo medio que es la mitad del máximo.

En la tabla 6.1.5 – 1 se detallan los consumos medios y máximos de cada uno de los dispositivos. El consumo total medio aproximado es de 160 mA y el máximo aproximado es de 345 mA. Ambos valores son inferiores a los valores máximos y medio de intensidad que puede proveer el regulador de tensión BA17805 por lo que este regulador es válido.

Dispositivo	Consumo medio (mA)	Pico Máximo (mA)
Módulo Free2Move	69	168
Controlador CAN SJA1000	10	15
Transceptor CAN	18	70
ATmega128	22.5	50
MAX232 USART0	14	20
MAX232 USART1	14	20
Líneas Led	12.5	25
Supervisor Reset MAX823	0.010	0.024
TOTAL APROXIMADO	160	395
Intensidad de salida de BA17805	500	875

Tabla 6.1.5 – 1: Estimación del consumo total.

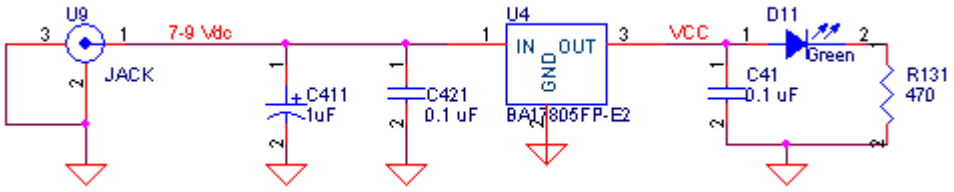


Figura 6.1.5 – 1: Circuito regulador de alimentación.

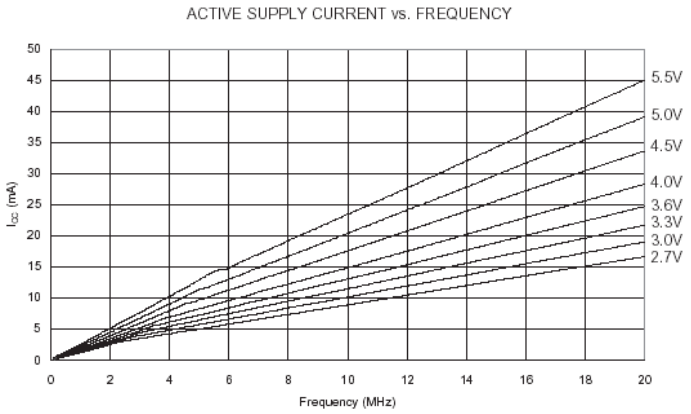


Figura 6.1.5 – 2: Característica de consumo del microcontrolador ATmega128

6.1.6 Circuito de Reset y Brown-Out

El circuito integrado MAX823 provee una buena señal de RESET , provoca un RESET interno cuando la alimentación del sistema esta estabilizada, monitoriza la tensión de alimentación y fuerza un RESET si ésta cae por debajo de un umbral determinado (detección de Brown-Out) y además tiene un watchdog que resetea el

sistema si no recibe la orden contraria del microcontrolador antes de un determinado plazo.

El microcontrolador ATmega dispone internamente de la generación de un RESET una vez que se ha estabilizado la tensión de alimentación, del detector de Brown-Out y de un watchdog interno. Aún así es aconsejable el empleo de circuito supervisor y RESET para proveer de un buen RESET al controlador CAN y en cualquier caso proveer una señal de RESET limpia.

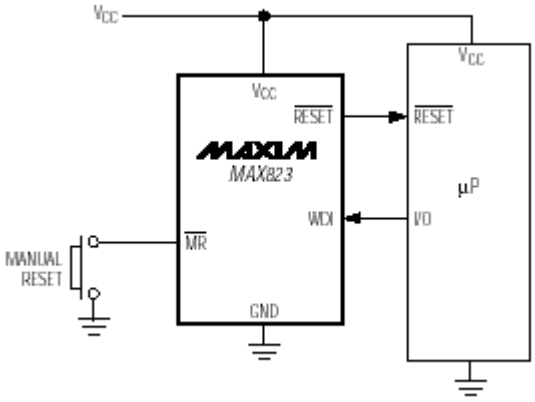


Figura 6.1.6 -1: Circuito supervisor de RESET MAX823.

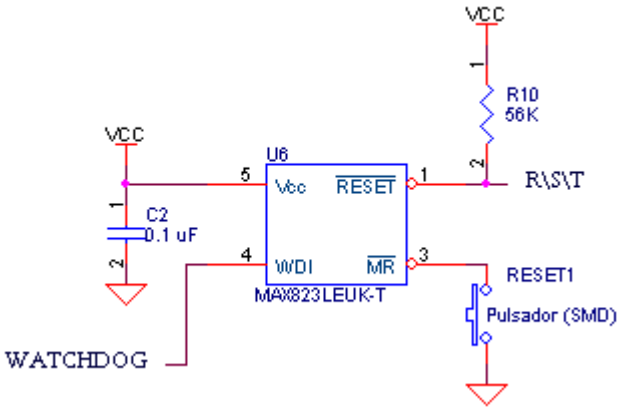


Figura 6.1.6 –2: Acondicionamiento del circuito MAX823.

6.1.7 Pulsadores y Leds de Prueba

Para facilitar la depuración del SW se incorporan un par de pulsadores y leds luminosos par tener un mecanismo fácil para interactuar con el microcontrolador. Las

señales **SW1** y **SW2** se configuran en el microcontrolador como entradas digitales y **RED** y **ORANGE** como salidas.

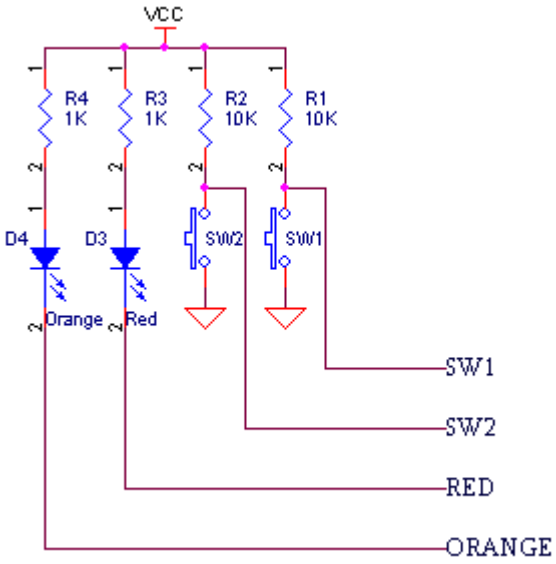
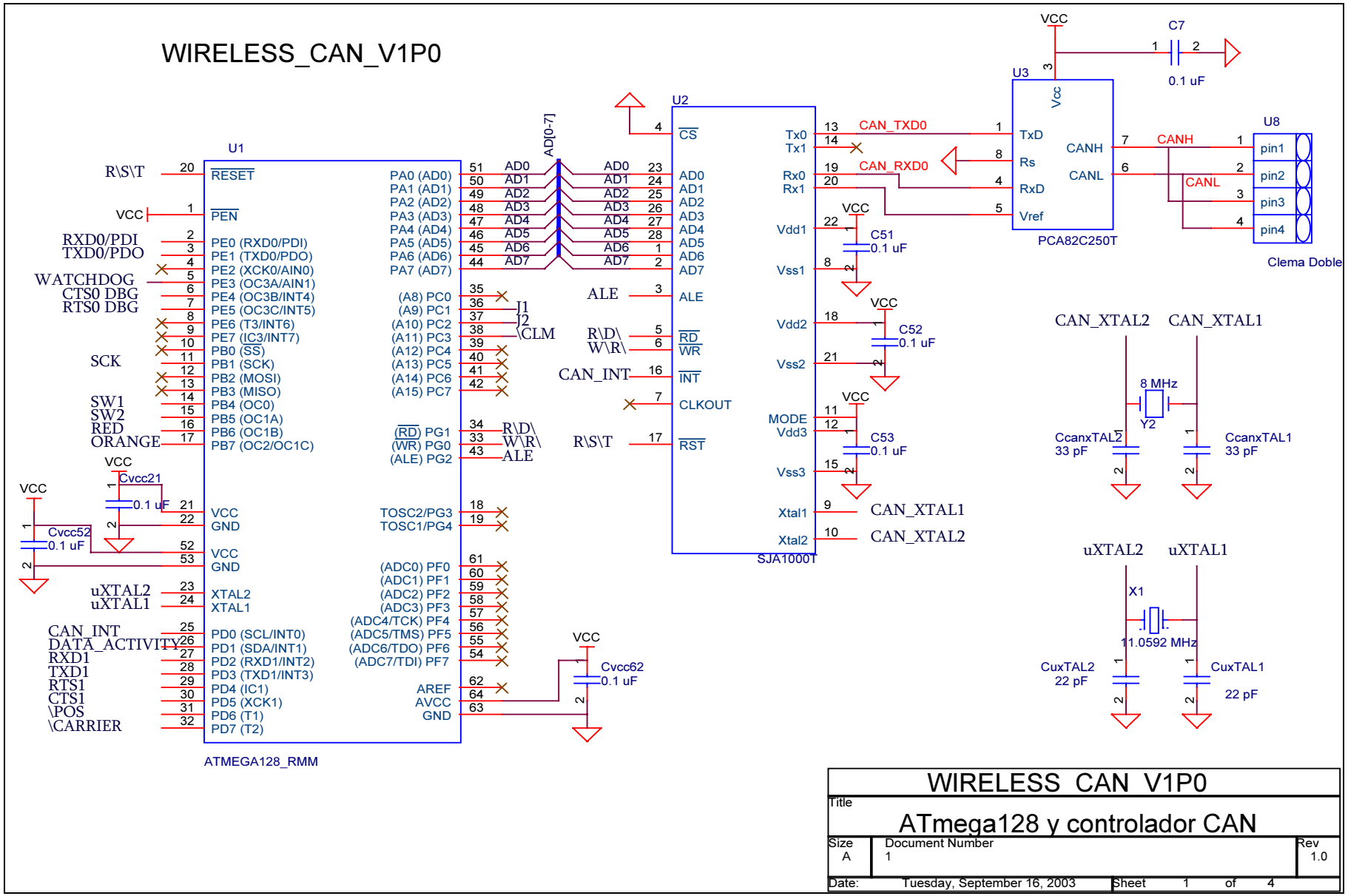


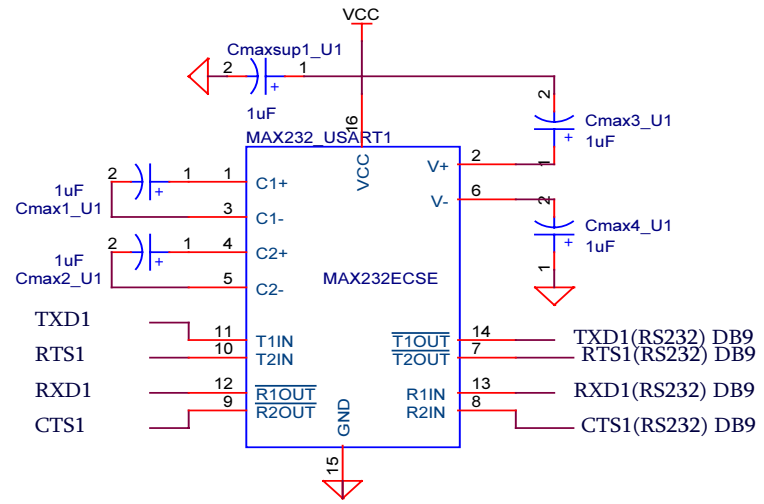
Figura 6.1.7 –1. Pulsadores de entrada y leds para pruebas.

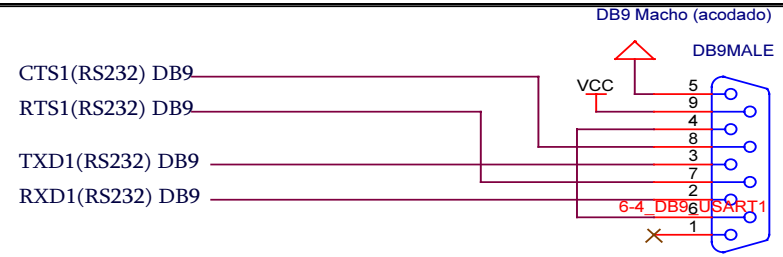
6.2 Esquemático global en Orcad.

En la siguientes páginas se adjuntan los planos del esquemático realizado con el paquete de diseño ORCAD. Se han agrupado en hojas según la funcionalidad vista en el punto 6.1 donde se detalló cada uno de los subsistemas.

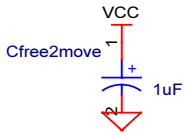

WIRELESS_CAN_V1P0
U1
ATMEGA128_RMM
U2
SJA1000T
U3
PCA82C250T
U8
Clema Doble
R\S\T
RESET
VCC
PEN
RXD0/PDI
TXD0/PDO
PE0 (RXD0/PDI)
PE1 (TXD0/PDO)
WATCHDOG
CTS0 DBG
RTS0 DBG
SCK
SW1
SW2
RED
ORANGE
uXTAL2
uXTAL1
CAN_INT
DATA_ACTIVITY
RXD1
TXD1
RTS1
CTS1
\POS
\CARRIER
AD[0-7]
ALE
R\D\
W\R\
\CLM
J1
J2
CAN_TXD0
CAN_RXD0
CANH
CANL
CAN_XTAL2
CAN_XTAL1
8 MHz
Y2
11.0592 MHz
X1
C7
0.1 uF
C51
C52
C53
Cvcc21
Cvcc52
Cvcc62
CcanxTAL2
CcanxTAL1
CuxTAL2
CuxTAL1
33 pF
22 pF
WIRELESS_CAN_V1P0
ATmega128 y controlador CAN
Document Number
1
Rev
1.0
Tuesday, September 16, 2003
Sheet 1 of 4

Conversión 5V <-> RS232 del Interfaz para el módulo Free2Move



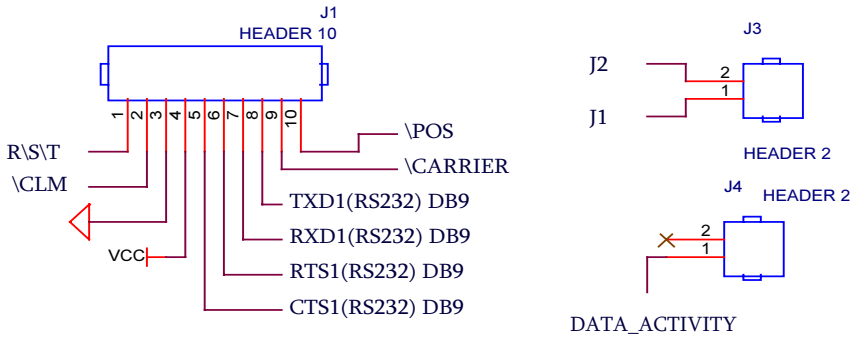


en este caso el módulo Free2Move presenta una interfaz DCE por lo que para usar un cable extensor del puerto serie no es necesario hacer el cruce de señales, el módulo Free2Move hace este cruce internamente.
NOTA: el módulo Free2Move se presenta como un DCE, por lo que no hay que cruzar señales, solo adaptar niveles.

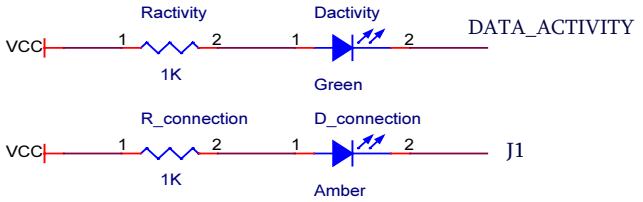


ATmega	MAX232	DB9
TXD1 (28) ---------->	T1IN / T1OUT ---------->	TX (3)
RXD1 (27) <----------	R1OUT / R1IN <----------	RX (2)
RTS1 (29) ---------->	T2IN / T2OUT---------->	RTS (7)
CTS1 (30) <----------	R2OUT / R2IN <----------	CTS (8)

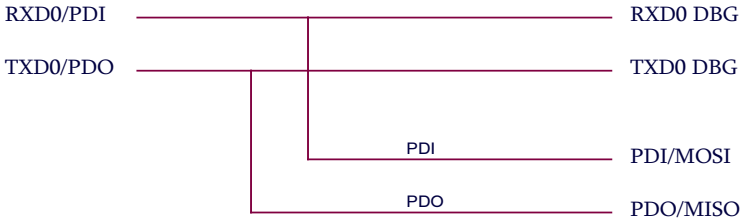
Zócalo para compatibilidad módulo BRM01



Leds indicadores del estado de la comunicación.



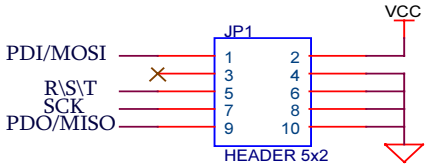
Title	WIRELESS CAN V0P1		
	Interfaz para módulo Free2Move		
Size A	Document Number 1		Rev 1.0
Date: Tuesday, September 16, 2003		Sheet 3 of 4	



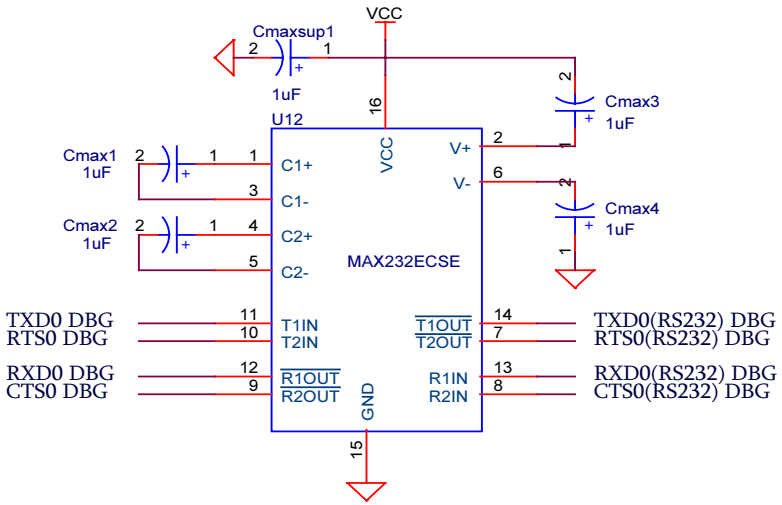
Para la programación mediante el interfaz SPI, se emplean los pines PDI y PDO que son compartidos con el dispositivo USART0.

Conexión programador

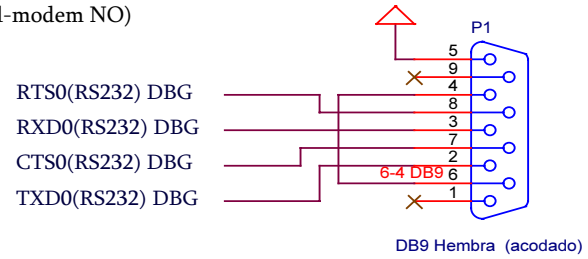
Conexión SPI GENERICO



Conversión 5V <-> RS232



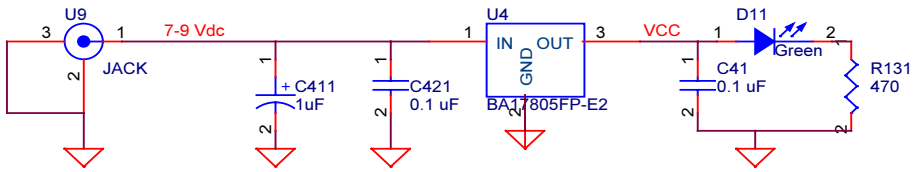
NOTA: el intercambio de pines está hecho, de modo que hay que utilizar un cable extensor del puerto serie (Null-modem NO)



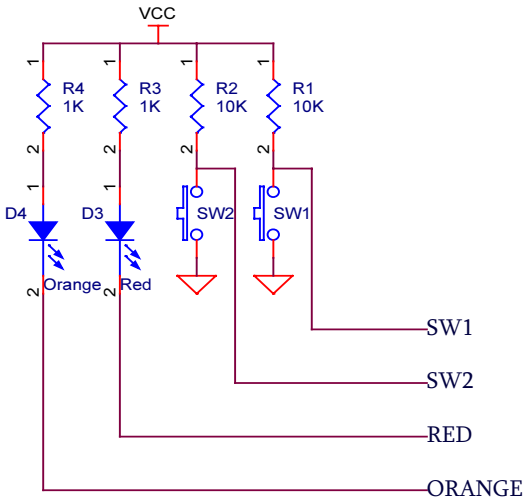
WIRELESS CAN V0P1		
Title: USART0 e Interfaz SPI		
Size: A	Document Number: 1	Rev: 1.0
Date: Tuesday, September 16, 2003	Sheet 4 of 4	

WIRELESS_CAN_V1P0

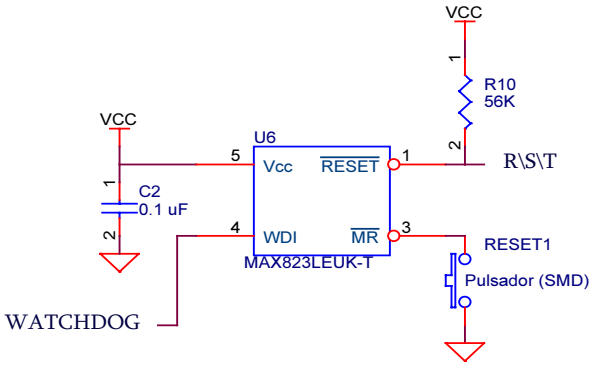
Regulación de la alimentación



Pulsadores y Leds de prueba



Brown-Out detector y pulsador de reseteo



WIRELESS_CAN_V1P0		
Title: Alimentación, Reset y Pulsadores/Leds.		
Size: A	Document Number: 1	Rev: 1.0
Date: Tuesday, September 16, 2003	Sheet 2 of 4	

6.3 Características del trazado de pistas de la tarjeta.

La tarjeta del sistema se ha diseñado para ser realizada sobre una placa de circuito impreso de **doble cara** dada la complejidad del sistema y la necesidad de obtener una tarjeta de reducidas dimensiones y peso. Se han empleado componentes de montaje superficial (**SMD**) que son mas pequeños y ligeros que los componentes tradicionales que se montan a través de agujeros en la placa, ocupando espacio en ambas caras de la placa.

