

DOCUMENTACIÓN TÉCNICA

TALLER DE PROYECTOS I SISTEMAS ELECTRÓNICOS



Datalogger for IoT

Autores:

Andrés Martín Yeves
Pablo Villacorta Aylagas
Rubén Serrano Rodríguez
Óscar Martín Casares

Tutor:

Jesús Hernández Mangas

27 de enero de 2023

Índice

1. Resumen Cierre de Proyecto	6
1.1. Datos básicos del proyecto.....	6
1.2. Resultado económico	6
1.2.1. Contrataciones	6
1.2.2. Dispositivos y materiales.....	6
1.3. Informe de situación final	7
1.3.1. Evolución temporal del proyecto.....	7
2. Informe Técnico Detallado	8
2.1. Objetivos del proyecto	8
2.2. Especificaciones del proyecto.....	9
2.3. Planificación	10
2.4. Diseño del logotipo	18
2.5. Herramienta de control de versiones	18
2.6. Diseño electrónico y esquemático.....	20
2.6.1. Búsqueda de componentes.....	20
2.6.2. Creación de componentes.....	21
2.6.3. Posicionado y rutado de componentes.....	21
2.6.4. Generación de listado de materiales	34
2.6.5. Análisis de consumo eléctrico	43
2.6.6. Análisis de consumo térmico.....	47
2.7. Diseño de la placa de circuito impreso	53
2.7.1. Creación de huellas de nuevos componentes	53
2.7.2. Definición de borde y zonas de conectores.....	54
2.7.3. Posicionado de componentes.....	54
2.7.4. Planos de masa y disipación térmica	54
2.7.5. Rutado de conexiones.....	55
2.7.6. Generación de ficheros para el fabricante (Gerber)	60
2.8. Fabricación de la placa de circuito impreso	60
2.8.1. Envío al fabricante	60
2.8.2. Recepción y revisión	61
2.8.3. Montaje de componentes.....	62
2.8.4. Depuración Hardware	66
2.9. Descripción del hardware de la FPGA (Verilog)	68
2.9.1. Generalidades system.v	72
2.9.2. UART0	73
2.9.3. UART1	74
2.9.4. UART2	74
2.9.5. SPI0 (ADC & BME)	75
2.9.6. SPI1 (LORA).....	75
2.9.7. I2C.....	76
2.9.8. GPIN	76
2.9.9. GPOUT.....	76
2.9.9. Temporizador	77

2.9.10. Interrupciones	77
2.9.11. Pines.pcf.....	79
2.9.12. Ampliación de la memoria.....	79
2.10. Programación del software	80
2.10.1. Programación del microcontrolador auxiliar	80
2.10.2. Programación de las rutinas	80
2.11. Verificación final.....	90
<i>Anexo A: Código en C</i>	<i>91</i>
<i>Anexo B: GeneraEntregas.batch.....</i>	<i>120</i>
<i>Anexo C: Código en Verilog</i>	<i>123</i>
<i>Anexo D: Pines.pcf.....</i>	<i>138</i>

Índice de figuras

Figura 1. Evolución temporal del número de horas dedicadas al proyecto.....	7
Figura 2. CubeSat desarrollado en Noruega.....	8
Figura 3. Esquema general del Datalogger.	9
Figura 4. Disposición de las placas de circuito impreso.	10
Figura 5. Diagrama de Gantt final del proyecto – Parte 1.....	12
Figura 6. Diagrama de Gantt final del proyecto – Parte 2.....	13
Figura 7. Diagrama de Gantt final del proyecto – Parte 3.....	14
Figura 8. Diagrama de Gantt al inicio del proyecto - Parte 1.....	15
Figura 9. Diagrama de Gantt al inicio del proyecto - Parte 2.....	16
Figura 10. Diagrama de Gantt al inicio del proyecto - Parte 3.....	17
Figura 11. Logotipo del equipo	18
Figura 12. Jerarquía de subdirectorios dentro del repositorio del proyecto	19
Figura 13. Cantidad de commits hechos por semana.....	20
Figura 14. Esquemático de la placa PCB0 - Parte 1.....	23
Figura 15. Esquemático de la placa PCB0 - Parte 2.....	24
Figura 16. Esquemático de la placa PCB0 - Parte 3.	25
Figura 17. Esquemático de la placa PCB1 - Parte 1.....	26
Figura 18. Esquemático de la placa PCB1 - Parte 2.....	27
Figura 19. Esquemático de la placa PCB1 - Parte 3.	28
Figura 20. Esquemático de la placa PCB2 - Parte 1.....	29
Figura 21. Esquemático de la placa PCB2 - Parte 2.....	30
Figura 22. Esquemático de la placa PCB2 - Parte 3.	31
Figura 23. Esquemático de la placa PCB3 - Parte 1.....	32
Figura 24. Esquemático de la placa PCB3 - Parte 2.	33
Figura 25. Listado de materiales de la placa PCB0 - Parte 1.....	35
Figura 26. Listado de materiales de la placa PCB0 - Parte 2.....	36
Figura 27. Listado de materiales de la placa PCB1 - Parte 1.....	37
Figura 28. Listado de materiales de la placa PCB1 - Parte 2.	38
Figura 29. Listado de materiales de la placa PCB2 - Parte 1.....	39
Figura 30. Listado de materiales de la placa PCB2 - Parte 2.....	40
Figura 31. Listado de materiales de la placa PCB3 - Parte 1.....	41
Figura 32. Listado de materiales de la placa PCB3 - Parte 2.	42
Figura 33. Análisis eléctrico PCB0.....	44
Figura 34. Análisis eléctrico PCB1.....	45
Figura 35. Análisis eléctrico PCB2.....	46
Figura 36. Análisis eléctrico PCB3.....	47
Figura 37. Análisis térmico PCB0.	49
Figura 38. Análisis térmico PCB1.	50
Figura 39. Análisis térmico PCB2.	51
Figura 40. Análisis térmico PCB3.	52
Figura 41. Dimensiones del sensor de polvo extraídas de la hoja de especificaciones. .	53
Figura 42. Huella sobre la PCB del sensor de polvo GP2Y1010AU0F.	53
Figura 43. Huella de placa de circuito impreso y conectores entre placas.	54
Figura 44. Layouts de la placa PCB0.....	56
Figura 45. Layouts de la placa PCB1.....	57
Figura 46. Layouts de la placa PCB2.....	58
Figura 47. Layouts de la placa PCB3.....	59

Figura 48. Anverso de la PCB2.....	61
Figura 49. Reverso de la PCB2.....	61
Figura 50. Anverso de la PCB3.....	62
Figura 51. Reverso de la PCB3.....	62
Figura 52. Error cometido durante el montaje de la placa PCB2.....	63
Figura 53. Correcto posicionamiento del conector del sensor de gas.....	63
Figura 54. Correcto posicionamiento del sensor GPS sobre la PCB2.....	64
Figura 55. Anverso de la placa PCB2 con todos los componentes.....	64
Figura 56. Reverso de la laca PCB2 con todos los componentes.....	65
Figura 57. Anverso de la placa PCB3 con todos los componentes.....	65
Figura 58. Reverso de la placa PCB3 con todos los componentes.....	65
Figura 59. Transistor mosfet Q3 erróneo de la placa de sensores sobre esquemático....	66
Figura 60. Transistor mosfet Q3 erróneo de la placa de sensores sobre PCB.....	66
Figura 61. Error solventado en transistor mosfet Q3.....	67
Figura 62. Diagrama de bloques con los componentes del sistema que es necesario definir dentro de la FPGA.....	69
Figura 63. Esquema de interconexiones sobre el microcontrolador implementado.....	70
Figura 64. Jerarquía de ficheros y módulos principales.....	71
Figura 65. Interconexiones laRVA – Memoria RAM & IO.....	72
Figura 66. Descripción hardware del multiplexor que permite obtener la señal iodo....	73
Figura 67. Descripción hardware de las señales de habilitación chip select.....	73
Figura 68. Registros utilizados para la UART0.....	74
Figura 69. Registros utilizados para la UART1.....	74
Figura 70. Registros utilizados para la UART2.....	75
Figura 71. Registros utilizados para el SPI0.....	75
Figura 72. Registros utilizados para el SPI1.....	76
Figura 73. Registros utilizados para el GPIN.....	76
Figura 74. Registros utilizados para el GPOUT.....	77
Figura 75. Registros utilizados para el timer.....	77
Figura 76. Funcionamiento de los flags TMF que determinan la señal de interrupción.	77
Figura 77. Registros utilizados para el control de interrupciones.....	78
Figura 78. Esquema hardware del control de interrupciones.....	79
Figura 79. Señal de control del sensor de gas y señal de salida para distintas concentraciones de gas.....	82
Figura 80. Tiempo de muestreo del pulso de salida en el sensor de polvo.....	83
Figura 81. Señal de control del sensor de polvo.....	83
Figura 82. Cronograma de comunicación SPI con el sensor BME680.....	86
Figura 83. Cronograma de comunicación SPI con el ADC.....	87
Figura 84. Esquema de comunicación SPI con el LoRA	88

Índice de tablas

Tabla 1. Direcciones de memoria utilizadas por los periféricos.....	71
---	----

1. Resumen Cierre de Proyecto

1.1. Datos básicos del proyecto

Nombre del proyecto: Datalogger for IoT

Tipo de proyecto: Diseño electrónico

Sector: Telecomunicaciones

Subsector: Electrónica

Ubicación: Escuela Técnica Superior de Ingenieros de Telecomunicación

Localidad: Valladolid

Dimensiones del proyecto: Cuatrimestral

Nombre de los responsables del proyecto: Andrés Martín Yeves

Nombre de los responsables del proyecto: Pablo Villacorta Aylagas

Nombre de los responsables del proyecto: Rubén Serrano Rodríguez

Nombre de los responsables del proyecto: Óscar Martín Casares

Email de los responsables del proyecto: andres.martin@estudiantes.uva.es

Email de los responsables del proyecto: pablo.villacorta@uva.es

Email de los responsables del proyecto: rubenfrancisco.serrano@estudiantes.uva.es

Email de los responsables del proyecto: oscar.martin.casares@estudiantes.uva.es

1.2. Resultado económico

En esta sección, se busca hacer una síntesis de toda la información contable del proyecto con el objetivo de evaluar el valor que éste tiene. Los costos se pueden dividir principalmente en ‘contrataciones’ y ‘dispositivos y materiales’.

1.2.1. Contrataciones

El grupo encargado de llevar a cabo el proyecto está compuesto por cuatro estudiantes de máster de ingeniería de telecomunicaciones. Tomando en cuenta que el salario base promedio de un ingeniero en telecomunicaciones en España es alrededor de 29.500 €/año y considerando una aproximación de 1888 horas laborables al año, se puede estimar que cada miembro del equipo tiene un salario promedio de 15.625 €/hora.

Gracias al registro de control de cambios, que se discutirá más adelante, se ha podido llevar un seguimiento de las horas empleadas en el proyecto por los miembros del equipo. La información de seguimiento ha sido escrita en un Excel donde se recogen todas las horas por cada componente del grupo. El total de horas conjuntas de trabajo empleadas han sido de 325,65 dando un total de costes en salarios de 5088,28€

1.2.2. Dispositivos y materiales

Los costes materiales de los dispositivos que se han empleado se pueden encontrar en la sección 2.5.4. Generación de listado de materiales. Teniendo en cuenta todos los componentes empleados el coste asciende a los 258,82€.

El coste de las placas de circuito impreso asciende de manera aproximada a 8,5 € la unidad, lo que da un total de 34€. Entonces, el coste material del proyecto asciende a 292€.

1.3. Informe de situación final

Es importante diferenciar los costes de diseño de los costes de producción, los destinados al diseño ascienden a 5088,28€ mientras que los de producción se estiman en 292€.

Se puede observar de manera clara que el mayor esfuerzo económico ha sido el destinado al diseño del *datalogger*, si tasáramos su valor individual en 350€ sería necesario vender al menos 87,72 unidades para amortizar el esfuerzo invertido.

1.3.1. Evolución temporal del proyecto

El proyecto no ha seguido una evolución lineal y equidistribuida debido a que cada fase del proyecto ha tenido una dificultad diferente, contando además con que se ha tenido que compatibilizar con el trabajo de otras asignaturas de la titulación. En la figura a continuación podemos visualizar cuál ha sido la distribución en función de cada integrante del grupo a modo de diagrama de barras, siendo el eje horizontal el conteo de la vida útil del proyecto y el vertical las horas dedicadas. El proyecto se inició el día 22 de septiembre de 2022 y se finalizó el día 26 de enero de 2023, es decir, 4 meses aproximadamente.

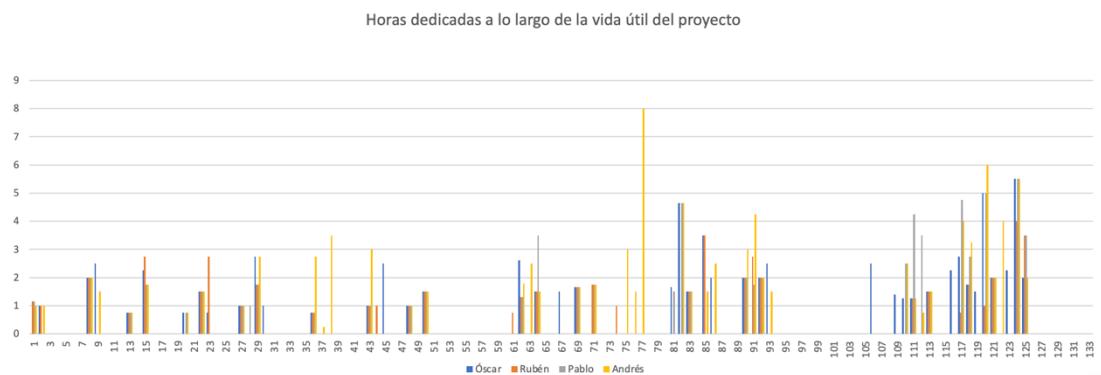


Figura 1. Evolución temporal del número de horas dedicadas al proyecto.

Podemos apreciar esta evolución temporal mediante el desglose de las distintas tareas realizadas en el proyecto y, valga la redundancia, mostrar el tiempo invertido en cada una de ellas. Esta evolución se puede encontrar en el punto 2.3, además de una comparación con los tiempos estimados para cada una de las tareas que se hizo en un inicio.

2. Informe Técnico Detallado

2.1. Objetivos del proyecto

Antes de comenzar con el diseño de un proyecto, es conveniente definir cuál es el objetivo que se persigue. En primera instancia, se busca desarrollar un sistema de comunicación inalámbrica que permita capturar diferentes magnitudes físicas a través de diferentes sensores. Asimismo, se quiere que los datos capturados mediante los sensores sean transmitidos a una infraestructura de ordenadores para poder procesarlos posteriormente.

La propuesta inicial para obtener el objetivo mencionado fue el desarrollo de una satélite de comunicaciones o CubeSat, como los utilizados por agencias espaciales NASA o ESA. Dichos satélites poseen unas dimensiones de alrededor de 10x10x10 cm tal y como podemos observar en la figura 2, se estima que desde que se lanzó el primero en el año 2003 hasta día de hoy, se encuentran en el espacio más de 3.200 CubeSats.

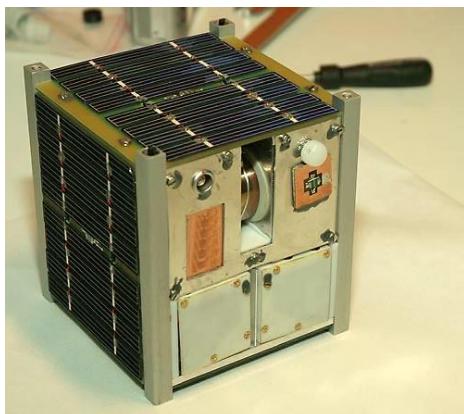


Figura 2. CubeSat desarrollado en Noruega.

Sin embargo, debido a su alta complejidad y coste, es completamente inviable para el taller de proyectos.

En segunda instancia, surge otra idea de diseño consistente en el desarrollo de una radio sonda capaz de obtener diferentes magnitudes. Éstas son comúnmente utilizadas en meteorología, se usan en globos meteorológicos que usan gases de elevación con el objetivo de medir parámetros atmosféricos y transmitirlos a un receptor fijo. Entre los parámetros más importantes nos podemos encontrar la presión, la altitud, la temperatura o la humedad relativa.

Al contrario del satélite miniatura, la opción de la radiosonda es más plausible, por ello buscaremos aproximarnos a esta idea. Sin embargo, dado que una radiosonda se puede poner en marcha sin necesidad de lanzarla a las alturas, nosotros simplemente la desplazaremos con la ayuda de un vehículo y obtendremos datos de temperatura, humedad, presión, monóxido de carbono, partículas que estén geolocalizadas y transmitiremos dicha información mediante radio.

2.2. Especificaciones del proyecto

En conocimiento del sistema que se quiere diseñar, es conveniente presentar las ideas y componentes generales necesarios para la herramienta, podemos observar estos en la figura 3. Entre los componentes principales nos encontramos con una antena, una CPU, una FPGA, una memoria *flash*, los sensores y su correspondiente alimentación.

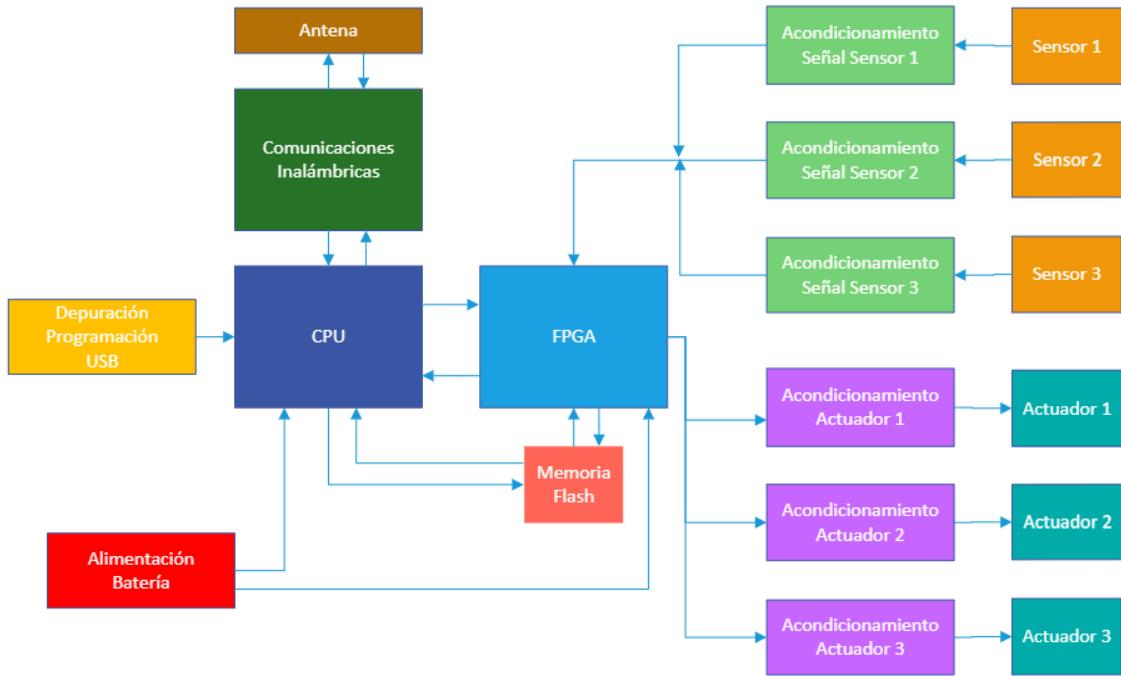


Figura 3. Esquema general del Datalogger.

Dicho sistema se dispondrá en diferentes placas de circuito impreso o PCBs, montadas unas sobre otras de la manera mostrada en la figura 4. Con este montaje conseguiremos que se puedan reusar o sustituir partes no funcionales de manera muy sencilla, además de poder así rediseñar nuevas partes en el futuro.

En la primera placa nos encontraremos con la alimentación y la programación incluida en la unidad central de procesamiento o CPU. Contendrá los conectores de la batería y USB, el microcontrolador o un puerto USB – serie. En segundo lugar, tendremos una placa que incluirá la FPGA, conectores de acceso a parte del patillaje de la FPGA o la memoria *flash*. La tercera placa estará destinada a la incorporación de los diferentes sensores, así como sus circuitos de acondicionamiento necesarios. Por último, la placa restante contendrá las comunicaciones, es decir, el módulo de comunicaciones inalámbricas WiFi y el transceptor LoRa.

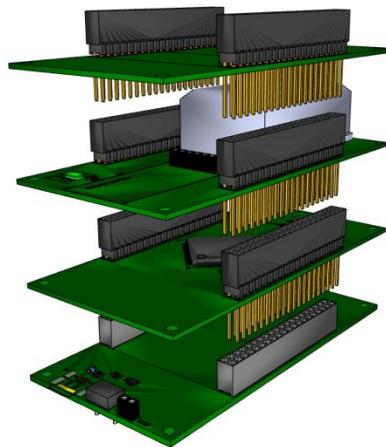


Figura 4. Disposición de las placas de circuito impreso.

2.3. Planificación

Para poder gestionar el proyecto, existen diferentes herramientas de planificación y control. Una de dichas herramientas es el Diagrama de Gantt, la cual hemos utilizado para planificar nuestro trabajo y que nos proporciona una vista general de las tareas programadas además de cuáles han de completarse y en qué fecha.

En primer lugar, se debe definir la estructura de descomposición del trabajo en la cual deben listarse las tareas y proporcionarlas una estimación del tiempo necesario para completar cada una de ellas, uno de los cometidos más complicados. Se puede observar que la planificación inicial dista de la final. Al comienzo de un proyecto es muy complicado anticipar los problemas que van a surgir y más cuando se carece de experiencia en ello.

En un inicio podemos ver que el ‘diseño electrónico y la captura esquemática’ ha sufrido un retraso de 12 días sobre el planteamiento inicial del proyecto. Se puede observar claramente en el diagrama final del proyecto que se debe a que se atrasó el inicio del diseño electrónico y esquemático de la *PCB2* respecto a la *PCB1*. En el diagrama de Gantt inicial tras finalizar esta tarea sobre la 1 se empieza de seguido con la 2, pero en este caso tenemos un lapso de 27 días, casi 1 mes. Aunque se haya aprovechado ese tiempo en la realización de otras tareas en paralelo, no se puede obviar que ha sido un cambio en la planificación del proyecto que ha resultado en un atraso del resto de las tareas.

Si que es cierto, que, por suerte el equipo se ha podido dividir para estar trabajando en paralelo, siendo ésto una clara ventaja sobre el diagrama de Gantt inicial, el cual distribuye todas las tareas de una forma muy secuencial, por lo que se ha conseguido ganar mucho tiempo.

En la tarea ‘diseño de la placa de circuito impreso’ aunque también se aprecia el mismo efecto de las placas ya mencionado, es cierto que el diagrama inicial dista del final debido a que en el inicial se consideró que esta tarea demoraría mucho más tiempo del debido.

En un principio por desconocimiento, marcamos una fecha para mandar a fabricar las placas, no obstante, se mandaron a fabricar antes de lo que nosotros habíamos planeado.

Sin embargo, a posteriori conocimos por parte de nuestro tutor que dichas placas se mandaron a fabricar más tarde de lo planeado. Por dichas razones el diagrama final dista del diagrama inicial.

Por último, cabe destacar la diferencia de tiempos entre la verificación hardware y programación del firmware. Estos dos apartados han acaparado mucho más tiempo del que se pensaba, aunque como las tareas y objetivos anteriores se consiguieran en fechas anteriores a las previstas, por lo que el proyecto ha finalizado en la *deadline* especificada.

Las figuras 5 a 10 muestran, respectivamente, el diagrama de Gantt una vez finalizado el proyecto y el diagrama de Gantt que se diseñó en su comienzo.

Diagrama de Gantt

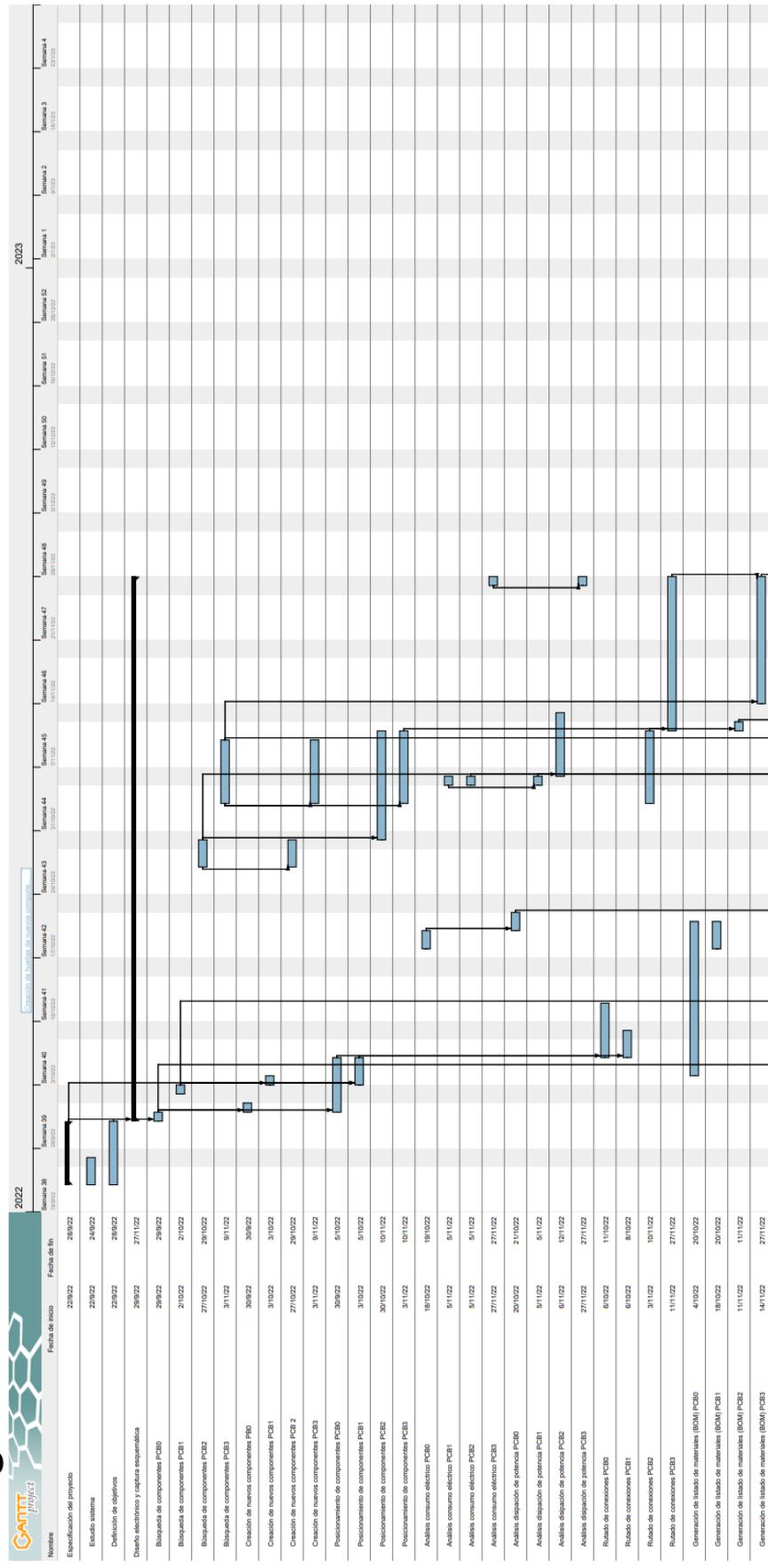


Figura 5. Diagrama de Gantt final del proyecto – Parte 1.

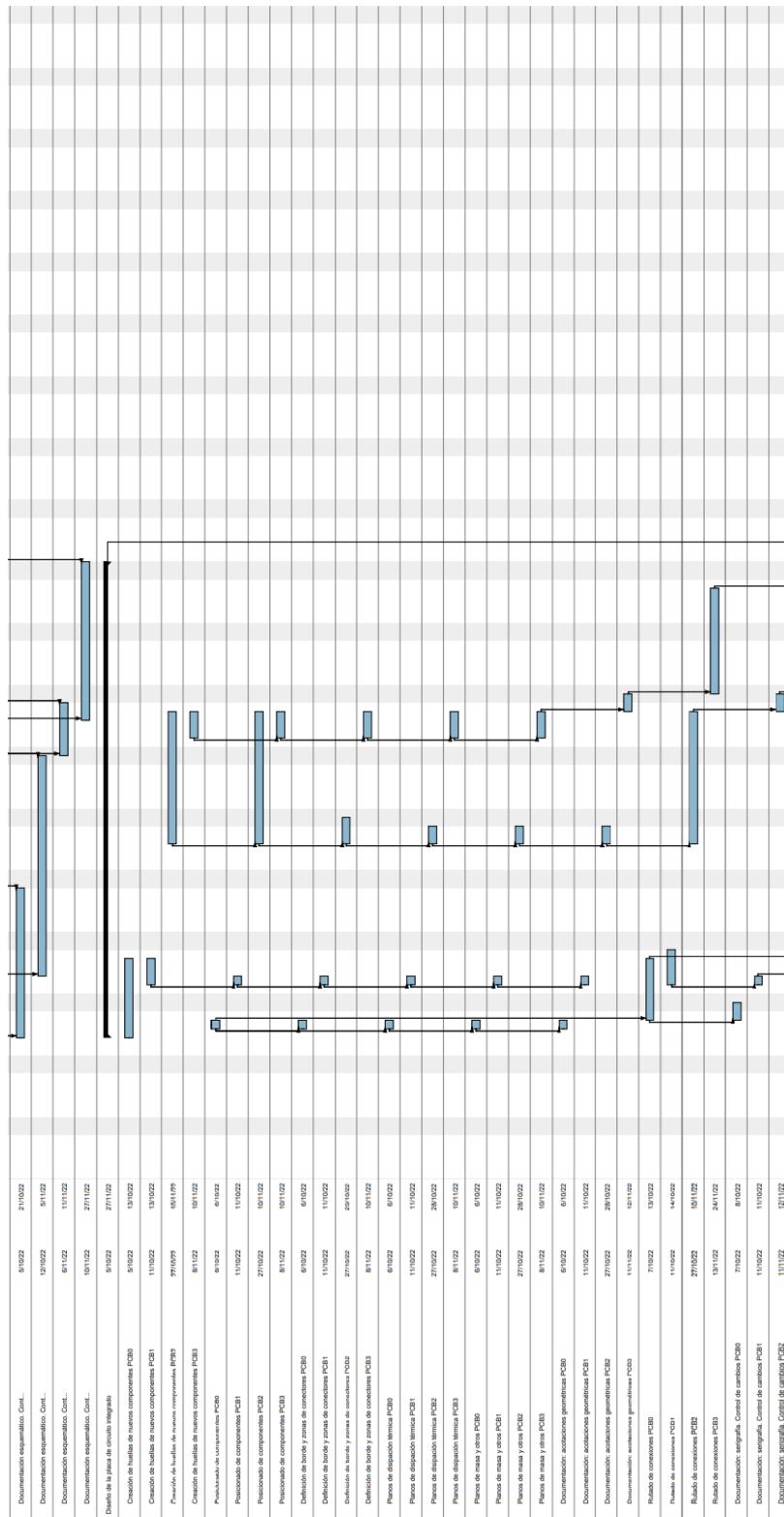


Figura 6. Diagrama de Gantt final del proyecto – Parte 2.

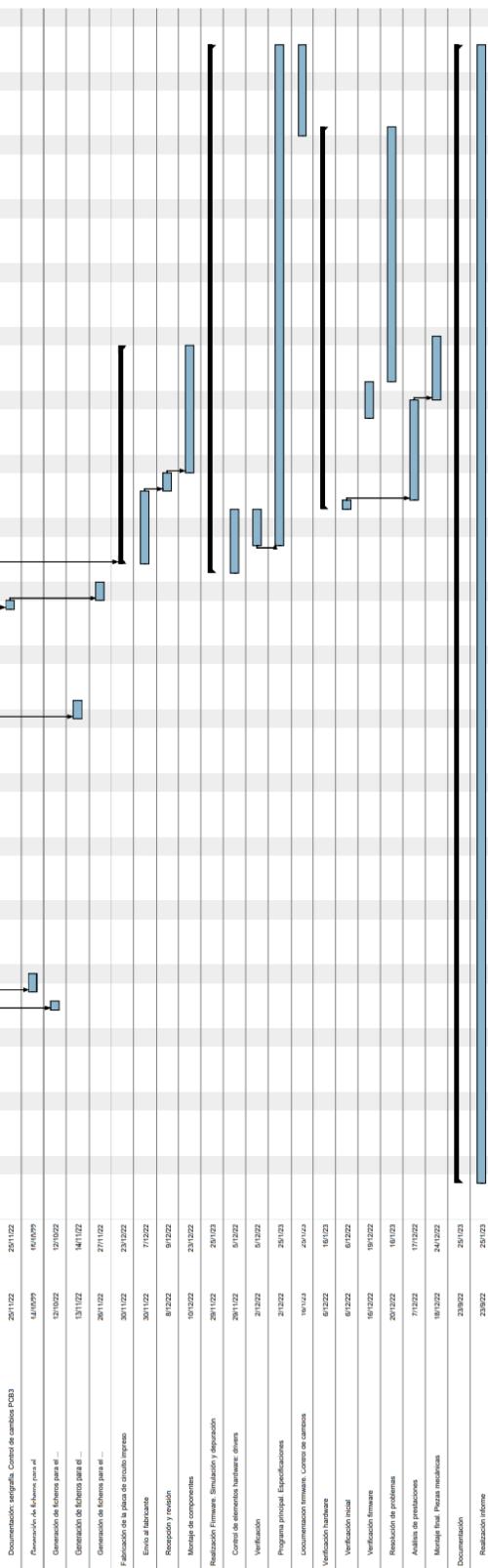


Figura 7. Diagrama de Gantt final del proyecto – Parte 3.

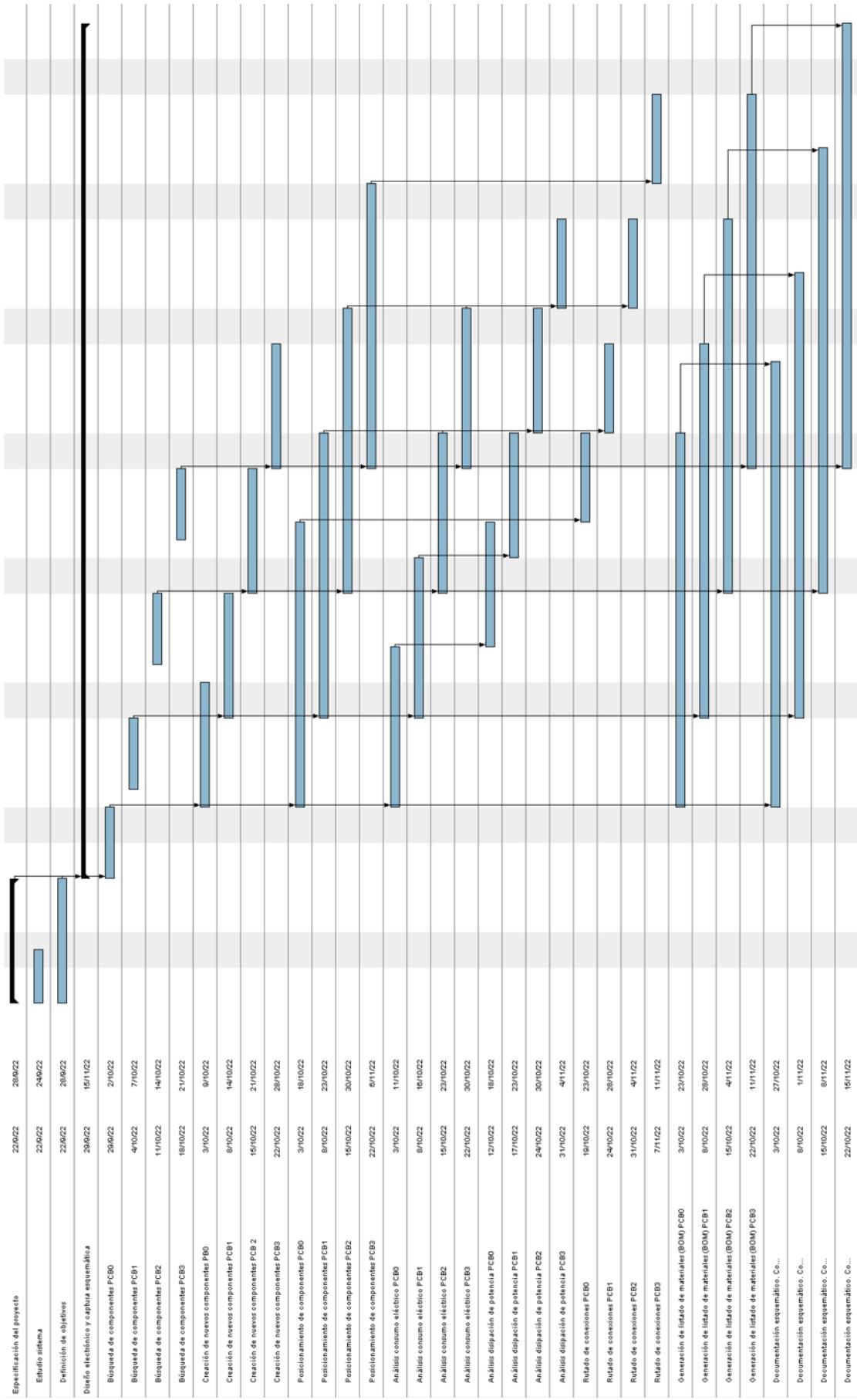


Figura 8. Diagrama de Gantt al inicio del proyecto - Parte 1.

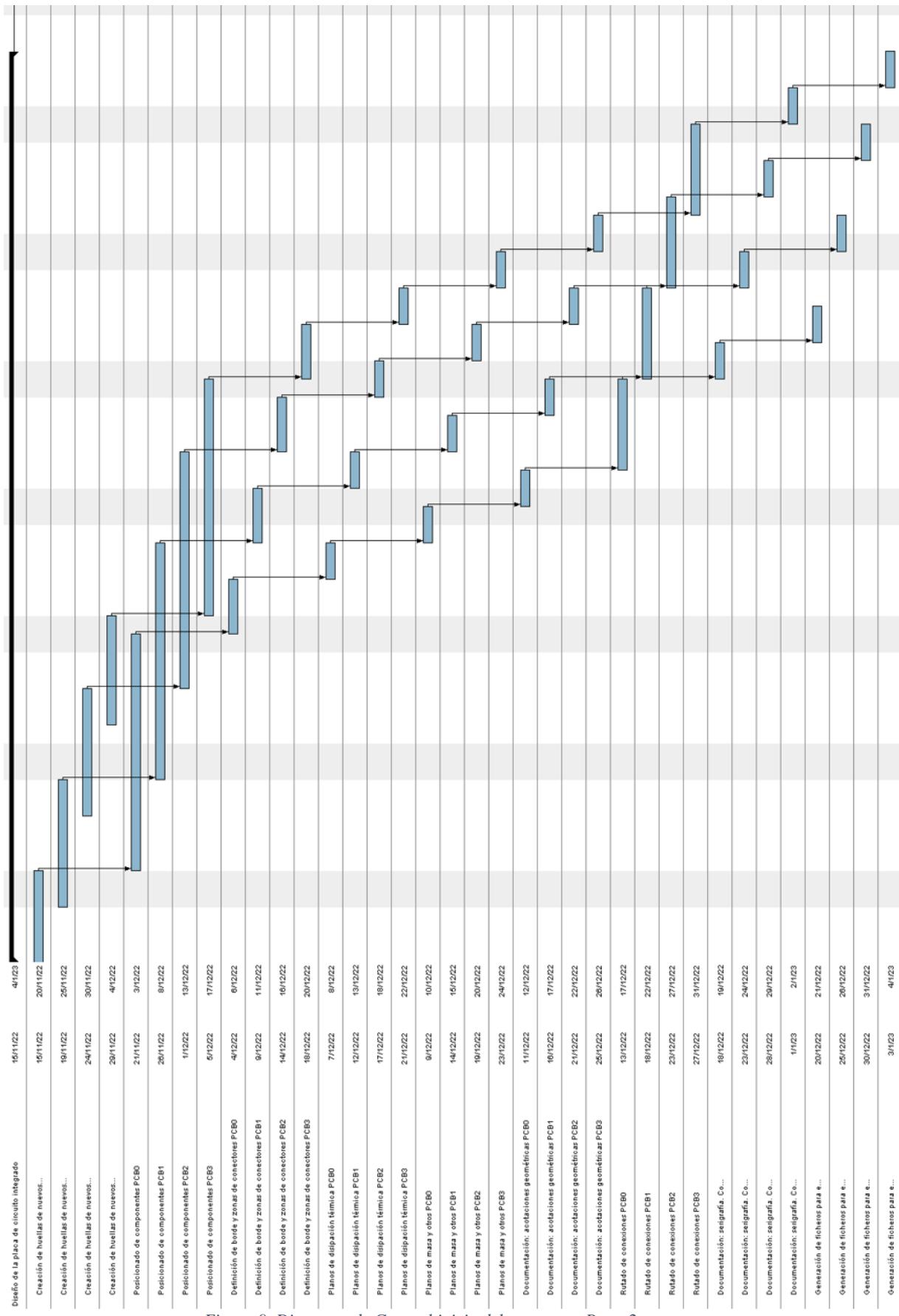


Figura 9. Diagrama de Gantt al inicio del proyecto - Parte 2.

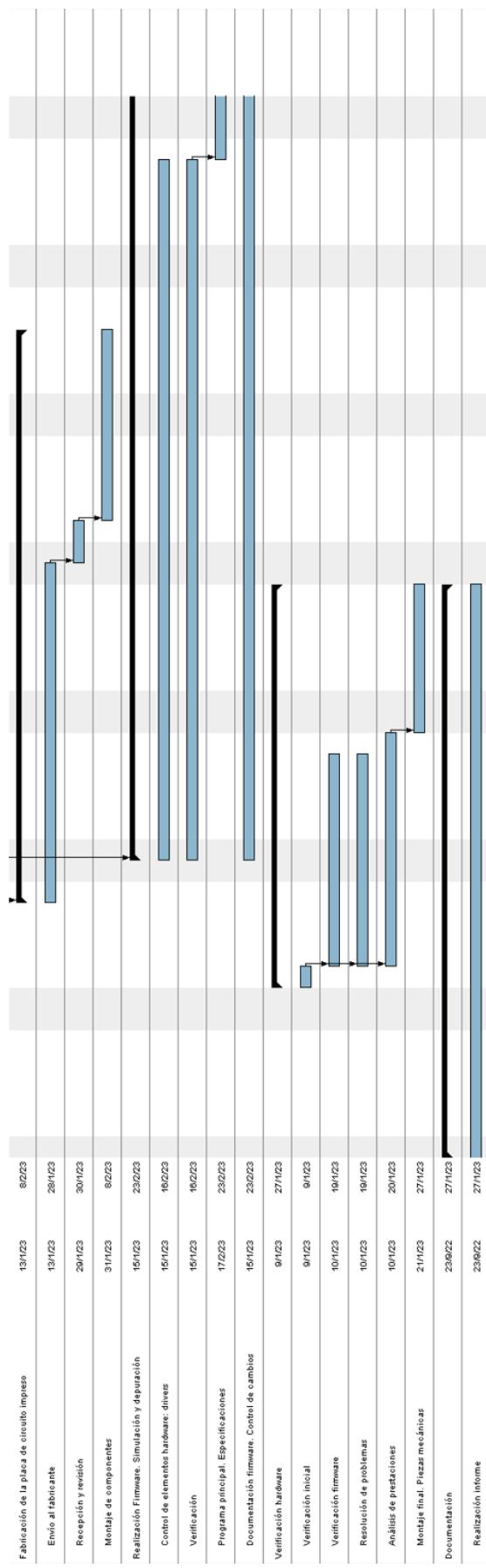


Figura 10. Diagrama de Gantt al inicio del proyecto - Parte 3.

