UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Desarrollo de una Interfaz de Comunicación Radio-Bluetooth-CAN para automóviles.

Autor:

Juan Ramón Gómez Martín

Tutora:

María Isabel del Valle González

Título del TFG: DESARROLLO DE UNA INTERFAZ DE COMUNICACION RADIO-BLUETOOTH-CAN PARA AUTOMOVILES

Breve resumen: El TFG propuesto aborda el diseño de una interfaz para adaptar las comunicaciones de un automóvil fabricado hace más de una década a la conectividad disponible en los automóviles actuales. Para ello se desarrollará una solución a partir de la utilización de un emisor de banda FM de radio y con conexión bluetooth.

Se partirá de un microcontrolador Cortex, que desarrollará las funciones de centralita y se comunicará con distintos módulos y periféricos para añadir funcionalidades.

Se analizará también la posibilidad de añadir comunicaciones vía bus CAN para comunicar directamente con los distintos periféricos y sistemas del automóvil.

Palabras Clave:

Buses de comunicación, comunicación inalámbrica, I2C, Radio, Bluetooth, CAN bus, Conectividad, Microcontrolador, Microchip Studio, Atmel START, RTOS.

AGRADECIMIENTOS:

Mi principal agradecimiento es a mi madre, que a base de esfuerzo me ha dado la posibilidad de estudiar y terminar esta carrera.

A mi querido compadre Juan Fernando, por apoyarme y ofrecerme tu ayuda en todo.

A mi responsable Jorge Muelas, por enseñarme que no todo son conocimientos técnicos.

Y en general, a toda la gente que ha creído en mí y me ha apoyado en este viaje tan largo.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

A. JUSTIFICACIÓN DEL PROYECTO

Como se ha comentado en la introducción, este proyecto surge de una necesidad personal. Realizar viajes en automóviles de mediana edad puede suponer no tener las comodidades que se han ido desarrollando en los últimos años y de las que ya disponen los automóviles más modernos. Este proyecto es un medio para acercar esos avances y comodidades a los usuarios de los automóviles que no disponen de ellos.

Uno de estos avances es la conectividad teléfono-automóvil. Actualmente existen dispositivos comerciales para facilitar esta conectividad, pero no presentan una solución completa, como se explicará más adelante. Estos módulos se conectan al teléfono móvil utilizando comunicación inalámbrica por Bluetooth y emite en broadcast en una frecuencia de la radio FM previamente seleccionada. En este trabajo partimos de uno de estos dispositivos comerciales para presentar una solución más completa.

El problema que presenta este dispositivo es que, si viajamos una distancia considerable, muy probablemente encontraremos otros emisores radio que solaparán nuestra emisión y producirán un corte en la transmisión además de interferencias. La solución, a priori, partiría por buscar a mano una nueva frecuencia libre de emisiones. Esta solución no es viable porque implica modificar la configuración del dispositivo mientras se conduce. Al hacer esto, nuestra atención en la conducción se reduce y se crea un alto riesgo de accidente.

Para desarrollar la solución, vamos a utilizar un microcontrolador conectado a un módulo receptor de radio FM para medir la fuerza de emisión de las diferentes frecuencias que encontramos en el trayecto, para que al final de este, nos muestre la banda menos utilizada y óptima para nuestra comunicación. Esto requiere hacer un viaje "de prueba" para el trayecto seleccionado. De igual manera, es necesario cambiar la frecuencia tanto de nuestro emisor comercial como el receptor del coche de forma manual.

Por ello, se presenta la posibilidad de comunicar directamente con el automóvil para automatizar este proceso. Partiendo del mismo microcontrolador y de módulos radio similares, se desarrollará una placa electrónica con todos los dispositivos necesarios para realizar la automatización, entre ellos, la adicción del bus CAN para la comunicación directa con el sistema del automóvil. Además, se añadirán distintos periféricos que añadirán funciones extra como geolocalización, una interfaz por pantalla y memorias para extracción de datos.

B. OBJETIVOS DEL PROYECTO

El objetivo principal es crear un sistema que aúne todas las ventajas que tienen estos sistemas modernos. En una primera parte del proyecto comprobaremos que el sistema puede llevarse a cabo, a partir de la programación de distintos módulos por separado. En la segunda parte, se presentará una PCB con todos los componentes necesarios para unificar el sistema.

Este desarrollo, para cumplir con su cometido, debe cumplir con los siguientes objetivos:

- Funcionalidad: El dispositivo debe ser capaz de extraer la frecuencia óptima para el funcionamiento de la interfaz. Esta frecuencia debe estar libre durante todo el trayecto, o en todo caso, debe ser la menos utilizada.

- Memoria: Debe de ser capaz de mantener los distintos análisis que se hagan en el tiempo, para alcanzar un espectro lo más analizado posible. Es importante este punto porque puede darse el caso, si viajamos mucho, de que no encontremos una frecuencia totalmente limpia para todo el camino. En este caso, nos apoyaremos en la geolocalización.

- Geolocalización: A partir de esta, podemos trazar diferentes zonas y sintonizar en diferentes frecuencias dependiendo de la zona objetivo. En esta parte del proyecto sólo crearemos la base para su utilidad futura.

- Automatización y comodidad: Este proceso debe ser automático a partir de la comunicación por igual entre el módulo radio y el bus CAN del automóvil. De esta manera, no nos distraeremos mientras conducimos. Este apartado no será tratado en este proyecto ya que no disponemos del sistema unificado.

- Interfaz: El sistema necesita tener una interfaz sencilla de manejar, al ser un producto destinado a usuarios no técnicos. Por otro lado, tiene que mantener un número de opciones adecuadas al cometido del desarrollo.

- Concentración de dispositivos: Debemos concentrar los diferentes dispositivos en una sola placa, de tal manera que se presente una solución global. Este objetivo también va relacionado con la comodidad.

C. FASES DEL PROYECTO

Los pasos que se han seguido a la hora de desarrollar esta solución son los siguientes:

1. Programación base del microcontrolador: A partir del software de programación y de las diferentes herramientas que presenta este, se configurará el microcontrolador con todo lo necesario para empezar a trabajar con él. Al partir de una placa de desarrollo en la que se incluye este microcontrolador, se reducirá mucho el tiempo de configuración.

2. Comunicación con el emisor radio: Configuraremos uno de los buses I2C del microcontrolador para empezar a comunicar con el emisor radio. De igual manera, se establecerá un protocolo de comunicación con este y configuraremos un temporizador que cumpla con las especificaciones de este protocolo. A partir de aquí se empezarán a medir potencias y a trabajar con ellas.

3. Temporizador / Timer: Se configurará un temporizador de 10 ms para que haga de tiempo base para todos los periféricos. Se desarrollarán diferentes task o rutinas para poder trabajar con diferentes tiempos utilizando el timer base.

4. Memoria: Se utilizará una memoria incluida en la propia placa de desarrollo del microcontrolador para guardar los niveles de potencia máximos obtenidos por el módulo radio. Al ser una memoria de baja capacidad, tras las primeras pruebas decidiremos si seguir usando esta memoria o utilizar otra de mayor capacidad.

5. Pantalla OLED: Para crear una solución que no necesite de interfaz (dicho de otra manera, un ordenador conectado en depuración al micro), se utilizará una pantalla externa de la misma familia que la placa de desarrollo. De esta manera, también reduciremos también el tiempo de desarrollo. Se crearán diferentes opciones que puedan resultar útiles al usuario.

6. GPS: Para generar más funcionalidad en la placa, se añadirá un GPS que nos provea de la posición global de nuestro dispositivo. Se configurará la UART para comunicar con él y se creará un pequeño protocolo para extraer la posición y la hora.

7. RTC: Dato que el GPS nos proporciona la hora UTC (o tiempo universal coordinado), podremos guardar también la hora a la que se ha producido cada lectura.

A partir de aquí y después de comprobar el funcionamiento de estos, se presentará una placa de circuito impreso que aúne todos estos periféricos, añadiendo además las siguientes funcionalidades:

1. Bus CAN: Para comunicar con los periféricos del coche y permitir el cambio automático de frecuencia de radio. El protocolo depende del ISO que utilice esa familia de automóviles.

2. Emisor radio: La primera parte para eliminar el emisor radio externo comercial. De esta manera, podremos emitir nosotros mismos y cambiar de frecuencia, a la vez que en la radio.

3. Módulo Bluetooth: La segunda parte para eliminar el emisor radio externo. Lo necesitamos para generar el último paso en la conectividad de la placa y permitir la transmisión de datos con el teléfono móvil.

2. ESTADO DEL ARTE

A. MICROCONTROLADORES

El primer microprocesador creado fue el Intel 4004 en 1971. Sin embargo, el primer microcontrolador fue diseñado por Texas Instruments, por los ingenieros Gary Boone y Michael Cochran en 1971. Crearon el TMS1000, un microcontrolador de 4 bits que combina en un solo empaquetado las funciones de ROM y RAM, procesador y reloj (1). Su objetivo principal era su uso en sistemas embebidos.

Intel, en parte en respuesta a la creación del TMS1000, desarrolló su microcontrolador en un solo chip optimizado para aplicaciones de control, el Intel 8048. Entre sus numerosas aplicaciones, este chip llego a miles de millones de teclados del mundo. De hecho, el presidente de Intel en aquel momento, Luke J. Valenter, anunció que el microcontrolador fue uno de sus productos más exitosos.

A partir de aquí, surgieron multitud de variantes. Quizás la variante más digna de comentar fue la inclusión de la EPROM, o memorias reprogramables. Esto supuso una gran ventaja frente a los rivales que utilizaban ROM o su variante PROM, que no permitían reprogramaciones.

En 1993, la introducción de la memoria EEPROM permitió que los microcontroladores fueran fácilmente reprogramables, siendo el primero el PIC16C84 de Microchip. Esto aumento las posibilidades de los microcontroladores exponencialmente, al tener dos de las funciones más importantes del microcontrolador: El prototipo rápido y la programación en el sistema.

Un poco antes de esto, en 1987, se empezaron a desarrollar los primeros ARM (como el nuestro). ARM son las siglas de “Advanced RISC Machine”, llamados así por la arquitectura “RISC”, u “Ordenador con Conjunto Reducido de Instrucciones”. Esta arquitectura permite conjuntos de instrucciones de 32 y hasta 64 bits (2).

Debido a sus bajos costes, consumo mínimo de potencia y baja generación de calor los hace esenciales para el desarrollo de aplicaciones ligeras y portátiles. En 2005, aproximadamente el 98% de los dispositivos móviles utilizaba procesador ARM (uno al menos). Desde 2009, los ARM ocupan el 90% de todos los procesadores con arquitectura RISC de 32 bits. Con más de 180 mil millones de chips ARM producidos hasta 2021, ARM es la arquitectura de conjunto de instrucciones (ISA) más utilizada y producida.

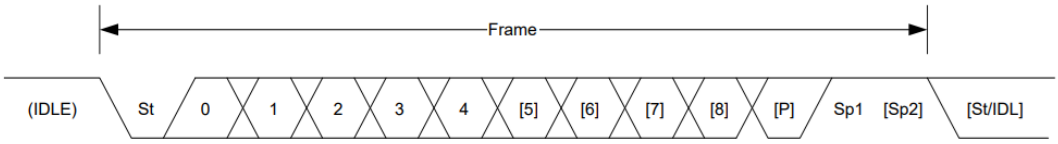
B. BUSES DE COMUNICACIÓN

I. UART

Uno de los primeros protocolos en aparecer es la UART, o Receptor-Transmisor Asíncrono Universal. Es el protocolo utilizado para controlar puertos y dispositivos serie. Permite la comunicación punto a punto o, dicho de otra manera, entre solamente dos dispositivos. Nosotros lo utilizaremos para comunicarnos con el GPS.

A continuación, se muestra el formato de cada frame de la transmisión:

Figure 34-2. Frame Formats



St Start bit. Signal is always low.

n, [n] Data bits. 0 to [5..9]

[P] Parity bit. Either odd or even.

Sp, [Sp] Stop bit. Signal is always high.

IDLE No frame is transferred on the communication line. Signal is always high in this state.

Figura 1. Formato de comunicación del bus UART

El formato de mensajes mostrado en la figura sigue las siguientes reglas:

1. El bit de inicio siempre está a bajo nivel.
2. De 5 a 9 bits de datos. Típicamente es 1 byte (8 bits).
3. Bit de paridad, para comprobación de errores.
4. Bits de parada. Típicamente se usa 1.
5. Nivel alto cuando no se produce comunicación.

Algunas de las características de este bus son:

- En su versión básica utiliza solamente dos líneas de datos, RX o recepción, y TX o transmisión. La conexión se hace desde el TX de un dispositivo al RX del otro dispositivo, y viceversa.

- Podemos tener los tres tipos de comunicación:
    - Simplex, cuando la comunicación sólo se produce en un sentido.
    - Full duplex, cuando ambos dispositivos envían y reciben datos simultáneamente.
    - Half duplex, cuando los dispositivos se alternan para enviar y recibir datos.
- Como nuestro bus utiliza comunicación asíncrona, se debe configurar previamente la velocidad de transmisión del bus. Se puede configurar uno de los dos dispositivos para que se adapte a la velocidad del otro dispositivo.
- Se puede utilizar para comunicación entre sistemas, permitiendo así la comunicación entre dispositivos que no se encuentra en la misma placa de circuito impreso o PCB.
- Existe una versión con línea de reloj o síncrona, llamada USART, o Receptor-Transmisor Síncrono-Asíncrono Universal. Actualmente este bus permite ambas configuraciones independientemente del microcontrolador que estemos usando.
- Existe una versión con "handshaking", con dos líneas a mayores: RTS y CTS.
- Existen dos estándares que sobresalen del resto, el RS-232 (a 12V) y el RS-485 (a 5V).
- Se suele utilizar en dispositivos de comunicación inalámbrica, unidades GPS y módulos bluetooth.
- Es el precursor del protocolo USB.
- Existe una versión llamada LIN utilizada ampliamente en los automóviles, que comentaremos después.

II. I2C

El siguiente bus que vamos a comentar es el I2C, o Circuito Inter-Integrado. Es uno de los primeros protocolos en permitir comunicar con varios dispositivos, a partir de la jerarquía maestro-esclavo. El maestro, cuando quiere comunicar con uno de los esclavos, manda primero la dirección física de I2C del esclavo. A esta dirección, compuesta de 7 bits, le acompaña un octavo bit para informar si la operación es de escritura o de lectura. A partir de ahí empieza la transmisión de datos a nivel de bytes.

Nosotros vamos a utilizar este bus para el módulo radio y la memoria EEPROM externa.

En la siguiente figura podemos ver de manera genérica como es el formato de comunicación de este bus, a partir de las líneas de dato (SDA) y reloj (SCL):

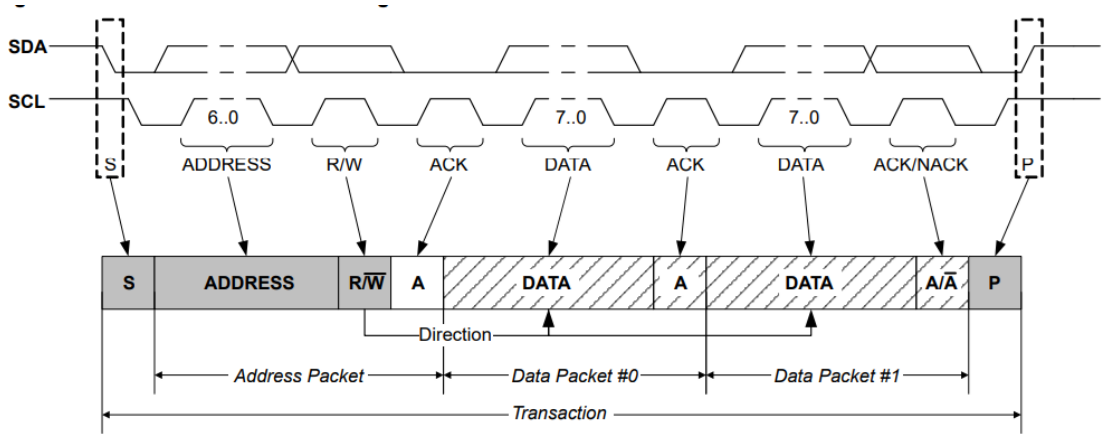


Figura 2. Formato de comunicación del bus I2C

Pasamos a explicar el protocolo de comunicación de este bus:

1. Se produce una condición de Start, marcada como S. Se pone la línea de datos a bajo nivel, mientras se mantiene la señal de reloj en alta.

2. En el siguiente ciclo de reloj, en el flanco de subida, se genera la condición de captación del dato. El primer byte mandado siempre son 8 bits, 7 de dirección física y 1 de operación (R/nW), y siempre lo manda el maestro.

3. Se envía una señal de acuse del mensaje o ACK.

4. Si la operación es de escritura, el maestro manda de nuevo un dato en el siguiente ciclo de reloj y de nuevo el esclavo manda la señal ACK. Si

por el contrario se produce una lectura, los papeles de maestro y esclavo se invierten.

5. Si en algún momento, uno de los datos no es correcto, la señal de ACK se convierte en NACK, o de error en el acuse del mensaje.

6. Tanto si se termina la operación, como si aparece un NACK, el maestro genera una condición forzando la señal de reloj a nivel alto y posteriormente la de datos.

Algunas características de este bus son:

- Como ya se ha comentado, se compone de dos líneas: Datos y Reloj. A nivel físico ambas líneas tienen que llevar resistencias de pull-up.
- El maestro controla el reloj para temporizar los mensajes. Sin embargo, los esclavos pueden tirar abajo la señal de reloj para hacer esperar al maestro. A esto se le llama "clock streching" o estiramiento del reloj.
- La frecuencia del reloj puede variar desde 100kHz hasta 3,4MHz.
- Permite la comunicación con varios dispositivos, pero siempre debe haber un maestro. Se pueden utilizar varios maestros a partir del arbitraje de la línea de reloj.
- Usualmente el direccionamiento se hace a partir de 7 bits. Sin embargo, existe la posibilidad de utilizar 10 bits.
- Por esto mismo, existe una limitación al definir las direcciones I2C. Normalmente los dispositivos permiten modificar de 1 a 3 bits de una dirección especificada para colocar varios en el mismo bus.
- Está recomendado para comunicaciones dentro de la misma PCB. Se puede utilizar a velocidades bajas para pequeñas distancias para comunicar distintos dispositivos.
- Existe una variante SMBus, o Bus de Administración del Sistema definido por Intel en 1995. Describe un uso más estricto del bus.
- A la hora de recibir datos de los periféricos, el protocolo habitual es mandar la dirección de dispositivo con operación de escritura más el comando y automáticamente generar un reinicio en el bus para mandar de nuevo la dirección de dispositivo con operación de lectura, y a partir

de aquí el esclavo manda sus datos. En la siguiente figura podemos ver este sistema aplicado a la comunicación con memorias externas:

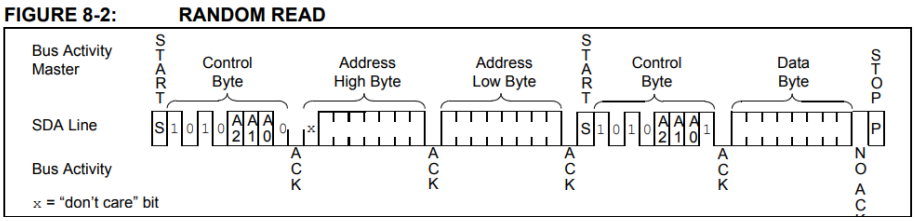


Figura 3. Lectura aleatoria de una memoria a través del bus I2C

La secuencia en este caso es la siguiente:

1. Maestro envía el byte de control con la dirección física del dispositivo y la operación de escritura.

2. Maestro envía 2 bytes de dirección (lógica), o dicho de otra manera, la dirección de la memoria que queremos leer.

3. Se produce un RESTART en el bus.

4. Maestro envía de nuevo el byte de control con la dirección física, pero esta vez la operación es de lectura.

5. Esclavo comienza a transmitir los bytes hasta que el maestro recibe el número de bytes requeridos.

III. SPI

Otro de los buses más conocidos es el SPI, o Interfaz de Periféricos Serie, es muy parecido al bus I2C, utilizando también la jerarquía Maestro-Esclavo. Conserva la línea de Reloj (SCLK), pero la línea de datos se divide en dos: La línea MOSI, o Salida de Datos del Maestro y Entrada del Esclavo; y la línea MISO, o Entrada de Datos del Maestro y Salida del Esclavo. Además, es necesario una línea más por cada dispositivo, el SS, o Selector de Esclavo. Para comunicar con los esclavos, se debe poner a nivel bajo el selector del dispositivo con el que vamos a comunicarnos. Esto permite transmitir datos a varios dispositivos esclavos a la vez. Obviamente, la

recepción de varios dispositivos a la vez sigue sin ser posible porque produciríamos una colisión en el bus.

En nuestro caso, no vamos a utilizar este bus. Lo nombramos por su importancia en el campo electrónico, así como para compararlo con el I2C.

Las principales características de este bus son las siguientes:

- Al contrario que en el I2C, no se necesitan resistencias de Pull-Up.
- La velocidad máxima que el bus puede alcanzar hasta 50 MHz, aunque bajo condiciones muy específicas (3)(4).
- Existen muchas variantes del SPI. Una de las más interesantes es el QuadSPI, en el que los datos se transmiten a partir de 4 líneas de datos. Los datos se reparten a nivel de bit en las 4 líneas, de tal manera que se multiplica por 4 la transmisión de datos. A continuación, podemos ver un esquema de esto:

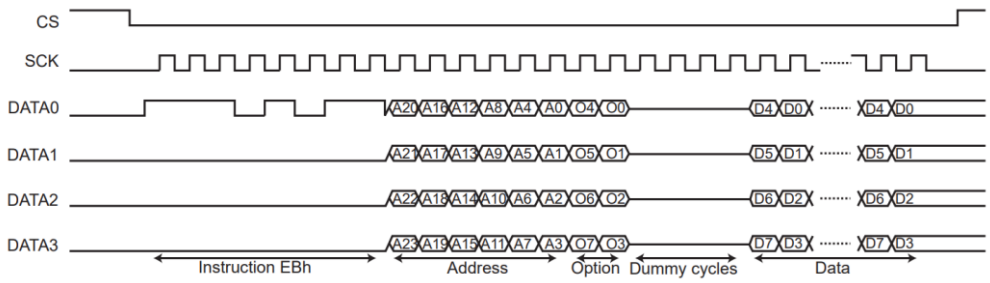


Figura 4. Formato de comunicación del bus QuadSPI

IV. DIFERENCIAS ENTRE I2C Y SPI

Los buses I2C y SPI son muy parecidos y dependiendo de nuestra aplicación, podemos usar uno y otro. Los dos principales factores son el número de dispositivos esclavos, el número de líneas y la velocidad del bus.

Presentamos ahora una tabla con las diferencias entre ambos (5):

Tabla 1. Diferencias entre los buses I2C y SPI

Característica	I2C	SPI
Número de dispositivos	Limitado a 128 con direccionamiento de 7 bits.	Ilimitado siempre tengamos suficientes líneas de selección de esclavo.
Velocidad máxima	3,4 MHz	50 MHz
Multimaestro	Permitido	No permitido
Clock streching	Permitido	No permitido
Número de líneas	2	4 + 1 x esclavo
Resistencia al ruido	Alta	Media
Verificación del dato	Sí	No
Rendimiento del bus	Reducido, al necesitar start/stop bits y direccionamiento físico.	Total
Distancias	Funciona bien a grandes distancias	No funciona tan bien en largas distancias.
Potencia	Consume más energía.	Consume menos energía.

C. BUSES DE COMUNICACIÓN ESTÁNDARES EN AUTOMOCIÓN

I. INTRODUCCIÓN

Uno de los principales campos tecnológicos de constante avance, es el del automóvil. Inicialmente, todos los subsistemas de la automoción se conectaban a través de cables dedicados. Para la dirección y el freno se utilizaban partes hidráulicas y mecánicas. Con el paso del tiempo, la cantidad de subsistemas ha ido aumentando exponencialmente, por lo que este método de comunicación ha quedado obsoleto. Para reducir la cantidad de cableado necesario para la intercomunicación de todos estos sistemas, se introdujo el concepto de "bus de campo" en 1986. Este bus de campo sería el equivalente a un bus serie en el que todos los dispositivos (llamados nodos) estaría conectados. Al compartir el bus de comunicación, se reduciría la cantidad de cableado utilizado, viéndose reflejado en una disminución del coste y del peso del automóvil (6).