Es importante una buena disposición de componentes para obtener un correcto funcionamiento del sistema y que el trazado de pistas no se convierta en una tarea inviable. Cada uno de los subsistemas se ha realizado en una zona separada de la placa para reducir las interferencias entre estos subsistemas. Para facilitar el diseño se ha optado por una distribución radial de los subsistemas desde el microcontrolador. En la figura 6.3 –1 se aprecia como se halla en el microcontrolador en el centro de la placa. En la parte superior izquierda se halla el interfaz asociado al puerto serie de depuración y el conector SPI. El subsistema CAN, compuesto por el controlador CAN, su transceptor asociado y la clema doble para la conexión al bus, se halla en la parte superior derecha. En la parte izquierda se hallan los pulsadores y leds para funciones de prueba y depuración. En la esquina inferior izquierda se ha situado el subsistema de alimentación y en la esquina inferior derecha el interfaz de acceso al módulo de radio frecuencia. Se ha intentado, en la medida de lo posible dejar en la parte inferior izquierda la circuitería de valores de tensión continua o baja frecuencia (como los pulsadores de leds y prueba). En la parte derecha se han colocado los subsistemas que trabajan a mayores frecuencias.

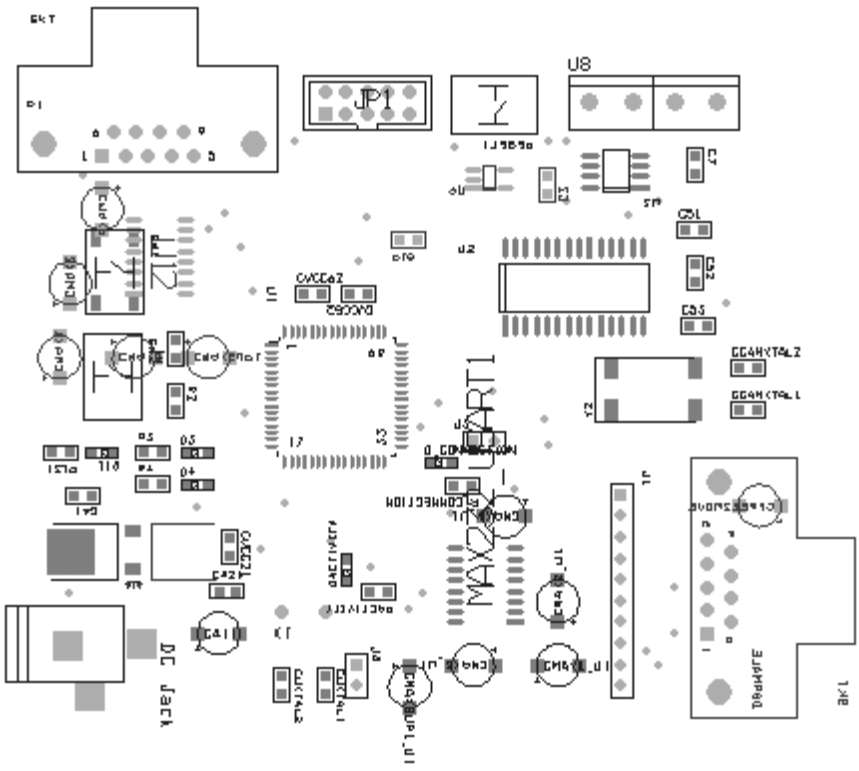


Figura 6.3 –1: Ubicación de componentes en la tarjeta.

Se ha optado por ubicar la mayoría de los componentes en cara superior para facilitar la conexión de los componentes al microcontrolador y dejar la cara inferior para realizar el rutado mas complicado. De esta manera se evita el uso excesivo de vías que comunican ambas caras.

El rutado ha sido una tarea laboriosa pues sistemas similares de tal complejidad suelen ser realizados con tecnologías de 4 o mas caras, equipos para taladradado automático y soldado de componentes con técnicas de soldado por ola de estaño, en la que se pegan los componentes sobre la placa (convenientemente preparada) y el conjunto de placa más componentes pasa a través de una fina película de estaño.

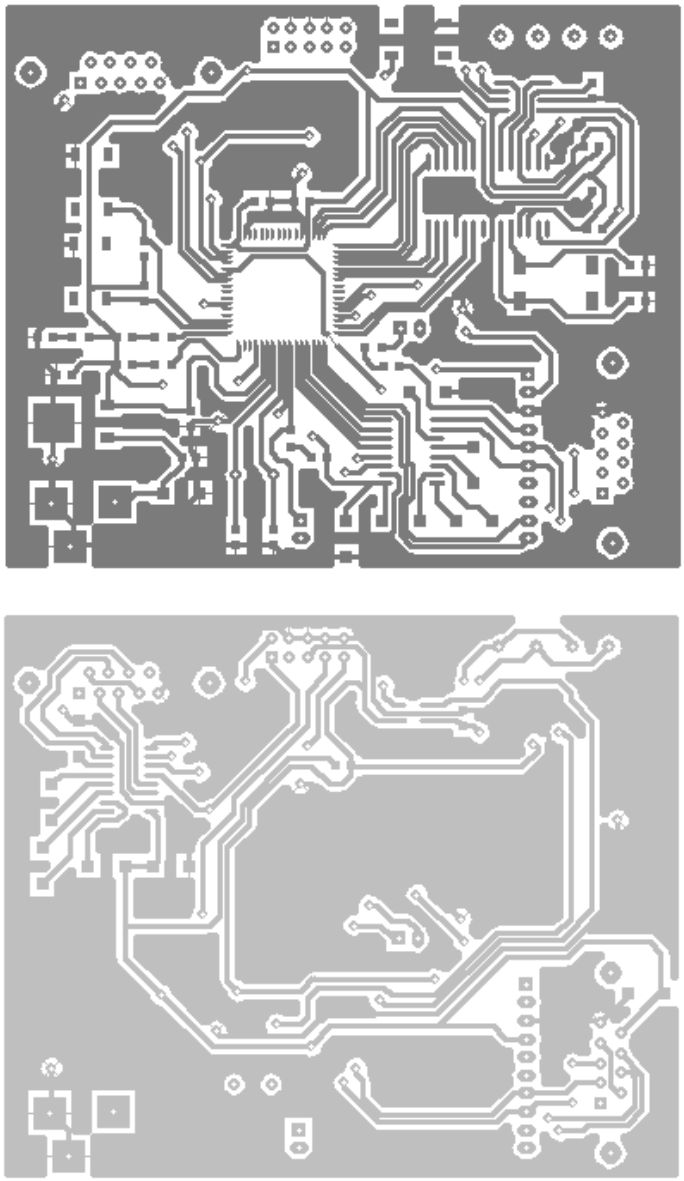


Figura 6.3 –2: Fotolitos de cara superior e inferior.

En la figura 6.3 –2 se representan los fotolitos de la cara superior e inferior están escalados aproximadamente al tamaño real de la placa pero no tienen la exactitud

suficiente como para extraer directamente el fotolito de esta representación. Para obtener un fotolito exacto debe imprimirse desde la herramienta GerberTool de Orcad y procediendo a la reflexión previa de la cara superior para que el lado de la transparencia con tinta esté más próximo a la placa objeto de la exposición ultravioleta.

6.4 Fotografías de los módulos.

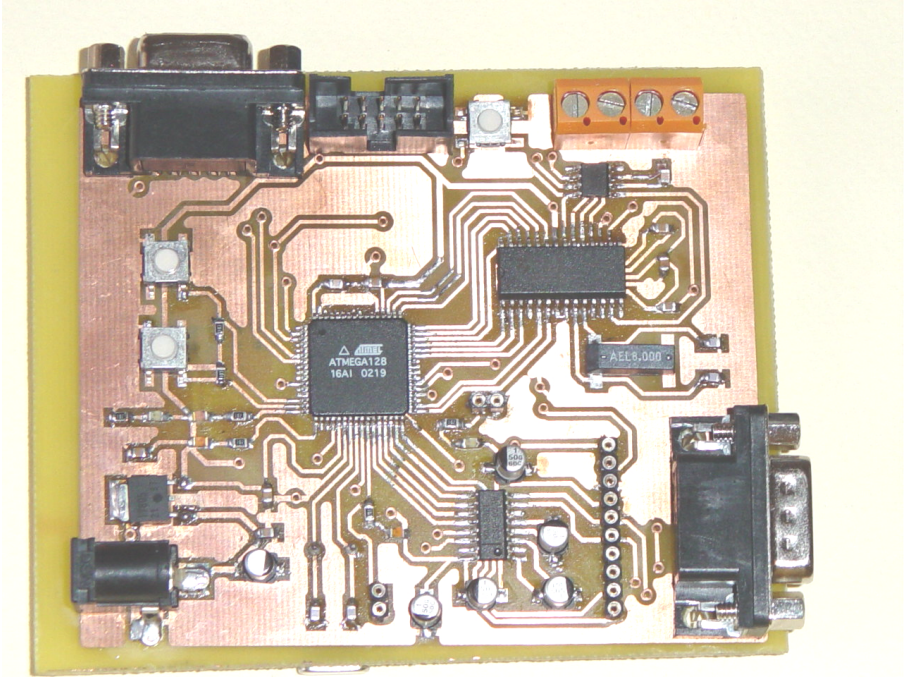


Figura 6.4 –1: Cara superior de la tarjeta.

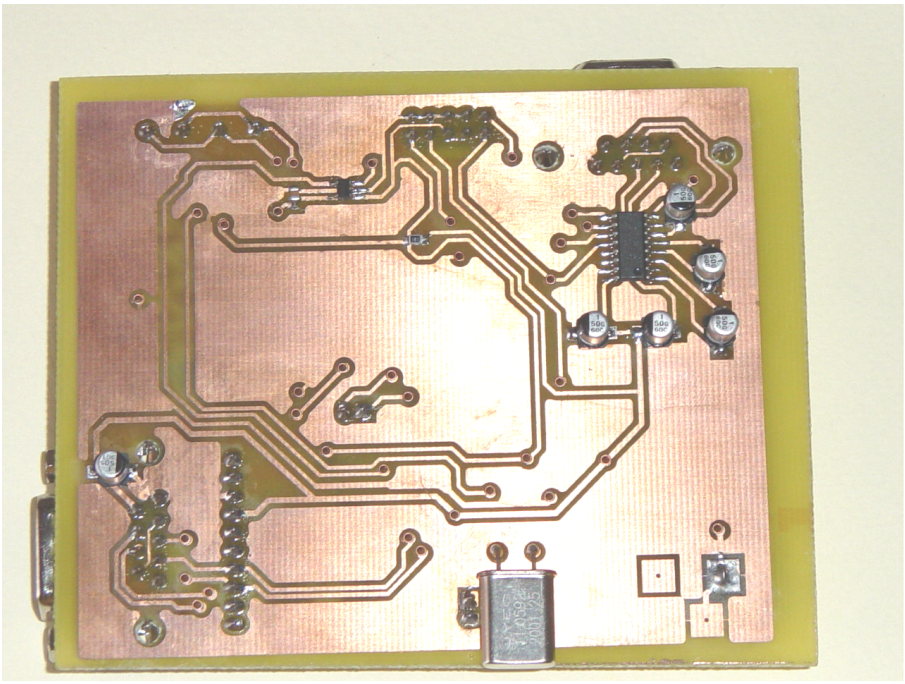


Figura 6.4 –2: Cara inferior de la tarjeta.

7. Programación del dispositivo.

7.1 Esquema general del programa.

El programa que se ejecuta en el microcontrolador de la tarjeta objetivo de este proyecto intenta aprovechar en la medida de lo posible las características del microcontrolador. Para ello se hace uso de interrupciones que avisan al microcontrolador de que se ha recibido un dato (por el interfaz serie o que el controlador CAN tiene un dato listo para que lo extraiga el microcontrolador) o de interrupciones que avisan al microcontrolador que los dispositivos de transmisión están disponibles para enviar datos (registro de datos del dispositivo serie o buffer de transmisión del controlador CAN).

La función principal de este sistema, que es actuar como interfaz entre los dos dispositivos de comunicación, se realiza entonces por medio de interrupciones con lo que la atención del sistema es prioritaria a la función de comunicación.

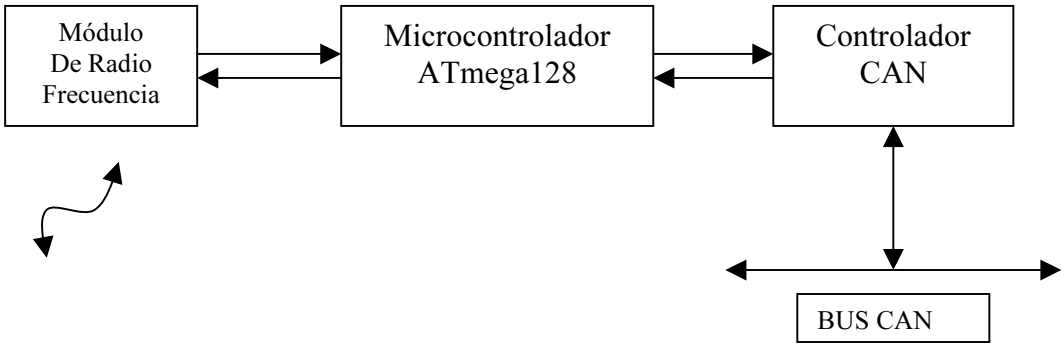


Figura 7.1 –1: Esquema de comunicación

El sistema se basa en dos buffers o colas, una para cada sentido de la comunicación. Los mensajes recibidos por el dispositivo serie procedentes del módulo de radiofrecuencia se introducen en una cola o estructura FIFO y son tomados de esta cola para su transmisión por el bus CAN cada vez que el controlador CAN se halle disponible. De la misma manera existe una cola en la que los mensajes recibidos del bus CAN son almacenados para su envio por el dispositivo de radio.

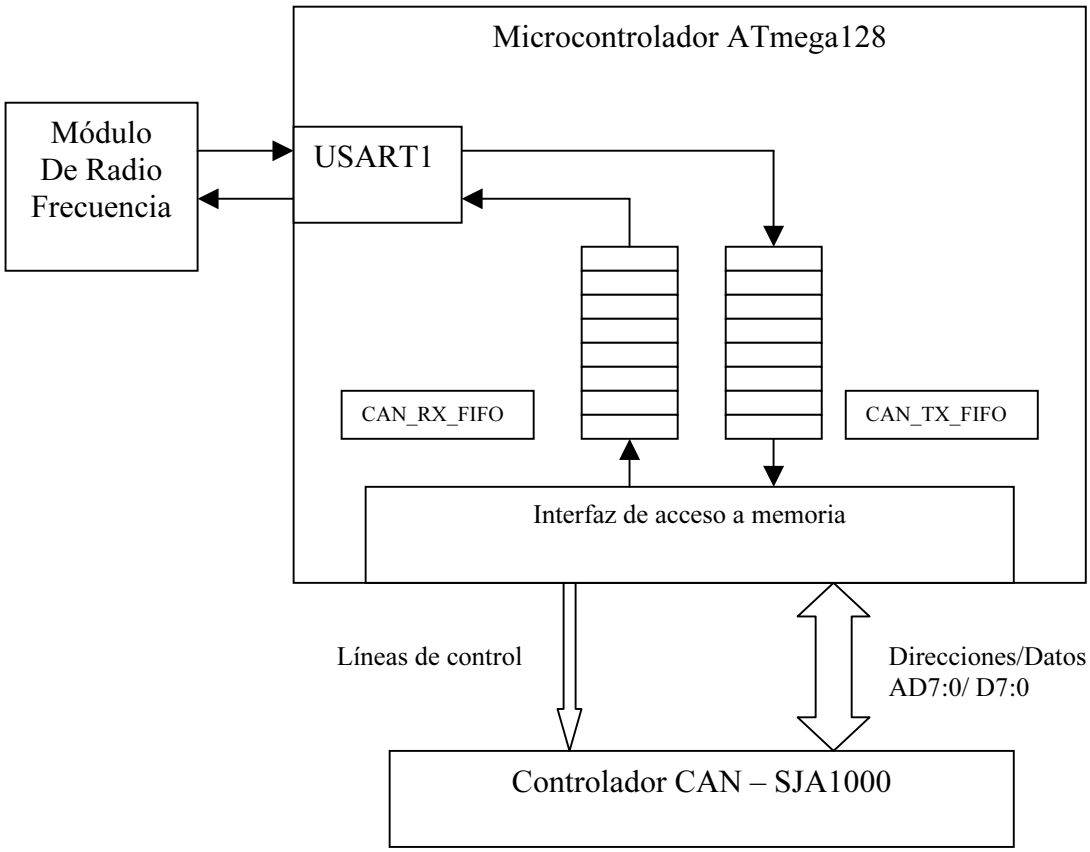


Figura 7.1 –2: Esquema de comunicación detallando microcontrolador.

Otras funciones que se realizan en el microcontrolador son el manejo del puerto serie USART0 para funciones de depuración, el manejo de temporizadores y de entradas/salidas digitales.

Los buffers o colas tienen estructura circular con índices de cabeza y cola y tamaño potencia de dos para facilitar la actualización de índices al final físico del buffer. El dato que se almacena en estos buffers es de tipo Can_Frame que es un tipo estructura compuesto de 3 bytes que contiene la información que se pasa o recibe a / de el controlador CAN. La comunicación con el controlador CAN se hace escibiendo / leyendo en registros de este controlador que se acceden como posiciones de memoria externa. Estos registros son de ancho byte y por este motivo la estructura Can_frame es una agrupación de la información a introducir o leer en estos registros.

7.2 Rutina de cada sección.

En este apartado se procede a describir el funcionamiento de cada una de las rutinas de atención a las interrupciones, rutinas de configuración y otras rutinas secundarias.

7.2.1 Sección CAN.

La sección CAN se compone de una rutina de configuración, una rutina de atención a la interrupción provocada por el controlador CAN y funciones para el acceso a los buffers de transmisión y recepción.

7.2.1.1 Configuración del controlador CAN.

Después de encender la tarjeta o tras un RESET del sistema, es necesario configurar el controlador CAN. También se puede reconfigurar durante el funcionamiento para adaptarse cambios del sistema.

La configuración del módulo CAN se realiza en el modo RESET interno de este controlador (no se debe confundir con el estado de RESET). Para acceder a este modo hay que solicitarlo expresamente activando el bit correspondiente del registro MOD.

El microcontrolador tiene que configurar los siguientes registros del controlador CAN:

- Registro de modo (MOD)
  - Filtro de aceptación
  - Modo Self-Test
  - Modo Listen- Only

- Registro divisor de reloj
  - BasicCAN/PeliCAN
  - Habilitación pin CLKOUT
  - By-Pass del comparador
  - Configuración alternativa de pin TX1

- Registros para mascara y código de aceptación
  - Bit-Rate
  - Número de muestras por bit y punto de muestra.

- Registro de configuración de pines de salida RX

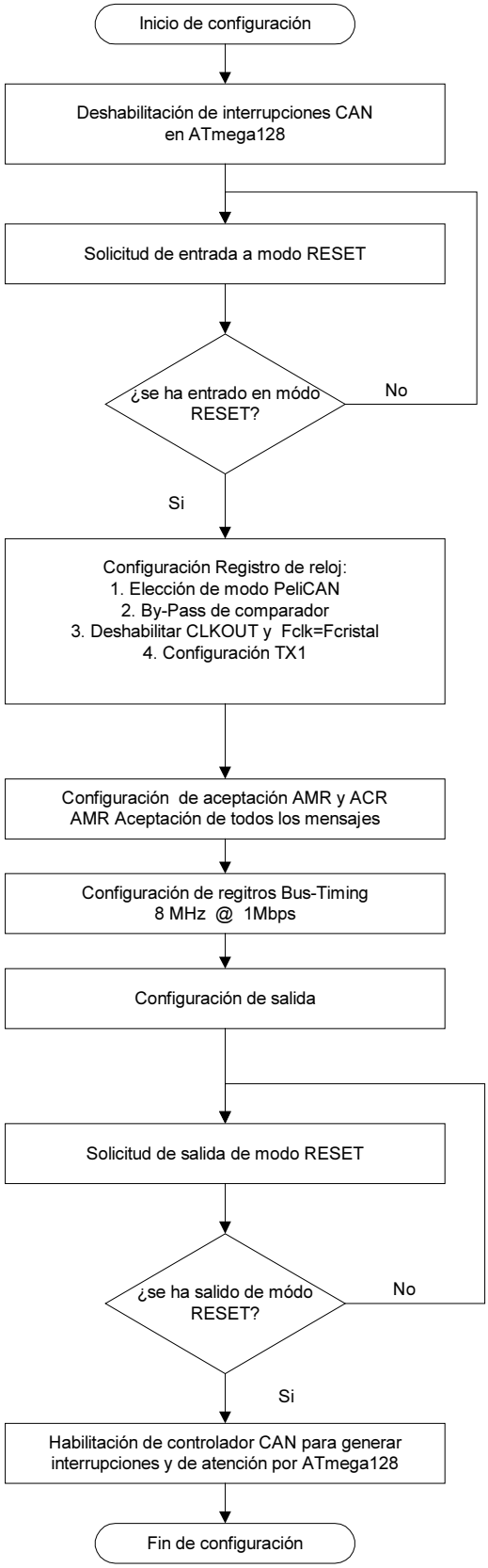


Figura 7.2.1.1 - 1: Configuración del controlador CAN

7.2.1.2 Atención de interrupciones.

El controlador CAN genera una interrupción a nivel bajo. El microcontrolador debe averiguar cual es la causa de esta interrupción, para ello lee el registro que contiene la causa de interrupción en el controlador CAN y se procede en función del valor de este registro.

