



Universidad de Valladolid



**ESCUELA DE INGENIERÍAS
INDUSTRIALES**

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

**Implementación de un entorno de
comunicación Bluetooth, basado en el
módulo CC2650, para la transmisión del
ritmo cardiaco.**

Autor:

Álvarez Urueña, Alonso

Tutor:

Pérez Turiel, Javier

Ingeniería de Sistemas y Automática

Valladolid, julio de 2021.

RESUMEN

En el marco del desarrollo de una plataforma de rehabilitación neuromotora robotizada, se pretende incorporar la posibilidad de establecer comunicaciones inalámbricas (Bluetooth Low Energy). Para ello, se integra un módulo comercial CC2650 (TI) al dispositivo robotizado, que está basado en el microcontrolador TMS320F28069M (TI).

Como primera fase de desarrollo, se plantea la transmisión del ritmo cardiaco por Bluetooth: el microcontrolador F28069M recibe las señales electrocardiográficas registradas mediante un módulo de Sparkfun basado en el AD8232, calcula el ritmo cardiaco mediante el algoritmo de Pan-Tomkins y lo transmite al módulo CC2650 mediante comunicación I2C para que este envíe dicha información mediante Bluetooth a otro módulo CC2650, el cual transmitirá el valor del ritmo cardiaco a otro microcontrolador F28069M.

ABSTRACT

Within the framework of the development of a robotic neuromotor rehabilitation platform, the aim is to incorporate the possibility of establishing wireless communications (Bluetooth Low Energy). To this end, a commercial CC2650 (TI) module is integrated into the robotic device, which is based on the TMS320F28069M (TI) microcontroller.

As a first stage of development, the transmission of the heart rate via Bluetooth is proposed: the F28069M microcontroller receives the electrocardiographic signals recorded by a Sparkfun module based on the AD8232, calculates the heart rate using the Pan-Tomkins algorithm and transmits it to the CC2650 module via I2C communication so that it can send this information via Bluetooth to another CC2650 module, which will transmit the heart rate value to another F28069M microcontroller.

PALABRAS CLAVE

Bluetooth Low Energy, comunicación inalámbrica, microcontrolador, ritmo cardiaco y Texas Instruments.

KEYWORDS

Bluetooth Low Energy, wireless communication, microcontroller, heart rate and Texas Instruments.



ÍNDICE

1. Introducción y objetivos.....	9
2. Estado del arte.....	13
2.1. Introducción	13
2.2. LAUNCHXL-F28069M	17
2.3. LAUNCHXL-CC2650	22
2.4. Protocolo de comunicación BLE	39
2.5. Protocolo de comunicación I2C	56
2.6. Electrocardiograma (ECG)	64
3. Desarrollo y programación del proyecto.....	71
3.1. Paso 1: F28069M_TX.....	71
3.2. Paso 2: CC2650_Peripheral_BLE.....	77
3.3. Paso 3: CC2650_Central_BLE	85
3.4. Paso 4: F28069M_RX	93
3.5. Tabla de conexiones del proyecto	96
3.6. Comprobación de resultados y captura del montaje final	97
4. Conclusiones.....	103
5. Bibliografía	105
6. Anexos.....	111
6.1. Cálculo resistencia pull-up para el bus I2C.....	111

ÍNDICE DE ILUSTRACIONES

Ilustración 1. Esquema del proyecto.....	9	
Ilustración 2. Logo de ITAP.....	11	
Ilustración 3. Foto del dispositivo Trazein.	13	
Ilustración 4. Foto del prototipo detector de ECG/EEG/EMG.	14	
Ilustración 5. Foto del sistema de cuidado de la salud portátil desarrollado.	14	
Ilustración 6. Esquema de funcionamiento de RobHand.....	15	
Ilustración 7. Un usuario realizando una terapia bilateral asistida por EMG con la plataforma robótica RobHand.	15	
Ilustración 8. Microcontrolador LAUNCHXL-F28069M.	17	
Ilustración 9. Funcionamiento de los núcleos C28x y CLA dentro de la CPU.....	18	
Ilustración 10. Esquema de puertos y funcionalidades del microcontrolador.	18	
Ilustración 11. Code Composer Studio.	Ilustración 12. Energía. Ilustración 13. controlSUITE.....	21
Ilustración 14. Microcontrolador LAUNCHXL-CC2650	22	
Ilustración 15. Esquema de puertos y funcionalidades del microcontrolador.	23	
Ilustración 16. Diagrama de bloques del microprocesador.	24	
Ilustración 17. “Pinout” del microcontrolador (I).....	26	
Ilustración 18. “Pinout” del microcontrolador (II).....	26	
Ilustración 19. Instalador de CCS.....	27	
Ilustración 20. Explorador de recursos.	28	
Ilustración 21. Menú desplegable “Help”.....	29	
Ilustración 22. Seleccionar “ARM Compiler Tools 5.2.9”.	29	
Ilustración 23. Menú desplegable “Project”.....	30	
Ilustración 24. Importar los programas “App” y “Stack”.	30	
Ilustración 25. Seleccionar el compilador “TI v5.2.9”.	31	
Ilustración 26. Hacer clic en “Edit” en la ventana de propiedades del proyecto.....	32	
Ilustración 27. Seleccionar “Preferences” en la ventana desplegable.	32	
Ilustración 28. Hacer clic en “Refresh” para actualizar los paquetes de software que el IDE tiene listados.....	33	
Ilustración 29. Cambiar la versión de XDXtools a la 3.62.0.06.....	33	
Ilustración 30. Estructura de un programa típico de un microcontrolador con superloop.	34	
Ilustración 31. Comparación de un programa de un microcontrolador sin SO (Bare- Metal) con un RTOS.....	35	
Ilustración 32. Ejemplo de ejecución de varias tareas con Preemptive Scheduling. .	35	
Ilustración 33. Estados de una tarea.....	36	
Ilustración 34. Tipos de hilos (threads) en TI-RTOS.	37	
Ilustración 35. Estructura de un programa en CCS.	39	
Ilustración 36. Comunicación entre las dos imágenes de un programa a través de ICall.....	40	
Ilustración 37. Logo Bluetooth versión 4.0.	40	
Ilustración 38. Arquitectura de la pila del protocolo BLE.	41	
Ilustración 39. Canales RF de BLE.	42	

Ilustración 40. Intervalos de tiempo de emisión y espera de paquetes de respuesta.	43
Ilustración 41. Secuencia de eventos para iniciar una conexión.	44
Ilustración 42. Comunicación entre dispositivos maestro y esclavo.	45
Ilustración 43. Diagrama de estados del nivel Link Layer.	45
Ilustración 44. Formatos de los paquetes de datos.	46
Ilustración 45. Formato de paquete de datos para transmisión de información.	47
Ilustración 46. Operaciones de solicitud y respuesta.	49
Ilustración 47. Operación de tipo orden.	49
Ilustración 48. Operaciones de indicación y confirmación.	49
Ilustración 49. Operación de notificación.	50
Ilustración 50. Propiedades de un atributo.	50
Ilustración 51. Perfil GATT.	51
Ilustración 52. Elementos de una característica.	52
Ilustración 53. Ejemplo de una tabla de datos de un atributo.	53
Ilustración 54. Pila del protocolo BLE con perfiles.	55
Ilustración 55. Logo del bus I2C.	56
Ilustración 56. Ejemplo de implementación de comunicación entre dispositivos mediante I2C.	56
Ilustración 57. Configuración "open drain".	57
Ilustración 58. Estructura básica de los mensajes en la comunicación I2C.	57
Ilustración 59. Esquema del módulo I2C del microcontrolador.	59
Ilustración 60. Esquema para obtener la señal de reloj del módulo I2C y del reloj SCL.	60
Ilustración 61. Configuración de la señal SCL.	60
Ilustración 62. Fórmula para obtener la señal de reloj SCL.	61
Ilustración 63. Valores de "d" en función de IPSC.	61
Ilustración 64. Validación de datos.	61
Ilustración 65. Condiciones de inicio y parada.	61
Ilustración 66. Estructura del mensaje si la dirección del esclavo son 7 bits.	62
Ilustración 67. Estructura del mensaje si la dirección del esclavo son 10 bits.	62
Ilustración 68. Estructura del mensaje en formato libre.	62
Ilustración 69. Estructura del mensaje en repetición de la condición de inicio.	63
Ilustración 70. Sincronización de la señal de reloj SCL.	63
Ilustración 71. Representación teórica de un ECG.	64
Ilustración 72. Representación del complejo QRS.	65
Ilustración 73. Opciones de colocación de los electrodos periféricos según la American Heart Association (AHA).	66
Ilustración 74. Diagrama de bloques de la fase de pre-procesado del algoritmo.	66
Ilustración 75. Sensor electrocardiográfico.	67
Ilustración 76. Electrodo.	68
Ilustración 77. Colocación errónea de los electrodos.	68
Ilustración 78. Colocación correcta de los electrodos.	69
Ilustración 79. Parches desechables para electrodos.	69
Ilustración 80. Gráfica de los valores detectados por el sensor.	70

Ilustración 81. Árbol del proyecto "F28069M_TX_P1".....	71
Ilustración 82. Código de la función "Config_ADC()".....	72
Ilustración 83. Código de la función "adc_isr()".....	73
Ilustración 84. Código de la función "I2C_Init()".....	73
Ilustración 85. Código de la función "i2c_tx_isr()".....	74
Ilustración 86. Código de la función "ePWM2_Timer()".....	75
Ilustración 87. Código de la función "epw2_timer_isr()".....	75
Ilustración 88. Código del bucle for infinito del "main()".....	76
Ilustración 89. Fragmento de la función "detect()".....	76
Ilustración 90. Árbol del proyecto "CC250_Peripheral_BLE_P2".....	78
Ilustración 91. Captura de la herramienta generadora de servicios.....	79
Ilustración 92. Constantes definidas en "ConstantesVitales.h".....	79
Ilustración 93. Instrucción que añadir en "ConstantesVitales.c" (I).....	80
Ilustración 94. Instrucción que añadir en "ConstantesVitales.c" (II).....	80
Ilustración 95. Instrucción que añadir en "ConstantesVitales.c" (III).....	80
Ilustración 96. Modificación (I) en "BLE_Peripheral.c".....	80
Ilustración 97. Modificación (II) en "BLE_Peripheral.c".....	81
Ilustración 98. Modificación (II) en "BLE_Peripheral.c".....	81
Ilustración 99. Modificación (III) en "BLE_Peripheral.c".....	81
Ilustración 100. Código ubicado en el fichero "I2C_RX.c".....	81
Ilustración 101. Código de la función "I2C_RX_createTask()".....	82
Ilustración 102. Código de la función "I2C_RX_TaskFxn()" (I).....	82
Ilustración 103. Código de la función "I2C_RX_TaskFxn()" (II).....	83
Ilustración 104. Código de la función "I2C_RX_TaskFxn()" (III).....	83
Ilustración 105. Modificación (I) en "main.c".....	84
Ilustración 106. Modificación (II) en "main.c".....	84
Ilustración 107. Árbol del proyecto "CC2650_Central_BLE_P3".....	85
Ilustración 108. Modificación (I) del fichero "simple_central.c".....	86
Ilustración 109. Modificación (II) del fichero "simple_central.c".....	86
Ilustración 110. Modificación (III) del fichero "simple_central.c".....	86
Ilustración 111. Modificación (IV) del fichero "simple_central.c".....	87
Ilustración 112. Modificación (V) del fichero "simple_central.c".....	87
Ilustración 113. Modificación (VI) del fichero "simple_central.c".....	87
Ilustración 114. Fragmento de código de la función "SimpleBLECentral_init()".....	87
Ilustración 115. Fragmento de código (I) de la función "SimpleBLECentral_processRoleEvent()".....	88
Ilustración 116. Fragmento de código (II) de la función "SimpleBLECentral_processRoleEvent()".....	88
Ilustración 117. Fragmento de código (I) de la función "SimpleBLECentral_handleKeys()".....	88
Ilustración 118. Fragmento de código (II) de la función "SimpleBLECentral_handleKeys()".....	89
Ilustración 119. Fragmento de código (III) de la función "SimpleBLECentral_handleKeys()".....	89