II. CAN BUS

Con vistas al bus de campo nombrado anteriormente, el bus CAN fue desarrollado por Bosch entre 1983 y 1986. Se presentó oficialmente el bus CAN en la Sociedad de Ingenieros de Automoción (SAE) en Detroit, Michigan, en 1986. Un año después, Intel introduciría el primer chip controlador CAN, seguido por Philips. En 1991, el Mercedes Benz W140 fue el primer coche producido con sistema de cableado CAN.

Bosch publicó varias versiones de la especificación del bus CAN. La última, CAN 2.0, fue publicada en 1991. A partir de aquí han ido surgiendo diferentes ISO que tratan diferentes capas de este bus. Partiendo del CAN 2.0 de Bosch, pasaremos a explicar cómo funciona este bus.

Este bus es un bus de solamente 2 líneas en su capa física, CAN HIGH y CAN LOW. Estas 2 líneas son líneas diferenciales. Esto se traduce en que ambas líneas pasan a estado "dominante" cuando la línea CAN HIGH tiene más tensión que la línea CAN LOW, y pasan a un estado pasivo (o 0 lógico) cuando ambas están al mismo nivel. Esto tiene dos ventajas:

- Arbitración del bus Automática: Si varios IDs transmiten a la vez, el que tenga menor ID tendrá prioridad a la hora de transmitir. Si un ID mayor transmite a la vez que uno de menor ID, el mayor será capaz de detectar que otro dispositivo ha puesto el bus a 0 lógico y le cederá el control del bus, pasando a la espera hasta que este haya terminado.

- Inmunidad al ruido: A partir de la ISO 11898-2:2003, se consigue inmunidad al ruido a raíz de utilizar resistencias de terminación del bus (120 ohmios) en cada extremo, consiguiendo así mantener la impedancia diferencia del bus a bajo nivel.

Se adjunta ahora un gráfico en el que se pueden ver cómo funciona la comunicación diferencial de este bus, superponiendo las líneas HIGH y LOW del bus CAN y su representación a nivel lógico:

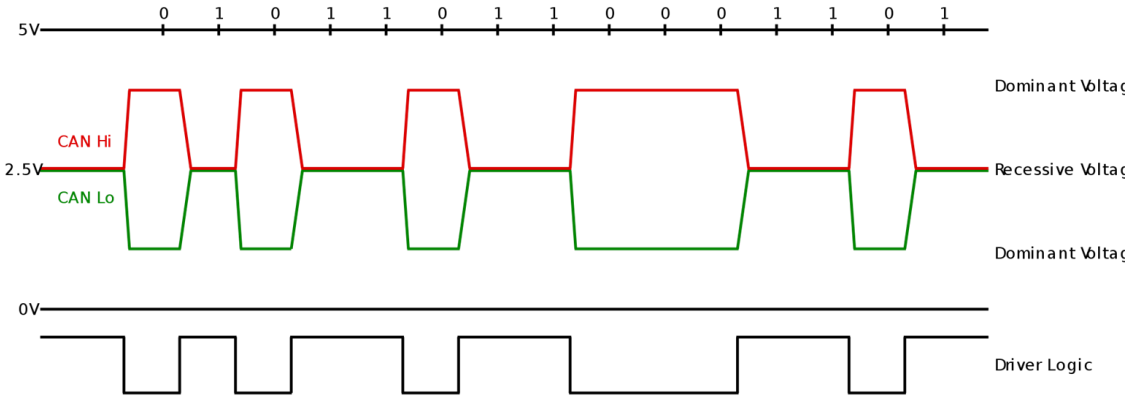


Figura 5. Formato físico de comunicación del bus CAN

En la especificación de CAN BUS 2.0 de Bosch, se definen dos partes: La A, que parte de un formato estándar con un identificador de tamaño 11 bits; y una B, que parte de un formato extendido con un identificador de tamaño 29 bits. Esto nos permite tener hasta 2048 nodos teóricos en el A, y más de 530.000.000 en el B. En el siguiente gráfico se puede ver el frame de la línea CAN y cómo influye utilizar el formado estándar y el extendido:

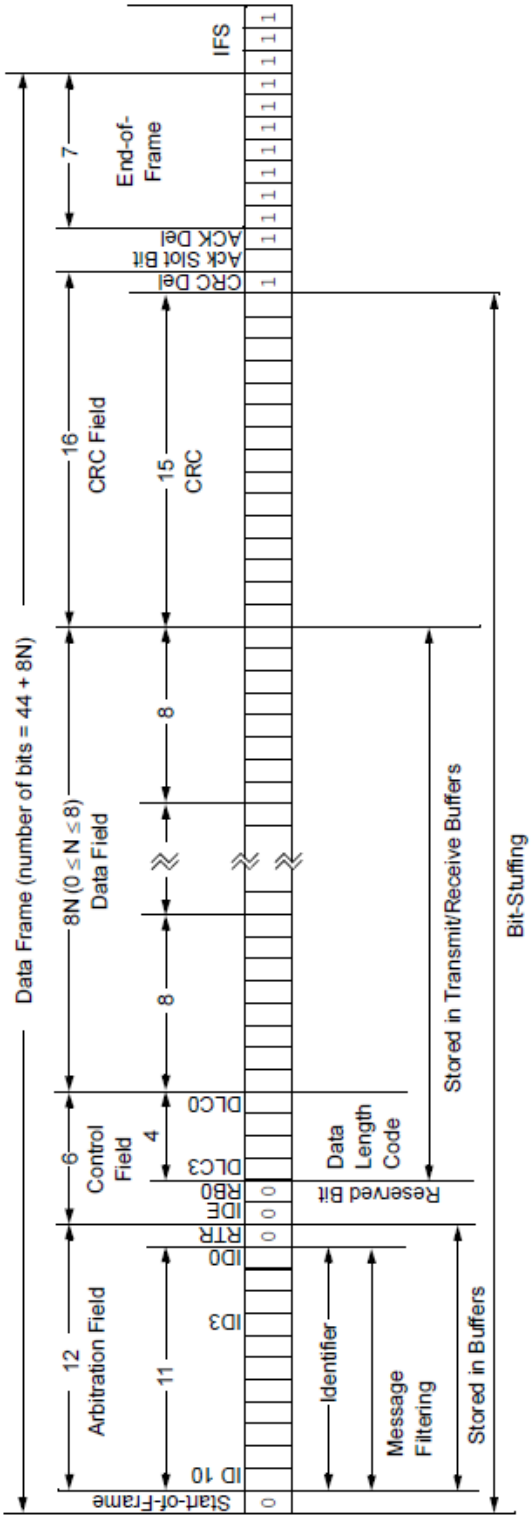


Figura 6. Formato de mensaje del bus CAN con formato estándar

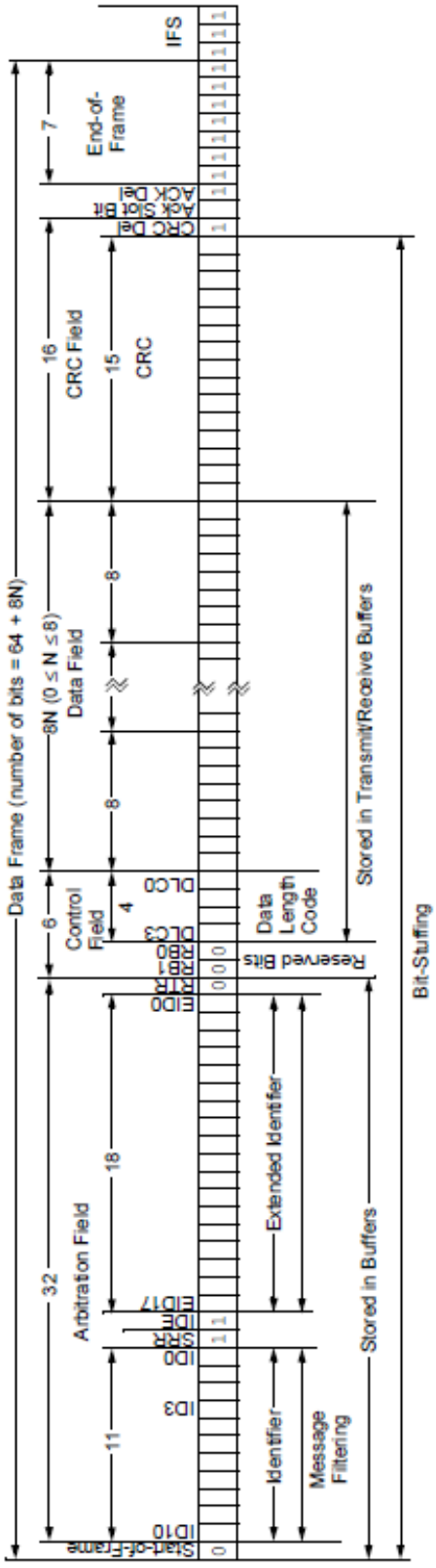


Figura 7. Formato de mensaje del bus CAN con formato extendido

Cada nodo, necesita además de los siguientes componentes:

- Microcontrolador (como nuestro SAM) o unidad central de procesamiento (una ECU en el automóvil).
- Controlador CAN, integrado normalmente en el propio microcontrolador o unidad central.
- Transceptor definido por los estándares de la ISO 11898-2/3.

Algunas de las características principales de este bus son:

- Como ya se ha comentado, sólo necesita dos líneas de comunicación. Además, se puede obtener inmunidad al ruido a partir de la instalación de las resistencias de terminación.
- Arbitración por ID, teniendo prioridad el ID más bajo.
- El tamaño de datos de los mensajes es de hasta 8 bytes. Se puede elegir cuantos a partir del campo DLC, o código de longitud de datos.
- Detección y señalización de errores, así como la retransmisión automática de tramas erróneas y desconexión autónoma de nodos defectuosos.
- Autodiagnóstico.
- La velocidad máxima del bus estándar es de 1 Mbit/s. En 2021, Bosch lanzó CAN FD 1.0, ofreciendo un aumento de la tasa de transferencia, aumentando en 8 el campo de datos (o Payload) y aumentando la velocidad hasta 5 Mbits/s para este.
- La tensión en modo común del bus debe estar entre -2 y 7V. La tensión diferencial del bus en modo dominante debe estar entre 1,5 y 3V.

III. LIN

Posterior al bus CAN, se presentó el LIN, o Red de Interconexión Local. El objetivo de este bus es reducir el coste asociado de implementar CAN a todos los componentes de un automóvil. A finales de los 90, se fundó el consorcio LIN, con algunos de los principales fabricantes de automóviles. La primera versión funcional de la nueva

especificación LIN (LIN versión 1.3) fue publicada en noviembre de 2020. Casi 1 año después, en septiembre, la versión 2.0 se introdujo para expandir las capacidades y extraer datos para futuras funcionalidades de diagnóstico (7)(8).

El formato de mensaje para el LIN se compone de un "Frame Header" y de un "Frame Response". En las siguientes figuras podemos ver esto:

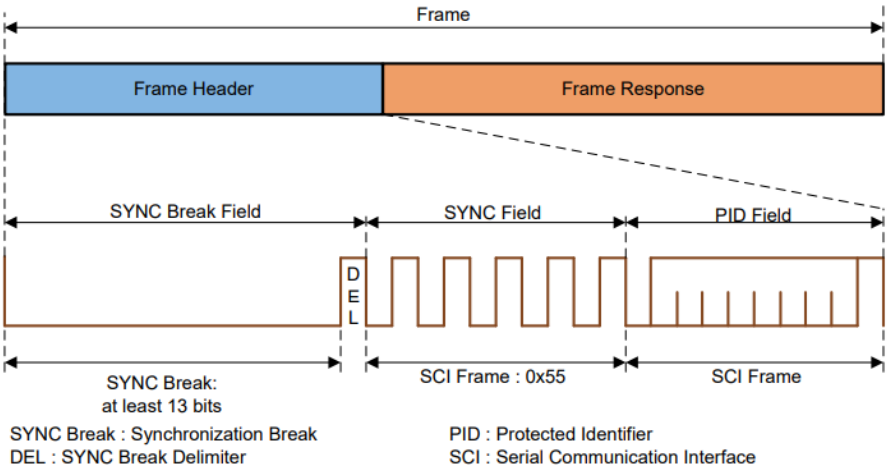


Figura 8. Formato del Frame Header del bus LIN

El Frame Header (o marco de cabecera) siempre es mandando por el maestro, y está divido por el Sync Break (o interrupción de sincronización), el Sync Field (o campo de sincronización) y el PID (o identificador protegido). Tanto el Sync Break como el Sync Field se utilizan para tener a todos los esclavos sincronizados con el timing maestro (sin necesidad de cristal u oscilador) y el PID es lo que define qué esclavo responde, recibe o ignora la cabecera recibida. La cabecera en total consiste en:

- Al menos 13 bits para el Sync Break.
- 1 bit delimitador.
- 10 bits para el Sync Field (descompuesto en 1 bit de Start, 8 bits de sincronización y 1 bit de Stop).
- 10 bits de identificación (compuesto por 1 bit de Start, 6 bits para el identificador, 2 bits de paridad y 1 bit de Stop).

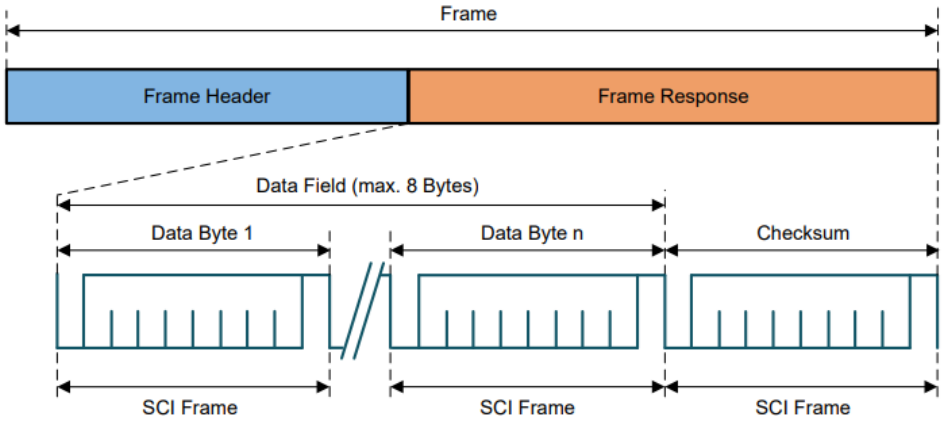


Figura 9. Formato del Frame Response del bus LIN

El Frame Response (o marco respuesta) es la información para la tarea del esclavo, que a su vez puede ser mandada tanto por el nodo maestro como por el esclavo, dependiendo de la instrucción mandada en el PID. La respuesta puede llegar a ocupar hasta 8 bytes de datos, acabando siempre con un Checksum, para verificar que el mensaje está correcto y no ha habido errores durante la transmisión. La respuesta se compone de:

- 10 bits para cada campo de datos (compuestos por 1 bit de Start, 8 bits de Datos y 1 bit de Stop).
- 10 bits para el Checksum (compuesto de igual forma que los del campo de datos).

Algunas de las características de este bus son las siguientes:

- Parten de un bus UART. Sin embargo, sólo se utiliza una línea que hace a su vez de transmisión y recepción.
- Los transceptores de LIN utilizan valores de tensión entre 9 y 18V, aunque algunos llevan a 30V.
- El nodo maestro requiere un pull-up tipicamente de 1 kΩ (aunque se pueden llegar a ver valores entre 600 Ω y 500 Ω) en serie con un diodo de protección.

Los nodos esclavos típicamente utilizan pull-ups de 30 kΩ, que los transceptores modernos ya suelen llevar incluida.

- El bit rate va desde 1 hasta 20kbps, con una tolerancia de ±14 %.
- El byte de sincronización consiste en un "0x55". Esto es una alternancia de 1s y 0s, empezando por el 0.
- Se puede llegar a conectar un maestro con 16 esclavos como máximo.

IV. MOST Y FLEXRAY

Hay otras dos tecnologías que ya se están empezando a implementar y poco a poco empiezan a reemplazar al CAN: El MOST, o Transporte de Sistemas Orientados a Media, y el FlexRay. Ambos utilizan fibra óptica como bus, lo que les permite alcanzar tasas mucho más altas que el CAN, aunque también encarecen el producto. No vamos a entrar en detalle en estos dos buses, porque están fuera del objetivo principal de este proyecto.

D. COMUNICACIÓN INALÁMBRICA

I. INTRODUCCIÓN

El primero en estudiar la comunicación el fenómeno magnético en el que se basa toda comunicación inalámbrica fue Maxwell. Su publicación de 1861 "On Physical Lines of Force" trata las líneas de fuerza como entidades reales, basadas en el movimiento de limaduras de hierro en un campo magnético y apoyándose en la analogía de una rueda inactiva. En su investigación, Maxwell mostró como el comportamiento de los campos eléctricos y magnéticos estaba reaccionado con unas pocas ecuaciones matemáticas simples.

Poco después, Hertz clarificó y expandió la teoría de la luz electromagnética en la que tanto empeño puso Maxwell. Hertz probó que la electricidad puede ser transmitida in ondas electromagnéticas, que pueden viajar a la velocidad de la luz y las cuales poseen muchas otras propiedades de la luz. Entre 1885 y 1889, fue capaz de producir ondas electromagnéticas en el laboratorio y medir su ancho de onda y velocidad.

II. RADIO

Marconi empezó a trabajar con las investigaciones de Maxwell y Hertz y llevo a cabo las primeras comunicaciones vía radio. A finales de 1896, Marconi había demostrado su sistema a la Oficina General de Correos británica y al servicio armamentístico. En 1897, presentó la patente de su invento en la Oficina Británica (No. 12039).

A partir de aquí, Marconi empezó a testear los límites de su invento y pensó en utilizarlo para comunicaciones transoceánicas. En 1899, se realizó la primera llamada de emergencia entre dos busques utilizando el aparato inalámbrico de Marconi. La embarcación mercante Elbe había encallado en Goodwin Sands, un banco de arena de 10 millas de largo en la costa de Deal en Kent, Inglaterra. Un mensaje de auxilio fue recibido por un operador de radio en el faro de South Foreland.

En febrero de 1902, Marconi zarpó hacia los Estados Unidos a bordo del SS Philadelphia, que había sido equipado con antenas unidas a los mástiles del barco, de unos 150 pies de altura (unos 46 metros). Mientras el barco navegaba hacia el oeste, se enviaron señales desde Poldhu (Cornwall, Inglaterra). El capitán aseguró que se recibieron mensajes legibles hasta 1550 millas hacia el mar.

Hacia finales de 1902, Marconi había establecido una estación de radio permanente en Nueva Escocia, Canadá; así como en el Cabo Cod en los Estados Unidos. El Gobernador General de Canadá y Marconi mandaron mensajes al rey Edward VII. En enero de 1903, el primer mensaje inalámbrico fue transmitido directamente desde USA a Inglaterra, por el presidente al rey Edward VII.

III. TELEFONÍA MÓVIL

Ericsson fue una de las personas más influyentes detrás de la fabricación de los teléfonos. Contribuyó sustancialmente al diseño de las primeras centrales telefónicas, diseñando y produciendo el primer “escritorio múltiple” en Europa en 1884, que se utilizaron durante más de medio siglo.

El 17 de junio de 1946 en Sant Louis (Missouri), AT&T y Southwestern Bell presentaron el primer teléfono móvil comercial con servicio de radiotelefonía a clientes particulares. Operaba en seis canales en la banda de los 150MHz con un ancho entre canales de 60kHz. Interferencias en el canal, así como conversaciones cruzadas obligó a Bell a utilizar solamente 3 canales.

En enero de 1969, Bell System puso en funcionamiento la radio celular comercial al emplear la reutilización de frecuencias por primera vez. Se utilizaron seis canales en la banda de los 450 MHz una y otra vez a lo largo de una ruta de 362 km.

La creación del primer microprocesador en 1971, el 4004, y su adición a los sistemas electrónicos, mejoró enormemente el rendimiento de estos.

En 1982, el grupo de 26 compañías móviles europeas llamado GSM, o Grupo de Especialidades Móviles, ahora conocido como Sistema Global de Comunicaciones Móviles, empezó el desarrollo de un nuevo tipo de comunicación que se adaptara a las necesidades de la época, utilizando una nueva banda de radio, alrededor de los 900 MHz. Este grupo patentó un nuevo estándar inalámbrico estructurado de manera celular pero totalmente digital.

El futuro de esta comunicación móvil dependerá en gran medida de las tecnologías disponibles y del concepto de micro células. Estas micro células son las áreas servidas por cada estación base de radio. En el futuro, estas bases serán de uno o dos órdenes de magnitud más pequeños que el sistema actual.

IV. WIFI

WiFi, o Wireless Fidelity, es el nombre dado por Wi-Fi Alliance al paquete de estándares IEEE 802.11, que a su vez define el estándar para redes de área locales inalámbricas (o WLAN). Se lanzó por primera vez en 1997. Las frecuencias que utilizaba este WiFi eran principalmente 2: 2,4GHz y 5 GHz.

En 2003, se lanzó un nuevo estándar, el 802.11g, que daba mayores velocidades y una mayor cobertura de distancia que los anteriores estándares. A partir de aquí, el WiFi empezó a competir en velocidad con las conexiones de cable más rápidas.

En 2009 se lanzó el estándar 802.11n, aún más rápido y robusto que su predecesor. Esta mejora se atribuye a la adición de la tecnología MIMO, o Múltiples entradas y Múltiples Salidas.

En los últimos años, se ha visto como la banda de los 2,4GHz se ha ido congestionando (en parte también debido al Bluetooth, del que hablaremos después). Esto ha provocado que la banda de los 5GHz repunte, y a su vez, que todos los dispositivos dispongan de un enrutador con las dos bandas, incluso simultánea (9).

V. BLUETOOTH

A mediados de la década de los 90, la compañía Ericsson se encontraba desarrollando una nueva tecnología que permitiera la comunicación express entre dos dispositivos cercanos, todo de manera inalámbrica y con poco uso energético. El avance en esta nueva tecnología permitió que se formara una SIG, o grupo de especial interés, entre las grandes empresas del sector. Entre estos grandes se encuentran, por ejemplo: Apple, Intel, Lenovo o Toshiba. Hoy en día, más de 14.000 empresas forman parte de este grupo (10).

El nombre de esta nueva tecnología inicialmente era MCLink, mientras que Bluetooth era el nombre en clave para las distintas betas del proyecto. Al grupo le pareció un buen nombre y lo adoptaron. El origen de este nombre proviene del rey nórdico Herald Blåtand, cuya traducción al inglés sería "Bluetooth", o diente azul traducido al español. A su vez, el logo proviene de las iniciales de éste en alfabeto rúnico, la H y la B.

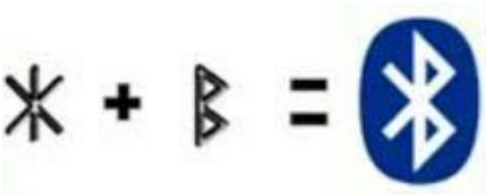


Figura 10. Creación del logo de Bluetooth

En 1998, Bluetooth vio la luz, con unas características muy notables:

- Poco gasto energético.
- Conexiones y enlaces por lo general de corta duración.
- Seguridad mediante diversas maneras de cifrado de datos.
- Soporta voz y datos.
- Bajo coste de producción, planteado para que no sobrepasara los $5 americanos por dispositivo.
- Permitía una velocidad teórica de 1Mbps a una distancia menor de 10 metros.

A partir de aquí, se han ido creando nuevas versiones de este estándar, hasta llegar a la actual, la versión 5.0 (11), cuyas principales mejoras son:

- Un consumo aún más bajo.
- Audio dual, permitiendo así reproducir audio diferente en varios dispositivos a la vez.
- Hasta 2Mbps, y una distancia de hasta 200 metros.

3. REQUISITOS DEL SISTEMA EN EL AMBITO COMERCIAL

A. DIRECTIVA 2014/30/UE – COMPATIBILIDAD ELECTROMAGNÉTICA

Uno de los objetivos de este proyecto es poder presentar una placa que pueda llegar a convertirse en un modelo comercial. Para ello, debemos tener en cuenta la normativa actual.

La primera directiva es la de compatibilidad electromagnética (EMC a partir de ahora) (12). Esta directiva se encarga de que todos los equipos eléctricos y electrónicos ni generen, ni sean afectados por interferencias electromagnéticas. La EMC limita las emisiones electromagnéticas para asegurar que, durante su uso cotidiano, tales equipos no provocan interferencias en radios o telecomunicaciones, así como en otros equipos eléctricos o electrónicos. Esta directiva también controla la inmunidad de tales dispositivos a interferencias externas y busca, por contrapartida, que tampoco sean afectados por emisiones radio cuando se hace un uso cotidiano de él.