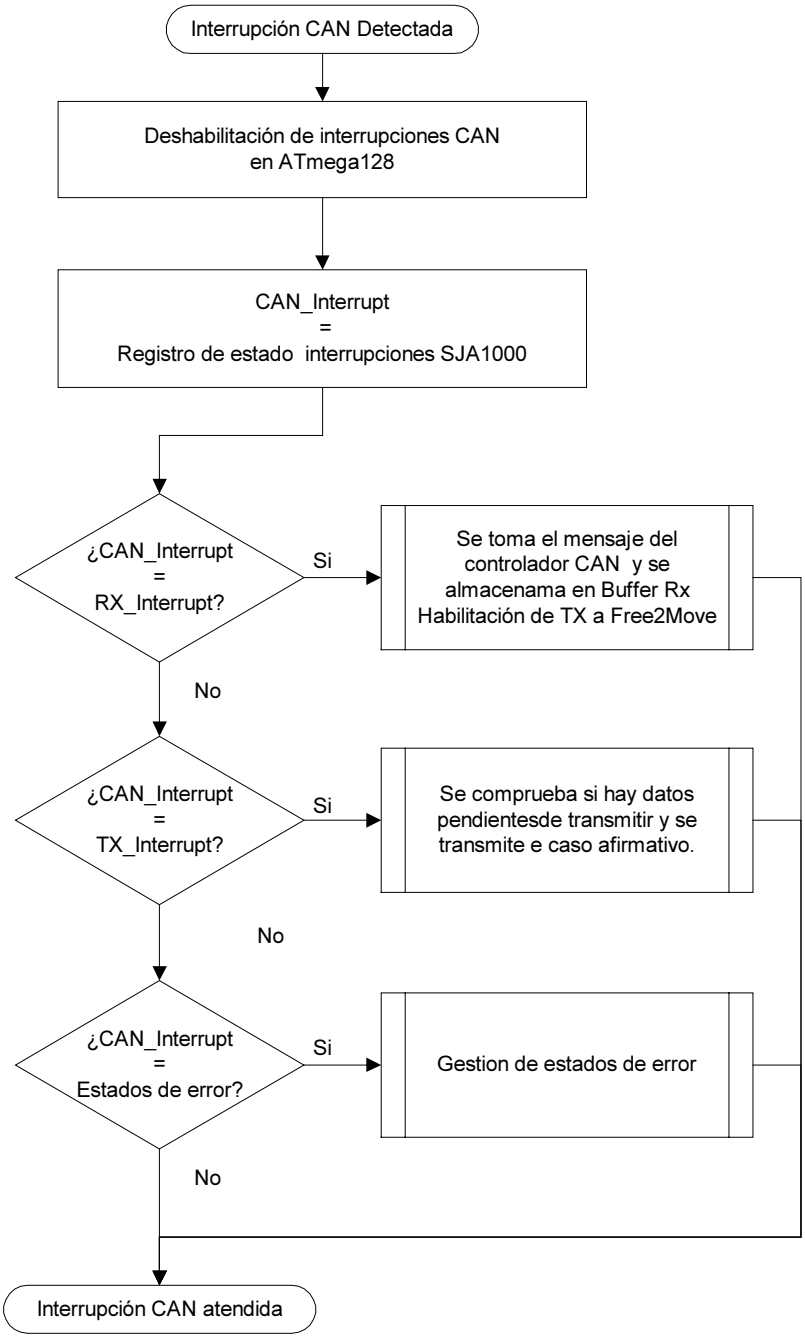


Figura 7.2.1.2 – 1:Atención de la interrupción CAN.

7.2.1.3 Almacenamiento / extracción mensajes.

Para el acceso a los mensajes CAN en los nodos terminales de la comunicación se dispones de las funciones CAN_TX_p(struct * Can_Frame) y CAN_RX_p(). CAN_TX_p almacena un mensaje en el buffer de transmisión si hay mensajes pendientes por transmitir. Si no hay mensajes pendientes para transmitir y el buffer del controlador CAN admite mensajes entonces se transmite directamente. El parámetro de CAN_TX_p es un puntero a una estructura que contiene los bytes a introducir en el buffer de transmisión del controlador. CAN_RX_p extrae un mensaje del controlador, previa comprobación de si hay mensajes pendientes de extraer del buffer de recepción CAN_Rx_Buf (CAN_RX_FIFO) , y devuelve un puntero a una variable que contiene el mensaje CAN.

7.2.1.3 Creación/Recuperación de mensajes.

Para facilitar el empleo de las funciones transmisión y recepción se han incorporado funciones que adaptan los mensajes que se quieren manejar a las tramas de comunicación con el controlador CAN. Por ejemplo un mensaje CAN a nivel de aplicación se compone de un identificador de señal , un bit de control, y un dato por ejemplo de 12 bits. La trama CAN (la información que se pasa/recibe al/del controlador CAN) tiene 3 bytes en los que se inserta la información anterior y su estructura se detalla en el capitulo 3. Para tener tramas pequeñas el dato de 12 bits se reparte entre el byte1 de datos y el byte de identificador 1 de la trama CAN. Para facilitar que el usuario no tenga que hacer estas conversiones se han diseñado las funciones Create_Msg y Get_Msg.

```
void * Create_Msg_u_byte(struct Can_Message_unsigned_byte *p_can_msg_temp);
void * Get_Msg_u_byte(struct Can_Frame rxframe);
void * Create_Msg_s_byte(struct Can_Message_signed_byte *p_can_msg_temp);
void * Get_Msg_s_byte(struct Can_Frame  rxframe);
void * Create_Msg_u_12bit(struct Can_Message_unsigned_12bit *p_can_msg_temp);
void * Get_Msg_u_12bit(struct Can_Frame rxframe);
void * Create_Msg_s_12bit(struct Can_Message_signed_12bit *p_can_msg_temp);
void * Get_Msg_s_12bit (struct Can_Frame  rxframe);
void * Create_Msg_Command(struct Can_Message_command *p_can_msg_temp);
void * Get_Msg_Command(struct Can_Frame rxframe);
```

A la función Create_Msg se le pasa el puntero estructura según el tipo de mensaje a transmitir y devuelve un puntero a una estructura Can_Frame. Este puntero se le pasa a la función CAN_TX_p.

De la misma manera CAN_RX_p() devuelve un puntero a una estructura Can_Frame, el contenido de esta estructura se pasa la función Get_Msg que devuelve un mensaje más manejable.

7.2.2 Sección Módulo Free2Move.

La comunicación con el módulo de radiofrecuencia Free2Move se realiza a través del puerto serie USART1. Se configura el interfaz serie USART1 para que provoque interrupciones asociadas a los eventos llegada de carácter al dispositivo USART1 y registro de datos vacío (con lo que se puede introducir un nuevo carácter para su transmisión).

Para no duplicar los buffers de USART1 y de mensajes CAN se han implementado dos maquinas de estado que adaptan el flujo de bytes de datos serie a los buffers de mensajes CAN. Conforme llegan caracteres al puerto serie se conforma un mensaje CAN que una vez completado se pasa al bufer de transmisión de mensajes CAN (CAN_TxBuf) para introducir este mensaje en el segmentode bus CAN local. Cada vez que se puede transmitir un mensaje CAN se acude a la rutina de interrupción asociada. En esta rutina hay una maquina de estado que transmite secuencialmente en cada llamada cada uno los tres bytes de un elemento del buffer de recepción CAN (CAN_RxBuf).

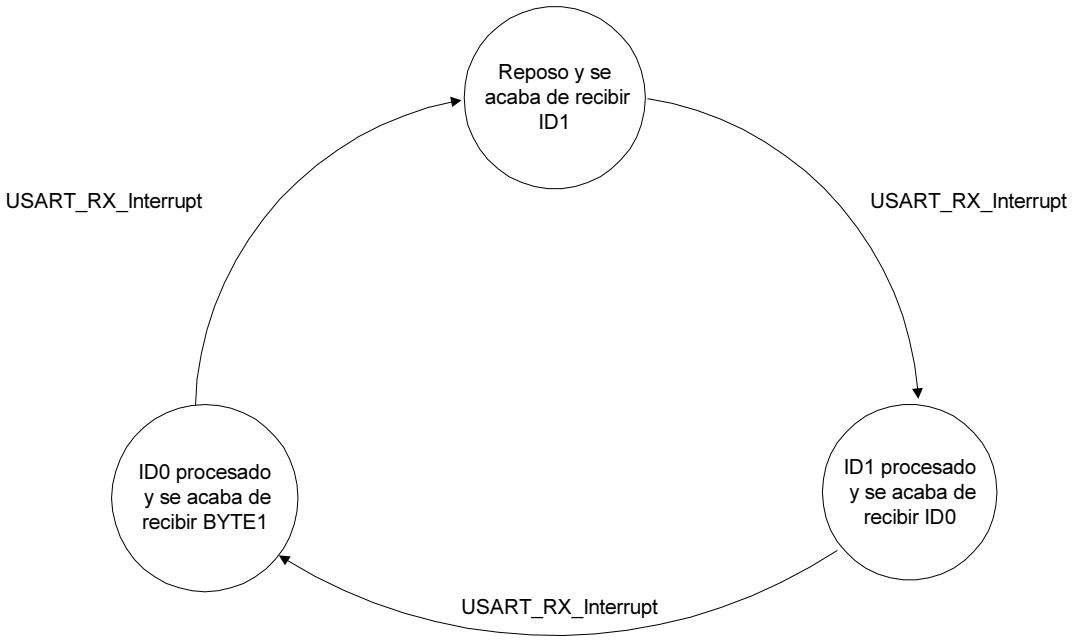


Figura 7.2.2 - 1: Máquina de estados asociada a la recepción de caracteres.

En la sección de código se puede apreciar como existe una comprobación para detectar si el mensaje va dirigido a este nodo CAN en concreto y si este es el caso pasarlo a un buffer local de mensajes destinados a este nodo.

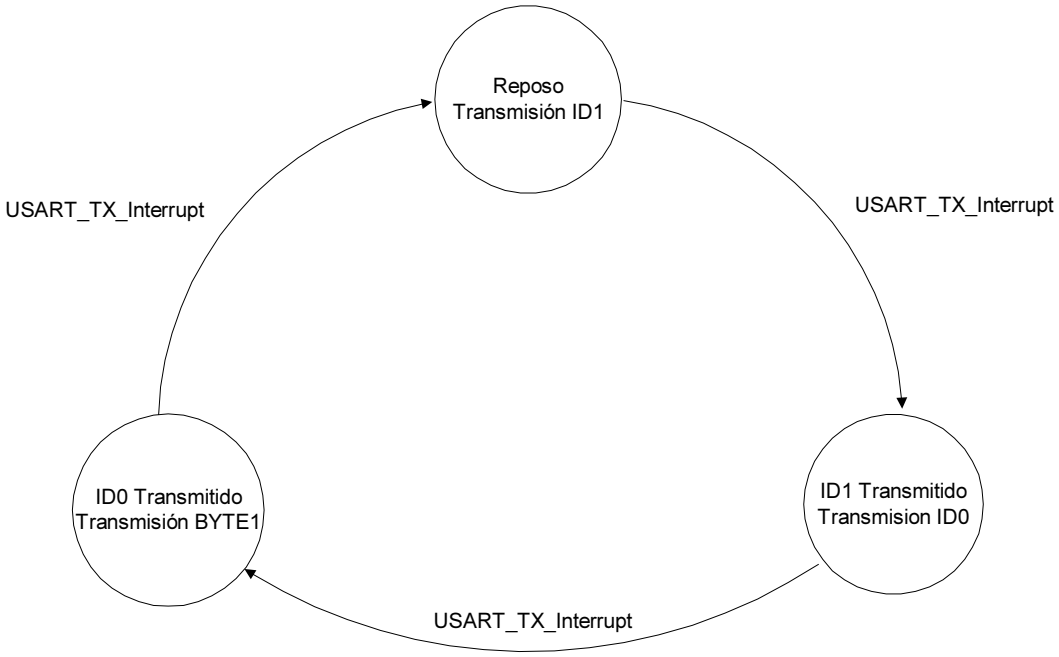


Figura 7.2.2 - 2: Máquina de estados asociada a la transmisión de caracteres.

7.2.3 Puerto Serie para depuración.

El puerto serie USART0 se usa para funciones de depuración. Esta configurado para trabajar con interrupciones si se recibe carácter o si se puede recargar el registro de transmisión. Estas interrupciones toman o introducen caracteres de / en los búferes USART0_TX_Buf y USART0_RX_Buf. Para facilitar la depuración y el empleo de funciones de iostdio.h se han redefinido las funciones putchar y getchar para que llamen USART0_Transmit y USART0_Receive.

USART0_Transmit introduce los bytes a transmitir en un buffer de transmisión al que acude la rutina de atención USART0_TX_Interrupt para recargar el registro de transmisión cada vez que este queda libre. Si detecta que este buffers está vacío se deshabilita esta interrupción que será habilitada por USART0_Transmit cuando introduzca algún carácter.

USART0_Receive extrae los caracteres del buffer de recepción que fueron introducidos por USART0_RX_Interrupt. Conviene comprobar si hay datos en el buffer antes de llamar a USART0_Receive porque está diseñada para esperar si está vacío el buffer en el momento de la llamada.

7.2.4 Funciones auxiliares del microcontrolador.

Se han diseñado funciones complementarias sobre todo para la depuración como el empleo de un temporizador para medidas de tiempo y funciones para el manejo de

los pulsadores y leds de señalización. En cuanto al programa principal como todas las tareas se realizan bajo el control de interrupciones realmente no tiene porque haber nada en el bucle principal, se ha introducido una función que hace parpadear un led para tener una señal visual de que la tarjeta está funcionando correctamente. Si requiere un procesado local de determinados mensajes dirigidos específicamente a este nodo se puede comprobar en cada iteración del bucle principal si algún mensaje en el buffer de mensajes CAN local y procesarlo adecuadamente, pero en principio no es necesario que las tarjetas WIRELESS_CAN procesen mensajes pues su función es actuar como interfaces o pasarelas.

7.2.5 Código fuente del microcontrolador.

A continuación se adjunta el código fuente de las tarjetas. El código es el mismo, salvo alguna pequeña diferencia como por ejemplo la dirección local CAN de cada tarjeta. Para hacer una compilación distinta se tienen 2 archivos de encabezados distintos para cada tarjeta, la elección de que encabezado se toma se hace en el archivo select.h que contiene una sentencia #define TIERRA o #define AIRE para hacer compilaciones distintas.

El código se ha realizado con el entorno IDE ImageCraft ICC for AVR (Professional), los ficheros comunes se hallan en una carpeta llamada comunes. En la carpeta AIRE está el fichero AIRE.C que contiene el código de la tarjeta que irá a bordo del helicóptero pues es mas ligera compacta que la tarjeta TIERRA que es una adaptación un modelo inicial.

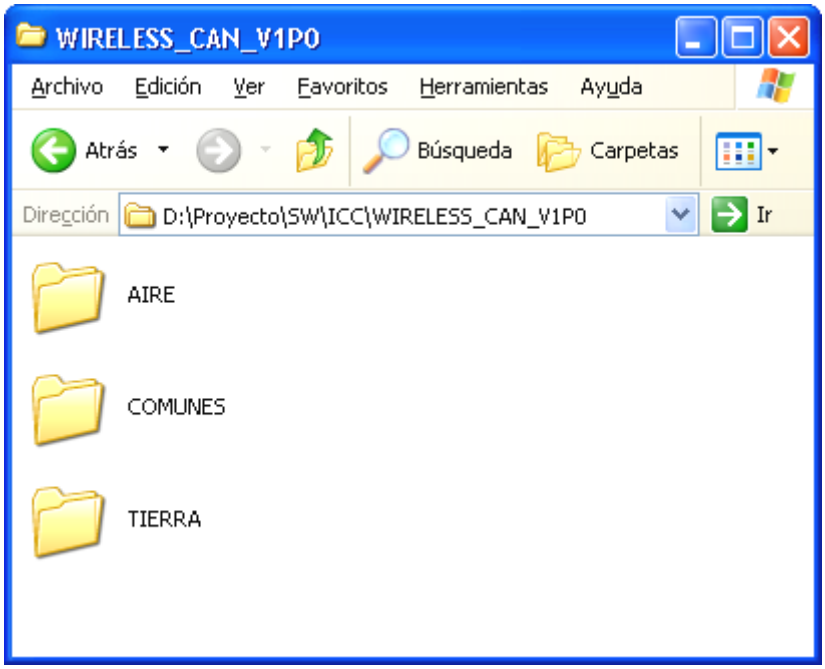


Figura 7.2.5 – 1: Carpetas del proyecto

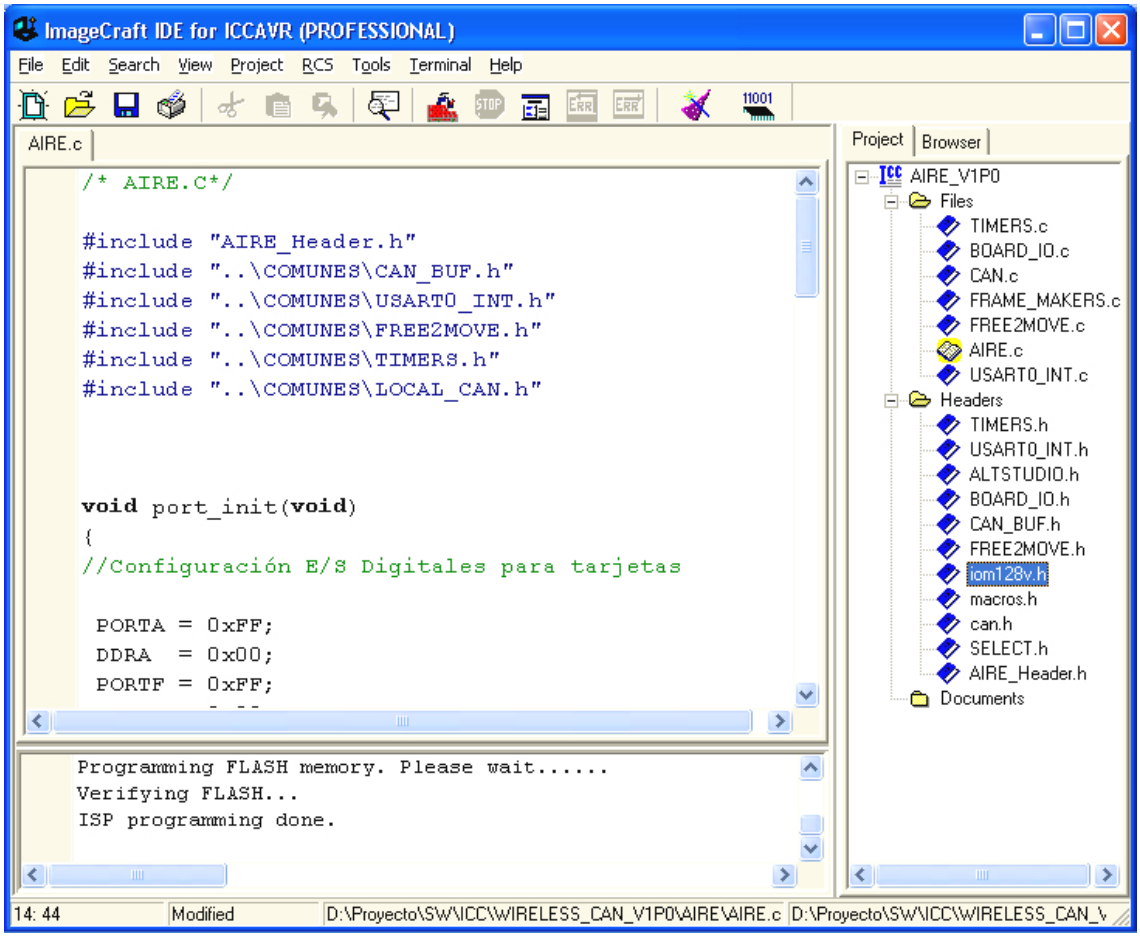


Figura 7.2.5 – 2: Vista del entorno ImageCraft ICC

En las páginas siguientes se adjuntan los archivos de programa.