Ilustración 120. Fragmento de código (IV) de la función "SimpleBLECentral_handleKeys()".	89
Ilustración 121. Fragmento de código de la función "SimpleBLECentral_processGATTMsg()".	90
Ilustración 122. Fragmento de código (I) de la función "SimpleBLECentral_processGATTDiscEvent()".	90
Ilustración 123. Fragmento de código (II) de la función "SimpleBLECentral_processGATTDiscEvent()".	91
Ilustración 124. Código localizado en el fichero "I2C_TX.h".	91
Ilustración 125. Código de la función "recible_BLE()" en el fichero "I2C_TX.c".	91
Ilustración 126. Declaración de la variable global "dato_ble".	92
Ilustración 127. Fragmento de código de la función "I2C_TX_TaskFxn()".	92
Ilustración 128. Código del fichero "main.c".	92
Ilustración 129. Fragmento de código de la función "main()".	92
Ilustración 130. Árbol del programa "F28069M_RX_P4".	93
Ilustración 131. Declaraciones de las variables globales y de las funciones auxiliares del programa.	93
Ilustración 132. Código de la función "I2C_Init()".	94
Ilustración 133. Código de la función "i2c_rx_isr()".	95
Ilustración 134. Captura (I) del depurador de CCS.	97
Ilustración 135. Captura (I) del programa Logic 2.	97
Ilustración 136. Captura (II) del depurador de CCS.	98
Ilustración 137. Captura (I) de la app "BLE Scanner".	98
Ilustración 138. Captura (II) de la app "BLE Scanner".	99
Ilustración 139. Captura (III) de la app "BLE Scanner".	99
Ilustración 140. Captura (I) del terminal serie.	100
Ilustración 141. Captura (II) del terminal serie.	100
Ilustración 142. Captura (III) del terminal serie.	100
Ilustración 143. Captura (III) del depurador de CCS.	101
Ilustración 144. Captura (IV) del depurador de CCS.	101
Ilustración 145. Captura (II) del programa Logic 2.	101
Ilustración 146. Foto del montaje final del proyecto.	102
Ilustración 147. Esquema de la resistencia pull-up embebida en la LaunchPad.	111
Ilustración 148. Frecuencia señal SCL a 100 KHz.	112
Ilustración 149. Frecuencia señal SCL a 400 KHz.	112
Ilustración 150. Especificaciones del protocolo I2C. Tabla de Texas Instruments.	114
Ilustración 151. Frecuencia señal SCL a 100 KHz.	115
Ilustración 152. Frecuencia señal SCL a 400 KHz.	115

1. Introducción y objetivos

La intencionalidad de este trabajo es desarrollar la infraestructura necesaria para comunicar de forma inalámbrica (con el protocolo Bluetooth Low Energy) una plataforma de rehabilitación neuromotora robotizada llamada RobHand, la cual incorpora un microcontrolador LAUNCHXL-F28069M de Texas Instruments, a través de un módulo Bluetooth incorporado en otro microcontrolador de Texas Instruments, el LAUNCHXL-CC2650.

Como primera fase de desarrollo, se plantea la transmisión del ritmo cardiaco por Bluetooth: un microcontrolador F28069M recibe las señales electrocardiográficas registradas mediante un módulo de Sparkfun basado en el AD8232, calcula el ritmo cardiaco mediante el algoritmo de Pan-Tomkins y lo transmite al módulo CC2650 mediante comunicación I2C para que este envíe dicha información mediante Bluetooth a otro módulo CC2650, el cual transmitirá el valor del ritmo cardiaco a otro F28069M.

Tal y como ilustra la Ilustración 1, el desarrollo de este proyecto consiste en la programación de 4 microcontroladores de Texas Instruments, los cuales implementan las siguientes funciones y características:

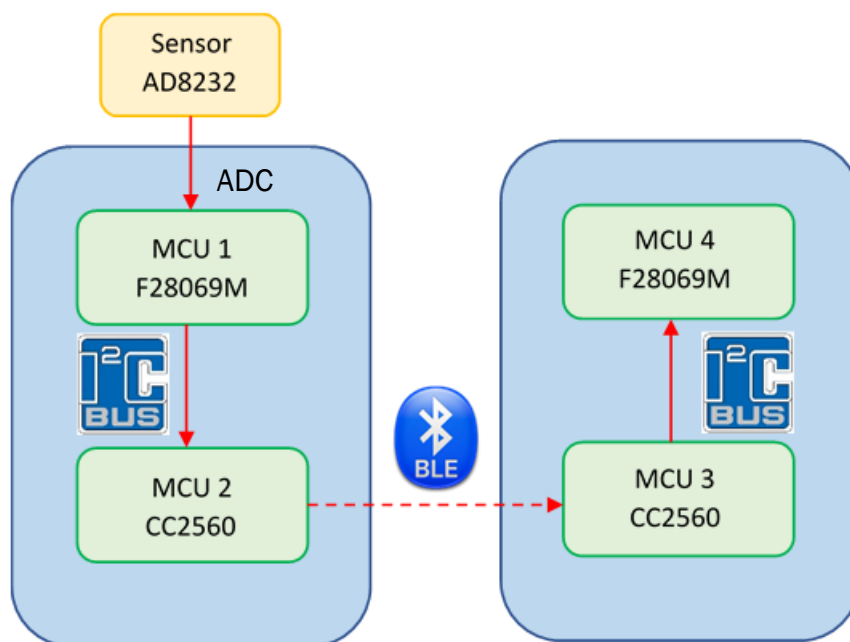


Ilustración 1. Esquema del proyecto.

- ❖ **MCU 1:** Primer módulo F28069M. Este microcontrolador se encarga de recibir los datos analógicos recogidos por el sensor de Sparkfun, aplicarles el algoritmo de Pan-Tomkins para hallar el valor del ritmo cardiaco y enviar dicho dato a través del bus I2C (funcionando como esclavo) al segundo microcontrolador.

- ❖ MCU 2: Primer módulo CC2650, el cual recibe el valor BPM (“beats per minute” o ritmo cardiaco) por el bus I2C, funcionando en modo maestro, para a continuación enviarlo mediante el protocolo BLE al segundo microcontrolador CC2650. Este primer microcontrolador deberá trabajar en el rol de dispositivo periférico o esclavo, implementando el servicio GATT ‘Heart Rate Service’ (explicado más adelante), el cual incorpora la característica que nos interesa, la ‘Heart Rate Measurement’, cuyo atributo podrá únicamente ser leído por un dispositivo trabajando en el rol de ‘central’ una vez realizada la conexión Bluetooth.
- ❖ MCU 3: Segundo módulo CC2650. Este microcontrolador trabajará en el rol de maestro en BLE con el objetivo de conectarse al primer CC2650 y leer la característica del ritmo cardiaco cuando detecte una señal de entrada por parte del usuario. Después, funcionando como dispositivo maestro en el bus I2C, se transmitirá el valor del ritmo cardiaco al último microcontrolador del proyecto.
- ❖ MCU 4: Segundo módulo F28069M. El último microcontrolador del proyecto recibe por el bus I2C, trabajando en modo esclavo, el valor de BPM calculado por el primer F28069M conectado al sensor electrocardiográfico.

El proyecto se ha desarrollado en el ITAP [1] (Instituto de las Tecnologías Avanzadas de la Producción), que es un instituto universitario de investigación (IUI) de la Universidad de Valladolid, cuya sede se sitúa en la Escuela de Ingenierías Industriales, en la sede Paseo del Cauce, 59.

Está formado por más de 32 profesores de múltiples áreas de conocimiento. La mayor parte de ellos pertenece a la Escuela de Ingenierías Industriales, si bien hay miembros de la Facultad de Medicina y de la Escuela Universitaria de Ingenierías Agrarias. Asimismo, colaboran múltiples profesionales, destacando investigadores de la Fundación Cartif y médicos del Hospital Clínico Universitario de Valladolid.

ITAP abarca proyectos de muchos ámbitos dentro de la ingeniería industrial; sin embargo, se pueden definir tres grandes áreas de investigación:

- ❖ Modelado, simulación y control: poder simular el comportamiento de un sistema proporciona grandes ventajas a la hora de realizar un diseño y un control óptimo sin necesidad de prototipos o pruebas previas, que conlleven gastos innecesarios.
- ❖ Robótica y visión artificial: surge de la necesidad de desarrollar robots que ayuden al ser humano a realizar diferentes tareas que requieran mayor precisión, que tengan mayor dificultad y que, además, proporcionen mayor comodidad y seguridad.
- ❖ Tecnologías ambientales: enfocado a la reducción de contaminantes y a la utilización de energías renovables en varios aspectos.

El instituto también promueve la publicación de artículos científicos, patentes, proyectos de investigación competitivos y contratos con empresas e institución, así como la formación especialidad de investigadores y tecnólogos.

Además de los proyectos de investigación propios de la entidad, los miembros del instituto también dirigen tesis y se encargan de las tutorías de trabajador de fin de grado y de máster. Estos trabajos en muchos casos forman parte de las investigaciones propias del ITAP y contribuyen a las mismas.

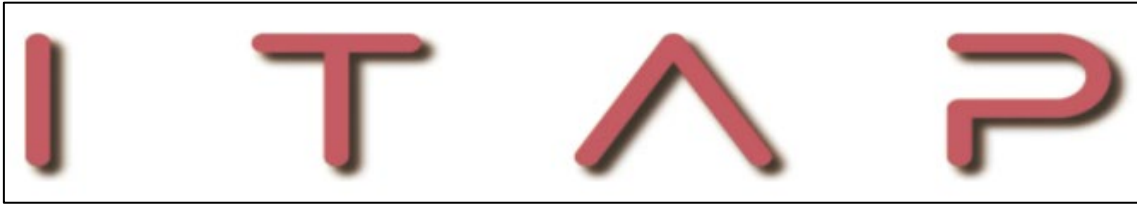


Ilustración 2. Logo de ITAP.

2.Estado del arte

2.1. Introducción

Gracias a la miniaturización y portabilidad de los dispositivos electrónicos de hoy en día, se está generalizando el desarrollo de una amplia gama de productos y aplicaciones relacionados con la asistencia sanitaria y la monitorización continua y en tiempo real de biomarcadores críticos para la salud de las personas. Estos dispositivos poseen sensores capaces de realizar diagnósticos médicos para registrar el estado de salud de una persona a lo largo del tiempo o para anticipar lo antes posible un posible fallo en el sistema fisiológico y recibir asistencia médica de manera casi inmediata [2].

A estos artilugios se los llama “wearables”, y utilizan tecnologías de comunicación inalámbrica tales como el Wi-Fi o el Bluetooth. Algunos de los dispositivos inalámbricos encontrados en la bibliografía para medir señales fisiológicas, como el ritmo cardiaco o las señales electromiográficas (EMG) de los músculos son los siguientes:

- ❖ Trazein (Ilustración 3) [3]: Dispositivo “wearable” de bajo coste desarrollado por ITAP que integra sensores para la lectura y registro de las señales de ECG y GSR (Galvanic Skin Response), que posteriormente se comunica vía Wi-Fi con un servicio basado en la nube para realizar un procesamiento de datos online y adaptar dinámicamente las tareas de rehabilitación de los miembros superiores.

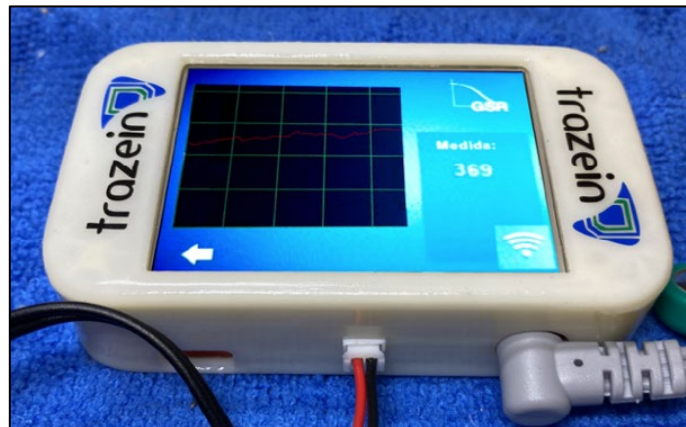


Ilustración 3. Foto del dispositivo Trazein.