Para cumplir los requisitos de esta directiva, existen pequeñas guías de trabajo, que a su vez son un compendio de buenas prácticas ingenieriles. Nosotros vamos a focalizarnos en el diseño electrónico. Aquí se enumeran algunas de estas prácticas:

- Unión a tierra o planos de masa para RF.
- *Cables y carcasas apantallados*. Dicho de otra manera, todos los cables y carcasas deben tener un punto de unión a tierra, o al menos al plano de masas de la placa. De esta manera, se impide el escape de cualquier señal del núcleo del conductor.
- Desacoplo o filtro de puntos críticos, como entradas de cables o interruptores de alta velocidad, con el uso de bobinas y/o circuitos RC.
- Técnicas para las líneas de transmisión de datos, como pueden ser señales diferenciales balanceadas o líneas de impedancia controlada.
- Evitar usar estructuras tipo antena, como son los bucles de corriente o estructuras mecánicas resonantes.

B. DIRECTIVA 2014/53/UE – LEGISLACIÓN ARMONIZADA SOBRE LA COMERCIALIZACIÓN DE EQUIPOS RADIOELÉCTRICOS

La directiva sobre equipos radioeléctricos (a partir de ahora RED) (13) establece un marco de trabajo regulado para poner a la venta equipos radioeléctricos. Con esta normativa, se busca un mercado unificado para este tipo de equipos cuyos requerimientos pasan por seguridad y salud, compatibilidad electromagnética y un uso eficiente del espectro de radio. También proporciona los básicos para la regulación por parte del gobierno en algunos aspectos adicionales. Estos incluyen aspectos técnicos para la protección de la privacidad, de los datos personales y ante el fraude. Por último, también incluye aspectos relacionados con la interoperabilidad, acceso a servicios de emergencia y el cumplimiento normativo (Compliance) basándose en la combinación de equipos radioeléctricos y software.

Algunos de los requisitos esenciales, que seguiremos a la hora de la creación de este proyecto, son los siguientes:

- Aplicación de la normativa de la compatibilidad electromagnética, nombrada en el anterior párrafo.
- Uso eficiente del espectro radioeléctrico con el fin de evitar interferencias.
- Salvaguardas para garantizar la protección y privacidad de los datos personales.
- Facilidades a la hora del uso por parte de usuarios con discapacidad.
- Interacción con otros equipos radioeléctricos a través de redes.

4. HARDWARE EMPLEADO

A. COMPONENTES ELECTRÓNICOS

I. PLACA DE DESARROLLO CON MICROCONTROLADOR CORTEX: *MICROCHIP SAME54 XPLAINED PRO*

En nuestro caso, vamos a utilizar el ARM Cortex-M4F ATSAME54P20, SAM a secas a partir de ahora, con la placa de desarrollo proporcionada por Microchip, Xplained PRO.

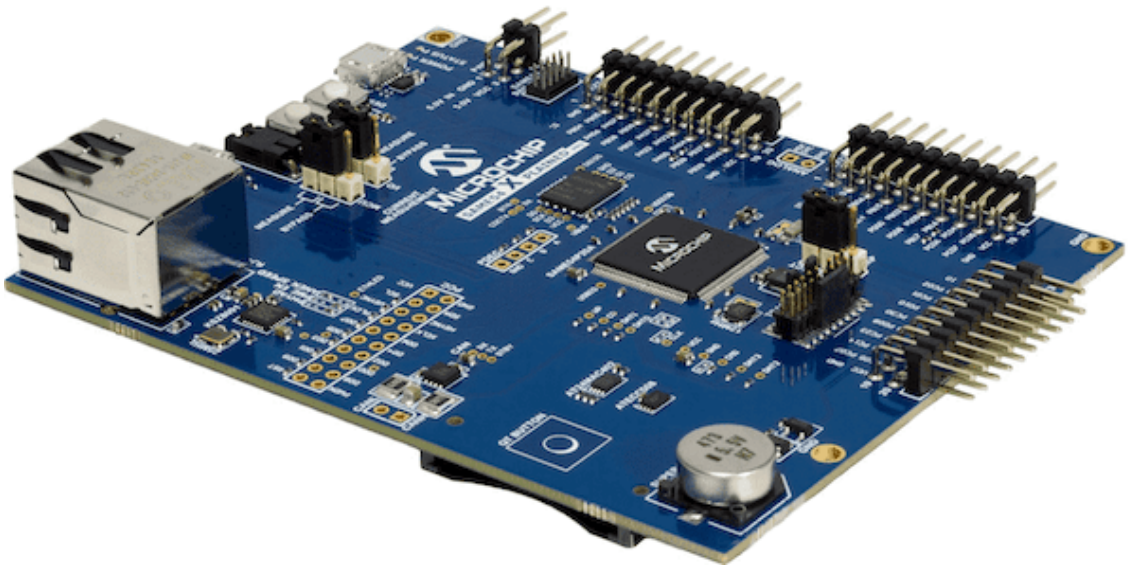


Figura 11. Placa de desarrollo Xplained PRO SAME54

Este microcontrolador pertenece a la familia de los ARM de 32 bits, que hemos comentado anteriormente. Es capaz de trabajar a frecuencias de hasta 120 MHz. Lo podemos encontrar en sus versiones de 120 y 128 pines, en función del empaquetado que utilice. Entre sus innumerables funciones (14), podemos encontrar las siguientes:

- Hasta 2 interfaces CAN-FD.
- USB de máxima velocidad para comunicación mediante puerto serie.
- Soporta 5 modos de bajo consumo. Entre ellos, destacaremos el modo hibernación en el que el consumo decae hasta 3 µA.

- Seguridad integrada incluyendo aceleración por CryptoHardware simétrica y asimétrica.
- Unidad de punto flotante, utilizando el estándar IEEE 754 en precisión simple.
- Detección de caída de tensión (Brown-out detection o BOD en inglés).
- Generador de números aleatorios real (TRNG).
- Sensor de temperatura.
- Múltiples buses de comunicación, como SPI y QSPI, I2C, LIN y RS485.
- 32 canales para un controlador de acceso directo a memoria (DMAC).
- Timer Watchdog (WDT) con modo ventana.
- Decodificador de posición a partir de la lectura de encoders de posición de cuadratura o tipo Hall.

Al utilizar la placa de desarrollo, nos encontramos facilidades a la hora de utilizar este microcontrolador, sin necesidad de soldar ni hacer modificaciones. Enumeramos algunas de estas ventajas:

- Pines disponibles, con fácil conexionado a las diferentes funciones del SAM.
- Cristal, botones y Leds incluidos.
- Transductor de CAN incluido.
- Memoria EEPROM incluida.
- Headers para conectar otros dispositivos de la familia Xplained PRO de forma directa. Nosotros utilizaremos la pantalla OLED de esta familia.
- Depuración por USB, sin necesidad de depurador externo.
- Puesto de Comunicaciones Virtual, para comunicar por el puerto serie del PC de manera instantánea.

Se ha elegido esta placa de desarrollo por varias razones. La primera es que utilizamos uno de los microcontroladores más potentes del mercado en una placa que incluye todo lo necesario para empezar a trabajar. Posteriormente se verá que las funcionalidades de éste están muy por encima de los requerimientos del proyecto.

Por otro lado, al tener incluido la depuración por USB, nos ahorramos el tener que adquirir un depurador para el microcontrolador. Los depuradores para este tipo de microcontroladores suelen valer más de 200€.

Además, es muy fácil utilizar los conectores para las diferentes funciones que necesitamos. Por ejemplo, en la placa van integrados un transductor de CAN, botones, Leds. Incluso la creación de módulos facilita la conexión con diferentes módulos de la misma familia. A continuación, se adjunta una tabla con los pines utilizados para cada uno de estos módulos:

Tabla 2. Descripción de los pines de módulos Xplained PRO

Pin Number	Pin Name	Description
1	ID	Pin to communicate with the ID chip of an extension board.
2	GND	Ground.
3	ADC(+)	Analog-to-Digital Converter; alternatively, a pin for the positive terminal of a differential ADC.
4	ADC(-)	Analog-to-Digital Converter; alternatively, a pin for the negative terminal of a differential ADC.
5	GPIO1	General purpose I/O pin.
6	GPIO2	General purpose I/O pin.
7	PWM(+)	Pulse-Width Modulation; alternatively, a pin for the positive part of a differential PWM.
8	PWM(-)	Pulse-Width Modulation; alternatively, a pin for the negative part of a differential PWM.
9	IRQ/GPIO	Interrupt request pin and/or general purpose I/O pin.
10	SPI_SS_B/GPIO	Slave select pin for Serial Peripheral Interface (SPI) and/or general purpose I/O pin.
11	I2C_SDA	Data pin for I2C interface. Always connected, bus type.
12	I2C_SCL	Clock pin for I2C interface. Always connected, bus type.
13	UART_RX	Receiver pin of target device UART.
14	UART_TX	Transmitter pin of target device UART.

15	SPI_SS_A	Slave select pin for Serial Peripheral Interface (SPI). This pin should preferably not be connected to anything else.
16	SPI_MOSI	SPI master out slave in pin. Always connected, bus type.
17	SPI_MISO	SPI master in slave out pin. Always connected, bus type.
18	SPI_SCK	SPI clock pin. Always connected, bus type.
19	GND	Ground pin for extension boards.
20	VCC	Power pin for extension boards.

Por último, al incluir una memoria EEPROM dentro de la placa, nos ahorra el tener que conectar otra más. Aunque al final se ha visto que esta memoria no dispone de la memoria suficiente, nos ha servido para empezar a trabajar y realizar las primeras pruebas.

II. MEMORIAS: *AT24MAC402 & AT24C256*

Como ya se ha comentado al final del apartado interior, se empezó utilizando la memoria EEPROM incluida en la Xplained PRO, pero finalmente se ha decidido usar una memoria externa de más capacidad en el diseño final de la placa. La Xplained PRO también incluye una memoria Flash de mucha más capacidad comandada por QSPI. Sin embargo, no se ha utilizado al darse situaciones de incompatibilidad con otras funcionalidades del proyecto.

La memoria incluida y utilizada inicialmente es la AT24MAC402, EEPROM de 2 kb de memoria. Además de memoria, incluye una dirección MAC para su uso con la interfaz Ethernet.

Figura 12. Memoria AT24MAC402

Sin embargo, 2 kb es muy poco y se ha reemplazado por una AT24C256, con capacidad para 256 kb (o 32.768 bytes), dividido en 512 páginas de 64 bytes. La comunicación con esta memoria se realiza por I2C y permite modificar su dirección física en función de 2 entradas, permitiendo así tener 4 memorias conectadas y aumentar por 4 la capacidad total del sistema.

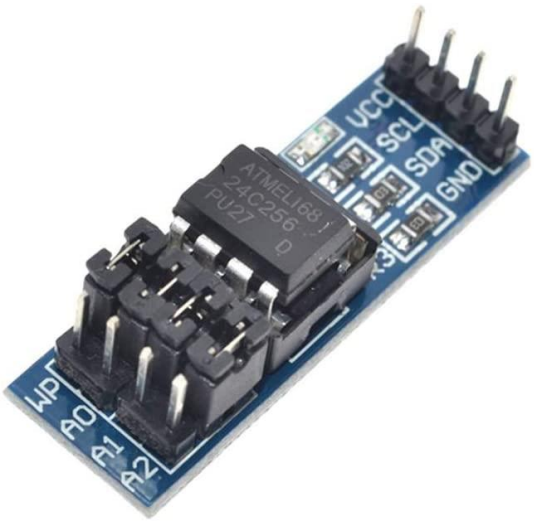


Figura 13. Módulo con memoria AT24C256

Esta memoria también permite habilitar la protección contra escritura, a partir de la puesta a punto del pin correspondiente a VCC. Como no vamos a utilizar esta función, el fabricante recomienda ponerlo a GND.

Esta memoria nos va a servir para guardar los valores máximos de potencia para cada frecuencia de uso FM, así como las coordenadas de cada uno de estos puntos y la fecha a la que se ha hecho el escaneo. También guardaremos otros datos, como puede ser el número de escaneos completos.

III. MÓDULO RADIO: *TEA5767*

El módulo radio utilizado es el TEA5767. Se vende en una placa, en la que se incluyen dos conectores Jack (uno para conectar la antena y otro para la salida de audio), los conectores de alimentación y comunicación y un módulo TDA1308 controlador de sonido estéreo de clase AB.

Figura 14. Módulo TEA5767

Este módulo nos permite convertir la señal de radio a una Frecuencia Inmediata (IF) y entre otras cosas, extraer la fuerza de la señal y reproducirla con el módulo adicional de sonido. Permite utilizar tanto las bandas europea y americana (US), desde los 87.5 hasta los 108 MHz, como la banda japonesa, desde los 76 hasta los 91 MHz. No necesita de un oscilador externo para alcanzar, ya que ya lleva incluidos dos, uno tipo RTC (32.768 kHz) y otro de 13 MHz. El bus que utiliza para la comunicación es I2C, con una frecuencia máxima de 400 kHz.

Al inicio del programa, configuraremos este módulo para que trabaje en la banda europea. Luego, iremos modificando la frecuencia FM de la que debe extraer la fuerza de la señal. Como el módulo necesita hasta 10 ms para contestar a nuestros mensajes, este será nuestro tiempo para cada ciclo. Para asegurarnos de tener lecturas correctas, leeremos la fuerza de la señal varias veces y haremos una media de los valores obtenidos.

IV. PANTALLA OLED: *MICROCHIP OLED1 XPLAINED PRO*

Una de las ventajas de usar esta placa de desarrollo, es la cantidad de dispositivos que tenemos totalmente compatibles diseñados específicamente para ella. Uno de ellos es el módulo de Microchip OLED1 Xplained PRO, que como su propio nombre indica, incluye una pantalla OLED. Su interfaz de conexión coincide con la de la placa, lo que facilita el uso de esta. Además, el configurador Atmel Start nos añade toda la estructura de firmware de forma directa y nos permite empezar a trabajar de manera inmediata. Además de la propia pantalla, este módulo incorpora 3 Leds y 3 Botones que podemos usar para realizar diferentes funciones.

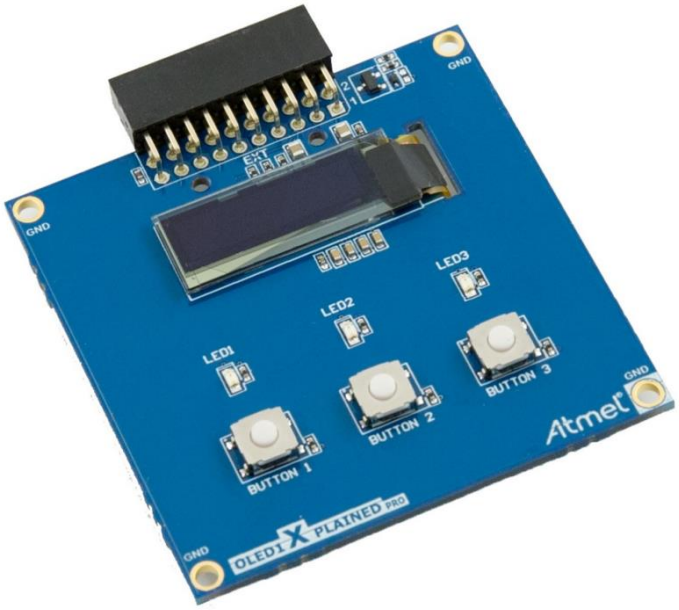


Figura 15. Módulo OLED1 Xplained PRO

La pantalla es una UG-2832HSWEG04 monocroma de 128 x 32 píxeles, controlada por una interfaz SPI de 4 líneas, de hasta 100 MHz. El controlador incorporado es el

SSD1306, de Solomon Systech. La interfaz de la familia Xplained Pro utilizada es la siguiente:

Tabla 3. Interfaz de pines del módulo OLED1 Xplained PRO

Pin Number	Pin Name	Description
1	ID	Communication line to ID chip.
2	GND	Ground.
3	BUTTON2	Push button 2, active-low.
4	BUTTON3	Push button 3, active-low.
5	DATA_CMD_SEL	Data/command select for OLED display. High = data, low = command.
6	LED3	LED3, active-low.
7	LED1	LED1, active-low.
8	LED2	LED2, active-low.
9	BUTTON1	Push button 1, active-low.
10	DISPLAY_RESET	Reset line for OLED display, active-low.
11	NC	Not connected.
12	NC	Not connected.
13	NC	Not connected.
14	NC	Not connected.
15	DISPLAY_SS	OLED display slave select, active-low
16	SPI_MOSI	MOSI signal SPI connected to OLED display.
17	NC	Not connected.
18	SPI SCK	Clock signal for SPI connected to OLED display.
19	GND	Ground.
20	VCC	Target supply voltage.

Utilizaremos esta pantalla para mostrar la selección de las diferentes operaciones que puede realizar nuestro dispositivo. Así mismo, utilizaremos los botones para navegar entre las distintas opciones, y los Leds acompañaran mostrando cuando se están pulsados estos botones.

V. GPS: *U-BLOX NEO-8M*

El GPS utilizado es el NEO-8M de la compañía u-blox. Con su alta sensibilidad de -167 dBm en la navegación y su bajo consumo, es uno de los GPS más punteros de la actualidad a nivel de usuario. Sus módulos GNSS (o sistema global de navegación por satélite) le permiten ser compatible con la gran mayoría de sistemas de posicionamiento. Entre ellos, tenemos el universal GPS o el tan comentado actualmente Galileo. También posee una batería recargable y una EEPROM para guardar los valores de configuración, además de un odómetro para extraer la distancia recorrida.

Figura 16. Módulo GPS u-blox NEO-8M

El módulo de evaluación que utilizaremos es el que nos proporciona WAVGAT. Dispone de los conectores de alimentación y comunicación, dos leds de alimentación y aviso de geolocalización y la antera cerámica. Se adjunta el esquema de este:

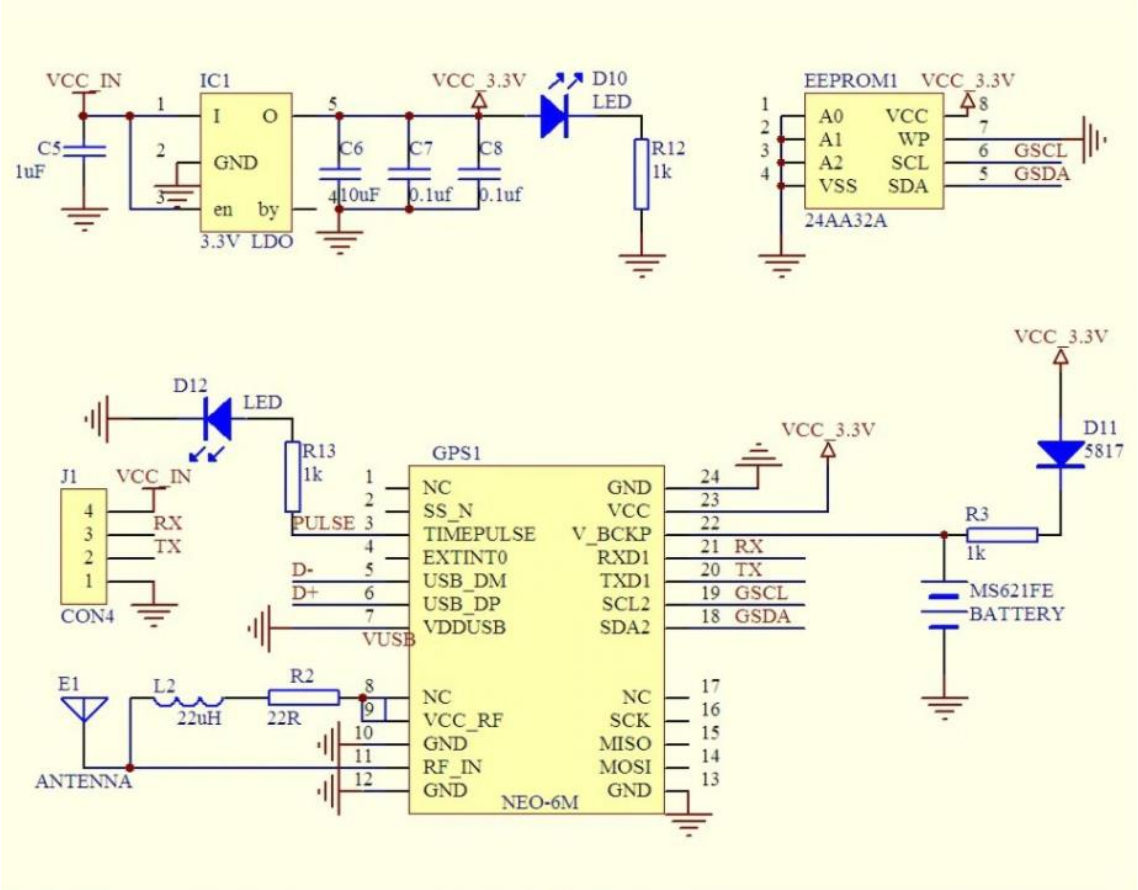


Figura 17. Esquemático del módulo GPS u-blox NEO-6M

Nota: Aunque en el esquema aparece el NEO-6M, es igual para el modelo 8M.

En nuestro caso, vamos a utilizar este módulo para extraer la localización y vincularlos a la fuerza de la señal, y guardar ambos en nuestra EEPROM externa. Además, como recibimos la hora de nuestra zona geográfica, utilizaremos esta como referencia para emular nuestro RTC.

B. COMPONENTES COMERCIALES

I. TRANSMISOR RADIO BLUETOOTH: *VICTSING GEBH378AB*

Para el emisor hemos utilizado uno de tantos que abundan en el mercado. En nuestro caso hemos seleccionado uno de la empresa VicTsing. Entre las ventajas de este modelo, destacan el estándar Bluetooth 5.0 y un puerto de carga capaz de entregar

hasta 3 A, que nos permite alimentar nuestra placa utilizando la alimentación del mechero del coche.

Figura 18. Transmisor Radio-Bluetooth Victsing GEBH378AB

El funcionamiento es muy sencillo. El dispositivo se vincula a nuestro dispositivo Bluetooth, en este caso nuestro teléfono móvil. Desde este momento, todo el sonido que transmita nuestro teléfono móvil será enviado al emisor radio, y este a su vez, lo transmitirá en la banda FM que hayamos seleccionado previamente. Es importante que en la banda FM no exista ya otra transmisión, habitualmente de una estación de radio, ya que empezaremos a ver como se acoplan ambas señales y muy probablemente la otra señal tape la nuestra. Aquí la importancia de seleccionar una banda FM libre, e incluso cambiar en función de la localización en la que estemos.

En nuestro caso, para las pruebas, lo único que haremos será cambiar de frecuencia de transmisión manualmente para comprobar el funcionamiento correcto de todo el sistema.

5. ESTRUCTURA DEL FIRMWARE

A. ARQUITECTURA DEL MICROCONTROLADOR

Como se ha comentado anteriormente, el microcontrolador a utilizar es el ATSAME54P20A, nombrado SAM a partir de ahora.

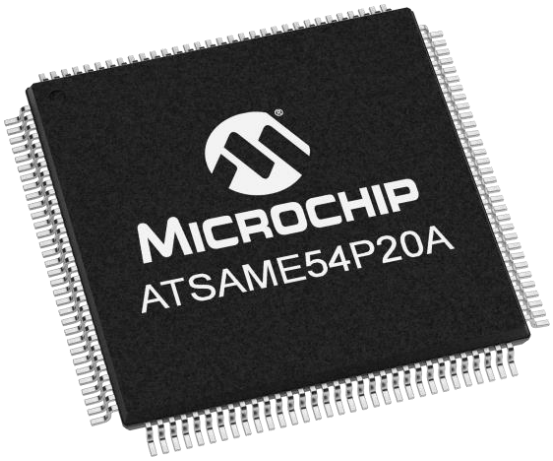


Figura 19. Microcontrolador ATSAME54P20A

Nuestro SAM es un procesador ARM® Cortex™ M4 de 32 bits, de alto rendimiento. Algunos de los beneficios que nos ofrece son:

- Un excelente rendimiento de trabajo, combinado con un rápido manejo de las interrupciones.
- Depuración del sistema mejorada con amplias capacidades de seguimiento y puntos de interrupción.
- Núcleo del procesador, sistema y memorias eficientes.
- Consumo de energía ultra bajo con múltiples modos de suspensión integrados.
- Seguridad del sistema robusta, con unidad de protección de memoria integrada (MPU).
- Arquitectura Harvard de 3 estados, convirtiéndolo en un microcontrolador ideal para aplicaciones embebidas.
- Controlador de interrupción vectorial anidadas (NVIC) incluido, lo que le da un rendimiento líder en industria. Permite utilizar hasta 8 niveles de interrupciones (NMI).