Fichero AIRE.C

```
//ICC-AVR application builder : 01/04/2003 16:25:50
// Target : M128
// Crystal: 11.0592Mhz

#include "..\AIRE\AIRE_Header.h"
#include "..\AIRE\CAN_BUF.h"
#include "..\AIRE\USART0_INT.h"
#include "..\AIRE\FREE2MOVE.h"
#include "..\AIRE\TIMERS.h"
#include "..\AIRE\LOCAL_CAN.h"



void port_init(void)
{
//Configuración E/S Digitales para tarjetas con
//módulo Free2Move sin control de flujo.
//E/S iguales en ambas tarjetas
 PORTA = 0xFF;
 DDRA  = 0x00;
 PORTF = 0xFF;
 DDRF  = 0x00;
 PORTB = 0x3F;
 DDRB  = 0xC0;
 PORTC = 0xFF; //m103 output only
 DDRC  = 0xFF;
 PORTD = 0x00;
 DDRD  = 0x00;
 PORTE = 0xF7;
 DDRE  = 0x00;
 PORTG = 0x1F;
 DDRG  = 0x00;
}

void init_devices(void)
{
 //deshabilitación interrupciones
 CLI();
 XDIV  = 0x00; //divisor del cristal
 XMCRA = 0x00; //memoria externa
 XMCRB = 0x07;
 port_init();
 Delay();
 Swap_PORTB_LED(7);
 timer1_init();
 timer3_init();
 Swap_PORTB_LED(6);
 Delay();
 USART0_Init(5);
 Delay();
 printf("UARTINIT ok\n\r");

 MCUCR = 0x80;
 EICRA = 0x00; //interrupciones
 EICRB = 0x00; //
 EIMSK = 0x01;
 TIMSK = 0x04; //fuentes de interrupción timer
 ETIMSK = 0x04; //fuentes de interrupción timer extendidas
```

```
 SEI(); //habilitación de interrupciones
 //los perifericos están iniciados
}


void main (void)
{

char a,b,i,j,tmphead,tmptail;
char temp1, temp2;
char local_tmphead,local_tmptail;
int dato=0, data2tx=-512;

init_devices();
printf("\n\nProgram Restarted!!\n");
Delay();
Free2Move_Init_4(5);
printf("\n\rFree2Move_Init(5)\n");
CLI();
can_init();
SEI();

for(;;)
{

            Delay();
            Swap_PORTB_LED(6);


           printf("Segundos: %d ms: %d \r",s,ms);
           if (DataInReceiveBufferUSART0())
           {
                    putchar((temp1=getchar()));
                    //USART1_Transmit(temp1);
                    if (temp1 == 'r')
                    {
                    loop_time_request();

                    printf("\n\rloop_request_\n\r");

                    }

           }
           if (Message_CAN_Rx_Local_Buf())
           {
           Local_CAN_Proccess();
           }

}

}
```

Fichero AIRE_Header.h

```
/*Identificadores CAN*/
#define MULTICAST_CAN_ID   0b00000000

// Definiciones CAN
#define LOCAL_CAN_ID                0b11110000
#define OPPOSITE_CAN_ID    0b11100000
#define LOCAL_CAN_MASK              0b11111000
//Definiciones de lineas E/S distintas
#define DATA_ACTIVITY 1
#define CONNECTION 1
#define WATCHDOG 5
/* CAN Buffer Defines for M128*/
#define CAN_RX_BUFFER_SIZE 64     /* 2,4,8,16,32,64,128 or 256 bytes */
#define CAN_TX_BUFFER_SIZE 64     /* 2,4,8,16,32,64,128 or 256 bytes */
#define CAN_RX_BUFFER_MASK ( CAN_RX_BUFFER_SIZE - 1 )
#define CAN_TX_BUFFER_MASK ( CAN_TX_BUFFER_SIZE - 1 )
#if ( CAN_RX_BUFFER_SIZE & CAN_RX_BUFFER_MASK )
        #error RX buffer size is not a power of 2
#endif
#if ( CAN_TX_BUFFER_SIZE  & CAN_TX_BUFFER_MASK )
        #error RX buffer size is not a power of 2
#endif
#define CAN_RX_LOCAL_BUFFER_SIZE 64
#define CAN_TX_LOCAL_BUFFER_SIZE 64
#define CAN_RX_LOCAL_BUFFER_MASK ( CAN_RX_LOCAL_BUFFER_SIZE - 1 )
#define CAN_TX_LOCAL_BUFFER_MASK ( CAN_TX_LOCAL_BUFFER_SIZE - 1 )
#if ( CAN_RX_LOCAL_BUFFER_SIZE & CAN_RX_LOCAL_BUFFER_MASK )
        #error RX buffer size is not a power of 2
#endif
#if ( CAN_TX_LOCAL_BUFFER_SIZE  & CAN_TX_LOCAL_BUFFER_MASK )
        #error RX buffer size is not a power of 2
#endif

/* UART Buffer Defines */
#define USART0_RX_BUFFER_SIZE 256     /* 2,4,8,16,32,64,128 or 256 bytes */
#define USART0_TX_BUFFER_SIZE 256     /* 2,4,8,16,32,64,128 or 256 bytes */
#define USART0_RX_BUFFER_MASK ( USART0_RX_BUFFER_SIZE - 1 )
#define USART0_TX_BUFFER_MASK ( USART0_TX_BUFFER_SIZE - 1 )
#if ( USART0_RX_BUFFER_SIZE & USART0_RX_BUFFER_MASK )
        #error RX buffer size is not a power of 2
#endif
#if ( USART0_TX_BUFFER_SIZE & USART0_TX_BUFFER_MASK )
        #error TX buffer size is not a power of 2
#endif
/*Definiciones para estado LEDS*/
#define OFF 1
#define ON 0
#define PULSE 2
//Archivos necesarios
#include "..\comunes\iom128v.h"
#include "..\comunes\altstdio.h"
#define PeliCANMode
#include "..\comunes\can.h"
#include "..\comunes\macros.h"
#include "..\comunes\board_IO.h"
```

FICHERO CAN.C

```
/*Implementacion de funciones para tramas fijas
de 3 bytes con hasta 12 bits de información*/


#include "..\COMUNES\SELECT.h"

#if defined (AIRE)
#include "..\AIRE\AIRE_header.h"
#endif
#if defined (TIERRA)
#include "..\TIERRA\TIERRA_header.h"
#endif



extern unsigned char External_CAN_Memory[128];

struct Can_Message_unsigned_byte{
        unsigned char ID;
        unsigned char Frame_info;
        unsigned char u_byte;}
        can_msg_u_byte_in, *p_can_msg_u_byte_in,
        can_msg_u_byte_out, *p_can_msg_u_byte_out;

struct Can_Message_signed_byte{
        unsigned char ID;
        unsigned char Frame_info;
        signed char s_byte;}
        can_msg_s_byte_in, *p_can_msg_s_byte_in,
        can_msg_s_byte_out, *P_can_msg_s_byte_out;

/*los enteros de 12 bits se manejan como ints
para facilitar el trabajo y se conforman a 12 bits
para introducirlos en la trama*/

struct Can_Message_unsigned_12bit{
        unsigned char ID;
        unsigned char Frame_info;
        unsigned int u_12bit;}
        can_msg_u_12bit_in,
        *p_can_msg_u_12bit_in,
        can_msg_u_12bit_out,
        *p_can_msg_u_12bit_out;

struct Can_Message_signed_12bit{
        unsigned char ID;
        unsigned char Frame_info;
        signed int s_12bit;}
         can_msg_s_12bit_in,
                *p_can_msg_s_12bit_in,
         can_msg_s_12bit_out,
                *p_can_msg_s_12bit_out;


struct Can_Frame{
        unsigned char id1;
        unsigned char id0;
        unsigned char byte1;
```

```
}           can_frame_in,
            *can_frame_in_p,
            can_frame_out,
            *can_frame_out_p,
            can_frame,
            *can_frame_p,
            received_frame,
            *received_frame_p;


struct Can_Frame CAN_RxBuf[CAN_RX_BUFFER_SIZE];
volatile unsigned char CAN_RxHead;
volatile unsigned char CAN_RxTail;
struct Can_Frame CAN_TxBuf[CAN_TX_BUFFER_SIZE];
volatile unsigned char CAN_TxHead;
volatile unsigned char CAN_TxTail;

struct Can_Frame CAN_Rx_Local_Buf[CAN_RX_LOCAL_BUFFER_SIZE];
volatile unsigned char CAN_Rx_Local_Head;
volatile unsigned char CAN_Rx_Local_Tail;
struct Can_Frame CAN_Tx_Local_Buf[CAN_TX_LOCAL_BUFFER_SIZE];
volatile unsigned char CAN_Tx_Local_Head;
volatile unsigned char CAN_Tx_Local_Tail;
char FORCE_TX=0;

void *CAN_RX_p( void );
void CAN_TX_p(struct Can_Frame *frame2tx_p);
void CAN_Back_TX(struct Can_Frame *frame2tx_p);
unsigned char DataInCAN_RxBuffer( void );
unsigned char DataInCAN_TxBuffer( void );


char can_init(void)
/* external SDRAM enabled for port access to SJA1000
CAN Controller at absolute address 0x1100 */
{
//define mappings
 asm(
  ".area memory(abs)\n"
  ".org 0xF000\n"
  " _External_CAN_Memory:: .blkb 128\n"
  ".text\n"
          );
 can_configure();

          CAN_RxTail = 0;
          CAN_RxHead = 0;
          CAN_TxTail = 0;
          CAN_TxHead = 0;

 return 1;
}

void wr_CAN_Register(unsigned char CANRegister,unsigned char value)
{
 External_CAN_Memory[CANRegister]=value;
}

unsigned char rd_CAN_Register(unsigned char CANRegister)
{
```

```
 return External_CAN_Memory[CANRegister];
}

void can_configure(void) /* SJA1000 configuration function */
{
/*Solicitud de modo RESET para configuración*/
ModeControlReg = ModeControlReg | RM_RR_Bit ;
while((ModeControlReg & RM_RR_Bit ) == ClrByte)
{
/* no se cambia el resto de bits */
ModeControlReg = ModeControlReg | RM_RR_Bit ;
}
/*Configuración
                    modo PeliCAN
                    Comparador BY-PASS desactivado
                    Freloj=Fcristal
*/
ClockDivideReg = CANMode_Bit | CBP_Bit | DivBy1;
/*deshabilitción interrpciones CAN*/
InterruptEnReg = ClrIntEnSJA;

/*Aceptación total*/
AcceptCode0Reg = ClrByte;
AcceptCode1Reg = ClrByte;
AcceptCode2Reg = ClrByte;
AcceptCode3Reg = ClrByte;
/*se acepta cualquier identificador*/
AcceptMask0Reg = DontCare;
AcceptMask1Reg = DontCare;
AcceptMask2Reg = DontCare;
AcceptMask3Reg = DontCare;

/* configuración de velocidad */
/* bit-rate = 1 Mbit/s @ 8 MHz, bus  sampled 1 */
BusTiming0Reg = 0x00;
BusTiming1Reg = 0x10;

/*configuracion lieas salida TX*/
OutControlReg = Tx1Float | Tx0PshPull | NormalMode;

/*solicitud salida de modo RESET*/
ModeControlReg = ModeControlReg & 0xFE;
do{
        ModeControlReg = ClrByte;
} while((ModeControlReg & RM_RR_Bit ) != ClrByte);

ModeControlReg = ModeControlReg & 0xFE;
printf("ModeControlReg %X",ModeControlReg);

InterruptEnReg= InterruptEnReg |RIE_Bit|TIE_Bit;
InterruptEnReg= 0xFF;
}


void can_tx (unsigned char ID1,unsigned char ID0,unsigned char dato)
{

/*Transmisión de mensaje por espera activa*/
/*espera activa a buffer transmisión libre*/
```

```
do{
} while((StatusReg & TBS_Bit ) != TBS_Bit );
/*
Buffer de transmisión libre y se introduce datos
*/
TxFrameInfo = 0x01; /*Trama estandar DLC = 1*/
TxBuffer1 = ID1;
TxBuffer2 = ID0;
TxBuffer3 = dato;
/*Transmisión simple sin confirmación
por parte del receptor*/
CommandReg = TR_Bit | SRR_Bit;
}

void can_tx_frame (struct Can_Frame *frame_p)
{
/*Transmisión de mensaje por espera activa
de mensaje almancenado en estructura Can_Frame*/
/*espera activa a buffer transmisión libre*/
do{

} while((StatusReg & TBS_Bit ) != TBS_Bit );
TxFrameInfo = 0x01; /*Trama estandar DLC=8*/
TxBuffer1 = (*frame_p).id1;
TxBuffer2 = (*frame_p).id0;
TxBuffer3 = (*frame_p).byte1;
/*Transmisión simple sin confirmación
por parte del receptor*/
CommandReg = TR_Bit | SRR_Bit; /* Set Transmission Request bit */
}

/*Funciones para comprobar si hay datos en buffers TX o RX*/
unsigned char DataInCAN_RxBuffer( void )
{
        return ( CAN_RxHead != CAN_RxTail );
        /* Return 0 (FALSE) if the receive buffer is empty */
}

unsigned char DataInCAN_TxBuffer( void )
{
        return ( CAN_TxHead != CAN_TxTail );
        /* Return 0 (FALSE) if the receive buffer is empty */
}

/*
Transmisión de mensajes CAN.
Almacenamiento si el controlador CAN no acepta
momentaneamente mensajes.
*/
void CAN_TX_p(struct Can_Frame * frame2tx_p)
{
unsigned char tmphead;
WDR();
InterruptEnReg |= TIE_Bit; /* Enable TIE interrupt */

if(DataInCAN_TxBuffer() || ((StatusReg & TBS_Bit ) != TBS_Bit))
{
        /* Calculo de indice */
        tmphead = ( CAN_TxHead + 1 ) & CAN_TX_BUFFER_MASK;
        /* Espera de espacio libre */
```

```
        while ( tmphead == CAN_TxTail );
        /*Almacenamiento de mensaje*/
        CAN_TxBuf[tmphead].id1 = (*frame2tx_p).id1;
        CAN_TxBuf[tmphead].id0 = (*frame2tx_p).id0;
        CAN_TxBuf[tmphead].byte1 = (*frame2tx_p).byte1;
        /*Almacenamiento de nuevo indice*/
        CAN_TxHead = tmphead;
}
else
{
TxFrameInfo = 0x01; /* SFF (data), DLC=8 */
TxBuffer1 = (*frame2tx_p).id1;
TxBuffer2 = (*frame2tx_p).id0;
TxBuffer3 = (*frame2tx_p).byte1;
/* Start the transmission */
CommandReg = TR_Bit |SRR_Bit; /* Se fuerza la transmisión */

}

}



/*
Recepción de mensajes del buffer de recepción
*/
void * CAN_RX_p( void )
{
        unsigned char tmptail;
        WDR();

        /*Espera a que haya datos*/
        while ( CAN_RxHead == CAN_RxTail )
                ;
        /*Calculo de indice*/
        tmptail = ( CAN_RxTail + 1 ) & CAN_RX_BUFFER_MASK;

        /*Almacenamiento indice*/
        CAN_RxTail = tmptail;

        return  (void *)&(CAN_RxBuf[tmptail]);
        /* Return data */

}


/*Para transmitir mensajes CAN al otro segmento desde este nodo*/
void CAN_Back_TX(struct Can_Frame * frame2tx_p)
{
unsigned char tmphead;

        /* Calculo de indice */
        tmphead = ( CAN_RxHead + 1 ) & CAN_RX_BUFFER_MASK;
        /*Espera de espacio en buffer*/
        while ( tmphead == CAN_RxTail );
        /* Store data in buffer */
        CAN_RxBuf[tmphead].id1 = (*frame2tx_p).id1;
        CAN_RxBuf[tmphead].id0 = (*frame2tx_p).id0;
        CAN_RxBuf[tmphead].byte1 = (*frame2tx_p).byte1;
        CAN_RxHead = tmphead;
```

```
        UCSR1B |= (1<<UDRIE1);
}

/*Rutina de interrución asociada al comtrolador cAN*/
#pragma interrupt_handler int0_isr:iv_INT0
void int0_isr(void)
{
unsigned char CANInterrupt, first_byte;
unsigned char tmphead, tmptail,tmp_local_head;
CLI();
CANInterrupt = InterruptReg;
printf("CANINTERRUPT %X\n\r",CANInterrupt);
if (CANInterrupt == RI_Bit)
{
        /*Lectura de primer dato*/
        WDR();
        first_byte = RxBuffer1;

        if ((first_byte & LOCAL_CAN_MASK)==LOCAL_CAN_ID)
        {
        /* Calculo de indice */
        tmp_local_head = ( CAN_Rx_Local_Head + 1 )
                                & CAN_RX_LOCAL_BUFFER_MASK;
        CAN_Rx_Local_Head = tmp_local_head;

        if ( tmphead == CAN_Rx_Local_Tail )
        {
                printf(" Receive buffer overflow\n\r");
                /* ERROR! Receive buffer overflow */
        }
        /* Lectura de datos*/
        CAN_Rx_Local_Buf[tmp_local_head].id1 = RxBuffer1;
        CAN_Rx_Local_Buf[tmp_local_head].id0 = RxBuffer2;
        CAN_Rx_Local_Buf[tmp_local_head].byte1 = RxBuffer3;
        }
        else{

        /*Calculo de indice*/
        tmphead = ( CAN_RxHead + 1 ) & CAN_RX_BUFFER_MASK;
        CAN_RxHead = tmphead;      /* Almacenamiento de indice */

        if ( tmphead == CAN_RxTail )
        {
                printf(" Receive buffer overflow\n\r");
                /* ERROR! Receive buffer overflow */
        }
        /* Read the received data */
        CAN_RxBuf[tmphead].id1 = RxBuffer1;
        CAN_RxBuf[tmphead].id0 = RxBuffer2;
        CAN_RxBuf[tmphead].byte1 = RxBuffer3;

        }
/*Liberación e espacio en el buffer CAN del controlador*/
CommandReg = RRB_Bit;
/*Habilitación de interrupción USART1 pues hay
dato para que esta trabaje*/
UCSR1B |= (1<<UDRIE1);
}
else if ((CANInterrupt == TI_Bit)|| FORCE_TX==1)
{
```

```
        /* Se comprueba si es necesaria la transmisión */
        WDR();
        if (CAN_TxHead != CAN_TxTail)
        {
                /* Calculate buffer index */
                tmptail = ( CAN_TxTail + 1 ) & CAN_TX_BUFFER_MASK;
                CAN_TxTail = tmptail;      /* Store new index */

                TxFrameInfo = 0x01; /* SFF (data), DLC=2 */
                TxBuffer1 = CAN_TxBuf[tmptail].id1;
                TxBuffer2 = CAN_TxBuf[tmptail].id0;
                TxBuffer3 = CAN_TxBuf[tmptail].byte1;
                /*Trasnmisión simple si confirmación*/
                CommandReg = TR_Bit | SRR_Bit;
        }
        else
        {
                /*Si no hay datos se deshabilita USART1*/
                InterruptEnReg &= ~(TIE_Bit);
        }

FORCE_TX=0;
}
else if (CANInterrupt & EI_Bit)
{
ModeControlReg = ModeControlReg | RM_RR_Bit ;
while((ModeControlReg & RM_RR_Bit ) == ClrByte)
{
/* other bits than the reset mode/request bit are unchanged */
ModeControlReg = ModeControlReg | RM_RR_Bit ;
}
RxErrCountReg=0;
TxErrCountReg=0;
ModeControlReg = ModeControlReg & 0xFE;
do{
ModeControlReg = ClrByte;
} while((ModeControlReg & RM_RR_Bit ) != ClrByte);
printf("EI_Bit\n\r");
}
else if (CANInterrupt & DOI_Bit)
{
printf("DOI_Bit\n\r");
}
else if (CANInterrupt & WUI_Bit)
{
printf("WUI_Bit\n\r");
}
else if (CANInterrupt & EPI_Bit)
{
ModeControlReg = ModeControlReg | RM_RR_Bit ;
while((ModeControlReg & RM_RR_Bit ) == ClrByte)
{
/* other bits than the reset mode/request bit are unchanged */
ModeControlReg = ModeControlReg | RM_RR_Bit ;
}
RxErrCountReg=0;
TxErrCountReg=0;
ModeControlReg = ModeControlReg & 0xFE;
do{
ModeControlReg = ClrByte;
```

```
} while((ModeControlReg & RM_RR_Bit ) != ClrByte);
printf("EPI_Bit\n\r");
}
else if (CANInterrupt & ALI_Bit)
{
printf("ALI_Bit\n\r");
}
else if (CANInterrupt & BEI_Bit)
{
ModeControlReg = ModeControlReg & 0xFE;
printf("BEI_Bit\n\r");
}
else
printf("Otra INT\n\r");
SEI();
}



/*Función para comprobar si hay datos en buffer CAN_Rx_Local*/
unsigned char Message_CAN_Rx_Local_Buf( void )
{
        return ( CAN_Rx_Local_Head != CAN_Rx_Local_Tail );
        /*Se devuelve 0 si el buffer está vacío*/
}
```