- ❖ Un ASIC (Application Specific Integrated Circuit) de adquisición de datos biomédicos multicanal con cancelación de la frecuencia de red [4]. El prototipo de la Ilustración 4 permite la adquisición de datos ECG/EEG/EMG.

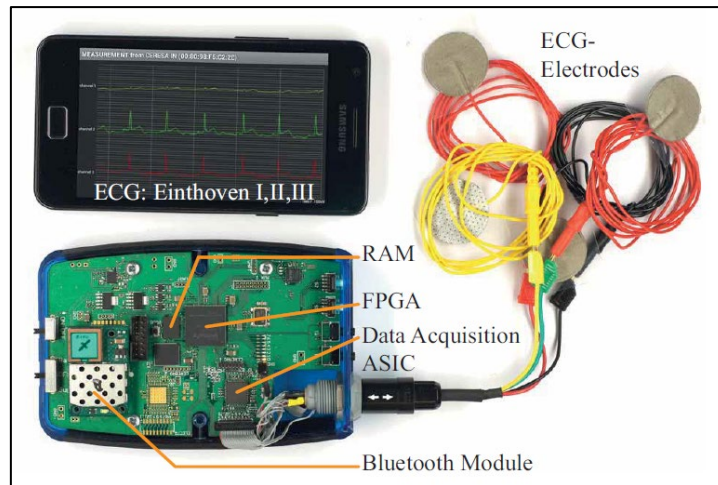


Ilustración 4. Foto del prototipo detector de ECG/EEG/EMG.

- ❖ Sistema portátil multiparamétrico de atención sanitaria portátil (Ilustración 5) [5]. Este dispositivo está diseñado modularmente y utiliza un MCU PSoC4, un ASIC para la adquisición y medición de ECG, SPO2 y NIBP, y a través del módulo Bluetooth integrado puede transferir los resultados en tiempo real o ser visualizados a través de una pantalla LCD.



Ilustración 5. Foto del sistema de cuidado de la salud portátil desarrollado.

Este tipo de dispositivos también pueden ser utilizados para el control de plataformas robóticas. En este trabajo, se realiza una primera fase de desarrollo de una infraestructura de software de comunicación inalámbrica para su implementación en la plataforma de rehabilitación de mano RobHand.

La plataforma RobHand [6] (Robot para la Rehabilitación de la Mano) realiza terapias bilaterales asistidas por EMG, las cuales permiten entrenamiento activo-asistencial (Ilustración 7). La plataforma de rehabilitación robótica incorpora una solución integrada de EMG hecha a medida para el reconocimiento de gestos en tiempo real cuya salida se utiliza en el controlador bio-cooperativo del exoesqueleto (Ilustración 6). Las señales EMG registradas se procesan creando dos vías de biorretroalimentación (una fuente de fuerza y otra visual) para ayudar al usuario a tomar conciencia y confianza en el control voluntario del sistema mientras realiza tareas de rehabilitación.

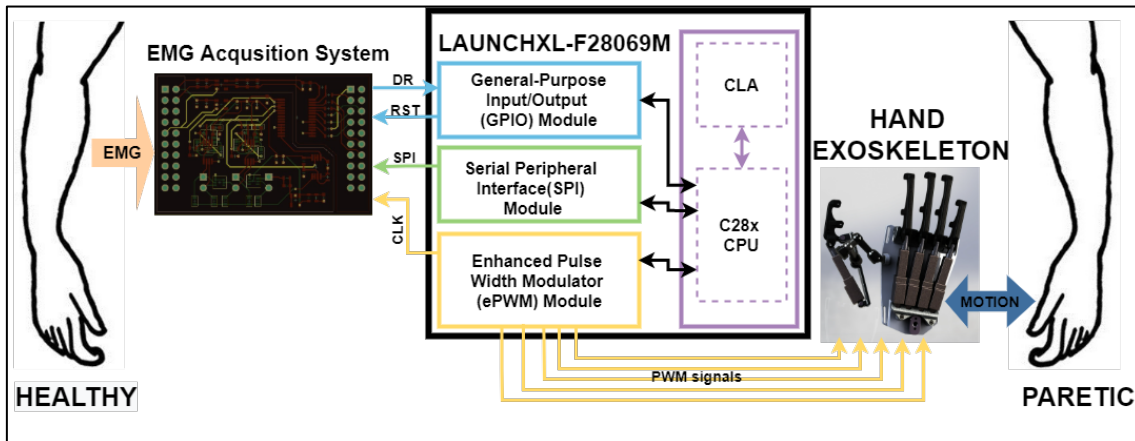


Ilustración 6. Esquema de funcionamiento de RobHand.

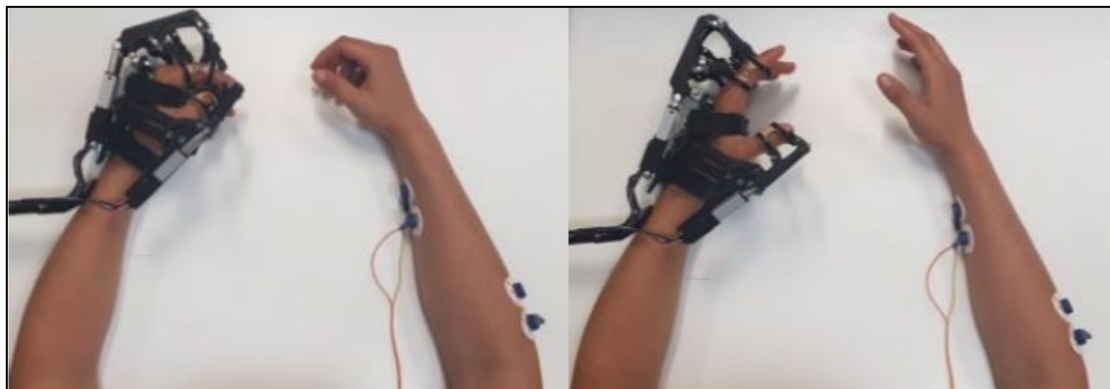


Ilustración 7. Un usuario realizando una terapia bilateral asistida por EMG con la plataforma robótica RobHand.

2.2. LAUNCHXL-F28069M

La placa de desarrollo LAUNCHXL-F28069M (Ilustración 8 e Ilustración 10) [7] desarrollada por Texas Instruments pertenece a la familia de procesadores C2000 [8] e integra el microcontrolador TMS320F28069M [9]. Es una herramienta de desarrollo y evaluación de aplicaciones electrónicas, estandarizada y de bajo costo (P.V.P. de 24,99\$). Además, es compatible con multitud de “BoosterPacks” o módulos de expansión de Texas Instruments y ofrece la herramienta de depuración JTAG, la cual proporciona una interfaz de comunicación directa con un ordenador para facilitar la programación, la depuración y la evaluación del código desarrollado [10] [11] [12].

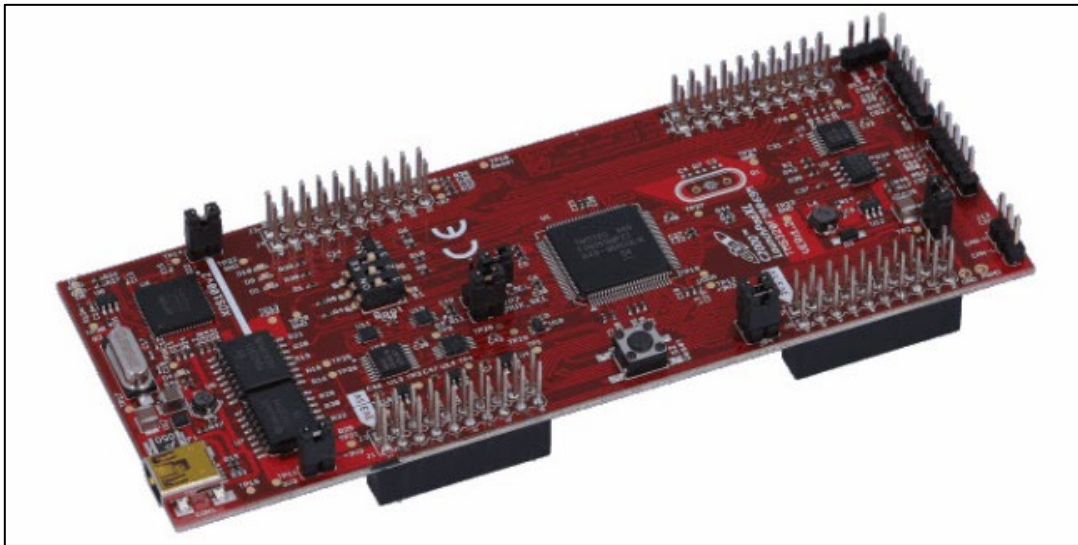


Ilustración 8. Microcontrolador LAUNCHXL-F28069M.

Todos los manuales técnicos y el resto de la documentación se encuentran ubicados en la carpeta adjunta a este documento llamada “Documentación/LaunchPad F28069M”.

CARACTERÍSTICAS TÉCNICAS

- ❖ CPU: TMS320F28069 de 32 bits a 90 MHz. Arquitectura Harvard modificada (Ilustración 9). Programable en C/C++ y ensamblador.
- ❖ CLA (Control Law Accelerator): Acelerador de hardware de coma flotante de 32 bits totalmente programable e independiente de la CPU principal, diseñado para cálculos matemáticos intensos. Se ejecuta en paralelo con la CPU C28, duplicando el rendimiento computacional. Además, está diseñado para responder rápidamente a interrupciones y tiene acceso a todos los periféricos del microcontrolador.

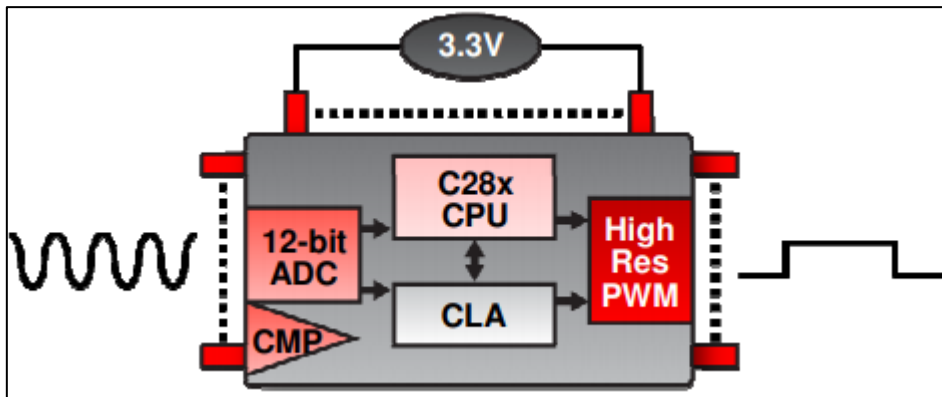


Ilustración 9. Funcionamiento de los núcleos C28x y CLA dentro de la CPU

- ❖ FPU (Unidad de Punto Flotante): Operaciones en coma flotante de forma nativa y precisión única.
- ❖ Memoria: 100KB de RAM, 256KB de Flash y 2KB de ROM.
- ❖ 6 DMA (Canales de acceso directo a memoria)
- ❖ Soporte JTAG para analizar, emular y depurar código en tiempo real vía hardware.
- ❖ 8 módulos ePWM (“enhanced” PWM): 16 canales PWM en total (8 son HRPWM) con temporizadores de 16 bits independientes para cada módulo.
- ❖ 16 canales de conversores ADC de 12 bits de resolución (0 a 3,3V) con muestreo y retención doble.
- ❖ PIE (Peripheral Interrupt Expansion): Bloque que soporta todo tipo de interrupciones producidas por los módulos periféricos del microcontrolador.