2.4. Diseño del logotipo

Uno de los primeros pasos del proyecto ha sido la elaboración de un logotipo de referencia que fuese representativo del equipo y sirva de referencia como marca personal en el resto de los documentos.

Utilizando las iniciales de cada uno de los miembros del equipo se probaron distintas combinaciones y finalmente utilizando dall-e, una inteligencia artificial disponible en <https://openai.com/dall-e-2/> se consiguió el logotipo final del equipo que además hace referencia a lo mucho que a nuestro compañero Óscar le gusta el prado de su pueblo.



Figura 11. Logotipo del equipo

2.5. Herramienta de control de versiones

En el mundo profesional cada vez es más común el trabajo colaborativo y por ende las herramientas destinadas a ello. Es importante hoy en día estar familiarizado con sistemas de control versiones, los cuales permiten almacenar todas las versiones que ha habido durante la realización del proyecto, permitiéndonos además acceder a versiones antiguas en cualquier momento actuando así a modo de ‘copias de seguridad’.

Para la realización de este proyecto se ha utilizado la herramienta GitHub en la que se ha creado un repositorio de trabajo. Dicho repositorio ha sido organizado en distintas subcarpetas creadas sobre la marcha según las necesidades, con el objetivo de poder encontrar todo de manera ordenada e intuitiva, tal y como podemos observar en la Figura 12.

En éste además hemos ido añadiendo en el fichero README .md todos los comentarios o instrucciones de utilidad necesarias durante todo el proyecto, para poder así recordar en cualquier momento cómo hicimos determinada acción o cosas de suma importancia.

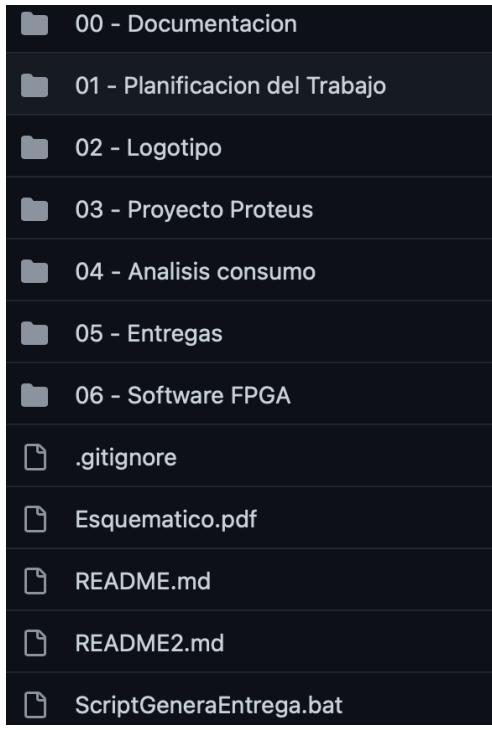


Figura 12. Jerarquía de subdirectorios dentro del repositorio del proyecto

Para manejarnos con la herramienta se han hecho uso de comandos básicos, entre los cuales nos podemos encontrar con:

- git clone 'url': con este comando se ha podido clonar el repositorio en los respectivos ordenadores para poder trabajar de forma más cómoda
- git add *: comando utilizado para añadir cambios de archivos en el directorio clonado.
- git commit -m 'nombre del commit': Este comando toma todos los cambios de archivos descritos por el 'git add', crea un nuevo objeto de confirmación y se establece una rama para que apunte a esta nueva confirmación.
- git push: comando que envía todos los objetos modificados localmente el repositorio remoto. Con el uso de este comando se ha podido ir subiendo las nuevas actualizaciones.
- git pull: comando utilizado para actualizar el repositorio local.
- git status: este comando muestra el estado de los archivos que hay en el repositorio remoto en comparación con los del directorio de trabajo. Esto ha servido para verificar que en todo momento el repositorio local se encontraba en buen estado sin errores.
- git mv old_filename new_filename: Renombra los archivos
- git reset --hard: Borra todos los cambios locales del repositorio actualizando a la versión más reciente en remoto.
- En caso de tener problemas con el vim (repositorio), para salir, presionar la secuencia de teclas: ESC, :q, ENTER

Esta herramienta nos proporciona la capacidad de ver las estadísticas acerca del número de *commits* que se han realizado, podemos observarlos en la Figura 13 en la que nos presenta el número de *commits* por semana.

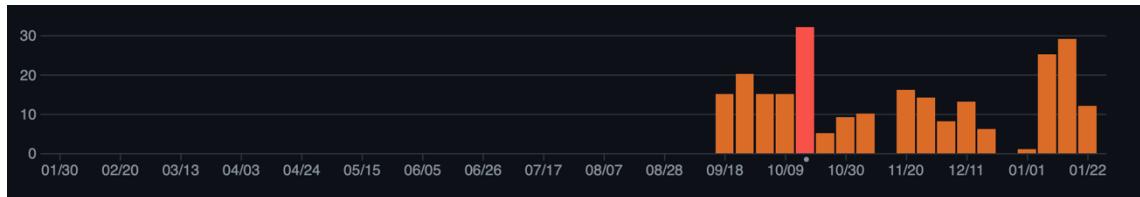


Figura 13. Cantidad de commits hechos por semana.

En ésta podemos observar que las semanas de mayor tráfico en nuestro repositorio fueron la tercera semana de octubre junto con la penúltima semana del proyecto, con 33 y 29 *commits* respectivamente.

Una vez por semana era necesario realizar una entrega en la página web de la asignatura de todos aquellos avances que habíamos realizado. Para facilitar dicha tarea se realizó un fichero ‘.batch’ que lo automatizaba, dándole además el nombre correspondiente con su día de entrega. Éste lo podemos visualizar en el anexo B.

Destacar además que hemos incluido un segundo fichero README2 .md con el objetivo de documentar todos los fallos (con sus soluciones) que se nos han presentado durante el desarrollo del proyecto, debido a que en varias ocasiones hemos compartido las mismas dudas de cómo hacer determinadas tareas. Además, éstos se han documentado porque pensamos nos podrán servir en nuestro futuro profesional, ya que es probable que se nos presenten circunstancias similares.

2.6. Diseño electrónico y esquemático

Para llevar a cabo el desarrollo del *datalogger* se ha recurrido a una herramienta software de automatización de diseño electrónico. En la actualidad nos encontramos con numerosas opciones punteras como pudieran ser *Eagle*, *Altium*, *KiCad* u *Orcad*, sin embargo, en nuestro caso hemos utilizado *Proteus Professional* debido a lo familiarizados que nos encontrábamos con la herramienta. Ésta ha sido desarrollada por *Labcenter Electronics Ltd*.

En primer lugar, conociendo cuáles son las especificaciones del proyecto estamos en disposición de buscar qué componentes vamos a utilizar para elaborar el dispositivo.

2.6.1. Búsqueda de componentes

En el proyecto se ha trabajado con uno de los principales distribuidores de productos para el diseño de sistemas electrónicos, la empresa *Premier Farnell Ltd*. En ésta se han obtenido la mayoría de los componentes como pueden ser el microcontrolador ARM, diodos, leds, sensores, conectores o baterías. De cada uno de los componentes se han anotado sus números de referencia y su precio, para así poder elaborar un presupuesto del proyecto completo, el cual desglosaremos más adelante.

Además, dado que en este momento nos encontramos en una situación complicada a nivel mundial en cuanto al abastecimiento de este tipo de sistemas electrónicos, hemos adquirido un transceptor LoRa en la conocida empresa china *Aliexpress*.

Dado que no se quieren cometer fallos de diseño, una parte importante del proyecto consiste en analizar cuál será el consumo de cada elemento, por ello hemos obtenido la hoja de características de cada uno de ellos. Esta parte la podremos conocer en profundidad en el apartado de consumo eléctrico y térmico.

2.6.2. Creación de componentes

Una gran parte de los componentes los podemos encontrar en diferentes librerías, aunque haya que verificar con la hoja de especificaciones que dicho diseño sea correcto. Sin embargo, no todos la poseen, por lo que se requiere realizar su diseño para poder generar así el esquemático. Para crear el componente se han seguido los siguientes pasos en la herramienta de dibujo 2D:

1. Dibujar el contorno del componente.
2. Crear los pines de cada componente.
3. Dar nombre e indicar el tipo de pin.
4. Seleccionar todo el componente.
5. Seleccionar a la opción *make device* con el botón derecho.
6. Asignar las características del componente:
 - a. Nombre completo.
 - b. Tipo de componente.
 - c. Asignarle una huella

Importante tener cuidado en la numeración de cada uno de los pines y hacerlo conforme a la información extraída del *datasheet* en cuestión.

2.6.3. Posicionado y rutado de componentes

El diseño del *Datalogger* se ha llevado a cabo en diferentes hojas *Proteus*, donde cada una de ellas tiene un nombre representativo en función de los componentes que contiene. Nos encontramos con la siguiente estructura de hojas (cada esquemático se corresponde con una PCB):

- PCB0 Power-USB.
- PCB1 FPGA.
- PCB2 Sensors.
- PCB3 Wireless.

En la hoja **PCB0 Power-USB** (Figura 14 - Figura 16) han posicionado la batería, el microcontrolador ARM, el interruptor de alimentación, un cristal de cuarzo, un puente USB-serie, leds y los diferentes conectores, así como el USB mini B.

En la segunda hoja **PCB1 FPGA** (Figura 17 - Figura 19) se han incluido la FPGA, un regulador lineal ajustable, una memoria *flash*, un oscilador integrado y los diferentes conectores necesarios.

En la segunda hoja **PCB2 Sensors** (Figura 20 - Figura 22) se han incorporado los diferentes sensores incorporados, así como sus circuitos de acondicionamiento, el GPS,

un convertidor ADC con interfaz SPI, un regulador conmutado síncrono *step-up* 5V, condensadores electrolíticos, un inductor y los diferentes conectores.

En la última hoja **PCB3 Wireless** (Figura 23 - Figura 24) se encuentran los diferentes transceptores, es decir, el transceptor LoRa y su alternativa, el transceptor WiFi y los conectores.

En todas las hojas mencionadas se han ido posicionando diferentes pines de test, los cuales nos permitirán verificar el correcto funcionamiento del proyecto cuando las placas de circuito impreso hayan sido fabricadas y montadas.

Finalmente, se realiza el conexionado de todos los componentes, prestando mucha atención para no confundir cables y poner nombres equivocados. Destacar que sobre la marcha, dado que es un proceso largo, se han ido detectando fallos y se han corregido antes de mandar a fabricar.

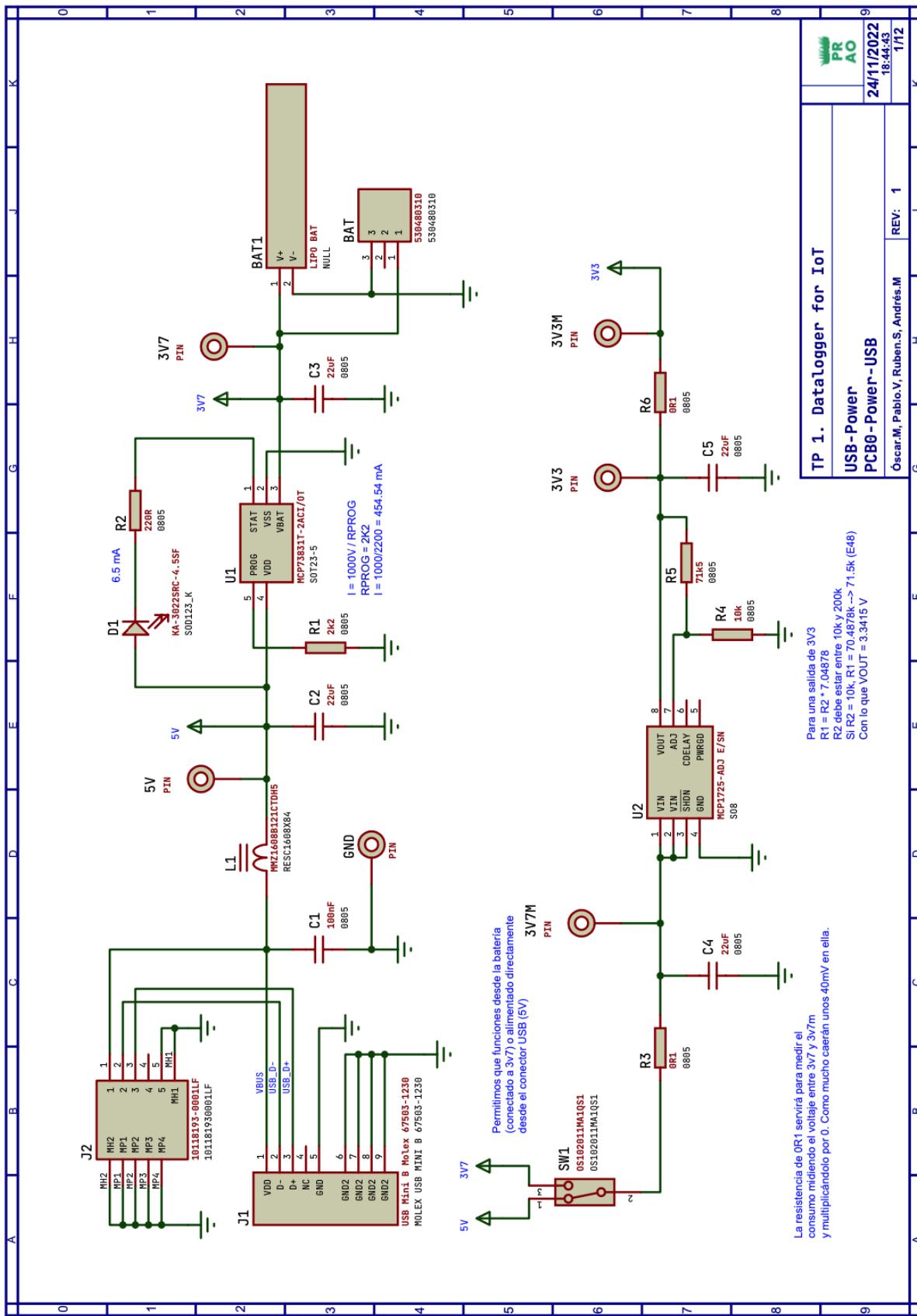


Figura 14. Esquemático de la placa PCB0 - Parte 1.

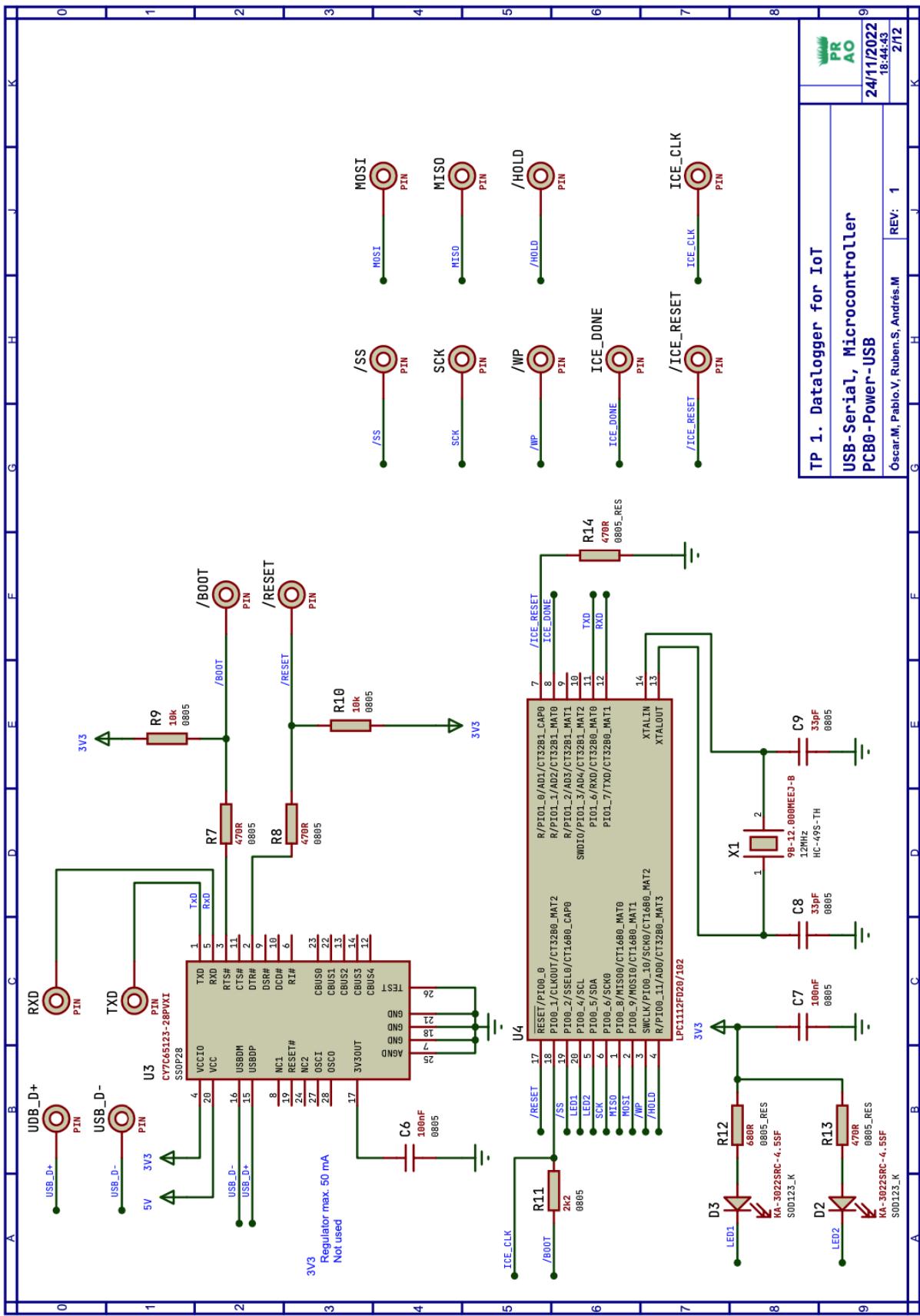


Figura 15. Esquemático de la placa PCB0 - Parte 2.

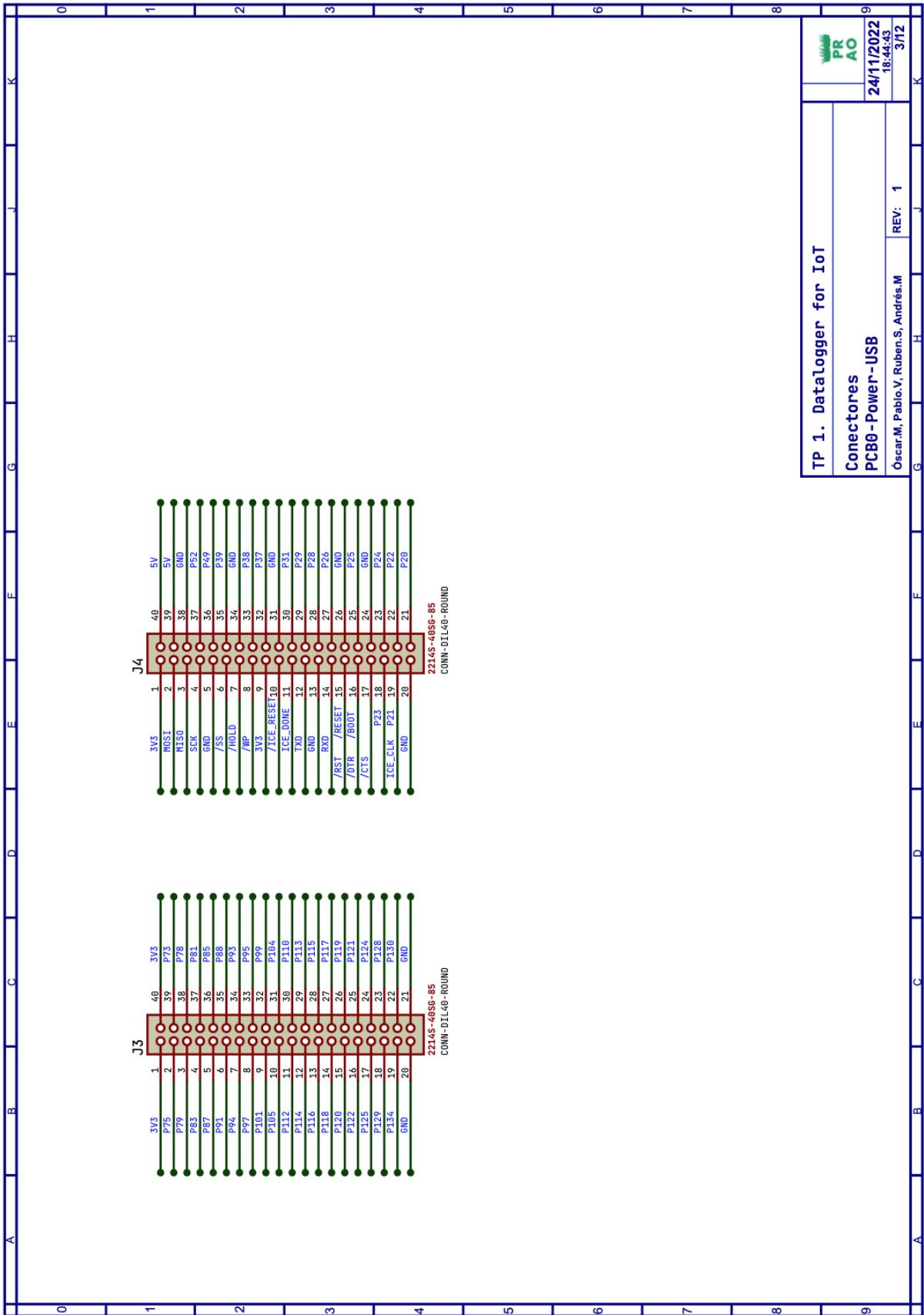


Figura 16. Esquemático de la placa PCB0 - Parte 3.

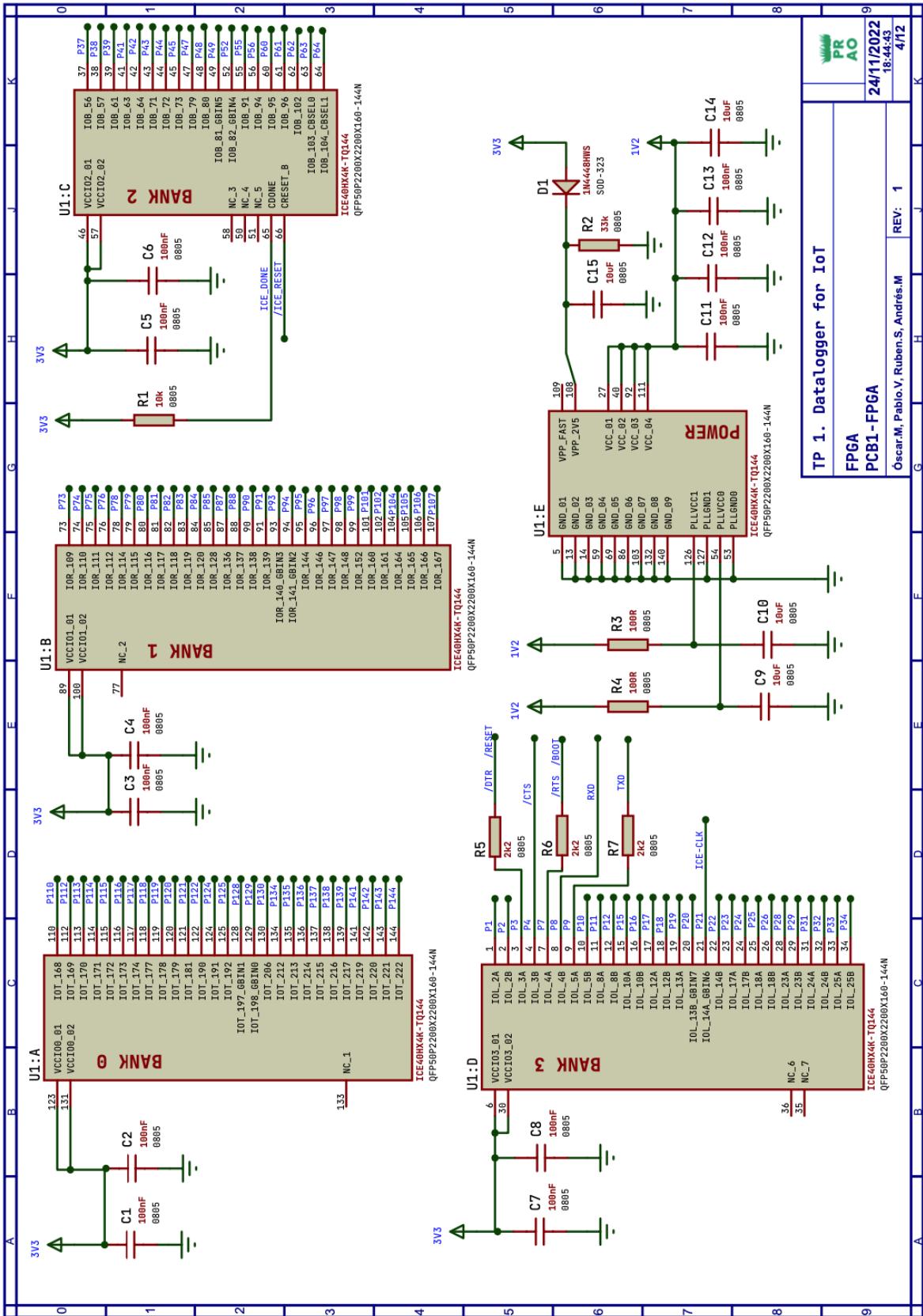


Figura 17. Esquemático de la placa PCB1 - Parte 1.

TP 1. Datalogger for IoT

FPGA
PCB1-FPGA

Oscar.M, Pablo.V, Ruben.S, Andres.M
REV: 1
24/11/2022
18:44:43
4/12

PR
AO

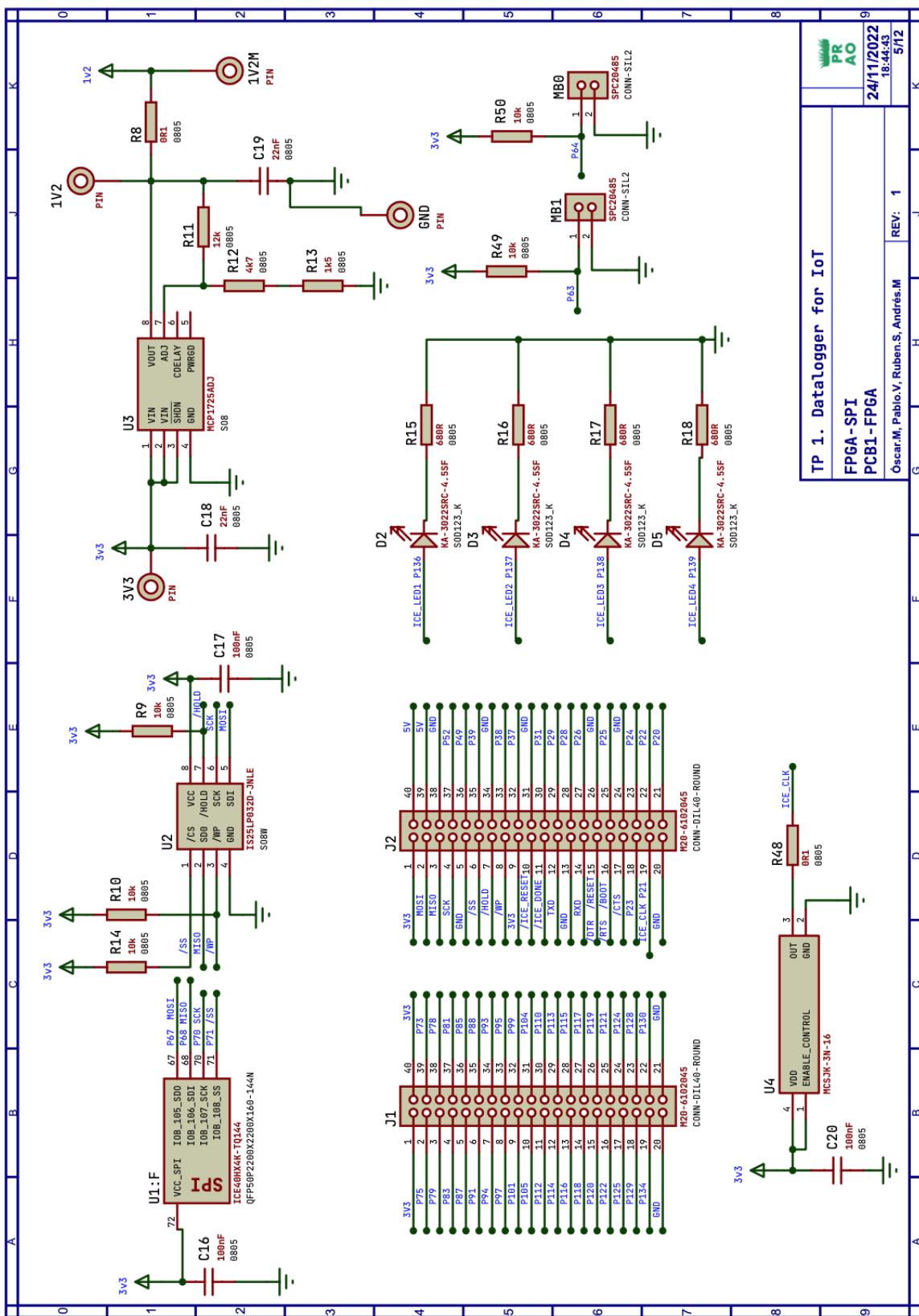


Figura 18. Esquemático de la placa PCB1 - Parte 2.

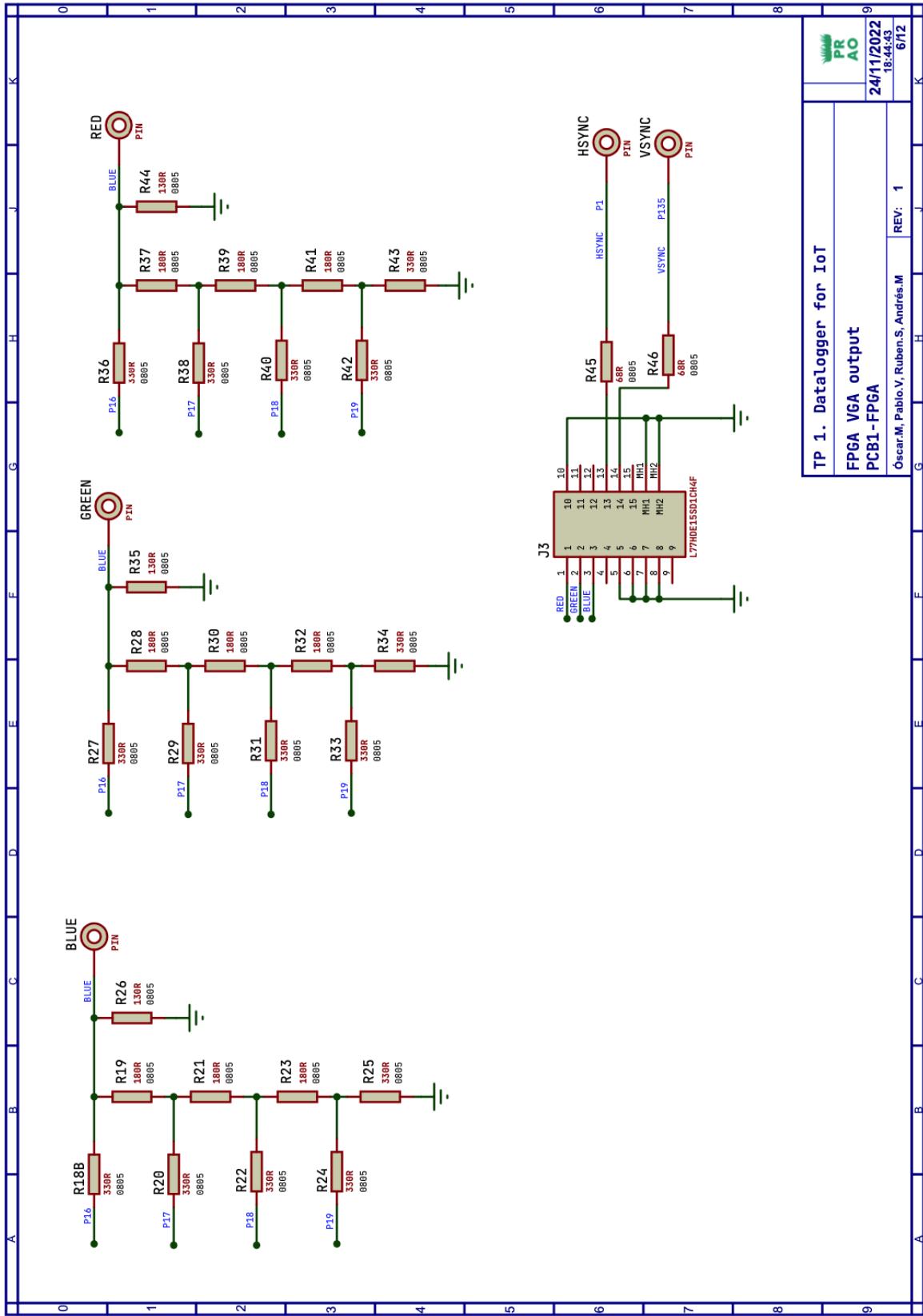


Figura 19. Esquemático de la placa PCB1 - Parte 3.

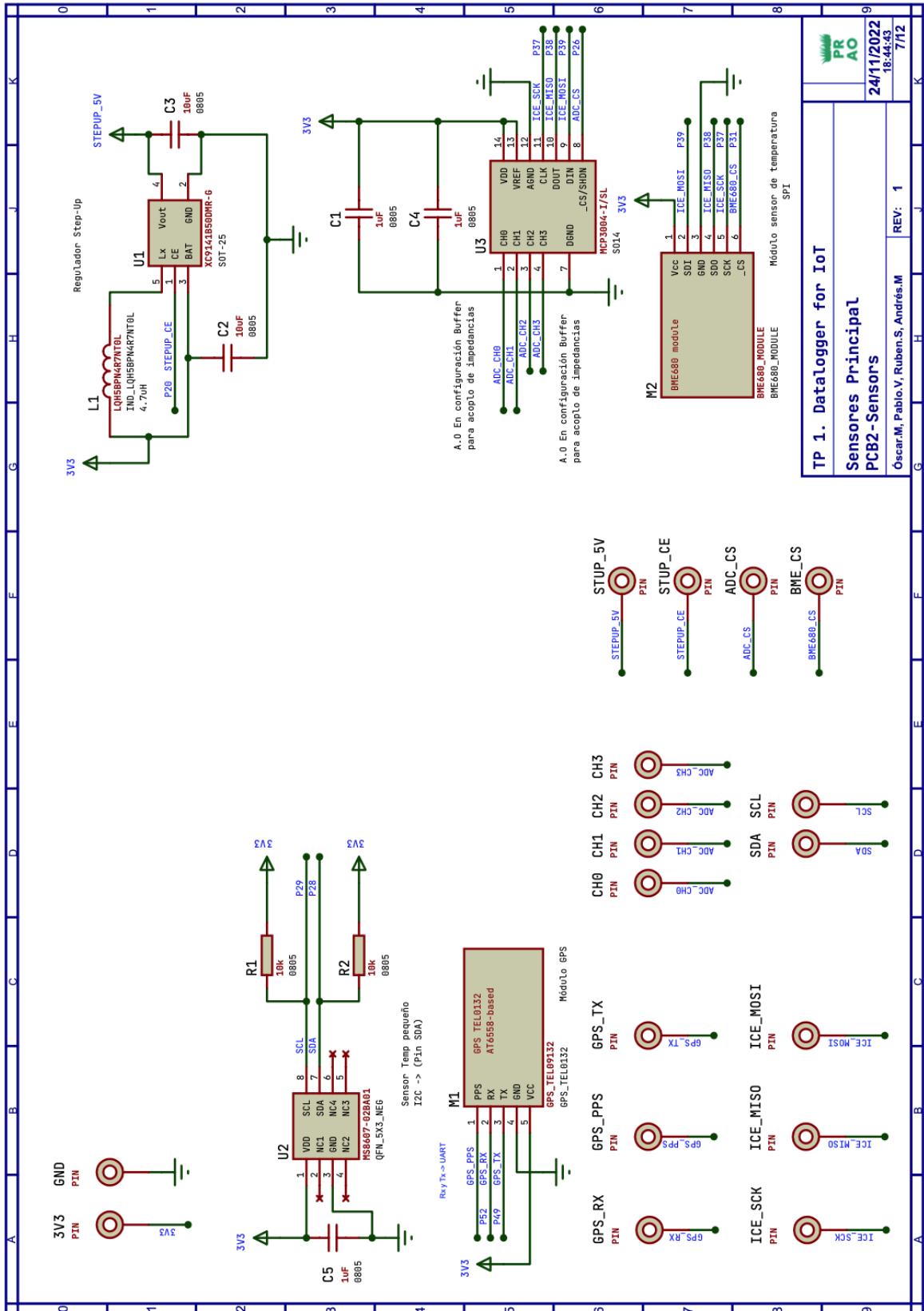


Figura 20. Esquemático de la placa PCB2 - Parte 1.

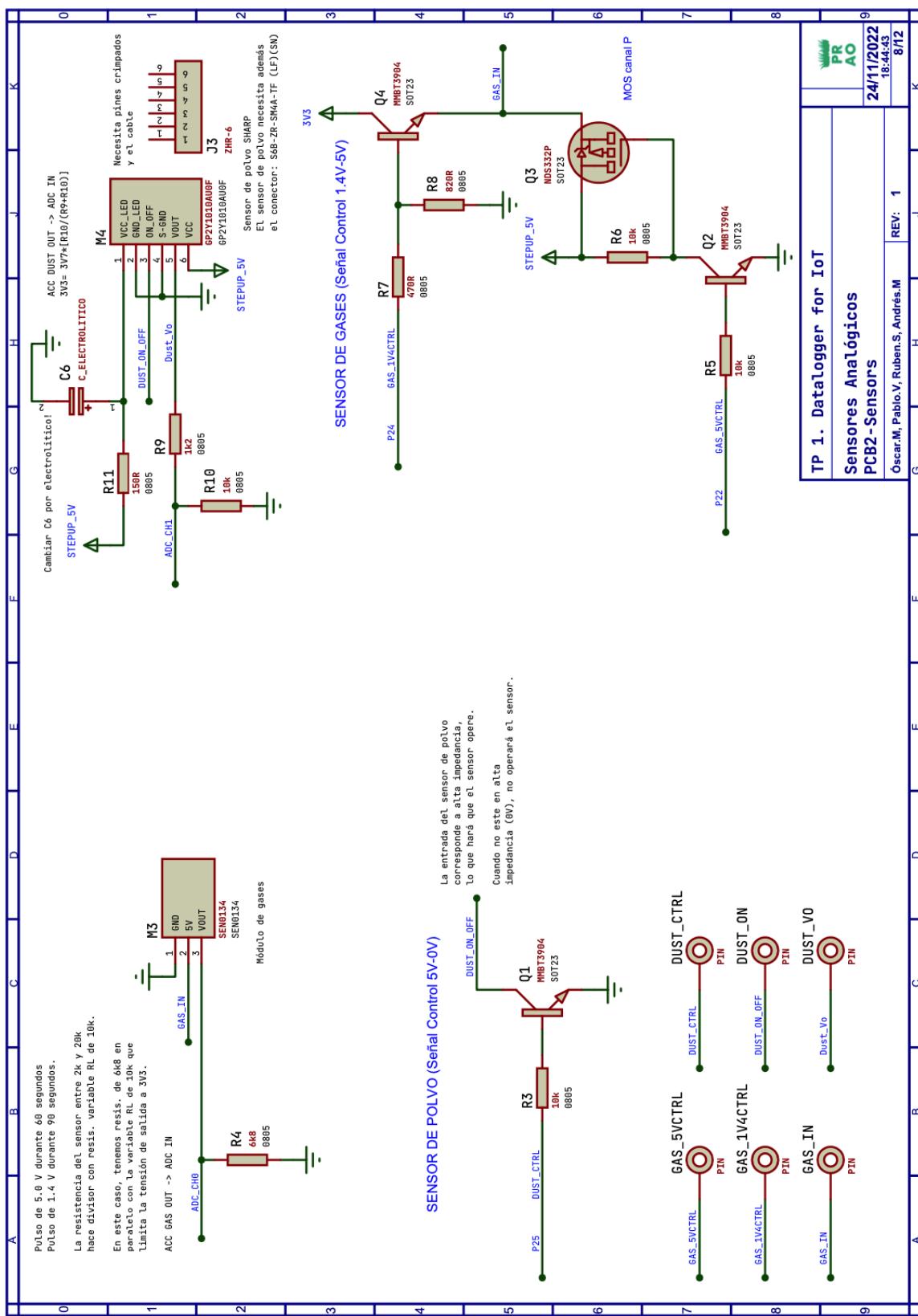


Figura 21. Esquemático de la placa PCB2 - Parte 2.

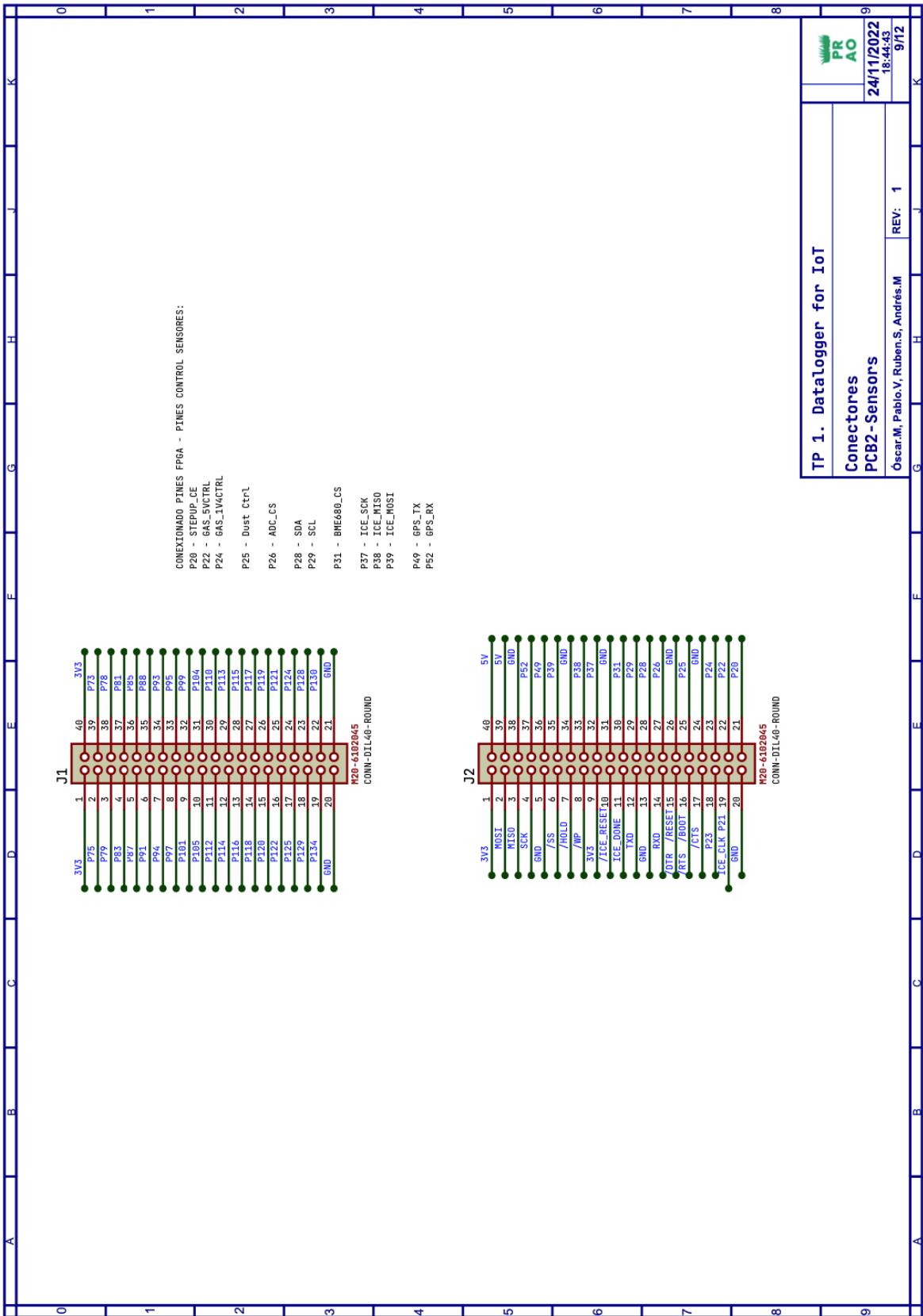


Figura 22. Esquemático de la placa PCB2 - Parte 3.