- Frecuencia de trabajo de hasta 120 MHz.

A continuación, se incluye la tabla comparativa con las características principales de los Cortex M4 frente a las características de nuestro ATSAME54:

Tabla 4. Principales características del Microcontrolador Cortex M4 genérico vs ATSAME54

Interrupts	*1 to 240*	*138*
Number of priority bits	*3 to 8.*	*3 = eight levels of priority.*
Data endianness	*Little-endian or big-endian.*	*Little-endian.*
SysTick Timer calibration value		*0x80000000.*
MPU	*Present or not present.*	*Present.*
Debug support level	*0 = No debug. No DAP, breakpoints, watchpoints, Flash patch, or halting debug.* *1 = Minimun debug. Two breakpoints, one watchpoint, no Flash patch.* *2 = Full debug minus DWT data matching.* *3 = Full debug plus DWT data matching.*	*3 = Full debug plus DWT data matching.*
Trace support level	*0 = No trace. No ETM, ITM or DWT triggers and counters.* *1 = Standard trace. ITM and DWT triggers and counters, but no ETM.* *2 = Full trace. Standard trace plus ETM.* *3 = Full trace plus HTM port.*	*2 = Full trace. ITM, TPIU, ETM and DWT triggers and counters are present. HTM port is not present.*
JTAG	*Present or not present.*	*Not present.*
Bit Banding	*Present or not present.*	*Not present.*
FPU	*Present or not present.*	*Present.*

Por último, los periféricos de sistema son los siguientes:

- Sistema de eventos de 32 canales.
- Hasta ocho puertos de comunicación serie, cada uno de ellos configurable como:
    - USART con configuración full-duplex y half-duplex de línea única.
    - ISO7816.
    - I2C de hasta 3.4 Mhz.
    - SPI.
    - LIN maestro o esclavo.
    - RS485.
- Hasta ocho temporizadores/contadores (TC) de 16 bits.
- Dos temporizadores/controladores de control (TCC) de 24 bits, con funciones extendidas:
    - Hasta seis chanales de comparación con salidas complementarias.
    - Generación de PWM sincronizados.
    - Protección ante fallo determinista y tiempo muerto configurable entre salidas complementarias.
- Hasta tres temporizadores/controladores de control (TCC) de 16 bits, con funciones extendidas:
    - Hasta tres canales de comparación con salidas complementarias.
- Modos de generación de PWM usando TCs y TCCs:
    - Hasta seis canales PWM con TCCs de 24 bits en cada uno.
    - Hasta tres canales PWM con TCCs de 16 bits en cada uno.
    - Hasta dos canales PWM con TCs de 16 bits en cada uno.
- Contador de tiempo real (RTC) de 32 bits con función de reloj y calendario.
- Hasta cinco pines despertadores con detección de sabotaje y antirrebote.
- Temporizador Watchdog (WDT) con modo ventana.
- Generador CRC-32.
- Interfaz de sonido de dos canales Intra-IC (I2S).
- Decodificador de posición (PDEC).
- Medidor de frecuencia (FREQM).
- Un controlador de lógica con 4 tablas de verdad customizables (CCL).
- Convertidor Analógico Digital (ADC) de 1 MSPS y 12 bits dual, con hasta 16 canales preparados para:
    - Entradas individuales o diferenciales.
    - Compensación de error en ganancia y offset automático.
    - Supermuestreo y truncamiento en hardware para soportar resoluciones de 13, 14, 15 o 16 bits.
- Convertidor Digital Analógico (DAC) de 1 MSPS y 12 bits dual.
- Dos comparadores analógicos (AC) con modo ventana.
- Dos sensores de temperatura.
- Controlador de captura paralela (PCC) de hasta 14 bits.

- Controlador táctil periférico (PTC):
  - Acepta botones, deslizadores y ruedas de toque capacitivos.
  - Despertador al toque.
  - Hasta 32 canales de capacitancia propia y hasta 256 canales de capacitancia compartida.
- Gran soporte en modelos y sistemas de criptografía.
- Generador de número aleatorio real.

B. LENGUAJE Y COMPILADOR

El lenguaje que vamos a utilizar es C++, apoyado por la cadena de herramientas de GNU para microcontroladores ARM. Esta colección de herramientas viene incluida en nuestro software de desarrollo Microchip Studio, del que hablaremos después. Las herramientas al software pueden ser enumeradas en las siguientes:

I. COMPILADOR

El compilador utilizado es la colección de compiladores de GNU, o GCC. Este compilador es extremadamente flexible y puede ser utilizado en muchas plataformas.

Este GCC incluido va orientado a procesadores ARM y puede ser configurado para compilar C o C++. GCC compila en lenguaje de computador de alto nivel en ensamblador.

II. EMSAMBLADOR, ENLAZADOR Y ARCHIVADOR

GNU Binutiles es una colección de utilidades binarias. Incluye también la herramienta ensambladora, el enlazador y el archivador, entre otras muchas utilidades.

III. LIBRERÍA C

Newlib es la librería estándar para GCC y ARM. Está orientado para usarse en sistemas embebidos. Es una conglomeración de diversas partes de librerías. La librería se ha porteado para soportar procesadores ARM.

Además de la librería estándar para C, newlib-nano también se ha añadido al paquete. Esta newlib-nano es una rama de la newlib optimizada para tamaño de código en ARM.

IV. DEPURADOR

Preparada para ser totalmente compatible con nuestro software de desarrollo a través de diferentes depuradores físicos.

C. ESTRUCTURAS DE TIEMPO REAL (RTOS)

Los RTOS o sistemas operativos de tiempo real son sistemas operativos orientados a servir en aplicaciones de tiempo real que procesan la información en el momento que llega, típicamente sin retrasos por culpa de los buffers. Los requerimientos de tiempo de procesado son muy exigentes, utilizando de base incrementos de decenas de segundo o incluso menores. En sistema de tiempo real es un sistema muy orientado a la respuesta prácticamente inmediata. Por ello, deben tener bien definidos los tiempos necesarios para cada operación.

Nuestro microcontrolador no utiliza tiempo real per se. Sin embargo, nosotros vamos a emular esta funcionalidad (con claras diferencias). Para ello, seguiremos una pauta imprescindible: Todo el procesamiento debe ir en el bucle principal. Por lo tanto, vamos a procurar que el procesamiento y tratamiento de las interrupciones sea lo más rápido posible. Por ejemplo, si tenemos una interrupción de un temporizador, simplemente activaremos un flag para decirle al programa principal que ha ocurrido esa temporización. En el caso de las comunicaciones, guardaremos los datos recibidos en un buffer, que posteriormente se analizará en el bucle principal también tras la activación de un flag (15).

Al liberar las interrupciones al máximo, podemos asegurar que nuestro microcontrolador, aun trabajando a bajas frecuencias, podrá actuar como un sistema de tiempo real.

D. ENTORNOS DE DESARROLLO

I. MICROCHIP STUDIO

Para programar nuestro microcontrolador, utilizaremos Microchip Studio, el Entorno de Desarrollo Integrado (IDE). Este software es el utilizado para programar y depurar las familias de microcontroladores AVR y SAM. Combina todas las funcionalidades del software Atmel Studio, que a su vez se apoya en Visual Studio de Microsoft, para ofrecer una herramienta muy completa y funcional para programar en los lenguajes de C y C++. También permite importar los sketches de Arduino como proyectos de C++.

Una de las funcionalidades a destacar es el Visualizador de Datos, o Data Visualizer. Ésta nos permite de manera directa procesar y mostrar datos a partir de diferentes fuentes como, por ejemplo, la Interfaz Integrada de Puerta de Enlace del Depurador, o DGI, y los puertos serie. En nuestro caso la utilizaremos para ver en tiempo real la comunicación con el GPS que se produce a través de la UART del microcontrolador.

II. ATMEL START

A. RESUMEN

Además del IDE, también utilizaremos la herramienta Atmel Start. Esta herramienta se puede utilizar tanto de manera autónoma como incluida dentro de Microchip Studio como otra de sus funcionalidades. Con ella, podremos preconfigurar nuestro microcontrolador y la placa de desarrollo y apoyarnos en las diferentes librerías que ya están creadas.

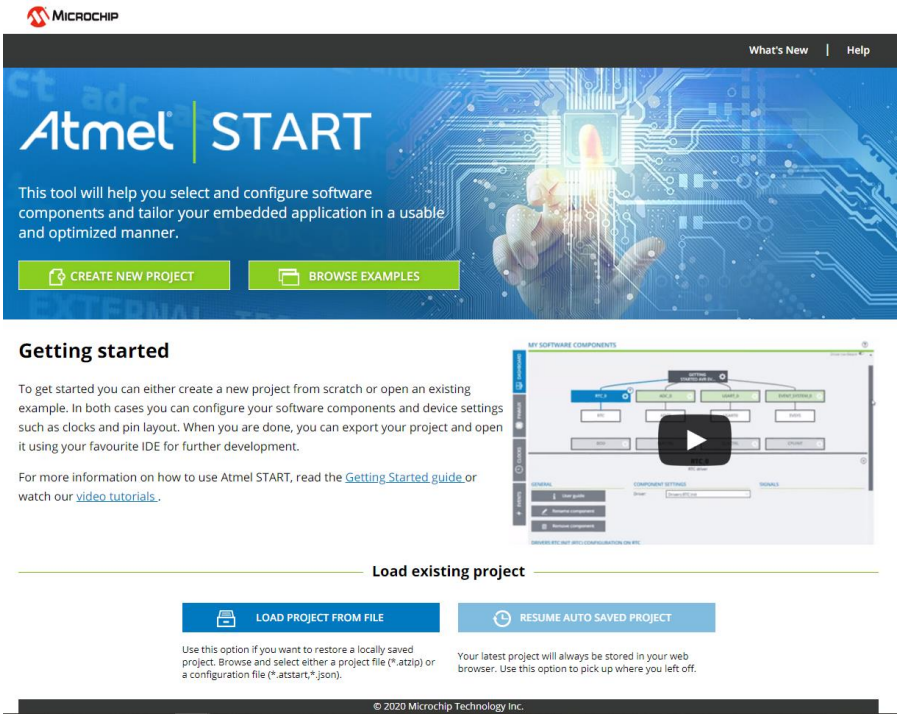


Figura 20. Pantalla de presentación de proyecto de Atmel START

Para empezar a configurar nuestra placa de desarrollo y el SAM, deberemos crear un nuevo proyecto y elegir entre las diferentes opciones la SAME54 Xplained PRO. Posteriormente pulsaremos en Create New Project.

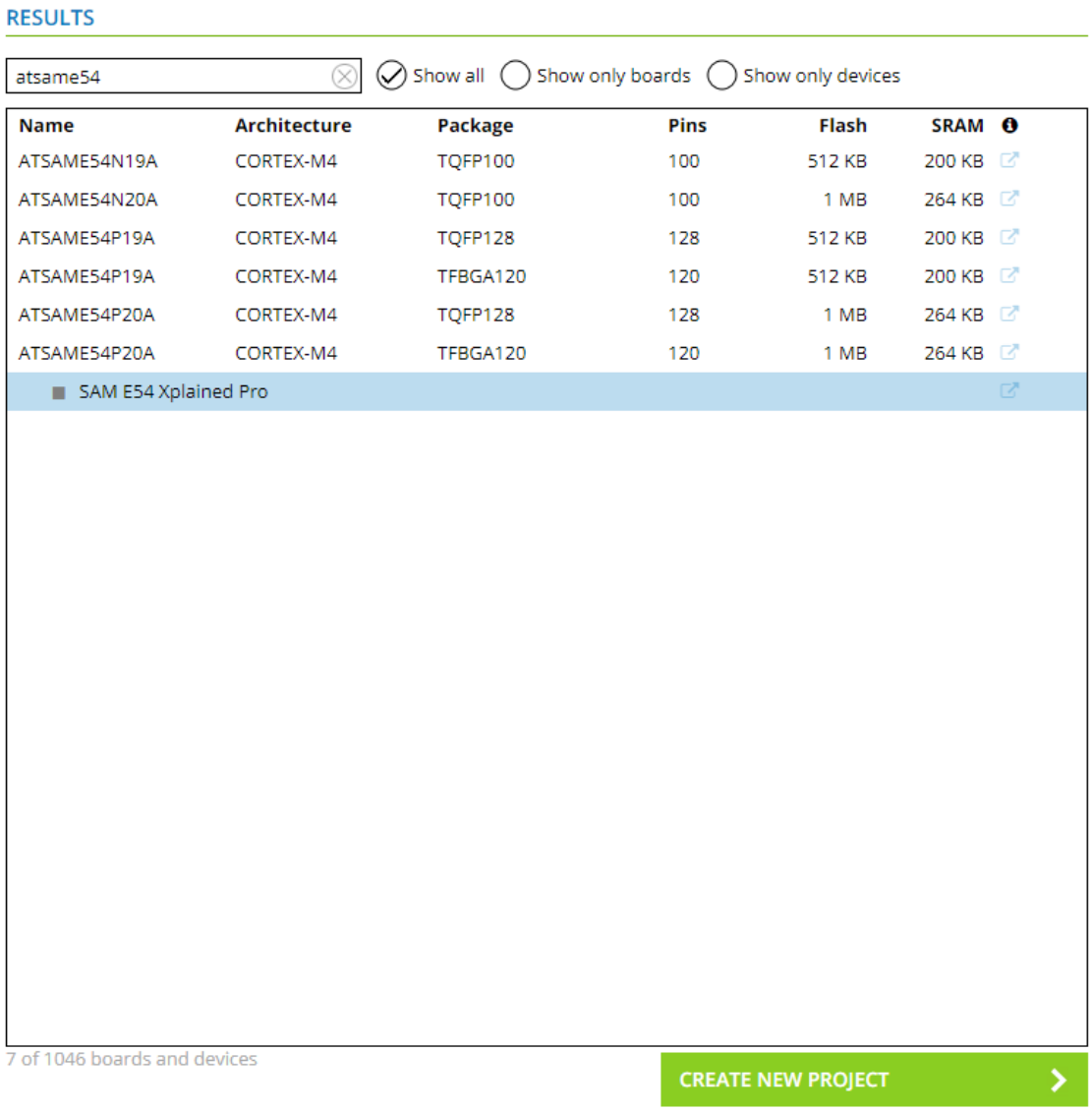


Figura 21. Selección de dispositivo de proyecto de Atmel START

En nuestro caso, vamos a partir del proyecto creado para este trabajo. Para ello, pulsaremos en Load Project from File de la primera página y elegiremos el proyecto de Atmel Start creado.

A partir de aquí, tendremos tres pantallas disponibles:

- Dashboard: La pantalla principal del proyecto, donde veremos todos los drivers que hemos ido incluyendo, así como la configuración de cada uno de ellos si pulsamos en ellos. Si pulsamos, por ejemplo, en el Timer_Base_10ms, podremos ver la configuración de este más abajo.

Para añadir cada periférico, tendremos que pulsar en Add Software Components. Veremos dos tipos de componentes:

- Middleware, con las librerías ya creadas para funciones y dispositivos externos específicos. En nuestro caso
- Drivers, para una configuración básica de cada periférico del microcontrolador, pero de menor nivel que las incluidas en Middleware.

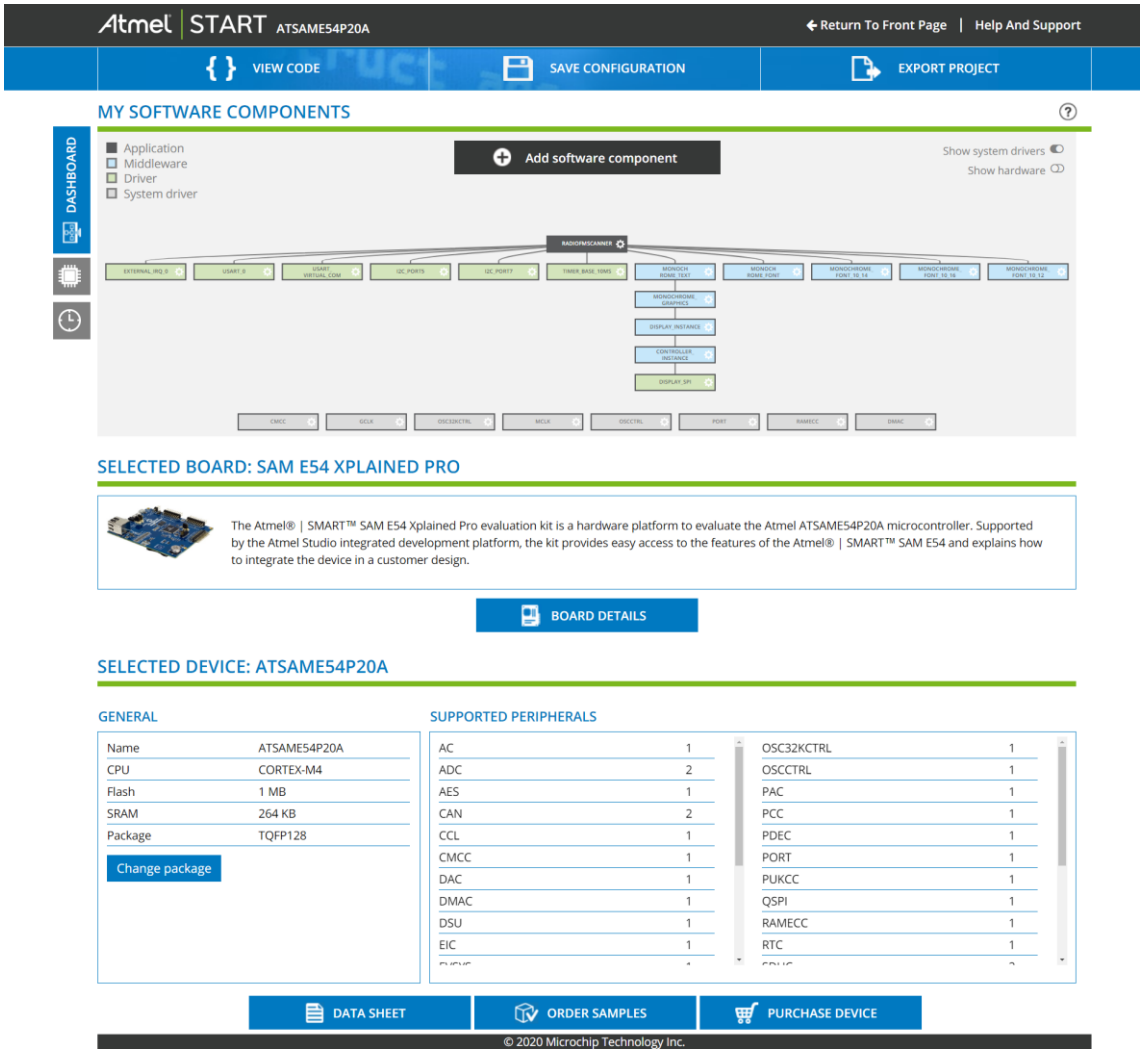


Figura 22. Pantalla de proyecto de Atmel START

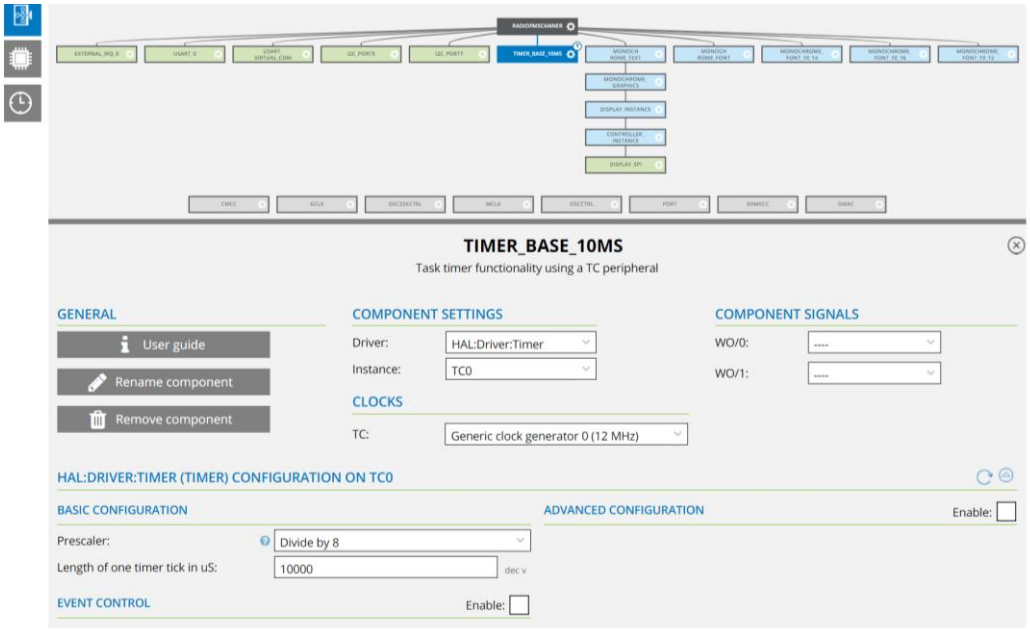


Figura 23. Configuración de un controlador dentro de la pantalla de proyecto de Atmel START

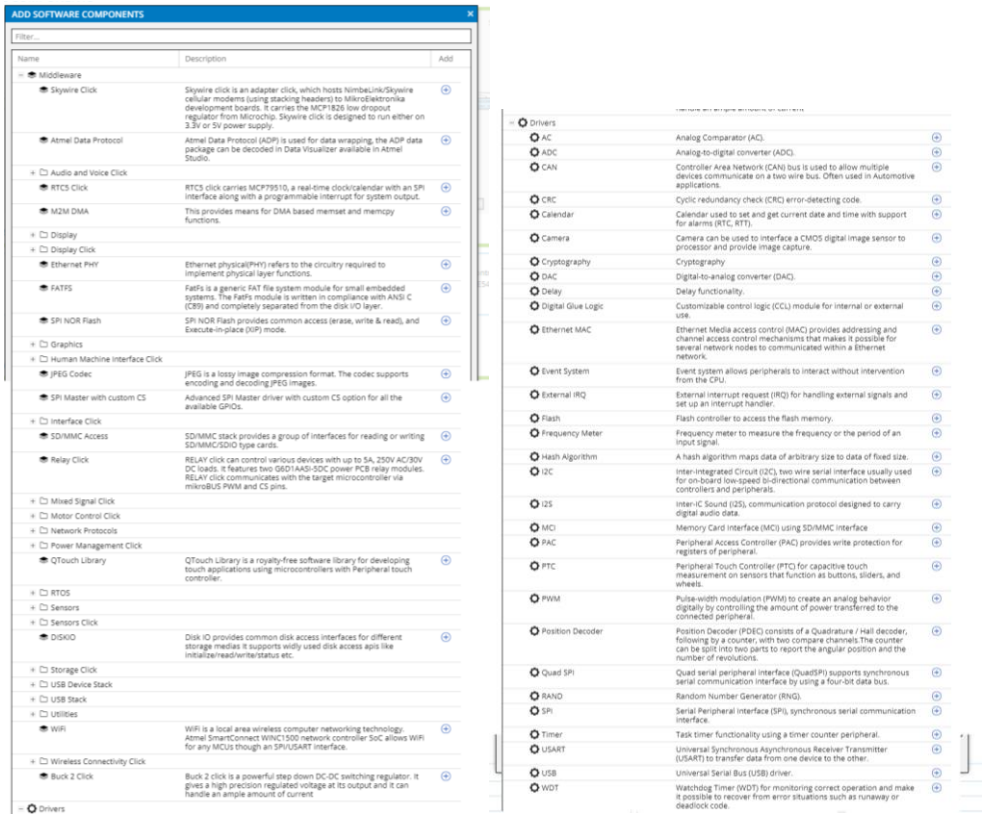


Figura 24. Periféricos y controladores disponibles para proyecto Atmel START

- PinMux: En esta pantalla se encuentran todos los pines del SAM y de la configuración de cada uno de ellos. En la primera imagen se pueden ver todos los pines que utilizaremos, recortados en dos columnas. En la segunda imagen, podemos ver la configuración de uno de estos pines.