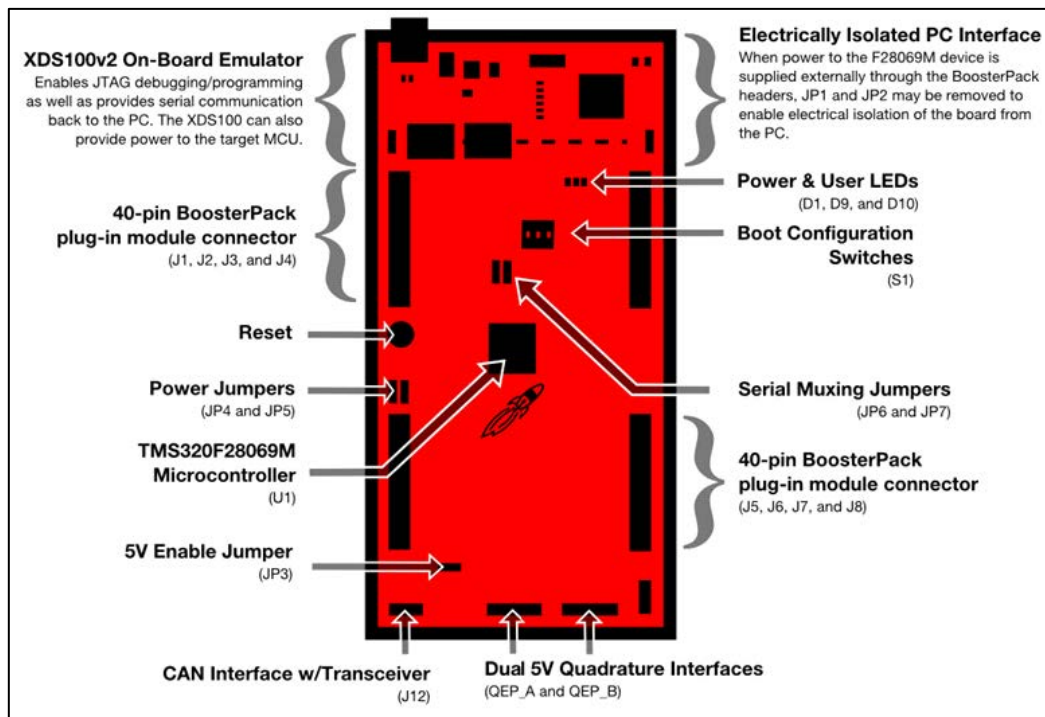


Ilustración 10. Esquema de puertos y funcionalidades del microcontrolador.

- ❖ 80 pines GPIO multiplexados que ofrecen diferentes funciones.
- ❖ 3 temporizadores de 32 bits.
- ❖ Sensor de temperatura.
- ❖ Método de seguridad “Key and Lock” de 128 bits para proteger el acceso a los bloques de memoria e impedir la ingeniería inversa del firmware.
- ❖ Módulos de comunicación serie:
 - 2 módulos SCI (UART).
 - 2 módulos SPI.
 - 1 módulo I2C.
 - 1 módulo con interfaz eCAN con un transmisor-receptor integrado.
 - 1 USB 2.0.
- ❖ 2 LEDs integrados programables.
- ❖ Botón RESET.
- ❖ 3 interruptores de selección de arranque (boot).
- ❖ Biblioteca InstaSPIN cargada en la ROM.
- ❖ 2 interfaces para encoders de 5V.
- ❖ Modo de funcionamiento de bajo consumo.
- ❖ Interfaz de comunicación con un ordenador a través de XDS100v2 USB aislada galvánicamente.

SOFTWARE DE DESARROLLO

El LaunchPad F28069M es compatible con los IDE Code Composer Studio [13] y Energía [14], aunque durante el desarrollo de estas prácticas se trabajará únicamente con Code Composer Studio. Este entorno de desarrollo está diseñado por la propia Texas Instruments, por lo que este dispone de un conjunto muy completo de herramientas para desarrollar y optimizar aplicaciones embebidas. Dispone de un compilador de C/C++ optimizado, editor de código, árbol de proyecto y depurador de código, entre otras cosas.

Una vez descargado e instalado, el programa sugiere al usuario la instalación de controlSUITE [15] y C2000WARE [16], los cuales son software añadido al IDE con documentación, bibliotecas, drivers específicos y ejemplos para diferentes microcontroladores de la familia C2000.

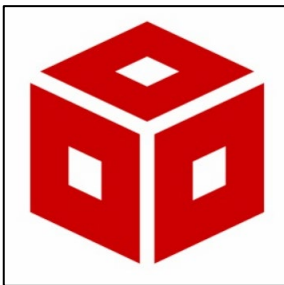


Ilustración 11. Code Composer Studio.

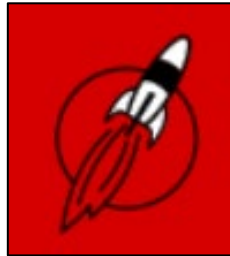


Ilustración 12. Energía.



Ilustración 13. controlSUITE.

Para aprender a utilizar y programar el microcontrolador con el IDE Code Composer Studio antes de desarrollar el programa objetivo, se hará uso de la documentación técnica oficial proporcionada por Texas Instruments, tutoriales y ejemplos incluidos en el software C2000WARE [17] [18] [19] [20].

2.3. LAUNCHXL-CC2650

Esta plataforma de desarrollo (Ilustración 14) [21] manufacturada por Texas Instruments, incorpora un microprocesador CC2650, perteneciente a la familia de procesadores C26xx. Esta herramienta de desarrollo de aplicaciones electrónicas, con un P.V.P. de 29\$, permite desarrollar proyectos con conectividad Bluetooth de baja energía en el ecosistema SimpleLink [22], el cual provee manuales de uso y multitud de ejemplos.

El propósito del microcontrolador es facilitar el desarrollo de software para aplicaciones relacionadas con el Internet de las Cosas o “IoT” utilizando el kit de desarrollo de software (SDK) de Texas Instruments llamado TI BLE-Stack [23], el cual cuenta con la pila de software certificada de la versión 4.1 de Bluetooth. El controlador de BLE (Bluetooth Low Energy) y de la MAC de IEEE 802.15.4 está embebido en una ROM y se ejecuta en parte en un procesador ARM Cortex-M0.

La LaunchPad CC2650 [24] [25] [26] [27] posee un microprocesador ARM Cortex-M3 de 32 bits a 48 MHz de frecuencia, y tiene multitud de módulos periféricos junto a un sensor de baja potencia que se encarga de interactuar con sensores externos almacenando información de manera autónoma a la CPU cuando el resto del sistema está en suspensión o “sleep mode”.

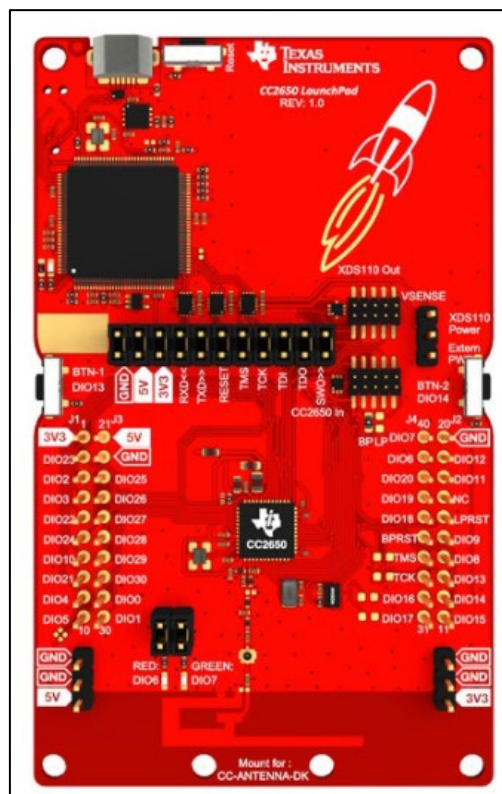


Ilustración 14. Microcontrolador LAUNCHXL-CC2650

El desarrollo y depuración de proyectos para este microcontrolador puede llevarse a cabo en los IDE Code Composer Studio e IAR Embedded Workbench [28] pero no en Energia. También puede cargarse firmware a través de SmartRF Studio [29] y SmartRF Flash Programmer 2 [30].

Al igual que la LaunchPad F28069M, este microcontrolador es compatible con multitud de “BoosterPacks” o módulos de expansión de Texas Instruments y ofrece la herramienta de depuración JTAG, la cual proporciona una interfaz de comunicación directa con un ordenador para facilitar la programación, la depuración y la evaluación del código desarrollado.

Todos los manuales técnicos y el resto de la documentación (Ilustración 15) se encuentran ubicados en la carpeta adjunta a este documento llamada “Documentación/LaunchPad CC2650”, de la misma manera que con la LAUNCHXL-F28069M.

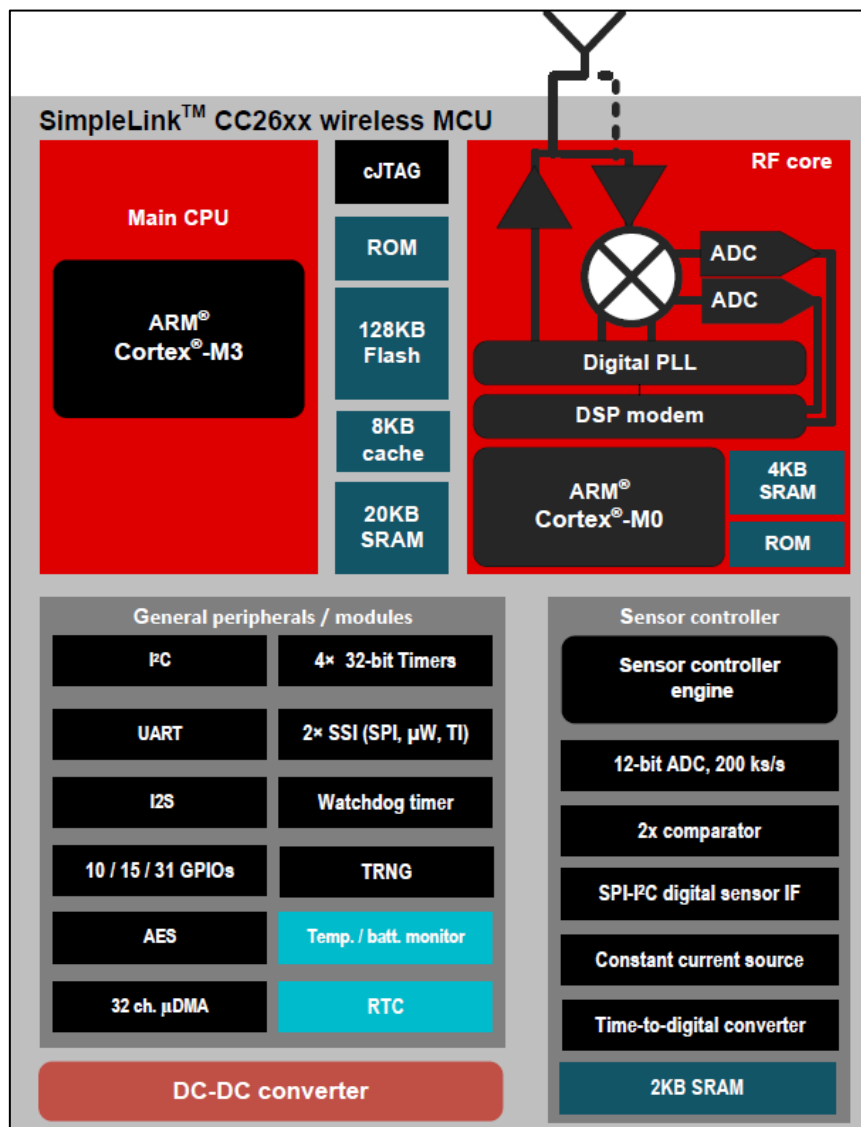


Ilustración 15. Esquema de puertos y funcionalidades del microcontrolador.

CARACTERÍSTICAS TÉCNICAS

- ❖ CPU: ARM Cortex-M3 de 32 bits a 48 MHz.
- ❖ Memoria: 128KB de Flash, 8KB de SRAM (Cache) y 20KB de SRAM.
- ❖ Controlador de sensores de muy bajo consumo: Arquitectura de 16 bits, 2 KB de SRAM y puede ejecutarse de manera autónoma con respecto al resto del sistema.