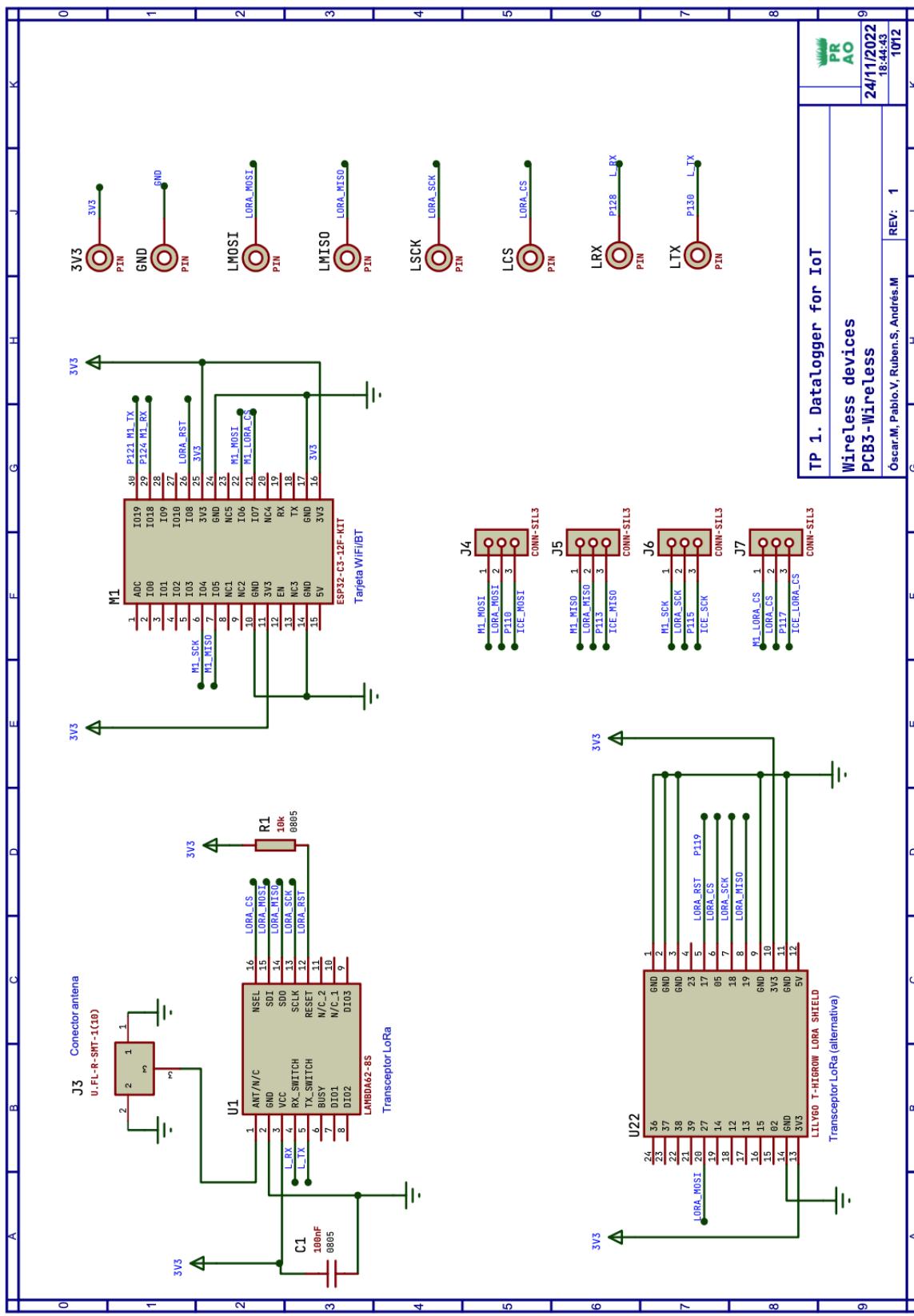


Figura 23. Esquemático de la placa PCB3 - Parte 1.

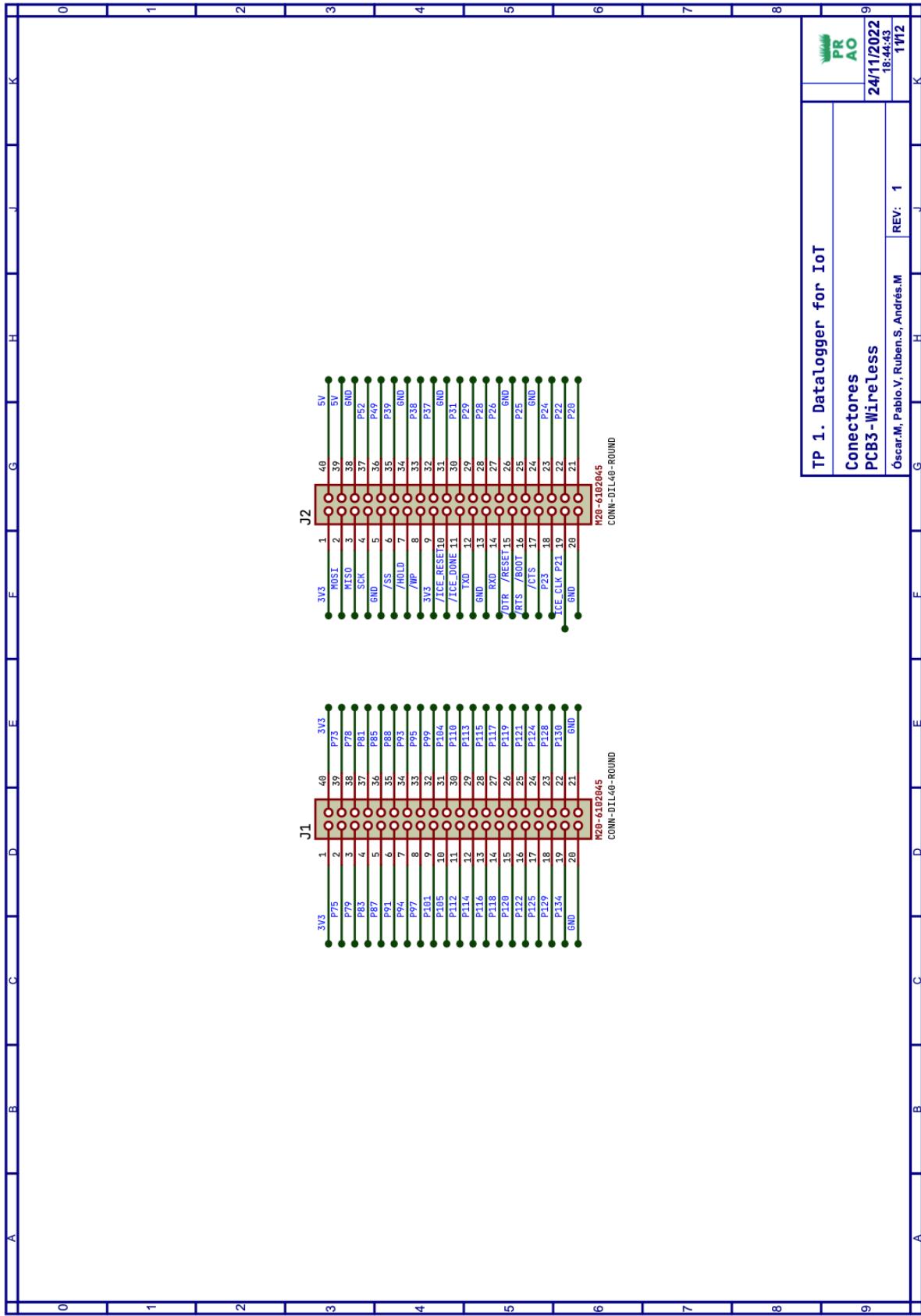


Figura 24. Esquemático de la placa PCB3 - Parte 2.

2.6.4. Generación de listado de materiales

Una vez seleccionados todos los componentes e insertados debidamente en el proyecto, de manera automática *Proteus* nos generará un listado también llamado *Bill of materials (BOM)* con todos ellos haciendo click sobre el icono de pdf. Asimismo, para tenerlos anotados correctamente como antes mencionamos, registramos su código de referencia asociado en la distribuidora *Farnell* y su precio, para así poder tener el presupuesto total.

Dicho listado de materiales requiere de una configuración previa de sus ajustes para poder ver el precio en unidades de euros y separar las décimas de céntimo mediante comas. Cada una de las placas diseñadas poseerá su propio listado, por ello nos encontramos con cuatro, los podemos observar a continuación. Nos encontramos con que la placa donde se encuentra el microcontrolador asciende a 57,36 € (Figura 25 - Figura 26), la placa que contiene la FPGA asciende a 32,48 € (Figura 27 - Figura 28), la placa correspondiente a los sensores posee el precio más elevado, debido al su elevado precio, con 83,45 € (Figura 29 - Figura 30) y la última placa destinada a las comunicaciones posee un precio de 46,13€ (Figura 31 - Figura 32).

Bill Of Materials for TP 1. Datalogger for IoT

Design Title	TP 1. Datalogger for IoT
Author	Oscar.M, Pablo.V, Ruben.S, Andrés.M
Document Number	1
Revision	1
Board	PCB0-Power-USB
Design Created	miércoles, 21 de septiembre de 2022
Design Last Modified	jueves, 20 de octubre de 2022
Total Parts In Design	60

Notes

Falta el IVA (21%)

0 Modules

9 Capacitors

Quantity	References	Value	Stock Code	Unit Cost
3	C1, C6-C7	100nF	1759037	0,10€
4	C2-C5	22uF	1657942	0,50€
2	C8-C9	33pF	317603	0,09€
Sub-totals:				
14 Resistors				
Quantity	References	Value	Stock Code	Unit Cost
2	R1, R11	2k2	1099803	0,08€
1	R2	220R	9240446	0,04€
2	R3, R6	0R1	8067708	0,38€
3	R4, R9-R10	10k	1174160	0,01€
1	R5	71k5	2059483	0,05€
4	R7-R8, RL3-R14	470R	9233334	0,01€
1	R12	680R	1653028	0,09€
Sub-totals:				
4 Integrated Circuits				
Quantity	References	Value	Stock Code	Unit Cost
1	U1	MCP73831T-2AC1/OT	1332158	0,76€
1	U2	MCP1725-ADJ E/SN	1834878	0,92€

Figura 25. Listado de materiales de la placa PCB0 - Parte 1.

1	U3		CY7C65123-28PVXI	2767866	1,89€
1	U4		LPC1112FD20/102	771-	30,20€
Sub-totals :					
0 Transistors					
Quantity	References	Value	Stock Code	Unit Cost	
				0,00€	
3 Diodes					
Quantity	References	Value	Stock Code	Unit Cost	
3	D1-D3	KA-3022SRC-4.5SF	1142615	0,52€	
Sub-totals :					
30 Miscellaneous					
Quantity	References	Value	Stock Code	Unit Cost	
21	/BOOT, /HOLD, /ICE_RESET, /RESET, /SS, /WP, 3V3, 3V7, 3V3M, 3V7M, 5V, GND, ICE_CLK, ICE_DONE, PIN MISO, MOSI, RXD, SCK, TXD, UDB_D+, USB_D-	53048310	9732918	0,35€	
1	BAT	LIPO BAT	2819236	13,48€	
1	BAT1	USB Mini B Molex	2313554	1,16€	
1	J1	67503-1230			
1	J2	10118193-0001LF	2751675	0,30€	
2	J3-J4	2214S-40SG-85	2847248	1,11€	
1	L1	MMZ1608B121CTDH5	3386884	0,08€	
1	SW1	OS102011MA1QS1	2435103	0,52€	
1	X1	9B-12.000MEEJ-B	2508484	0,27€	
Sub-totals :					
Totals:					
57,36€					

jueves, 20 de octubre de 2022 21:22:08



Figura 26. Listado de materiales de la placa PCB0 - Parte 2.

Bill Of Materials for TP 1. Datalogger for IoT

Design Title	TP 1. Datalogger for IoT
Author	Óscar.M, Pablo.V, Rubén.S, Andrés.M
Document Number	1
Revision	1
Board	PCB1-FPGA
Design Created	miércoles, 21 de septiembre de 2022
Design Last Modified	jueves, 20 de octubre de 2022
Total Parts In Design	93

Notes

Falta el IVA (21%)

0 Modules		Quantity	References	Value	Stock Code	Unit Cost
Sub-totals:						
20 Capacitors						
Quantity	References			Value	Stock Code	Unit Cost
14	C1-C8,C11-C13,C16-C17,C20			100nF	2300839	0,10€
4	C9-C10,C14-C15			10μF	24098565	0,04€
2	C18-C19			22nF	2310684	0,01€
Sub-totals:						
50 Resistors						
Quantity	References			Value	Stock Code	Unit Cost
6	R1,R9-R10,R14,R49-R50			10k	3226745	0,05€
1	R2			33k	3133469	0,01€
2	R3-R4			100R	1652906	0,14€
3	R5-R7			2k2	1469887	0,07€
2	R8,R48			0R1	8067708	0,40€
1	R11			12k	1469865	0,06€
1	R12			4k7	1469923	0,75€
1	R13			1k5	1652940	0,09€
4	R15-R18			680R	1653028	0,09€
9	R19,R21,R23,R28,R30,R32,R37,R39,R41			180R	1652931	0,07€
15	R20,R22,R24-R25,R27,R29,R31,R33-R34,R36,R38,R40,R42-R43,R18B			330R	1469918	0,09€

Figura 27. Listado de materiales de la placa PCB1 - Parte 1.

3	R26, R35, R44	130R	1692530	0,07€
2	R45-R46	68R	2138823	0,07€
Sub-totals:				
4 Integrated Circuits				
Quantity	References	Value	Stock Code	Unit Cost
1	U1	ICE40HX4K-TQ144	3768754	10,66€
1	U2	IS25LP32D-TNUE	2787053	1,29€
1	U3	MCP1725ADJ	1851961	0,69€
1	U4	MCSJK-3N-16	2854286	0,62€
Sub-totals:				
5 Transistors				
Quantity	References	Value	Stock Code	Unit Cost
Sub-totals:				
6 Diodes				
Quantity	References	Value	Stock Code	Unit Cost
1	D1	1N4448HWS	3127174	0,21€
4	D2-D5	KA-3022SRC-4-5SF	1142615	0,52€
Sub-totals:				
14 Miscellaneous				
Quantity	References	Value	Stock Code	Unit Cost
9	1V2, 1V2M, 3V3, BLUE, GND, GREEN, HSYNC, RED, VSYNC	PIN		
2	J1-J2	M20-6102045	1569230	4,75€
1	J3	L77HDE15SD1CH4F	2401182	1,28€
2	MB0-MB1	SPC20485	1258956	0,14€
Sub-totals:				
Totals:				
			32,48€	

Figura 28. Listado de materiales de la placa PCBI - Parte 2.

jueves, 20 de octubre de 2022 21:22:27



Bill Of Materials for TP 1. Datalogger for IoT

Design Title	TP 1. Datalogger for IoT
Author	Óscar.M, Pablo.V, Rubén.S, Andrés.M
Document Number	1
Revision	1
Board	PCB2-Sensors
Design Created	miércoles, 21 de septiembre de 2022
Design Last Modified	miércoles, 23 de noviembre de 2022
Total Parts In Design	56

Notes

Falta el IVA (21%)

4 Modules

Quantity	References	Value	Stock Code	Unit Cost
1	M1	GPS_TEL09132	3769990	17,75€
1	M2	BME680_MODULE	3932094	22,41€
1	M3	SEN0134	2946118	7,44€
1	M4	GP2Y1010AU0F	9707956	9,50€
Sub-totals:				
				57,10€
6 Capacitors				
Quantity	References	Value	Stock Code	Unit Cost
3	C1,C4-C5	1uF	2280849	0,14€
2	C2-C3	10uF	23220887	0,83€
1	C6	220uF	2494472	0,79€
Sub-totals:				
				2,86€
11 Resistors				
Quantity	References	Value	Stock Code	Unit Cost
6	R1-R3,R5-R6,R10	10k	1174160	0,01€
1	R4	6k8	1469831	0,04€
1	R7	470R	9233334	0,01€
1	R8	820R	1576627	0,08€
1	R9	3133435		0,04€

Figura 29. Listado de materiales de la placa PCB2 - Parte 1.

1	R11		150R	9240896	0,05€
	Sub-totals:				0,26€
3 Integrated Circuits					
Quantity	References		Value	Stock	Unit Cost
1	U1		XC9141B50DMR-G	3535653	1,12€
1	U2		MS8607-02BA01	2748850	7,11€
1	U3		MCP3004-I/SL	1852016	2,78€
	Sub-totals:				11,01€
4 Transistors					
Quantity	References		Value	Stock	Unit Cost
3	Q1-Q2,Q4		MMBT3904-7-F	1773602	0,13€
1	Q3		NDS332P	1471069	0,41€
	Sub-totals:				0,80€
0 Diodes					
Quantity	References		Value	Stock	Unit Cost
	Sub-totals:				0,00€
28 Miscellaneous					
Quantity	References		Value	Stock	Unit Cost
24	3V3,ADC_CS,BME_CS,CHO-CH3,DUST_CTRL,DUST_ON,DUST_VO,GAS_1V4CTRL,GAS_SVCTRL,GAS_IN,GND, GPS_PPS,GPSS_RX,GPSS_TX,ICE_MISO,ICE_MOSI,ICE_SCK,SDA,SDA_STUP_SV,STUP_CE		PIN		
2	J1-J2		M20-6102045	1569230	5,51€
1	J3		ZHR-6	3357570	0,10€
1	L1		LQH5BPN4R7NT0L	2219307	0,31€
	Sub-totals:				11,42€
	Totals:				83,45€

Figura 30. Listado de materiales de la placa PCB2 - Parte 2.



Bill Of Materials for TP 1. Datalogger for IoT

Design Title	TP 1. Datalogger for IoT			
Author	Óscar.M, Pablo.V, Rubén.S, Andrés.M			
Document Number	1			
Revision	1			
Board	PCB3-Wireless			
Design Created	miércoles, 21 de septiembre de 2022			
Design Last Modified	miércoles, 23 de noviembre de 2022			
Total Parts In Design	6			
Notes	Falta el IVA (21%)			
1 Modules				
Quantity	References	Value	Stock Code	Unit Cost
1	M1	ESP32-C3-12F-KIT	3932671	9,16€ 9,16€
Sub-totals:				
0 Capacitors				
Quantity	References	Value	Stock Code	Unit Cost
0				0,00€
Sub-totals:				
0 Resistors				
Quantity	References	Value	Stock Code	Unit Cost
0				0,00€
Sub-totals:				
2 Integrated Circuits				
Quantity	References	Value	Stock Code	Unit Cost
1	U1	LAMBDA62-8S	2988568	14,98€
1	U22	LILYGO T-HIGROW LORA SHIELD	TTGO t-higrow LoRa Shield	9,77€ 24,75€
Sub-totals:				
0 Transistors				
Quantity	References	Value	Stock Code	Unit Cost
0				0,00€
Sub-totals:				
0 Diodes				
Quantity	References	Value	Stock Code	Unit Cost
0				0,00€
Sub-totals:				
3 Miscellaneous				
Quantity	References	Value	Stock Code	Unit Cost

Figura 31. Listado de materiales de la placa PCB3 - Parte 1.

1	J1	U.FL-R-SMT-1 (10)	1688077	1,20€
2	J2-J3	M20-6102045	1569230	5,51€
Sub-totals:				
				12,22€

Totals:	46,13€
---------	--------



miércoles, 23 de noviembre de 2022 21:11:56

Figura 32. Listado de materiales de la placa PCB3 - Parte 2.

2.6.5. Análisis de consumo eléctrico

El objetivo es realizar una lista con todos los consumos eléctricos de los componentes clasificándolos mediante tensiones, es decir, contabilizar cuál es la corriente máxima consumida para cada uno de ellos en el peor caso.

Este proceso se ha llevado a cabo a través de una serie de tablas, donde para cada tensión de alimentación hemos evaluado los distintos componentes que tiene conectados. Para cada elemento se ha evaluado su máximo consumo y las condiciones en las que sucede, así como la ubicación de donde se encuentra dicha información en su hoja de especificaciones. Asimismo, se ha anotado cuál sería el consumo total para cada una de las cuatro placas, esta información la podemos observar desde la **¡Error! No se encuentra el origen de la referencia.** a la **¡Error! No se encuentra el origen de la referencia..**

Además, para alguno de los componentes se ha anotado la corriente quiescente o *quiescent current*, es decir, la cantidad de corriente utilizada cuando está en un estado quiescente. Este estado se da en cualquier periodo de tiempo en el que el circuito integrado está habilitado, pero sin actividad.

Nos encontramos entonces en disposición de calcular cuál sería la duración teórica de la batería recargable de polímero de litio de 1000 mAh. Se ha obtenido que el consumo aproximado será de 1433,2 mA, por lo que utilizando la siguiente fórmula:

$$I(\text{mA}) * h(\text{horas}) = CE(\text{mAh})$$

Tenemos que la duración aproximada de la batería será de 0,7 horas, o lo que es lo mismo, una duración de 42 minutos en el peor de los casos.

Sin embargo, los cálculos teóricos difieren del consumo real que se ha obtenido. A través de los pines 3V3 y 3V3M que se encuentran en la placa PCB0 hemos medido el voltaje consumido que cae en la resistencia de 0,1 ohmios. Se han analizado distintas situaciones, en primer lugar, el consumo obtenido con la alimentación a través del USB, después activando el *step-up*, transmitiendo a través del LoRa y activando los sensores de polvo y gas.

Ley de Ohm mediante, el consumo obtenido del *datalogger* estando únicamente alimentado a través del conector USB ha sido de 90 mA. Comenzando por activar el *step-up* se ha obtenido un consumo de 107 mA, los sensores de polvo y gas han presentado un consumo de 128 mA y 379 mA respectivamente. Por último, la transmisión por LoRa ha presentado un consumo 195 mA.

Dado que no tenemos todas las cosas a la vez habilitadas, entre las posibilidades que se nos presentan, en la aquella que hemos conseguido tener el mayor consumo de todos ha sido mediante la habilitación del sensor de gas junto con la transmisión por LoRa, presentando un consumo de 493 mA. Como observamos estos valores difieren mucho del peor valor teórico posible. En estas circunstancias la duración aproximada de la batería será de 2 horas aproximadamente.

Análisis Eléctrico					
	Nombre	Consumo (mA)	Quiescent current (mA)	Componentes a 5V Condiciones	Ubicación
Puente USB-Serie C7765213-28PVK1		13	-	max 16 mA	Página 9. Datasheet
Regulador de tensión MCP1725 (3V7)		-	0,22	máx 500 mA	Página 5. Datasheet
Componentes a 3V7					
	Nombre	Consumo (mA)	Quiescent current (mA)	Condiciones	Ubicación
Cargador Batería MCP73831T-2AC1OT		1,5	-	Cargando batería	Página 3. Datasheet
Componentes a 3V3					
	Nombre	Consumo (mA)	Quiescent current (mA)	Condiciones	Ubicación
LED rojo		30	-	a 25°C	Página 2 Datasheet
LED verde		30	-	a 25°C	Página 2 Datasheet
Microcontrolador ARM LPC1112FD20102/52		100	-	por cada pin de alimentación (hay 1)	Página 5s. Datasheet
Consumo total					174,5

Figura 33. Análisis eléctrico PCB.

Análisis Eléctrico					
Nombre	Consumo (mA)	Componentes a 3V3	Quiescent current (mA)	Condiciones	Ubicación
Memoria NOR Flash IS25LP032D-JNLE	25	-	-	-	Página 99. Datasheet
Regulador lineal MCP1725-ADJ E/S/N	500	0,22	0,22	a 25 °C	Página 5. Datasheet
FPGA CEC401P-HX	22,3	-	-	-	Página 19. Datasheet
Diodo IN4448BSW-7-F	60,6	-	-	-	Página 1. Datasheet
Oscilador integrado MCSIK-3N-16.00.3.3-50-B	15	-	-	-	Página 1. Datasheet
LED KA-3022SRC-4.5SF (x4)	30 x 4 = 120	30 x 4 = 120	30 x 4 = 120	a 25°C	Página 2. Datasheet
Consumo total				742,9	

Figura 34. Análisis eléctrico PCB1.

Análisis Eléctrico					
Nombre	Consumo (mA)	Componentes a 3V3	Condiciones	Ubicación	Notas
Sensor H/T/P BME680_MODULE	-	Quiescent current (mA)	-	Página 2 Datasheet	El consumo de corriente corresponde al pico máximo con el que se va a encontrar el sensor, que durante la medición de Temperatura o de Presión
Sensor H/T/P MS8607-02BA01	1,25	-	-	Página 10. Datasheet	La corriente quiescente se encuentra en la página 7, el consumo en la página 9.
GPS TEL09132	25	-	-	Página 9. Datasheet	
Regulador Comutado Síncrono XC9141BS0DMR-G	1,04	0,04	a 25°C	Página 4. Datasheet	
Convertidor ADC con interfaz SPI MCP3004-I_SI	0,55	-	a 25°C		
Componentes a 5V					
Nombre	Consumo (mA)	Componentes a 5V	Condiciones	Ubicación	Notas
Sensor de Polvo GP2Y1010AUOF	20	Quiescent current (mA)	a 25°C	Página 4. Datasheet	
Sensor de Gases SEN0134	-	-	-		
Consumo total	47,84				

Figura 35. Análisis eléctrico PCB2.

Análisis Eléctrico					
Nombre	Consumo (mA)	Componentes a 3V3 Quiescent current (mA)	Condiciones	Ubicación	Notas
Transceptor radio LoRa LAMBDAg2-8D	118	-	A frecuencia 868 MHz a 25°C	Página 7. Datasheet Página 5. Datasheet	Corresponde con el peor caso entre todos los que hay, es decir, cuando la potencia de salida son +22dBm.
Transceptor WiFi_BT_LTE (Plan B)	350	-	-	-	Corresponde con el peor caso, que será transmitiendo en 802.11b con modulación CCK a una tasa de 1Mbps. Si modulásemos con OFDM o MC7 el consumo sería menor.
Transceptor LoRa LILYGO T-Higrow Shield	-	-	-	-	Carece de Datasheet de donde extraer la información
Consumo total	468				

Figura 36. Análisis eléctrico PCB3.

2.6.6. Análisis de consumo térmico

Una vez realizado el análisis eléctrico de las cuatro placas, nos interesa conocer cuál será el calentamiento aproximado de los componentes electrónicos, es decir, poder controlar que los semiconductores no varíen sus características técnicas y obtener una mayor longevidad de los componentes.

Para proceder con este análisis se ha calculado cuál es la potencia consumida por cada componente, las condiciones de temperatura en las que sucede, así como la ubicación en la hoja de características de dicha información. Asimismo, se ha calculado la resistencia térmica para los componentes que la precisan. Podemos encontrar dichos cálculos en formato de tabla desde la **¡Error! No se encuentra el origen de la referencia.** hasta la Figura 39.

Para llevarlo a cabo el cálculo de la potencia se ha utilizado la siguiente fórmula, en la que tendremos en cuenta cuál es la corriente que consume el elemento y la alimentación que éste posee.

$$P(W) = V(V) * I(A)$$

Calculada dicha potencia, ha servido para calcular las distintas resistencias térmicas mediante la siguiente fórmula, en la que se ha tenido en cuenta la temperatura ambiente.

$$\text{Temperatura} = \text{Potencia} * \Theta(JA) + \text{Temp. ambiente}$$

Análisis Térmico					
Nombre	Potencia (mW) = I * V	Resistencia térmica	Temperatura	Ubicación	Notas
Cargador Batería MCP73831T2ACOT	5,55	230°C/W	-	Página 5. Datasheet	
Regulador de tensión MCP1725 (3V7)	1,1	Theta(jA) = 169°C/W	0,132682 9C+Ta	Página 7. Datasheet	
Puente USB-Serie CY7C65213-28PVI	65	62°C/W	-	Página 24. Datasheet	THJ: Package 28pin SSOP
LED rojo	75	-	-	Página 2. Datasheet	
LED verde	75	-	-	Página 2. Datasheet	
Microcontrolador ARM LPC1112F020102/52	330	-	-	Página 58. Datasheet	

Figura 37. Análisis térmico PCB0.

Análisis Térmico					
Nombre	Potencia (mW) = $I * V$	Resistencia térmica	Temperatura	Ubicación	Notas
Memoria NOR Flash IS25LP032D-INLE Regulador lineal MCP1725-ADJ/SN	82,5 1650	- Thetal(A)=163 °C/W	- 0,132682°C + Ta	Página 99. Datasheet Página 7. Datasheet	
FPGA ICE40 LP/HX	73,59	-	-	Página 19. Datasheet	
Diodo 1N4448ISW-7-f	200	Thetal(A)=625 °C/W	-	Página 1. Datasheet	
Oscilador integrado MC14468-16.00-3.3-50-B LED KA-3022SRC-4.5SF (x4)	49,5 $75 \times 4 = 300$	- -	-	Página 1. Datasheet Página 2. Datasheet	

Figura 38. Análisis térmico PCB1.

Análisis Térmico					
Nombre	Potencia (mW) = I * V	Resistencia térmica	Temperatura	Ubicación	Notas
Sensor H/T/P BME680_MODULE	-	-	-	-	Página 2. Datasheet
Sensor H/T/P MS8607_02BA01	4,125	-	-	-	Página 10. Datasheet
GPT-EL09132	82,5	-	-	-	Página 9. Datasheet
Regulador Commutado Síncrono XC9141B50DMR_G	3,43	-	-	-	Página 4. Datasheet
Convertidor ADC con interfaz SPI MCP3004-I_SL	1,81	Theta(J/A) = 108°C/W	-	-	Página 4. Datasheet
Sensor de Polvo GP2Y10AU0F	100	-	-	-	

Figura 39. Análisis térmico PCB2.

Análisis Térmico					
Nombre	Potencia (mW) = I^*V	Resistencia térmica	Temperatura	Ubicación	Notas
Transceptor radio LoRa LAMRDA6280	389,4	-	-	Página 7. Datasheet	A frecuencia 868 MHz y potencia de salida +22dBm
Transceptor WiFi BT LE (Plan B)	1155	-	a 25°C	Página 5. Datasheet	En 802.11b con modulación CCX y tasa de 1Mbps.
Transceptor LoRa LLYC01-Higrow Shield	-	-	-	-	Carece de Datasheet de donde extraer la información

Figura 40. Análisis térmico PCB3.

2.7. Diseño de la placa de circuito impreso

Tras la realización del diseño electrónico toma lugar el diseño de la placa de circuito impreso. Para llevar a cabo esta tarea se requiere que cada componente del *Datalogger* posea una huella o una representación en 2D de su tamaño y conexiones, además habrá que añadir los planos de masa y de dissipación térmica. Finalmente, con la ayuda de la herramienta *Autorouter* del *Layout* de *Proteus* se realizan las conexiones.

Con el diseño de la PCB completado bastará con generar los ficheros Gerber para enviárselos al fabricante. A continuación, detallaremos todos y cada uno de estos pasos.

2.7.1. Creación de huellas de nuevos componentes

Dado que algunos de los componentes no poseen una huella asociada e inclusive no es posible encontrarla en alguna de las librerías que *Proteus* pone a nuestra disposición, es necesario crear las huellas de estos. En ese caso, con la ayuda de la hoja de especificaciones del componente debemos crear la huella atendiendo a las dimensiones que éste posee.

Es importante tener en cuenta que algunas de estas hojas de especificaciones hacen referencia a una familia entera de componentes, entre los cuales pueden variar sus dimensiones. Por ello, es necesario identificar cuáles son las dimensiones exactas del componente en cuestión.

Por último, a modo de ejemplo, fue necesario crear una huella para el sensor de polvo GP2Y1010AU0F alojada en la placa PCB2 atendiendo a las dimensiones, éstas las podemos ver en la siguiente figura.

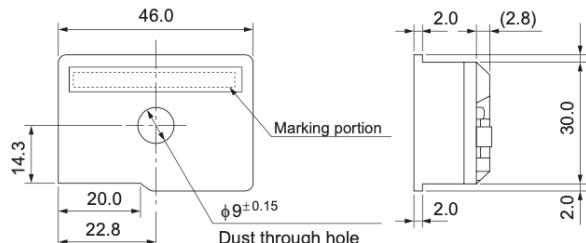


Figura 41. Dimensiones del sensor de polvo extraídas de la hoja de especificaciones.

Mediante dichas dimensiones dadas en milímetros, procedimos a la creación de su huella, la cual podemos observar en la **!Error! No se encuentra el origen de la referencia.**41.

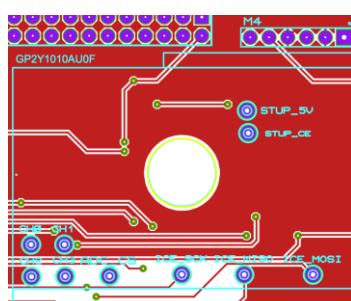


Figura 42. Huella sobre la PCB del sensor de polvo GP2Y1010AU0F.

2.7.2. Definición de borde y zonas de conectores

Es de vital importancia tener en cuenta en qué posición y lugar se encontrarán los conectores entre sucesivas placas, así como las limitaciones de espacio entre placas de circuito impreso. Por ello, a partir de una huella general que engloba ambas cosas procedemos a diseñar la placa. A continuación, en la **¡Error! No se encuentra el origen de la referencia.**, observamos dicha huella tomada como base del proyecto.

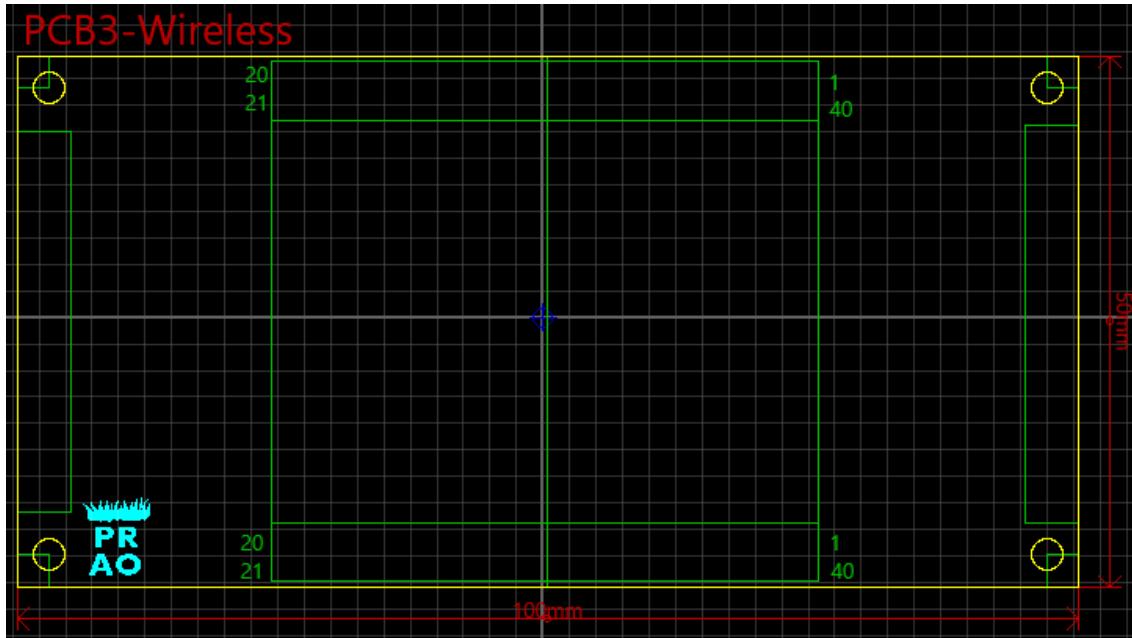


Figura 43. Huella de placa de circuito impreso y conectores entre placas.

2.7.3. Posicionado de componentes

Fruto del pequeño tamaño que poseen nuestras placas de circuito impreso, que son de 10x5 cm, es necesario posicionar cada elemento de manera ordenada. De esta forma, seremos capaces de conseguir adentrar todos los componentes dentro de ellas, dado que son muchas las huellas a incluir.

Además, hay que procurar que los componentes que están conectados entre sí estén lo más cerca posible, para que las pistas sean más cortas en el momento de hacer el rutado. Teniendo estos factores en cuenta se han diseñado las placas, éstas las encontramos en la sección referente al rutado desde la Figura 43 a 46 de la PCB0 a la PCB3 respectivamente.

Cabe destacar que la placa se ha personalizado añadiendo en la serigrafía el logotipo del equipo, que contiene las iniciales de cada uno de los integrantes.

2.7.4. Planos de masa y disipación térmica

Previo al rutado de los componentes es imprescindible añadir los planos de masa y disipación térmica. Su razón principal es evitar la interferencia que pueden ocasionar las señales entre distintas pistas. El objetivo es conseguir el máximo rendimiento en nuestro dispositivo. Los planos de disipación térmica sirven para disipar el calor generado por los componentes, el cual ya mencionamos en la sección del análisis térmico. Los planos de masa se pueden añadir desde el menú PCB en Tools/Power Plane Generator.

2.7.5. Rutado de conexiones

Para llevar a cabo el rutado de las conexiones podemos proceder de dos maneras diferentes, haciéndolo de manera manual o de forma automatizada haciendo uso de la herramienta *Autorouter* de *Proteus*. Dado que puede ser muy tedioso hacerlo de manera manual nosotros hemos optado por hacer un autorutado y corregir entonces los errores que pudieran surgir. Se puede acceder al autorutado haciendo click en el icono de cables cruzados/ autoruter. Es interesante comprobar antes el design rule manager en el icono de una escuadra y un lápiz.

Errores que pueden surgir pueden ser pistas que pisen la serigrafía de la huella de un componente, también *through holes* o agujeros pasantes que se autoposicionen justo debajo de la huella de un componente y no nos permita acceder a ellos en caso de que sea necesario.

Podemos visualizar todas las placas ya diseñadas con rutado incluido desde la Figura 43 a la 46, en éstas nos encontramos el *layout* del anverso, del reverso y del *mech*.

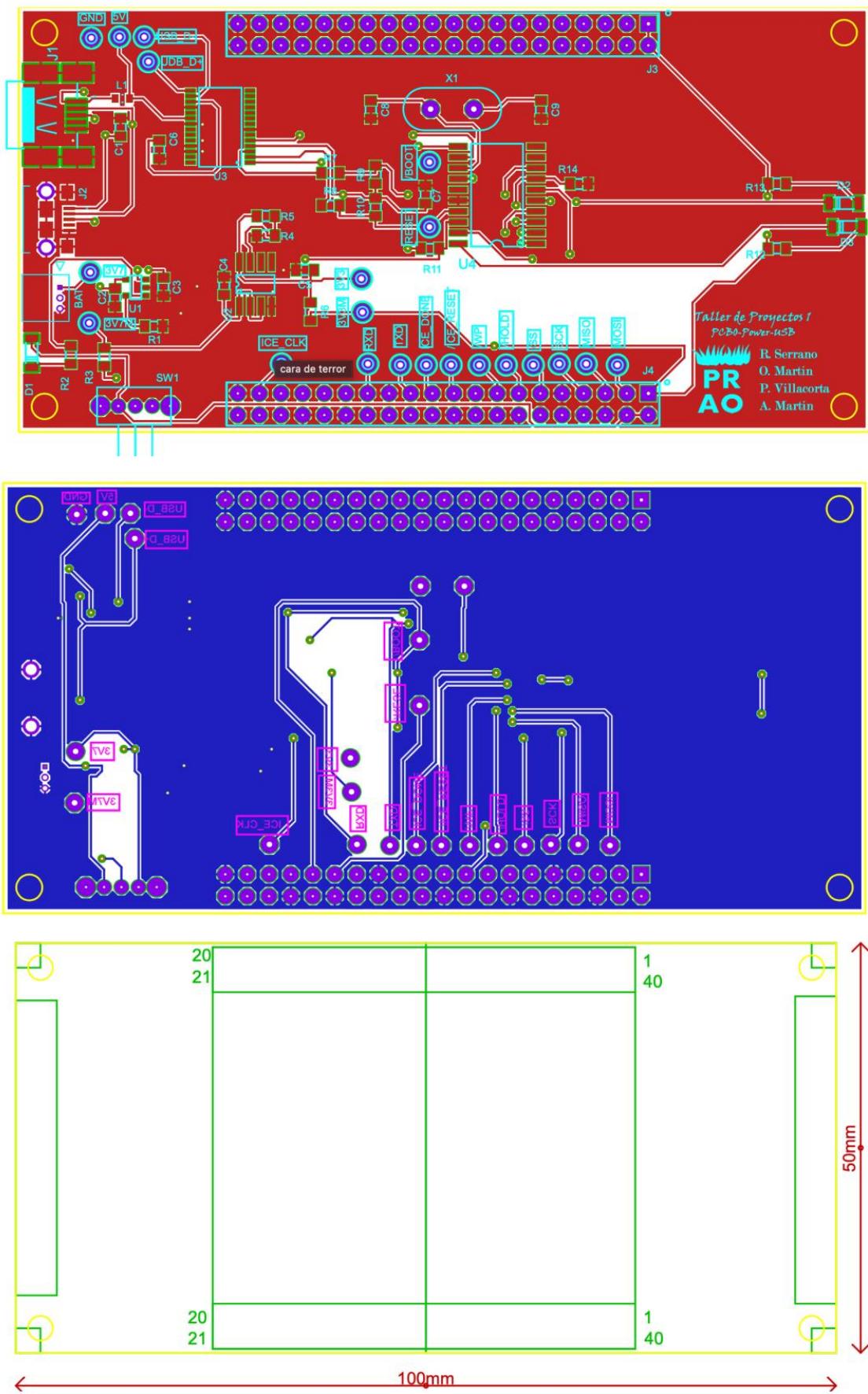


Figura 44. Layouts de la placa PCB0.