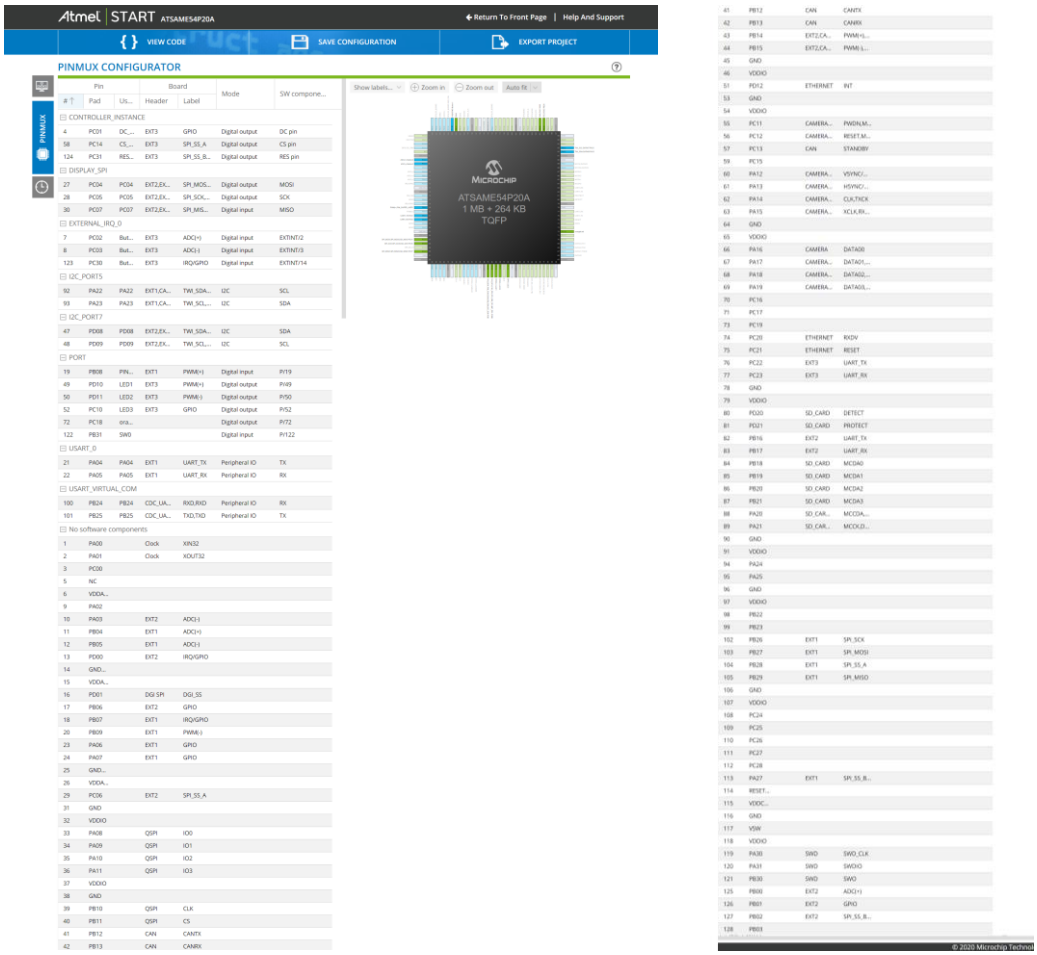


Figura 25. Pines disponibles para el microcontrolador dentro del proyecto Atmel START

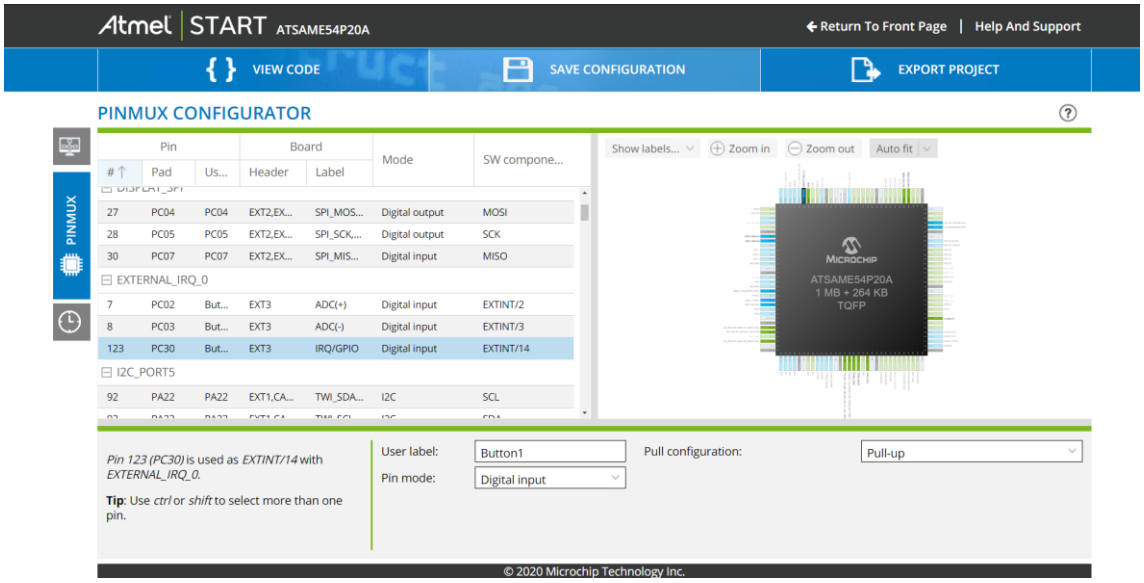


Figura 26. Configuración del pin seleccionado dentro del proyecto Atmel START

- Clocks: En esta última ventana, podemos comprobar los relojes de cada uno de los periféricos, enlazarlos y modificar las opciones de estos al pulsar en el engranaje de opciones. Podemos ver que los relojes están divididos en tres secciones:
    - Oscillators u osciladores, donde tendremos que elegir cuál será nuestro oscilador principal, si uno externo, uno interno o la combinación de uno de estos con el boost del Digital Phase Locked Loop incluido en el micro.
    - Sources o fuentes, donde generaremos la señal de reloj a partir del oscilador principal. Podemos tener hasta 12 generadores de reloj más uno de tiempo real, y enlazarlos como veamos a cada uno de nuestros periféricos.
    - CPU + Components o componentes, donde veremos todos los componentes que hayamos añadido, además de la CPU.

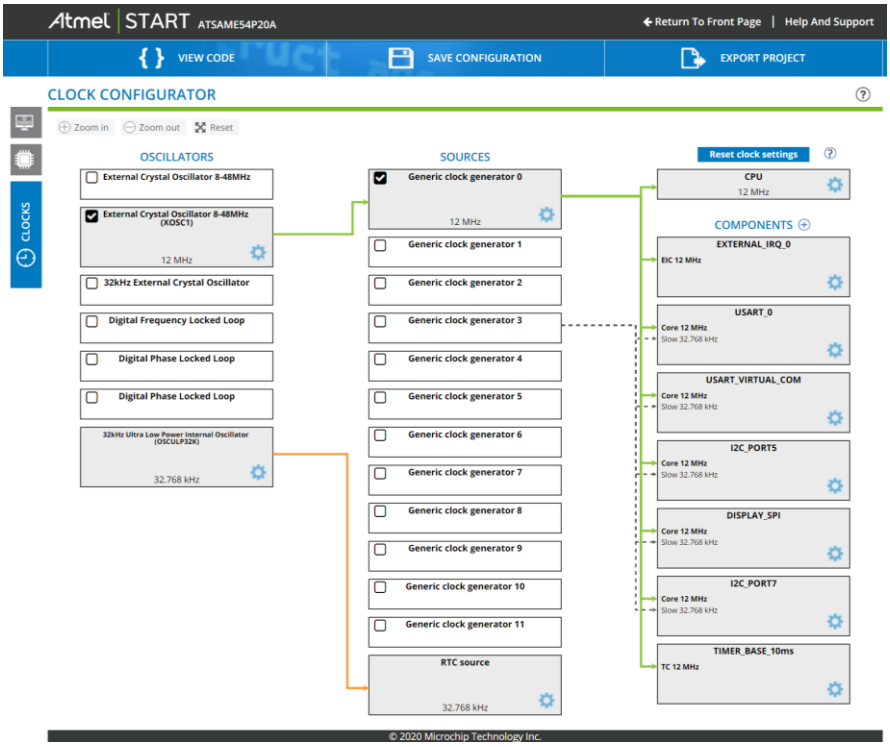


Figura 27. Configuración de los relojes del proyecto Atmel START

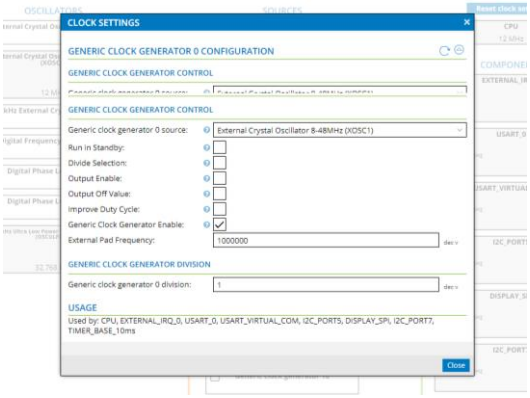


Figura 28. Configuración del generador de reloj 0 del proyecto Atmel START

B. CONFIGURACIÓN PARA EL PROYECTO

1. PERIFÉRICOS Y MIDDLEWARE

A continuación, se enumeran cada uno de los componentes que se han añadido en al proyecto de configuración del microcontrolador, así como los ajustes de cada uno de ellos y la razón de haberlo hecho así.

1.1. INTERRUPCIONES EXTERNAS: EXTERNAL_IRQ_0

Necesitamos habilitar las interrupciones externas para tener un control en tiempo real de los botones incluidos en la Xplained PRO OLED1. Estos se encuentran en los pines PC02, PC03 y PC30 del microcontrolador. Utilizaremos la frecuencia base de nuestro oscilador ya que no necesitamos más.

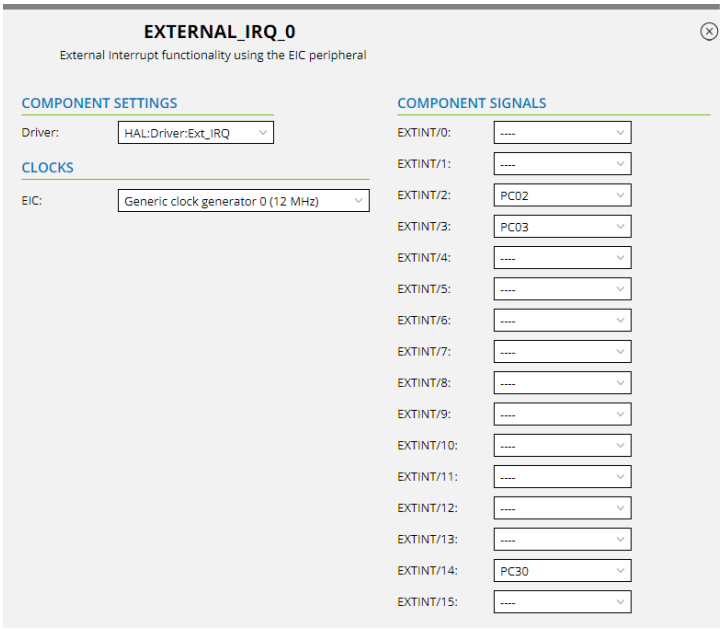


Figura 29. Configuración básica del controlador de interrupciones externas del proyecto Atmel START

Configuraremos ahora tres opciones principales para las tres interrupciones:

- Filtro: Para leer varios valores y evitar picos en la lectura.
- Anti-rebote: Para filtrar y evitar ruidos en la lectura también.
- Detección de ambos flancos: Para tener una interrupción tanto en el flanco de subida como de bajada, y así poder saber tanto cuando pulsamos como cuando dejamos de pulsar.

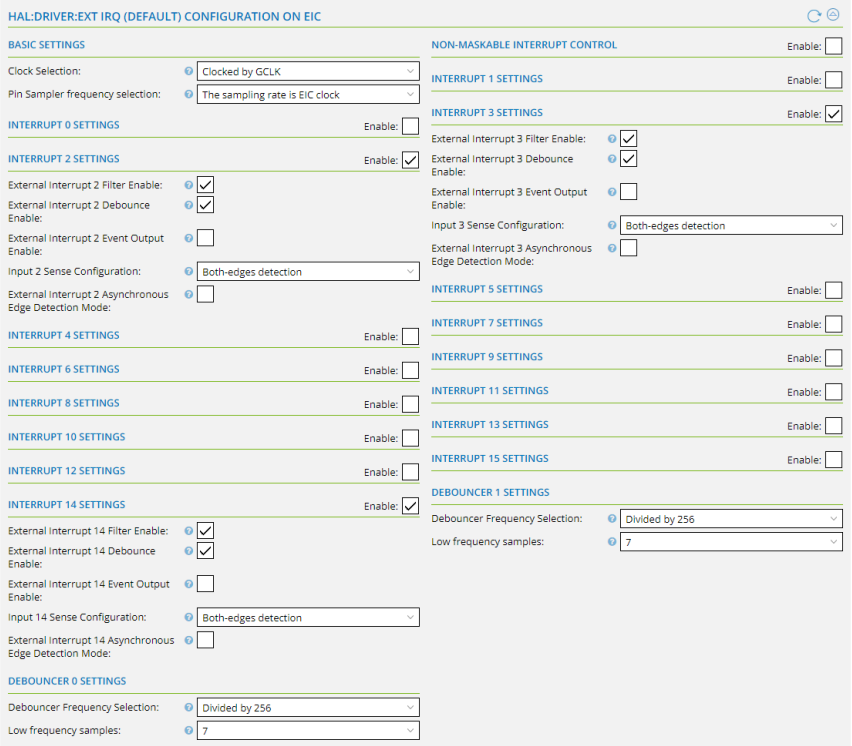


Figura 30. Configuración avanzada del controlador de interrupciones externas del proyecto Atmel START

1.2. PUERTO U(S)ART 0: USART_0

Configuraremos este puerto ya que es necesario para comunicarnos con el GPS. Es necesario el puerto de comunicaciones que vamos a usar, así como el modo UART de este y el tipo de librería a usar. También configuraremos el reloj a utilizar (de nuevo el oscilador base) y donde irán las señales de comunicación.

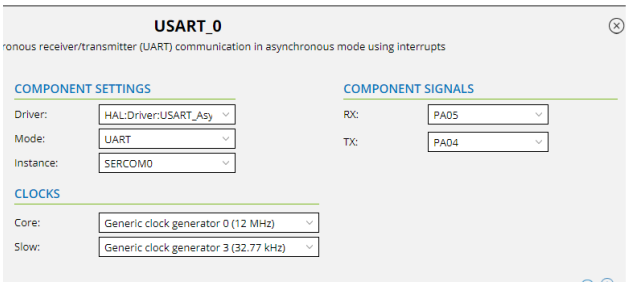


Figura 31. Configuración básica del controlador de bus UART del proyecto Atmel START

Ahora tendremos que configurar los ajustes específicos del bus UART. En este caso, configuraremos 8 como tamaño de caracteres, 1 como bit de parada y 9600 como

baud rate. También tenemos que asegurarnos de que tanto buffer de recepción como buffer de transmisión están activos.

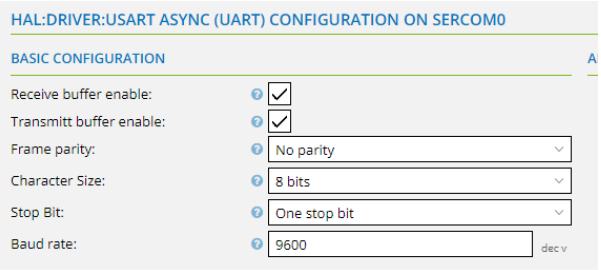


Figura 32. Configuración avanzada del controlador de bus UART del proyecto Atmel START

1.3. PUERTO U(S)ART PARA COMUNICACIÓN VIRTUAL: USART_VIRTUAL_COM

Para ver en tiempo real los datos que el GPS transmite a nuestro micro, necesitamos activar esta interfaz UART. De esa manera, podremos hacer un eco de estos datos y que pasan del micro al Data Visualizer de Microchip Studio. Esta funcionalidad ha sido necesaria para la creación del código, pero una vez establecidos los formatos de transmisión del GPS, se pueden desactivar.

Lo configuraremos igual que para el otro puerto UART, con la diferencia de que el Baud Rate será de 115200 baudios. Al utilizar el cable USB, que cuenta con aislamiento propio, podemos utilizar esta velocidad sin miedo a la perdida de datos o a la aparición de ruido.

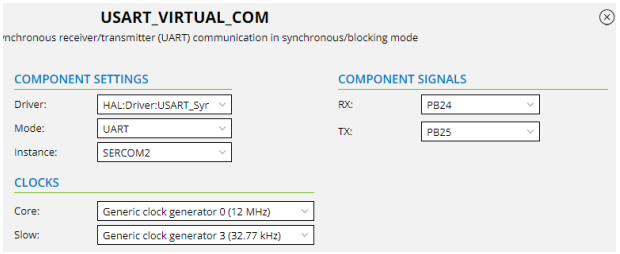


Figura 33. Configuración básica del controlador de bus UART Virtual del proyecto Atmel START

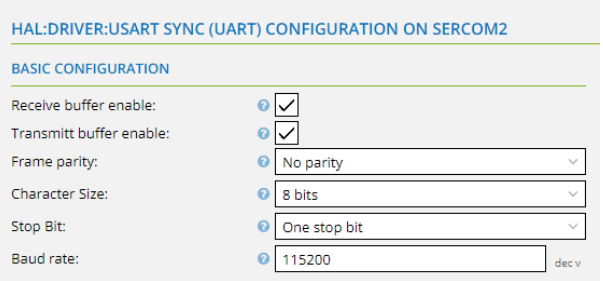


Figura 34. Configuración avanzada del controlador de bus UART Virtual del proyecto Atmel START

1.4. COMUNICACIÓN I2C PUERTO 5: I2C_PORT5

Necesitaremos este puerto para comunicarnos tanto con el módulo radio como con la segunda memoria EEPROM presentada. Configuraremos el puerto de comunicaciones para que utilice el SERCOM5, en modo I2C maestro y con el controlador estándar. También utilizaremos la frecuencia base de nuestro oscilador, así como las señales de Reloj y Datos del I2C. También configuraremos debajo la velocidad del bus I2C a 400 kHz.

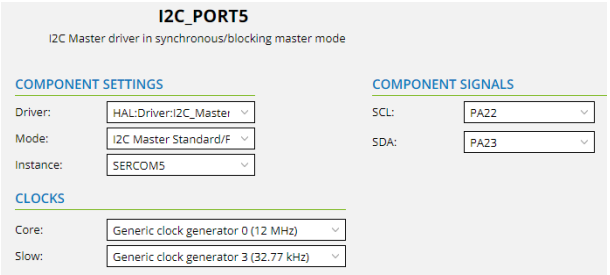


Figura 35. Configuración básica del controlador de bus I2C utilizando SERCOM5 del proyecto Atmel START

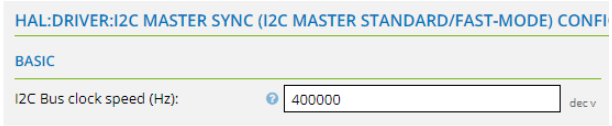


Figura 36. Configuración avanzada del controlador de bus I2C utilizando SERCOM5 del proyecto Atmel START

1.5. COMUNICACIÓN I2C PUERTO 7: I2C_PORT7

Este puerto esta destinado a la comunicación con la primera memoria EEPROM presentada. Al ser la memoria que ya no se utiliza, se puede modificar el código para no seguir escribiendo en esta memoria y podría suprimirse de la configuración del proyecto.

La configuración es la misma que para el anterior puerto, con la diferencia de que utilizaremos el puerto SERCOM7 y la velocidad del bus será 100kHz.

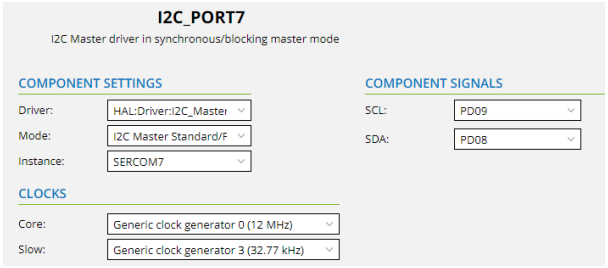


Figura 37. Configuración básica del controlador de bus I2C utilizando SERCOM7 del proyecto Atmel START

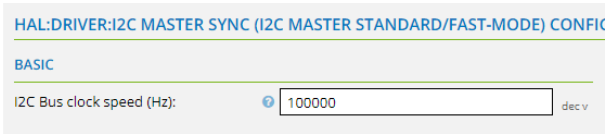


Figura 38. Configuración avanzada del controlador de bus I2C utilizando SERCOM7 del proyecto Atmel START

1.6. TEMPORIZADOR DE 10 MS: TIMER_BASE_10MS

Como necesitamos temporizar diferentes operaciones, es necesario utilizar al menos un temporizador. Aunque lo vamos a usar con diferentes tiempos, como estos tiempos son muy diferentes entre ellos y en la mayor parte del proceso no trabajan a la vez, con uno tenemos suficiente.

Configuraremos el temporizador como Temporizador Contador (TC) y utilizaremos la frecuencia base de nuestro oscilador. No utilizaremos ninguna señal externa en él. Como necesitamos un temporizador rápido para algunas tareas, lo configuraremos para que cuente hasta 10.000 µs, o lo que es lo mismo, 10 ms.

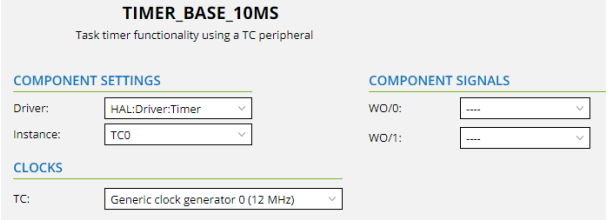


Figura 39. Configuración básica del temporizador de 10 ms del proyecto Atmel START

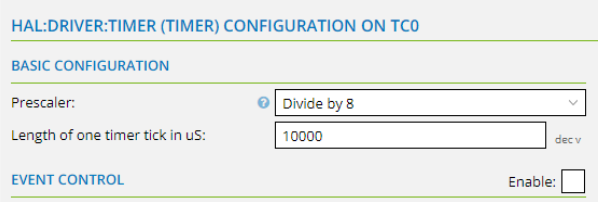


Figura 40. Configuración avanzada del temporizador de 10 ms del proyecto Atmel START

1.7. SPI PARA DISPLAY OLED1: DISPLAY_SPI

Para comunicarnos con el display OLED1 de la familia Xplained PRO, es necesario preconfigurar el SPI. En este caso, utilizaremos el puerto de comunicaciones SERCOM6 y de nuevo la frecuencia base del oscilador. Configuraremos los pines como nos recomienda el manual de la OLED1 para utilizar señales MISO, MOSI y Clock. Pondremos la velocidad del bus a 1 MHz. Tenemos activado el buffer de recepción, pero realmente no hace falta porque el display no comunica hacia el maestro.

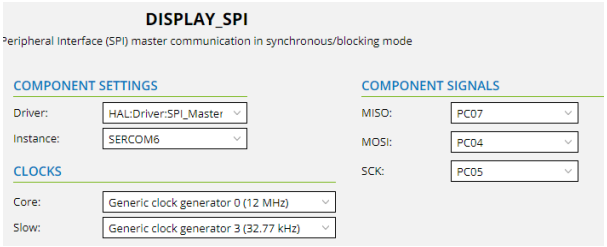


Figura 41. Configuración básica del bus SPI del proyecto Atmel START

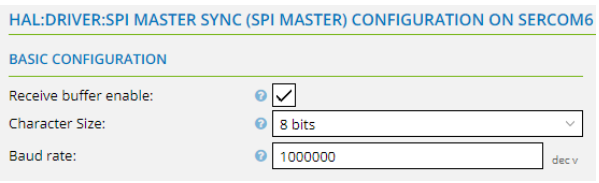


Figura 42. Configuración básica del bus SPI del proyecto Atmel START

1.8. CONTROLADOR PARA OLED1: CONTROLLER_INSTANCE + DISPLAY_INSTANCE + MONOCHROME_GRAPHICS

Para utilizar correctamente el display, debemos configurar el controlador específico. En nuestro caso, el nuestro es el SSD1306. Los pines deben ser configurados también, pero son heredados del SPI que hemos configurado anteriormente.