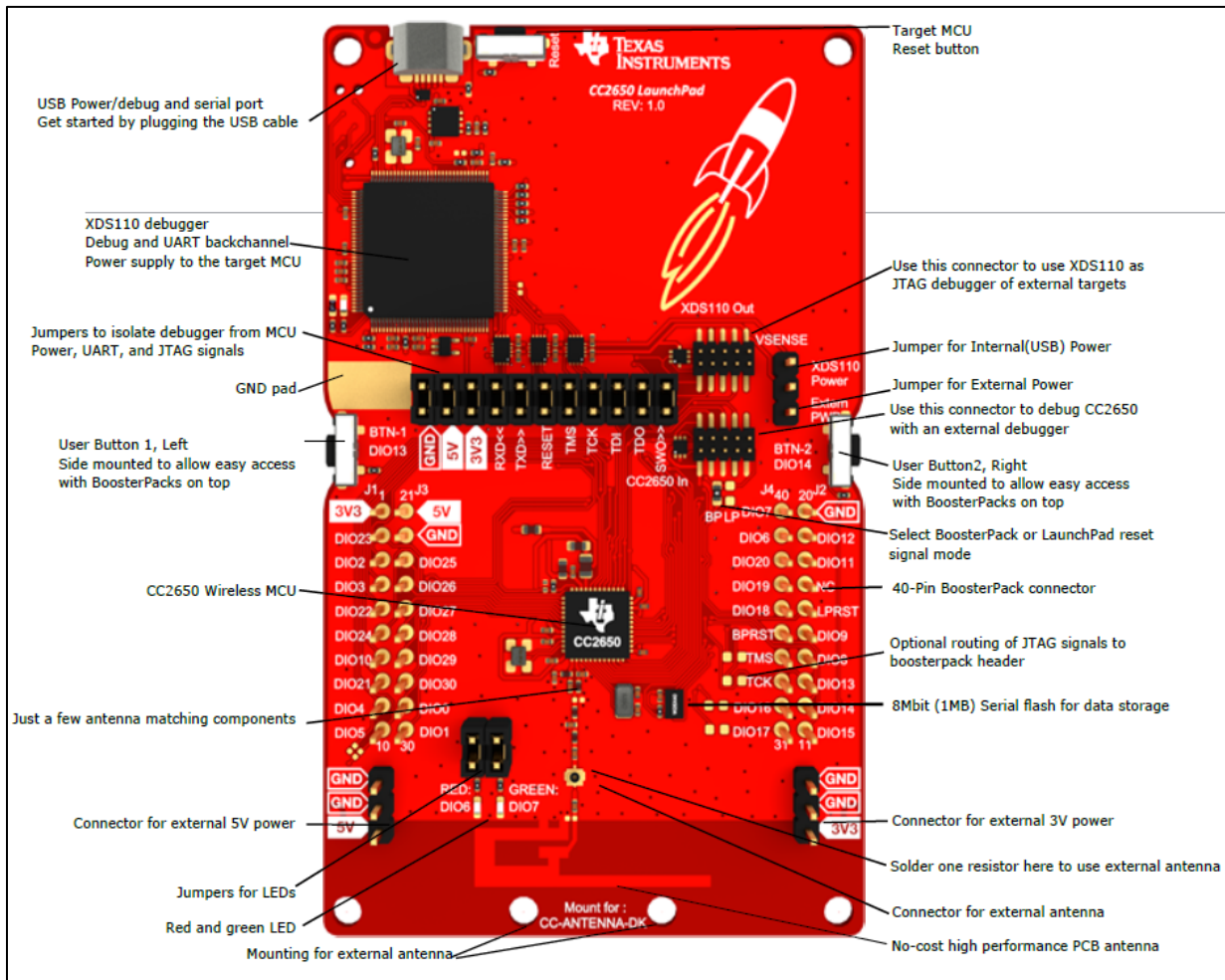


Ilustración 16. Diagrama de bloques del microprocesador.

- ❖ Módulos periféricos:
 - 40 pines GPIO configurables.
 - 4 contadores de tiempo (8 de 16 bits o 4 de 32 bits).
 - 8 canales analógicos multiplexados con ADCs de 12 bits con una tasa de muestreo de 200.000 muestras por segundo.
 - 1 comparador de tiempo continuo.
 - 1 comparador analógico de muy bajo consumo.
 - 1 módulo UART.
 - 2 módulos SSI (para SPI, Microwire o TI).
 - 1 módulo I2C.
 - 1 módulo I2S.
 - 1 reloj de tiempo real.
 - Sensor de temperatura integrado.
 - Generador de números aleatorios (TRNG).
 - Módulo de seguridad AES-128.
- ❖ Conversor DC-DC.
- ❖ Soporte JTAG para analizar, emular y depurar código en tiempo real vía hardware.
- ❖ Módulo RF:
 - Transmisor-receptor de 2,4 GHz compatible con la versión 4.2 de BLE (Bluetooth Low Energy) y la norma IEEE 802.15.4 PHY y MAC.
 - Potencia de salida programable hasta 5 dBm.
 - Sensibilidad de recepción de -97 dBm para BLE y -100 dBm para 802.15.4
- ❖ 2 LEDs integrados programables.
- ❖ 2 botones integrados programables.
- ❖ Botón RESET.
- ❖ Interfaz de comunicación con un ordenador a través de XDS110 USB aislada galvánicamente.

DISPOSICIÓN DE LOS PINES O "PINOUT"

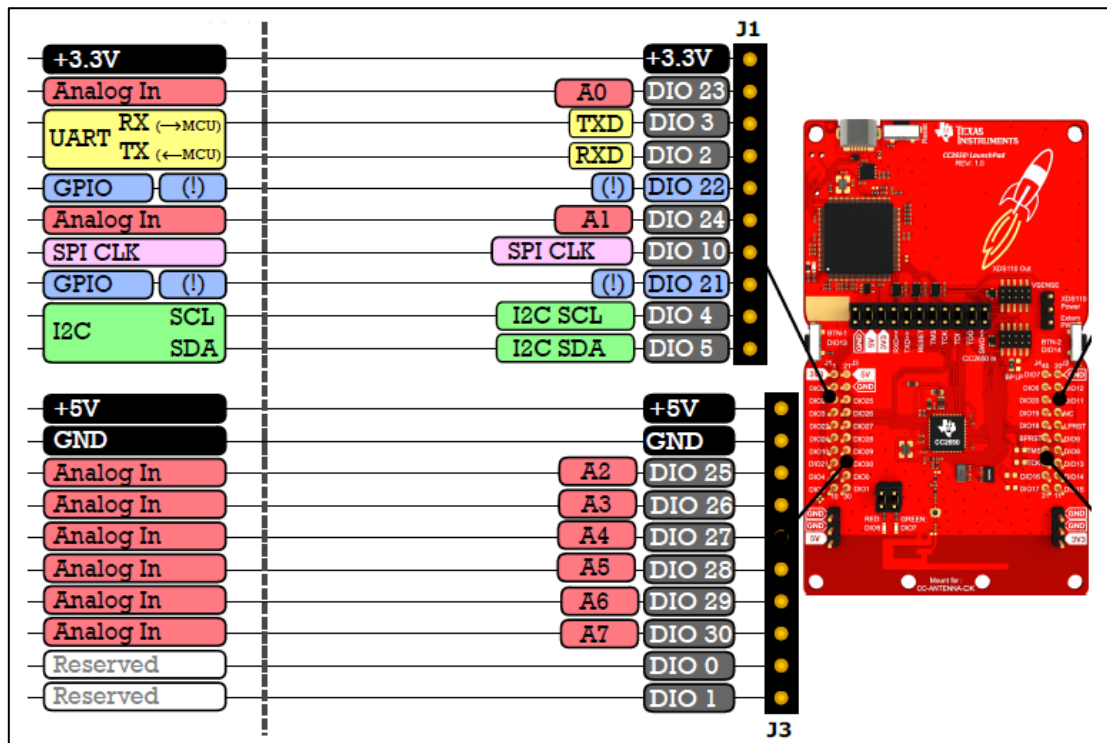


Ilustración 17. "Pinout" del microcontrolador (I).

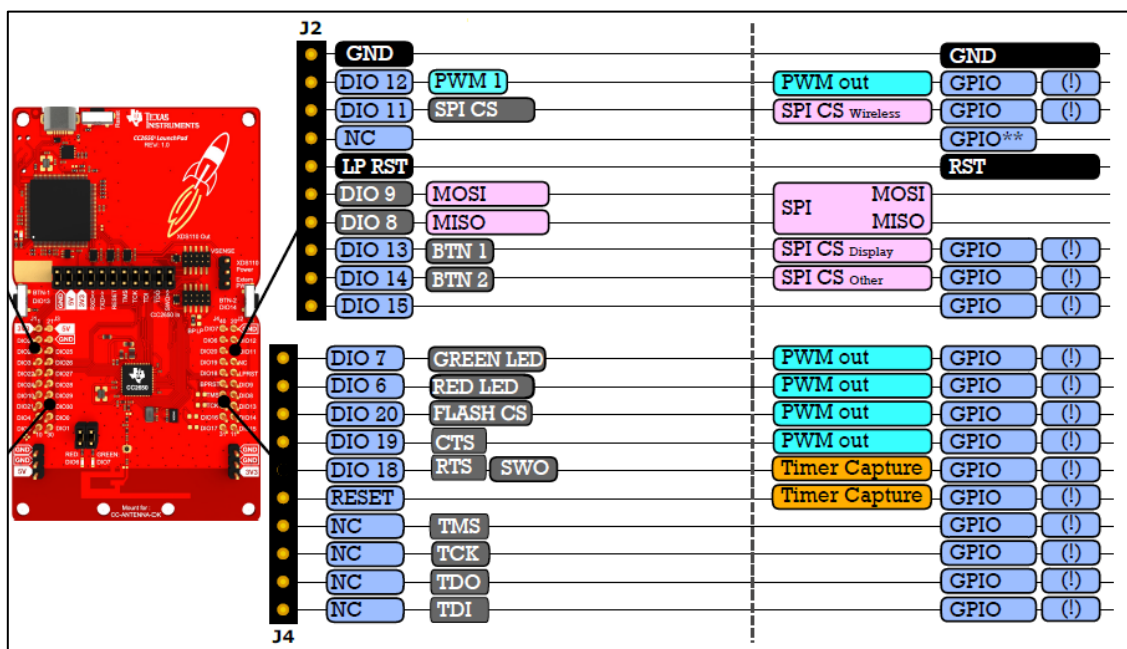


Ilustración 18. "Pinout" del microcontrolador (II).

SOFTWARE DE DESARROLLO

Para trabajar con este microcontrolador se va a seguir utilizando el IDE Code Composer Studio, puesto que ya lo tenemos configurado, pero será necesario descargar e instalar varios paquetes añadidos.

Lo primero que hay que hacer es instalar el controlador de la LaunchPad. Para ello se ejecuta el instalador del IDE Code Composer Studio (versión 10.3), se selecciona “SimpleLink CC13xx and CC26xx Wireless MCUs” (Ilustración 19) y se instala.