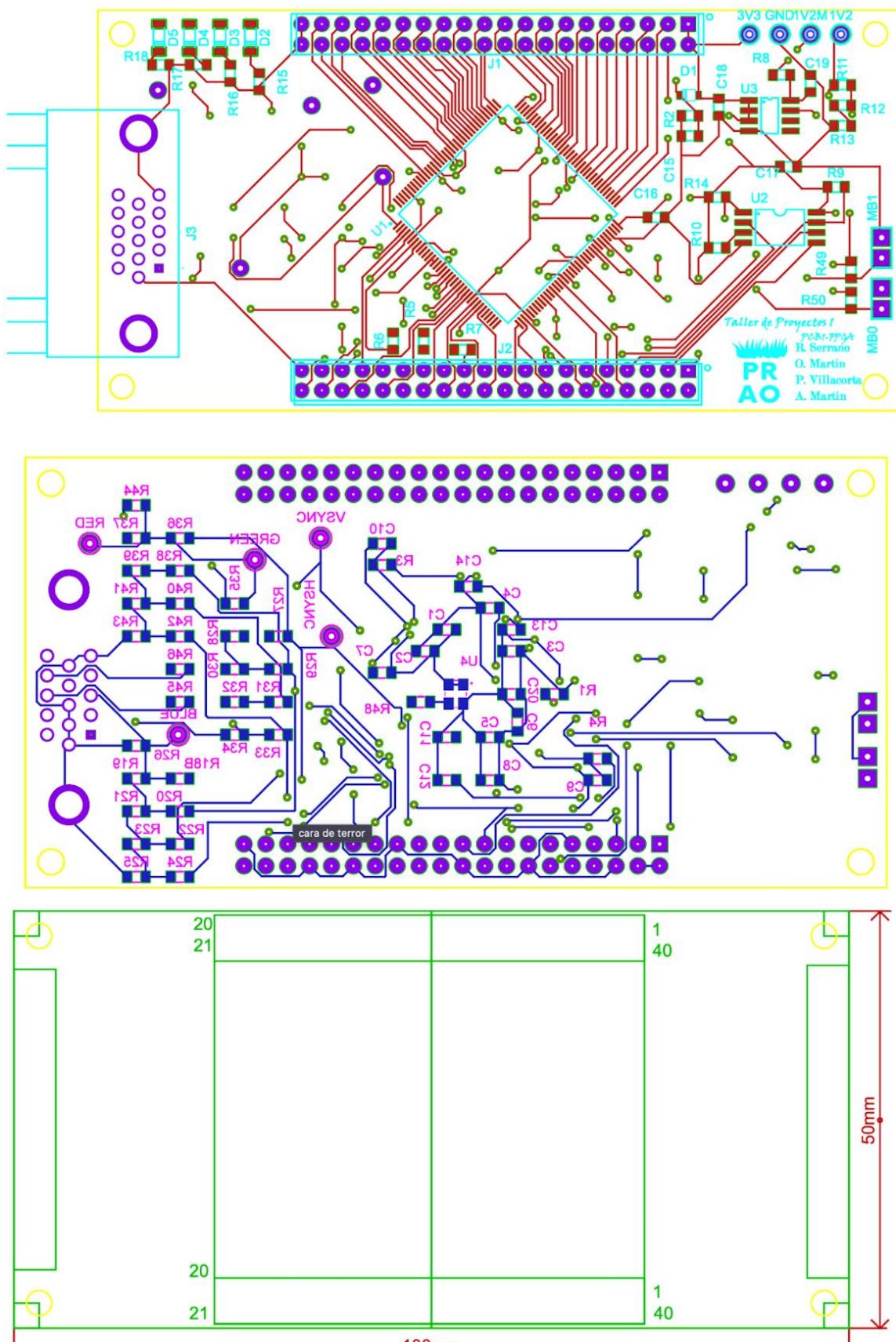


Figura 45. Layouts de la placa PCB1.

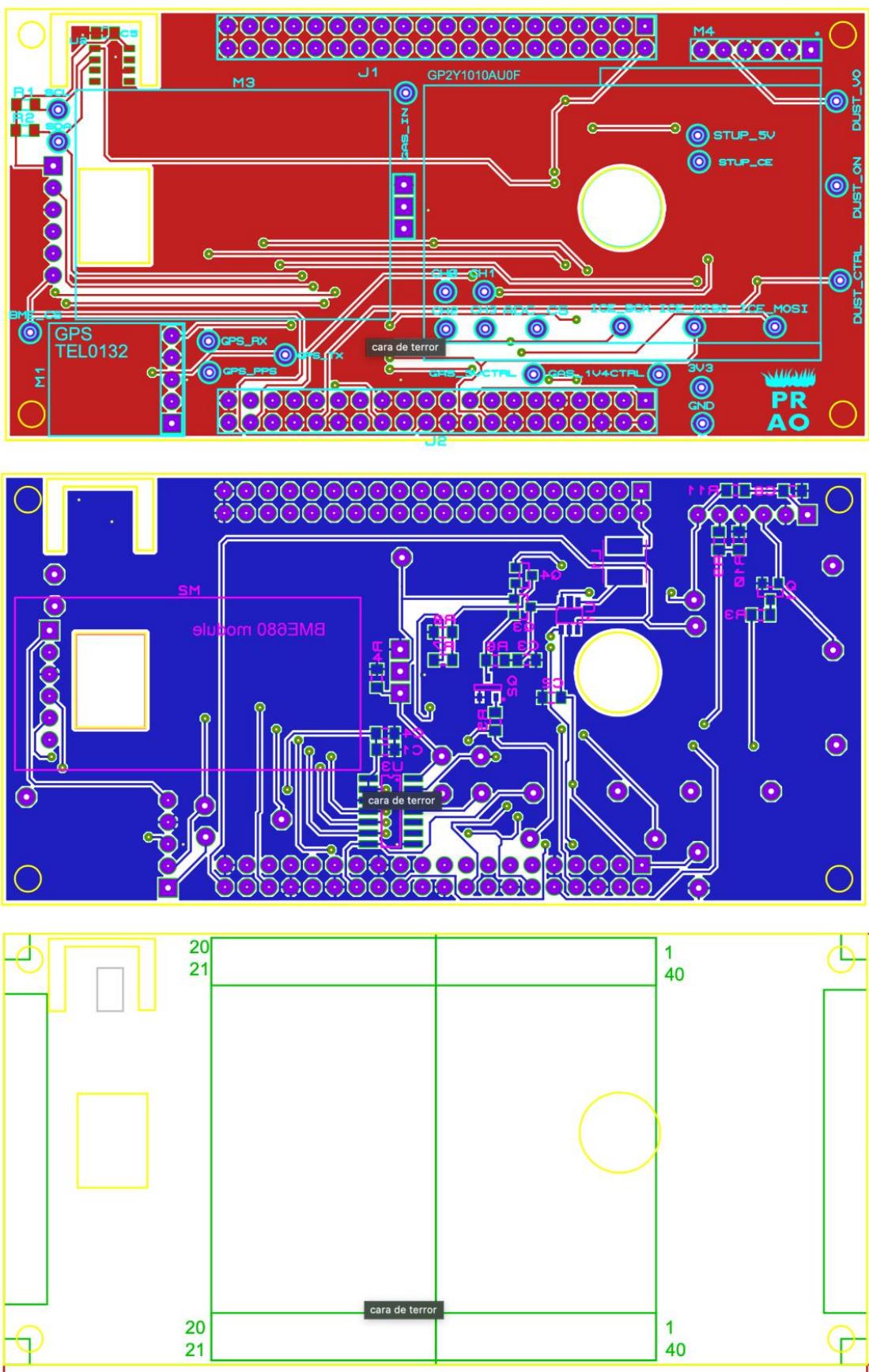
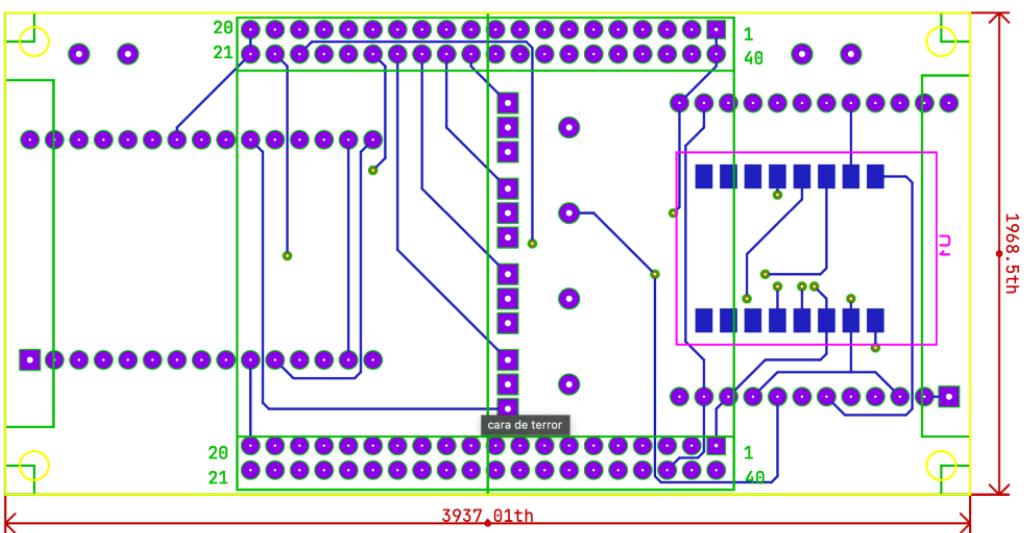
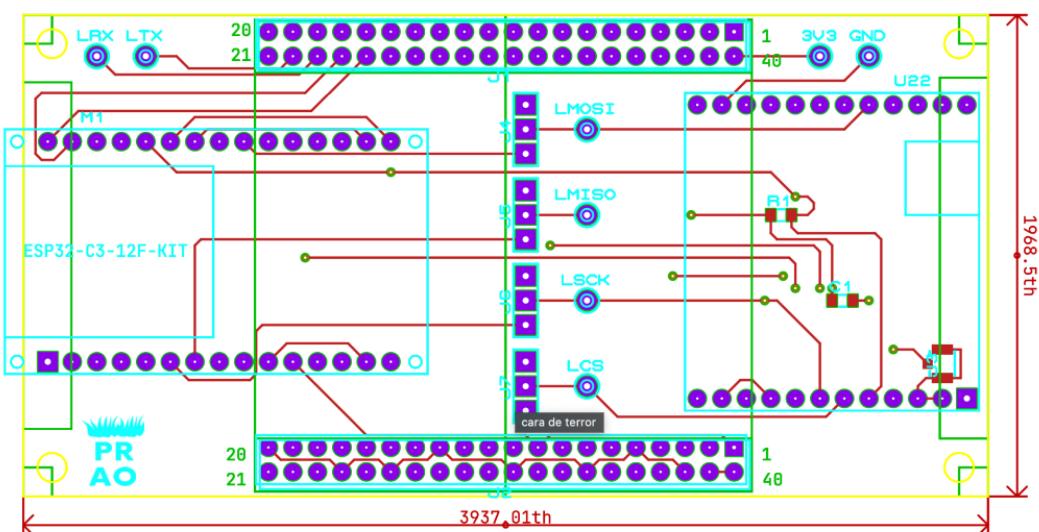


Figura 46. Layouts de la placa PCB2.

PCB3-Wireless



PCB3-Wireless



PCB3-Wireless

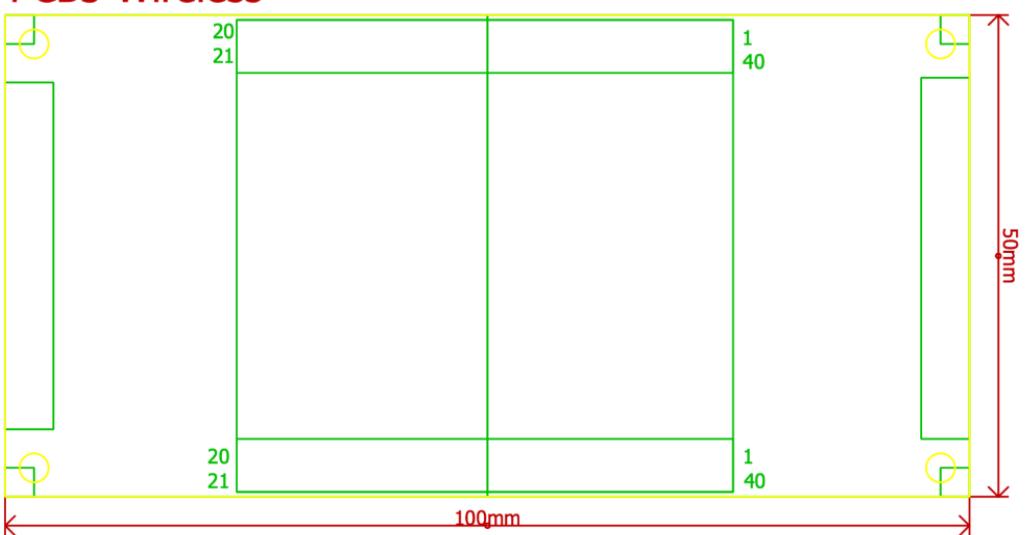


Figura 47. Layouts de la placa PCB3.

2.7.6. Generación de ficheros para el fabricante (Gerber)

En último lugar, tras llevar a cabo todo el diseño del *Datalogger* se han generado los ficheros Gerber, un tipo de archivos utilizados para describir diseños de circuitos impresos. Son utilizados por programas de diseño de circuitos y por fabricantes para describir la geometría de los componentes en un circuito impreso, como las pistas, los componentes o las etiquetas. Además, contienen información sobre la capa, el tamaño y la posición de cada elemento en el circuito, lo que permite que la información sea interpretada y utilizada para fabricar el circuito impreso. Los ficheros Gerber son aceptados por la mayoría de los fabricantes de circuitos impresos.

Previamente a la generación de dichos ficheros nos debemos asegurar de lo siguiente:

1. Todos los componentes han sido posicionados.
2. Se han trazado todas las pistas.
3. Se cumplen todas las reglas de diseño.
4. Se han situado los planos de masa y de disipación térmica.

De hecho, el programa *Proteus* nos ayuda con esta comprobación. Es importante deseleccionar todas las capas mecánicas excepto la *Mech1* que se empleará para hacer huecos en la placa. Cuando se hayan completado dichos pasos se pasan a generar los ficheros en cuestión. Para ello se debe acceder a Menú → Output → Generate Gerber/Excellon Files. Asimismo, se generarán también los ficheros Excellon, los cuales definirán los taladrados que se han de realizar.

Para asegurarnos de que nuestro proveedor sea capaz de fabricar las placas configuraremos el modo RS274X, dado que es el formato más antiguo existente y tendremos la certeza de que es compatible durante su proceso de fabricación.

2.8. Fabricación de la placa de circuito impreso

Parte fundamental del proyecto es mandar a fabricar nuestro diseño, de esa manera comprobaremos si todo lo planteado y diseñado se ha hecho correctamente.

2.8.1. Envío al fabricante

En primer lugar, se debe enviar nuestro diseño al fabricante, en este caso nos hemos decantado por la empresa china PCBWay, disponible en <https://www.pcbway.es/>. Ésta es una compañía de fabricación de circuitos impresos en línea, en la que ofrecen servicios de diseño, fabricación y montaje de PCBs para clientes individuales y empresas.

La empresa ofrece una variedad de opciones para adaptarse a las necesidades de los clientes, incluyendo diferentes tipos de materiales, tamaños de PCB, colores de placa, espesores de laminado, y opciones de acabado. También ofrecen servicios adicionales como ensamblaje de componentes y pruebas de funcionamiento. Además, se caracteriza por ofrecer precios competitivos y servicio al cliente, ofreciendo un tiempo de entrega rápido.

2.8.2. Recepción y revisión

Tras alrededor de dos o tres semanas se recibieron las placas diseñadas tal y como podemos observar de la Figura 47 a la 50. El primer paso fue revisar que las pistas no pasaban por la zona de corte y que no había ningún error en diseño.

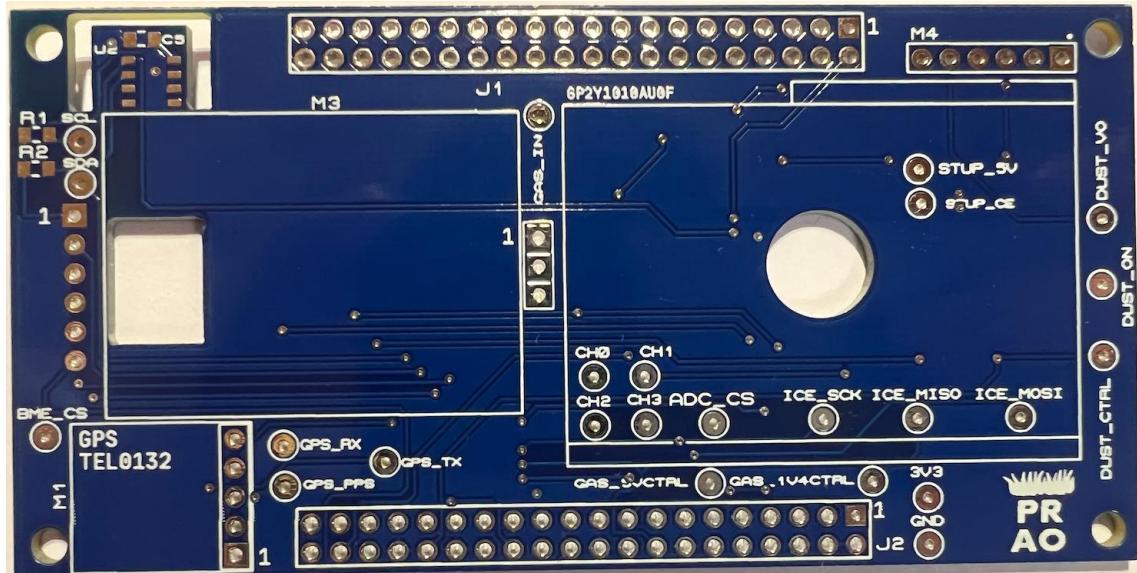


Figura 48. Anverso de la PCB2.

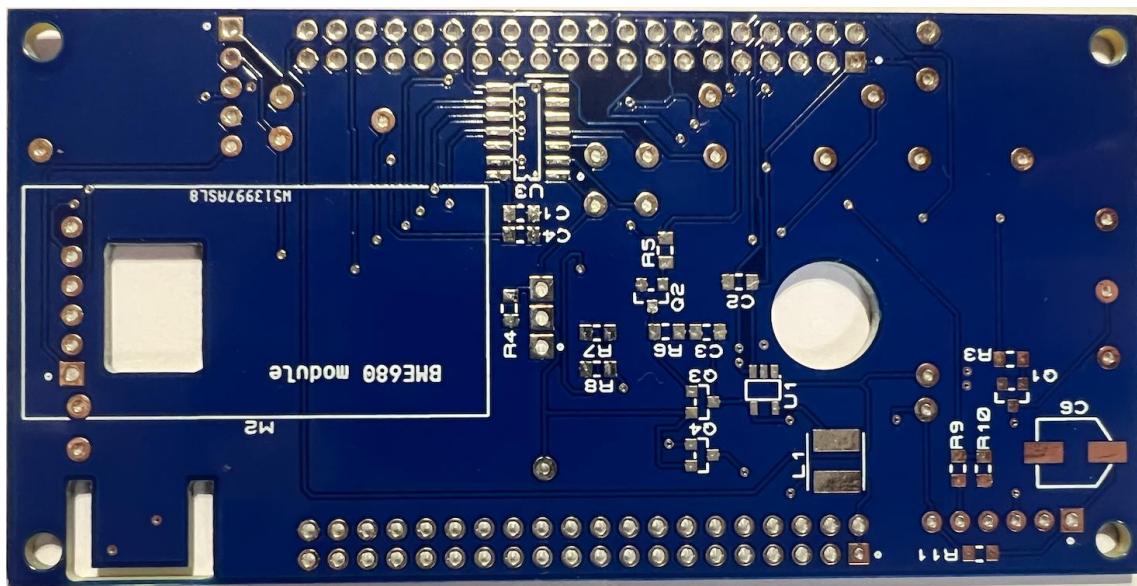


Figura 49. Reverso de la PCB2.

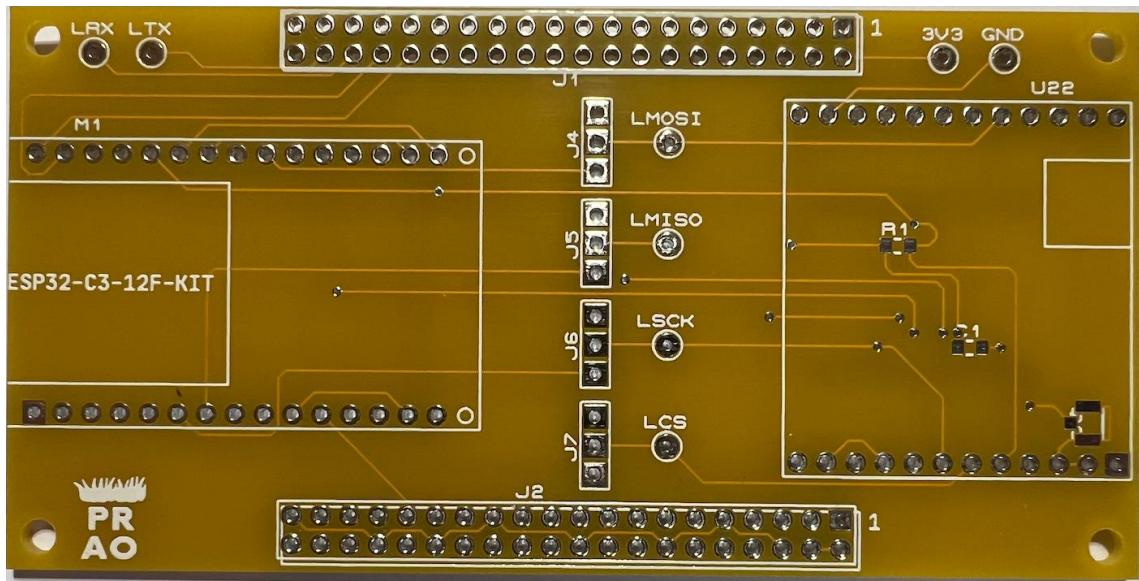


Figura 50. Anverso de la PCB3.

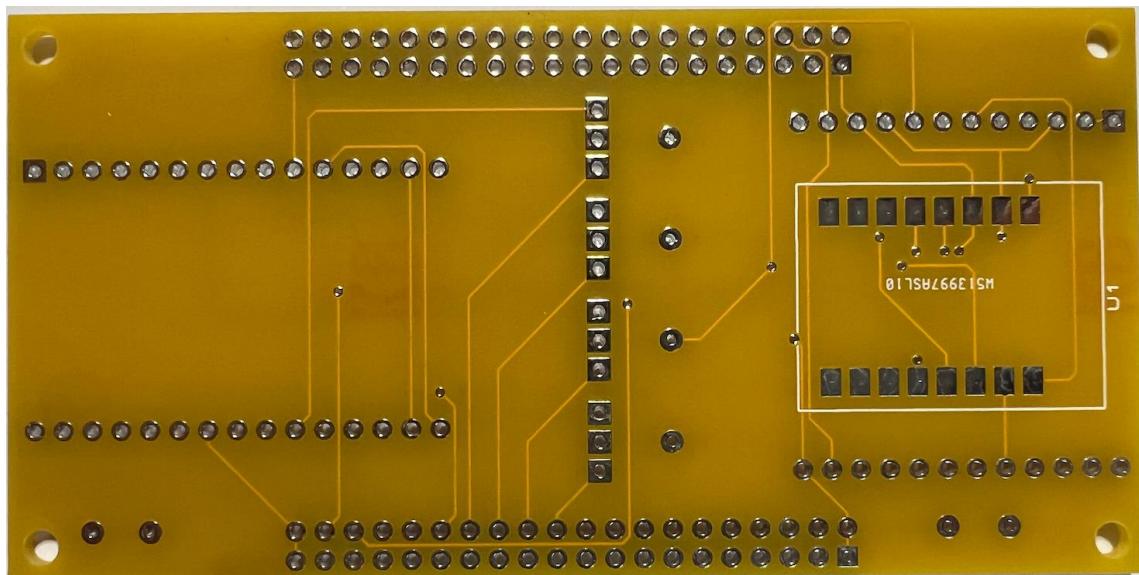


Figura 51. Reverso de la PCB3.

2.8.3. Montaje de componentes

El proceso de colocar los componentes en la placa de circuito impreso ha sido una de las partes más interesantes del proyecto, se ha llevado a cabo íntegramente en el laboratorio de soldadura de la facultad.

Para llevar a cabo el montaje, es necesario verificar que la placa de circuito impreso está limpia. Teniendo claro cuál es el componente que queremos soldar debemos tener claro cómo llevarlo a cabo. Nos encontramos ante tres posibilidades, soldar el componente con Flux, una sustancia proporcionada a modo de rotulador para mejorar la adherencia y soldadura del componente, soldarlo con estaño o hacer ambos procesos juntos.

Los componentes que se soldarán con Flux serán aquellos que poseen las patas muy juntas y que soldar con estaño sería prácticamente imposible debido a sus dimensiones, por ejemplo, un circuito integrado. Entonces, aplicando Flux sobre la superficie, colocando el componente en el lugar y posición correcta, aplicaremos calor en cada una de las patillas para soldarlo.

En caso de que nos encontramos con componentes más grandes como conectores, resistencias o condensadores, podremos soldarlos directamente mediante estaño y si quisiéramos, podríamos ayudarnos a fijarlos mediante Flux. Es de vital importancia tener cuidado en este punto dado que no queremos emplear una cantidad de estaño excesiva.

Dado que queremos que nuestros componentes puedan ser reutilizados sin ningún problema en futuros proyectos de la titulación, algunos de ellos no los soldaremos directamente sobre la placa, sino que soldaremos un conector donde podremos quitar y poner los sensores u otros componentes según nos parezca.

Asimismo, debido a que nuestro proyecto engloba numerosos componentes, hay que ser minucioso a la hora de posicionarlos, pues con todos los sensores montados queremos que no se molesten unos a otros. Teniendo en cuenta la disposición de las placas, superpuestas unas sobre otras, decidimos poner todos los sensores por el mismo lado y tras realizar la soldadura de sus conectores nos dimos cuenta de que habíamos cometido dos errores, tomamos una mala decisión dado que dos sensores no podían posicionarse correctamente por conflicto con otros componentes de la placa.

En la figura 51 a la izquierda de la resistencia R4 nos encontramos el conector del módulo sensor de gases SEN0134, donde lo soldamos justo por el lado contrario al que se encuentra en dicha figura, es decir, tal y como podemos ver en la figura 52 donde se encuentra ya correctamente.



Figura 52. Error cometido durante el montaje de la placa PCB2.



Figura 53. Correcto posicionamiento del conector del sensor de gas.

El siguiente error que cometimos fue soldar por el lado que no debíamos el conector del sensor GPS, dado su posicionamiento sobre la placa era imposible conectarlo porque ocuparía el mismo espacio que ocupa unos de los conectores entre placas. En la figura 53 podemos encontrar dicho sensor ubicado correctamente.



Figura 54. Correcto posicionamiento del sensor GPS sobre la PCB2.

Finalmente, tras posicionar y soldar cada uno de los componentes tenemos el resultado final del montaje. Éste lo podemos observar en las figuras 54 a 57.

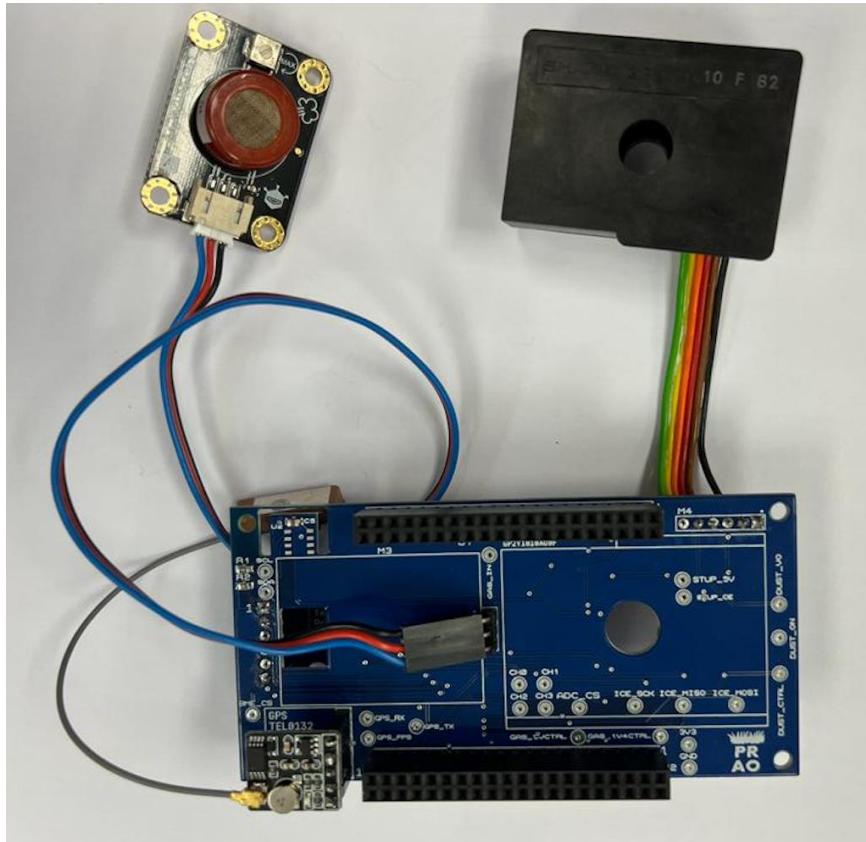


Figura 55. Anverso de la placa PCB2 con todos los componentes.

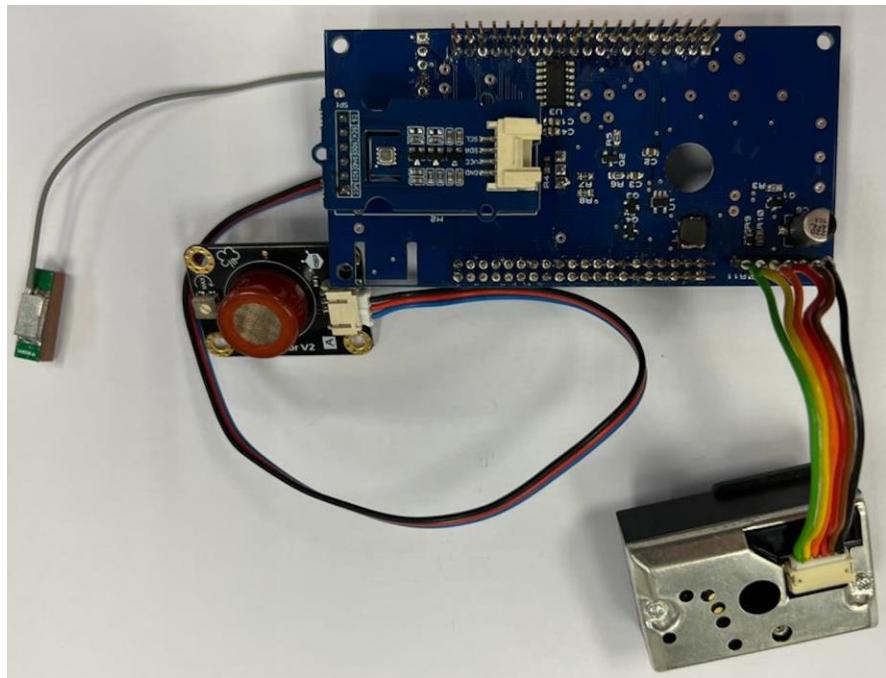


Figura 56. Reverso de la laca PCB2 con todos los componentes.

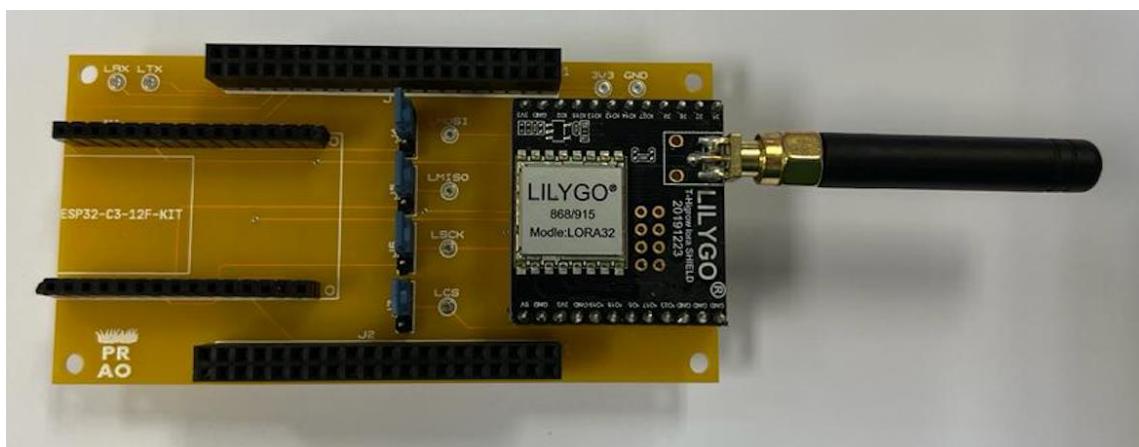


Figura 57. Anverso de la placa PCB3 con todos los componentes.

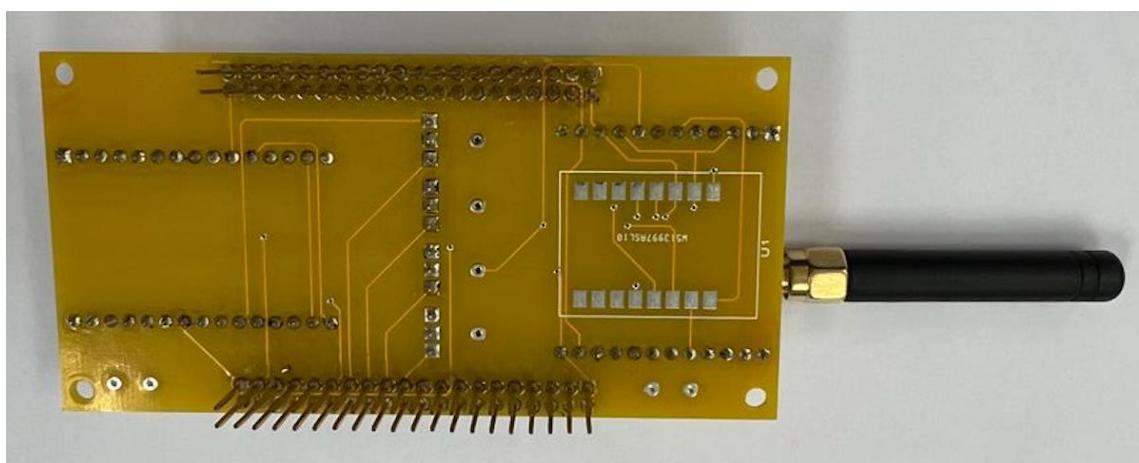


Figura 58. Reverso de la placa PCB3 con todos los componentes.

2.8.4. Depuración Hardware

Tras la realización del montaje de los componentes se ha de realizar una comprobación del correcto funcionamiento del hardware, verificando que no haya ningún error en las conexiones realizadas. Esta verificación y comprobación es básicamente la depuración del hardware, el último paso que se ha de seguir en la fabricación y montaje de las placas diseñadas para el proyecto.

En esta parte se detectó que la lectura de los sensores de polvo y de gas a través del ADC no era correcta. Revisando las soldaduras que habíamos realizado observamos que quizás alguna de ellas no estaba haciendo un buen contacto, por lo que las soldamos de nuevo para asegurarnos de que todo fuera correcto. Sin embargo, de esta manera no logramos solventar el problema, por lo que optamos por revisar punto a punto los voltajes que obteníamos. Dado que por ejemplo requerimos de un pulso que oscile entre 5 y 1,4V para obtener los datos de manera correcta del sensor de gas, con ayuda del multímetro vimos que en el drenador del transistor mosfet Q3, que se puede observar en las figuras 58 y 59, no obteníamos señal.

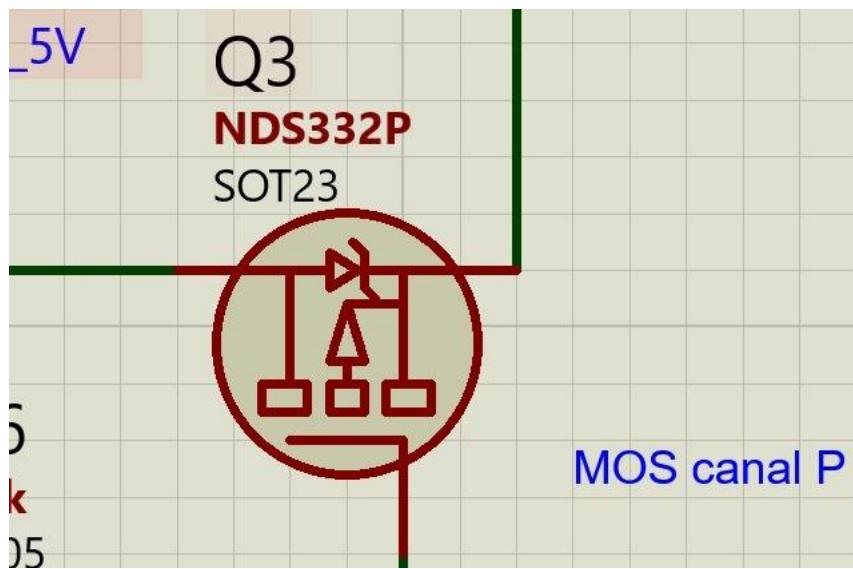


Figura 59. Transistor mosfet Q3 erróneo de la placa de sensores sobre esquemático.

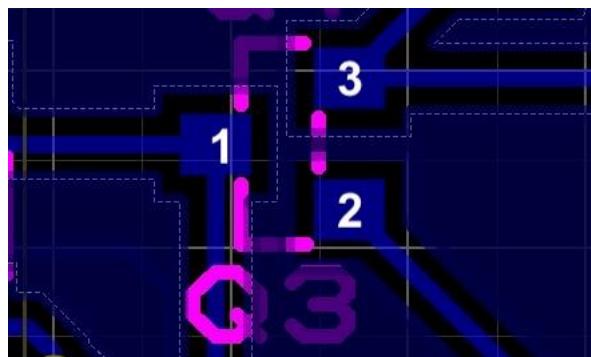


Figura 60. Transistor mosfet Q3 erróneo de la placa de sensores sobre PCB.

Revisando el esquemático nos dimos cuenta de que no conectamos dicho transistor correctamente, en nuestro diseño el pin 1 correspondía con el drenador y el pin 3 con la fuente mientras que debería ser al revés. Por ello decidimos eliminar las soldaduras y

posicionarlo como debería ser, es decir, rotarlo de tal manera que el pin 2, que corresponde a la puerta, conserve su conexión, y que en el pin 1 se encuentre conectado a la fuente para así conectar el drenador al pin 3 mediante un cable. Podemos visualizar los cambios realizados en la figura 60.

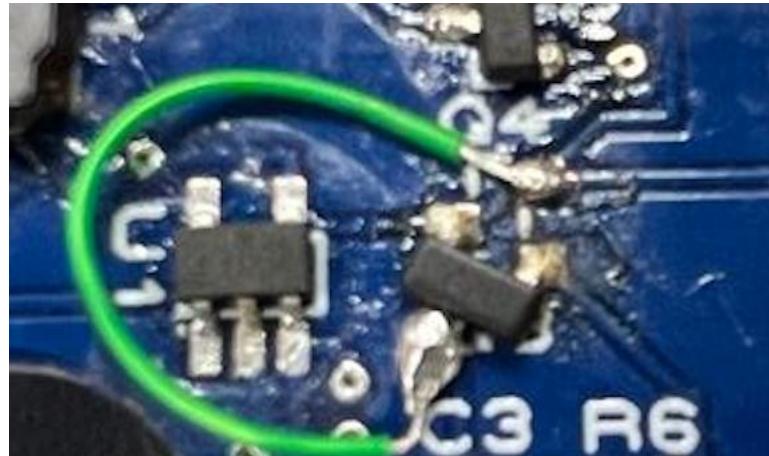


Figura 61. Error solventado en transistor mosfet Q3.

Finalmente realizando de nuevo las pruebas con el multímetro obtuvimos los valores que esperábamos, por lo que decidimos lanzar el programa para leer de nuevo a través del ADC los valores de los sensores y el resultado fue correcto, habíamos solventado el error.

2.9. Descripción del hardware de la FPGA (Verilog)

Un componente fundamental dentro del sistema desarrollado es la FPGA (*Field-Programmable Gate Array*). Éste es un dispositivo programable que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada en el momento, mediante un lenguaje de descripción especializado. Esta característica dota a estos dispositivos de una increíble versatilidad, pudiendo reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional, hasta complejos sistemas como el sistema microcontrolador larva y los periféricos implementados.

Tal y como se acaba de indicar, las interconexiones y funcionalidades de la FPGA se definen mediante lenguajes de descripción de hardware (HDLs), entre los que destaca Verilog, el lenguaje utilizado en este proyecto. Es importante remarcar que con Verilog no estamos programando el software, sino definiendo directamente la distribución física del circuito, como si se tratase del cableado manual de puertas lógicas y circuitos integrados.

El compilador yosys permite ver en qué se ha sintetizado el diseño hardware final en la FPGA a partir del fichero sint.txt. Estos han sido los resultados:

Number of wires:	3180
Number of wire bits:	6512
Number of public wires:	3180
Number of public wire bits:	6512
Number of memories:	0
Number of memory bits:	0
Number of processes:	0
Number of cells:	4821
SB_CARRY	467
SB_DFF	167
SB_DFFE	896
SB_DFFER	38
SB_DFFESR	133
SB_DFFNE	16
SB_DFFR	33
SB_DFFSR	68
SB_LUT4	2970
SB_PLL40_CORE	1
SB_RAM40_4KNRNW	32

Además, observando el fichero pnr.txt se puede ver la frecuencia máxima de funcionamiento del circuito, en nuestro caso 19.50 MHz:

“Info: Max frequency for clock 'clk_\$glb_clk': 19.50 MHz (PASS at 12.00 MHz)”

Por otra parte, para poder definir el hardware es importante saber qué componentes es necesario integrar dentro de la FPGA, y sus pertinentes conexiones. La **¡Error! No se encuentra el origen de la referencia**. Figura 61 muestra un primer esquema de los módulos que se planteó integrar en un inicio.

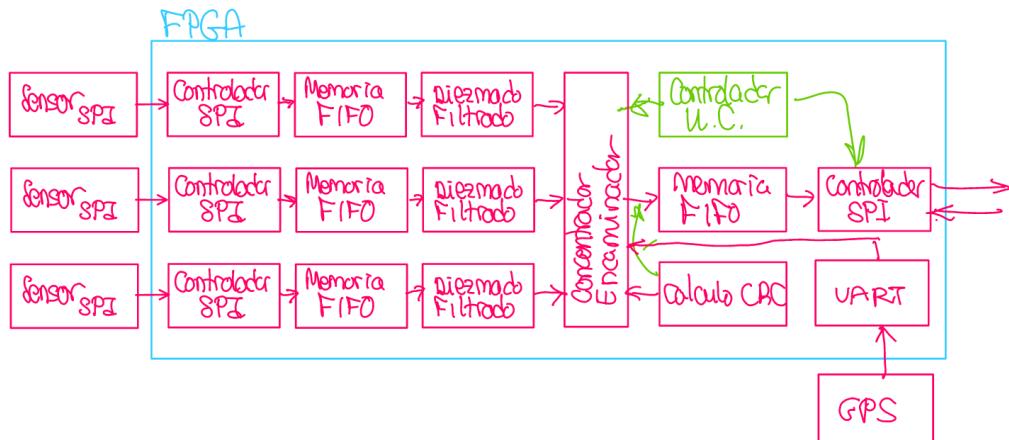


Figura 62. Diagrama de bloques con los componentes del sistema que es necesario definir dentro de la FPGA.