Fichero CAN.h

Este fichero define constantes útiles para el trabajo con el controlador CAN.

```
#ifndef __CAN_H
#define __CAN_H



extern unsigned char External_CAN_Memory[128];
#define PeliCANMode

//Register and bit definitions for the SJA1000

/* definition for direct access to SRAM memory areas */
#define XBYTE External_CAN_Memory
/* address and bit definitions for the Mode & Control Register */
#define ModeControlReg (*(volatile char*) (XBYTE+0))
#define RM_RR_Bit 0x01 /* reset mode (request) bit */
#if defined (PeliCANMode)
#define LOM_Bit 0x02 /* listen only mode bit */
#define STM_Bit 0x04 /* self test mode bit */
#define AFM_Bit 0x08 /* acceptance filter mode bit */
#define SM_Bit 0x10 /* enter sleep mode bit */
#endif


/* address and bit definitions for the
Interrupt Enable & Control Register */

#if defined (PeliCANMode)
#define InterruptEnReg (*(volatile char*) (XBYTE+4)) /* PeliCAN mode */
#define RIE_Bit 0x01 /* receive interrupt enable bit */
```

```
#define TIE_Bit 0x02 /* transmit interrupt enable bit */
#define EIE_Bit 0x04 /* error warning interrupt enable bit */
#define DOIE_Bit 0x08 /* data overrun interrupt enable bit */
#define WUIE_Bit 0x10 /* wake-up interrupt enable bit */
#define EPIE_Bit 0x20 /* error passive interrupt enable bit */
#define ALIE_Bit 0x40 /* arbitration lost interr. enable bit*/
#define BEIE_Bit 0x80 /* bus error interrupt enable bit */
#else /* BasicCAN mode */
#define InterruptEnReg (*(volatile char*) (XBYTE+0)) /* Control Register */
#define RIE_Bit 0x02 /* Receive Interrupt enable bit */
#define TIE_Bit 0x04 /* Transmit Interrupt enable bit */
#define EIE_Bit 0x08 /* Error Interrupt enable bit */
#define DOIE_Bit 0x10 /* Overrun Interrupt enable bit */
#endif


/* address and bit definitions for the Command Register */

#define CommandReg (*(volatile char*) (XBYTE+1))
#define TR_Bit 0x01 /* transmission request bit */
#define AT_Bit 0x02 /* abort transmission bit */
#define RRB_Bit 0x04 /* release receive buffer bit */
#define CDO_Bit 0x08 /* clear data overrun bit */
#if defined (PeliCANMode)
#define SRR_Bit 0x10 /* self reception request bit */

#else /* BasicCAN mode */
#define GTS_Bit 0x10 /* goto sleep bit (BasicCAN mode) */
#endif


/* address and bit definitions for the Status Register */

#define StatusReg (*(volatile char*) (XBYTE+2))
#define RBS_Bit 0x01 /* receive buffer status bit */
#define DOS_Bit 0x02 /* data overrun status bit */
#define TBS_Bit 0x04 /* transmit buffer status bit */
#define TCS_Bit 0x08 /* transmission complete status bit */
#define RS_Bit 0x10 /* receive status bit */
#define TS_Bit 0x20 /* transmit status bit */
#define ES_Bit 0x40 /* error status bit */
#define BS_Bit 0x80 /* bus status bit */


/* address and bit definitions for the Interrupt Register */

#define InterruptReg (*(volatile char*) (XBYTE+3))
#define RI_Bit 0x01 /* receive interrupt bit */
#define TI_Bit 0x02 /* transmit interrupt bit */
#define EI_Bit 0x04 /* error warning interrupt bit */
#define DOI_Bit 0x08 /* data overrun interrupt bit */
#define WUI_Bit 0x10 /* wake-up interrupt bit */
#if defined (PeliCANMode)
#define EPI_Bit 0x20 /* error passive interrupt bit */
#define ALI_Bit 0x40 /* arbitration lost interrupt bit */
#define BEI_Bit 0x80 /* bus error interrupt bit */
#endif


/* address and bit definitions for the Bus Timing Registers */
```

```
#define BusTiming0Reg (*(volatile char*) (XBYTE+6))
#define BusTiming1Reg (*(volatile char*) (XBYTE+7))
#define SAM_Bit 0x80
/* sample mode bit
1 == the bus is sampled 3 times
0 == the bus is sampled once */


/* address and bit definitions for the Output Control Register */
#define OutControlReg (*(volatile char*) (XBYTE+8))
/* OCMODE1, OCMODE0 */

#define BiPhaseMode 0x00 /* bi-phase output mode */
#define NormalMode 0x02 /* normal output mode */
#define ClkOutMode 0x03 /* clock output mode */


/*output pin configuration for TX1 */

#define OCPOL1_Bit 0x20 /* output polarity control bit */
#define Tx1Float 0x00 /* configured as float */
#define Tx1PullDn 0x40 /* configured as pull-down */
#define Tx1PullUp 0x80 /* configured as pull-up */
#define Tx1PshPull 0xC0 /* configured as push/pull */


/* output configuration for TX0 */

#define OCPOLO_Bit 0x04 /* output polarity control bit */
#define Tx0Float 0x00 /* configured as float */
#define Tx0PullDn 0x08 /* configured as pull-down */
#define Tx0PullUp 0x10 /* configured as pull-up */
#define Tx0PshPull 0x18 /* configured as push/pull */


/* address definitions of Acceptance Code & Mask Registers */

#if defined (PeliCANMode)
#define AcceptCode0Reg (*(volatile char*) (XBYTE+16))
#define AcceptCode1Reg (*(volatile char*) (XBYTE+17))
#define AcceptCode2Reg (*(volatile char*) (XBYTE+18))
#define AcceptCode3Reg (*(volatile char*) (XBYTE+19))
#define AcceptMask0Reg (*(volatile char*) (XBYTE+20))
#define AcceptMask1Reg (*(volatile char*) (XBYTE+21))
#define AcceptMask2Reg (*(volatile char*) (XBYTE+22))
#define AcceptMask3Reg (*(volatile char*) (XBYTE+23))
#else /* BasicCAN mode */
#define AcceptCodeReg (*(volatile char*) (XBYTE+4))
#define AcceptMaskReg (*(volatile char*) (XBYTE+5))
#endif


/* address definitions of the Rx-Buffer */

#if defined (PeliCANMode)
#define RxFrameInfo (*(volatile char*) (XBYTE+16))
#define RxBuffer1 (*(volatile char*) (XBYTE+17))
#define RxBuffer2 (*(volatile char*) (XBYTE+18))
#define RxBuffer3 (*(volatile char*) (XBYTE+19))
```

```
#define RxBuffer4 (*(volatile char*) (XBYTE+20))
#define RxBuffer5 (*(volatile char*) (XBYTE+21))
#define RxBuffer6 (*(volatile char*) (XBYTE+22))
#define RxBuffer7 (*(volatile char*) (XBYTE+23))
#define RxBuffer8 (*(volatile char*) (XBYTE+24))
#define RxBuffer9 (*(volatile char*) (XBYTE+25))
#define RxBuffer10 (*(volatile char*) (XBYTE+26))
#define RxBuffer11 (*(volatile char*) (XBYTE+27))
#define RxBuffer12 (*(volatile char*) (XBYTE+28))
#else /* BasicCAN mode */
#define RxBuffer1 (*(volatile char*) (XBYTE+20))
#define RxBuffer2 (*(volatile char*) (XBYTE+21))
#define RxBuffer3 (*(volatile char*) (XBYTE+22))
#define RxBuffer4 (*(volatile char*) (XBYTE+23))
#define RxBuffer5 (*(volatile char*) (XBYTE+24))
#define RxBuffer6 (*(volatile char*) (XBYTE+25))
#define RxBuffer7 (*(volatile char*) (XBYTE+26))
#define RxBuffer8 (*(volatile char*) (XBYTE+27))
#define RxBuffer9 (*(volatile char*) (XBYTE+28))
#define RxBuffer10 (*(volatile char*) (XBYTE+29))
#endif


/* address definitions of the Tx-Buffer */
#if defined (PeliCANMode)
/* write only addresses */
#define TestReg (*(volatile char*) (XBYTE+9))

#define TxFrameInfo (*(volatile char*) (XBYTE+16))
#define TxBuffer1 (*(volatile char*) (XBYTE+17))
#define TxBuffer2 (*(volatile char*) (XBYTE+18))
#define TxBuffer3 (*(volatile char*) (XBYTE+19))
#define TxBuffer4 (*(volatile char*) (XBYTE+20))
#define TxBuffer5 (*(volatile char*) (XBYTE+21))
#define TxBuffer6 (*(volatile char*) (XBYTE+22))
#define TxBuffer7 (*(volatile char*) (XBYTE+23))
#define TxBuffer8 (*(volatile char*) (XBYTE+24))
#define TxBuffer9 (*(volatile char*) (XBYTE+25))
#define TxBuffer10 (*(volatile char*) (XBYTE+26))
#define TxBuffer11 (*(volatile char*) (XBYTE+27))
#define TxBuffer12 (*(volatile char*) (XBYTE+28))

/* read only addresses */
#define TxFrameInfoRd (*(volatile char*) (XBYTE+96))
#define TxBufferRd1 (*(volatile char*) (XBYTE+97))
#define TxBufferRd2 (*(volatile char*) (XBYTE+98))
#define TxBufferRd3 (*(volatile char*) (XBYTE+99))
#define TxBufferRd4 (*(volatile char*) (XBYTE+100))
#define TxBufferRd5 (*(volatile char*) (XBYTE+101))
#define TxBufferRd6 (*(volatile char*) (XBYTE+102))
#define TxBufferRd7 (*(volatile char*) (XBYTE+103))
#define TxBufferRd8 (*(volatile char*) (XBYTE+104))
#define TxBufferRd9 (*(volatile char*) (XBYTE+105))
#define TxBufferRd10 (*(volatile char*) (XBYTE+106))
#define TxBufferRd11 (*(volatile char*) (XBYTE+107))
#define TxBufferRd12 (*(volatile char*) (XBYTE+108))
#else /* BasicCAN mode */
#define TxBuffer1 (*(volatile char*) (XBYTE+10))
#define TxBuffer2 (*(volatile char*) (XBYTE+11))
#define TxBuffer3 (*(volatile char*) (XBYTE+12))
```

```
#define TxBuffer4 (*(volatile char*) (XBYTE+13))
#define TxBuffer5 (*(volatile char*) (XBYTE+14))
#define TxBuffer6 (*(volatile char*) (XBYTE+15))
#define TxBuffer7 (*(volatile char*) (XBYTE+16))
#define TxBuffer8 (*(volatile char*) (XBYTE+17))
#define TxBuffer9 (*(volatile char*) (XBYTE+18))
#define TxBuffer10 (*(volatile char*) (XBYTE+19))
#endif


/* address definitions of Other Registers */

#if defined (PeliCANMode)
#define ArbLostCapReg (*(volatile char*) (XBYTE+11))
#define ErrCodeCapReg (*(volatile char*) (XBYTE+12))
#define ErrWarnLimitReg (*(volatile char*) (XBYTE+13))
#define RxErrCountReg (*(volatile char*) (XBYTE+14))
#define TxErrCountReg (*(volatile char*) (XBYTE+15))
#define RxMsgCountReg (*(volatile char*) (XBYTE+29))
#define RxBufStartAdr (*(volatile char*) (XBYTE+30))
#endif


/* address and bit definitions for the Clock Divider Register */

#define ClockDivideReg (*(volatile char*) (XBYTE+31))
#define DivBy1 0x07 /* CLKOUT = oscillator frequency */
#define DivBy2 0x00 /* CLKOUT = 1/2 oscillator frequency */
#define ClkOff_Bit 0x08 /* clock off bit,control of the CLK OUT pin */
#define RXINTEN_Bit 0x20 /* pin TX1 used for receive interrupt */
#define CBP_Bit 0x40 /* CAN comparator bypass control bit */
#define CANMode_Bit 0x80 /* CAN mode definition bit */
#define ClkOutMode 0x03 /* clock output mode */
/* output pin configuration for TX1 */

#define OCPOL1_Bit 0x20 /* output polarity control bit */
#define Tx1Float 0x00 /* configured as float */
#define Tx1PullDn 0x40 /* configured as pull-down */
#define Tx1PullUp 0x80 /* configured as pull-up */
#define Tx1PshPull 0xC0 /* configured as push/pull */

/* output pin configuration for TX0 */

#define OCPOL0_Bit 0x04 /* output polarity control bit */
#define Tx0Float 0x00 /* configured as float */
#define Tx0PullDn 0x08 /* configured as pull-down */


//Definitions of Variables and Constants for the Examples


/*- definition of hardware / software connections ----------------------*/
/* controller: S87C654; CAN controller: SJA1000(see Figure 3 on page 11)*/
/* ChipSelect for the SJA1000 to ground*/
#define SJAIntInp int0 /* external interrupt 0 (from SJA1000) */
#define SJAIntEn EX0 /* external interrupt 0 enable flag */
/*- definition of used constants ---------------------------------------*/
#define YES 1
#define NO 0
#define ENABLE 1
```

```
#define DISABLE 0
#define ENABLE_N 0
#define DISABLE_N 1
#define INTLEVELACT 0
#define INTEDGEACT 1
#define PRIORITY_LOW 0
#define PRIORITY_HIGH 1
/* default (reset) value for register content, clear register */
#define ClrByte 0x00
/* constant: clear Interrupt Enable Register */
#if defined (PeliCANMode)
#define ClrIntEnSJA ClrByte
#else
#define ClrIntEnSJA ClrByte | RM_RR_Bit /* preserve reset request */
#endif
/* definitions for the acceptance code and mask register */
#define DontCare 0xFF
/*- definition of bus timing values for different examples -------*/
/* bus timing values for the example given in (AN97046)
- bit-rate : 250 kBit/s
- oscillator frequency : 24 MHz, 1,0%
- maximum propagation delay : 1630 ns
- minimum requested propagation delay : 120 ns */
#define PrescExample 0x02 /* baud rate prescaler : 3 */
#define SJWExample 0xC0 /* SJW : 4 */
#define TSEG1Example 0x0A /* TSEG1 : 11 */
#define TSEG2Example 0x30 /* TSEG2 : 4 */
/* bus timing values for
- bit-rate : 1 MBit/s
- oscillator frequency : 24 MHz, 0,1%
- maximum tolerated propagation delay : 747 ns
- minimum requested propagation delay : 45 ns */
#define Presc_MB_24 0x00 /* baud rate prescaler : 1 */
#define SJW_MB_24 0x00 /* SJW : 1 */

#define TSEG1_MB_24 0x08 /* TSEG1 : 9 */
#define TSEG2_MB_24 0x10 /* TSEG2 : 2 */
/* bus timing values for
- bit-rate : 100 kBit/s
- oscillator frequency : 24 MHz, 1,0%
- maximum tolerated propagation delay : 4250 ns
- minimum requested propagation delay : 100 ns */
#define Presc_kB_24 0x07 /* baud rate prescaler : 8 */
#define SJW_kB_24 0xC0 /* SJW : 4 */
#define TSEG1_kB_24 0x09 /* TSEG1 : 10 */
#define TSEG2_kB_24 0x30 /* TSEG2 : 4 */
/* bus timing values for
- bit-rate : 1 MBit/s
- oscillator frequency : 16 MHz, 0,1%
- maximum tolerated propagation delay : 623 ns
- minimum requested propagation delay : 23 ns */
#define Presc_MB_16 0x00 /* baud rate prescaler : 1 */
#define SJW_MB_16 0x00 /* SJW : 1 */
#define TSEG1_MB_16 0x04 /* TSEG1 : 5 */
#define TSEG2_MB_16 0x10 /* TSEG2 : 2 */
/* bus timing values for
- bit-rate : 100 kBit/s
- oscillator frequency : 16 MHz, 1,0%
- maximum tolerated propagation delay : 4450 ns
- minimum requested propagation delay : 500 ns */
```

```
#define Presc_kB_16 0x04 /* baud rate prescaler : 5 */
#define SJW_kB_16 0xC0 /* SJW : 4 */
#define TSEG1_kB_16 0x0A /* TSEG1 : 11 */
#define TSEG2_kB_16 0x30 /* TSEG2 : 4 */
/*- end of definitions of bus timing values ----------------------*/


#endif



void wr_CAN_Register(unsigned char CANRegister,unsigned char value);
unsigned char rd_CAN_Register(unsigned char CANRegister);
void can_configure(void);


extern unsigned char External_CAN_Memory[128];
```

FICHERO FRAME_MAKERS.h

```
/*Implementacion de funciones para tramas fijas
de 3 bytes con hasta 12 bits de información*/

#define PeliCANMode

#include "..\COMUNES\SELECT.h"

#if defined (AIRE)
#include "..\AIRE\AIRE_header.h"
#endif
#if defined (TIERRA)
#include "..\TIERRA\TIERRA_header.h"
#endif

#include "..\COMUNES\can.h"
#include "..\COMUNES\CAN_BUF.h"
#include "..\COMUNES\LOCAL_CAN.h"


void * Create_Msg_u_byte(struct Can_Message_unsigned_byte *p_can_msg_temp)
{
char temp;
static struct Can_Frame frame2tx;
frame2tx.id1 = (*p_can_msg_temp).ID;
frame2tx.id0 = (*p_can_msg_temp).Frame_info;
frame2tx.byte1 = (*p_can_msg_temp).u_byte;
return (void *) &frame2tx;
}


void * Get_Msg_u_byte(struct Can_Frame rxframe)
{
static struct Can_Message_unsigned_byte can_msg_temp;
can_msg_temp.ID = rxframe.id1 & 0xF8;
can_msg_temp.Frame_info = rxframe.id0;
can_msg_temp.u_byte = rxframe.byte1;
return (void *) &can_msg_temp;
}
```

```
void * Create_Msg_s_byte(struct Can_Message_signed_byte *p_can_msg_temp)
{
static struct Can_Frame frame2tx;
frame2tx.id1 = (*p_can_msg_temp).ID;
frame2tx.id0 = (*p_can_msg_temp).Frame_info;
frame2tx.byte1 = (*p_can_msg_temp).s_byte;
return (void *) &frame2tx;
}


void * Get_Msg_s_byte(struct Can_Frame  rxframe)
{
static struct Can_Message_signed_byte can_msg_temp;
can_msg_temp.ID = rxframe.id1 & 0xF8;
can_msg_temp.Frame_info = rxframe.id0;
can_msg_temp.s_byte = rxframe.byte1;
return (void *) &can_msg_temp;
}

void * Create_Msg_u_12bit(struct Can_Message_unsigned_12bit *p_can_msg_temp)
{
unsigned int temp;
static struct Can_Frame frame2tx;
temp = 0x0780 &((*p_can_msg_temp).u_12bit >> 1);
frame2tx.id1 = (0xF8 &(*p_can_msg_temp).ID) | (unsigned char)((0xFF00 & temp) >> 8);
frame2tx.id0 = (*p_can_msg_temp).Frame_info | (unsigned char) (0xFF00 & temp);
frame2tx.byte1 = (unsigned char)(0x00FF & (*p_can_msg_temp).u_12bit);
return (void *) &frame2tx;
}


void * Get_Msg_u_12bit(struct Can_Frame rxframe)
{
unsigned int temp;
static struct Can_Message_unsigned_12bit can_msg_temp;
temp = (((unsigned int)(rxframe.id1 & 0x07)) << 8);
temp = (temp & 0xFF00) |(0x00FF & rxframe.id0);
temp = temp << 1;
can_msg_temp.ID = rxframe.id1 & 0xF8;
can_msg_temp.Frame_info = rxframe.id0;
can_msg_temp.u_12bit = (temp & 0xFF00) | rxframe.byte1;
return (void *) &can_msg_temp;
}


void * Create_Msg_s_12bit(struct Can_Message_signed_12bit *p_can_msg_temp)
{
int temp;
static struct Can_Frame frame2tx;
WDR();
temp = (*p_can_msg_temp).s_12bit;
//if((temp < -2048 ) || (temp > 2047 ))
//printf("Only Accept -2048<=int<=2047 (12 bits)");
temp = temp >> 1;
temp = (temp & 0x0780);
frame2tx.id1 = (0xF8 &(*p_can_msg_temp).ID) | (unsigned char)((0xFF00 & temp) >> 8);
frame2tx.id0 = (0x60 & (*p_can_msg_temp).Frame_info) | ((unsigned char) (0x0080 & temp));
frame2tx.byte1 = (unsigned char)(0x00FF & (*p_can_msg_temp).s_12bit);
```

```
return (void *) &frame2tx;
}


void * Get_Msg_s_12bit (struct Can_Frame  rxframe)
{
unsigned char temp_h, temp_l;
int temp;
static struct Can_Message_signed_12bit can_msg_temp;
WDR();
temp = (((unsigned int)(rxframe.id1 & 0x07)) << 8);
temp = (temp & 0xFF00) |(0x0080 & rxframe.id0);
temp = temp << 1;
temp = temp | rxframe.byte1;
can_msg_temp.ID = rxframe.id1 & 0xF8;
can_msg_temp.Frame_info = rxframe.id0;
can_msg_temp.s_12bit = (temp & 0xFF00) | rxframe.byte1;
if ((temp & 0x0800) == 0x0800){
temp = temp | 0xF000;}
can_msg_temp.ID = rxframe.id1 & 0xF8;
can_msg_temp.Frame_info = rxframe.id0;
can_msg_temp.s_12bit = temp;
return (void *) &can_msg_temp;
}
```

FICHERO CAN_BUF.h

Este fichero contiene la declaración de las variables externas de can.c para el acceso de otros ficheros a dichas variables y la declaración de funciones relacionadas con mensajes CAN tanto del fichero can.c como de FRAME_MAKERS.c. Por este motivo no se va listar pues son las mismas definiciones de variables can.c con **extern** delante para notificar que son variables externas y la declaraciones de funciones definidas en otros ficheros.

FICHERO ALT_STDIO**.h**

Este fichero es similar a stdio.h salvo que no se declaran **getchar** y **putchar** pues estas funciones se sobrescriben para que trabajen con la USART0 llamando a USART0_Receive y USART0_Transmit repectivamente.