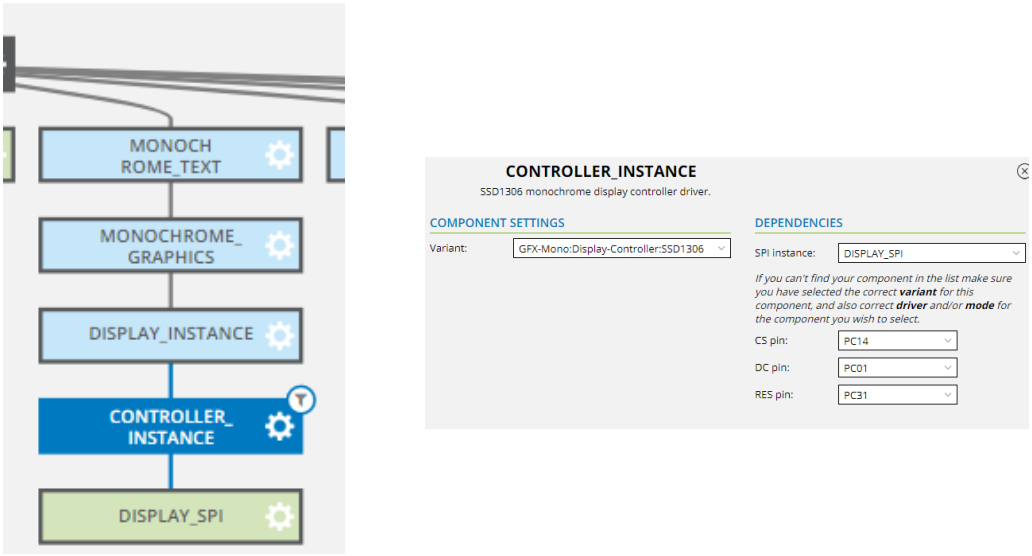


Figura 43. Configuración del controlador del display del proyecto Atmel START

También debemos configurar la instancia del display. En nuestro caso utilizaremos la librería para UG2832HSWEG04.

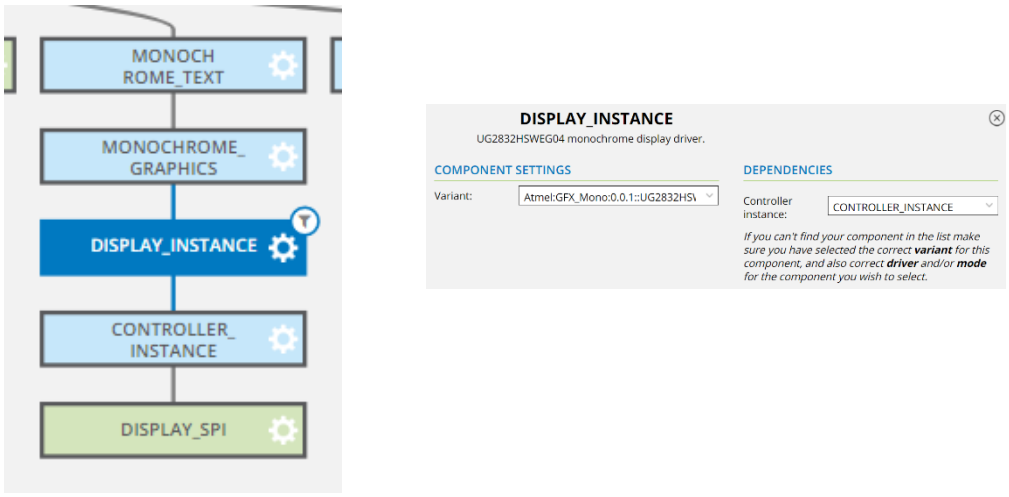


Figura 44. Configuración de la instancia del display del proyecto Atmel START

Por último, debemos configurar el stack de gráficos.

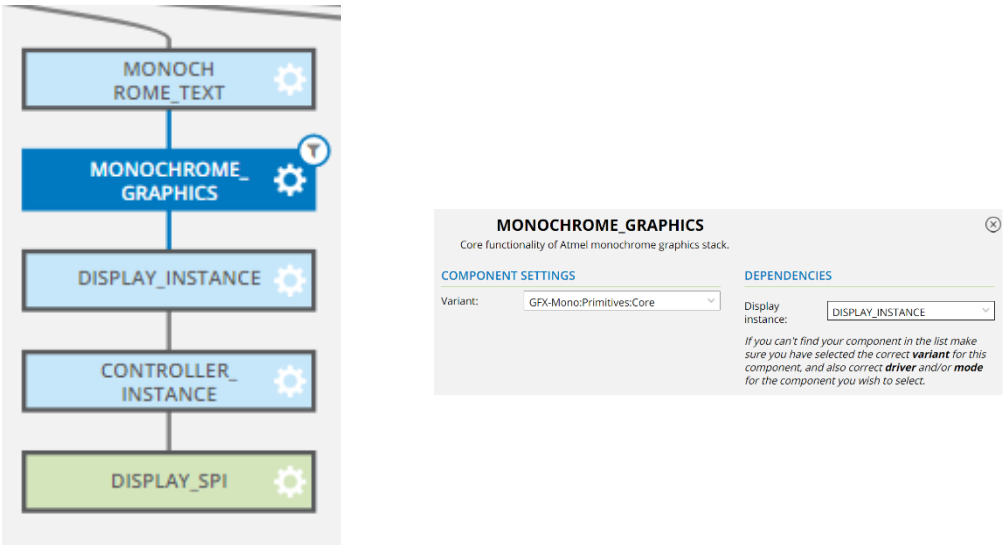


Figura 45. Configuración de gráficos del display del proyecto Atmel START

1.9. FUENTES PARA OLED1: MONOCHROME

Por último, debemos configurar las fuentes monocolores que utiliza el display. En este caso, es sólo ir añadiendo una por una desde el gestor de middleware.

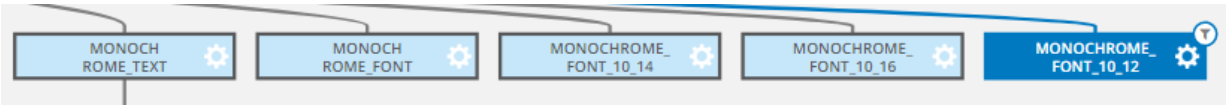


Figura 46. Fuentes añadidas para el display para el proyecto Atmel START

No necesitamos modificar ninguno de los valores predeterminados de ajuste de estas librerías.

2. CONFIGURACIÓN DE LOS PINES

En principio no hace falta modificar los pines. Al ir añadiendo las librerías, estos pines se han ido configurado automáticamente según las necesidades de estas.

3. CONFIGURACIÓN DEL OSCILADOR

Como vamos a usar el oscilador incluido en la propia placa para tener una señal de reloj limpia, seleccionamos el XOSC1 en la columna de Osciladores en la pestaña de Clocks. Debemos configurarlo con su frecuencia, 12 MHz, asegurarnos de que está habilitado y habilitar la lectura del cristal conectado a XIN/XOUT.

Nota: Inicialmente se iba a usar el RTC interno para controlar la hora y fecha. Sin embargo, como no podemos tener la placa alimentada de forma ininterrumpida, este RTC no va a funcionar correctamente. Utilizaremos los datos dados por el GPS para fijar fecha y hora en cada inicio.

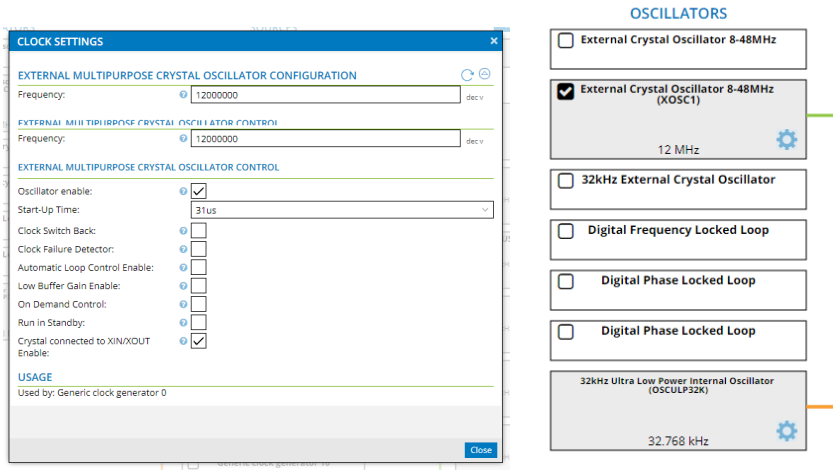


Figura 47. Configuración del oscilador para el proyecto Atmel START

Debemos generar un reloj a partir de esta señal. Como 12 MHz es suficiente para controlar todos los dispositivos, generaremos el reloj a partir de esta señal utilizando el primer generador.

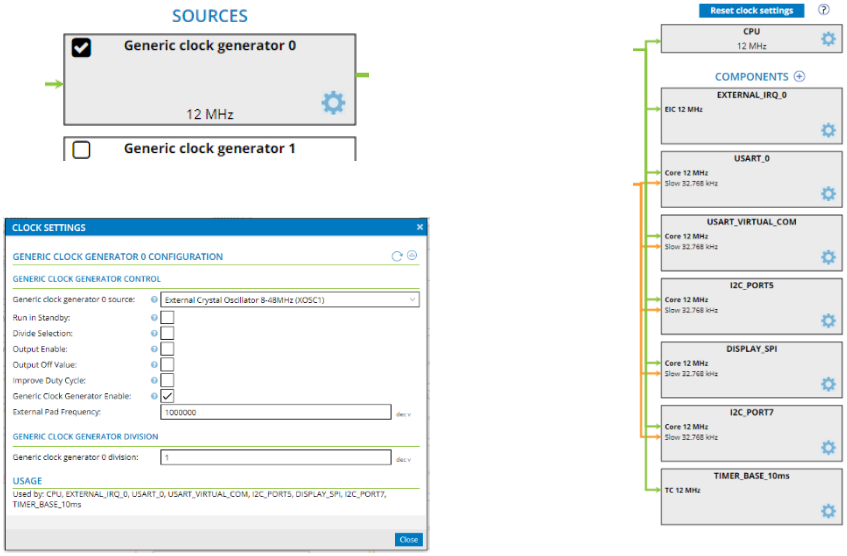


Figura 48. Configuración del generador de reloj para proyecto Atmel START

6. SOLUCIÓN DESARROLLADA

El código al completo se encuentra disponible en el Anejo correspondiente. En ocasiones se harán comentarios de algunas de las funciones y/o variables desarrolladas para el sistema. En todas ellas, se identifica el archivo en el que se encuentran.

A. DIAGRAMA DE FLUJO GENERAL DEL PROGRAMA PRINCIPAL

En el siguiente diagrama podemos ver el flujo general de trabajo de nuestro sistema.

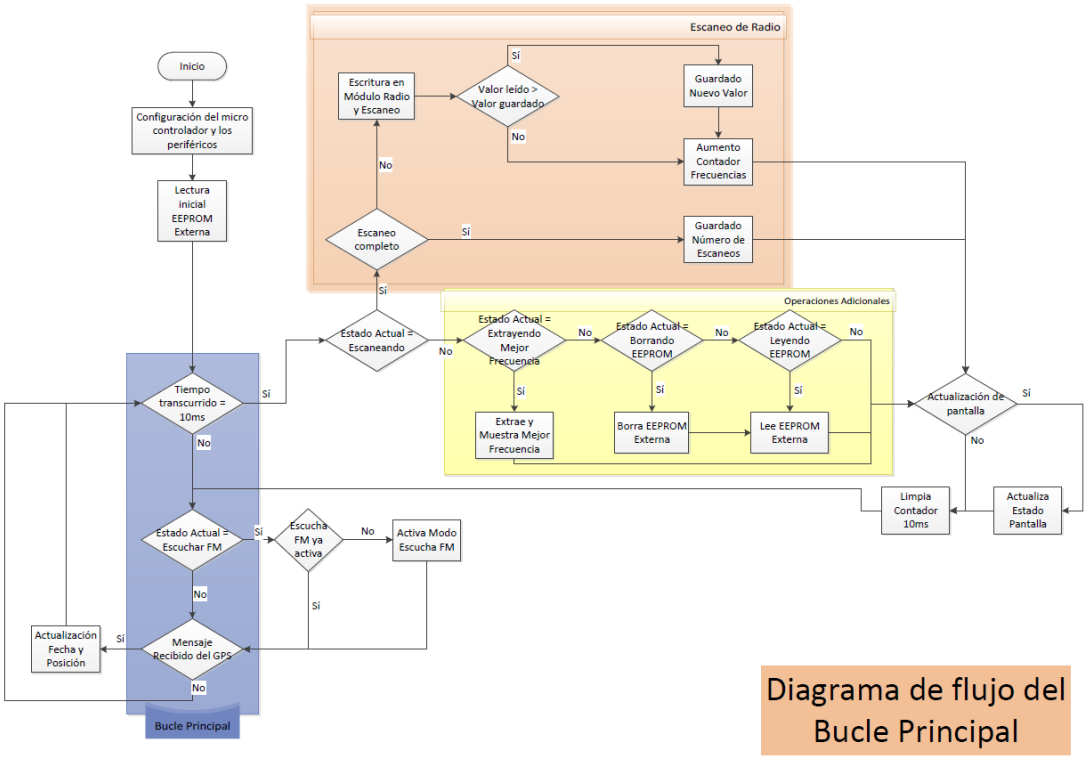


Figura 49. Diagrama de flujo general del funcionamiento del bucle principal

Como hemos comentado anteriormente, nos basamos en un temporizador base para controlar las operaciones. El control de estas operaciones es lo que llamaremos "Bucle principal".

El otro bloque que debemos comentar es el de escaneo de radio, variando entre tres posibles operativas en función de si estamos escribiendo o leyendo en el módulo radio, o si hemos completado el escaneo de todo el espectro de frecuencias FM.

Además, se muestran operaciones a mayores que nos permiten hacernos una idea del funcionamiento del sistema. Entre estas operaciones están la recepción de un mensaje completo del GPS, la activación del modo "Escucha FM" o las operaciones adicionales de la pantalla.

B. TEMPORIZADORES CON TAREAS

Para el funcionamiento correcto del programa y trabajar como un sistema RTOS, debemos asignar tiempos a cada tarea. O, mejor dicho, debemos asegurar que cada cierto tiempo se ejecuten las operaciones requeridas.

A partir del temporizador base de 10 ms que hemos configurado en el Atmel Start, vamos a configurar los siguientes temporizadores:

- 10 ms: Como nuestro temporizador base. Lo utilizaremos para las operaciones del bucle principal, entrando y ejecutando una acción cada vez que entra en él. Por esto necesitamos que se repita continuamente. La principal razón de utilizar este tiempo es que es el tiempo máximo que necesita nuestro módulo radio entre mensajes. De esta manera, cuando estemos en modo "escaneo", haremos 15 escrituras y lecturas en una frecuencia dada, una en cada tic del temporizador. Después de esto, en las siguientes 15 iteraciones repetiremos la operación, pero con la siguiente frecuencia, repitiéndose este método para frecuencia. De esta manera, escanearemos todo el espectro radio en algo más de 3 minutos.
- 1 s: Utilizado en versiones anteriores para la lectura de los botones incluidos en la placa, cuando no teníamos la pantalla conectada y, por tanto, tampoco teníamos los botones de esta. Cada segundo comprobaba el estado de la señal de estos botones y realizaba las siguientes operaciones en función del tiempo pulsado y el conteo de tics:
    - Si partíamos del estado "esperando", el contar 2 tics (2 segundos) provocaba que entráramos en modo "escaneo".
    - Si partíamos del estado "esperando", y no teníamos ningún pulso después de 60 tics (60 segundos), apagaríamos la placa directamente.
    - Si partíamos del estado "escaneando", y contábamos 2 tics (2 segundos), el programa pasaba a estado esperando.

  Este temporizador presentaba el problema de que el conteo de tics, aunque correctos, no representaba la realidad, ya que podríamos encontrarnos el caso en el que, justo después de leer un tic, pulsábamos el botón, añadiendo casi 1 segundo al conteo real. Al tener las interrupciones de los botones nos podemos ahorrar este temporizador y partir directamente de las interrupciones.

- 10 s para la salida del menú: Cuando pulsamos cualquier de los botones, entramos en el menú de acciones que podemos lanzar al microcontrolador. Este temporizador controla la salida del menú cuando estamos estos segundos sin pulsar ningún botón. De esta manera, volveremos a la operación actual pasado ese tiempo.
  El disparador para empezar a contar es que detectemos como uno de los botones se deja de pulsar. A su vez, el disparador para terminarlo es, por un lado, que se acabe de contar y tengamos que volver a la acción, y por otro, que pulsemos otro botón, de tal manera que tendremos que reiniciar el temporizador.
- 20 s para apagar la pantalla: Las pantallas OLED sufren mucho desgaste cuando se deja una imagen estática, de tal manera que queda marcada con la imagen que estuviera puesta. Para evitar esto, cuando no se está usando la pantalla, la apagamos totalmente. Esto se hace a partir de este temporizador de inactividad de 20 segundos.
  El disparador para empezar a contar con este temporizador es volver a mostrar la acción en pantalla (ya sea por la selección de una de las opciones del menú o por el fin del anterior temporizador). Por el contrario, el disparador para terminarlo es que, pulsemos uno de los botones, o que, por el contrario, lleguemos a contar los 20 segundos y apaguemos la pantalla.
- 1 s para los gráficos: En algunas de las operaciones, existe un reloj dinámico que cambia su aguja cada segundo. Este temporizador es el encargado de controlar este cambio.
  El disparador para iniciarlo es principalmente que activemos una de las acciones que tienen este reloj, o que volvamos a ella después de pulsar algún botón. Como es cíclico, no hace falta reiniciarlo en cada uno de los movimientos de aguja. Para terminarlo, basta con pulsar uno de los botones, como se acaba de comentar.

C. COMUNICACIÓN I2C CON EL MÓDULO RADIO Y ESCANEO DE FRECUENCIAS

Como se ha comentado anteriormente en la configuración de proyecto de Atmel Start, utilizaremos el bus I2C a una velocidad de 400 kHz. Tras inicializarlo con las librerías del software, pasaremos a comunicarnos con él.

Este dispositivo tiene un montón de opciones para configurar, entre las que destacaremos:

- Silenciar la salida de audio del dispositivo.
- Modo de operación, permitiendo la búsqueda automática de frecuencias que cumplan un determinado nivel de señal.
- Límites de la banda, distinguiendo entre la banda japonesa y las bandas europeo-americana.
- Referencia del PLL o, dicho de otra manera, señal de reloj a utilizar.

Para llevar a cabo la operación de medida de la señal, lo que debemos hacer es mandar los datos de configuración, en especial la de la señal de reloj, y más específicamente, la banda de frecuencia a escanear; y posteriormente recibir la respuesta de este. Para ello, nos apoyaremos en el temporizador de 10 ms nombrado anteriormente.

Se ha propuesto hacer 15 lecturas de potencia de señal para cada frecuencia. En total son 206 frecuencias distintas, empezando por la 87.5 y acabando en la 108.0, con incrementos de 0.1. Con esto tendríamos un escaneo completo de todo el espectro en 3,09 segundos. A nivel operativo, se traduce en mandar los 5 bytes de configuración del módulo y recibir los 5 bytes de respuesta, entre cuyos datos estará la potencia de la señal de radio presente, si hay. Con las 15 lecturas, extraeremos un valor medio de señal. Como el dato que obtendremos será uno de tipo flotante y necesitamos guardarlo en memoria, lo multiplicaremos por 10 para después guardar su parte entera en un vector de "potencias medidas". Así nos ahorramos realizar complejas operaciones para guardar variables tipo flotantes en memorias que guardan variables tipo entero de 8 bytes. Guardaremos este valor en la memoria EEPROM externa y repetiremos este proceso para cada frecuencia de radio.

En siguientes versiones, se debería considerar apoyarse en la escritura de páginas directamente, de tal manera que sólo necesitaríamos una operación por cada 16 bytes en el caso de la AT24MAC402 y por cada 64 bytes en el caso de la AT24C256.

D. MENÚ DE OPCIONES EN PANTALLA CONTROLADO POR PULSADORES

Los objetivos de nuestra pantalla principalmente son dos:

- Mostrar la acción que está realizando el sistema en ese momento, lo que llamaremos "Estado del sistema".
- Cambiar la acción a otra mediante la interfaz de botones y presentándolas en pantalla.

I. ESTADO DEL SISTEMA

Para mostrar el estado del sistema, primero debemos definir las acciones y/o estados que puede realizar la pantalla. En nuestro caso, como acciones que está realizando el sistema hemos definido las siguientes como una enumeración, convertida a una variable personalizada STATE dentro del archivo "TimerBase.h"

- IDLE es el estado en el que apagamos la pantalla tras 60 segundos sin ejecutar ninguna acción. Es necesario apagar la pantalla porque las OLED tienen el problema de que, si mantenemos durante mucho tiempo una imagen, al final se acaba quemando con la imagen que tengamos.
- WAITING es el estado base, en el que nos encontramos si no estamos realizando ninguna otra y el que iniciamos cuando el sistema se enciende.
- SCANNING es el estado en el que pasamos a escanear todo el rango de frecuencias moduladas. Se mantendrá aquí si no seleccionamos otra.
- DELETING_EEPROM es la acción de reiniciar todos los datos en la memoria externa. No se encuentra accesible por métodos convencionales.
- READING_EEPROM es la acción de leer de manera interna, cuando estamos depurando, toda la EEPROM externa. Tampoco se encuentra accesible por métodos convencionales porque no podemos mostrar de manera adecuada estos datos en nuestra pantalla.
- EXTRACTING_BEST_FREQ es el estado en el que empezamos a buscar en memoria la mejor frecuencia para configurar nuestro emisor radio. Una vez termine, pasará al siguiente estado.
- SHOWING_BEST_FREQ es al estado al que llegamos después de buscar la mejor frecuencia. Como es entendible, mostraremos en pantalla la frecuencia más correcta.
- LISTENING_FM_RADIO es un estado adicional en el que podemos usar nuestro módulo radio como emisor radio a través del Jack de Audio.
- SHOW_EEPROM es otro estado diferente en el que vamos mostrando la memoria EEPROM poco a poco, pero reservada para el futuro.
- TEST_MODE es una acción reservada para depuración y que aún no está disponible.
- ERROR es el estado al que llegamos si detectamos un malfuncionamiento del sistema.

II. MENÚ DE ACCIONES

En nuestro menú de posibles acciones podemos ver una analogía con el estado del sistema. Repetimos la manera de definir estas acciones, con una enumeración convertida a variable personalizada STATE dentro del archivo "DisplayOled.h"

- START_SCANNING es la orden que pulsaremos para pasar a escanear la radio. La acción que hemos llamado anteriormente SCANNING.
- SHOW_BEST_FREQ es la orden que lanzaremos si queremos saber cuál es la mejor frecuencia de todas las escaneadas. Pasaremos al estado EXTRACTING_BEST_FREQ y luego a SHOWING_BEST_FREQ.
- LISTEN_FM es la orden para pasar a escuchar la radio en frecuencia FM, LISTENING_FM.
- STOP_OPERATION es la orden para cancelar la acción actual. Pasaríamos al estado WAITING.
- BACK es la orden para salir del menú y no modificar nada.
- NONE es una orden ficticia para poder diferenciar cuando tenemos lanzado el menú y cuando no.

III. CAMBIO DE ACCIONES

Para pasar de una acción a un menú y viceversa, se ha definido una función que permite, sabiendo el estado del programa y la selección del menú, mostrar uno u otro y seleccionar entre todos los posibles estados. Esa función es `RefreshDisplay` cuya definición se encuentra en "DisplayOled.c"

Esta función recibe tanto el estado actual del sistema, como la selección dentro del menú. Siempre va a tener prioridad el menú sobre el sistema. Dicho de otra manera, al entrar en el menú, estaremos en una selección de menú y, por lo tanto, no tendría sentido mostrar la acción actual del sistema. Aquí es donde cobra sentido dentro del menú la orden NONE. Si no hemos entrado en el menú, el estado del menú será NONE. En ese caso es cuando mostraremos la acción actual del sistema.

Al ver con más atención la función, vemos como se guarda también el estado anterior, tanto del sistema como del menú. Esto es para evitar escribir 2 veces la misma opción. Si esto ocurriera, veríamos como la pantalla entra en bucle mostrando la acción y limpiando luego la pantalla completa, una detrás de otra.

Si vamos a la función para limpiar la pantalla `ClearFullDisplay`, también en "DisplayOled.C", podemos ver el esqueleto de las funciones que utiliza la librería para este controlador:

En este caso, la función se utiliza para pintar un rectángulo lleno. Los parámetros que damos son, en este orden:

- El descriptor de la pantalla que hemos definido al principio del código.
- El punto en el eje X para empezar del punto inicial del cuadrado.
- El punto en el eje Y para empezar a pintar el punto inicial del cuadrado.