Finalmente, el hardware implementado cuenta con:

- **CPU (Central Processing Unit):** es el elemento más importante. Ejecuta instrucciones basadas en la realización de operaciones lógico-aritméticas con el contenido de registros. La CPU es la encargada, por lo tanto, de ejecutar el código en C (el *software* propiamente dicho) que escribiremos para definir el comportamiento del programa. Concretamente, la CPU que se integrará será LaRVa, un core basado en la arquitectura RISC V.
- **Periféricos:** son módulos hardware encargados de conectar el microcontrolador LaRVa RISC V con los sensores y transceptores del sistema. Concretamente, hemos implementado:
 - o 3 módulos UART:
 - UART0: encargada de comunicar con el terminal de texto del ordenador.
 - UART1: encargada de la comunicación con el módulo GPS.
 - UART2: encargada de la comunicación con el módulo WiFi.
 - o 2 módulos SPI: se utilizarán para la comunicación con todos los módulos o circuitos que utilicen este protocolo. Concretamente:
 - SPI0: comunicación con el ADC (sensores de gas y de partículas) y con el sensor de humedad, presión y temperatura BME680.
 - SPI1: comunicación con el transceptor LoRa.
 - o Periférico para el acceso a pines de entrada (GPIN) y salida (GPOUT). Servirán tanto para introducir señales lógicas en el dispositivo como para extraerlas.
 - o Periférico temporizador: necesario para la implementación de retardos en las rutinas, así como para definir correctamente las señales de control de algunos sensores -concretamente, el de gas y el de polvo-. Deberá generar interrupciones.
 - o Módulo de control de interrupciones (VIC): permitirá gestionar las interrupciones provenientes de los periféricos. Estará formado por un registro de habilitación de interrupciones, los vectores de interrupción y un codificador de prioridad.

- **Memoria RAM:** en ella se introducirá el código del programa a ejecutar. Se sitúa en las direcciones más bajas del mapa de memoria de la FPGA.

A nivel interno, el esquema de interconexiones a alto nivel se puede ver en el siguiente ejemplo:

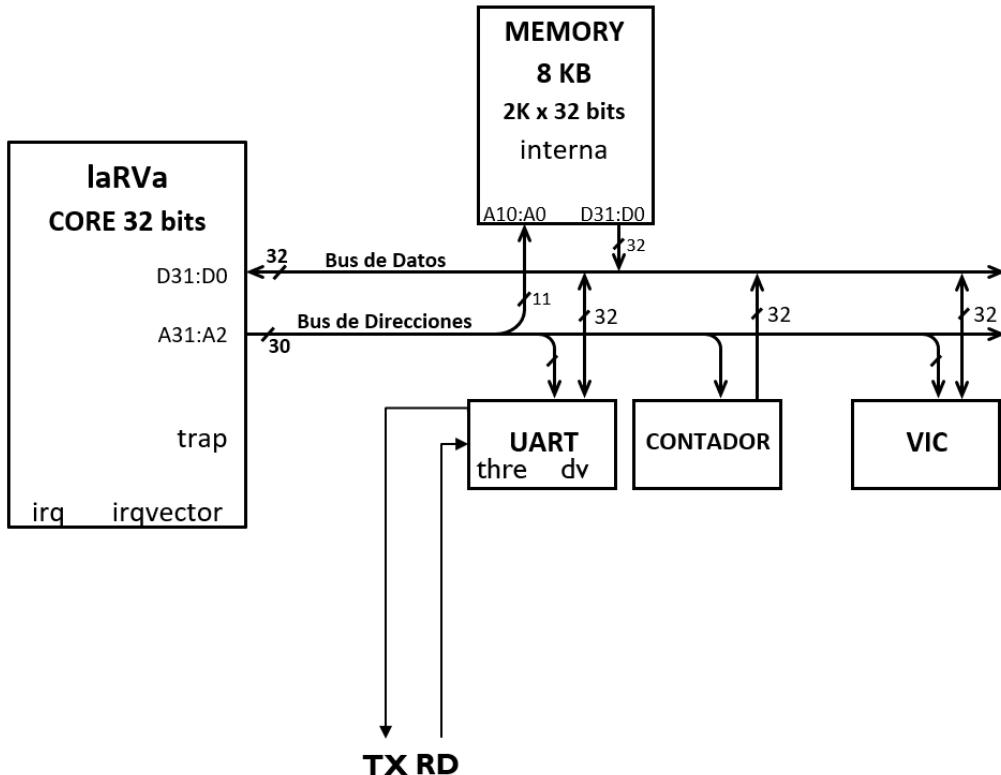


Figura 63. Esquema de interconexiones sobre el microcontrolador implementado.

En la figura se muestran únicamente las conexiones para un módulo contador, UART y VIC, pero el razonamiento se puede extender al resto de módulos implementados. La idea es que el registro sobre el que actúen las operaciones de lectura/escritura sea seleccionado desde LaRVA a través del bus de direcciones. Para ello, obviamos los últimos 2 bits de direccionamiento (puesto que las palabras están alineadas en memoria x4).

Con el fin de seguir un criterio razonable, los tres primeros bits del bus de direcciones (A [31:29]) indican el acceso o bien a la memoria o bien a los dispositivos de entrada / salida. Por otra parte, los bits (A[7:5]), denominados como *chip select* son encargados de seleccionar el dispositivo correspondiente, aunque finalmente debido a la cantidad de módulos implementados, las 3 UART o los 2 SPI comparten el mismo código de *chip select*. Finalmente los bits (A[4:2]) permiten seleccionar de cada uno de los dispositivos, el registro sobre el cual se quiere realizar la operación.

Tanto en la etapa de diseño hardware como en la etapa de diseño software, ha sido imprescindible contar con una tabla referencia en la que hemos recogido todo el conjunto de direcciones necesarias para los dispositivos implementados (2).

IO/Mem			Chip Select			Registro						
	31	30	29	(...)	7	6	5	4	3	2	1	0
UART0	TX/RX	0XE0000000	1	1	1	0	0	0	0	0	0	0
	DIVIDER/FLAGS	0xE0000004	1	1	1				0	0	1	0
UART1	UART1 TX/RX	0XE0000008	1	1	1	0	0	0	0	1	0	0
	DIVIDER/FLAGS	0xE000000C	1	1	1				0	1	1	0
UART2	TX/RX	0XE0000010	1	1	1	0	0	0	1	0	0	0
	DIVIDER/FLAGS	0xE0000014	1	1	1				1	0	1	0
SPI0	TXDATA/RXDATA	0xE0000020	1	1	1	0	0	1	0	0	0	0
	CONTROL/FLAGS	0xE0000024	1	1	1				0	0	1	0
	SS (SLAVE SELECT)	0xE0000028	1	1	1				0	1	0	0
SPI1	TXDATA/RXDATA	0xE0000030	1	1	1	0	0	0	1	0	0	0
	CONTROL/FLAGS	0xE0000034	1	1	1				1	0	1	0
	SS (SLAVE SELECT)	0xE0000038	1	1	1				1	1	0	0
I2C	DATA/CONTROL DATA/STATUS	0xE0000040	1	1	1	0	1	0	0	0	0	0
	DIVIDER	0xE0000044	1	1	1				0	0	1	0
TEMP	MAXCOUNT/TIMER	0XE0000060	1	1	1	0	1	1	0	0	0	0
GP	GPOUT/GPOUT	0xE0000080	1	1	1	1	0	0	0	0	0	0
	GPOUT/GPIN	0xE0000084	1	1	1				0	0	1	0
IRQ	INTERRUPT ENABLE	0xE00000C0	1	1	1	1	1	1	0	0	0	0
	VECTOR 0: Trap	0xE00000E0	1	1	1				0	0	0	0
	VECTOR 1: RX	0xE00000E4	1	1	1				0	0	1	0
	VECTOR 2: TX	0xE00000E8	1	1	1				0	1	0	0
	VECTOR 3: TIMER	0xE00000EC	1	1	1				0	1	1	0
	VECTOR 4: RX1	0xE00000F0	1	1	1				1	0	0	0
	VECTOR 5: TX1	0xE00000F4	1	1	1				1	0	1	0
	VECTOR 6: RX2	0xE00000F8	1	1	1				1	1	0	0
	VECTOR 7: TX2	0xE00000FC	1	1	1				1	1	1	0

Tabla 1. Direcciones de memoria utilizadas por los periféricos.

Además, el código correspondiente a cada dirección mapea un registro u otro dependiendo si la operación que se realiza es de lectura o de escritura.

A nivel de programación, para describir el hardware se han utilizado los ficheros `main.v`, `system.v`, `laRVA.v`, `uart.v` y `SPI.v` aunque hay otros módulos que se definen dentro del fichero `system.v`. A continuación, se muestra un esquema que representa la jerarquía de ficheros y módulos principales que se tratarán en detalle.

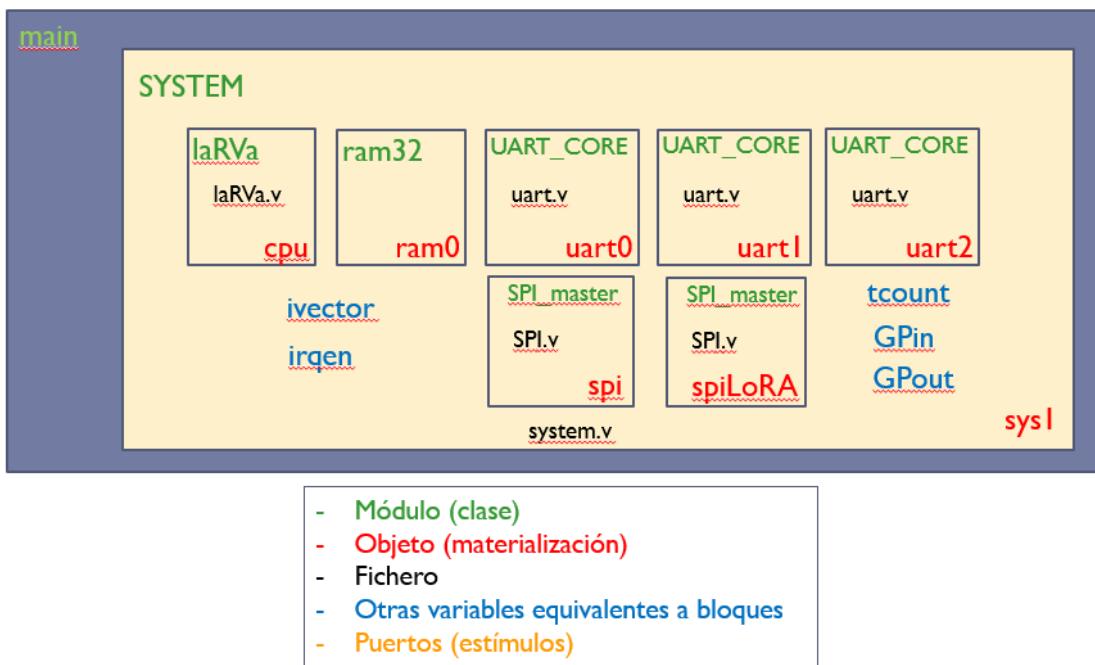


Figura 64. Jerarquía de ficheros y módulos principales.

2.9.1. Generalidades system.v

La descripción de los periféricos implementados en la FPGA se ha realizado en el fichero `system.v`, donde se instancian los distintos módulos tras incluirlos en el fichero (por ejemplo antes de instanciar el módulo SPI “`include "SPI.v"`”) por lo que éste es el más relevante para el trabajo en cuestión. Además al incluir un nuevo módulo, también se debe editar el makefile correspondiente (“`DEPSIM = system.v laRVa.v uart.v SPI.v DEPSINT= pll.v system.v laRVa.v uart.v SPI.v`”)

Las operaciones de lectura y escritura de los periféricos se realizan constantemente para poder gestionar la información exterior desde el microcontrolador.

Los valores leídos de los registros de los periféricos llegan a laRVA desde el bus iodo y se seleccionan a través de un multiplexor que utiliza para ello los bits del bus de direcciones `ca[7:2]` (*chip select* y registro).

Por otra parte, tanto para realizar una lectura como una escritura en cualquiera de los registros de los periféricos es imprescindible seleccionar dicho periférico (puesto que comparten los buses de datos). Para ello se han definido distintas salidas de habilitación (`spics`, `uartcs`, `tempcs`, `gpcs` ...) que codifican los códigos de *chip select*.

A modo de ejemplo se puede observar a continuación un esquema de la filosofía de interconexiones que sigue el sistema para comunicar los periféricos y la memoria RAM con laRVA mostrando algunas de las entradas al multiplexor del bus iodo y la obtención de las señales de control `spi_wr` y `spi_wr_ctrl` de SPI0.

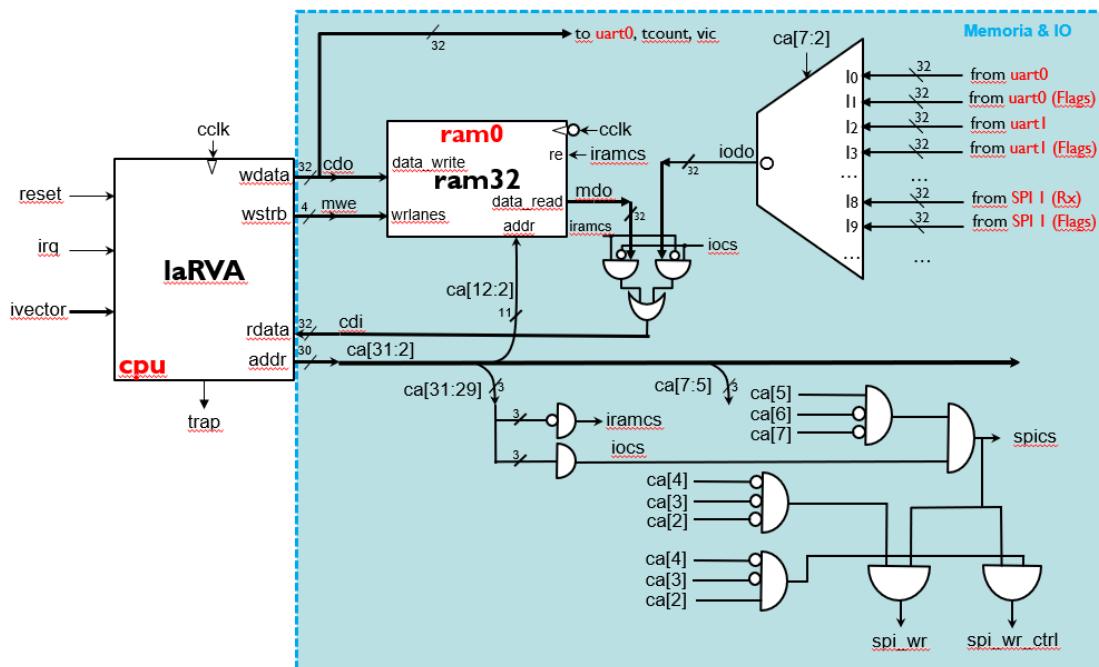


Figura 65. Interconexiones laRVA – Memoria RAM & IO.

La descripción de dicho hardware se puede ver en el código del fichero `system.v`, tanto para la obtención de la señal `iodo` a partir de las salidas de los periféricos hacia LaRVA:

```

// Peripheral output bus mux
reg [31:0]iodo; // Not a register
always@*
casex (ca[7:2])
  6'b000000: iodo<={24'h0,uart0_do};
  6'b000001: iodo<={27'h0,ove0,fe0,tend0,thre0,dv0};
  6'b000010: iodo<={24'h0,uart1_do};
  6'b000011: iodo<={27'h0,ove1,fe1,tend1,thre1,dv1};
  6'b000100: iodo<={24'h0,uart2_do};
  6'b000101: iodo<={27'h0,ove2,fe2,tend2,thre2,dv2};

  6'b001000: iodo<=rx_spi;           // SPI_RX
  6'b001001: iodo<={31'h0,busy};   // SPI FLAG (busy)
  6'b001100: iodo<=rx_spiLoRA;     // SPI_LoRA_RX
  6'b001101: iodo<={31'h0,busyLoRA}; // SPI_LoRA FLAG (busy)

  6'b100000: iodo<={24'h0,GPOUT}; // GPOUT
  6'b100001: iodo<={24'h0,gpin7,gpin6,gpin5,gpin4,gpin3,gpin2,gpin1,gpin0};

//6'b010000: iodo<={...}; // I2C...
//6'b010001: iodo<={...}; // I2C...

  6'b011xxx: iodo<=tcount; // TIMER
  6'b110xxx: iodo<={24'h0,irqen};
  default: iodo<=32'hxxxxxxxx;
endcase

```

Figura 66. Descripción hardware del multiplexor que permite obtener la señal iodo.

Como la descripción de las señales de habilitación *chip select*:

```

// ---> CONFIGURACIÓN DE LOS CHIP SELECT [CS]
wire uartcs;    // UART                      at offset 0x00
wire spics;     // SPI                       at offset 0x20
wire i2ccs;      // I2C                      at offset 0x40
wire tempcs;    // Timer                     at offset 0X60
wire gpcs;       // GENERAL PURPOSE      at offset 0x80
wire iencs;      // INTERRUPT ENABLE      at offset 0xC0
wire irqcs;     // IRQEN                     at offset 0xE0
// ...
// other at offset 0xE0

assign uartcs = iocs&(ca[7:5]==3'b000);
assign spics = iocs&(ca[7:5]==3'b001);
assign i2ccs = iocs&(ca[7:5]==3'b010);
assign tempcs = iocs&(ca[7:5]==3'b011);
assign gpcs = iocs&(ca[7:5]==3'b100);

assign iencs = iocs&(ca[7:5]==3'b110); //Interrupt Enable Chip Select
assign irqcs = iocs&(ca[7:5]==3'b111);

```

Figura 67. Descripción hardware de las señales de habilitación chip select.

2.9.2. UART0

La UART0 es muy necesaria para el funcionamiento del sistema puesto que permite la comunicación con un PC para cargar los programas software que se quieran ejecutar en el sistema diseñado.

Para la UART0 se han utilizado los siguientes registros mapeados en las direcciones **0xE0000000** (para la transmisión/recepción de datos) y **0xE0000004** para escritura en el divisor de baudios/lectura de *flags*.

Escritura

0XE0000000	31 undefined	8 7	TX DATA	0
0XE0000004	31 BAUDBITS undefined	BAUDBITS-I	DIVIDER	0

Lectura

0XE0000000	31 undefined	8 7	RD DATA	0
0XE0000004	31 undefined	5 4 3 2 1 0	OVE FE TEND THRE DV	

Figura 68. Registros utilizados para la UART0.

El campo DIVIDER permite configurar la velocidad de funcionamiento de la UART tal que:

$$\text{velocidad (Baudios)} = \frac{Fcclk}{DIVIDER + 1}$$

2.9.3. UART1

Como el módulo GPS es un componente que ha de estar enviando información de seguido se hace necesario la implementación de esta segunda UART, para controlar el flujo de datos que provengan del GPS de manera separada al flujo de datos proveniente de los periféricos de la FPGA.

El funcionamiento es análogo a la UART0 sólo que las direcciones utilizadas son **0xE0000008** (para la transmisión/recepción de datos) y **0xE000000C** (para indicar el divisor de baudios/lectura de flags).

Escritura

0XE0000008	31 undefined	8 7	TX DATA	0
0XE000000C	31 BAUDBITS undefined	BAUDBITS-I	DIVIDER	0

Lectura

0XE0000008	31 undefined	8 7	RD DATA	0
0XE000000C	31 undefined	5 4 3 2 1 0	OVE FE TEND THRE DV	

Figura 69. Registros utilizados para la UART1.

2.9.4. UART2

Esta tercera UART ha sido implementada para la comunicación con el módulo WiFi. De nuevo, el funcionamiento es análogo a las UART explicadas previamente. En este caso las direcciones empleadas son **0xE0000008** (para la transmisión/recepción de datos) y

0xE000000C (para indicar el divisor de baudios/lectura de flags), como bien podemos apreciar en la *tabla 1*.

Escritura

0XE0000010	31	undefined	8 7	TX DATA	0
------------	----	-----------	-----	---------	---

0XE0000014	31	BAUDBITS	BAUDBITS-I	0
------------	----	----------	------------	---

Lectura

0XE0000010	31	undefined	8 7	RD DATA	0
------------	----	-----------	-----	---------	---

0XE0000014	31	undefined	5 4 3 2 1 0	OVE FE TEND THRE DV	
------------	----	-----------	-------------	---------------------	--

Figura 70. Registros utilizados para la UART2.

2.9.5. SPI0 (ADC & BME)

El módulo SPI0 sirve para comunicar el sensor de temperatura BME680 y el ADC con el microcontrolador. Para ello, se utilizan dos bits de selección (*slave select*). SS0 selecciona el esclavo SPI 0 (BME680_CS) cuando se activa en baja. SS1 selecciona el esclavo SPI 1 (ADC_CS) cuando se activa en baja. Los registros mapeados en la memoria se pueden ver a continuación:

Escritura

0XE0000020	31	undefined	8 7	TX DATA	0
------------	----	-----------	-----	---------	---

0XE0000024	31	14 13	8 7	DLEN	0
------------	----	-------	-----	------	---

0XE0000028	31	undefined	2 1 0	SSI	SS0
------------	----	-----------	-------	-----	-----

Lectura

0XE0000020	31	undefined	8 7	RX DATA	0
------------	----	-----------	-----	---------	---

0XE0000024	31	undefined	I 0	BUSY
------------	----	-----------	-----	------

Figura 71. Registros utilizados para el SPI0.

2.9.6. SPI1 (LORA)

Se ha instanciado un segundo módulo **SPI** con el fin de que la FPGA pueda comunicarse con el módulo LoRa. Ha sido necesario contar con un periférico SPI aparte para el LoRa puesto que las señales SPI siguen rutas diferentes en la PCB. Los registros mapeados en la memoria se pueden ver a continuación:

Escritura

0XE0000030	31	undefined	8 7	TX DATA	0
0XE0000034	31	undefined	14 13 8 7	DLEN DIVIDER	0
0XE0000038	31	undefined	I 0	SS0	

Lectura

0XE0000030	31	undefined	8 7	RX DATA	0
0XE0000034	31	undefined	I 0	BUSY	

Figura 72. Registros utilizados para el SPII.

2.9.7. I2C

Aunque se planificó inicialmente, finalmente no se ha implementado el periférico I2C puesto que no se ha montado el sensor que hace uso de dicho protocolo.

2.9.8. GPIN

El periférico **GPIN** que se ha diseñado sirve para medir y mostrar los niveles lógicos de los pines de entrada descritos en el sistema, es decir, para comprobar el correcto funcionamiento de los pines. Cabe destacar que los pines utilizados por la FPGA para la comunicación entre componentes y placas no pueden ser utilizados por este periférico, pues ya tienen un funcionamiento intrínseco en el sistema. Al mismo modo que los anteriores periféricos, a este se le han asignado la dirección de memoria **0xE0000084**.

Escritura

0XE0000084	31	undefined	8 7 6 5 4 3 2 I 0	GP[7] GP[6] GP[5] GP[4] GP[3] GP[2] GP[1] GP[7]	
------------	----	-----------	-------------------	---	--

Lectura

0XE0000084	31	undefined	8 7 6 5 4 3 2 I 0	GP[7] GP[6] GP[5] GP[4] GP[3] GP[2] GP[1] GP[7]	
------------	----	-----------	-------------------	---	--

Figura 73. Registros utilizados para el GPIN.

2.9.9. GPOUT

El periférico GPOUT se utiliza para habilitar o deshabilitar señales lógicas de salida desde la FPGA. Se han utilizado principalmente con el fin de activar señales de control en la placa o LEDs. Al igual que en otros periféricos, a este se le han asignado una serie de direcciones de memoria **0xE0000080**.

Escritura

	31	8	7	6	5	4	3	2	1	0
0XE0000080		undefined	GP[7]	GP[6]	GP[5]	GP[4]	GP[3]	GP[2]	GP[1]	GP[7]

Lectura

	31	8	7	6	5	4	3	2	1	0
0XE0000080		undefined	GP[7]	GP[6]	GP[5]	GP[4]	GP[3]	GP[2]	GP[1]	GP[7]

Figura 74. Registros utilizados para el GPOUT.

2.9.9. Temporizador

Uno de los periféricos más importantes es el **TIMER**. El temporizador permite generar las señales de control de los sensores analógicos o activar los LEDs mediante interrupciones en lugar de generar un programa bloqueante. Su funcionamiento se basa en un contador desde 0 hasta la cuenta máxima que es configurable y se puede determinar escribiendo en el registro MAX_COUNT. El registro timer (tcount) almacena el valor de lectura

Escritura

	31	0
0XE0000060		MAX_COUNT

Lectura

	31	0
0XE0000060		TIMER

Figura 75. Registros utilizados para el timer.

Una vez que llega al final de cuenta, el temporizador se reinicia y genera una señal que hace saltar una interrupción.

Para mantener la señal de interrupción al menos dos ciclos de reloj (necesario para el correcto funcionamiento de las interrupciones), se han utilizado tres registros (*Time match flag* 1, 2 y 3) tal que la salida de habilitación de interrupciones sea igual a ((~TMF1) & TMF), como se ejemplifica a continuación:

```
// Necesitamos 3 registros ya que tenemos que mantener la señal
// de interrupcion durante al menos 2 ciclos de reloj!!!
// La señal de interrupcion posedge_TMF salta cuando TMF & ~TMF1

// Ciclo:          TMF           TMFO          TMF1
//                0             0             0      (No Interr)
//    tcount==TMR   1             0             0      (Interr)
//                1             1             0      (Interr)
//                1             1             1      (No Interr)
```

Figura 76. Funcionamiento de los flags TMF que determinan la señal de interrupción.

2.9.10. Interrupciones

Por último, ha sido realmente importante en el diseño software poder contar con interrupciones para la programación del dispositivo. Para ello, se ha utilizado un registro de 8 bits (IRQ enable) que permite habilitar o deshabilitar las distintas fuentes de interrupciones. Dicho registro se encuentra en la dirección **0xE00000C0**.

Por otra parte, el hardware ha sido diseñado para albergar un total de 8 interrupciones por lo que se requiere de un total de 8 registros más que almacenen el vector de interrupciones que determine que función se ejecutará dependiendo de cada fuente.

Escritura

0XE00000C0	31	9	8	7	6	5	4	3	2	1	0
	undefined	Irq[7]	Irq[6]	Irq[5]	Irq[4]	Irq[3]	Irq[2]	Irq[1]	Irq[0]		

Lectura

0XE00000C0	31	9	8	7	6	5	4	3	2	1	0	
	undefined	Irq[7]	Irq[6]	Irq[5]	Irq[4]	Irq[3]	Irq[2]	Irq[1]	Irq[0]			
0xE00000E0	31	IRQVEC[0] (TRAP)										
0xE00000E4	31	IRQVEC[1] (U0RX)										
0xE00000E8	31	IRQVEC[2] (U0TX)										
0xE00000EC	31	IRQVEC[3] (TIMER)										
0xE00000F0	31	IRQVEC[4] (UIRX)										
0xE00000F4	31	IRQVEC[5] (UITX)										
0xE00000F8	31	IRQVEC[6] (U2RX)										
0xE00000FC	31	IRQVEC[7] (U2TX)										

Figura 77. Registros utilizados para el control de interrupciones.

El hardware implementado se puede ver en el siguiente esquema:

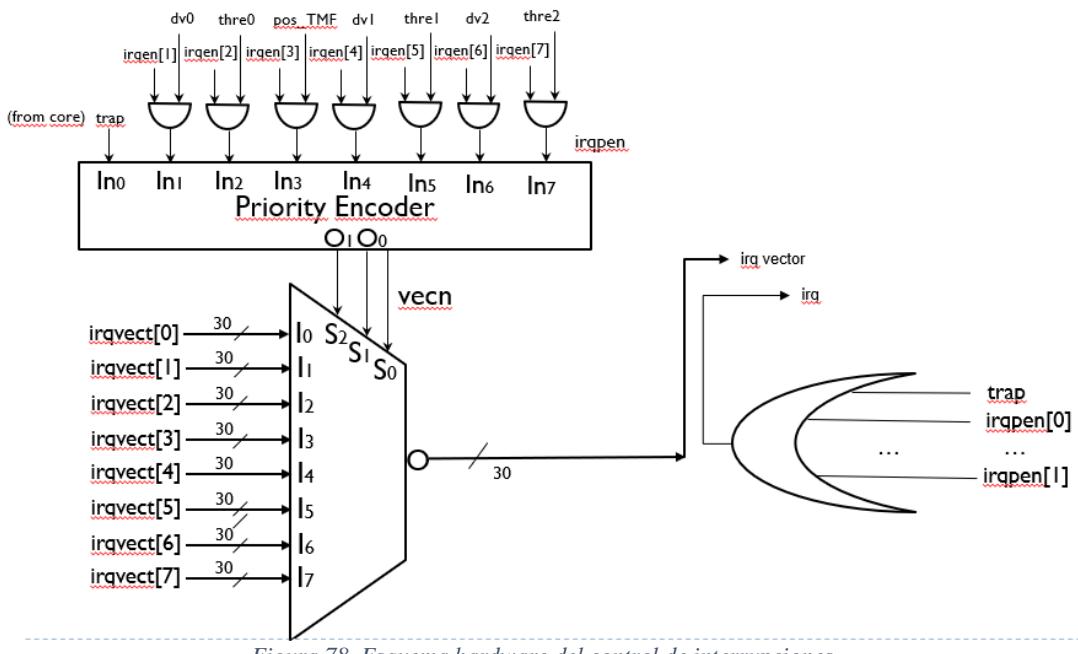


Figura 78. Esquema hardware del control de interrupciones.

La señal `irq` se habilitará cada vez que estando habilitada, una fuente de interrupciones dispare una interrupción e indicará al PC que se debe atender la interrupción. Por otra parte, el codificador de prioridad además de establecer cierta preferencia de unas interrupciones frente a otras devuelve 3 bits de selección que permiten al multiplexor inferior escoger el vector de interrupción correspondiente.

2.9.11. Pines.pcf

El archivo ‘pines.pcf’ se puede ver en el anexo D. Este describe la relación entre los pines reales de la FPGA y las entradas/salidas descritas en el hardware. En este fichero se han descrito todas las entradas y salidas desde y hacia el exterior de la FPGA. Los errores al asignar los pines en este archivo se pueden ver tras compilar en el fichero prn.txt. Por ejemplo en el archivo prn.txt se podría leer "ERROR: IO 'RXD0' is unconstrained in PCF (override this error with --pcf-allow-unconstrained)" lo que indicaría que no se ha asignado un pin a la entrada/salida RXD0 definida en el hardware.

2.9.12. Ampliación de la memoria

Debido a la gran cantidad de funcionalidades implementadas en el software, la memoria diseñada se llenó y fue necesario realizar una ampliación de la misma. Para ello, simplemente destacar la necesidad de instalar un compilador gcc para lo cual se siguieron los siguientes pasos:

- (1) Descargar el compilador "<https://sourceforge.net/projects/mingw/>"
- (2) Añadir el compilador mingw32gcc al path (Variables de entorno)
- (3) Comprobar que se reconoce \$gcc como comando

2.10. Programación del software

Tras definir el hardware de la FPGA y realizar el test de funcionamiento del dispositivo al completo, comenzamos la programación del software que se ejecutará en el sistema. Esta etapa pasa por la programación del microcontrolador auxiliar -encargado de la inicialización del sistema y concretamente, de la FPGA- y continúa con la programación de las rutinas propiamente dichas.

2.10.1. Programación del microcontrolador auxiliar

El microcontrolador auxiliar incluido en la PCB0 es el LPC1112, es el encargado de la inicialización y configuración de la placa en una primera instancia. Esta configuración la carga mediante el programa que se encuentra en la carpeta ‘tools’, más concretamente ‘lpc11loader’.

2.10.2. Programación de las rutinas

El código principal del datalogger se ha organizado en distintos ficheros .c que se incluyen en el fichero main.c del programa. Estos ficheros y las funciones aquí descritas se encuentran en el subdirectorio `Firmware` del proyecto.

Fichero principal: ‘main.c’

Este fichero sirve en primer lugar para mapear las direcciones de todos los registros de entrada/salida, entre ellos se encuentran la UART0, la UART1, la UART2, el SPI, el SPI del transceptor LoRa, el temporizador y su contador, o la habilitación de interrupción. Además, se define la frecuencia de reloj de funcionamiento y los pines de salida.

Tras definir varias funciones de utilidad para imprimir por pantalla o para conocer el tamaño de un *string*, definimos todas aquellas variables de las que vamos a hacer uso durante la ejecución del programa para manejar todos los registros.

Rutina uint8_t _getch()

Nos encontramos con la función `uint8_t _getch()` que se utiliza para leer datos de la UART0 a través de la estructura de datos FIFO y que toma un solo argumento, una variable de 8 bits llamada ‘d’, que se utilizará para almacenar el dato leído.

A través del *while* comprobamos si la variable ‘rdix0’ es igual ‘wrix0’. Si dichas variables son iguales significa que la FIFO está vacía y, por lo tanto, la función se detiene en espera bloqueante hasta que se reciba algún dato. Una vez que se recibe un dato a través del FIFO de la UART0 se lee y almacena, y se incrementa el valor de ‘rdix0’ para dejarlo apuntando al siguiente dato. Finalmente se realiza un direccionamiento circular, es decir que si ‘rdix0’ supera el valor 31 volverá a tomar el valor y se retorna el valor leído.

Rutina uint8_t haschar()

La función `uint8_t haschar()` se utiliza para determinar si hay algún carácter disponible para leer en la UART0 a través de la FIFO, se realiza la resta de las variables ‘rdix0’ y ‘wrix0’ con el objetivo de conocer la cantidad de caracteres disponibles en la FIFO para ser leídos.

Rutina uint32_t getMEPC()

La función `uint32_t getMEPC()` se utiliza para obtener el valor del contador del programa, utiliza la sintaxis ‘attribute(naked)’ para poder escribir en código ensamblador directamente. Contiene un bloque ‘asm volatile’ con el código ensamblador específico para el microcontrolador, a través de las dos instrucciones ‘word’ accedemos a los registros y obtenemos la dirección del programa. Finalmente, mediante ‘ret’ le indicamos que puede devolver el control al programa principal.

Rutina uint8_t *_memcpy(uint8_t *pdst, uint8_t *psrc, uint32_t nb)

La función `uint8_t *_memcpy(uint8_t *pdst, uint8_t *psrc, uint32_t nb)` se ha utilizado para copiar cadenas de una dirección a otra y así lograr concatenar texto para transmitir a través del LoRA. A través de sus argumentos se indica hacia dónde se quieren copiar los datos “pdst”, desde donde “psrc” y la cantidad de datos que se quieren copiar “nb”.

Rutina void _printfBin(unit8_t byte)

La función `void _printfBin(unit8_t byte)` toma como argumento un número entero sin signo de 8 bits con el objetivo de mostrarlo en formato binario y hexadecimal. Ha sido muy útil y se utiliza principalmente para conocer el valor que poseen los pines de entrada GPIN aunque también se ha utilizado en depuración.

Rutina void my_itoa(long i, char *string)

Mediante `void my_itoa(long i, char *string)` convertimos el valor entero ‘long’ en una cadena de caracteres. Nos será de gran utilidad a la hora de trabajar con el transceptor LoRa.

Rutina de interrupción void irq0_handler()

Tras definir todos los vectores de interrupción para las UARTs y temporizador pasamos a definir los manejadores de cada interrupción. En primer lugar, nos encontramos con `void irq0_handler()` encargada de manejar las interrupciones de tipo TRAP, nos servirá para conocer cuando se producen dichas interrupciones y cuál es el valor del contador de programa en ese momento.

Rutina de interrupción void irq1_handler()

La función `void irq1_handler()` se encarga de manejar las interrupciones de recepción de la UART0, es llamada cuando se recibe un dato a través de la UART0. Mediante la macro `UART0DAT` se accede al registro de datos de la uart y se lee el dato, éste se almacena en un buffer circular en la posición que indica ‘wrix0’, se incrementa el puntero para señalar la siguiente posición en el buffer y mediante una máscara nos aseguramos de que nunca se salgo de su rango de 0 a 31.

Rutina de interrupción void irq2_handler()

La función `void irq2_handler()` se ejecutará cuando se genere una interrupción de transmisión en la UART0, tiene una variable llamada ‘a’ que se utiliza para almacenar un valor que se transmite a través de la uart.

Rutina de interrupción void irq3_handler()

La función `void irq3_handler()` se encargará de manejar las interrupciones del temporizador. Cada vez que el *timer* llegue al final de la cuenta entrará. Mediante la variable ‘a’ se actualizará el valor del contador de tiempo TCNT y a través un *switch statement* se contemplará cada caso de operación del temporizador. Esto es así porque sólo contamos con un módulo temporizador y éste es necesario para varias tareas. En el primer caso, el *timer* controla el modo de parpadeo del LED, donde el valor del GPOUT se actualizará para poder contar de 0 a 15 a través de los LEDs disponibles en la placa PCB1. El resto de los casos se corresponden con rutinas para la definición de las señales de control de los sensores analógicos:

- Sensor de gas MQ-9:

Dado que el sensor de gas requiere de dos pulsos de 5 y 1,4 voltios con unas determinadas duraciones, con el segundo y tercer caso podremos controlar dichas señales. Cuando el sensor de gas posee los 5 voltios medirá el valor de CO (monóxido de carbono), por lo que en el segundo caso leerá el valor analógico que viene del ADC por el canal cero y lo transformará a voltios, activando posteriormente un pin de GPOUT determinado para indicarnos que se encuentre en 5 voltios. Activará el modo correspondiente a la señal de 1,4 voltios ya que es este valor con el que debe comutar y configurará la duración del pulso a 60 segundos.

Cuando el sensor de gas posee 1.4 voltios medirá el valor de CH4 (metano) o LPG (gas licuado del petróleo), por lo que en el tercer caso se leerá el valor analógico del ADC a través del canal cero y de igual manera al caso del CO, activará el pin correspondiente de GPOUT, conmutará el modo para pasar a 5V y configurará la duración del pulso a 90 segundos. La figura 77 muestra, simultáneamente, la señal de control de entrada al sensor y la salida que el sensor proporciona para distintas concentraciones de gas.

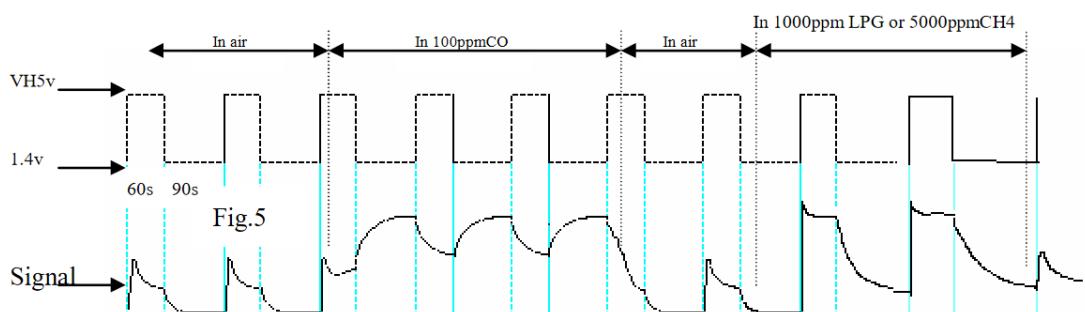


Figura 79. Señal de control del sensor de gas y señal de salida para distintas concentraciones de gas.

- Sensor de polvo:

En el cuarto caso del temporizador se maneja una interrupción destinada al control del sensor de polvo. En primer lugar, se activa la señal de control de dicho sensor y se activa el LED correspondiente a través de GPOUT. Como podemos observar en la figura 78, para medir el valor de polvo se requiere muestrear el valor obtenido del ADC con un retardo de 28 milisegundos, recogemos entonces el valor analógico del ADC junto con su conversión a milivoltios. Según nos indica la figura 79, la duración del pulso es de 32 milisegundos, por lo que introducimos un nuevo retardo hasta llegar a dicha duración y ponemos en baja la señal de control.

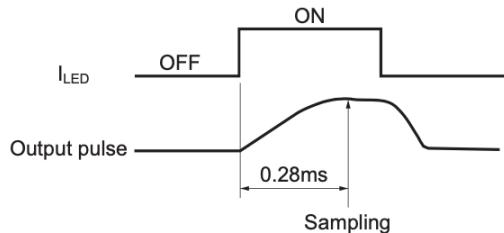


Figura 80. Tiempo de muestreo del pulso de salida en el sensor de polvo.

Pulse-driven wave form

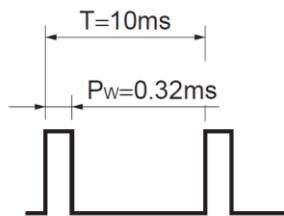


Figura 81. Señal de control del sensor de polvo.

Rutina de interrupción void irq4_handler()

La función `void irq4_handler()` se encarga de manejar las interrupciones de recepción que se producen en la UART1, dentro de ella nos podemos encontrar dos modos, uno de ellos destinado a volcar la trama recibida a través del GPS, y otro destinado a decodificar las tramas. En el momento en el que se recibe a través de la UART1 el fin de trama ‘r’ lo indicamos activando el *flag* GPF_FF. En ambos casos se vuelca el dato en la FIFO y se actualiza el puntero de escritura.

Rutina de interrupción void irq5_handler()

La función `void irq5_handler()` se ejecutará cuando se genere una interrupción de transmisión en la UART1, tiene una variable llamada ‘a’ que se utiliza para almacenar un valor que se transmite a través de la uart.

Rutina de interrupción void irq6_handler()

La función `void irq6_handler()` se encarga de manejar las interrupciones de recepción de la UART2. Accederá al registro de datos de la uart y se leerá el dato, éste se almacena en un buffer circular en la posición que indica ‘wrix2’ y se incrementa el puntero para señalar la siguiente posición en el buffer.

Rutina de interrupción void irq7_handler()

La función `void irq7_handler()` de igual manera al resto de funciones dedicadas a manejar las interrupciones de transmisión, lo hará en esta ocasión con la UART2.

Finalmente nos encontramos con las funciones de utilidad referentes a cada unidad UART y la función `void main()`. En dicha función principal se comienzan declarando todas aquellas variables que serán utilizadas, se establecen las velocidades de transmisión para

la UART0, UART1 y UART2, se limpian sus buffers de entrada leyendo los datos almacenados y asignándolos a otra variable.

Se establecen las rutinas de interrupción para las distintas interrupciones que se manejarán en el programa utilizando un puntero a función para asignar la rutina correspondiente a cada IRQ. En último lugar, se llama a una función `test()` que se procederá a describir a continuación.

Fichero de test: 'test.c'

Podremos interactuar con nuestro *datalogger* a través de la función `test()` incluida en el fichero raíz `main.c`. En primer lugar, inicializaremos a valor nulo las variables que necesitamos para trabajar, como puede ser la habilitación de interrupciones o los pines de salida, y definiremos los registros de configuración del sensor BME y del transceptor LoRa. Junto con la visualización por pantalla del logo del grupo definido por consola habilitaremos la interrupción correspondiente al temporizador y configuraremos el reloj.