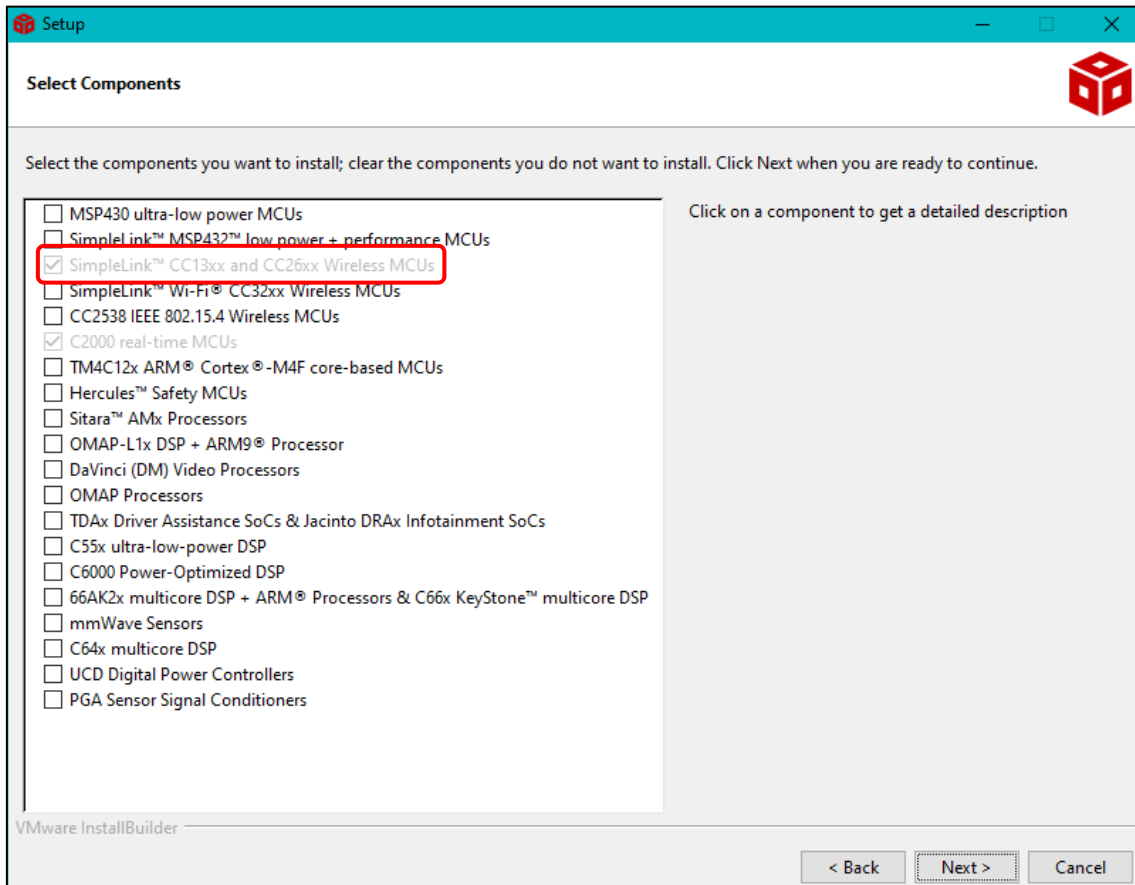


Ilustración 19. Instalador de CCS.

Si seguimos las instrucciones del manual de instalación, este dice que hay que ejecutar el IDE, abrir el explorador de recursos (Ilustración 20), conectar el microcontrolador al PC, esperar a que el programa detecte la placa automáticamente y descargar e instalar el software que nos recomienda, que es “TI-RTOS 2.21.00.06”.

Después de reiniciar el IDE, el manual dice que hay que descargar “SimpleLink SDK BLE Plugin 3.20.00.24” y “SimpleLink Academy 3.10.02.00”. Sin embargo, instalando este software, los proyectos que se desarrollen con el CC2650 dan error, no compilan y no funcionan.

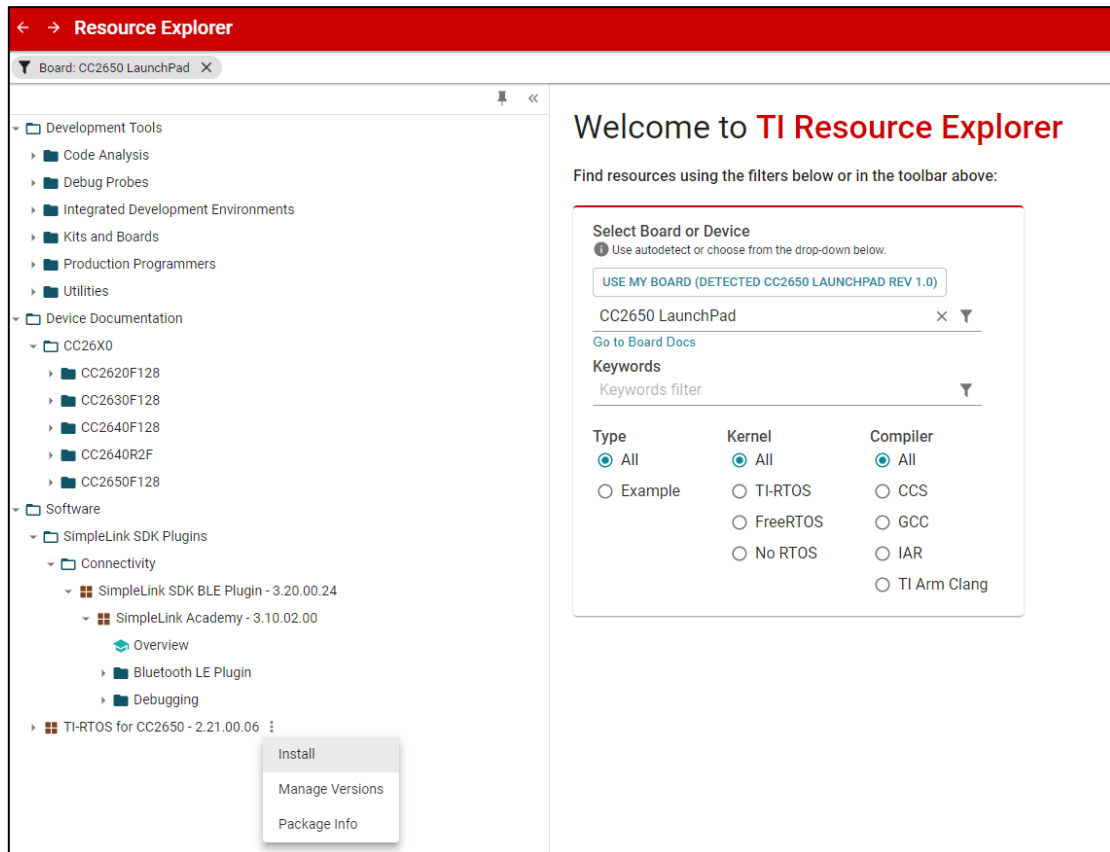


Ilustración 20. Explorador de recursos.

Los pasos que hay que seguir y las herramientas y paquetes de software que hay que descargar e instalar son los siguientes:

❖ **TI-RTOS:**

Buscar en el navegador web TI-RTOS o ir directamente a la web http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/tirtos/index.html [31] y descargar la versión 2.21.01.08. Instalar el paquete en la ruta por defecto.

❖ **BLE SDK TI:**

Buscar en el navegador web TI BLE-SDK o ir directamente a la web <https://www.ti.com/tool/BLE-STACK-ARCHIVE> [32] y descargar la versión 2.02.03.08 (es necesario registrarse con cuenta en Texas Instruments). Instalar el paquete en la ruta por defecto.

❖ Instalación en CSS:

Reiniciar el IDE. En la barra de opciones superior seleccionar “Help” (Ilustración 21) y hacer clic en “Install Code Generation Compiler Tools...”.

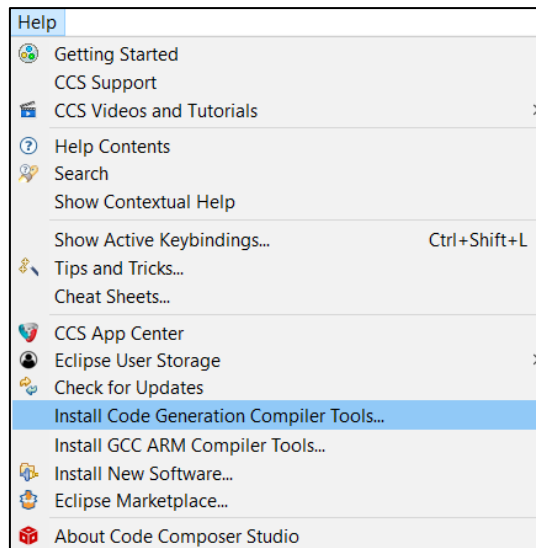


Ilustración 21. Menú desplegable “Help”.

Desplegar la pestaña “Desplegar TI Compiler Updates” y seleccionar “ARM Compiler Tools” (Ilustración 22), versión 5.2.9. Instalar y reiniciar el IDE.

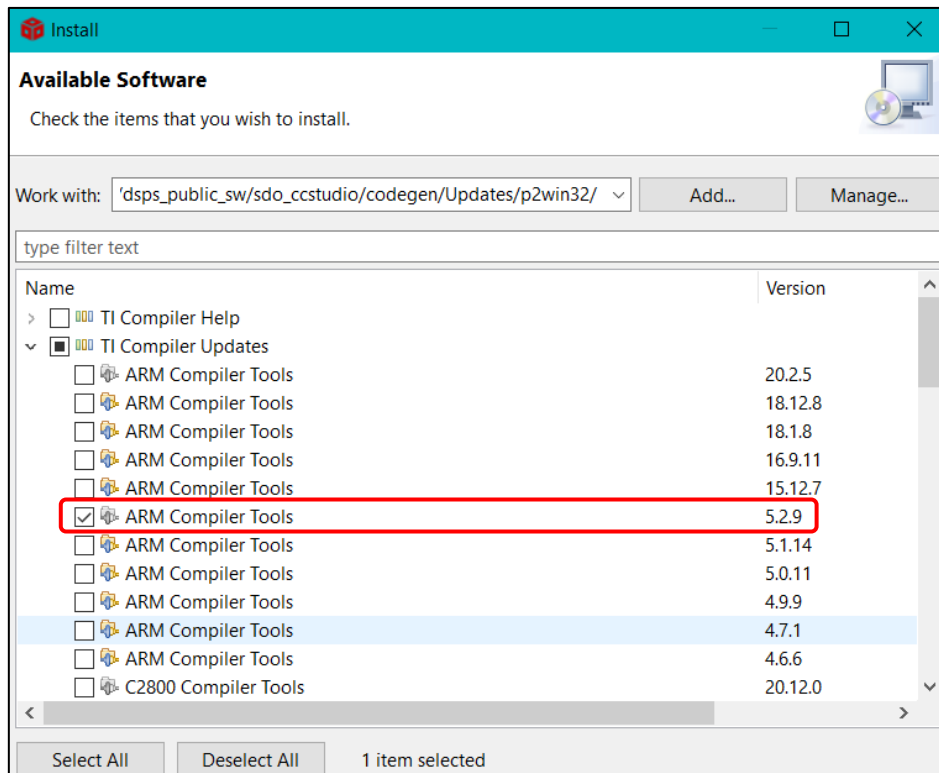


Ilustración 22. Seleccionar “ARM Compiler Tools 5.2.9”.

A continuación, importaremos un proyecto ejemplo haciendo clic en “Project” (Ilustración 23) e “Import CCS Projects...”, ubicado en la dirección “C:\ti\simplelink\ble_sdk_2_02_03_08\examples\cc2650lp\simple_peripheral\ccs”. Seleccionar ambos proyectos en la ventana emergente (Ilustración 24) y finalizar. En este microcontrolador, cada proyecto se divide en dos programas, la pila o *stack* y la aplicación. De esta manera el programa puede actualizarse sin necesidad de recompilar y ejecutar la pila.

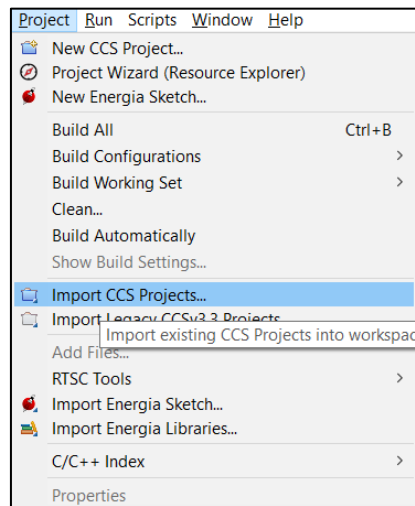


Ilustración 23. Menú desplegable “Project”.