- El punto en el eje X para terminar de pintar en el punto diagonal al inicial del cuadrado.
- El punto en el eje Y para terminar de pintar en el punto diagonal al inicial del cuadrado.
- El tipo de operación, que puede ser una de las 3 siguientes:
    - GFX_PIXEL_SET, para pintar los píxeles correspondientes o, dicho de otra manera, para encender los píxeles.
    - GFX_PIXEL_CLR, para borrar los píxeles correspondientes o, dicho de otra manera, para apagar los píxeles.
    - GFX_PIXEL_XOR, para invertir el estado de los píxeles. Si el píxel estaba encendido, se apagará. Si el píxel estaba apagado, se encenderá.

Además de para pintar rectángulos rellenos, tenemos funciones del mismo tipo para por ejemplo pintar rectas, círculos o directamente píxeles. Los argumentos necesarios son prácticamente los mismos para todas ellas.

Además de utilizar los gráficos para pintar letras y cuadrados, también vamos a utilizar para pintar por un lado unas flechas que marquen como se desplazan las opciones del menú (y así también diferenciamos de las acciones), un cuadrado vacío para diferenciar el estado WAITING y un reloj cuyas agujas se mueven para dar dinamismo a las opciones.

Para pintar las flechas vamos a utilizar la función `DrawArrows`, cuya definición se encuentra en "DisplayOled.c" Con ella pintaremos 4 rectas, dos horizontales y dos oblicuas.

Para el reloj, lo primero que vamos a hacer es pintar el círculo vacío. Predefinimos las coordenadas y el tamaño en "DisplayOled.h".

Luego creamos una función `InitiateDrawingClock` que inicialice los contadores necesarios, y que a su vez llame a otras dos funciones, a `DrawBaseClock` para pintar el circulo base del reloj y a `DrawNextClockTick` para pintar cada uno de los movimientos de la aguja del reloj, y en este caso concreto sólo el primer movimiento. Todas estás funciones están incluidas en "DisplayOled.c".

También lanzará la tarea del temporizador encargado de controlar el movimiento de la aguja a través de la función `LaunchTimerClock`. También en "DisplayOled.c"

IV. PULSADORES PARA SELECCIÓN

Para movernos y seleccionar entre todas las opciones disponibles utilizaremos los pulsadores incluidos. Tras inicializarlos como interrupciones, configuraremos las "callbacks" de estos.

De los tres pulsadores que hemos configurado, vamos a diferenciar entre los botones laterales y el central. Los laterales son los que van a permitir desplazarnos entre todas las opciones disponibles. El central será el que haga efectivo el cambio de orden.

Las funciones para la pulsación se encuentran en el archivo "DisplayIOs.c". Tanto la del botón izquierdo `leftButtonPressed` y como la del botón derecho `rightButtonPressed` son prácticamente iguales. La única diferencia es la dirección para desplazarnos por el menú. Si tomamos como ejemplo la función para el botón izquierdo, lo primero que hacemos es desactivar la función de apagado de la pantalla. Posteriormente, terminaremos los posibles temporizadores que hubiera asociados a los gráficos en la pantalla.

Llegados a este punto debemos diferenciar si estamos realizando una pulsación o estamos dejando de pulsar. Los pulsadores están conectados a 3,3V en su estado de reposo, por lo cual leeremos a nivel lógico un valor alto cuando no pulsamos y un valor bajo cuando pulsamos. Al pulsar, encenderemos el LED correspondiente a ese botón. Luego comprobaremos si ya estábamos en el menú o no. Si no nos encontrábamos en el menú, cambiaremos la orden del menú de NONE a la primera del menú. Si ya nos encontrábamos en el menú, reduciremos en 1 el valor de la orden (lo aumentamos en el caso del botón derecho). Al haber creado la variable con enumeración, esto nos permite pasar de uno a otro estado sólo con estos decrementos (o incrementos). El único caso que no se encuentra directamente vinculado es cuando pasamos del estado mínimo al estado máximo. Por eso añadimos una excepción para este caso. En todos los casos activaremos una bandera o flag, para que cuando salgamos de la interrupción, el bucle principal sea el encargado de comunicar con la pantalla y realizar los cambios en esta. Cuando dejamos de pulsar, reiniciamos el contador y el temporizador para salir del menú si no pulsamos ningún botón. También apagaremos el LED.

En el caso del botón central, actuaremos de manera diferente cuando pulsamos los botones. Si aún no habíamos entrado en el menú, pasaremos a mostrar la primera orden disponible. Si ya estábamos dentro del menú, entonces haremos efectiva la orden y cambiaremos la acción vinculada a esa orden. De nuevo, será el bucle principal el que ejecute los cambios en la pantalla a través del flag.

A continuación se muestran los diagramas de flujo utilizados para los 3 botones.

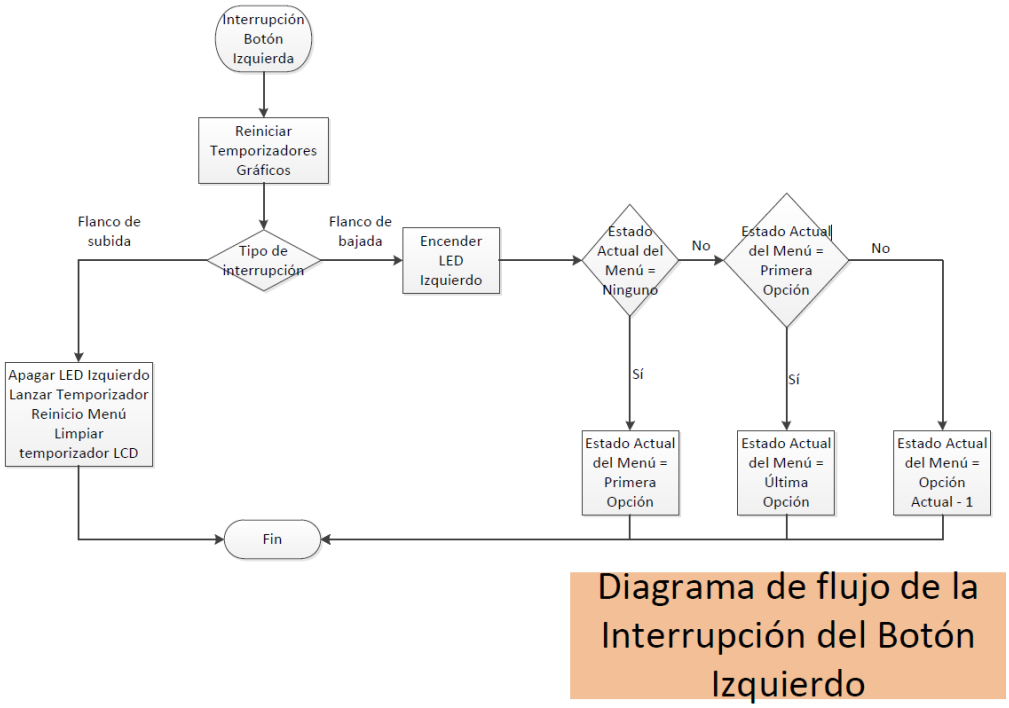


Figura 50. Diagrama de flujo de la Interrupción del Botón Izquierdo

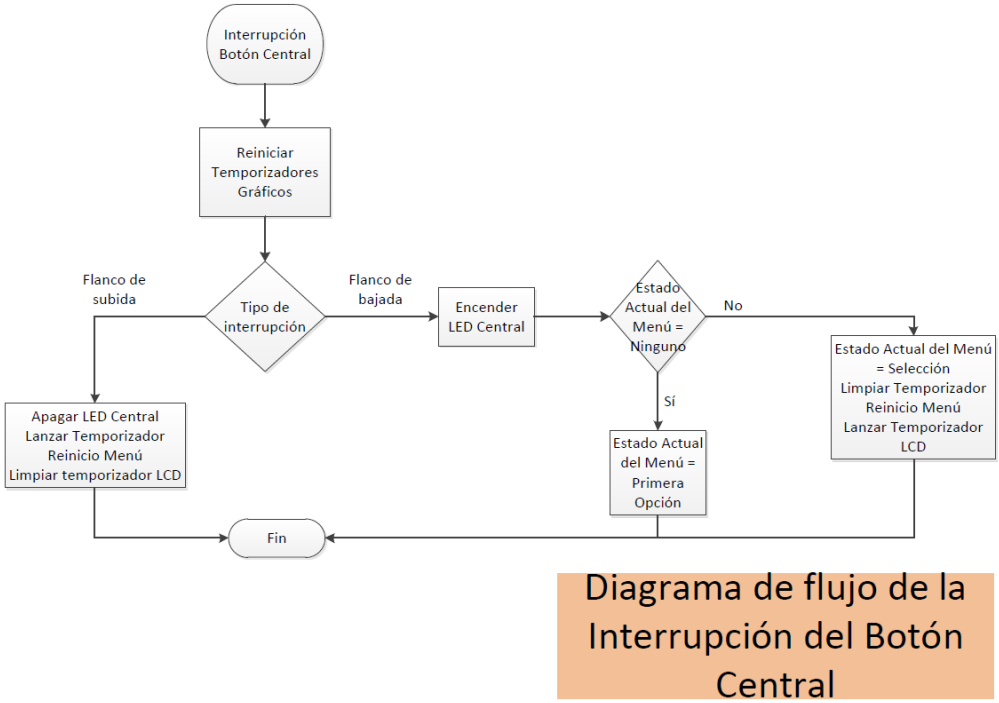


Figura 51. Diagrama de flujo de la Interrupción del Botón Central

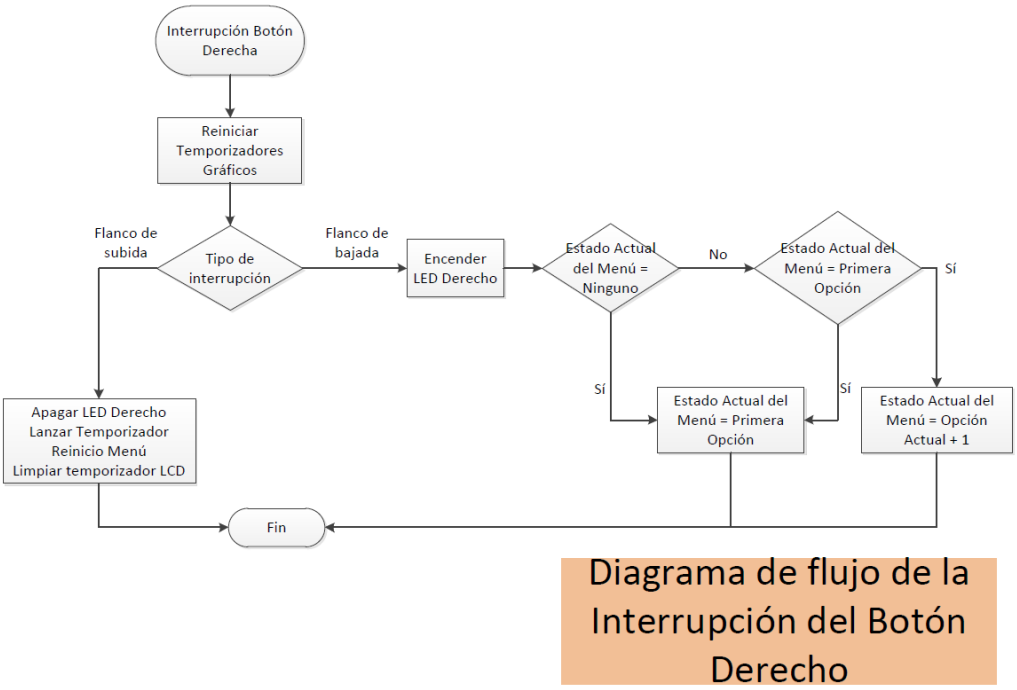


Figura 52. Diagrama de flujo de la Interrupción del Botón Derecho

E. COMUNICACIÓN CON EL GPS A TRAVÉS DE LA UART

Para comunicarnos con el GPS, simplemente utilizaremos la UART. Cada segundo el GPS transmitirá a través de la UART toda la información disponible. Entre toda la información que nos brinda, nosotros utilizaremos dos parámetros: Las coordenadas de posición y la fecha de la transmisión.

La trama del GPS está compuesta por 8 subtramas, que nos dan diferentes valores del GPS. Para extraer tanto las coordenadas como la fecha, debemos filtrar la trama que comienza por la cadena $GNGGA. Esta cadena es la trama más completa que nos da el GPS. Aparte de la hora y las coordenadas, obtenemos otros datos como pueden ser altitud sobre el mar, cantidad de GPS de seguimiento disponibles. A continuación, se adjunta una tabla donde se muestran en detalle los datos que contiene esta trama.

Tabla 5. Datos proporcionados en la cadena GNGGA por el GPS

Field	Name	Description
hhmmss.ss	*UTC Time*	*UTC of position in hhmmss.sss format, (000000.000 ~ 235959.999).*
llll.lll	*Latitude*	*Latitude in ddmm.mmmm format. Leading zeros are inserted.*
A	*N/S Indicator*	*N' = North, 'S' = South.*
yyyyy.yyy	*Longitude*	*Longitude in dddmm.mmmm format. Leading zeros are inserted.*
A	*E/W Indicator*	*E' = East, 'W' = West.*
x	*GPS quality indicator*	*GPS quality indicator. 0: Position fix unavailable. 1: Valid position fix, SPS mode. 2: Valid position fix, differential GPS mode.*
uu	*Satellites Used*	*Number of satellites in use, (00~24).*
v.v	*HDOP*	*Horizontal dilution of precision, (00.0~99.9).*
w.w	*Altitude*	*Mean sea level altitude (-9999.9~17999.9) in meter.*
x.x	*Geodial Separation*	*In meter*
zzzz	*DGPS Station ID*	*Differential reference station ID, 0000~1023. NULL when DGPS not used.*
hh	*Checksum*	

Para extraer esta cadena, lo primero que debemos hacer es configurar la interrupción de la UART para recibir cada dato. Una vez configurada la interrupción, dentro del handler de esta filtraremos el mensaje a partir de comprobar cada dato. Con un sencillo switch podremos comprobar si la cabecera es la que nosotros buscamos. Una vez comprobemos esto, añadiremos el resto de los datos al buffer hasta el siguiente retorno de carro. Con el mensaje completo, activaremos un flag para avisar de que hemos recibido el dato y limpiaremos los contadores utilizados.

Para no sobrecargar la interrupción, la asignación de los nuevos datos la haremos en el bucle principal, teniendo como condición el flag que acabamos de activar. Por ahora vamos a ignorar la función `SendDataGPS`, la cual se explicará después en el apartado de la comunicación serie.

Debido a que no siempre tendremos conexión directa con el GPS que nos dé nuestra posición, algunas veces tendremos algunos datos vacíos. Para poder distinguir este caso, comprobaremos una de las comas situadas al comienzo del comando, donde debería estar la latitud. Si aparece esta coma, es que no tenemos la posición y

rellenaremos con los datos con un carácter simbólico como puede ser 'E'. En caso contrario, rellenaremos con todos los datos.

A continuación, mostraremos el diagrama de flujo para la recepción de datos del GPS:

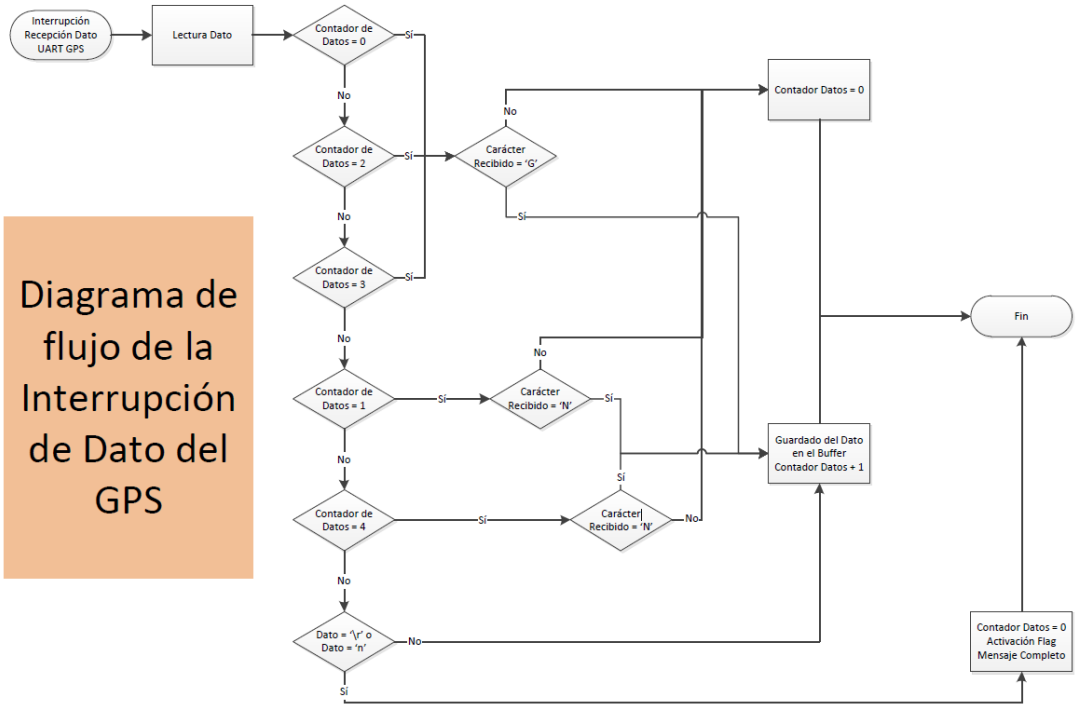


Figura 53. Diagrama de flujo de la Interrupción de Recepción de Dato del GPS

F. COMUNICACIÓN I2C CON MEMORIA EXTERNA Y GUARDADO DE DATOS EN MEMORIA

Como se ha comentado anteriormente, se han hecho pruebas con dos memorias distintas. Esto se traduce en que en el código tenemos las funciones de tratamientos de datos duplicadas. Aunque finalmente se ha seleccionado la AT24C256 como memoria principal, se ha optado por dejar las funciones de ambas para que sirvan de base en el futuro, por si se quieren utilizar ambas (por ejemplo, la de menor capacidad serviría para guardar solamente la frecuencia y la fuerza de la señal, mientras que elegida definitivamente tendría estos y más datos como la hora y la posición). Sin embargo, a tener algoritmos similares, sólo explicaremos el de la definitiva.

Para guardar datos en una memoria, lo primero y más conveniente es hacernos un mapa de los datos a guardar en la memoria. Los datos y tamaños que vamos a guardar son los siguientes:

- El contador de la frecuencia, con tamaño 1 byte. Las frecuencias son valores entre 87.5 y 108.0 MHz. Lo primero que nos salta al ojo es que estos valores son flotantes y guardarlos en memoria utilizaría mucha memoria. Podríamos pasarlos a enteros multiplicando por 10, pero necesitaríamos 2 bytes. Optamos por asignar un contador a cada frecuencia (empezando en 0 para la 87.5 y terminando en 205 para la 108.0) y haremos la conversión cuando sea necesaria.
- El valor de fuerza de señal, con tamaño 1 byte. El valor máximo que recibimos del módulo radio es 15 (4 bits), por lo que tenemos espacio más que de sobra.
- Horas, minutos y segundos donde se ha recibo la señal, con 1 byte para cada dígito. En total tenemos 6 bytes.
- Latitud, con 12 bytes. La latitud que nos da el GPS está formada a partir de 4 bytes enteros y 5 decimales, con el indicador Norte/Sur y una coma y un punto delimitadores. Para no complicarnos, guardaremos también con este formato.
- Longitud, con 13 bytes. Igual que la latitud sólo que tenemos un dígito más en el valor entero.
- Reserva, con 31 bytes. Dado que el total de los anteriores datos es 33 bytes y la escritura de página de la EEPROM permite escribir hasta 64, reservamos estos valores y así obtenemos una escritura con valores que cuadran correctamente, además de tener espacio de sobra para guardar más valores en el futuro.
- Además, nos reservamos 4 bytes al final de estos datos para guardar el número de escaneos completos que hemos realizado.

Todos los valores correspondientes a datos de cada frecuencia se escribirán tras escanear esa frecuencia. Como en nuestro caso queremos encontrar la frecuencia más libre de señal, guardaremos sólo los máximos que vayamos encontrando, para luego extraer la más libre de interferencias de entre todos los peores casos. Por lo tanto, una vez hayamos escaneado la frecuencia, deberemos leer la memoria para comprobar si el valor guardado anteriormente es mayor, y en caso contrario guardar el nuevo valor.

Para agilizar el proceso de extraer el mejor valor vamos a mantener una matriz con dos familias de valores: Las frecuencias ordenadas y la fuerza de señal máxima leída en cada una de ellas. Para ello, al iniciar el programa indexaremos estos valores a partir de la lectura de la EEPROM externa. Además, en esta función se utilizará el

puerto serie (del que hablaremos en el apartado siguiente) para mostrar todos los datos. También extraeremos el número de escaneos completados.

G. COMUNICACIÓN CON PUERTO SERIE PARA DEPURACIÓN

Otra de las ventajas de utilizar esta placa de desarrollo es que tenemos incluido un puerto serie para comunicarnos directamente con el PC. En nuestro caso, vamos a utilizarlo para la depuración de nuestro sistema. En especial, nos vamos a centrar en dos funcionalidades: Mostrar el barrido de la memoria, para ver fácilmente todos los datos que tenemos guardados; y mostrar la cadena de datos que vamos a utilizar del GPS.

Para mostrar los datos de la memoria lo haremos al comiendo del programa, cuando indexamos todos los valores de potencia de señal en la RAM. De esta manera, a la vez que leemos y guardamos estos valores, podemos mostrar toda la memoria. Prácticamente, utilizaremos los mismos datos tanto para indexar como para mostrar por el puerto serie. Añadiremos además una cabecera de los datos mostrados.

Por otro lado, para mostrar los datos del GPS, vamos a esperar al flag de mensaje completo que hemos comentado anteriormente. Una vez lo tengamos y el bucle principal este libre, mostraremos directamente el buffer de recepción.

7. FIABILIZACIÓN Y PRUEBAS

Para la fiabilización de este sistema se han realizado diferentes pruebas, tanto estáticas como en movimiento dentro de un automóvil. Después hemos extraído los datos mediante el modo depuración para comprobar que la fiabilidad de estos.

Como primera toma de contacto, vamos a dejar el dispositivo funcionando varias horas sin moverse para ver cómo funcionan cada una de las principales funciones del sistema.

A. PRIMERA ITERACIÓN

En nuestra primera iteración, nuestro objetivo es comprobar que la funcionalidad número 1 del dispositivo, el escaneo de la radio, funciona correctamente. No vamos a guardar los datos en memoria porque no es necesario para esta primera prueba. Sin embargo, definiremos dos buffers donde se guarden, tanto de manera temporal la última lectura del espectro de la radio como otro donde se guarden las mejores frecuencias. Para esta prueba vamos a dejar el dispositivo funcionando durante 24 horas en una posición estática. Nos aseguramos de que la radio esté correctamente conectada y lanzamos el escaneo.

Tras las 24 horas de trabajo, comprobamos la mejor frecuencia es la 88.9 MHz. Para validar este escaneo, debemos probar esta frecuencia en nuestro emisor. Al introducir esta frecuencia en nuestro emisor comprobamos que no existen interferencias en nuestra señal y podemos dar por válidos tanto la frecuencia como el funcionamiento del escáner.

B. SEGUNDA ITERACIÓN

Para nuestra segunda iteración vamos a comprobar el correcto guardado de las frecuencias en nuestra memoria externa. Repetimos la misma prueba que en la primera iteración, pero añadiendo la función de escritura de frecuencias en la memoria externa.

Tras 24 horas podemos comprobar, entrando en depuración, que la lectura de la EEPROM es correcta, y por tanto también la escritura. Podemos dar por validado el funcionamiento de la memoria externa. A su vez, podemos comprobar como la frecuencia más liberada sigue siendo la misma que en la primera iteración.

C. TERCERA ITERACIÓN

Para nuestra tercera iteración vamos a comprobar el funcionamiento del GPS, específicamente el funcionamiento del bus UART y del guardado de datos en EEPROM. De nuevo partimos de la misma base que en las anteriores iteraciones, aunque en este caso haremos la prueba en el exterior. La razón de esto es que en pruebas intermedias se ha comprobado que el GPS no funciona bien en interior.