Algunas de las operaciones que podremos realizar en nuestro *datalogger* son bloqueantes y otras no. Las acciones se han programado son la siguientes:

1. Leer los registros del transceptor LoRa.
2. Leer los registros del sensor BME.
3. Leer los canales del convertidor analógico-digital.
4. Activar o desactivar la señal de control referente al step-up.
5. Activar o desactivar la señal de control referente al sensor de polvo.
6. Activar o desactivar la señal de control referente a la señal de 1.4V.
7. Activar o desactivar la señal de control referente a la señal de 5V.
8. Hacer un reset.
9. Probar el temporizador de los LEDs.
10. Decodificar la trama GPS.
11. Pasar la salida del GPS (UART1) a la UART0.
12. Pasar la salida de la UART2 a la UART0.
13. Leer el estado del temporizador.
14. Visualizar el menú a través de la UART0.
15. Enviar datos a través de la UART0 vía interrupciones.
16. Lectura de los pines de entrada.
17. Activar el sensor de gas
18. Activar el sensor de polvo.
19. Transmitir datos a través del LoRa.
20. Visualizar los datos de gas y polvo extraídos de los sensores.

GPS: 'gps.c'

Dada la funcionalidad de un *datalogger*, una de las facetas más interesantes puede ser la transmisión de datos obtenidos de un GPS. Para llevar a cabo esta tarea se capturarán los datos pertinentes a través de la UART1, los cuales son capturados a través del sensor GPS con un formato concreto de codificación denominado NMEA.

En el momento que habilitemos en el menú principal del *datalogger* la interrupción correspondiente para recibir datos del GPS acudiremos a la función principal que nos

presta este servicio, `uint8_t getGPSFrame()`. En primer lugar, dado que la información que queremos extraer de las tramas NMEA se manda con unas cabeceras concretas, se nos avisará de que nos encontramos esperando la recepción de una de dichas tramas.

Estaremos en conocimiento de que las tramas son las indicadas a través nuestra función propia de comparación de cadenas `int strncmp(char *string1, char *buscar)`. A continuación mediante la función `int parse_comma_delimited_str(char *string, char **fields, int max_fields)` extraeremos cada campo de la trama con formato NMEA.

Finalmente, mostraremos los datos que consideramos de relevancia. De las tramas ‘\$GNGGA’ y ‘\$GPGGA’ extraeremos la hora en formato UTC, los datos de localización a través de la latitud, longitud y altitud, junto con el número de satélites con los que nuestro sensor se ha comunicado para posibilitarnos dicha información. De las tramas ‘\$GNRMC’ y ‘\$GPGGA’ nos quedaremos con la fecha dada por día, mes y año.

Destacar que cabe la posibilidad de que no se nos pueda proporcionar toda la información mencionada por parte de los satélites, ya que por ejemplo para obtener todos los datos de geolocalización es necesario que nuestro sensor GPS tenga visión y se comunique con cuatro satélites de manera simultánea, tarea que puede llevarle un tiempo.

Sensores: ‘spiSensors.c’

Dado que todos los sensores utilizados van a transmitir, de una u otra manera, sus datos mediante SPI, se decidió dedicar uno de los dos módulos SPI implementados (concretamente, el SPI0) íntegramente a la recepción de estos datos. El fichero `spiSensors.c` contiene las rutinas de control de este periférico.

`uint8_t spixfer(uint8_t d):` transfiere por SPI el dato que se pasa como argumento de entrada. Para ello, escribe este dato en la dirección de memoria `0xE0000020`, correspondiente al registro de datos del SPI0. Después espera a que finalice la transmisión, comprobando el bit `busy`. Cuando finalice la transmisión, este bit se pondrá a 0. La función entonces devolverá el contenido del registro de datos, en el que ahora se habrá almacenado el dato recibido del MISO.

Sensor de temperatura, presión y humedad BME60:

Para comunicar con el BME680 a través de SPI se ha seguido el siguiente esquema que proporciona el fabricante:

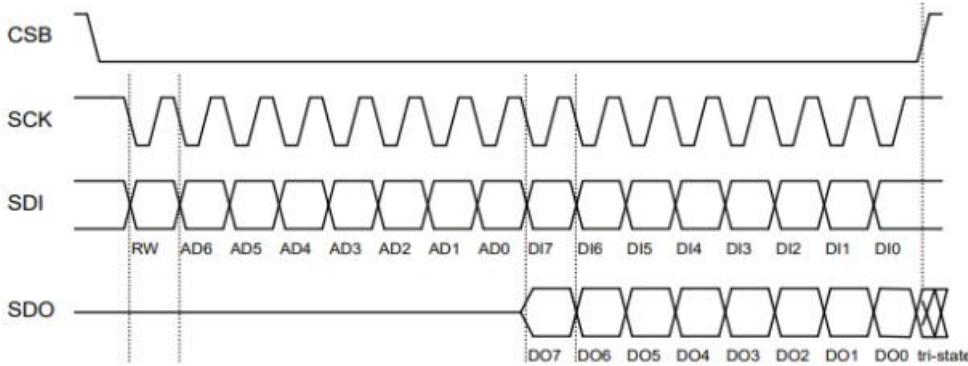


Figura 82. Cronograma de comunicación SPI con el sensor BME680

`char readBME680(char dir)`: lee de un registro arbitrario del sensor BME. La dirección del registro se introduce como parámetro de entrada. Para realizar la lectura por SPI, primero hay que poner el bit CS a cero. Después se envían 16 bits por el MOSI. El primero de estos bits corresponde al bit de lectura/escritura. Ponerlo a 1 implica realizar una lectura. Los siguientes 7 bits corresponden a la dirección del registro que se desea leer. Los 8 bits restantes se enviarán, pero no tendrán ningún efecto en el registro, por lo que su contenido es irrelevante (se pondrán a cero por convención). Durante la transmisión de este *dummy byte*, se leerá, a través del MISO, el contenido del registro. Por último, es importante volver a fijar el bit CS a 1.

`void writeBME680(char data, char dir)`: escribe en un registro arbitrario del sensor BME. Como parámetros de entrada se pasan la dirección del registro y el dato a escribir. Primero se pone a cero el bit CS. A continuación, se transmiten, el bit r/w (que ahora valdrá 0 debido a que queremos escribir), la dirección del registro y el byte que se quiere introducir en él.

`void readAllBMERegs()`: lee los 256 registros del sensor BME y los almacena en la matriz `bmeRegs`, que es una variable global.

`void printBMERegs()`: imprime por pantalla (UART0) el contenido de la matriz `bmeRegs`, de forma que se puede visualizar el estado de cada uno de los registros.

`char bmeReg(char dir)`: función auxiliar que permite obtener el valor de un elemento concreto de la matriz `bmeRegs`. Convierte el argumento de entrada (un número entre 0 y 255) en una fila y una columna, y devuelve el valor del elemento correspondiente.

`void startBME680()`: inicializa el sensor BME680. Esta inicialización, necesaria antes de leer cualquier medida que haga el sensor, se basa en la escritura secuencial de una serie de registros de configuración del sensor.

`void measureBME680()`: realiza las medidas de temperatura, presión y humedad leídas por el sensor. Para ello, hace primero una llamada a `readAllBMERegs()`, ya que la información de todas estas variables meteorológicas se encuentra almacenada en los registros del sensor. Una vez obtenido el contenido de estos registros, será necesario procesar estos datos, concatenándolos y realizando distintas operaciones con ellos, para

poder obtener finalmente medidas legibles y con sentido. Todas las operaciones con los registros están recogidas en el *datasheet* del sensor.

Sensor de gas MQ-9 y sensor de partículas GP2Y1010AU0F:

Estos dos sensores, a diferencia del BME680, son analógicos, por lo que es necesaria, tanto una etapa de acondicionamiento de sus salidas, como una digitalización de estas. Para la digitalización, cada sensor pasará su salida (una vez acondicionada) a un canal del ADC integrado en el sistema. Por lo tanto, no son los sensores analógicos los que se comunican con el periférico SPI0, sino que en realidad es el ADC el encargado de esta tarea. Para comunicar con el ADC a través de SPI se ha seguido el siguiente esquema que proporciona el fabricante:

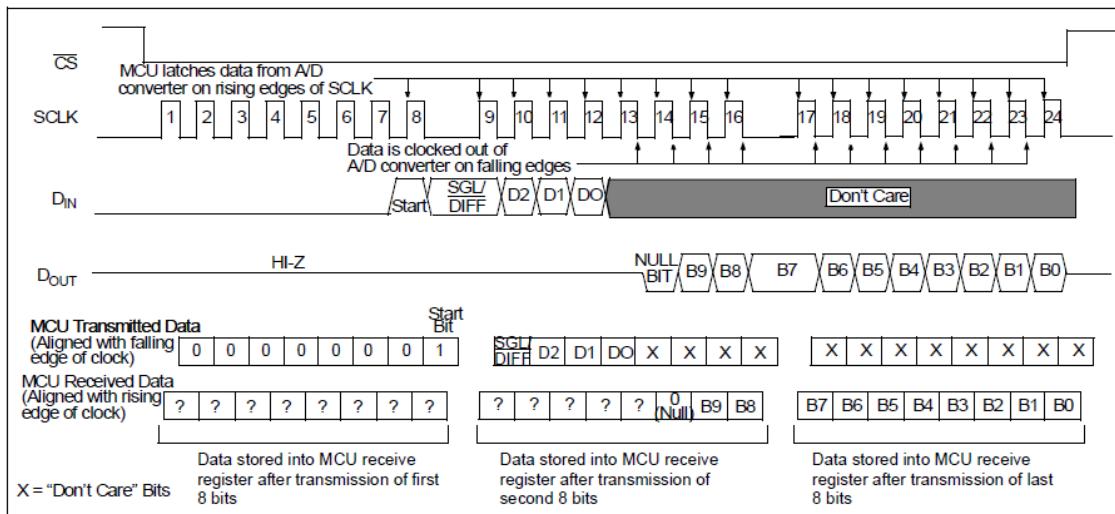


Figura 83. Cronograma de comunicación SPI con el ADC.

`uint32_t ReadADC(char cmd_ch)`: lee un canal del ADC. Para ello, activa primero el CS (el cual ya se ha visto que es activo en baja) correspondiente al ADC. En este caso, para leer, el primer bit a 1 que se transmita por el MOSI se corresponderá con el bit de *start*. A continuación, se transmiten tres bits de selección de canal y se comienza a recibir el byte más significativo del dato. Mientras estemos recibiendo este dato, lo que se envíe por el MOSI es irrelevante, tal y como se ha visto anteriormente. Despues, se envía un *dummy byte* a la vez que se recibe el segundo byte, el menos significativo. Para conformar el dato, se concatena en una variable auxiliar, B, el byte más significativo y el menos significativo, en el orden adecuado. Por último, se aplica una máscara de 10 bits y se devuelve el resultado.

`void printAdcChannels()`: imprime por pantalla el valor de los voltajes (en mV) de entrada a los 4 canales del ADC. Para obtener estos voltajes de entrada a partir de los datos de salida (que son a los que se tiene acceso directo), se aplica la función de transferencia inversa del ADC:

$$V_{in} = \frac{Output\ Code \cdot V_{ref}}{1024}$$

Cabe destacar que el contenido de los canales 3 y 4 del ADC será aproximadamente cero, ya que a su entrada no hay conectado ningún sensor.

`void ReadDust()`: pone al dispositivo en el modo de lectura del sensor de polvo. Para ello, primero activa el *step up* y el LED 4, pone el reloj en el modo correspondiente y lo configura cada (10 ms - 0,32ms) = 9,68ms, que se corresponde con el tiempo durante el que la señal de control del sensor está desactivada.

`void ReadGAS()`: pone al dispositivo en el modo de lectura del sensor de gas. Para ello, pone el reloj en el modo correspondiente.

`void printCO()`, `void printCh4LPG()`, `void printDust()`: imprimen por pantalla el valor de las variables globales en las que se almacena la información sobre la concentración de monóxido de carbono, metano y polvo, respectivamente. Estos valores están representados en mV, es decir, que hacen referencia a la salida analógica de los sensores.

LoRa: 'spiLoRa.c'

Para comunicar a través de SPI el dispositivo LoRA con la FPGA se ha seguido el siguiente esquema de comunicaciones proporcionado por el fabricante:

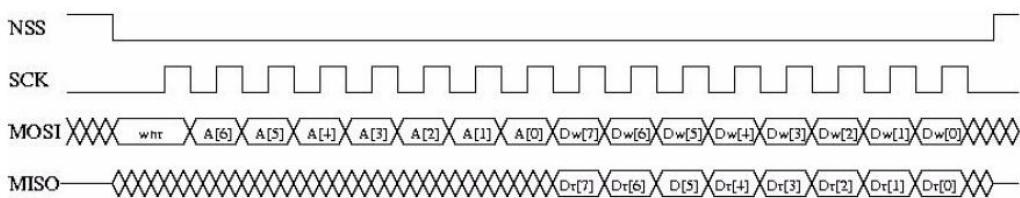


Figura 84. Esquema de comunicación SPI con el LoRA

Este fichero contiene las rutinas de control del periférico SPI1, encargado de la comunicación con el módulo LoRa Semtech SX1276. Las funciones, tanto de transmisión y recepción por SPI, como de escritura y lectura de registros, son muy similares a las del fichero `spiSensors.c`, debido a que la configuración y uso del módulo LoRa es muy similar a la del BME, funciona todo mediante registros.

`uint8_t spiLoRAXfer(uint8_t d)`: transfiere por SPI el dato que se pasa como argumento de entrada. Para ello, escribe este dato en la dirección de memoria 0xE0000030, correspondiente al registro de datos del SPI1. Después espera a que finalice la transmisión, comprobando el bit *busy*. Cuando finalice la transmisión, este bit se pondrá a 0. La función entonces devolverá el contenido del registro de datos, en el que ahora se habrá almacenado el dato recibido del MISO.

`char readLoRA(char dir)`: lee el contenido de un registro del módulo LoRa, activando primero el CS (activo en baja). Después, envía el byte correspondiente a la dirección. Es importante ver cómo, a diferencia del BME, aquí el bit de escritura/lectura debe valer 0 si queremos escribir, y 1 si queremos leer. El contenido del registro llega por el MISO y es leído mientras por el MOSI se transmite un *dummy byte*. Por último, vuelve a poner a 1 el bit CS. La función devuelve el contenido del registro leído.

`void writeLoRA(char data, char dir)`: escribe en un registro del módulo LoRa. Es análoga a `writeBME680()`, salvo por el detalle del bit de escritura/lectura.

`void printLoRaRegs ()`: imprime por pantalla, en forma de matriz, el contenido de los 127 registros del módulo LoRa.

`uint8_t loraInit ()`: inicializa el módulo LoRa. Hay que llamar a esta función antes de enviar/recibir cualquier dato. La inicialización se basa en la escritura, en un determinado orden, de una serie de registros, de forma que se pueda cambiar el modo de operación del módulo y fijar parámetros importantes de la comunicación, como son la frecuencia, el tamaño de las cabeceras o la potencia de transmisión, entre otros.

Concretamente, primero se pone el módulo en modo `LONG_RANGE` (LoRa) y `SLEEP`, escribiendo en el registro `RegOpMode`. Es necesario estar en este modo para fijar las direcciones base de transmisión y recepción dentro de la FIFO del módulo, que estarán ambas en 0x00. Después se pone el módulo en modo `IDLE` y se llama a las funciones `setModemRegisters ()`, `setPreambleLength (8)` y `setFrequency898 ()`. Por último, se fija la potencia de transmisión a 15 dBm, escribiendo 0xAD en el registro `RegPaConfig`.

`void setModemRegisters ()`: configura el modem mediante la escritura en los registros `RegModemConfig1`, `RegModemConfig2` y `RegModemConfig3`.

`void setPreambleLength (uint16_t bytes)`: fija la longitud del preámbulo de las tramas LoRa que se transmitirán. Para ello, escribe en los registros `RegPreambleMsb` y `RegPreambleLsb`.

`void setFrequency868 ()`: fija la frecuencia de la comunicación (portadora) a 868 MHz, escribiendo en los tres registros `RegFrf`.

`void loraSend (char *mensaje)`: escribe en los registros del módulo LoRa necesarios para que este transmita la cadena de texto que se pasa como argumento de entrada. Primero se activa el modo `IDLE` y se pone el puntero de la FIFO (el cual apunta a la dirección en la que se comenzará la escritura) en la dirección 0x00. Se escribe después en el registro `RegPayloadLength`, en el que se define la longitud, en bytes, del mensaje a transmitir. Esta longitud será igual al tamaño de la cadena de texto a transmitir más 4 bytes de cabeceras. Después se escribe todo el mensaje en la FIFO, comenzando por las 4 cabeceras que acabamos de mencionar, y acto seguido los caracteres de la cadena de texto. Una vez escrito el mensaje, se activa el modo transmisión, en el que el módulo transmitirá automáticamente todo lo que encuentre en la FIFO desde la dirección 0x00. Cuando termine, activará el flag `TxDone` y además, volverá automáticamente al modo `IDLE`. Una vez transmitido el dato, la función baja el flag `TxDone` y devuelve el puntero de la FIFO a la dirección 0x00.

`void transmitPRAOFrame ()`: esta función es llamada desde el programa principal. Realiza todo lo necesario para que el módulo LoRa transmita la información recogida por el módulo GPS, el sensor BME y los sensores analógicos de polvo y gas. Por lo tanto, primero debe inicializar el módulo llamando a la función `loraInit ()`. Después, hace uso reiteradamente de la función `loraSend ()`, para enviar las diferentes cadenas de texto con la información de los sensores. Para conformar estas cadenas de texto, se hace uso de la función `_memcpy ()`, para concatenar *arrays* de caracteres. La información de los sensores se recoge de variables globales.

GPIN: 'gpinc.c'

Puede ser de utilidad detectar a través de uno de los pines si existe una señal de entrada externa al circuito físico. Por ello a través de la función `GpinRead()` podremos leer el estado de los pines 32 al 39 correspondientes al conector J1 de nuestras placas de circuito impreso puesto que han sido los pines correspondientes a los de la FPGA asignados.

2.11. Verificación final

Tras finalizar el proceso de diseño y fabricación del datalogger, se ha verificado su correcto funcionamiento tanto de hardware como de software. Todos los componentes se encuentran funcionando correctamente y no se han cometido errores software significativos.

Anexo A: Código en C

Código de main.c

```
// =====
// Proyecto Datalogger for IoT Curso 2022-2023
// Fecha: 26/01/2023
// Autor: Pablo Villacorta, Rubén Serrano, Óscar Martín y Andrés Martín
// Asignatura: Taller de Proyectos I
// File: main.c Programa principal
// =====

#include <stdint.h>

typedef unsigned char u8;
typedef unsigned short u16;
typedef unsigned int u32;

typedef signed char s8;
typedef signed short s16;
typedef signed int s32;

// =====
// ----- REGISTROS MAPEADOS -----
// =====

#define UART0DAT (*(volatile uint8_t*)0xE0000000) //UART0 DATA
#define UART0STA (*(volatile uint32_t*)0xE0000004) //UART0 STATE
#define UART0BAUD (*(volatile uint32_t*)0xE0000004) //UART0 BAUD DIVIDER

#define UART1DAT (*(volatile uint8_t*)0xE0000008) //UART1 DATA
#define UART1STA (*(volatile uint32_t*)0xE000000C) //UART1 STATE
#define UART1BAUD (*(volatile uint32_t*)0xE000000C) //UART1 BAUD DIVIDER

#define UART2DAT (*(volatile uint8_t*)0xE0000010) //UART2 DATA
#define UART2STA (*(volatile uint32_t*)0xE0000014) //UART2 STATE
#define UART2BAUD (*(volatile uint32_t*)0xE0000014) //UART2 BAUD DIVIDER

#define SPIDAT (*(volatile uint32_t*)0xE0000020) //ICE_SPI DATA
#define SPICTL (*(volatile uint32_t*)0xE0000024) //ICE_SPI CONTROL
#define SPISTA (*(volatile uint32_t*)0xE0000024) //ICE_SPI STATE
#define SPISS (*(volatile uint32_t*)0xE0000028) //ICE_SPI SLAVE SELECT

#define SPILDAT (*(volatile uint32_t*)0xE0000030) //LORA_SPI DATA
#define SPILCTL (*(volatile uint32_t*)0xE0000034) //LORA_SPI CONTROL
#define SPILSTA (*(volatile uint32_t*)0xE0000034) //LORA_SPI STATE
#define SPILSS (*(volatile uint32_t*)0xE0000038) //LORA_SPI SLAVE SELECT

#define TCNT (*(volatile uint32_t*)0xE0000060) //TIMER COUNTER REGISTER

#define GPOUT (*(volatile uint8_t*)0xE0000080) // GPOUT
#define GPIN (*(volatile uint8_t*)0xE0000084) // GPIN

#define IRQEN (*(volatile uint32_t*)0xE00000C0) //INTERRUPT ENABLE
#define IRQVECT0 (*(volatile uint32_t*)0xE00000E0) //TRAP
#define IRQVECT1 (*(volatile uint32_t*)0xE00000E4) //RX0
#define IRQVECT2 (*(volatile uint32_t*)0xE00000E8) //TX0
#define IRQVECT3 (*(volatile uint32_t*)0xE00000EC) //TIMER
#define IRQVECT4 (*(volatile uint32_t*)0xE00000F0) //RX1
#define IRQVECT5 (*(volatile uint32_t*)0xE00000F4) //TX1
#define IRQVECT6 (*(volatile uint32_t*)0xE00000F8) //RX2
#define IRQVECT7 (*(volatile uint32_t*)0xE00000FC) //TX2

void delay_loop(uint32_t val); // (3 + 3*val) cycles
```

```

#define CCLK (18000000)      // 18 MHz Frec de Reloj
#define _delay_us(n) delay_loop((n*(CCLK/1000)-3000)/3000)
#define _delay_ms(n) delay_loop((n*(CCLK/1000)-30)/3)

// GPOUT
#define ice_led1    (0b00000001)  //GPOUT0 -> ice_led1
#define ice_led2    (0b00000010)  //GPOUT1 -> ice_led2
#define ice_led3    (0b00000100)  //GPOUT2 -> ice_led3
#define ice_led4    (0b00001000)  //GPOUT3 -> ice_led4
#define STEPUP_CE   (0b00010000)  //GPOUT4 -> STEPUP_CE
#define GAS_5V_CTRL (0b00100000)  //GPOUT5 -> GAS_5V_CTRL
#define GAS_1V4_CTRL (0b01000000)  //GPOUT6 -> GAS_1V4_CTRL
#define DUST_CTRL   (0b10000000)  //GPOUT7 -> DUST_CTRL

void _putch(int c)
{
    while((UART0STA&2)==0); // When THRE = 0 (Uart0) [Espera a que no este ocupado el THR]
    //if (c == '\n') _putch('\r');
    UART0DAT = c; //Escribo el dato entero a transmitir (4bytes)
}

void _puts(const char *p)
{
    while (*p)
        _putch(*(p++));
}

uint8_t _sizeof(char *string){ // Tamaño de un array
    uint8_t size = 1;
    while((*string++)!='\0'){
        size++;
    }
    return size;
}

#define putchar(d) _putch(d)
#include "printf.c"

const static char *menuTxt="\n"
"\n\n"
"8888888888 88888888 8888888888 8888888888\n"
"888 888 888 Y88b 888 888 888 888\n"
"888 888 888 888 888 888 888 888\n"
"8888888888 888 d88P 888 888 888 888\n"
"888 8888888P 8888888888 888 888\n"
"888 888 T88b 888 888 888 888\n"
"888 888 T88b 888 888 888 888\n"
"888 888 T88b 888 888 8888888888\n"
"\nIts Alive :-)\n"
"\n";

const static char *gpoutTxt="-- GPOUT ASSIGNMENTS: --\n"
"GPOUT[0] -> ice_led1\n"
"GPOUT[1] -> ice_led2\n"
"GPOUT[2] -> ice_led3\n"
"GPOUT[3] -> ice_led4\n"
"GPOUT[4] -> STEPUP_CE\n"
"GPOUT[5] -> GAS_5V_CTRL\n"
"GPOUT[6] -> GAS_1V4_CTRL\n"
"GPOUT[7] -> DUST_CTRL\n"
"-----\n";

//FIFO UART 0:
uint8_t udat0[32]; //FIFO de recepcion para la UART0 (tamaño 32 bits)
volatile uint8_t rdix0,wrix0; // Punteros de lectura y escritura (unsigned char, otra notacion que viene en el include, 8 bit)

```

```

//FIFO UART 1 (GPS)
uint8_t udat1[128]; //FIFO de recepcion para la UART0 (tamaño 32 bits)
volatile uint8_t rdix1,wrix1; // Punteros de lectura y escritura (unsigned char, otra notacion que viene en el include, 8 bit)

//FIFO UART 2 (GPS)
uint8_t udat2[128]; //FIFO de recepcion para la UART0 (tamaño 32 bits)
volatile uint8_t rdix2,wrix2; // Punteros de lectura y escritura (unsigned char, otra notacion que viene en el include, 8 bit)

uint8_t GPS_FF = '0'; //Full Frame Flag (GPS)
uint8_t GPS_FRAME[80];

// Timer
uint8_t clkMode = 0;
uint8_t binaryCount = 0;

// GPS
uint8_t volcarOutputGPS=0;

// SensorAnalogico GAS:
uint16_t Ch4LPGValue=0;
uint16_t COValue=0;
uint16_t polvoValue=0;

// Variables BME
uint8_t temp = 50;
uint16_t presion = 0;
uint8_t humedad = 0;

// Variables GPS
char UTC_time[10]="";
char date[10]="";

// -- LECTURA UART0 -----
uint8_t _getch() //leer de la uart0 a través de la fifo
{
    uint8_t d;
    while(rdx0==wrix0{}) //fifo vacia, espera bloqueante

    d=udat0[rdix0++]; //leer el dato e incremento el puntero después para colocarlo en el siguiente dato
    rdix0&=31; //direcccionamiento circular (mirar escritura)
    return d;
}

uint8_t haschar() {return wrix0-rdx0; }

// -----
// Return PC
uint32_t __attribute__((naked)) getMEPC()
{
    asm volatile(
        // csrrw a0,0x341,zero \n"
        // csrrw zero,0x341,a0 \n"
        " .word 0x34101573 \n"
        " .word 0x34151073 \n"
        " ret \n"
    );
}

// -----
// Print Byte in binary & hex

void _printfBin(uint8_t byte){
    _printf("%d%d%d%d %d%d%d%d || %x\n",
    (byte & 0x80 ? 1 : 0),

```

```

        (byte & 0x40 ? 1 : 0),
        (byte & 0x20 ? 1 : 0),
        (byte & 0x10 ? 1 : 0),
        (byte & 0x08 ? 1 : 0),
        (byte & 0x04 ? 1 : 0),
        (byte & 0x02 ? 1 : 0),
        (byte & 0x01 ? 1 : 0),
        byte
    );
}
// ----

// ITOA -> Integer to String -----
void my_itoa(long i, char *string)
{
    int power = 0, j = 0;
    j = i;
    for (power = 1; j>10; j /= 10)
        power *= 10;
    for (; power>0; power /= 10){
        *string++ = '0' + i / power;
        i %= power;
    }
    *string = '\0';
}
//-----

#include "spiSensors.c" //Rutinas de test

// =====
// ----- INTERRUPCIONES -----
// =====

// HABILITACION DE INTERRUPCIONES IRQEN:
#define IRQEN_U0RX (0b00000010) //IRQ VECT 1 (1<<1)
#define IRQEN_U0TX (0b00000100) //IRQ VECT 2 (1<<2)
#define IRQEN_TIMER (0b00001000) //IRQ VECT 3 (1<<3)
#define IRQEN_U1RX (0b00010000) //IRQ VECT 4 (1<<4)
#define IRQEN_U1TX (0b00100000) //IRQ VECT 5 (1<<5)
#define IRQEN_U2RX (0b01000000) //IRQ VECT 6 (1<<6)
#define IRQEN_U2TX (0b10000000) //IRQ VECT 7 (1<<7)

void __attribute__((interrupt ("machine"))) irq0_handler() //TRAP
{
    _printf("\nTRAP at 0x%lx\n",getMEPC());
}

void __attribute__((interrupt ("machine"))) irq1_handler() // UART0 RX
{
    udat0[wrix0++]=UART0DAT; // Escribe el dato en la FIFO y postincremento del puntero de escritura
    wrix0&=31; // direccionamiento circular del puntero de escritura (es 31 porque tenemos buffer de 32) Mascara +
    // rapido que comparacion
    // if(wrix0==32) wrix0=0;
}

void __attribute__((interrupt ("machine"))) irq2_handler(){ //UART0 TX
    static uint8_t a=32;
    UART0DAT=a;
    if (++a>=128) a=32;
}

void __attribute__((interrupt ("machine"))) irq3_handler(){ //TIMER
volatile int a;
switch (clkMode)
{

```

```

case 0: // (0) LED BLINK MODE
    GPOUT = (GPOUT & 0b11110000)+binaryCount;
    binaryCount=binaryCount+1;
    if(binaryCount==0b00010000) //Cuenta hasta 16 (0 to 15)
        binaryCount=0;
    break;

case 1: // (1) GAS SENSOR MODE 5V Cicle -> Cuando salta activo 5V
    //Muestrear el final de 1v4
    COValue=((ReadADC(CMD_CH0)*3300)>>10);
    GPOUT = (STEPUP_CE|GAS_5V_CTRL|ice_led2); //Activa 5V Control

    clkMode=2; // Para que cuando salte el reloj pase a Modo 1V4V
    TCNT=(60*CCLK); //Configuramos el reloj para los 60 seg en 5V
    break;

case 2: // (2) GAS SENSOR MODE 1V4 Cicle -> Cuando salta activo 1v4
    //Muestrear el final de 5v
    Ch4LPGValue=((ReadADC(CMD_CH0)*3300)>>10);

    GPOUT = (STEPUP_CE|GAS_1V4_CTRL|ice_led1); //Activa 1V4 CTRL

    clkMode=1; // Para que cuando salte el reloj pase a Modo 5V
    TCNT=(90*CCLK); //Configuramos el reloj para los 90 seg en 1V4
    break;

case 3: // (3) Sensor de Polvo
    GPOUT |= DUST_CTRL; //Activa Dust Control y el led 4
    _delay_ms(0.28); //Delay 0,28 mseg
    polvoValue=((ReadADC(CMD_CH1)*3300)>>10); //Muestreo en mV del CAD

    _delay_ms(0.4); // Delay 0,4 mseg y bajo el pulso
    GPOUT ^= DUST_CTRL; //Desactiva dust control
    break;

default:
    _puts("Clk Mode Error");
    break;
}

a = TCNT;
}

void __attribute__((interrupt ("machine"))) irq4_handler() // UART1 (GPS) RX
{
    // (1) Modo Volcar
    if(volcarOutputGPS==1){
        _putch(UART1DAT);
    }
    // (2) Modo Decodificar tramas
    else{
        if((UART1DAT=='r'))
            GPS_FF='1'; //Activo el Flag que me indica fin de linea
    }

    udat1[wrix1++]=UART1DAT;
    wrx1&=127;
}

void __attribute__((interrupt ("machine"))) irq5_handler(){ //UART1 (GPS) TX
static uint8_t a=32;
UART1DAT=a;
if (++a>=128) a=32;
}

```

```

void __attribute__((interrupt ("machine"))) irq6_handler() // UART2 RX
{
    udat2[wrix2++]=UART2DAT;
    _putch(UART2DAT);
    wrx2&=127;
}

void __attribute__((interrupt ("machine"))) irq7_handler(){ //UART2 TX
    static uint8_t a=32;
    UART2DAT=a;
    if (++a>=128) a=32;
}

// =====

#define NULL ((void *)0)

// -----
// --- UART0 ---

#define BAUD0 115200

uint32_t getw()
{
    uint32_t i;
    i=_getch();
    i|=_getch()<<8;
    i|=_getch()<<16;
    i|=_getch()<<24;
    return i;
}

uint8_t *_memcpy(uint8_t *pdst, uint8_t *psrc, uint32_t nb)
{
    if (nb) do {*pdst++=*psrc++;} while (--nb);
    return pdst;
}
// -----


char* cnc(char* str1, char* str2){
    return _memcpy(_memcpy(str1, str1,_sizeof(str1)) - 1, str2, _sizeof(str2)) -1;
}

// -----
// --- UART1 ---

#define BAUD1 9600

//> LECTURA:

uint8_t _getch1() //LEE DE LA UART1 A TRAVÉS DE LA FIFO
{
    uint8_t d;
    while(rdx1==wrix1); //fifo vacia, espera bloqueante
    d=udat1[rdix1++]; //leer el dato e incremento el puntero despues para colocarlo en el siguiente dato
    rdx1&=127; //direcccionamiento circular (mirar escritura)
    return d;
}

void _putch2(int c) // ESCRITURA EN UART1
{

```

```

while((UART1STA&2)==0); // Comprueba el flag THRE
//if (c == '\n') _putch('\r');
UART1DAT = c;
}

// -----
// --- UART2 ---

#define BAUD2 9600

// ----

#include "spiLoRA.c" //Rutinas de test
#include "gps.c" //Rutinas de GPS (UART1)
#include "gpin.c" //Rutinas de GPIO
#include "test.c" //Rutinas de test

// =====
// ----- MAIN -----
// =====

void main()
{
    char c,buf[17];
    uint8_t *p;
    unsigned int i,j;
    int n;
    void (*PCODE)();
    uint32_t *pi;
    uint16_t *ps;

UART0BAUD = (CCLK+BAUD0/2)/BAUD0 -1;
UART1BAUD = (CCLK+BAUD1/2)/BAUD1 -1;
UART2BAUD = (CCLK+BAUD2/2)/BAUD2 -1;
_delay_ms(100);
c = UART0DAT; // Clear RX garbage
c = UART1DAT; // Clear RX garbage
c = UART2DAT; // Clear RX garbage

IRQVECT0=(uint32_t)irq0_handler; //TRAP
IRQVECT1=(uint32_t)irq1_handler; //UART0 RX
IRQVECT2=(uint32_t)irq2_handler; //UART0 TX
IRQVECT3=(uint32_t)irq3_handler; //Timer
IRQVECT4=(uint32_t)irq4_handler; //UART1 RX
IRQVECT5=(uint32_t)irq5_handler; //UART1 TX
IRQVECT6=(uint32_t)irq6_handler; //UART2 RX
IRQVECT7=(uint32_t)irq7_handler; //UART2 TX

test();
}

```

Código de test.c

```
// =====
// Proyecto Datalogger for IoT Curso 2022-2023
// Fecha: 26/01/2023
// Autor: Pablo Villacorta, Rubén Serrano, Óscar Martín y Andrés Martín
// Asignatura: Taller de Proyectos I
// File: test.c
// =====

void test(){
    IRQEN = 0;
    GPOUT = 0; //Inicializa el GPOUT a 0

    _puts(menutxt);
    _puts("Hola mundo\n");

    asm volatile ("ecall"); //Salta interrupcion Software
    asm volatile ("ebreak"); //Salta interrupcion Software

    SPICTL = (8<<8)|8; // Define Registro control SPI 0 (BME y ADC)
    startBME680(); //Programa los registros de configuracion

    SPILCTL = (8<<8)|8; // Define Registro control SPI 1 (LoRa)

    IRQEN = IRQEN_TIMER;
    TCNT=CCLK; //Configuramos el reloj cada segundo

    while (1)
    {
        IRQEN |= IRQEN_U0RX;
        _puts("\n--- TEST ---\n");
        _puts("-> z: Lee los registros del transceptor LoRa\n");
        _puts("-> 5: Lee los registros del sensor BME680\n");
        _puts("-> 4: Lee los canales del ADC\n");
        _puts("-> 6: Activa/desactiva STEPUP_CE, bit gpout[4]\n");
        _puts("-> 7: Activa/desactiva DUST_CTRL, bit gpout[7]\n");
        _puts("-> 8: Activa/desactiva GAS_1V4_CTRL, bit gpout[6]\n");
        _puts("-> 9: Activa/desactiva GAS_5V_CTRL, bit gpout[5]\n");
        _puts("-> q: Salta a la direccion 0 (casi como un reset)\n");
        _puts("-> t: Prueba el temporizador de los LED (T = 0.5 seg, al arrancar 1 seg) [BLOQ]\n");
        _puts("-> g: Decodificar Trama GPS (Espera trama)\n");
        _puts("-> h: La salida del GPS (UART1) a la UART0 [BLOQ]\n");
        _puts("-> k: La salida de la UART2 a la UART0 [BLOQ]\n");
        _puts("-> l: Lee estado del TIMER\n");
        _puts("-> 1: Pinta menu por UART0\n");
        _puts("-> 2: Envia datos por UART0 via interrupciones \n");
        _puts("-> 3: Lectura GPIN \n");
        _puts("-> G: Activar sensor GAS \n");
        _puts("-> P: Activar sensor Polvo \n");
        _puts("-> L: Transmitir datos por LoRA \n");
        _puts("-> T: Ver Gas/Polvo \n\n");

        _puts("Command [z456789rqtgkl123GPLT]> ");
        char cmd = _getch();
        if (cmd > 32 && cmd < 127)
            _putch(cmd);
        _puts("\n");
    }

    switch (cmd){
        case 'z': //Lee los registros del transceptor LoRa
            printLoRaRegs();
            break;
    }
}
```

```

case '5': //Lee los registros del sensor BME680
    readAllBMERegs();
    printBMERegs();
    measureBME680();
    break;

case '4': //Lee los canales del ADC
    printAdcChannels();
    break;

case '6': //Activa/desactiva STEPUP_CE
    GPOUT^= STEPUP_CE;
    _puts("GPOUT = ");
    _printfBin(GPOUT);
    _puts(gpoutTxt);
    break;

case '7': //Activa/desactiva DUST_CTRL
    GPOUT^= DUST_CTRL;
    _puts("GPOUT = ");
    _printfBin(GPOUT);
    _puts(gpoutTxt);
    break;

case '8': //Activa/desactiva GAS_1V4_CTRL
    GPOUT^= GAS_1V4_CTRL;
    _puts("GPOUT = ");
    _printfBin(GPOUT);
    _puts(gpoutTxt);
    break;

case '9': //Activa/desactiva GAS_5V_CTRL
    GPOUT^= GAS_5V_CTRL;
    _puts("GPOUT = ");
    _printfBin(GPOUT);
    _puts(gpoutTxt);
    break;

case 'q': //Salta a la dirección 0 (casi como un reset)
    asm volatile ("jalr zero,zero");
    break;

case 't': //Prueba el temporizador de los LED (periodo 0.5 segundos, al arrancar 1 segundo)
    _puts("Secuencia Iniciada");
    clkMode=0;
    IRQEN |= IRQEN_TIMER; //Habilito interrupciones del temporizador y deshabilito UART0 (Ya que tiene
prioridad)
    TCNT=CCLK>>1; // Periodo de 1/2 seg
    break;

case 'g': //Imprimir GPS Decodificado
    volcarOutputGPS=0;
    IRQEN=IRQEN_U1RX;
    getGPSFrame();

    break;

case 'h': //La salida del GPS (UART1) a la UART0
    volcarOutputGPS=1;
    IRQEN=IRQEN_U1RX;
    while(1){
        _delay_ms(1);
    } // Bloqueante
    break;

```

```

case 'K': //La salida de la UART2 a la UART0
    //IRQEN |= IRQEN_U2RX;
    while(1){
        _putch(UART2DAT); //No contiene nada
    }
    break;

case 'I': //Lee estado del TIMER
    _printf("\nTCNT: %d\n",TCNT);
    break;

case '1': //Pinta menú por UART0
    _puts(menutxt);
    break;

case '2': //Envía datos por UART0 vía interrupciones
    IRQEN=IRQEN_U0TX;
    while(1){
        UART0DAT='A';
    }
    break;

case '3': //Lectura del GPIN
    GpinRead();
    break;

case 'G': //Activar Sensor GAS
    IRQEN |= IRQEN_TIMER;
    ReadGAS();
    break;

case 'P': //Activar Sensor Polvo
    IRQEN |= IRQEN_TIMER;
    ReadDust();
    break;

case 'L': //Transmitir datos por LoRa
    transmitPRAOFrame();

    break;

case 'T': //Activar Sensor Polvo
    printCO();
    printDust();
    printCh4LPG();
    break;

default:
    _puts("No valid code selected");
    continue;
}
_puts("\n-----\n\n");
_delay_ms(1000);
}
}

```

Código de gps.c

```
// =====
// Proyecto Datalogger for IoT Curso 2022-2023
// Fecha: 26/01/2023
// Autor: Pablo Villacorta, Rubén Serrano, Óscar Martín y Andrés Martín
// Asignatura: Taller de Proyectos I
// File: gps.c Programa controlador del GPS
// =====

// El GPS se encuentra conectado a través de la UART1

//FIFO UART 1:
//uint8_t udat1[128]; //FIFO de recepcion para la UART1 (tamaño 1028 bits)

//volatile uint8_t rdixU1,wriU2; // Punteros de lectura y escritura (unsigned char -> 8 bit)

// -----
// -----      FUNCIONES      -----
// -----
```



```
char * strchr( const char str[], char ch )
{
    while ( *str && *str != ch ) ++str;

    return ( char * )( ch == *str ? str : "\0" );
}

int parse_comma_delimited_str(char *string, char **fields, int max_fields)
{
    int i = 0;
    fields[i++] = string;
    //while ((i < max_fields) && strchr(string,',') != NULL) {
    while ((i < max_fields) && "\0" != (string = strchr(string, ','))) {
        *string = '\0';
        fields[i++] = ++string;
    }
    return --i;
}

int strncmp(char *string1, char *buscar){
    uint8_t contador=0;
    uint8_t retorno=0;
    while(string1[contador+1]==buscar[contador]){
        retorno++;
        contador++;
        if(retorno == 6){
            return 0;
        }
    }
    return -1;
}

uint8_t getGPSFrame() //LEE DEL GPS un Frame Completo
{
    int i;
    char *field[20];
    uint8_t comprobador=0;
    volatile uint8_t pointer=0;
    _puts("\nWaiting for conexion ($GNGGA,$GPGGA,$GNRMC & $GPRMC) \n");

    while (comprobador <=0){
```

```

        while(GPS_FF == '0'){
            GPS_FRAME[pointer] = _getch1();
            pointer++;
        }
        GPS_FF = '0';
        if ((strcmp(GPS_FRAME, "$GNGGA") == 0) || (strcmp(GPS_FRAME, "$GPGGA") == 0)){ //comparamos cadena
            con las que queremos encontrar

                parse_comma_delimited_str(GPS_FRAME, field, 20);

                _puts("-----INFORMACION GPS-----\n");
                _puts("UTC Time :");_puts("hora: ");_putch(field[1][0]);_putch(field[1][1]);
                _puts(" minutos: ");_putch(field[1][2]);_putch(field[1][3]);
                _puts(" segundos: ");_putch(field[1][4]);_putch(field[1][5]);_putch('\n');
                _puts("Latitude :");_puts(field[2]);_putch('\n');
                _puts("Longitude :");_puts(field[4]);_putch('\n');
                _puts("Altitude :");_puts(field[9]);_putch('\n');
                _puts("Satellites:");_puts(field[7]);_putch('\n');

                UTC_time[0] = field[1][0];
                UTC_time[1] = field[1][1];
                UTC_time[2] = ':';
                UTC_time[3] = field[1][2];
                UTC_time[4] = field[1][3];
                UTC_time[5] = '.';
                UTC_time[6] = field[1][4];
                UTC_time[7] = field[1][5];
                UTC_time[8] = '\0';

                comprobador++;
            }

            if ((strcmp(GPS_FRAME, "$GNRMC") == 0) || (strcmp(GPS_FRAME, "$GPRMC") == 0)){ //comparamos cadena
                con las que queremos encontrar
                parse_comma_delimited_str(GPS_FRAME, field, 20);

                _puts("Date :");_puts("dia: ");_putch(field[9][0]);_putch(field[9][1]);
                _puts(" mes: ");_putch(field[9][2]);_putch(field[9][3]);
                _puts(" anio: ");_putch(field[9][4]);_putch(field[9][5]);_putch('\n');
                _putch('\n');_putch('\n');

                date[0]=field[9][0];
                date[1]=field[9][1];
                date[2]='.';
                date[3]=field[9][2];
                date[4]=field[9][3];
                date[5]='.';
                date[6]=field[9][4];
                date[7]=field[9][5];
                date[8]='\0';

                comprobador++;
            }

            for(i=0;i<pointer-1;i++){
                GPS_FRAME[i]=0;      //para limpiar la cadena
            }
        }

        return pointer;
    }