FICHERO USART0_INT.c

```
/* Rutinas para el configuración y empleo del
dispositivo USART0 del microcontrolador ATmega128*/

/* Includes */
#include "..\COMUNES\SELECT.h"
#if defined (AIRE)
#include "..\AIRE\AIRE_Header.h"
#endif
```

```
#if defined (TIERRA)
#include "..\TIERRA\TIERRA_header.h"
#endif

/* UART Buffer Defines */
#define USART0_RX_BUFFER_SIZE 256     /* 2,4,8,16,32,64,128 or 256 bytes */
#define USART0_TX_BUFFER_SIZE 256     /* 2,4,8,16,32,64,128 or 256 bytes */
#define USART0_RX_BUFFER_MASK ( USART0_RX_BUFFER_SIZE - 1 )
#define USART0_TX_BUFFER_MASK ( USART0_TX_BUFFER_SIZE - 1 )
#if ( USART0_RX_BUFFER_SIZE & USART0_RX_BUFFER_MASK )
        #error RX buffer size is not a power of 2
#endif
#if ( USART0_TX_BUFFER_SIZE & USART0_TX_BUFFER_MASK )
        #error TX buffer size is not a power of 2
#endif

/* Variables del buffer */
static unsigned char USART0_RxBuf[USART0_RX_BUFFER_SIZE];
static volatile unsigned char USART0_RxHead;
static volatile unsigned char USART0_RxTail;
static unsigned char USART0_TxBuf[USART0_TX_BUFFER_SIZE];
static volatile unsigned char USART0_TxHead;
static volatile unsigned char USART0_TxTail;

/* Prototypes */
unsigned char USART0_Receive( void );
void USART0_Transmit( unsigned char data );


/* Configuración USART0*/
void USART0_Init( unsigned int baudrate )
{
        unsigned char x;

        UCSR0B=0;
        /* Configuración de Baud-Rate*/
        /* UBRR = 5 para 115200*/
        UBRR0H = (unsigned char) (baudrate>>8);
        UBRR0L = (unsigned char) baudrate;

        /* Habilitación de Transmisor, Receptor e interrupción
        cuando se recibe un carácter*/
        UCSR0B = ( ( 1 << RXCIE0 ) | ( 1 << RXEN0 )|( 1 << TXEN0 ));

        /*1 Bit Stop*/
        UCSR0C = (3<<UCSZ00);

        /* Inicio de indices de buffer de TX y RX*/
        x = 0;
        USART0_RxTail = x;
        USART0_RxHead = x;
        USART0_TxTail = x;
        USART0_TxHead = x;
}


/* Rutina de manejo de interrupción por caracter recibido*/
#pragma interrupt_handler USART0_RX_interrupt:iv_USART0_RXC
void USART0_RX_interrupt( void )
{
```

```
	unsigned char data;
	unsigned char tmphead;

	/* lectura del dato de entrada */
	data = UDR0;
	/* Calculo del indice */
	tmphead = ( USART0_RxHead + 1 ) & USART0_RX_BUFFER_MASK;
	USART0_RxHead = tmphead;

	if ( tmphead == USART0_RxTail )
	{
		/* ERROR! Receive buffer overflow */
	}

	USART0_RxBuf[tmphead] = data; /* Dato a bufer */
}



/* Rutina de manejo de interrupción que avisa que es posible TX*/
#pragma interrupt_handler USART0_TX_interrupt:iv_USART0_UDRE
void USART0_TX_interrupt( void )
{
	unsigned char tmptail;

	/* Chequeo de si es necesario TX */
	if ( USART0_TxHead != USART0_TxTail )
	{
		/* calculo de indice */
		tmptail = ( USART0_TxTail + 1 ) & USART0_TX_BUFFER_MASK;
		USART0_TxTail = tmptail;

		UDR0 = USART0_TxBuf[tmptail];  /* carga del registro */
	}
	else
	{
		/*Si no hay que TX se deshabilita la interrupción*/
		UCSR0B &= ~(1<<UDRIE0);
	}
}

/* Acceso a buffers de TX y RX*/
unsigned char USART0_Receive( void )
{
	unsigned char tmptail;

	while ( USART0_RxHead == USART0_RxTail )  /* Wait for incomming data */
		;
	tmptail = ( USART0_RxTail + 1 ) & USART0_RX_BUFFER_MASK;/* Calculate buffer
index */

	USART0_RxTail = tmptail;		/* Store new index */

	return USART0_RxBuf[tmptail];		/* Return data */
}

void USART0_Transmit( unsigned char data )
{
	unsigned char tmphead;
```

```
        /*Cálculo del indice*/
        tmphead = ( USART0_TxHead + 1 ) & USART0_TX_BUFFER_MASK;
        while ( tmphead == USART0_TxTail );

        /*Almacenamiento en buffer*/
        USART0_TxBuf[tmphead] = data;
        /*Actualización del indice*/
        USART0_TxHead = tmphead;
        /*Se habilita la int TX porque hay dato para TX*/
        UCSR0B |= (1<<UDRIE0);
}


/*Función para chequear si hay datos en buffer RX */
unsigned char DataInReceiveBufferUSART0( void )
{
        return ( USART0_RxHead != USART0_RxTail );
        /* Se devuelve 0 si esta vacío */
}

/*Redefinición de putchar y getchar para que
sean empleadas por funciones stdio*/
void putchar (unsigned char c)
{
  USART0_Transmit(c);
}

unsigned char getchar (void)
{
  return USART0_Receive();
}
```

FICHERO USART0_INT.h

Este fichero solo contiene la declaración de las variables y funciones, que ya aparecen en USART0_INT.c

FICHERO FREE2MOVE.c

```
/*
Interfaz para el módulo Free2Move
*/

/* Includes */
#include "..\COMUNES\SELECT.h"
#if defined (AIRE)
#include "..\AIRE\AIRE_Header.h"
#endif
#if defined (TIERRA)
#include "..\TIERRA\TIERRA_header.h"
#endif
#include "..\COMUNES\CAN_BUF.h"
#include "..\COMUNES\TIMERS.h"
```

```
/*USART1 Estados */
#define IDLE 0
#define CAN_ID1_BYTE_RECIBIDO 1
#define CAN_ID1_BYTE_TRANSMITIDO 1
#define CAN_ID0_BYTE_RECIBIDO 2
#define CAN_ID0_BYTE_TRANSMITIDO 2


/*Variables globales*/

unsigned char USART1_Rx_Status=IDLE,
                        USART1_Tx_Status=IDLE;
unsigned int  USART1_Transmited_Bytes,
                        USART1_Received_Bytes;
unsigned char DATA_ACTIVITY_Status=OFF,
                        CONNECTION_Status=OFF;
struct mensaje{
                char ID1;
                char ID0;
                char BYTE1;
                } mensaje_entrante = {0,0,0};

/*Prototipos*/


/* Configuración del dispositivo serie interfaz
del módulo Free2Move*/

void USART1_Init( unsigned int baudrate )
{
        /*Deshabilitación de USART0 durante configuración*/
        UCSR1B = 0x00;
        UCSR1A = 0x00;
        UCSR1C = 0x00;

       /* Configuración de Baud-Rate*/
       UBRR1H = (unsigned char) (baudrate>>8);
       UBRR1L = (unsigned char) baudrate;

       /* Habilitación de TX, RX e
       interrupción por caracter recibido*/
       UCSR1B = ( ( 1 << RXCIE1 )
                        | ( 1 << TXEN1 )
                        | ( 1 << RXEN1 )  );

       /* 8 bits de datos y 1 stop*/
       UCSR1C = (3<<UCSZ10);

       /*Inicio de maquina de estado*/
       USART1_Rx_Status=IDLE;
       USART1_Tx_Status=IDLE;
}

/* Maquina de estado para recibir mensajes CAN
via serie e introdudirlos correctamente en buffer CAN
para transmitir al segmento CAN*/
#pragma interrupt_handler USART1_RX_interrupt:iv_USART1_RXC
void USART1_RX_interrupt( void )
{
       unsigned char data, temp;
```

```
unsigned char tmphead, tmp_local_head;
static unsigned char CAN_Length=0, tmp1,tmp2;

CLI();
PORTD &= ~(1 << DATA_ACTIVITY);
/* Read the received data */
data = UDR1;
USART1_Received_Bytes++;
switch (USART1_Rx_Status)
{
case IDLE:
        /*Reposo */
        mensaje_entrante.ID1=data;
        USART1_Rx_Status= CAN_ID1_BYTE_RECIBIDO;
        break;

case CAN_ID1_BYTE_RECIBIDO:
        mensaje_entrante.ID0=data;
        USART1_Rx_Status = CAN_ID0_BYTE_RECIBIDO;
        break;

case CAN_ID0_BYTE_RECIBIDO:
        /*almacenar dato1*/
        mensaje_entrante.BYTE1=data;
        USART1_Rx_Status = IDLE;
        PORTD &= ~(1 << DATA_ACTIVITY);
        /*introducción atomica del mensaje en buffer
        de transmisión CAN o para proceso en nodo local*/
        tmp1=mensaje_entrante.ID1 & LOCAL_CAN_MASK;
        if ((tmp2 == LOCAL_CAN_ID))

        {
        /*el mensaje va a buffer de este ATmega*/
                /* Calculate buffer index */
                tmp_local_head = ( CAN_Rx_Local_Head + 1 ) &
                CAN_RX_LOCAL_BUFFER_MASK;

                if ( tmphead == CAN_Rx_Local_Tail )
                {
                printf("\n\rCAN_Rx_Local buffer overflow\n\r");
                /* ERROR! Receive buffer overflow */
                }
                else{
                CAN_Rx_Local_Buf[tmp_local_head].id1 = mensaje_entrante.ID1;
                CAN_Rx_Local_Buf[tmp_local_head].id0 = mensaje_entrante.ID0;
                CAN_Rx_Local_Buf[tmp_local_head].byte1 =
mensaje_entrante.BYTE1;
                CAN_Rx_Local_Head = tmp_local_head;
                }
        }
        else
        {
        /*el mensaje va a bus CAN*/
        /* Calculo buffer index */
                tmphead = ( CAN_TxHead + 1 ) & CAN_TX_BUFFER_MASK;
                if ( tmphead == CAN_TxTail )
                {
                printf(" CAN_TxBuf buffer overflow\n\r");
                /* ERROR! Receive buffer overflow */
                /*Se desecha el mensaje*/
```

```
                        }
                        else
                        {
                         /* Read the received data */
                    CAN_TxBuf[tmphead].id1 = mensaje_entrante.ID1;
                    CAN_TxBuf[tmphead].id0 = mensaje_entrante.ID0;
                    CAN_TxBuf[tmphead].byte1 = mensaje_entrante.BYTE1;
                    CAN_TxHead = tmphead;      /* Store new index */
                    FORCE_TX=1;
                    int0_isr();
                        }
                }
                PORTD |= (1 << DATA_ACTIVITY);
                break;
                SEI();
        }
        }
/* Maquina de estado para extraer mensajes
CAN recibidos del segmento CAN y transmitirlos
a traves del módulo Free2Move*/
#pragma interrupt_handler USART1_TX_interrupt:iv_USART1_UDRE
void USART1_TX_interrupt( void )
{
        static unsigned char tmptail, data;
        static unsigned char CAN_TX_Length=0;
        CLI();
        PORTD &= ~(1 << DATA_ACTIVITY);
        switch (USART1_Tx_Status)
        {
        case IDLE:

                 /*Estado de reposo*/
                 /*Coge un elemento de CAN_RX_Buffer
                  y lo envia en 2/3 llamadas
                 a esta interrupción*/
                /* Calculo buffer index */
                /* Se chequea si se hay datos por transmitir*/
                if ( CAN_RxHead != CAN_RxTail )
                {
                /*Calculo de indice*/
                tmptail = ( CAN_RxTail + 1 ) & CAN_RX_BUFFER_MASK;
                /*Almacenamiento del caracter*/
                UDR1 = CAN_RxBuf[tmptail].id1;
                USART1_Transmited_Bytes++;
                USART1_Tx_Status=CAN_ID1_BYTE_TRANSMITIDO;
                }
                else
                {
                PORTD |= (1 << DATA_ACTIVITY);
                UCSR1B &= ~(1<<UDRIE1);
                USART1_Tx_Status=IDLE;
                }
                break;

        case CAN_ID1_BYTE_TRANSMITIDO:

                data=CAN_RxBuf[tmptail].id0;
                CAN_TX_Length = data | 0b00001111;
                UDR1 = data;
                USART1_Transmited_Bytes++;
```

```
                if(CAN_TX_Length =0)
                        USART1_Tx_Status = IDLE;
                else
                        USART1_Tx_Status = CAN_ID0_BYTE_TRANSMITIDO;
                break;

        case CAN_ID0_BYTE_TRANSMITIDO:

                 UDR1=CAN_RxBuf[tmptail].byte1;
                 USART1_Transmited_Bytes++;
                 USART1_Tx_Status = IDLE;
                 CAN_RxTail = tmptail;
                 PORTD |= (1 << DATA_ACTIVITY);
                break;

        }
        SEI();

}

/*Recepción por espera activa*/
unsigned char USART1_Receive( void )
{
        /* Espera de dato */
        while ( !(UCSR1A & (1<<RXC1)) )
                ;
        /* Devolución del dato recibido */
        return UDR1;
}

/*Transmisión por espera activa*/
void USART1_Transmit( unsigned char data )
{
        /* ¿Bufffer TX libre? */
        while ( !(UCSR1A & (1<<UDRE1)) )
                ;
        /* Inicio de transmisión */
        UDR1 = data;
}

/*Inicio del interfaz con el módulo Free2Move*/
void Free2Move_Init (int baudrate)
{

         char temp0, temp1;

         UCSR1B = 0x00; //se deshabilita durante configuración
         UCSR1A = 0x00;
         UCSR1C = 0x00;

        /* Configuracion 115200 */
        UBRR1H = (unsigned char) (baudrate>>8);
        UBRR1L = (unsigned char) baudrate;

        /*Inicio de maquina de estado*/
        USART1_Rx_Status=IDLE;
        USART1_Tx_Status=IDLE;

        /* 8 bits datos 1 stop*/
        UCSR1C = (3<<UCSZ10);
```

```
        PORTC &= ~(1 << CONNECTION);
        PORTD |=  (1 << DATA_ACTIVITY);

        /*Habilitación TX, RX e interrupción
        por caracter recibido*/
        UCSR1B = ( ( 1 << RXCIE1 )
                        | ( 1 << TXEN1 )
                        | ( 1 << RXEN1 )  );

}
```

FICHERO FREE2MOVE.h

Este fichero contiene la declaración de variables que se quieren acceder desde otros ficheros y de las funciones definidas en FREE2MOVE.c por lo que no se detalla explícitamente.

FICHERO TIMERS.c y TIMERS.h

Este fichero configura el temporizador 1 y el 3. Estos no son esenciales para la aplicación en cuestión pero si útiles para cuestiones de depuración.TIMERS.h declara las variables de temporización para su uso desde otros ficheros.

```
/*Fichero Timers.c*/
/*Configuración de timers y actuaciones
asociadas a las interrupciones*/

#include "..\COMUNES\SELECT.h"

#if defined (AIRE)
#include "..\AIRE\AIRE_header.h"
#endif
#if defined (TIERRA)
#include "..\TIERRA\TIERRA_header.h"
#endif



/*Variables globales de temporización*/

int ms, s;
char decimas_seg_T3, decimas_seg_T3_enable;

//El TIMER1 se utiliza para generar una base de
//tiempos
//TIMER1 initialisation - prescale:1
// WGM: 0) Normal, TOP=0xFFFF
// desired value: 1mSec
// actual value:  1,000mSec (0,0%)
void timer1_init(void)
{
 TCCR1B = 0x00; //stop
 TCNT1H = 0xD4; //setup
 TCNT1L = 0xCD;
 OCR1AH = 0x2B;
```

```
 OCR1AL = 0x33;
 OCR1BH = 0x2B;
 OCR1BL = 0x33;
 OCR1CH = 0x2B;
 OCR1CL = 0x33;
 ICR1H  = 0x2B;
 ICR1L  = 0x33;
 TCCR1A = 0x00;
 TCCR1B = 0x01; //start Timer
 s=0;
 ms=0;

}

#pragma interrupt_handler timer1_ovf_isr:iv_TIMER1_OVF
void timer1_ovf_isr(void)
{
 CLI();
 //TIMER1 has overflowed
 TCNT1H = 0xD4; //reload counter high value
 TCNT1L = 0xCD; //reload counter low value
 if (ms == 999)
 {
          ms = 0;
          s++;
 }
 else
 {
           ms++;
 }


 SEI();
}

//TIMER3 initialisation - prescale:1024
// WGM: 0) Normal, TOP=0xFFFF
// desired value: 100mSec
// actual value: 100,000mSec (0,0%)
void timer3_init(void)
{
 TCCR3B = 0x00; //stop
 TCNT3H = 0xFB; //setup
 TCNT3L = 0xC8;
 OCR3AH = 0x04;
 OCR3AL = 0x38;
 OCR3BH = 0x04;
 OCR3BL = 0x38;
 OCR3CH = 0x04;
 OCR3CL = 0x38;
 ICR3H  = 0x04;
 ICR3L  = 0x38;
 TCCR3A = 0x00;
 TCCR3B = 0x05; //start Timer
 decimas_seg_T3=0;
}

#pragma interrupt_handler timer3_ovf_isr:30
void timer3_ovf_isr(void)
{
```

```
CLI();

 //TIMER3 has overflowed
 TCNT3H = 0xFB; //reload counter high value
 TCNT3L = 0xC8; //reload counter low value
 if (decimas_seg_T3_enable==1)
 {
        if (decimas_seg_T3 == 100)
        {
                decimas_seg_T3=0;
                decimas_seg_T3_enable=0;
        }
        else
        {
                decimas_seg_T3++;
        }
 }
SEI();
}
```

FICHERO BOARD_IO.c

```
#include "..\COMUNES\SELECT.h"

#if defined (AIRE)
#include "..\AIRE\AIRE_header.h"
#endif
#if defined (TIERRA)
#include "..\TIERRA\TIERRA_header.h"
#endif

void watchdog_init(void)
{
 /*Se recarga el Watchdog para que se active
 durante la iniciación*/
 WDR();
 /*Configuración y habilitación del WATCHDOG*/
 WDTCR = 0x0F;
}

/*Retardo aproximado de 1 segundo*/
void Delay()
        {
        unsigned char a, b, c;
        for (a = 1; a; a++)
                for (b = 1; b; b++)
                        for (c = 1; c<10; c++)
                        ;
        }

/*Retardo aproximado de 1 milisegundo*/
void Delay_ms()
        {
        unsigned char a, b, c;
        for (a = 1; a; a++)
                for (b = 1; b<40; b++)
                        ;
        }
```

```
void Swap_PORTB_LED(char i)
        {
        unsigned char State,LedPin;
        LedPin=i;
        State=CHECKPIN(PORTB,LedPin);
        PORTB = SETPIN(PORTB,LedPin);
        PORTB &= ~State;
        }
void Swap_PORTD_LED(char i)
        {
        unsigned char State,LedPin;
        LedPin=i;
        State=CHECKPIN(PORTD,LedPin);
        PORTD = SETPIN(PORTD,LedPin);
        PORTD &= ~State;
        }
char Pressed_SW(char i)
        {
        static unsigned char LastState=0;
        unsigned char Pressed=0,SwitchPin=i;

        if (LastState && !CHECKPIN(PINB,SwitchPin))
          Pressed=1;
        LastState = CHECKPIN(PINB,SwitchPin);
        return Pressed;
        }
```

FICHERO MACROS.h

```
#ifndef BIT
#define BIT(x)  (1 << (x))
#endif

// Activar y desactivar bits en registros I/O utilizando macros //
#define SETPIN(ADDRESS,PIN) (ADDRESS |= BIT(PIN))
#define CLEARPIN(ADDRESS,PIN) (ADDRESS &= ~BIT(PIN))
// Obtener el valor del bit de un registro I/O utilizando una macro //
#define CHECKPIN(ADDRESS,PIN) (ADDRESS & BIT(PIN))

#ifndef C_task
#define C_task
#endif

#define _asm    asm        /* old style */

#define WDR()   asm("wdr")
#define SEI()   asm("sei")
#define CLI()   asm("cli")
#define NOP()   asm("nop")
#define _WDR()  asm("wdr")
#define _SEI()  asm("sei")
#define _CLI()  asm("cli")
#define _NOP()  asm("nop")
#define RESET() asm("jmp 0")
```