Tras 24 horas de funcionamiento, se comprueba que la escritura no es correcta. Tras muchísimas pruebas, se observan 3 fallos fundamentalmente:

- Tamaño insuficiente para la recepción del mensaje del GPS. Lo que provoca un guardado de datos extraños.
- Posiciones incorrectas a la hora de extraer los datos del mensaje del GPS. Provoca, de nuevo, un guardado de datos extraños.
- Función con acceso desde puntero, con un tamaño mayor al que realmente tiene. Provoca errores catastróficos en el programa, con fallos como la escritura descontrolada en RAM.

D. CUARTA ITERACIÓN

Se corrigen los errores comentados anteriormente y se prueba de nuevo. Tras hacer una primera prueba para hacer un primer diagnóstico, y comprobar que el funcionamiento es correcto, se deja 8 horas en el exterior.

Una vez pasadas las 8 horas, extraemos y mostramos los datos por el puerto serie. A continuación, se adjunta una pequeña muestra de los datos extraídos:

FREQ	RSSI	HORA	LATITUD	LONGITUD
087.5	098	17:19:01	4139.47135 N	00443.76751 W
087.6	125	17:18:00	EEEEEEEEEE E	EEEEEEEEEE E
087.7	076	17:39:36	4139.47617 N	00443.77019 W
087.8	042	17:37:33	EEEEEEEEEE E	EEEEEEEEEE E
087.9	039	20:53:17	4139.47250 N	00443.75924 W
088.0	046	19:13:59	4139.46073 N	00443.76278 W
088.1	077	19:00:24	4139.45158 N	00443.77284 W
088.2	104	19:18:08	4139.45823 N	00443.75937 W
088.3	069	19:15:03	4139.46286 N	00443.76547 W
088.4	121	19:15:03	4139.46286 N	00443.76547 W
088.5	130	19:14:01	4139.46061 N	00443.76269 W
088.6	096	19:14:01	4139.46061 N	00443.76269 W
088.7	037	19:00:26	4139.45143 N	00443.77319 W
088.8	033	19:14:02	4139.46052 N	00443.76259 W
088.9	035	18:39:25	EEEEEEEEEE E	EEEEEEEEEE E
089.0	032	17:17:01	EEEEEEEEEE E	EEEEEEEEEE E
089.1	033	20:16:16	4139.46900 N	00443.76714 W
089.2	040	17:38:39	EEEEEEEEEE E	EEEEEEEEEE E
089.3	037	19:49:29	4139.46361 N	00443.76188 W
089.4	038	19:14:04	4139.46034 N	00443.76244 W
089.5	036	19:15:06	4139.46307 N	00443.76566 W
089.6	047	20:55:26	4139.46218 N	00443.75738 W
089.7	054	20:48:14	4139.46455 N	00443.76016 W
089.8	044	20:06:00	4139.46834 N	00443.76763 W
089.9	041	19:18:13	4139.45769 N	00443.75930 W
090.0	066	17:29:25	4139.47318 N	00443.77002 W
090.1	105	17:29:25	4139.47318 N	00443.77002 W
090.2	116	17:28:24	EEEEEEEEEE E	EEEEEEEEEE E
090.3	078	17:21:13	4139.47571 N	00443.76808 W
090.4	057	20:17:21	4139.46807 N	00443.76655 W
090.5	042	19:32:02	4139.46191 N	00443.76825 W
090.6	038	20:09:08	4139.47073 N	00443.76813 W
090.7	060	17:28:25	4139.47461 N	00443.76709 W
090.8	120	17:38:44	EEEEEEEEEE E	EEEEEEEEEE E
090.9	126	19:14:08	4139.45981 N	00443.76189 W
091.0	093	18:39:31	EEEEEEEEEE E	EEEEEEEEEE E

Figura 54. Registro de frecuencias para la cuarta iteración.

Recordemos que los datos con 'EEEEE' son aquellos guardados en los que el GPS aún no había triangulado nuestra posición.

En este caso, la frecuencia ha variado un poco respecto a la primera. Hemos pasado de la 88.9 MHz como la frecuencia más libre a la 89.0. La variación es minúscula.

Para validar esta nueva frecuencia, cambiamos la configuración de nuestro emisor y comprobamos de nuevo que la emisión es correcta y sin interferencias.

E. QUINTA ITERACIÓN: VIAJE URBANO

Para esta prueba, vamos a realizar nuestro primer viaje en coche en zona urbana. Las pruebas las vamos a realizar son alrededor de la población de Valladolid durante varios viajes habituales, desde la iglesia de San Nicolas hasta la parte alta del polígono de San Cristóbal y viceversa. A continuación, se adjunta el recorrido realizado para ambos viajes.

Figura 55. Mapa de desplazamiento urbano

El resultado que obtenemos es de nuevo la frecuencia 89.0 MHz. Esto tiene sentido, ya que nuestro desplazamiento ha sido de poco menos de 10 km.

Ya que es la frecuencia que hemos obtenido también en el caso anterior, no hace falta modificar la configuración de nuestro emisor para comprobar el funcionamiento.

F. SEXTA ITERACIÓN: VIAJE INTERURBANO

Para la última prueba, vamos a realizar un viaje de algo más de 250km (500+ si contamos también la vuelta). Vamos a viajar desde la población de Valladolid hasta Plasencia, en Cáceres. En el viaje de ida, tendremos el dispositivo encendido y escaneando en todo momento. Una vez en el destino, comprobaremos la mejor frecuencia extraída y configuraremos el emisor radio con esta. En el viaje de vuelta tan sólo deberemos estar atentos a la emisión y comprobar que esta no sufre interferencias.

A continuación, se muestra la ruta tomada para el viaje, tanto para la ida como para la vuelta.

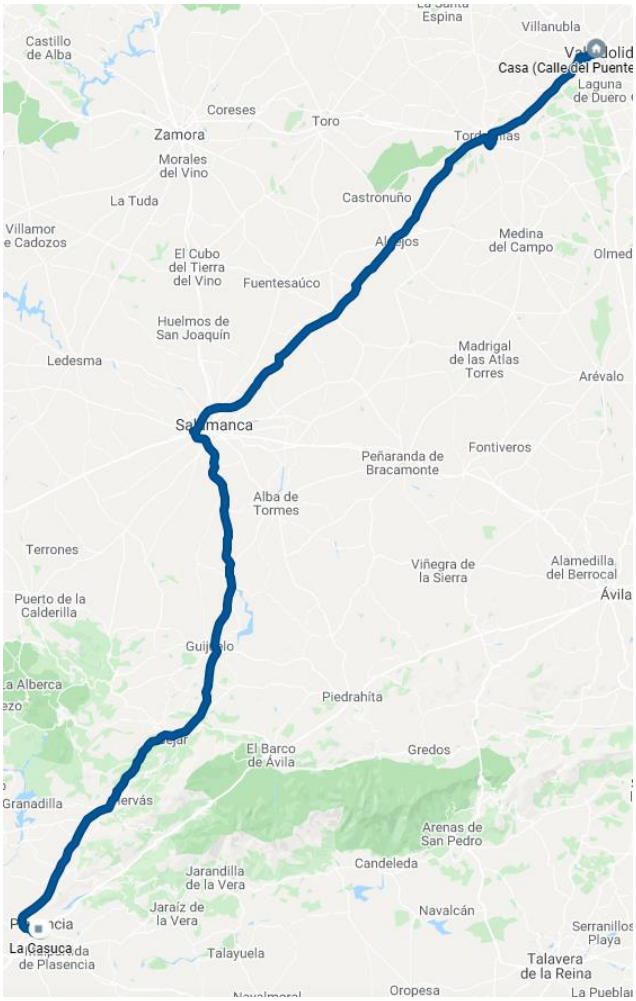


Figura 56. Mapa de desplazamiento interurbano

En el viaje de ida obtenemos que la frecuencia menos utilizada es la 107.1 MHz. Al utilizarla en el viaje de vuelta, comprobamos que el resultado es muy bueno. En todo el camino viajamos sin interferencias y con una calidad de emisión perfecta, por lo que podemos dar por válido el funcionamiento del sistema.

Si queremos profundizar un poco más en los datos para comprobar como ha actualizado las frecuencias y la fuerza de señal en memoria, solamente deberemos conectarnos con nuestro puerto serie a la placa.

El resultado que obtenemos es el siguiente:

- La gran mayoría de datos pertenecen a la zona geográfica de Valladolid. En general, todos los que corresponden a latitudes cercanas a 4139.46 N y a longitudes cercanas a 443.76 W.
- Obtenemos algunas señales más altas en la zona de Tordesillas, marcadas en azul. Son las correspondientes al valle comprendido entre las frecuencias 107.6 y 107.8 y cuya zona geográfica corresponde con latitud 4129.77 N y longitud 500.42 W.
- También tenemos algunas lecturas de la zona de Salamanca, marcadas en rojo. Las frecuencias comprendidas entre la 103.3 y la 103.5 MHz. La zona geográfica es alrededor de latitud 4058.82 N y longitud 541.41 W.
- Por último, tenemos algunas lecturas de la zona de Plasencia, marcadas en amarillo, con las frecuencias 103.1 y 103.2 MHz. Se corresponden con la zona geográfica de latitud 4002.03 N y longitud 605.48 W.
- También nos encontramos con algunas zonas en las que el GPS no tenía conexión y, por tanto, desconocemos nuestra posición. Estos valores son los marcados con latitudes y longitudes “EEEEE”.

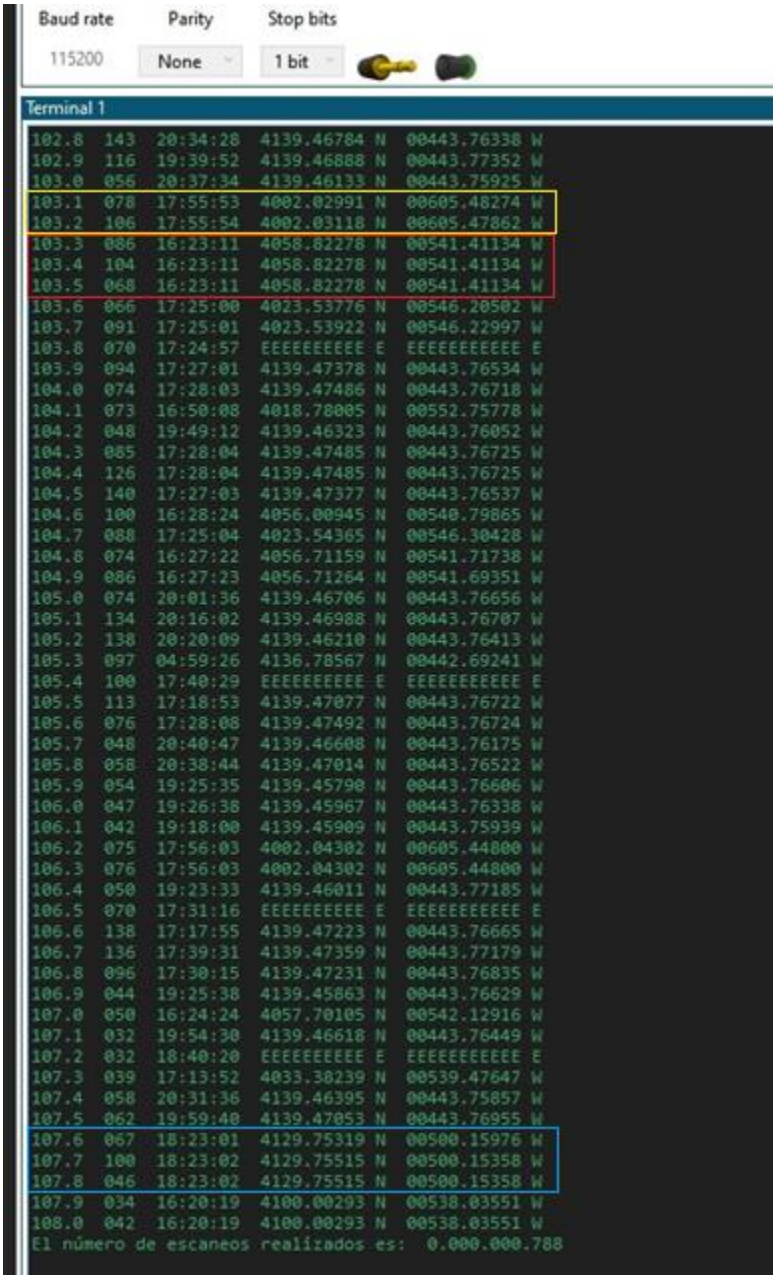


Figura 57. Registro de frecuencias en viaje interurbano

8. DESARROLLO DE UNA PLACA DE CIRCUITOS INTEGRADOS COMERCIAL

La prueba de funcionamiento que hemos obtenido ha sido correcta, por lo que vamos a realizar el siguiente paso para comercializar nuestro sistema: Desarrollar un primer prototipo de placa que aúne todos los componentes que hemos utilizado. Además, vamos a hacer hincapié en componentes que puedan permitir aumentar las funcionalidades de nuestro sistema. Al ser un proyecto inicial, se ha optado por utilizar los componentes más competitivos del mercado, llegando a un equilibrio entre funcionalidad y precio.

Para el sistema original, hemos elegido los siguientes componentes:

- Microcontrolador ATSAME54P20A. El mismo microcontrolador que viene incluido en la placa de desarrollo. Puede parecer que este microcontrolador está sobredimensionado para un proyecto de estas magnitudes, pero como vamos a dejar abierta la posibilidad de añadir nuevas funcionalidades, nos interesa mantenerlo y tener un micro todoterreno. Así, además, el código es totalmente compatible y no hace falta portearlo a otro.
- EEPROM Externa I2C AT24CM02. Un modelo más moderno de la que ya hemos utilizado. Compatibilidad con I2C de 1MHz y 2Mb de memoria, a priori más que suficiente para nuestro proyecto.
- Pulsadores B3FS-1012. Necesarios para movernos entre las distintas opciones de nuestra pantalla. Todos ellos acompañados con Leds 1N5819HW-7-F.
- Receptor RF SI4730-D60-GU. Totalmente compatible con las frecuencias de radio FM europeas y probablemente más fiable que el utilizado en nuestro sistema al ser un producto de la empresa "Silicon Labs". Por supuesto, tenemos el indicador de fuerza de señal recibida RSSI que necesitamos. Nos da la posibilidad de añadir un Jack de audio para escuchar la radio.
- Transmisor Radio SI4713-B30-GM. Totalmente compatible con las frecuencias europeas y con alta calidad de emisión.
- Módulo GPS NEO-M8M. El mismo que hemos utilizado en nuestro sistema. Con compatibilidad con la gran mayoría de sistemas de posicionamiento actuales (Galileo incluido) y con posibilidad de actualización de firmware, es un dispositivo perfecto para nuestro sistema. Sin embargo, a veces se pierde la conexión y puede ser un factor muy penalizante en nuestro sistema.

- Pila recargable. Para alimentar y mantener alimentado nuestro GPS cuando el resto de la placa está apagada. De esta manera, mejoras el rendimiento de este al mantenerse conectado y no reconectar en cada nuevo uso.
- Módulo Bluetooth Externo HC-06. La última parte esencial de nuestro proyecto. Compacto, compatible con Bluetooth 5.0, bajo consumo, firmware actualizable vía OTA y con Leds de estado. En resumen, un módulo super completo y barato. Al no encontrar el módulo integrable, se ha optado por utilizar un conector hacia el exterior de la placa.

A mayores, vamos a añadir los siguientes componentes para aumentar las funcionalidades de nuestro sistema:

- RTCC PCF2123TS/1,118. Fácil de configurar y con calendario incluido. De esta manera podemos guardar la fecha también y no sólo la hora que nos da el GPS.
- Transductor CAN MCP2562. Al conectarlo al bus CAN de nuestro coche, podemos realizar de manera automática el cambio de frecuencias tras mandar el comando correspondiente. Esta funcionalidad es muy importante ya que, junto al GPS, nos permite crear varias áreas de radio y cambiar de una a otra de forma autónoma, sin poner en riesgo al conductor.

Por último, vamos a dejar los siguientes puertos disponibles para añadir funcionalidades futuras:

- Puerto I2C libre. Sin ningún otro componente conectador. Así ganamos conectividad sin perder calidad de señal.
- Puerto Serie USB. Para mantener la depuración directa desde el entorno de desarrollo.
- Puerto para Displays. Se añade el puerto estándar para comunicación con pantallas OLED como la utilizada. Con este podríamos utizar pantallas OLED como la MCOT128064H1V-WM. Del doble de tamaño que la que hemos utilizado en nuestro sistema inicial (128 x 64 píxeles contra 128 x 32) y con los tres tipos de interfaz I2C, SPI y paralelo. Mantiene el blanco sobre negro de la Oled de nuestro sistema. Además, el controlador es totalmente compatible con nuestra librería de Atmel Start.

9. CONCLUSIONES

A. OBJETIVOS CUMPLIDOS

Los dos objetivos principales presentados en este proyecto se han cumplido. Primero, se ha programado y se ha conseguido una solución a partir de dispositivos separados. Y también, se ha presentado una placa de circuito impreso con diferentes componentes que pueda aunar los distintos dispositivos utilizados en un solo sistema. Podemos decir que el resultado ha sido correcto y positivo.

Vamos a detallar ahora estos objetivos a partir de los planteados inicialmente:

- Funcionalidad: El dispositivo es capaz de extraer la mejor frecuencia de entre todas las escaneadas durante el trayecto de escaneado.
- Memoria: Somos capaces de almacenar los valores más penalizantes, así como la hora de la detección de este valor y la posición global en la que ocurrió.
- Geolocalización: Nuestro módulo GPS nos informa correctamente de la posición global y se almacena correctamente con el resto de los datos. Sin embargo, a veces pierde la conexión GPS y no podemos obtener la posición global.
- Automatización y comodidad: Se presenta la placa de circuito impreso que permite cumplir este objetivo a partir de la adicción del Bus CAN y módulos externos.
- Interfaz: Se crean diferentes opciones para la gestión del sistema desde la pantalla OLED. Se añade además un conector para pantallas en la PCB que posibilita la opción de usar pantallas más grandes a través del mismo bus.
- Concentración de dispositivos: La placa de circuito impreso desarrollada aúna todos los dispositivos necesarios para el funcionamiento.

B. PROBLEMAS ENCONTRADOS

Aunque el resultado del proyecto ha sido positivo, nos hemos encontrado diferentes problemas a la hora de trabajar con este sistema:

- Memoria EEPROM de baja capacidad: La primera EEPROM utilizada ha sido la incorporada dentro de la placa de desarrollo Xplained PRO. Al añadir más campos a los datos, la memoria de 256 Bytes ha sido insuficiente. Por ello, la hemos sustituido por la AT24CM02.

- Precisión y fiabilidad del GPS: El GPS utilizado no siempre nos da correctamente la posición de nuestro dispositivo. Sin razón aparente, el dispositivo se desconecta de los puntos de triangulación. La solución ha pasado por marcar estas posiciones, como se ha comentado anteriormente.
- Número de frecuencias escaneadas durante muchos viajes: Al hacer viajes más largos, o simplemente, en mayor extensión global, es fácil llenar la memoria con valores de fuerza de señal recibida altos para todas las frecuencias. La primera opción es contar con un GPS fiable para crear "mapas de frecuencias". Sin embargo, como se ha comentado en el punto anterior, no disponemos de tal fiabilidad con este GPS y por ello, no se ha podido desarrollar esta solución.

C. LÍNEAS DE MEJORA

En este apartado vamos a comentar las posibles líneas de mejora y alguna solución a los problemas mostrados:

- Creación de los "mapas de frecuencias": A partir de la utilización de un GPS de más calidad y fiabilidad, podemos llevar a cabo los "mapas de frecuencias". De esta manera, podríamos tener varios valores de frecuencias libres en función de nuestra posición global.
- Programación del protocolo CAN y comunicación directa con el automóvil: Si conseguimos obtener los datos del protocolo CAN de nuestro modelo de automóvil, podemos mandar una orden directa desde nuestro microcontrolador a la centralita del coche (o a la radio) para cambiar de manera directa nuestra frecuencia de radio. Nos daría el plus de la automatización del sistema de cambio de frecuencia de nuestro automóvil, con las mejoras que eso conlleva a la hora de conducir.
- Modo administrador de la pantalla: Podemos tener acceso a funciones más técnicas a partir de, por ejemplo, introducir una contraseña. De esta manera, podríamos llevar a cabo un mantenimiento del sistema. Algunas de estas opciones podrían ser borrar toda la memoria, comandar la creación de un nuevo mapa de frecuencia o mostrar el valor de las últimas frecuencias leídas.
- Volcado de datos a partir de memoria SD: Dado que no siempre podremos conectarnos mediante PC al sistema o mostrar por pantalla los datos que necesitemos, se puede añadir una memoria SD al sistema para llevar un registro de todas las acciones que el módulo lleva a cabo, así como el escaneo de todas las frecuencias.

10. BIBLIOGRAFÍA

1. Differences between a processor and a microcontroller. The Definitive Guide to ARM® Cortex®-M3 and Cortex-M4 ProcessorsThird Edition (2014). Joseph Yiu [consulta: 7 de abril de 2021].
2. Background and history. The Definitive Guide to ARM® Cortex®-M3 and Cortex-M4 ProcessorsThird Edition (2014). Joseph Yiu [consulta: 7 de abril de 2021].
3. Why and How to Use the Serial Peripheral Interface to Simplify Connections Between Multiple Devices [consulta: 7 de abril de 2021]: https://www.digikey.es/es/articles/why-how-to-use-serial-peripheral-interface-simplify-connections-between-multiple-devices#:~:text=Se%20trata%20de%20una%20interfaz,direccionamiento%20y%20comprobaci%C3%B3n%20de%20estado
4. El estándar SPI para comunicación síncrona de alta velocidad [consulta: 10 de abril de 2021]: https://www.puntoflotante.net/COMUNICACION-SPI-TUTORIAL.htm#:~:text=Trat%C3%A1ndose%20de%20un%20canal%20de,puede%20llegar%20a%2010%20Mbps
5. Difference between I2C and SPI (I2C vs SPI), you should know [consulta: 10 de abril de 2021]: https://aticleworld.com/difference-between-i2c-and-spi/#:~:text=SPI%20does%20not%20have%20a,is%20better%20for%20long%2Ddistance
6. Automotive communications-past, current and future. DOI: 10.1109/ETFA.2005.1612631 [consulta: 12 de abril de 2021]: https://www.researchgate.net/publication/221503248_Automotive_communications-past_current_and_future/link/0c960516a556732948000000/download
7. LIN Protocol and Physical Layer Requirements [consulta: 12 de abril de 2021]: https://www.ti.com/lit/an/slla383/slla383.pdf?ts=1616325199344&ref_url=https%253A%252F%252Fwww.google.com%252F#:~:text=Message%20Frame%20Format,the%20protected%20identifier%20(PID)
8. Protocolo LIN BUS [consulta: 12 de abril de 2021]: https://sites.google.com/site/sistemasdemultiplexado/protocolos-de-comunicacin/lin-bus
9. La Historia del Wifi [consulta: 13 de abril de 2021]: https://purple.ai/es/blogs/la-historia-del-wifi/#:~:text=El%20WiFi%20se%20invent%C3%B3%20y,%C3%A1rea%20local%20inal%C3%A1mbricas%20(WLAN)

10. La Historia del Nacimiento del Bluetooth [consulta: 13 de abril de 2021]: https://www.fayerwayer.com/2011/09/la-historia-del-nacimiento-de-bluetooth/
11. Bluetooth 5.0, ¿qué es diferente y por qué importa? [consulta: 13 de abril de 2021]: https://hardzone.es/tutoriales/rendimiento/bluetooth-5-0-caracteristicas-novedades/
12. Electromagnetic Compatibility (EMC) Directive 2014/30/EU [consulta: 14 de abril de 2021]: https://ec.europa.eu/growth/sectors/electrical-engineering/emc-directive_en#:~:text=EMC%20designates%20all%20the%20existing,not%20affected%20by%2C%20electromagnetic%20disturbance
13. Radio Equipment Directive (RED) 2014/53/EU [consulta: 14 de abril de 2021]: https://ec.europa.eu/growth/sectors/electrical-engineering/red-directive_en
14. Advantages of the Cortex®-M processors. The Definitive Guide to ARM® Cortex®-M3 and Cortex-M4 ProcessorsThird Edition (2014). Joseph Yiu [consulta: 20 de abril de 2021].
15. Exceptions and Interrupts. The Definitive Guide to ARM® Cortex®-M3 and Cortex-M4 ProcessorsThird Edition (2014). Joseph Yiu [consulta: 20 de abril de 2021].