```

Código de spiSensors.c

```
// =====
```

```

// Proyecto Datalogger for IoT Curso 2022-2023
// Fecha: 26/01/2023
// Autor: Pablo Villacorta, Rubén Serrano, Óscar Martín y Andrés Martín
// Asignatura: Taller de Proyectos I
// File: spiSensors.c
//
// Rutinas para controlar el módulo SPI0. Este SPI se encarga de la comunicación
// con el BME680 y con el ADC (a su vez, al ADC entran las salidas de los sensores
// analógicos)
// =====

// ICE_SPI
#define BME680_CS    0b10 //SS0 (bit0 en baja)
#define ADC_CS       0b01 //SS1 (bit1 en baja)

///////////
// BME Registers //
///////////

// READ / WRITE REGS
#define status_BME      0x73 //Status [page]
#define reset_BME       0x60
#define Id_BME          0x50
#define Config_BME      0x75

#define Ctrl_meas_BME   0x74 // osrs_t [7:5] osrs_p [4:2] mode [1:0]
#define Ctrl_hum_BME    0x72 // osrs_h [2:0]

#define Ctrl_gas_1_BME  0x71
#define Ctrl_gas_0_BME  0x70

#define gas_wait0        0x6D
#define gas_wait1        0x6C
#define gas_wait2        0x6B
#define gas_wait3        0x6A
#define gas_wait4        0x69
#define gas_wait5        0x68
#define gas_wait6        0x67
#define gas_wait7        0x66
#define gas_wait8        0x65
#define gas_wait9        0x64

#define res_heat0         0x63
#define res_heat1         0x62
#define res_heat2         0x61
#define res_heat3         0x60
#define res_heat4         0x5F
#define res_heat5         0x5E
#define res_heat6         0x5D
#define res_heat7         0x5C
#define res_heat8         0x5B
#define res_heat9         0x5A

// READ ONLY REGS
#define gas_r_lsb_BME   0x2B
#define gas_r_msb_BME   0x2A

#define hum_lsb_BME     0x26
#define hum_msb_BME     0x25

#define temp_xlsb_BME   0x24
#define temp_lsb_BME    0x23
#define temp_msb_BME    0x22

#define press_xlsb_BME  0x21

```

```

#define press_lsb_BME    0x20
#define press_msb_BME    0x1F

#define eas_status_0_BME 0x1D

#define RD (0b10000000) // rw = 0 write, rw=1 read

///////////
// ADC COMM //
///////////

#define CMD_START (0b00000001)
#define CMD_CH0 (0b10000000) // CHANNEL 0: Módulo de gases (SEN0134)
#define CMD_CH1 (0b10010000) // CHANNEL 1: Sensor de Polvo (SHARP GPY1010AU0F)
#define CMD_CH2 (0b10100000) // CHANNEL 2: Unused
#define CMD_CH3 (0b10110000) // CHANNEL 3: Unused

// ---- TRANSFERENCIA SPI ----
// -----
uint8_t spixfer(uint8_t d)
{
    SPIDAT=d;
    while(SPISTA&1);
    return SPIDAT;
}
// ---- LECTURA BME680 ----
// -----
char readBME680(char dir){
    char dataread=0x00;
    SPISS=BME680_CS;
    spixfer(RD|dir);
    dataread = spixfer(0x00); //send dummy byte
    SPISS=0b11;
    return dataread;
}

// ---- ESCRITURA BME680 ----
// -----
void writeBME680(char data,char dir){
    SPISS=BME680_CS;
    spixfer(dir);
    spixfer(data);
    SPISS=0b11;
}

// ---- LECTURA ADC ----
// -----
uint32_t ReadADC(char cmd_ch) // Le meto el canal
{
    unsigned char B_MSB=0, B_LSB=0; //1 Byte
    uint32_t B=0; //2 Bytes = (B_MSB B_LSB)

    SPISS=ADC_CS; // Selecciono el ADC
    spixfer(CMD_START); // Enviar el bit de START (BYTE CMD_START)
    B_MSB = spixfer(cmd_ch); // Envio los bits de seleccion de canal
    //((BYTE CMD_CHi) [(diff)(D2)(D1)(D0) XXXX]
    //A la vez, recibo los primeros datos [???? ?0(B9)(B8)]
    B_LSB = spixfer(0X00); // Sent dummy Byte [0000 0000]
    //A la vez, recibo los segundos datos [(B7)(B6)(B5)(B4) (B3)(B2)(B1)(B0)]
    SPISS=0b11; //Finalizar comunicacion SPI. FIOSET = ADC_CS;
}

```

```

B = B_MSB<<8; // Desplazo el dato 8 bits a la izquierda (B_MSB) (0000 0000)
B|=B_LSB; // XOR con B_LSB -> (B_MSB) (0000 0000) XOR (0000 0000) (B_LSB)
// Agregar mascara de 10 bits (0x3ff) 0b0000001111111111
return (B&=0b0000001111111111);
}

void printAdcChannels() // In mV
{
// ADC Digital Output code = 1024 x Vin / Vref
// Vin = Digital Output Code x Vref / 1024 = DigOutCode >>10 x Vref (3300mV)

_puts("-- ADC CHANNELS: --\n");
_printf("Channel 0: %d mV\n", (ReadADC(CMD_CH0)*3300)>>10);
_printf("Channel 1: %d mV\n", (ReadADC(CMD_CH1)*3300)>>10);
_printf("Channel 2: %d mV\n", (ReadADC(CMD_CH2)*3300)>>10);
_printf("Channel 3: %d mV\n", (ReadADC(CMD_CH3)*3300)>>10);
}

/////////////////////////////////////////////////////////////////
//          BME680          ///////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////

/* ----- LECTURA DE TODOS LOS REGISTROS DEL BME -----
2 páginas de registros, se selecciona una página u otra con el registro spi_mem_page:
    spi_mem_page = 0: página 0 --> 128 registros (0x80 to 0xFF)
    spi_mem_page = 1: página 1 --> 128 registros (0x00 to 0x7F)
128*2 = 256 registros
*/
char bmeRegs[16][16];

void readAllBMERegs()
{
int i,j;

// Página 0
writeBME680(0b00000000,status_BME);
for(i=0;i<8;i++){
    for(j=0;j<16;j++){
        bmeRegs[i+8][j] = readBME680(0x80 + 16*i + j);
    }
}

// Página 1
writeBME680(0b00010000,status_BME);
for(i=0;i<8;i++){
    for(j=0;j<16;j++){
        bmeRegs[i][j] = readBME680(0x00 + 16*i + j);
    }
}

// -----
void printBMERegs(){
int i,j;

_puts("Registros del sensor BME:\n");

for(i=8;i<16;i++){
    for(j=0;j<16;j++){
        _printf("%02x ",bmeRegs[i][j]);
    }
}
}

```

```

        }
        _puts("\n");
    }

    _puts("\n");

    for(i=0;i<8;i++){
        for(j=0;j<16;j++){
            _printf("%02x ",bmeRegs[i][j]);
        }
        _puts("\n");
    }
}

// -----
char bmeReg(char dir){
    int row, col;

    row = dir/16;
    col = dir%16;

    return bmeRegs[row][col];
}

// ----- BME init -----
void startBME680(){
    // Página 1
    writeBME680(0b00010000,status_BME);

    // Set humidity oversampling to 1x (osrs_h = Ctrl_hum_BME[2:0] = 001)
    writeBME680(0b00000001,Ctrl_hum_BME);
    _delay_ms(1);

    // Set temperature oversampling to 2x (osrs_t = Ctrl_meas_BME[7:5] = 010)
    // Set pressure oversampling to 16x (osrs_p = Ctrl_meas_BME[4:2] = 101)
    writeBME680(0b01010100,Ctrl_meas_BME);
    _delay_ms(1);

    // Set nb_conv to 0x0 to select the previously defined heater settings (nb_conv = Ctrl_gas_1_BME[3:0])
    // Set run_gas_l to 1 to enable gas measurements (run_gas_l = Ctrl_gas_1_BME[4])
    writeBME680(0b00010000,Ctrl_gas_1_BME);
    _delay_ms(1);

    // Set gas_wait_0 to 0x59 to select 100 ms heat up duration
    writeBME680(0x59,gas_wait0);
    _delay_ms(1);

    // Set the corresponding heater set-point by writing the target heater resistance to res_heat_0
    writeBME680(0,res_heat0);
    _delay_ms(1);

    // Set mode to 0b01 to trigger a single measurement. (mode = Ctrl_meas_BME[1:0])
    writeBME680(0b01010101,Ctrl_meas_BME);
    _delay_ms(1);
}

void measureBME680(){
    // Read all BME registers
    readAllBMERegs();
}

```

```

/*----- Temperature measurement -----*/
LSB / MSB
par_t1 0xE9 / 0xEA
par_t2 0x8A / 0x8B
par_t3 0x8C
temp_adc 0x24<7:4>/ 0x23 / 0x22
*/
int par_t1, par_t2, par_t3, temp_adc;
int var1, var2, var3, t_fine, temp_comp;

par_t1 = (int32_t)((bmeReg(0xEA) << 8) | bmeReg(0xE9));
par_t2 = (int32_t)((bmeReg(0x8B) << 8) | bmeReg(0x8A));
par_t3 = (int32_t)bmeReg(0x8C);
temp_adc = (int32_t)((bmeReg(0x22) << 12) | (bmeReg(0x23) << 4) | (bmeReg(0x24) >> 4));

var1 = ((int32_t)temp_adc >> 3) - ((int32_t)par_t1 << 1);
var2 = (var1 * (int32_t)par_t2) >> 11;
var3 = (((var1 >> 1) * (var1 >> 1)) >> 12) * ((int32_t)par_t3 << 4)) >> 14;
t_fine = var2 + var3;
temp_comp = ((t_fine * 5) + 128) >> 8;

/* ----- Pressure measurement -----*/
LSB / MSB
par_p1 0x8E / 0x8F
par_p2 0x90 / 0x91
par_p3 0x92
par_p4 0x94 / 0x95
par_p5 0x96 / 0x97
par_p6 0x99
par_p7 0x98
par_p8 0x9C / 0x9D
par_p9 0x9E / 0x9F
par_p10 0xA0
press_adc 0x21<7:4>/ 0x20 / 0x1F
*/
int par_p1, par_p2, par_p3, par_p4, par_p5, par_p6, par_p7, par_p8, par_p9, par_p10, press_adc;
int press_comp;

par_p1 = (int32_t)((bmeReg(0x8F) << 8) | bmeReg(0x8E));
par_p2 = (int32_t)((bmeReg(0x91) << 8) | bmeReg(0x90));
par_p3 = (int32_t)bmeReg(0x92);
par_p4 = (int32_t)((bmeReg(0x95) << 8) | bmeReg(0x94));
par_p5 = (int32_t)((bmeReg(0x97) << 8) | bmeReg(0x96));
par_p6 = (int32_t)bmeReg(0x99);
par_p7 = (int32_t)bmeReg(0x98);
par_p8 = (int32_t)((bmeReg(0x9D) << 8) | bmeReg(0x9C));
par_p9 = (int32_t)((bmeReg(0x9F) << 8) | bmeReg(0x9E));
par_p10 = (int32_t)bmeReg(0xA0);
press_adc = (int32_t)((bmeReg(0x1F) << 12) | (bmeReg(0x20) << 4) | (bmeReg(0x21) >> 4));

var1 = ((int32_t)t_fine >> 1) - 64000;
var2 = (((var1 >> 2) * (var1 >> 2)) >> 11) * (int32_t)par_p6) >> 2;
var2 = var2 + ((var1 * (int32_t)par_p5) << 1);
var2 = (var2 >> 2) + ((int32_t)par_p4 << 16);
var1 = (((((var1 >> 2) * (var1 >> 2)) >> 13) * ((int32_t)par_p3 << 5)) >> 3) + (((int32_t)par_p2 * var1) >> 1);
var1 = var1 >> 18;
var1 = ((32768 + var1) * (int32_t)par_p1) >> 15;
press_comp = 1048576 - press_adc;
press_comp = (uint32_t)((press_comp - (var2 >> 12)) * ((uint32_t)3125));
if (press_comp >= (1 << 30)) press_comp = ((press_comp / (uint32_t)var1) << 1);

```

```

else press_comp = ((press_comp << 1) / (uint32_t)var1);
var1 = ((int32_t)par_p9 * (int32_t)((press_comp >> 3) * (press_comp >> 3)) >> 13) >> 12;
var2 = ((int32_t)(press_comp >> 2) * (int32_t)par_p8) >> 13;
var3 = ((int32_t)(press_comp >> 8) * (int32_t)(press_comp >> 8) * (int32_t)(press_comp >> 8) * (int32_t)par_p10) >>
17;
press_comp = (int32_t)(press_comp) + ((var1 + var2 + var3 + ((int32_t)par_p7 << 7)) >> 4);

/* ----- Humidity measurement ----- */

LSB / MSB
par_h1 <3:0>0xE2 / 0xE3
par_h2 <7:4>0xE2 / 0xE1
par_h3 0xE4
par_h4 0xE5
par_h5 0xE6
par_h6 0xE7
par_h7 0xE8
hum_adc 0x26 / 0x25
*/
int par_h1, par_h2, par_h3, par_h4, par_h5, par_h6, par_h7, hum_adc;
int temp_scaled, hum_comp, var4, var5, var6;

par_h1 = (int32_t)((bmeReg(0xE3) << 4) | (bmeReg(0xE2) & 0b00001111));
par_h2 = (int32_t)((bmeReg(0xE1) << 4) | (bmeReg(0xE2) >> 4));
par_h3 = (int32_t)bmeReg(0xE4);
par_h4 = (int32_t)bmeReg(0xE5);
par_h5 = (int32_t)bmeReg(0xE6);
par_h3 = (int32_t)bmeReg(0xE7);
par_h3 = (int32_t)bmeReg(0xE8);
hum_adc = (int32_t)((bmeReg(0x25) << 8) | bmeReg(0x26));

temp_scaled = (int32_t)temp_comp;
var1 = (int32_t)hum_adc - (int32_t)((int32_t)par_h1 << 4) - (((temp_scaled * (int32_t)par_h3) / ((int32_t)100)) >> 1);
var2 = ((int32_t)par_h2 * (((temp_scaled * (int32_t)par_h4) / ((int32_t)100)) + (((temp_scaled * ((temp_scaled *
(int32_t)par_h5) / ((int32_t)100))) >> 6) / ((int32_t)100)) + ((int32_t)(1 << 14))) >> 10;
var3 = var1 * var2;
var4 = (((int32_t)par_h6 << 7) + ((temp_scaled * (int32_t)par_h7) / ((int32_t)100))) >> 4;
var5 = ((var3 >> 14) * (var3 >> 14)) >> 10;
var6 = (var4 * var5) >> 1;
hum_comp = (((var3 + var6) >> 10) * ((int32_t) 1000)) >> 12;

// ----

temp = temp_comp/100;
presion = press_comp/100;
humedad = hum_comp/1000;

_printf("Temperatura: %d°C\n", temp, 167);
_printf("temp_adc: %d\n", temp_adc);
_printf("Presion: %d hPascal\n", presion);
_printf("Humedad: %d%\n", humedad, 37);
}

/////////////////////////////// ANALOG SENSORS ///////////////////
void ReadDust(){ // Sensor de particulas

GPOUT = (STEPUP_CE|ice_led4); //Activa El step Up y el led 4 que indica lectura Polvo
clkMode=3; //Modo Sensor de polvo

```

```

TCNT =(0.00968)*CCLK; //Configuramos el reloj cada 10 mseg - 0,32ms = 9,68ms = 0.00968 seg

}

void ReadGAS(){ // Sensor de GAS SEN0134

    _printf("-- ENABLE GAS SENSOR --\n"
    "Config: Step Up Enabled\n"
    "60 seg -> 5V (while LEFT LED)\n"
    "90 seg -> 1V4 (while RIGHT LED)"
    "\n");
    // LED IZQUIERDA Indica 5V
    // LED DERECHA Indica 1V4
    clkMode=1;
}

void printCO(){ //
    _printf("\n COValue (mV): %d\n",COValue);
}

void printDust(){ //
    _printf("\n polvoValue (mV): %d\n",polvoValue);

    // Seria necesario un Acondicionamiento: Factor 25/28 = 0.9
    // Funcion de transferencia del sensor: V= 5 * P + Offset
    // Offset tipico 0,9, (min 0 max 1.5) -> Spg Offset = 0.4 mg/m^3, Sens = 5000mv / (mg/m^3)
    // Polvo = ( ((Dig * 3300mv / 1024) / 0.9 ) - Offset=0.4 ) / 5
}

void printCh4LPG(){ //
    _printf("\n Ch4LPGValue (mV): %d\n",Ch4LPGValue);
}

```

Código de spiLora.c

```
// =====
// Proyecto Datalogger for IoT Curso 2022-2023
// Fecha: 26/01/2023
// Autor: Pablo Villacorta, Rubén Serrano, Óscar Martín y Andrés Martín
// Asignatura: Taller de Proyectos I
// File: spiLoRa.c Rutinas para controlar el Sensor LoRA
// =====

///////////
// LoRA Registers //
///////////

// This is the maximum number of interrupts the driver can support
// Most Arduinos can handle 2, Megas can handle more
#define RH_RF95_NUM_INTERRUPTS 3

// Max number of octets the LORA Rx/Tx FIFO can hold
#define RH_RF95_FIFO_SIZE 255

// This is the maximum number of bytes that can be carried by the LORA.
// We use some for headers, keeping fewer for RadioHead messages
#define RH_RF95_MAX_PAYLOAD_LEN RH_RF95_FIFO_SIZE

// The length of the headers we add.
// The headers are inside the LORA's payload
#define RH_RF95_HEADER_LEN 4

// This is the maximum message length that can be supported by this driver.
// Can be pre-defined to a smaller size (to save SRAM) prior to including this header
// Here we allow for 1 byte message length, 4 bytes headers, user data and 2 bytes of FCS
#ifndef RH_RF95_MAX_MESSAGE_LEN
#define RH_RF95_MAX_MESSAGE_LEN (RH_RF95_MAX_PAYLOAD_LEN - RH_RF95_HEADER_LEN)
#endif

// The crystal oscillator frequency of the module
#define RH_RF95_FXOSC 32000000.0

// The Frequency Synthesizer step = RH_RF95_FXOSC / 2^^19
#define RH_RF95_FSTEP (RH_RF95_FXOSC / 524288)

#define RH_RF95_REG_00_FIFO          0x00
#define RH_RF95_REG_01_OP_MODE       0x01
#define RH_RF95_REG_02_RESERVED      0x02
#define RH_RF95_REG_03_RESERVED      0x03
#define RH_RF95_REG_04_RESERVED      0x04
#define RH_RF95_REG_05_RESERVED      0x05
#define RH_RF95_REG_06_FRF_MSB       0x06
#define RH_RF95_REG_07_FRF_MID       0x07
#define RH_RF95_REG_08_FRF_LSB       0x08
#define RH_RF95_REG_09_PA_CONFIG     0x09
#define RH_RF95_REG_0A_PA_RAMP       0x0a
#define RH_RF95_REG_0B_OCP           0x0b
#define RH_RF95_REG_0C_LNA           0x0c
#define RH_RF95_REG_0D_FIFO_ADDR_PTR 0x0d
#define RH_RF95_REG_0E_FIFO_TX_BASE_ADDR 0x0e
#define RH_RF95_REG_0F_FIFO_RX_BASE_ADDR 0x0f
#define RH_RF95_REG_10_FIFO_RX_CURRENT_ADDR 0x10
#define RH_RF95_REG_11_IRQ_FLAGS_MASK 0x11
#define RH_RF95_REG_12_IRQ_FLAGS      0x12
#define RH_RF95_REG_13_RX_NB_BYTES    0x13
#define RH_RF95_REG_14_RX_HEADER_CNT_VALUE_MSB 0x14
#define RH_RF95_REG_15_RX_HEADER_CNT_VALUE_LSB 0x15
```

```

#define RH_RF95_REG_16_RX_PACKET_CNT_VALUE_MSB      0x16
#define RH_RF95_REG_17_RX_PACKET_CNT_VALUE_LSB      0x17
#define RH_RF95_REG_18_MODEM_STAT                  0x18
#define RH_RF95_REG_19_PKT_SNR_VALUE              0x19
#define RH_RF95_REG_1A_PKT_RSSI_VALUE             0x1a
#define RH_RF95_REG_1B_RSSI_VALUE                 0x1b
#define RH_RF95_REG_1C_HOP_CHANNEL                0x1c
#define RH_RF95_REG_1D_MODEM_CONFIG1              0x1d
#define RH_RF95_REG_1E_MODEM_CONFIG2              0x1e
#define RH_RF95_REG_1F_SYMB_TIMEOUT_LSB           0x1f
#define RH_RF95_REG_20_PREAMBLE_MSB               0x20
#define RH_RF95_REG_21_PREAMBLE_LSB               0x21
#define RH_RF95_REG_22_PAYLOAD_LENGTH             0x22
#define RH_RF95_REG_23_MAX_PAYLOAD_LENGTH         0x23
#define RH_RF95_REG_24_HOP_PERIOD                 0x24
#define RH_RF95_REG_25_FIFO_RX_BYTE_ADDR          0x25
#define RH_RF95_REG_26_MODEM_CONFIG3              0x26

#define RH_RF95_REG_27_PPM_CORRECTION            0x27
#define RH_RF95_REG_28_FEI_MSB                   0x28
#define RH_RF95_REG_29_FEI_MID                  0x29
#define RH_RF95_REG_2A_FEI_LSB                   0x2a
#define RH_RF95_REG_2C_RSSI_WIDEBAND            0x2c
#define RH_RF95_REG_31_DETECT_OPTIMIZE          0x31
#define RH_RF95_REG_33_INVERT_IQ                 0x33
#define RH_RF95_REG_37_DETECTION_THRESHOLD       0x37
#define RH_RF95_REG_39_SYNC_WORD                 0x39

#define RH_RF95_REG_40_DIO_MAPPING1              0x40
#define RH_RF95_REG_41_DIO_MAPPING2              0x41
#define RH_RF95_REG_42_VERSION                  0x42

#define RH_RF95_REG_4B_TCXO                    0x4b
#define RH_RF95_REG_4D_PA_DAC                  0x4d
#define RH_RF95_REG_5B_FORMER_TEMP             0x5b
#define RH_RF95_REG_61_AGC_REF                 0x61
#define RH_RF95_REG_62_AGC_THRESH1              0x62
#define RH_RF95_REG_63_AGC_THRESH2              0x63
#define RH_RF95_REG_64_AGC_THRESH3              0x64

// RH_RF95_REG_01_OP_MODE                  0x01
#define RH_RF95_LONG_RANGE_MODE                0x80
#define RH_RF95_ACCESS_SHARED_REG              0x40
#define RH_RF95_LOW_FREQUENCY_MODE            0x08
#define RH_RF95_MODE                          0x07
#define RH_RF95_MODE_SLEEP                   0x00
#define RH_RF95_MODE_STDBY                  0x01
#define RH_RF95_MODE_FSTX                   0x02
#define RH_RF95_MODE_TX                      0x03
#define RH_RF95_MODE_FSRX                   0x04
#define RH_RF95_MODE_RXCONTINUOUS            0x05
#define RH_RF95_MODE_RXSINGLE                0x06
#define RH_RF95_MODE_CAD                     0x07

// RH_RF95_REG_09_PA_CONFIG                0x09
#define RH_RF95_PA_SELECT                   0x80
#define RH_RF95_MAX_POWER                  0x70
#define RH_RF95_OUTPUT_POWER                0x0f

// RH_RF95_REG_0A_PA_RAMP                0x0a
#define RH_RF95_LOW_PN_TX_PLL_OFF           0x10
#define RH_RF95_PA_RAMP                     0x0f
#define RH_RF95_PA_RAMP_3_4MS               0x00
#define RH_RF95_PA_RAMP_2MS                 0x01
#define RH_RF95_PA_RAMP_1MS                 0x02

```

```

#define RH_RF95_PA_RAMP_500US      0x03
#define RH_RF95_PA_RAMP_250US      0x04
#define RH_RF95_PA_RAMP_125US      0x05
#define RH_RF95_PA_RAMP_100US      0x06
#define RH_RF95_PA_RAMP_62US       0x07
#define RH_RF95_PA_RAMP_50US       0x08
#define RH_RF95_PA_RAMP_40US       0x09
#define RH_RF95_PA_RAMP_31US       0x0a
#define RH_RF95_PA_RAMP_25US       0x0b
#define RH_RF95_PA_RAMP_20US       0x0c
#define RH_RF95_PA_RAMP_15US       0x0d
#define RH_RF95_PA_RAMP_12US       0x0e
#define RH_RF95_PA_RAMP_10US       0x0f

// RH_RF95_REG_0B_OCP           0x0b
#define RH_RF95_OCP_ON            0x20
#define RH_RF95_OCP_TRIM          0x1f

// RH_RF95_REG_0C_LNA           0x0c
#define RH_RF95_LNA_GAIN          0xe0
#define RH_RF95_LNA_GAIN_G1        0x20
#define RH_RF95_LNA_GAIN_G2        0x40
#define RH_RF95_LNA_GAIN_G3        0x60
#define RH_RF95_LNA_GAIN_G4        0x80
#define RH_RF95_LNA_GAIN_G5        0xa0
#define RH_RF95_LNA_GAIN_G6        0xc0
#define RH_RF95_LNA_BOOST_LF       0x18
#define RH_RF95_LNA_BOOST_LF_DEFAULT 0x00
#define RH_RF95_LNA_BOOST_HF       0x03
#define RH_RF95_LNA_BOOST_HF_DEFAULT 0x00
#define RH_RF95_LNA_BOOST_HF_150PC 0x03

// RH_RF95_REG_11_IRQ_FLAGS_MASK 0x11
#define RH_RF95_RX_TIMEOUT_MASK    0x80
#define RH_RF95_RX_DONE_MASK       0x40
#define RH_RF95_PAYLOAD_CRC_ERROR_MASK 0x20
#define RH_RF95_VALID_HEADER_MASK   0x10
#define RH_RF95_TX_DONE_MASK       0x08
#define RH_RF95_CAD_DONE_MASK      0x04
#define RH_RF95_FHSS_CHANGE_CHANNEL_MASK 0x02
#define RH_RF95_CAD_DETECTED_MASK  0x01

// RH_RF95_REG_12_IRQ_FLAGS      0x12
#define RH_RF95_RX_TIMEOUT          0x80
#define RH_RF95_RX_DONE             0x40
#define RH_RF95_PAYLOAD_CRC_ERROR   0x20
#define RH_RF95_VALID_HEADER         0x10
#define RH_RF95_TX_DONE              0x08
#define RH_RF95_CAD_DONE             0x04
#define RH_RF95_FHSS_CHANGE_CHANNEL 0x02
#define RH_RF95_CAD_DETECTED         0x01

// RH_RF95_REG_18_MODEM_STAT     0x18
#define RH_RF95_RX_CODING_RATE      0xe0
#define RH_RF95_MODEM_STATUS_CLEAR   0x10
#define RH_RF95_MODEM_STATUS_HEADER_INFO_VALID 0x08
#define RH_RF95_MODEM_STATUS_RX_ONGOING 0x04
#define RH_RF95_MODEM_STATUS_SIGNAL_SYNCHRONIZED 0x02
#define RH_RF95_MODEM_STATUS_SIGNAL_DETECTED 0x01

// RH_RF95_REG_1C_HOP_CHANNEL    0x1c
#define RH_RF95_PLL_TIMEOUT          0x80
#define RH_RF95_RX_PAYLOAD_CRC_IS_ON 0x40
#define RH_RF95_FHSS_PRESENT_CHANNEL 0x3f

```

```

// RH_RF95_REG_1D_MODEM_CONFIG1          0x1d
#define RH_RF95_BW           0xf0

#define RH_RF95_BW_7_8KHZ      0x00
#define RH_RF95_BW_10_4KHZ     0x10
#define RH_RF95_BW_15_6KHZ     0x20
#define RH_RF95_BW_20_8KHZ     0x30
#define RH_RF95_BW_31_25KHZ    0x40
#define RH_RF95_BW_41_7KHZ     0x50
#define RH_RF95_BW_62_5KHZ     0x60
#define RH_RF95_BW_125KHZ      0x70
#define RH_RF95_BW_250KHZ      0x80
#define RH_RF95_BW_500KHZ      0x90
#define RH_RF95_CODING_RATE    0x0e
#define RH_RF95_CODING_RATE_4_5 0x02
#define RH_RF95_CODING_RATE_4_6 0x04
#define RH_RF95_CODING_RATE_4_7 0x06
#define RH_RF95_CODING_RATE_4_8 0x08
#define RH_RF95_IMPLICIT_HEADER_MODE_ON 0x01

// RH_RF95_REG_1E_MODEM_CONFIG2          0x1e
#define RH_RF95_SPREADING_FACTOR   0xf0
#define RH_RF95_SPREADING_FACTOR_64CPS 0x60
#define RH_RF95_SPREADING_FACTOR_128CPS 0x70
#define RH_RF95_SPREADING_FACTOR_256CPS 0x80
#define RH_RF95_SPREADING_FACTOR_512CPS 0x90
#define RH_RF95_SPREADING_FACTOR_1024CPS 0xa0
#define RH_RF95_SPREADING_FACTOR_2048CPS 0xb0
#define RH_RF95_SPREADING_FACTOR_4096CPS 0xc0
#define RH_RF95_TX_CONTINUOUS_MODE     0x08

#define RH_RF95_PAYLOAD_CRC_ON       0x04
#define RH_RF95_SYM_TIMEOUT_MSB     0x03

// RH_RF95_REG_26_MODEM_CONFIG3          0x08 // HopeRF term
#define RH_RF95_MOBILE_NODE         0x08 // Semtechs term
#define RH_RF95_LOW_DATA_RATE_OPTIMIZE 0x04
#define RH_RF95_AGC_AUTO_ON         0x04

// RH_RF95_REG_4B_TCXO              0x4b
#define RH_RF95_TCXO_TCXO_INPUT_ON   0x10

// RH_RF95_REG_4D_PA_DAC            0x4d
#define RH_RF95_PA_DAC_DISABLE       0x04
#define RH_RF95_PA_DAC_ENABLE        0x07

///////////
///////////

#define RD2 (0b10000000) // rw = 1 write, rw=0 read

///////////
// AÑADIDO PRAO //
///////////

#define RHModeSleep     0x00
#define RHModeIdle     0x01
#define RHModeTx       0x02
#define RHModeRx       0x03

uint8_t _mode;

// -----
uint8_t spiLoRAxfer (uint8_t d)

```

```

{
    SPILDAT=d;
    while(SPILSTA&1);
    return SPILDAT;
}

// -----



char readLoRA(char dir){
    char dataread=0x00;
    SPILSS=0;
    spiLoRAxfer (dir);
    dataread = spiLoRAxfer(0x00);
    SPILSS=1;
    return dataread; //send dummy byte
}

// -----



void writeLoRA(char data,char dir){
    SPILSS=0;
    spiLoRAxfer (RD2|dir);
    spiLoRAxfer(data);
    SPILSS=1;
}

void printLoRaRegs()
{
    int i,j;

    _puts("Registros del modulo LoRa:\n");

    for(i=0;i<8;i++){
        for(j=0;j<16;j++){
            _printf("%02x ",readLoRA(0x00 + 16*i + j));
        }
        _puts("\n");
    }
}

void setModemRegisters(){
    writeLoRA(0x72, RH_RF95_REG_1D_MODEM_CONFIG1);
    writeLoRA(0x74, RH_RF95_REG_1E_MODEM_CONFIG2);
    writeLoRA(0x04, RH_RF95_REG_26_MODEM_CONFIG3);
}

// Los anteriores registros son de onfiguración
// El primero es de BW
// El segundo es de Cr (para lo del codigo de lora)
// El ultimo es de Sf (para lo del codigo de lora)
// 1d, 1e, 26
// {0x72, 0xb4, 0x04}, // Bw125Cr45Sf2048, AGC enabled
// {0x72, 0x74, 0x04}, // Bw125Cr45Sf128 (the chip default), AGC enabled
// {0x92, 0x74, 0x04}, // Bw500Cr45Sf128, AGC enabled
// {0x48, 0x94, 0x04}, // Bw31_25Cr48Sf512, AGC enabled
// {0x78, 0xc4, 0x0c}, // Bw125Cr48Sf4096, AGC enabled

void setPreambleLength(uint16_t bytes){
    writeLoRA(bytes >> 8, RH_RF95_REG_20_PREAMBLE_MSB);
    writeLoRA(bytes & 0xff, RH_RF95_REG_21_PREAMBLE_LSB);
}

```

```

void setFrequency868(){
    writeLoRA(0xD9, RH_RF95_REG_06_FRF_MSB);
    writeLoRA(0x00, RH_RF95_REG_07_FRF_MID);
    writeLoRA(0x00, RH_RF95_REG_08_FRF_LSB);
}

#define _txHeaderTo 0xff
#define _txHeaderFrom 0xff
#define _txHeaderId 0x00
#define _txHeaderFlags 0x00

void loraSend(char *mensaje){
    uint8_t size = _sizeof(mensaje);

    // STDBY Mode:
    writeLoRA(RH_RF95_MODE_STDBY | RH_RF95_LONG_RANGE_MODE, RH_RF95_REG_01_OP_MODE);

    writeLoRA(0, RH_RF95_REG_0D_FIFO_ADDR_PTR); // Puntero a la direccion inicial en la fifo

    writeLoRA(size+4,RH_RF95_REG_22_PAYLOAD_LENGTH); // Longitud de los datos

    // Write data FIFO -----
    // Cabeceras
    writeLoRA(_txHeaderTo, RH_RF95_REG_00_FIFO);
    writeLoRA(_txHeaderFrom, RH_RF95_REG_00_FIFO);
    writeLoRA(_txHeaderId, RH_RF95_REG_00_FIFO);
    writeLoRA(_txHeaderFlags, RH_RF95_REG_00_FIFO);

    // _puts("Reg IRQ antes de tx:\n");
    // _printfBin(readLoRA(RH_RF95_REG_12_IRQ_FLAGS));

    _puts("Transmitiendo");
    //el registro reg_fifo contiene el contenido de la fifo al que apunta la direccion fifo_addr_ptr
    //cada vez que escribimos/leemos en reg_fifo se incrementa el registro fifo_addr_ptr en 1
    for(uint8_t i = 0; i<size; i++){
        writeLoRA(*mensaje++,RH_RF95_REG_00_FIFO);
    }

    writeLoRA(RH_RF95_MODE_TX | RH_RF95_LONG_RANGE_MODE, RH_RF95_REG_01_OP_MODE); // Modo transmisión
    _delay_ms(1);

    while(1){
        if(readLoRA(RH_RF95_REG_01_OP_MODE) == (RH_RF95_MODE_TX | RH_RF95_LONG_RANGE_MODE)){
            _puts(".");
            _delay_ms(1);
        }
        //Tras la transmisión, cambia automáticamente al modo STAND-BY:
        if(readLoRA(RH_RF95_REG_01_OP_MODE) == (RH_RF95_MODE_STDBY |
RH_RF95_LONG_RANGE_MODE)){
            _puts("\nTransmision completada\n");
            writeLoRA(0xff,RH_RF95_REG_12_IRQ_FLAGS); // Bajamos el flag TxDone poniendo todo el registro irq_flags
a 0xff
            break;
        }
    }

    // Vuelta del puntero a la dirección 0:
    writeLoRA(0, RH_RF95_REG_0D_FIFO_ADDR_PTR); // Puntero a la direccion inicial en la fifo
}

uint8_t loraInit(){
    // Set sleep mode, so we can also set LORA mode:
    writeLoRA(RH_RF95_MODE_SLEEP | RH_RF95_LONG_RANGE_MODE, RH_RF95_REG_01_OP_MODE);
}

```

```

// Check if we are in sleep mode:
if (readLoRA(RH_RF95_REG_01_OP_MODE) != (RH_RF95_MODE_SLEEP | RH_RF95_LONG_RANGE_MODE)){
    _puts("No se detecta LoRa\n");
    return 0; // No device present?
}

// Set up FIFO
// We configure so that we can use the entire 256 byte FIFO for either receive
// or transmit, but not both at the same time
writeLoRA(0, RH_RF95_REG_0E_FIFO_TX_BASE_ADDR);
writeLoRA(0, RH_RF95_REG_0F_FIFO_RX_BASE_ADDR);

// STDBY Mode:
writeLoRA(RH_RF95_MODE_STDBY | RH_RF95_LONG_RANGE_MODE, RH_RF95_REG_01_OP_MODE);

setModemRegisters();
setPreambleLength(8);
setFrequency868();

// Set TX Power -----
writeLoRA(0xAD, RH_RF95_REG_09_PA_CONFIG);
//-----
}

void transmitPRAOFrame(){
    char *strtemp;
    char *strhum;
    char *strpresion;
    char *strpolvo;
    char *bufferTX;
    char *strCO;
    char *strCh4LPG;
    uint8_t indx= 0;

loraInit(); //Incializa el módulo LoRa

// 1a transmision:
loraSend("\n$$ INIT PRAO TX $$"); //Transmite

// 2a transmision: (UTC TIME)
bufferTX="";
	memcpy(_memcpy(bufferTX, "\n$ UTC Time: ", _sizeof("\n$ UTC Time: ")) - 1, UTC_time, _sizeof(UTC_time));
loraSend(bufferTX); //Transmite
_puts(bufferTX);
_puts("\n");

// 3A transmision: (DATE)
bufferTX="";
	memcpy(_memcpy(bufferTX, "\n$ Date: ", _sizeof("\n$ Date: ")) - 1, date, _sizeof(date));
loraSend(bufferTX); //Transmite
_puts(bufferTX);
_puts("\n");

// 3a transmision: (Temperatura)
bufferTX="";
	my_itoa(temp,strtemp);
	memcpy(_memcpy(_memcpy(bufferTX, "\n$ Temp: ", _sizeof("\n$ Temp: ")) - 1, strtemp, _sizeof(strtemp))- 1, "deg",
_sizeof("deg"));
loraSend(bufferTX); //Transmite
_puts(bufferTX);
_puts("\n");
}

```

```

// 4a transmision: (Humedad)
bufferTX=""; // Limpio el buffer
my_itoa(humedad,strhum);
Memcpy(Memcpy(Memcpy(bufferTX, "\n$ Humedad: ",_sizeof("\n$ Humedad: "))- 1, strhum, _sizeof(strhum))- 1,
"%", _sizeof("%"));
_puts(bufferTX);
_puts("\n");
loraSend(bufferTX); //Transmite

// 5a transmision: (Presión)
bufferTX=""; // Limpio el buffer
my_itoa(presion,strpresion);
Memcpy(Memcpy(Memcpy(bufferTX, "\n$ Presion: ",_sizeof("\n$ Presion: "))- 1, strpresion, _sizeof(strpresion))- 1,
" hPas", _sizeof(" hPas"));
_puts(bufferTX);
_puts("\n");
loraSend(bufferTX); //Transmite

// 6a transmision: (Polvo)
bufferTX=""; // Limpio el buffer
my_itoa(polvoValue,strpolvo);
Memcpy(Memcpy(Memcpy(bufferTX, "\n$ Polvo: ",_sizeof("\n$ Polvo: "))- 1, strpolvo, _sizeof(strpolvo))- 1, "mV
ADC", _sizeof("mV ADC"));
_puts(bufferTX);
_puts("\n");
loraSend(bufferTX); //Transmite

// 7a transmision: (CO)
bufferTX=""; // Limpio el buffer
my_itoa(COValue,strCO);
Memcpy(Memcpy(Memcpy(bufferTX, "\n$ CO: ",_sizeof("\n$ CO: "))- 1, strCO, _sizeof(strCO))- 1, "mV ADC",
(sizeof("mV ADC")));
_puts(bufferTX);
_puts("\n");
loraSend(bufferTX); //Transmite

// 8a transmision: (CH4)
bufferTX=""; // Limpio el buffer
my_itoa(Ch4LPGValue,strCh4LPG);
Memcpy(Memcpy(Memcpy(bufferTX, "\n$ Ch4LPG: ",_sizeof("\n$ Ch4LPG: "))- 1, strCh4LPG,
(sizeof(strCh4LPG))- 1, "mV ADC", _sizeof("mV ADC")));
_puts(bufferTX);
_puts("\n");
loraSend(bufferTX); //Transmite

//Ultima transmision:
loraSend("\n$$ END PRAO TX $$"); //Transmite
}

```

Código de gpin.c

```

// =====
// Proyecto Datalogger for IoT Curso 2022-2023
// Fecha: 26/01/2023
// Autor: Pablo Villacorta, Rubén Serrano, Óscar Martín y Andrés Martín
// Asignatura: Taller de Proyectos I
// File: gpin.c
// =====

// GPIN
#define GPIN0 (0b00000001) //P73 GPIN0 -> J1 [39]
#define GPIN1 (0b00000010) //P78 GPIN1 -> J1 [38]
#define GPIN2 (0b00000100) //P81 GPIN2 -> J1 [37]
#define GPIN3 (0b00001000) //P85 GPIN3 -> J1 [36]
#define GPIN4 (0b00010000) //P88 GPIN4 -> J1 [35]
#define GPIN5 (0b00100000) //P93 GPIN5 -> J1 [34]
#define GPIN6 (0b01000000) //P95 GPIN6 -> J1 [33]
#define GPIN7 (0b10000000) //P99 GPIN7 -> J1 [32]

const static char *gpintxt="\n"
"\n----- PINES GPIN ----- \n"
"\n          J1\n"
"\n----- \n"
"\n|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|\n"
"\n----- \n"
"\n----- \n"
"\n|x|x|x|x|x|x|x|x|x|x|x|x|GP7|GP6|GP5|GP4|GP3|GP2|GP1|GP0|x|\n"
"\n----- \n"
"\n"
"\n          J2\n"
"\n----- \n"
"\n|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|\n"
"\n----- \n"
"\n----- \n"
"\n|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|x|\n"
"\n----- \n"
"\n";
"\n----- \n"

// ---- Lectura GPIN ----
void GpinRead(void)
{
    _puts(gpintxt);
    _puts("GPIN:");
    _printfBin(GPIN);
    _puts("\n");
}