FICHERO iom128v.h

Este fichero contiene constantes útiles par trabajar con el microcontrolador dando nombre a registros que realmente se direccionan por numéricamente.

```
#ifndef __iom128_h
#define __iom128_h

/* ATmega128 header file for
 * ImageCraft ICCAVR compiler
 */

/* i/o register addresses
 * >= 0x60 are memory mapped only
 */

/* not strictly sorted by address no more
 */

/* interrupt vector number definitions added
 */

/* last changed 2001/05/01
 */

/* Port G */
#define PING    (*(volatile unsigned char *)0x63) /* m/m */
#define DDRG  (*(volatile unsigned char *)0x64) /* m/m */
#define PORTG           (*(volatile unsigned char *)0x65) /* m/m */

/* Port F */
#define PINF    (*(volatile unsigned char *)0x20)
#define DDRF  (*(volatile unsigned char *)0x61) /* m/m */
#define PORTF (*(volatile unsigned char *)0x62) /* m/m */

/* Port E */
#define PINE    (*(volatile unsigned char *)0x21)
#define DDRE  (*(volatile unsigned char *)0x22)
#define PORTE(*(volatile unsigned char *)0x23)

/* Port D */
#define PIND    (*(volatile unsigned char *)0x30)
#define DDRD  (*(volatile unsigned char *)0x31)
#define PORTD(*(volatile unsigned char *)0x32)

/* Port C */
#define PINC    (*(volatile unsigned char *)0x33)
#define DDRC  (*(volatile unsigned char *)0x34)
#define PORTC(*(volatile unsigned char *)0x35)

/* Port B */
#define PINB    (*(volatile unsigned char *)0x36)
#define DDRB  (*(volatile unsigned char *)0x37)
#define PORTB(*(volatile unsigned char *)0x38)

/* Port A */
#define PINA    (*(volatile unsigned char *)0x39)
#define DDRA  (*(volatile unsigned char *)0x3A)
```

```
#define PORTA(*(volatile unsigned char *)0x3B)

/* ADC */
#define ADC     (*(volatile unsigned int *)0x24)
#define ADCL   (*(volatile unsigned char *)0x24)
#define ADCH   (*(volatile unsigned char *)0x25)
#define ADCSRA          (*(volatile unsigned char *)0x26)
#define  ADEN     7
#define  ADSC     6
#define  ADFR     5
#define  ADRF     5
#define  ADATE    5
#define  ADIF     4
#define  ADIE     3
#define  ADPS2    2
#define  ADPS1    1
#define  ADPS0    0
#define ADMUX           (*(volatile unsigned char *)0x27)
#define  REFS1    7
#define  REFS0    6
#define  ADLAR    5
#define  MUX4     4
#define  MUX3     3
#define  MUX2     2
#define  MUX1     1
#define  MUX0     0

/* Analog Comparator Control and Status Register */
#define ACSR   (*(volatile unsigned char *)0x28)
#define  ACD      7
#define  ACBG     6
#define  ACO      5
#define  ACI      4
#define  ACIE     3
#define  ACIC     2
#define  ACIS1    1
#define  ACIS0    0

/* USART0 */
#define UBRR0H          (*(volatile unsigned char *)0x90) /* m/m */
#define UBRR0L          (*(volatile unsigned char *)0x29)
#define UCSR0C          (*(volatile unsigned char *)0x95) /* m/m */
#define  UMSEL0   6
#define  UPM01    5
#define  UPM00    4
#define  USBS0    3
#define  UCSZ01   2
#define  UCSZ00   1
#define  UCPOL0   0
#define UCSR0B          (*(volatile unsigned char *)0x2A)
#define  RXCIE0   7
#define  TXCIE0   6
#define  UDRIE0   5
#define  RXEN0    4
#define  TXEN0    3
#define  UCSZ02   2
#define  RXB80    1
#define  TXB80    0
#define UCSR0A          (*(volatile unsigned char *)0x2B)
#define  RXC0     7
```

```
#define  TXC0     6
#define  UDRE0    5
#define  FE0      4
#define  DOR0     3
#define  UPE0     2
#define  U2X0     1
#define  MPCM0    0
#define UDR0   (*(volatile unsigned char *)0x2C)

/* USART1 */
#define UBRR1H          (*(volatile unsigned char *)0x98) /* m/m */
#define UBRR1L          (*(volatile unsigned char *)0x99) /* m/m */
#define UCSR1C          (*(volatile unsigned char *)0x9D) /* m/m */
#define  UMSEL1   6
#define  UPM11    5
#define  UPM10    4
#define  USBS1    3
#define  UCSZ11   2
#define  UCSZ10   1
#define  UCPOL1   0
#define UCSR1B          (*(volatile unsigned char *)0x9A) /* m/m */
#define  RXCIE1   7
#define  TXCIE1   6
#define  UDRIE1   5
#define  RXEN1    4
#define  TXEN1    3
#define  UCSZ12   2
#define  RXB81    1
#define  TXB81    0
#define UCSR1A          (*(volatile unsigned char *)0x9B) /* m/m */
#define  RXC1     7
#define  TXC1     6
#define  UDRE1    5
#define  FE1      4
#define  DOR1     3
#define  UPE1     2
#define  U2X1     1
#define  MPCM1    0
#define UDR1   (*(volatile unsigned char *)0x9C) /* m/m */

/* 2-wire SI */
#define TWBR  (*(volatile unsigned char *)0x70) /* m/m */
#define TWSR  (*(volatile unsigned char *)0x71) /* m/m */
#define  TWPS1    1
#define  TWPS0    0
#define TWAR  (*(volatile unsigned char *)0x72) /* m/m */
#define  TWGCE    0
#define TWDR  (*(volatile unsigned char *)0x73) /* m/m */
#define TWCR  (*(volatile unsigned char *)0x74) /* m/m */
#define  TWINT    7
#define  TWEA     6
#define  TWSTA    5
#define  TWSTO    4
#define  TWWC     3
#define  TWEN     2
#define  TWIE     0

/* SPI */
#define SPCR   (*(volatile unsigned char *)0x2D)
#define  SPIE     7
```

```
#define  SPE      6
#define  DORD     5
#define  MSTR     4
#define  CPOL     3
#define  CPHA     2
#define  SPR1     1
#define  SPR0     0
#define SPSR   (*(volatile unsigned char *)0x2E)
#define  SPIF     7
#define  WCOL     6
#define  SPI2X    0
#define SPDR   (*(volatile unsigned char *)0x2F)

/* EEPROM */
#define EECR   (*(volatile unsigned char *)0x3C)
#define  EERIE    3
#define  EEMWE    2
#define  EEWE     1
#define  EERE     0
#define EEDR   (*(volatile unsigned char *)0x3D)
#define EEAR   (*(volatile unsigned int *)0x3E)
#define EEARL (*(volatile unsigned char *)0x3E)
#define EEARH(*(volatile unsigned char *)0x3F)

/* Special Function IO Register */
#define SFIOR  (*(volatile unsigned char *)0x40)
#define  TSM      7
#define  ADHSM    4
#define  ACME     3
#define  PUD      2
#define  PSR0     1
#define  PSR321   0

/* Watchdog Timer Control Register */
#define WDTCR          (*(volatile unsigned char *)0x41)
#define  WDCE     4
#define  WDE      3
#define  WDP2     2
#define  WDP1     1
#define  WDP0     0

/* OCDR */
#define OCDR  (*(volatile unsigned char *)0x42)
#define  IDRD     7

/* Timer/Counter3 */
#define ICR3    (*(volatile unsigned int *)0x80) /* m/m */
#define ICR3L  (*(volatile unsigned char *)0x80) /* m/m */
#define ICR3H  (*(volatile unsigned char *)0x81) /* m/m */
#define OCR3C(*(volatile unsigned int *)0x82) /* m/m */
#define OCR3CL         (*(volatile unsigned char *)0x82) /* m/m */
#define OCR3CH         (*(volatile unsigned char *)0x83) /* m/m */
#define OCR3B(*(volatile unsigned int *)0x84) /* m/m */
#define OCR3BL         (*(volatile unsigned char *)0x84) /* m/m */
#define OCR3BH         (*(volatile unsigned char *)0x85) /* m/m */
#define OCR3A(*(volatile unsigned int *)0x86) /* m/m */
#define OCR3AL         (*(volatile unsigned char *)0x86) /* m/m */
#define OCR3AH         (*(volatile unsigned char *)0x87) /* m/m */
#define TCNT3 (*(volatile unsigned int *)0x88) /* m/m */
#define TCNT3L         (*(volatile unsigned char *)0x88) /* m/m */
```

```
#define TCNT3H          (*(volatile unsigned char *)0x89) /* m/m */
#define TCCR3C          (*(volatile unsigned char *)0x8C) /* m/m */
#define  FOC3A    7
#define  FOC3B    6
#define  FOC3C    5
#define TCCR3B          (*(volatile unsigned char *)0x8A) /* m/m */
#define  ICNC3    7
#define  ICES3    6
#define  WGM33    4
#define  WGM32    3
#define  CS32     2
#define  CS31     1
#define  CS30     0
#define TCCR3A          (*(volatile unsigned char *)0x8B) /* m/m */
#define  COM3A1   7
#define  COM3A0   6
#define  COM3B1   5
#define  COM3B0   4
#define  COM3C1   3
#define  COM3C0   2
#define  WGM31    1
#define  WGM30    0

/* Timer/Counter2 */
#define OCR2   (*(volatile unsigned char *)0x43)
#define TCNT2 (*(volatile unsigned char *)0x44)
#define TCCR2 (*(volatile unsigned char *)0x45)
#define  FOC2     7
#define  WGM20    6
#define  COM21    5
#define  COM20    4
#define  WGM21    3
#define  CS22     2
#define  CS21     1
#define  CS20     0

/* Timer/Counter1 */
#define ICR1    (*(volatile unsigned int *)0x46)
#define ICR1L   (*(volatile unsigned char *)0x46)
#define ICR1H   (*(volatile unsigned char *)0x47)
#define OCR1C(*(volatile unsigned int *)0x78) /* m/m */
#define OCR1CL          (*(volatile unsigned char *)0x78) /* m/m */
#define OCR1CH          (*(volatile unsigned char *)0x79) /* m/m */
#define OCR1B(*(volatile unsigned int *)0x48)
#define OCR1BL          (*(volatile unsigned char *)0x48)
#define OCR1BH          (*(volatile unsigned char *)0x49)
#define OCR1A(*(volatile unsigned int *)0x4A)
#define OCR1AL          (*(volatile unsigned char *)0x4A)
#define OCR1AH          (*(volatile unsigned char *)0x4B)
#define TCNT1 (*(volatile unsigned int *)0x4C)
#define TCNT1L          (*(volatile unsigned char *)0x4C)
#define TCNT1H          (*(volatile unsigned char *)0x4D)
#define TCCR1C          (*(volatile unsigned char *)0x7A) /* m/m */
#define  FOC1A    7
#define  FOC1B    6
#define  FOC1C    5
#define TCCR1B          (*(volatile unsigned char *)0x4E)
#define  ICNC1    7
#define  ICES1    6
#define  WGM13    4
```

```
#define  WGM12    3
#define  CS12     2
#define  CS11     1
#define  CS10     0
#define TCCR1A          (*(volatile unsigned char *)0x4F)
#define  COM1A1   7
#define  COM1A0   6
#define  COM1B1   5
#define  COM1B0   4
#define  COM1C1   3
#define  COM1C0   2
#define  WGM11    1
#define  WGM10    0

/* Timer/Counter 0 */
#define ASSR   (*(volatile unsigned char *)0x50)
#define  AS0      3
#define  TCN0UB   2
#define  OCR0UB   1
#define  TCR0UB   0
#define OCR0   (*(volatile unsigned char *)0x51)
#define TCNT0 (*(volatile unsigned char *)0x52)
#define TCCR0 (*(volatile unsigned char *)0x53)
#define  FOC0     7
#define  WGM00    6
#define  COM01    5
#define  COM00    4
#define  WGM01    3
#define  CS02     2
#define  CS01     1
#define  CS00     0

/* Oscillator Calibration Register */
#define OSCCAL          (*(volatile unsigned char *)0x6F) /* m/m */

/* MCU */
#define MCUSR           (*(volatile unsigned char *)0x54)
#define MCUCSR          (*(volatile unsigned char *)0x54)
#define  JTD      7
#define  JTRF     4
#define  WDRF     3
#define  BORF     2
#define  EXTRF    1
#define  PORF     0
#define MCUCR           (*(volatile unsigned char *)0x55)
#define  SRE      7
#define  SRW10    6
#define  SE       5
#define  SM1      4
#define  SM0      3
#define  SM2      2
#define  IVSEL    1
#define  IVCE     0

/* SPM Control and Status Register */
#define SPMCSR          (*(volatile unsigned char *)0x68) /* m/m */
#define  SPMIE    7
#define  RWWSB    6
#define  RWWSRE   4
#define  BLBSET   3
```

```
#define  PGWRT    2
#define  PGERS    1
#define  SPMEN    0

/* Timer/Counter Interrupts */
#define TIFR     (*(volatile unsigned char *)0x56)
#define  OCF2     7
#define  TOV2     6
#define  ICF1     5
#define  OCF1A    4
#define  OCF1B    3
#define  TOV1     2
#define  OCF0     1
#define  TOV0     0
#define TIMSK  (*(volatile unsigned char *)0x57)
#define  OCIE2    7
#define  TOIE2    6
#define  TICIE1   5
#define  OCIE1A   4
#define  OCIE1B   3
#define  TOIE1    2
#define  OCIE0    1
#define  TOIE0    0
#define ETIFR  (*(volatile unsigned char *)0x7C) /* m/m */
#define  ICF3     5
#define  OCF3A    4
#define  OCF3B    3
#define  TOV3     2
#define  OCF3C    1
#define  OCF1C    0
#define ETIMSK          (*(volatile unsigned char *)0x7D) /* m/m */
#define  TICIE3   5
#define  OCIE3A   4
#define  OCIE3B   3
#define  TOIE3    2
#define  OCIE3C   1
#define  OCIE1C   0

/* Èxternal Interrupts */
#define EIFR     (*(volatile unsigned char *)0x58)
#define  INTF7    7
#define  INTF6    6
#define  INTF5    5
#define  INTF4    4
#define  INTF3    3
#define  INTF2    2
#define  INTF1    1
#define  INTF0    0
#define EIMSK  (*(volatile unsigned char *)0x59)
#define  INT7     7
#define  INT6     6
#define  INT5     5
#define  INT4     4
#define  INT3     3
#define  INT2     2
#define  INT1     1
#define  INT0     0
#define EICRB  (*(volatile unsigned char *)0x5A)
#define  ISC71    7
#define  ISC70    6
```

```
#define  ISC61    5
#define  ISC60    4
#define  ISC51    3
#define  ISC50    2
#define  ISC41    1
#define  ISC40    0
#define EICRA  (*(volatile unsigned char *)0x6A) /* m/m */
#define  ISC31    7
#define  ISC30    6
#define  ISC21    5
#define  ISC20    4
#define  ISC11    3
#define  ISC10    2
#define  ISC01    1
#define  ISC00    0

/* RAM page Z-pointer */
#define RAMPZ(*(volatile unsigned char *)0x5B)

/* XDIV Divide control register */
#define XDIV    (*(volatile unsigned char *)0x5C)
#define  XDIVEN   7

/* Stack Pointer */
#define SP       (*(volatile unsigned int *)0x5D)
#define SPL      (*(volatile unsigned char *)0x5D)
#define SPH      (*(volatile unsigned char *)0x5E)

/* Status REGister */
#define SREG   (*(volatile unsigned char *)0x5F)

/* eXternal Memory Control Register */
#define XMCRB           (*(volatile unsigned char *)0x6C) /* m/m */
#define  XMBK     7
#define  XMM2     2
#define  XMM1     1
#define  XMM0     0
#define XMCRA           (*(volatile unsigned char *)0x6D) /* m/m */
#define  SRL2     6
#define  SRL1     5
#define  SRL0     4
#define  SRW01    3
#define  SRW00    2
#define  SRW11    1


/* Port A bits */
#define  PORTA7   7
#define  PORTA6   6
#define  PORTA5   5
#define  PORTA4   4
#define  PORTA3   3
#define  PORTA2   2
#define  PORTA1   1
#define  PORTA0   0
#define  PA7      7
#define  PA6      6
#define  PA5      5
#define  PA4      4
#define  PA3      3
```

```
#define  PA2     2
#define  PA1     1
#define  PA0     0

#define  DDA7    7
#define  DDA6    6
#define  DDA5    5
#define  DDA4    4
#define  DDA3    3
#define  DDA2    2
#define  DDA1    1
#define  DDA0    0

#define  PINA7   7
#define  PINA6   6
#define  PINA5   5
#define  PINA4   4
#define  PINA3   3
#define  PINA2   2
#define  PINA1   1
#define  PINA0   0

/* Port B bits */
#define  PORTB7  7
#define  PORTB6  6
#define  PORTB5  5
#define  PORTB4  4
#define  PORTB3  3
#define  PORTB2  2
#define  PORTB1  1
#define  PORTB0  0
#define  PB7     7
#define  PB6     6
#define  PB5     5
#define  PB4     4
#define  PB3     3
#define  PB2     2
#define  PB1     1
#define  PB0     0

#define  DDB7    7
#define  DDB6    6
#define  DDB5    5
#define  DDB4    4
#define  DDB3    3
#define  DDB2    2
#define  DDB1    1
#define  DDB0    0

#define  PINB7   7
#define  PINB6   6
#define  PINB5   5
#define  PINB4   4
#define  PINB3   3
#define  PINB2   2
#define  PINB1   1
#define  PINB0   0

/* Port C bits */
#define  PORTC7  7
```

```
#define  PORTC6   6
#define  PORTC5   5
#define  PORTC4   4
#define  PORTC3   3
#define  PORTC2   2
#define  PORTC1   1
#define  PORTC0   0
#define  PC7      7
#define  PC6      6
#define  PC5      5
#define  PC4      4
#define  PC3      3
#define  PC2      2
#define  PC1      1
#define  PC0      0

#define  DDC7     7
#define  DDC6     6
#define  DDC5     5
#define  DDC4     4
#define  DDC3     3
#define  DDC2     2
#define  DDC1     1
#define  DDC0     0

#define  PINC7    7
#define  PINC6    6
#define  PINC5    5
#define  PINC4    4
#define  PINC3    3
#define  PINC2    2
#define  PINC1    1
#define  PINC0    0

/* Port D bits */
#define  PORTD7   7
#define  PORTD6   6
#define  PORTD5   5
#define  PORTD4   4
#define  PORTD3   3
#define  PORTD2   2
#define  PORTD1   1
#define  PORTD0   0
#define  PD7      7
#define  PD6      6
#define  PD5      5
#define  PD4      4
#define  PD3      3
#define  PD2      2
#define  PD1      1
#define  PD0      0

#define  DDD7     7
#define  DDD6     6
#define  DDD5     5
#define  DDD4     4
#define  DDD3     3
#define  DDD2     2
#define  DDD1     1
#define  DDD0     0
```

```
#define  PIND7    7
#define  PIND6    6
#define  PIND5    5
#define  PIND4    4
#define  PIND3    3
#define  PIND2    2
#define  PIND1    1
#define  PIND0    0

/* Port E bits */
#define  PORTE7   7
#define  PORTE6   6
#define  PORTE5   5
#define  PORTE4   4
#define  PORTE3   3
#define  PORTE2   2
#define  PORTE1   1
#define  PORTE0   0
#define  PE7      7
#define  PE6      6
#define  PE5      5
#define  PE4      4
#define  PE3      3
#define  PE2      2
#define  PE1      1
#define  PE0      0

#define  DDE7     7
#define  DDE6     6
#define  DDE5     5
#define  DDE4     4
#define  DDE3     3
#define  DDE2     2
#define  DDE1     1
#define  DDE0     0

#define  PINE7    7
#define  PINE6    6
#define  PINE5    5
#define  PINE4    4
#define  PINE3    3
#define  PINE2    2
#define  PINE1    1
#define  PINE0    0

/* Port F bits */
#define  PORTF7   7
#define  PORTF6   6
#define  PORTF5   5
#define  PORTF4   4
#define  PORTF3   3
#define  PORTF2   2
#define  PORTF1   1
#define  PORTF0   0
#define  PF7      7
#define  PF6      6
#define  PF5      5
#define  PF4      4
#define  PF3      3
```

```
#define  PF2     2
#define  PF1     1
#define  PF0     0

#define  DDF7    7
#define  DDF6    6
#define  DDF5    5
#define  DDF4    4
#define  DDF3    3
#define  DDF2    2
#define  DDF1    1
#define  DDF0    0

#define  PINF7   7
#define  PINF6   6
#define  PINF5   5
#define  PINF4   4
#define  PINF3   3
#define  PINF2   2
#define  PINF1   1
#define  PINF0   0

/* Port G bits */
#define  PORTG4  4
#define  PORTG3  3
#define  PORTG2  2
#define  PORTG1  1
#define  PORTG0  0
#define  PG4     4
#define  PG3     3
#define  PG2     2
#define  PG1     1
#define  PG0     0

#define  DDG4    4
#define  DDG3    3
#define  DDG2    2
#define  DDG1    1
#define  DDG0    0

#define  PING4   4
#define  PING3   3
#define  PING2   2
#define  PING1   1
#define  PING0   0


/* Lock and Fuse Bits with LPM/SPM instructions */

/* lock bits */
#define  BLB12   5
#define  BLB11   4
#define  BLB02   3
#define  BLB01   2
#define  LB2     1
#define  LB1     0

/* fuses low bits */
#define  BODLEVEL 7
#define  BODEN    6
```

```
#define  SUT1    5
#define  SUT0    4
#define  CKSEL3  3
#define  CKSEL2  2
#define  CKSEL1  1
#define  CKSEL0  0

/* fuses high bits */
#define  OCDEN   7
#define  JTAGEN  6
#define  SPIEN   5
#define  CKOPT   4
#define  EESAVE  3
#define  BOOTSZ1 2
#define  BOOTSZ0 1
#define  BOOTRST 0

/* extended fuses */
#define  M103C   1
#define  WDTON   0

/* Interrupt Vector Numbers */

#define iv_RESET       1
#define iv_INT0        2
#define iv_INT1        3
#define iv_INT2        4
#define iv_INT3        5
#define iv_INT4        6
#define iv_INT5        7
#define iv_INT6        8
#define iv_INT7        9
#define iv_TIMER2_COMP 10
#define iv_TIMER2_OVF  11
#define iv_TIMER1_CAPT 12
#define iv_TIMER1_COMPA 13
#define iv_TIMER1_COMPB 14
#define iv_TIMER1_OVF  15
#define iv_TIMER0_COMP 16
#define iv_TIMER0_OVF  17
#define iv_SPI_STC     18
#define iv_USART0_RX   19
#define iv_USART0_RXC  19
#define iv_USART0_DRE  20
#define iv_USART0_UDRE 20
#define iv_USART0_TX   21
#define iv_USART0_TXC  21
#define iv_ADC         22
#define iv_EE_RDY      23
#define iv_EE_READY    23
#define iv_ANA_COMP    24
#define iv_ANALOG_COMP 24
#define iv_TIMER1_COMPC 25
#define iv_TIMER3_CAPT 26
#define iv_TIMER3_COMPA 27
#define iv_TIMER3_COMPB 28
#define iv_TIMER3_COMPC 29
#define iv_TIMER3_OVF  30
#define iv_USART1_RX   31
#define iv_USART1_RXC  31
```

```
#define iv_USART1_DRE   32
#define iv_USART1_UDRE  32
#define iv_USART1_TX    33
#define iv_USART1_TXC   33
#define iv_TWI          34
#define iv_TWSI         34
#define iv_SPM_RDY      35
#define iv_SPM_READY    35

/* */

#endif
```

8. Resultados experimentales.

En este punto de la memoria se procede a obtener resultados del funcionamiento del sistema.