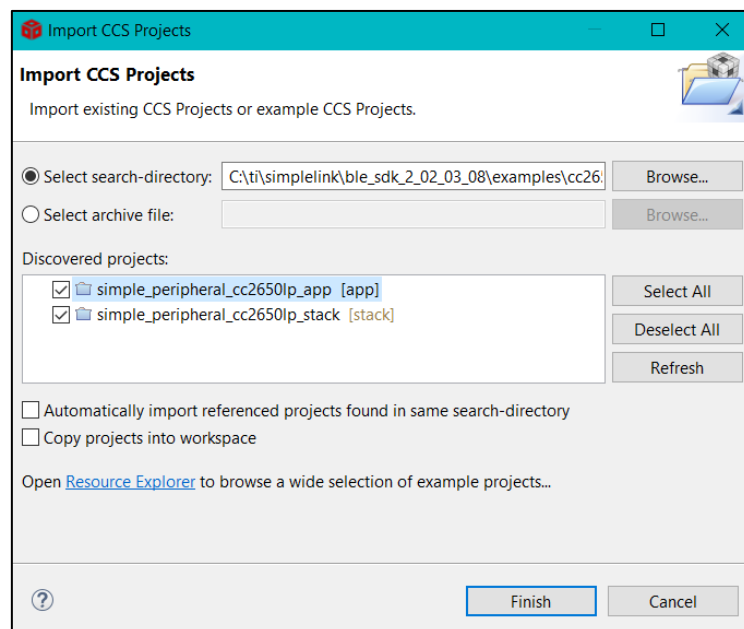


Ilustración 24. Importar los programas “App” y “Stack”.

En el árbol de proyectos, hacer clic derecho sobre el proyecto “simple_peripheral_cc2650lp_stack” y seleccionar “Properties”. En la pestaña “Project” (Ilustración 25) de la opción “CCS General” cambiar la versión del compilador a TI v5.2.9.

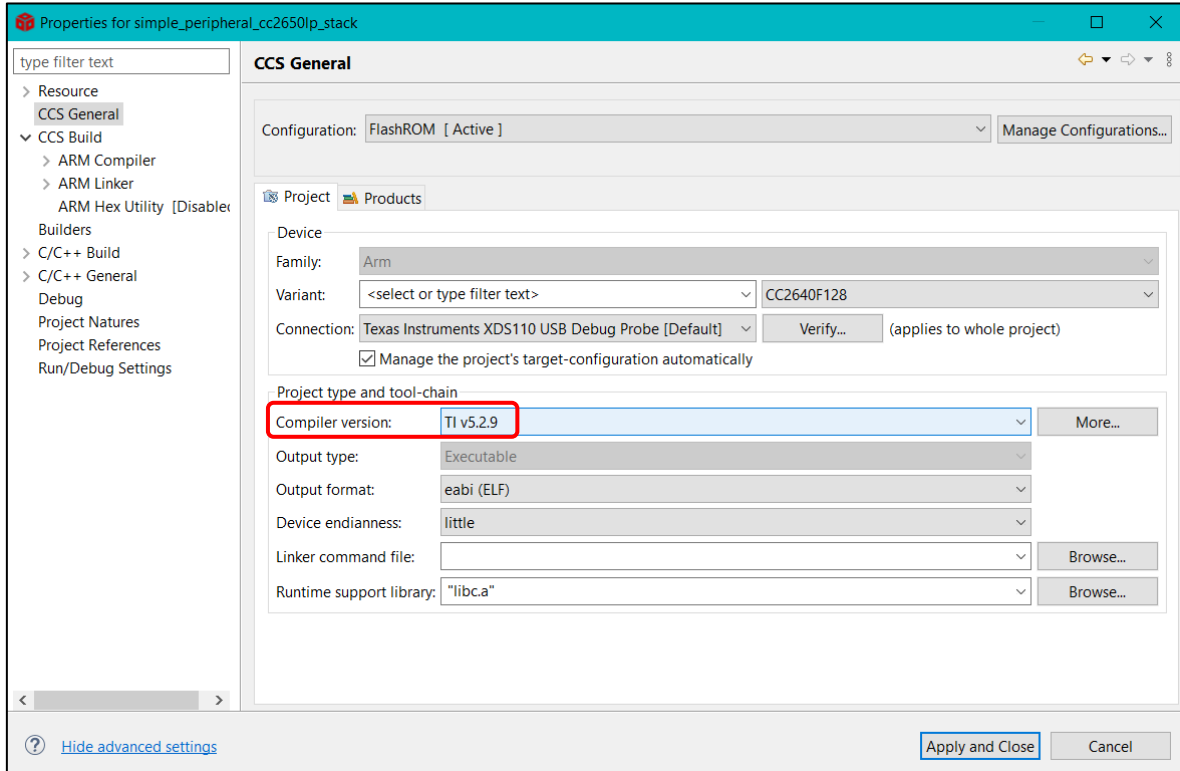


Ilustración 25. Seleccionar el compilador “TI v5.2.9”.

Después, de nuevo en el árbol de proyectos, hacer clic derecho sobre el proyecto “simple_peripheral_cc2650lp_app” y seleccionar “Properties”. En la pestaña “Project” de la opción “CCS General” cambiar la versión del compilador a TI v5.2.9, al igual que con el programa de la pila. Ahora, en la pestaña “Products”, seleccionar “XDCtools [3.62.0.08_core]” y seleccionar “Edit”, como muestra la Ilustración 26.

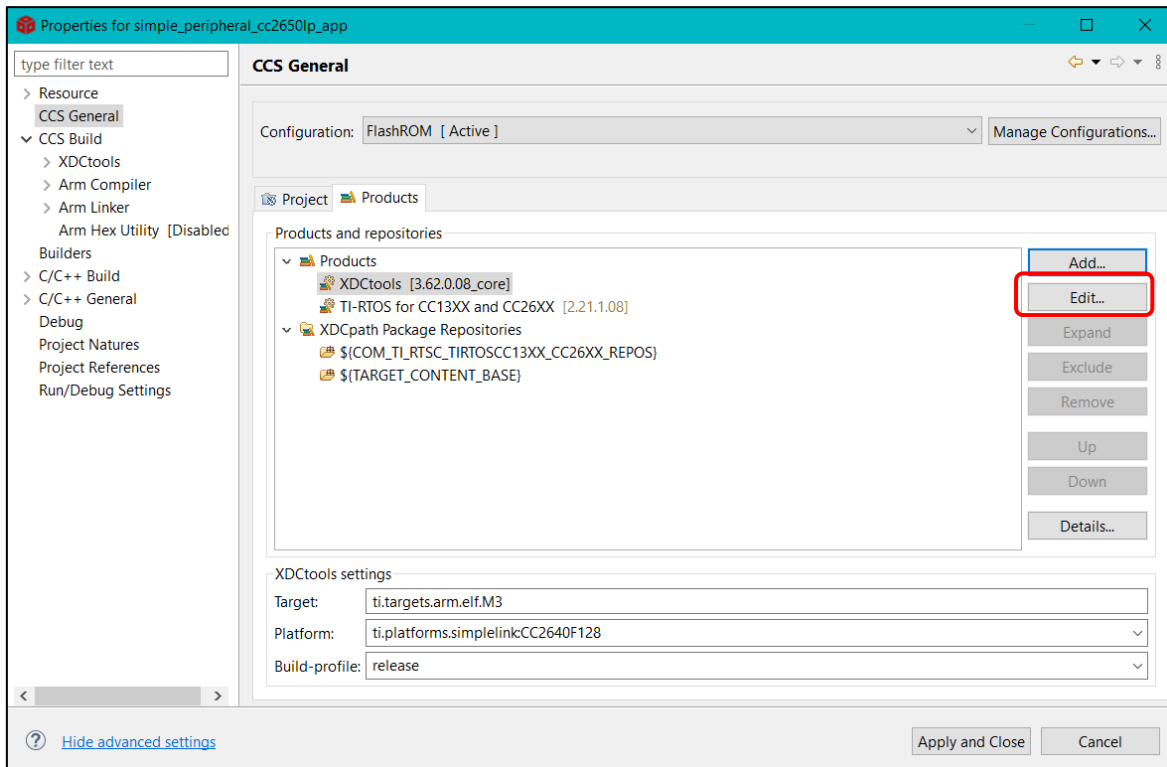


Ilustración 26. Hacer clic en “Edit” en la ventana de propiedades del proyecto.

En la ventana emergente, seleccionar “Preferences” (Ilustración 27).

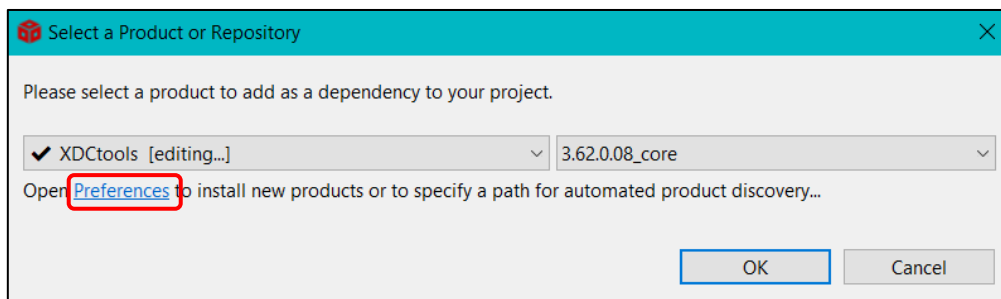


Ilustración 27. Seleccionar “Preferences” en la ventana desplegable.

Aparecerá otra ventana emergente y en su parte derecha se selecciona la opción “Refresh” (Ilustración 28).

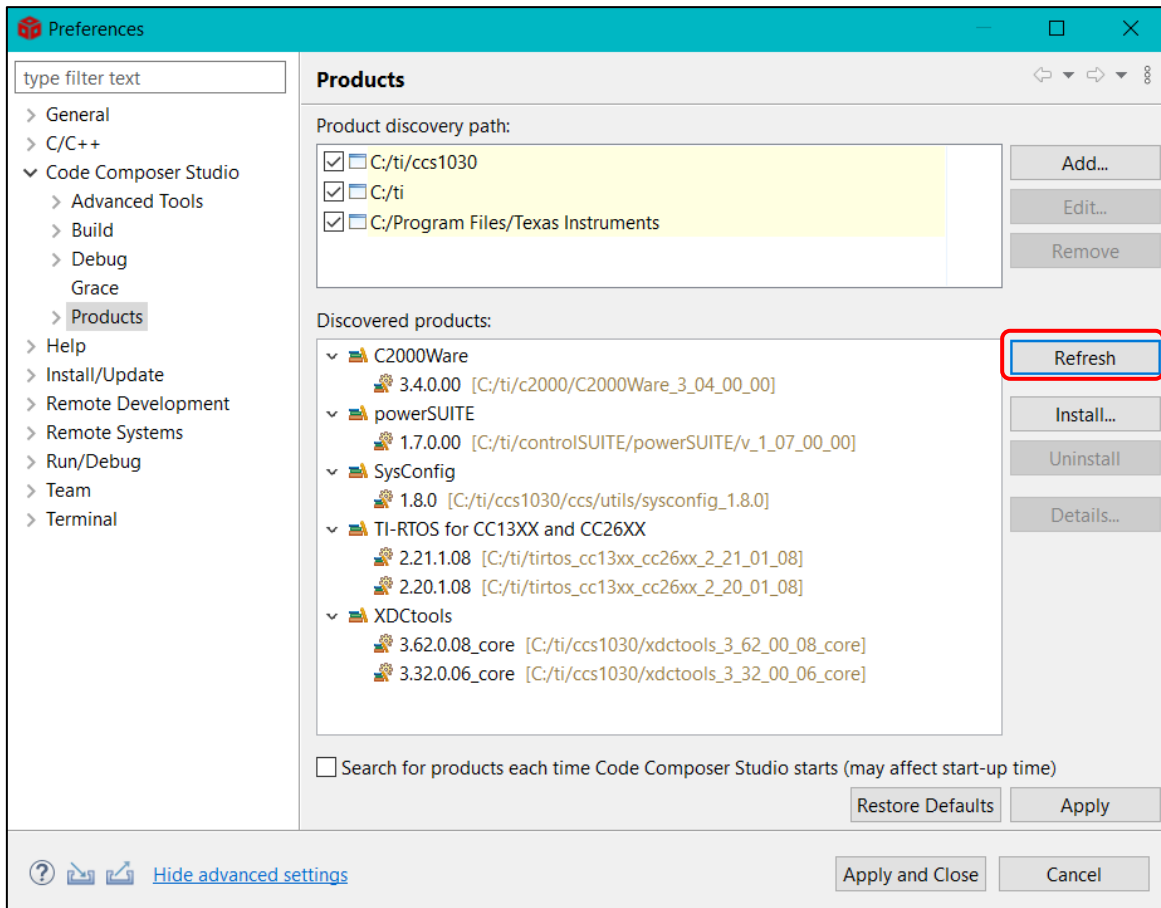


Ilustración 28. Hacer clic en “Refresh” para actualizar los paquetes de software que el IDE tiene listados.