```

Anexo B: GeneraEntregas.batch

Código de ScriptGeneraEntrega.bat

```
:: =====
:: Proyecto Datalogger for IoT Curso 2022-2023
:: Fecha: 26/01/2023
:: Autor: Pablo Villacorta, Rubén Serrano, Óscar Martín y Andrés Martín
:: Asignatura: Taller de Proyectos I
:: File: scriptCopia.bat (Script por lotes)
:: =====

@echo off
set "MM=%dt:~4,2%"
set "DD=%dt:~6,2%"
for /f "tokens=2 delims=:" %%a in ('wmic OS Get localdatetime /value') do set "dt=%%a"
set "AA=%dt:~2,2%"
set "AAAA=%dt:~0,4%"
set "MM=%dt:~4,2%"
set "DD=%dt:~6,2%"
set "HH=%dt:~8,2%"
set "Min=%dt:~10,2%"

:: --- Creacion de Carpetas ---
set "dia_hora=%DD%_%MM%_%AA%"
set "ArchivoDestino=05 - Entregas\Entrega_%dia_hora%"

IF EXIST "%ArchivoDestino%" (
    ECHO El archivo "%ArchivoDestino%" ya existe
) ELSE (md "%ArchivoDestino%")

IF EXIST "%ArchivoDestino%\00 - Doc" (
    ECHO El archivo "%ArchivoDestino%\00 - Doc" ya existe
) ELSE (md "%ArchivoDestino%\00 - Doc")

IF EXIST "%ArchivoDestino%\01 - Planification" (
    ECHO El archivo "%ArchivoDestino%\01 - Planification" ya existe
) ELSE (md "%ArchivoDestino%\01 - Planification")

IF EXIST "%ArchivoDestino%\02 - Design" (
    ECHO El archivo "%ArchivoDestino%\02 - Design" ya existe
) ELSE (md "%ArchivoDestino%\02 - Design")

IF EXIST "%ArchivoDestino%\03 - AnalisisConsumo" (
    ECHO El archivo "%ArchivoDestino%\03 - AnalisisConsumo" ya existe
) ELSE (md "%ArchivoDestino%\03 - AnalisisConsumo")

IF EXIST "%ArchivoDestino%\04 - Software" (
    ECHO El archivo "%ArchivoDestino%\04 - Software" ya existe
) ELSE (md "%ArchivoDestino%\04 - Software")

:: --- Doc ---
XCOPY "00 - Documentacion\Informe_proyecto.docx" "%ArchivoDestino%\00 - Doc" /S /Y /Q
IF /I "%ERRORLEVEL%" NEQ "0" (ECHO ERROR - No se ha podido copiar el archivo "00 - Documentacion\Informe_proyecto.docx")

:: --- Planification ---
XCOPY "01 - Planificacion del Trabajo\ControlCambios.txt" "%ArchivoDestino%\01 - Planification" /S /Y /Q
IF /I "%ERRORLEVEL%" NEQ "0" (ECHO ERROR - No se ha podido copiar el archivo "01 - Planificacion del Trabajo\ControlCambios.txt")
XCOPY "01 - Planificacion del Trabajo\DiagramaGannt_EnCurso.pdf" "%ArchivoDestino%\01 - Planification" /S /Y /Q
IF /I "%ERRORLEVEL%" NEQ "0" (ECHO ERROR - No se ha podido copiar el archivo "01 - Planificacion del Trabajo\DiagramaGannt_EnCurso.pdf")
XCOPY "01 - Planificacion del Trabajo\DiagramaGannt_Inicial.pdf" "%ArchivoDestino%\01 - Planification" /S /Y /Q
IF /I "%ERRORLEVEL%" NEQ "0" (ECHO ERROR - No se ha podido copiar el archivo "01 - Planificacion del Trabajo\DiagramaGannt_Inicial.pdf")
```

```

:: --- Design ---
:: (1)- Esquematico -
XCOPY "03 - Proyecto Proteus\Esquematico\Esquematico. Datalogger for IoT.prn" "%ArchivoDestino%\02 - Design" /S
/Y /Q
IF /I "%ERRORLEVEL%" NEQ "0" (ECHO ERROR - No se ha podido copiar el archivo "03 - Proyecto
Proteus\Esquematico\Esquematico. Datalogger for IoT.prn")
:: (2)- Proyecto Proteus -
XCOPY "03 - Proyecto Proteus\Datalogger_for_IoT_PRAO.pdsprj" "%ArchivoDestino%\02 - Design" /S /Y /Q
IF /I "%ERRORLEVEL%" NEQ "0" (ECHO ERROR - No se ha podido copiar el archivo "03 - Proyecto
Proteus\Datalogger_for_IoT_PRAO.pdsprj")
:: (3)- BOM -
XCOPY "03 - Proyecto Proteus\BOM" "%ArchivoDestino%\02 - Design\BOM" /S /Y /I /Q
IF /I "%ERRORLEVEL%" NEQ "0" (ECHO ERROR - No se ha podido copiar el archivo "03 - Proyecto Proteus\BOM")
:: (4)- Printed Layouts -
XCOPY "03 - Proyecto Proteus\Printed Layouts" "%ArchivoDestino%\02 - Design\Layouts" /S /Y /I /Q
IF /I "%ERRORLEVEL%" NEQ "0" (ECHO ERROR - No se ha podido copiar el archivo "03 - Proyecto Proteus\Printed
Layouts")

:: --- AnalisisConsumo ---
XCOPY "04 - Analisis consumo\Analisis Electrico y Termico PCB0.pdf" "%ArchivoDestino%\03 - AnalisisConsumo" /S
/Y /Q
IF /I "%ERRORLEVEL%" NEQ "0" (ECHO ERROR - No se ha podido copiar el archivo "04 - Analisis consumo\Analisis
Electrico y Termico PCB0.pdf")
XCOPY "04 - Analisis consumo\Analisis Electrico y Termico PCB1.pdf" "%ArchivoDestino%\03 - AnalisisConsumo" /S
/Y /Q
IF /I "%ERRORLEVEL%" NEQ "0" (ECHO ERROR - No se ha podido copiar el archivo "04 - Analisis consumo\Analisis
Electrico y Termico PCB1.pdf")
XCOPY "04 - Analisis consumo\Analisis Electrico y Termico PCB2.pdf" "%ArchivoDestino%\03 - AnalisisConsumo" /S
/Y /Q
IF /I "%ERRORLEVEL%" NEQ "0" (ECHO ERROR - No se ha podido copiar el archivo "04 - Analisis consumo\Analisis
Electrico y Termico PCB2.pdf")
XCOPY "04 - Analisis consumo\Analisis Electrico y Termico PCB3.pdf" "%ArchivoDestino%\03 - AnalisisConsumo" /S
/Y /Q
IF /I "%ERRORLEVEL%" NEQ "0" (ECHO ERROR - No se ha podido copiar el archivo "04 - Analisis consumo\Analisis
Electrico y Termico PCB3.pdf")

:: --- Software ---
XCOPY "06 - Software FPGA" "%ArchivoDestino%\04 - Software" /S /Y /I /Q
IF /I "%ERRORLEVEL%" NEQ "0" (ECHO ERROR - No se ha podido copiar el archivo "06 - Software FPGA")

echo -----
echo Documentacion generada con exito! & ::Este es un comentario
@pause

:: DOCUMENTACIÓN: (/S /Y /I ...)
:: https://www.minitool.com/es/respaldar-datos/comando-xcopy.html

```

Anexo C: Código en Verilog

Código de system.v

```
// =====
// RISC-V things
// by Jesús Arias
// -----
// -> EDITED:
// Proyecto Datalogger for IoT Curso 2022-2023
// Fecha: 26/01/2022
// Autor: Pablo Villacorta, Rubén Serrano, Óscar Martín y Andrés Martín
// Asignatura: Taller de Proyectos I
// File: system.v Se instancian todos los módulos diseñados en la FPGA
// =====
```

```
/*
Description:
A LaRVA RISC-V system with 8KB of internal memory, and three UART
```

Memory map:

0x00000000 to 0x00001FFF	Internal RAM (with initial contents)
0x00002000 to 0x1FFFFFFF	the same internal RAM repeated each 8KB
0x20000000 to 0xDFFFFFFF	xxxx
0xE0000000 to 0xE00000FF	IO registers
0xE0000100 to 0xFFFFFFFF	the same IO registers repeated each 256B

IO register map (all registers accessed as 32-bit words):

address	WRITE	READ
0xE0000000 UART0 TX data		UART0 RX data
0xE0000004 UART0 Baud Divider		UART0 flags
0xE0000008 UART1 TX data		UART1 RX data
0xE000000C UART1 Baud Divider		UART1 flags
0xE0000010 UART2 TX data		UART2 RX data
0xE0000014 UART2 Baud Divider		UART2 flags
0xE0000020 SPI0 TX data		SPI0 RX data
0xE0000024 SPI0 Control		SPI0 flags
0xE0000028 SPI0 Slave Select		xxxx
0xE0000030 SPI1 TX data		SPI1 RX data
0xE0000034 SPI1 Control		SPI1 flags
0xE0000038 SPI1 Slave Select		xxxx
0xE0000040 I2C data/control		I2C data/status
0xE0000044 I2C divider		
0xE0000060 MAX_COUNT [TMCF]		TIMER [tcount]
0xE0000080 GPOUT		GPOUT
0xE0000084 GPOUT		GPIN
0xE00000C0 Interrupt Enable		Interrupt enable
0xE00000E0 IRQ vector 0 Trap		
0xE00000E4 IRQ vector 1 RX		
0xE00000E8 IRQ vector 2 TX		
0xE00000EC IRQ vector 3 Timer		
0xE00000F0 IRQ vector 4 RX1		
0xE00000F4 IRQ vector 5 TX1		
0xE00000F8 IRQ vector 6 RX2		
0xE00000FC IRQ vector 7 TX2		

UART Baud Divider: Baud = Fcclk / (DIVIDER+1) , with DIVIDER >=7

UART FLAGS: bits 31-5 bit 4 bit 3 bit 2 bit 1 bit 0
xxxx OVE FE TEND THRE DV
DV: Data Valid (RX complete if 1. Cleared reading data register)
THRE: TX Holding register empty (ready to write to data register if 1)
TEND: TX end (holding reg and shift reg both empty if 1)
FE: Frame Error (Stop bit received as 0 if FE=1)
OVE: Overrun Error (Character received when DV was still 1)
(DV and THRE assert interrupt channels #4 and #5 when 1)

SPI Control: bits 31-14 bits 13-8 bits 7-0
xxxx DLEN DIVIDER
DLEN: Data lenght (8 to 32 bits)
DIVIDER: SCK frequency = Fclk / (2*(DIVIDER+1))

SPI Flags: bits 31-1 bit 0
xxxx BUSY
BUSY: SPI exchanging data when 1

SPI Slave Select: bits 31-2 bit 1 bit 0
xxxx ss1 ss0
ss0 : Selects the SPI slave 0 when 0 (active low) [BME680_CS]
ss1 : Selects the SPI slave 1 when 0 (active low) [ADC_CS]

I2C Data/Control: bit 10 bit 9 bit 8 bits 7-0
STOP START ACK DATA
STOP: Send Stop sequence
START: Send Start sequence
ACK: ACK bit. Must be 1 on writes and 0 on reads except last one
DATA: Data to write (Must be 0xFF on reads)
- ACK and DATA are ignored if START or STOP are one
- Do not set START and STOP simultaneously
- Repeated START is not supported
- Writing to this register sets the BUSY flag until the start,

stop, or data, is sent

I2C Data/Status: bit 9 bit 8 bits 7-0
BUSY ACK DATA
BUSY: Controller busy if 1.
ACK: Received ACK bit (for writes)
DATA: Received data (for reads)

I2C Divider: bits 6-0
SCL frequency = Fclk /(4*(DIVIDER+1))

MAX_COUNT: Holds the maximum value of the timer counter. When the timer reaches this value gets reset and request an interrupt if enabled.
Writes to MAX_COUNT also resets the timer and its interrupt flag.
TIMER: the current value of the timer (incremented each clock cycle)
Reads also clear the interrupt flag.

GPOUT: General purpose outputs.
GPIN: General purpose inputs.

// GPOUT[7] - GPOUT[6] - GPOUT[5] - GPOUT[4] - GPOUT[3] - GPOUT[2] - GPOUT[1] - GPOUT[0]
// DUST_CTRL - GAS_1V4_CTRL - GAS_5V_CTRL - STEPUP_CE - ice_led4 - ice_led3 - ice_led2 - ice_led1

Interrupt enable:

bit 0: Not used
bit 1: Enable UART0 RX interrupt if 1
bit 2: Enable UART0 TX interrupt if 1
bit 3: Enable TIMER interrupt if 1

```

bit 4: Enable UART1 RX interrupt if 1
bit 5: Enable UART1 TX interrupt if 1
bit 6: Enable UART2 RX interrupt if 1
bit 7: Enable UART2 TX interrupt if 1

```

Interrupt Vectors: Hold the addresses of the corresponding interrupt service routines.

```

*/
`include "laRVa.v"
`include "uart.v"
`include "SPI.v"

module SYSTEM (
    input clk,      // Main clock input 25MHz
    input reset,    // Global reset (active high)

    input rxd0,    // UART0
    output txd0,

    input rxd1,    // UART 1
    output txd1,   //

    input rxd2,    // UART 2
    output txd2,   //

    output ice_sck, // ICE SPI
    output ice_mosi,
    input ice_miso,
    output ice_ss0,
    output ice_ss1,

    output iceLoRA_sck, // ICE LORA SPI
    output iceLoRA_mosi,
    input iceLoRA_miso,
    output iceLoRA_ss,
    output iceLoRA_RST,

    output gpout0, // GPOUT
    output gpout1,
    output gpout2,
    output gpout3,
    output gpout4,
    output gpout5,
    output gpout6,
    output gpout7,

    input gpin0, // GPIN
    input gpin1,
    input gpin2,
    input gpin3,
    input gpin4,
    input gpin5,
    input gpin6,
    input gpin7,

    output fssb // Flash CS
);

wire cclk; // CPU clock
assign cclk=clk;

```

```

////////// CPU ///////////
wire [31:0] ca; // CPU Address
wire [31:0] cdo; // CPU Data Output
wire [3:0] mwe; // Memory Write Enable (4 signals, one per byte lane)
wire irq;
wire [31:2] ivecotor; // Where to jump on IRQ
wire trap; // Trap irq (to IRQ vector generator)

laRVa cpu (
    .clk (cclk),
    .reset (reset),
    .addr (ca[31:2]),
    .wdata (cdi),
    .wstrb (mwe),
    .rdata (cdi),
    .irq (irq),
    .ivecotor (ivecotor),
    .trap (trap)
);

```

```

////// Memory mapping
wire iramcs;
wire iocs;

// Internal RAM selected in lower 512MB (0-0x1FFFFFFF)
assign iramcs = (ca[31:29]==3'b000);
// IO selected in last 512MB (0xE0000000-0xFFFFFFFF)
assign iocs = (ca[31:29]==3'b111);

// Input bus mux
reg [31:0]cdi; // Not a register
always@*
casex ({iocs,iramcs})
    2'b01: cdi<=mdo;
    2'b10: cdi<=iodo;
    default: cdi<=32'hxxxxxxxx;
endcase

```

```

////////// internal memory ///////////
wire [31:0] mdo; // Output data
ram32 ram0 (.clk(~cclk), .re(iramcs), .wrlnes(iramcs?mwe:4'b0000),
    .addr(ca[13:2]), //Maximo 4096 direcciones
    .data_read(mdo), .data_write(cdo));

```

```

////////// Peripherals ///////////

```

```

// ---> CONFIGURACIÓN DE LOS CHIP SELECT [CS]
wire uartcs; // UART at offset 0x00
wire spics; // SPI at offset 0x20
wire i2ccs; // I2C at offset 0x40
wire tempcs; // Timer at offset 0X60
wire gpcs; // GENERAL PURPOSE at offset 0x80
wire iencs; // INTERRUPT ENABLE at offset 0xC0
wire irqcs; // IRQEN at offset 0xE0
// ...
// other at offset 0xE0

```

```

assign uartcs = iocs&(ca[7:5]==3'b000);
assign spics = iocs&(ca[7:5]==3'b001);
assign i2ccs = iocs&(ca[7:5]==3'b010);
assign tempcs = iocs&(ca[7:5]==3'b011);
assign gpcs = iocs&(ca[7:5]==3'b100);

assign iencs = iocs&(ca[7:5]==3'b110); //Interrupt Enable Chip Select
assign irqcs = iocs&(ca[7:5]==3'b111);

// Peripheral output bus mux
reg [31:0]iodo; // Not a register
always@*
casex (ca[7:2])
6'b000000: iodo<={24'h0,uart0_do};
6'b000001: iodo<={27'h0,ove0,fe0,tend0,thre0,dv0};
6'b000010: iodo<={24'h0,uart1_do};
6'b000011: iodo<={27'h0,ove1,fe1,tend1,thre1,dv1};
6'b000100: iodo<={24'h0,uart2_do};
6'b000101: iodo<={27'h0,ove2,fe2,tend2,thre2,dv2};

6'b001000: iodo<=rx_spi;      // SPI_RX
6'b001001: iodo<={31'h0,busy}; // SPI FLAG (busy)
6'b001100: iodo<=rx_spiLoRA; // SPI_LoRA_RX
6'b001101: iodo<={31'h0,busyLoRA}; // SPI_LoRA FLAG (busy)

6'b100000: iodo<={24'h0,GPOUT}; // GPOUT
6'b100001: iodo<={24'h0,gpin7,gpin6,gpin5,gpin4,gpin3,gpin2,gpin1,gpin0}; // GPIN

//6'b010000: iodo<={...}; // I2C...
//6'b010001: iodo<={...}; // I2C...

6'b011xxx: iodo<=tcount; // TIMER
6'b110xxx: iodo<={24'h0,irqen};
default: iodo<=32'hxxxxxxxx;
endcase

///////////////////////////////
// GPOUT

// GPIN: Siempre vale el dato de entrada.
// GPOUT: Se carga desde memoria

reg [7:0] GPOUT=0;

wire gpoutcs;
assign gpoutcs = gpcs & (~ca[2]);

always @(posedge cclk)
begin
  if(gpoutcs & mwe[0]) // Escritura en GPOUTCS
    GPOUT <= cdo[7:0];
end

assign gpout0 = GPOUT[0];
assign gpout1 = GPOUT[1];
assign gpout2 = GPOUT[2];
assign gpout3 = GPOUT[3];
assign gpout4 = GPOUT[4];
assign gpout5 = GPOUT[5];
assign gpout6 = GPOUT[6];
assign gpout7 = GPOUT[7];

/////////////////////////////

```

```

// TIMER

reg [31:0] tcount = 0; // TCNT: Timer Counter Register
reg [31:0] TMR = 180000; // Time Match Register [MAX_COUNT: Holds the maximum value of the timer counter] // 4
seg: TMR = 32'h05F5E100;
reg TMF = 0; //Time Match Flag Bit
reg TMF0 = 0; //Time Match Flag Bit 0
reg TMF1 = 0; //Time Match Flag Bit 1

// Necesitamos 3 registros ya que tenemos que mantener la señal
// de interrupcion durante al menos 2 ciclos de reloj!!!
// La señal de interrupcion posedge_TMF salta cuando TMF & ~TMF1

// Ciclo:      TMF      TMF0      TMF1
//           0        0        0  (No Interr)
// tcount==TMR   1        0        0  (Interr)
//           1        1        0  (Interr)
//           1        1        1  (No interr)

wire TMF_wire;
wire rd_timer;
wire wr_timer;

assign rd_timer = tempcs & (mwe==4'b0000);
assign wr_timer = tempcs & (mwe==4'b1111);
assign posedge_TMF = ((~TMF1) & TMF);

always @(posedge cclk)
begin
    if(wr_timer) //Escritura en TMR
    begin
        TMF <= 0; //Desactivo el Flag de fin de cuenta al leer de timer / escribir
        TMR <= cdo[31:0];
        tcount <= 0; //Reset del contador
    end
    else
    begin
        if(rd_timer) //Lectura en TMR
            TMF <= 0; //Desactivo el Flag de fin de cuenta al leer de timer / escribir

        tcount <= tcount+1; //Incremento del contador
        TMF0<=TMF; // Cargo el TMF en TMF0
        TMF1<=TMF0; // Cargo el TMF0 en TMF1
        if (tcount==TMR)
        begin
            tcount <= 0; //Reset del contador
            TMF <= 1; //Ativo el Flag de fin de cuenta
        end
    end
end
end

///////////////////////
// UART0

wire tend0,thre0,dv0,fe0,ove0; // Flags
wire [7:0] uart0_do; // RX output data
wire uwrtx0; // UART TX write
wire urd0; // UART RX read (for flag clearing)
wire uwrbaud0; // UART BGR write
// Register mapping
// Offset 0: write: TX Holding reg
// Offset 0: read strobe: Clear DV, OVE (also reads RX data buffer)
// Offset 1: write: BAUD divider
assign uwrtx0 = uartcs & (~ca[4])& (~ca[3])& (~ca[2]) & mwe[0];

```

```

assign uwrbaud0 = uartcs & (~ca[4])& (~ca[3])& (ca[2]) & mwe[0] & mwe[1];
assign urd0    = uartcs & (~ca[4])& (~ca[3])& (~ca[2]) & (mwe==4'b0000); // Clear DV, OVE flags

UART_CORE #(.BAUDBITS(12)) uart0 (.clk(cclk), .txd(tx0), .rxn(rx0),
.d(cdo[15:0]), .wrtx(wr0), .wrbaud(uwrbaud0), .rd(urd0), .q(uart0_do),
.dv(dv0), .fe(fe0), .ove(ove0), .tend(tend0), .thre(thre0) );

///////////////////////
// UART1

wire tend1,thre1,dv1,fe1,ove1; // Flags
wire [7:0] uart1_do; // RX output data
wire wrx1; // UART TX write
wire urd1; // UART RX read (for flag clearing)
wire uwrbaud1; // UART BGR write
// Register mapping
// Offset 0: write: TX Holding reg
// Offset 0: read strobe: Clear DV, OVE (also reads RX data buffer)
// Offset 1: write: BAUD divider
assign wrx1 = uartcs & (~ca[4])& (ca[3])& (~ca[2]) & mwe[0];
assign uwrbaud1 = uartcs & (~ca[4])& (ca[3])& (ca[2]) & mwe[0] & mwe[1];
assign urd1 = uartcs & (~ca[4])& (ca[3])& (~ca[2]) & (mwe==4'b0000); // Clear DV, OVE flags

UART_CORE #(.BAUDBITS(12)) uart1 (.clk(cclk), .txd(tx1), .rxn(rx1),
.d(cdo[15:0]), .wrtx(wr1), .wrbaud(uwrbaud1), .rd(urd1), .q(uart1_do),
.dv(dv1), .fe(fe1), .ove(ove1), .tend(tend1), .thre(thre1) );

///////////////////////
// UART2

wire tend2,thre2,dv2,fe2,ove2; // Flags
wire [7:0] uart2_do; // RX output data
wire wrx2; // UART TX write
wire urd2; // UART RX read (for flag clearing)
wire uwrbaud2; // UART BGR write
// Register mapping
// Offset 0: write: TX Holding reg
// Offset 0: read strobe: Clear DV, OVE (also reads RX data buffer)
// Offset 1: write: BAUD divider
assign wrx2 = uartcs & (ca[4])& (~ca[3])& (~ca[2]) & mwe[0];
assign uwrbaud2 = uartcs & (ca[4])& (~ca[3])& (ca[2]) & mwe[0] & mwe[1];
assign urd2 = uartcs & (ca[4])& (~ca[3])& (~ca[2]) & (mwe==4'b0000); // Clear DV, OVE flags

UART_CORE #(.BAUDBITS(12)) uart2 (.clk(cclk), .txd(tx2), .rxn(rx2),
.d(cdo[15:0]), .wrtx(wr2), .wrbaud(uwrbaud2), .rd(urd2), .q(uart2_do),
.dv(dv2), .fe(fe2), .ove(ove2), .tend(tend2), .thre(thre2) );

///////////////////////
// ICE SPI

reg [13:0] spi_ctrl=0; // Registro que contiene los valores de dlen y divider (procedentes de cdo)
reg [1:0] spi_ss=2'b11; // Registro Slave Select

wire[5:0] dlen_spi;
wire[7:0] divider_spi;

wire busy;
wire spi_wr; // Señal de activación de escritura tanto en el control como en el rx/tx
wire spi_wr_ctrl; // Señal de activación de escritura en el registro spi_control desde cdo
wire spi_wr_ss; // Señal de activación de escritura en el registro spi_ss desde cdo
wire[31:0] rx_spi;

assign spi_wr = spics & (~ca[4]) & (~ca[3]) & (~ca[2]) & (mwe == 4'b1111);

```

```

assign spi_wr_ctrl = spics & (~ca[4]) & (~ca[3]) & (ca[2]) & (mwe == 4'b1111);
assign spi_wr_ss = spics & (~ca[4]) & (ca[3]) & (~ca[2]) & (mwe == 4'b1111);

assign dlen_spi = spi_ctrl[13:8];
assign divider_spi = spi_ctrl[7:0];

assign ice_ss0 = spi_ss[0];
assign ice_ss1 = spi_ss[1];

always @(posedge cclk)
begin
    if(spi_wr_ctrl) //Escritura en spi_ctrl
        spi_ctrl <= cdo[13:0];
    if(spi_wr_ss) //Escritura en spi_ss
        spi_ss <= cdo[1:0];
end

SPI_master spi(
    .clk(cclk), .miso(ice_miso), .wr(spi_wr),
    .din(cdo[31:0]),
    .divider(divider_spi), .bits(dlen_spi),
    .sck(ice_sck), .mosi(ice_mosi), .busy(busy),
    .dout(rx_spi)
);

```

```

///////////////////////////////
// LORA SPI

reg [13:0] spiLoRA_ctrl=0; // Registro que contiene los valores de dlen y divider (procedentes de cdo)
reg spiLoRA_ss=1'b1; // Registro Slave Select
reg LoRA_RST = 0; // LoRA_RST

wire[5:0] dlen_spiLoRA;
wire[7:0] divider_spiLoRA;

wire busyLoRA;
wire spiLoRA_wr; // Señal de activación de escritura tanto en el control como en el rx/tx
wire spiLoRA_wr_ctrl; // Señal de activación de escritura en el registro spi_control desde cdo
wire spiLoRA_wr_ss; // Señal de activación de escritura en el registro spiLoRA_ss desde cdo
wire[31:0] rx_spiLoRA;

assign spiLoRA_wr = spics & (ca[4]) & (~ca[3]) & (~ca[2]) & (mwe == 4'b1111);
assign spiLoRA_wr_ctrl = spics & (ca[4]) & (~ca[3]) & (ca[2]) & (mwe == 4'b1111);
assign spiLoRA_wr_ss = spics & (ca[4]) & (ca[3]) & (~ca[2]) & (mwe == 4'b1111);

assign dlen_spiLoRA = spiLoRA_ctrl[13:8];
assign divider_spiLoRA = spiLoRA_ctrl[7:0];

assign iceLoRA_ss = spiLoRA_ss;
assign iceLoRA_RST = LoRA_RST;

always @(posedge cclk)
begin
    if(spiLoRA_wr_ctrl) //Escritura en spiLoRA_ctrl
        spiLoRA_ctrl <= cdo[13:0];
    if(spiLoRA_wr_ss) //Escritura en spiLoRA_ss
        spiLoRA_ss <= cdo[0];
end

SPI_master spiLoRA(
    .clk(cclk), .miso(iceLoRA_miso), .wr(spiLoRA_wr),

```

```

    .din(cdo[31:0]),
    .divider(divider_spiLoRA),.bits(dlen_spiLoRA),
    .sck(iceLoRA_sck), .mosi(iceLoRA_mosi), .busy(busyLoRA),
    .dout(rx_spiLoRA)
);

///////////
// Interrupt control

// -> IRQ enable reg (Registro de 8 bits) [bit0 unused]:
// IRQEN[7] - IRQEN[6] - IRQEN[5] - IRQEN[4] - IRQEN[3] - IRQEN[2] - IRQEN[1] - IRQEN[0]
// U2TX - U2RX - U1TX - U1RX - TIMER - U0TX - U0RX - unused

reg [7:0]irqen=0;
always @(posedge cclk or posedge reset) begin
  if (reset) irqen<=0; else
    if (iencs &mwe[0]) irqen<=cdo[7:0];
end

// -> IRQ vectors

// IRQVEC[7] - IRQVEC[6] - IRQVEC[5] - IRQVEC[4] - IRQVEC[3] - IRQVEC[2] - IRQVEC[1] - IRQVEC[0]
// U2TX - U2RX - U1TX - U1RX - TIMER - U0TX - U0RX - trap

reg [31:2]irqvect[0:7]; //Array of 8 irqvectors

always @(posedge cclk) if (irqcs & (mwe==4'b1111)) irqvect[ca[4:2]]<=cdo[31:2];

// Enabled IRQs
wire [6:0]irqpen={
  irqen[7]&thre2,      //irqpen[6] UART2TX
  irqen[6]&dv2,        //irqpen[5] UART2RX
  irqen[5]&thre1,      //irqpen[4] UART1TX
  irqen[4]&dv1,        //irqpen[3] UART1RX
  irqen[3]&posedge_TMF, //irqpen[2] TIMER. Posedge Time Match Flag
  irqen[2]&thre0,       //irqpen[1] UART0 TX
  irqen[1]&dv0,         //irqpen[0] UART0 RX
}; // pending IRQs

// Priority encoder
wire [2:0]vecn = trap ? 3'b000 : ( // ECALL, EBREAK: highest priority
  irqpen[0] ? 3'b001 : ( // UART0 RX
  irqpen[1] ? 3'b010 : ( // UART0 TX
  irqpen[2] ? 3'b011 : ( // TIMER
  irqpen[3] ? 3'b100 : ( // UART1 RX
  irqpen[4] ? 3'b101 : ( // UART1 TX
  irqpen[5] ? 3'b110 : ( // UART2 RX
  irqpen[6] ? 3'b111 : // UART2 TX
    3'bxxx ))))))));
assign ivecotor = irqvect[vecn];
assign irq = (irqpen!=0)|trap;

endmodule // System

```

```

///////////
//-----
//-- 32-bit RAM Memory with independent byte-write lanes
//-----
/////////

```

```

module ram32
(
    input  clk,
    input  re,
    input [3:0] wrlanes,
    input [L2N_RAM_SIZE-1:0] addr,
    output [31:0] data_read,
    input [31:0] data_write
);
parameter RAM_SIZE = 4096;
localparam L2N_RAM_SIZE = $clog2(RAM_SIZE); //Calcula el logaritmo para ver los bits

reg [31:0] ram_array [0:RAM_SIZE-1];
reg [31:0] data_out;

assign data_read = data_out;

always @(posedge clk) begin
    if (wrlanes[0]) ram_array[addr][ 7: 0] <= data_write[ 7: 0];
    if (wrlanes[1]) ram_array[addr][15: 8] <= data_write[15: 8];
    if (wrlanes[2]) ram_array[addr][23:16] <= data_write[23:16];
    if (wrlanes[3]) ram_array[addr][31:24] <= data_write[31:24];
end

always @(posedge clk) begin
    if (re) data_out <= ram_array[addr];
end

initial begin
`ifdef SIMULATION
    $readmemh("rom.hex", ram_array);
`else
    $readmemh("rand.hex", ram_array);
`endif
end

endmodule

```

Código de main.v

```
// =====
// RISC-V things
// by Jesús Arias
// -----
// -> EDITED:
// Proyecto Datalogger for IoT Curso 2022-2023
// Fecha: 26/01/2023
// Autor: Pablo Villacorta, Rubén Serrano, Óscar Martín y Andrés Martín
// Asignatura: Taller de Proyectos I
// File: main.v (top level entity)
// Incorpora las entradas en la FPGA de los modulos agregados
// =====
`include "system.v"
`include "pll.v"

// Top module. (signals assigned to actual pins in file "pines.pcf")
module main(
    input CLKIN,      // Input clock from crystal oscillator (16MHz)
    // SPI0
    output ICE_SCK,
    output ICE_MOSI,
    output BME680_CS, //ss0
    output ADC_CS,   //ss1
    input ICE_MISO,
    // SPI1
    output LoRA_SCK,
    output LoRA_RST,
    output LoRA_MOSI,
    output LoRA_CS, //ss
    input LoRA_MISO,
    // GPOUT
    output GPOUT0, // ice_led1
    output GPOUT1, // ice_led2
    output GPOUT2, // ice_led3
    output GPOUT3, // ice_led4
    output GPOUT4, // STEPUP_CE
    output GPOUT5, // GAS_5V_CTRL
    output GPOUT6, // GAS_1V4_CTRL
    output GPOUT7, // DUST_CTRL
    // GPIN
    input GPIN0, //P112 GPIN0
    input GPIN1, //P113 GPIN1
    input GPIN2, //P114 GPIN2
    input GPIN3, //P115 GPIN3
    input GPIN4, //P116 GPIN4
    input GPIN5, //P117 GPIN5
    input GPIN6, //P118 GPIN6
    input GPIN7, //P119 GPIN7
    // UART0
    input RXD0,
    output TXD0,
    // UART1
    input RXD1,
    output TXD1,
    // UART2
    input RXD2,
    output TXD2,
```

```

    output FSS // Flash SS
);

//-- PLL: generates a 25MHz master clock from a 16MHz input
wire clk,pll_lock;

pll
pll1(
    .clock_in(CLKIN),
    .clock_out(clk),
    .locked(pll_lock)
);

//assign clk=CLKIN;
//assign pll_lock=1'b1;

// Game controller and SPI pin mappings
assign JY4=1'b1; // Game controller power
wire [7:0]pinin;

//assign XBHE=1'b0;
//assign XBLE=1'b0;

// Instance of the system
SYSTEM sys1(.clk(clk), .reset(reset),
    .txd0(TXD0), .rxd0(RXD0),
    .txd1(TXD1), .rxd1(RXD1),
    .txd2(TXD2), .rxd2(RXD2),
    .ice_sck(ICE_SCK), .ice_mosi(ICE_MOSI), .ice_miso(ICE_MISO), .ice_ss0(BME680_CS), .ice_ss1(ADC_CS),
    .iceLoRA_sck(LoRA_SCK), .iceLoRA_mosi(LoRA_MOSI), .iceLoRA_miso(LoRA_MISO), .iceLoRA_ss(LoRA_CS),
    .iceLoRA_RST(LoRA_RST),
    .gpout0(GPOUT0), .gpout1(GPOUT1), .gpout2(GPOUT2), .gpout3(GPOUT3),
    .gpout4(GPOUT4), .gpout5(GPOUT5), .gpout6(GPOUT6), .gpout7(GPOUT7),
    .pin0(GPIN0), .pin1(GPIN1), .pin2(GPIN2), .pin3(GPIN3),
    .pin4(GPIN4), .pin5(GPIN5), .pin6(GPIN6), .pin7(GPIN7),
    .fssb(FSS)
);

// Automatic RESET pulse: Reset is held active for 255 cycles after PLL lock

reg [21:0]cnt=22'h3fffff;
wire reset=(cnt!=0);

always @(posedge clk) cnt<= reset ? cnt-1: cnt;

///////////////////////////////
// Bidirectional data bus of the external RAM
// Tristates have to be instantiated usign the SB_IO module (specific of the ICE40 FPGA)

wire [15:0]xdi; // internal data input bus
wire [15:0]xdo; // internal data output bus

///////////////////////////////
// SB_IOs
wire oe;
assign oe=XOE; // activate tristates on writes (XOE inactive == High)

SB_IO #( .PIN_TYPE(6'b 1010_01), .PULLUP(1'b 0) ) xd_instance0
(
    .PACKAGE_PIN( XD[0]),
    .OUTPUT_ENABLE( oe ),
    .D_OUT_0( xdo[0]),
    .D_IN_0( xdi[0] )
);
SB_IO #( .PIN_TYPE(6'b 1010_01), .PULLUP(1'b 0) ) xd_instance1

```

```

( .PACKAGE_PIN( XD[1]),
  .OUTPUT_ENABLE( oe ),
  .D_OUT_0( xdo[1]),
  .D_IN_0( xdi[1]) );
SB_IO #( .PIN_TYPE(6'b 1010_01), .PULLUP(1'b 0) ) xd_instance2
( .PACKAGE_PIN( XD[2]),
  .OUTPUT_ENABLE( oe ),
  .D_OUT_0( xdo[2]),
  .D_IN_0( xdi[2]) );
SB_IO #( .PIN_TYPE(6'b 1010_01), .PULLUP(1'b 0) ) xd_instance3
( .PACKAGE_PIN( XD[3]),
  .OUTPUT_ENABLE( oe ),
  .D_OUT_0( xdo[3]),
  .D_IN_0( xdi[3]) );
SB_IO #( .PIN_TYPE(6'b 1010_01), .PULLUP(1'b 0) ) xd_instance4
( .PACKAGE_PIN( XD[4]),
  .OUTPUT_ENABLE( oe ),
  .D_OUT_0( xdo[4]),
  .D_IN_0( xdi[4]) );
SB_IO #( .PIN_TYPE(6'b 1010_01), .PULLUP(1'b 0) ) xd_instance5
( .PACKAGE_PIN( XD[5]),
  .OUTPUT_ENABLE( oe ),
  .D_OUT_0( xdo[5]),
  .D_IN_0( xdi[5]) );
SB_IO #( .PIN_TYPE(6'b 1010_01), .PULLUP(1'b 0) ) xd_instance6
( .PACKAGE_PIN( XD[6]),
  .OUTPUT_ENABLE( oe ),
  .D_OUT_0( xdo[6]),
  .D_IN_0( xdi[6]) );
SB_IO #( .PIN_TYPE(6'b 1010_01), .PULLUP(1'b 0) ) xd_instance7
( .PACKAGE_PIN( XD[7]),
  .OUTPUT_ENABLE( oe ),
  .D_OUT_0( xdo[7]),
  .D_IN_0( xdi[7]) );
SB_IO #( .PIN_TYPE(6'b 1010_01), .PULLUP(1'b 0) ) xd_instance8
( .PACKAGE_PIN( XD[8]),
  .OUTPUT_ENABLE( oe ),
  .D_OUT_0( xdo[8]),
  .D_IN_0( xdi[8]) );
SB_IO #( .PIN_TYPE(6'b 1010_01), .PULLUP(1'b 0) ) xd_instance9
( .PACKAGE_PIN( XD[9]),
  .OUTPUT_ENABLE( oe ),
  .D_OUT_0( xdo[9]),
  .D_IN_0( xdi[9]) );
SB_IO #( .PIN_TYPE(6'b 1010_01), .PULLUP(1'b 0) ) xd_instance10
( .PACKAGE_PIN( XD[10]),
  .OUTPUT_ENABLE( oe ),
  .D_OUT_0( xdo[10]),
  .D_IN_0( xdi[10]) );
SB_IO #( .PIN_TYPE(6'b 1010_01), .PULLUP(1'b 0) ) xd_instance11
( .PACKAGE_PIN( XD[11]),
  .OUTPUT_ENABLE( oe ),
  .D_OUT_0( xdo[11]),
  .D_IN_0( xdi[11]) );
SB_IO #( .PIN_TYPE(6'b 1010_01), .PULLUP(1'b 0) ) xd_instance12
( .PACKAGE_PIN( XD[12]),
  .OUTPUT_ENABLE( oe ),
  .D_OUT_0( xdo[12]),
  .D_IN_0( xdi[12]) );
SB_IO #( .PIN_TYPE(6'b 1010_01), .PULLUP(1'b 0) ) xd_instance13
( .PACKAGE_PIN( XD[13]),
  .OUTPUT_ENABLE( oe ),
  .D_OUT_0( xdo[13]),

```

```
.D_IN_0(    xdi[13]) );
SB_IO #( .PIN_TYPE(6'b 1010_01), .PULLUP(1'b 0) ) xd_instance14
( .PACKAGE_PIN(  XD[14]),
  .OUTPUT_ENABLE( oe  ),
  .D_OUT_0(    xdo[14]),
  .D_IN_0(    xdi[14]) );
SB_IO #( .PIN_TYPE(6'b 1010_01), .PULLUP(1'b 0) ) xd_instance15
( .PACKAGE_PIN(  XD[15]),
  .OUTPUT_ENABLE( oe  ),
  .D_OUT_0(    xdo[15]),
  .D_IN_0(    xdi[15]) );

endmodule
```

Anexo D: Pines.pcf

Uso de los pines de la FPGA (PCB1)

del Taller de Proyectos 1

Máster de Telecomunicación

Universidad de

Valladolid

Curso 2022-2023

Jesús M. Hernández & Jesús Arias

BANK 3

set_io HSYNC 1 # HSYNC

set_io P2 2

set_io DTR 3 # /DTR /RESET

set_io CTS 4 # /CTS

set_io RTS 7 # /RTS /BOOT

set_io TXD0 8 # Antes RXD0

set_io RXD0 9 # Antes TXD0

set_io GREEN3 10 # GREEN3

set_io GREEN4 11 # GREEN2

set_io GREEN1 12 # GREEN1

set_io GREEN0 15 # GREEN0

set_io BLUE3 16 # BLUE3

set_io BLUE2 17 # BLUE2

set_io BLUE1 18 # BLUE1

set_io BLUE0 19 # BLUE0

set_io GPOUT4 20 # STEPUP_CE

set_io CLKIN 21 # ICE_CLK

set_io GPOUT5 22 # GAS_5V_CTRL

set_io P23 23

set_io GPOUT6 24 # GAS_1V4_CTRL

set_io GPOUT7 25 # DUST_CTRL

set_io ADC_CS 26 # ADC_CS (SS1)

set_io SDA 28 # SDA

set_io SCL 29 # SCL

set_io BME680_CS 31 # BME680_CS (SS0)

set_io P32 32

set_io P33 33

set_io P34 34

BANK 2

set_io ICE_SCK 37 # ICE_SCK (BME y ADC)

set_io ICE_MISO 38 # ICE_MISO (BME y ADC)

set_io ICE_MOSI 39 # ICE_MOSI (BME y ADC)

set_io P41 41

set_io P42 42

set_io P43 43

set_io P44 44

set_io P45 45

set_io P47 47

set_io P48 48

set_io RXD1 49 # GPS_TX

set_io TXD1 52 # GPS_RX

set_io P55 55

set_io P56 56

set_io P60 60

```
set_io P61    61
set_io P62    62
set_io MB1    63 # MB1  Jumpers multiboot
set_io MB0    64 # MB0  Jumpers multiboot
```

SPI

```
set_io MOSI   67 # MOSI
set_io MISO   68 # MISO

set_io SCK    70 # SCK
set_io FSS    71 # /SS Flash
```

BANK 1

```
set_io GPIN0  73 # * P73 GPIN0
set_io P74    74
set_io P75    75 # *
set_io P76    76

set_io GPIN1  78 # * P78 GPIN1
set_io P79    79 # *

set_io P80      80 # *
set_io GPIN2  81 # * P81 GPIN2
set_io P82      82 # *
set_io P83      83 # *
set_io P84      84 # *
set_io GPIN3  85 # * P85 GPIN3
```

```
set_io P87      87 # * P87 GPIN6
set_io GPIN4  88 # * P88 GPIN4
```

```
set_io P90    90
set_io P91    91 # *
```

```
set_io GPIN5  93 # * P93 GPIN5
set_io P94    94 # *
set_io GPIN6  95 # * P95 GPIN6
set_io P96    96
set_io P97    97 # *
set_io P98    98
set_io GPIN7  99 # * P95 GPIN7
```

```
set_io P101   101 # *
set_io P102   102
```

```
set_io P104   104 # *
set_io P105   105 # *
set_io P106   106
set_io P107   107
```

BANK 0

```
set_io LoRA_MOSI 110 # * P110 LORA_MOSI
set_io LoRA_MISO 113 # * P113 LORA_MISO
set_io LoRA_CS   115 # * P115 LORA_CS
set_io LoRA_SCK  117 # * P117 LORA_SCK
set_io LoRA_RST  119 # * P119 LORA_RST
```

```
set_io P120   120 # *
set_io RXD2   121 # M1_TX ESP32-C3
```

```
set_io P122 122 # *
set_io TXD2 124 # M1_RX ESP32-C3
set_io P125 125 # *

set_io P128 128 # *
set_io P129 129 # *

set_io P130 130 # *
set_io P134 134 # *
set_io VSYNC 135 # VSYNC
set_io GPOUT0 136 # ICE_LED1
set_io GPOUT1 137 # ICE_LED2
set_io GPOUT2 138 # ICE_LED3
set_io GPOUT3 139 # ICE_LED4

set_io RED3 141 # RED3
set_io RED2 142 # RED2
set_io RED1 143 # RED1
set_io RED0 144 # RED0
```