8.1 Sección Radiofrecuencia.

Primero se va a comprobar que el interfaz serie diseñado cumple los requisitos de velocidad, para ello se diseña un experimento en el que se conectan las tarjetas a través del interfaz radio, para comprobar el funcionamiento de los módulos Free2Move. Durante las pruebas se va a operar con los módulos Free2Move y también susituyendo el enlace de radio por un cable serie cruzado para comparar los módulos Free2Move que según el fabricante pueden remplazar perfectamente un cable RS-232.

Se va a manejar el interfaz serie como un puerto serie controlado por interrupciones recepción de carácter y buffer de transmisión libre de la misma manera que se trabaja con la USART0 del microcontrolador. Se trabaja con el archivo USART1_INT.c que es idéntico al archivo USART0_INT.c. Se va a usar el temporizador Timer1 para generar interrupciones periódicas y forzar la transmisión de caracteres. Se va a configurar para que cada milisegundo se envíe un número determinado de caracteres. En esta prueba se incrementará gradualmente el número de bytes transmitidos en cada interrupción para comprobar que caudal (bytes/milisegundo) que soporta el interfaz.

Antes de realizar la prueba se hace un cálculo del caudal previsto. Considerando que se ha configurado el interfaz a una tasa de datos de 115200 bps y que la porción de bits útiles para datos de este caudal son 8 de cada 10 resultan 92160 bps de datos, o lo que es lo mismo 92,16 bits/milisegundo. Por tanto se predice que el sistem soportará 92,16 (bits) / 8 (bits/byte) = 11,52 bytes/milisegundo. Considerando que cada mensaje CAN está formado 3 bytes resultan 3,84 mensajes CAN/milisegundo en cada sentido de la comunicación. Obviamente no caben las 6 señales previstas para el canal de subida o bajada pero según las especificaciones la tasa de muestreo de algunas señales se puede rebajar a 10 o 100 ms. Se pueden tener distintas configuraciones:

- Configuración asimétrica: considerando 100 milisegundos caben 384 mensajes en cada sentido (subida y bajada) que se pueden repartir:
  - 3 señales con tasa de muestreo 1ms ➔ 300 mensajes
  - 8 señales con tasa de muestreo 10 ms ➔ 80 mensajes
  - 4 señales con tasa de muestreo 100 ➔ 4 mensajes

- Configuraciones simétricas: 6 señales para cada sentido con una tasa de muestreo de 2 milisegundos o 12 señales para cada sentido con 4 milisegundos de muestreo

Una vez hecho este estudio teórico se procede a obtener los resultados de la prueba para comprobar cuantos bytes por milisegundo admite el sistema. Se configura el temporizador Timer1 para provocar un interrupción cada diezmilésima de segundo y obtener un fuente de reloj.

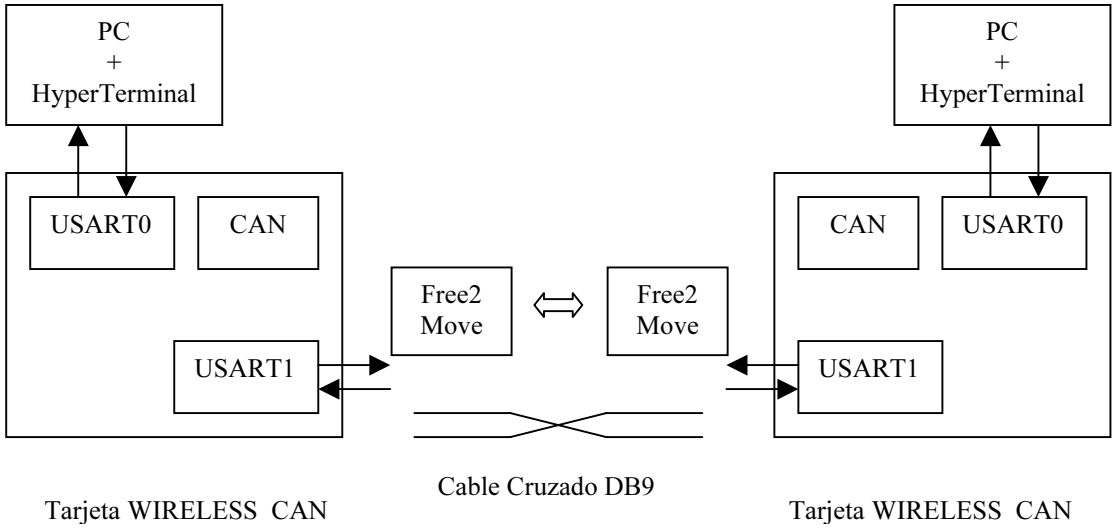


Figura 8.1 – 1: Montaje para medida de caudal en interfaz radio.

El código que se ha empleado se ajunta a continuación. En una tarjeta se ha programado el temporizador y una serie de comandos para evaluar el comportamiento de las tarjetas. En la otra tarjeta se ha programado la retransmisión de todo lo que se reciba.

TARJETA 1:

Esta tarjeta genera la base de tiempos que son décimas de milisegundo. Cada 10 decimas se actualiza la variable ms y se procede a la transmisión de caracteres si se habilita con la variable enable. En el bucle del módulo main se analizan y procesan los comandos enviados desde HyperTerminal, que solicitan diversas pruebas. En cada una de estas pruebas se reincian las variables que actúan como cronometro se procede a la prueba y se anota el tiempo para imprimirlo por pantalla.

CODIGO main.c

```
void main (void)
{

int a,b,received_bytes=0;
char temp0, temp1, permiso_tx, permiso_rx;
int medida_ms=9999, medida_s=9999, medida_decimas_ms=9999;
int  i, j , numero_bytes=100;


init_devices();
printf("\n\nProgram Restarted!!\n");
Delay();
USART1_Init(5);
printf("\n\rUSART1_Init(5)\n");
CLI();
can_init();
SEI();

for(;;)
{

        if (DataInReceiveBufferUSART0())
        {
          temp0=USART0_Receive();
          if (temp0 == 'X')
                {
                  permiso_tx=0;
                  ms=0;s=0;j=0;

                  USART1_Transmit('A');
                  temp1=USART1_Receive();
                  medida_decimas_ms=decimas_ms;
                  medida_ms=ms;
                  medida_s=s;
                  printf("\n\rInicio medida 1 bytes TX/RX");
                  printf("\n\r medida_decimas_ms= %d",medida_decimas_ms);
                  printf("\n\r medida_ms= %d",medida_ms);
                  printf("\n\r medida_s= %d\n\r",medida_s);

                }

          if (temp0 == 'T')
                {

                  ms=0;s=0;decimas_ms=0;
                  for(i=1;i<=999;i++)
                        {

                                //Transmisión por espera activa
                                while ( !(UCSR1A & (1<<UDRE1)) )
                                        ;
                                UDR1 = 'A';
                        }
                        medida_decimas_ms=decimas_ms;
                        medida_ms=ms;
                        medida_s=s;
```

```
                        printf("\n\rInicio medida 1000 bytes TX");
                        printf("\n\r medida_decimas_ms= %d",medida_decimas_ms);
                        printf("\n\r medida_ms= %d",medida_ms);
                        printf("\n\r medida_s= %d\n\r",medida_s);
                }

                if (temp0 == 'R')
                {
                  ms=0;s=0; decimas_ms=0;
                  j=0;received_bytes=0;
                        //Transmisión de 1000 caracteres
                        for (a=1;a<=1000;a++)
                        {
                        USART1_Transmit('A');
                        }
                        //USART1_Receive();

                        //Recepción de 1000 caracteres
                        do
                        {
                        if(DataInReceiveBufferUSART1())
                        {temp1=USART1_Receive();
                        putchar(temp1);
                        received_bytes++;
                        }
                        }while (received_bytes<1000);
                        medida_decimas_ms=decimas_ms;
                        medida_ms=ms;
                        medida_s=s;
                        printf("\n\rD");
                        printf("\n\rInicio medida Loop_Time");
                         printf("\n\r medida_decimas_ms= %d",medida_decimas_ms);
                        printf("\n\r medida_ms= %d",medida_ms);
                        printf("\n\r medida_s= %d\n\r",medida_s);
                        medida_ms=9999;
                        medida_s=9999;

                }
                // Habilitación/Deshabilitación de TX 11 caracteres/ms
                if (temp0 == 'E')
                {
                enable=1;

                }
                if (temp0 == 'D')
                {
                enable=0;

                }
        }
        //Impresión de caracteres recibidos.
        if (DataInReceiveBufferUSART1())
        {
                temp1=USART1_Receive();
                putchar(temp1);
        }

}

}
```

CODIGO DE LA INTERRUPCIÖN TIMER

```
#pragma interrupt_handler timer1_ovf_isr:iv_TIMER1_OVF
void timer1_ovf_isr(void)
{
 CLI();

// Configuración de Timer1 para interrupciones
// periodicas cada 0,1 ms.

TCNT1H = 0xFB;
TCNT1L = 0xAF;
 if (decimas_ms == 9)
 {
          decimas_ms = 0;
          if(ms==999)
          {
                    ms=0;
                    s++;
          }
          else
                    ms++;

          cont++;
          if (enable == 1)
          {
            for(numero_bytes_ms=1; numero_bytes_ms<=NUM_BYTES;  numero_bytes_ms++)
                      USART1_Transmit(cont);
          }
 }
 else
 {
           decimas_ms++;

 }

 SEI();
}
```

TARJETA 2:

El código de esta tarjeta es muy sencillo pues simplemente se retransmiten los caracteres recibidos.

```
void main (void)
{
          for(;;)
          {
                    if (DataInReceiveBufferUSART1())
                    {  temp1=USART1_Receive();
                     USART1_Transmit(temp1);
                      putchar(temp1);
                    }
          }
}
```

Para comprobar el máximo caudal que admite la tarjeta se tienen 2 pruebas. La primera es habilitar la transmisión cada milisegundo de varios caracteres e ir aumentando el numero de solicitudes de transmisión hasta que se produzca el desbordamiento. De esta manera se obtuvo que para 11 bytes introducidos por milisegundo (en cada milisegundo se envían 11 caracteres idénticos) el sistema trabaja perfectamente tanto con el cable serie como con los módulos Free2Move. Si se toman mas de 11 bytes el sistema se bloquea.

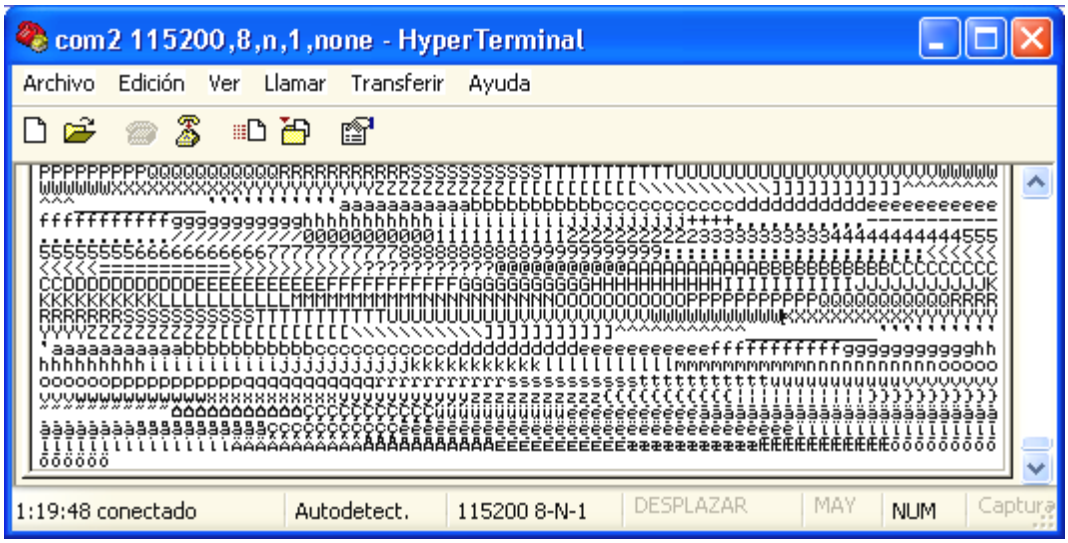


Figura 8.1-2:Tranmsión/Recepción continua de 11 bytes cada ms.

Otra forma de medir la velocidad es enviar una sucesión larga de caracteres con el comando asociado al carácter 'T', comprobando que llegan a la otra tarjeta, con esta prueba se comprueba que se transmiten 1000 bytes en unos 82 milisegundos lo que arroja el resultado de que casi se pueden transmitir 12 bytes por milisegundo. Este resultado es idéntico par el cable serie y para los módulos Free2Move.

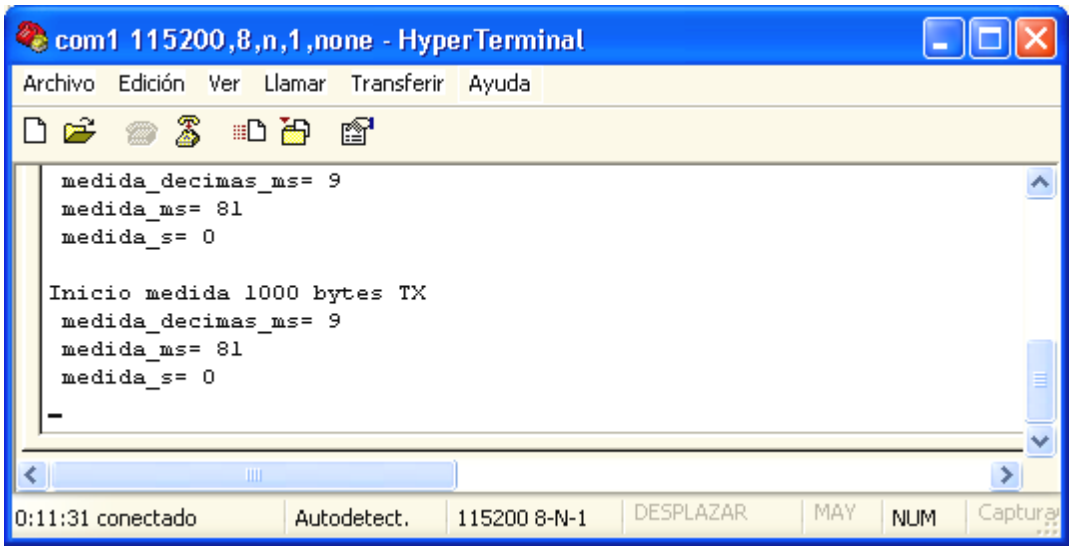


Figura 8.1 – 3:Transmisión simple de 1000 bytes.

Para medir el retardo se emplea el procedimiento asociado a comando 'X' que realiza una transmisión y espera una recepción. Para los módulos se obtiene un resultado de 50 a 60 milisegundos en el tiempo ida y vuelta con lo que se tiene un retardo de 25 a 30 ms pues para un byte el tiempo de transmisión es despreciable. Se ha comprobado también que el retardo varía ligeramente en función de la carga y del acoplo con el buffer del módulo. Este resultado contrasta con el resultado que se obtiene para el cable serie sustituyendo a los módulos de radio Free2Move que resulta 1 milisegundo. Por lo cual se concluye que estos módulos no son realmente una solución de reemplazo de RS-232 en cuanto al retardo introducido. Es lógico que los módulos Free2Move introduzcan cierto retraso pues se ha de pensar que para enviar una trama de radio estos módulos deben almacenar previamente un numero de bytes procedentes del interfaz RS-232 para lo que hacen es transmitir cada vez que tiene un cierto numero de bytes para llenar una trama o con los bytes que tenga cuando pase un determinado periodo.

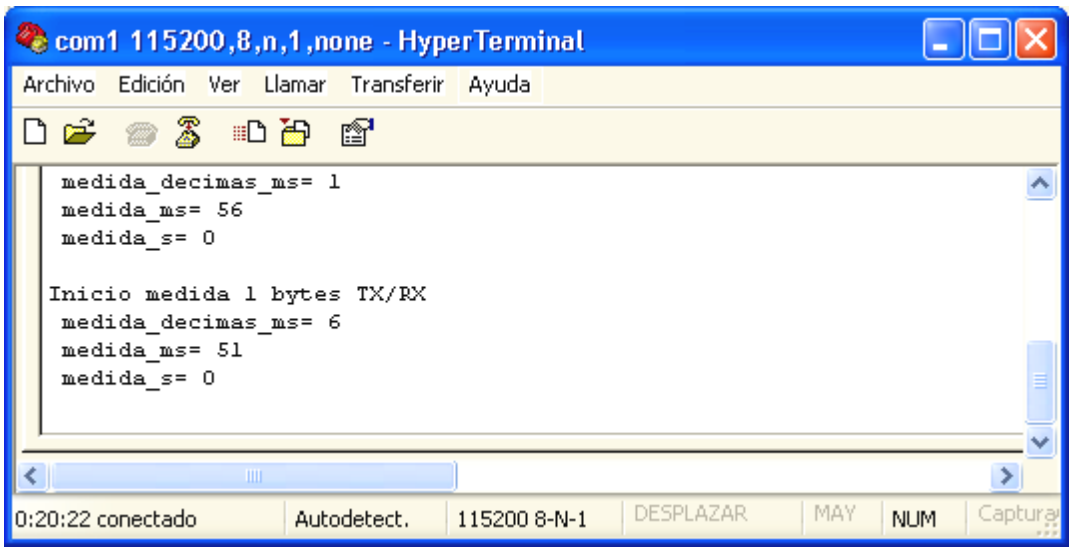


Figura 8.1 – 3:Medida del tiempo de ida y vuelta.

8.2 Sección CAN.

El sistema de bus de campo CAN se ha configurado para trabajar a una velocidad física de 1 Mbps. Los mensajes CAN tienen una longitud de 3 bytes de datos pero que al introducir en la trama CAN a nivel físico consta de 55 bits al incluirlos bits o campos de bits SOF, CRC, ACK, EOF, etc. Por lo tanto el sistema admite 18 mensajes CAN en cada milisegundo por lo que cumple la especificación de 12 señales muestreadas cada milisegundo.

9. Conclusiones y desarrollos futuros.

En el apartado de los resultados experimentales se ha demostrado que si bien no se cumplen las especificaciones más severas las prestaciones obtenidas no distan mucho de éstas.

En este proyecto se ha demostrado la viabilidad de este sistema para utilizar en buses de campo en los que por necesidades de emplazamiento de los elementos a comunicar sea recomendable dividir el bus de campo en secciones a comunicar por radiofrecuencia. El ámbito de aplicaciones es enorme pues en todo tipo de sistemas con partes móviles donde haya dificultad para acceder físicamente de manera cableada a estas partes, resulta muy cómodo ubicar un segmento de bus CAN con acceso inalámbrico.

Como posibles ampliaciones y desarrollos futuros se propone el seguimiento de las tecnologías empleadas (u otras nuevas) que con el paso del tiempo irán mejorando y aumentando en prestaciones para mejorar las prestaciones del sistema. La tecnología Bluetooth comienza a hacerse popular y el precio de los dispositivos va a disminuir considerablemente en los próximos años pues uno de los objetivos de este estándar es que los chips sean báratos.

Bibliografía

- Programming and customizing the AVR microcontroller.
  Dhananjay V. Gadre

- Embedded C Programming and the Atmel AVR
  Richard Barnet, Larry O'cull, Sara Cox

- Propagación por radio
  Jose Maria Hernando Rabanos.

- CNAF y notas de utilización del espectro radioeléctrico.
  Secretaría de Estado para las Telecomunicaciones y la Sociedad de la Información. (SETSI) Ministerio de Ciencia y Tecnología

- Atmel AVR Atmega128 DataSheet
- Atmel AVR Application Notes

- SJA1000 DataSheet
- SJA 1000 Application Note AN97076
- SJA 1000 Application Note AN97046: Determination of Bit Timing Parameters for the CAN Controller SJA1000.
- PCA82C250T Datasheet

- Maxim 202E-241E DataSheet
- Maxim 5-pin Reset Supervisor

- Free2Move User Manual

- BRM01 Datasheet and User Manual

- BIM Radio Module Data Sheet

- Aurel XTR Datasheet

- Aerocomm Wireless Radio Modules Datasheet

- B2400 Radio Modem Datasheet