Instalar todo lo que nos sugiere el IDE y reiniciar. Volvemos a la pestaña “Products” en las propiedades del proyecto y seleccionamos “Edit” habiendo elegido XDCtools previamente. En la ventana emergente (Ilustración 29), seleccionar la versión 3.32.0.06_core en el desplegable de la parte derecha. Aceptar. Aplicar cambios y cerrar las propiedades.

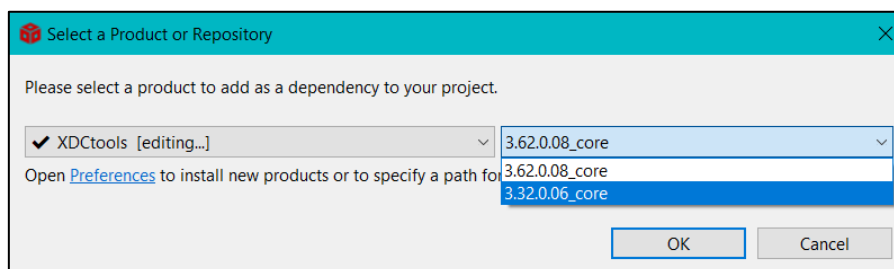


Ilustración 29. Cambiar la versión de XDCtools a la 3.62.0.06.

Por último, para asegurarnos de que todos los paquetes se han instalado correctamente y se ha configurado el IDE adecuadamente, se compilan ambos proyectos y se comprueba que no hay errores.

TI-RTOS

La LAUNCHXL-CC2650 pertenece a la familia de microcontroladores del ecosistema SimpleLink, el cual provee el entorno y las herramientas necesarias para desarrollar proyectos y aplicaciones, siendo en este caso el TI BLE-SDK, que permite y facilita la programación del microcontrolador para aplicaciones con BLE (bluetooth de baja energía). Sin embargo, por debajo de esta pila o *stack* subyace un sistema operativo en tiempo real (RTOS) que gobierna el microcontrolador y se encarga de organizar todos los procesos que ejecuta la CPU.

En un microcontrolador que no tenga que realizar un número significativo de tareas simultáneamente o con un programa con una lógica sencilla, normalmente se ejecuta un único proceso en un bucle infinito (*super-loop*) tras haberse inicializado todas las funciones, variables y módulos necesarios. El único suceso que puede parar el desarrollo de este único proceso o hilo en ejecución es una interrupción (ISR, *Interrupt Service Routine*), la cual es una llamada a la CPU para que realice una acción específica cuando se da una circunstancia determinada por el programador y, a continuación, se vuelve a ejecutar el bucle infinito en el lugar donde se había sido detenido por la interrupción. La siguiente imagen muestra un esquema de la estructura de un programa típico de un microcontrolador. A esta estructura se la conoce como “Bare-Metal” (Ilustración 30).

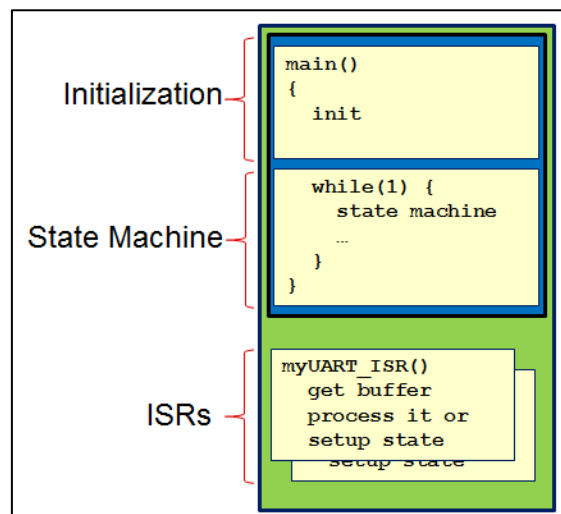


Ilustración 30. Estructura de un programa típico de un microcontrolador con superloop.

Sin embargo, la LaunchPad CC2650 trabaja con un sistema operativo en tiempo real desarrollado por la propia Texas Instruments, el TI-RTOS (antiguo SYS/BIOS) [33] [34], aunque también puede funcionar con FreeRTOS si se desea. Al trabajar con un sistema operativo (SO), el proceso de crear una aplicación que haga uso de varios módulos periféricos y de la pila de la comunicación Bluetooth es más sencillo y llevadero para el programador.

Al utilizar un sistema operativo en tiempo real, el planificador o *scheduler* de tareas del kernel se encarga de asignar tiempo de ejecución en la CPU y organizar todos los hilos o tareas que posea la aplicación.

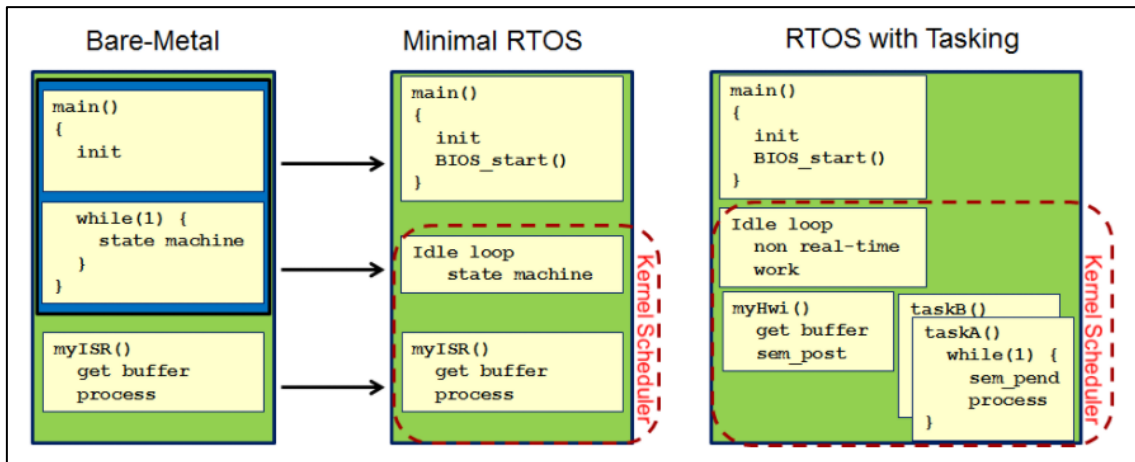


Ilustración 31. Comparación de un programa de un microcontrolador sin SO (Bare-Metal) con un RTOS.

El scheduler de TI-RTOS puede funcionar de dos maneras:

- ❖ Preemptive Scheduling (programación preferente, Ilustración 32): Este es el funcionamiento por defecto de TI-RTOS y el que se va a utilizar en este caso. Con este método de planificación, una tarea que se esté ejecutando continua su funcionamiento hasta que bien termine, bien sea interrumpida por otra tarea con mayor prioridad o bien la propia tarea ceda el uso de la CPU porque pasa a estado de suspensión (la tarea “se duerme”).

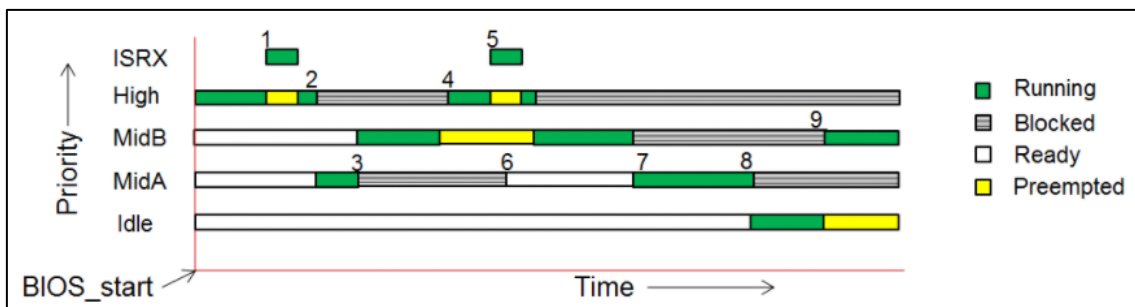


Ilustración 32. Ejemplo de ejecución de varias tareas con Preemptive Scheduling.

- ❖ Time-slice Scheduling (programación de intervalos de tiempo): En este tipo de planificación cada tarea dispone de un determinado intervalo de tiempo para ejecutarse. No obstante, este método de planificación no es aconsejable ni propicio para aplicaciones de tiempo real.

Tipos de hilos o procesos

En TI-RTOS hay 4 tipos de hilos (*threads*, Ilustración 34), con prioridad de mayor a menor según se definan a continuación, y el *scheduler* se encarga de que el proceso activo que tenga mayor prioridad se ejecute:

- ❖ **Hardware Interrupts (Hwi):** Hilo que se ejecuta hasta acabar su trabajo. No bloquea nada y solo pueden ser reemplazados por otro hilo Hwi de mayor prioridad. Todos los hilos Hwi comparten la misma pila (la pila del sistema) y hay un número máximo de Hwi. Otro aspecto a destacar es que el *scheduler* no puede administrar la interrupción, ya que para que tenga la menor latencia posible (teóricamente latencia cero), el kernel es quien se encarga de tratar la interrupción. Sin embargo, esta situación trae la desventaja de que estas interrupciones no pueden llamar a las API para utilizar semáforos, colas u otros métodos de comunicación entre procesos.

Este tipo de procesos se generan por interrupciones a nivel de hardware para hacer que la BIOS genere un guardado, anidado o restauración de contexto. Un Hwi debe ser tratado inmediatamente después de generarse.

- ❖ **Software Interrupts (Swi):** Hilo muy similar al Hwi, las únicas diferencias son que este tipo de interrupciones son disparadas por eventos de software y que tienen menor prioridad que las Hwi. También se ejecutan hasta su finalización una vez iniciados y comparten la misma pila del sistema junto con los Hwi. Los Swi se utilizan para realizar trabajos de Hwi diferidos para minimizar la latencia de las interrupciones.
- ❖ **Tasks (tareas):** Tipo de hilo más común en el SO. En este caso, cada tarea tiene su propia pila donde mantiene su estado y de esta manera se pueden bloquear utilizando semáforos u otros tipos de comunicación entre procesos. Pueden clasificarse por prioridad y no hay un número máximo de tareas permitidas.

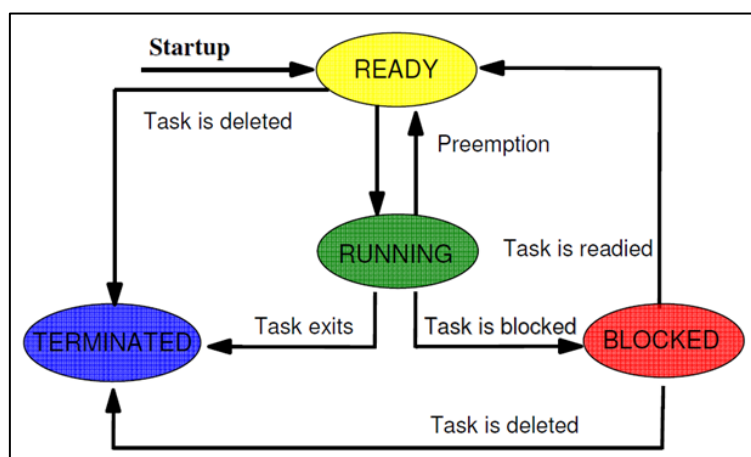


Ilustración 33. Estados de una tarea.

- ❖ Idle (inactivo, ocioso): Hilo por defecto que siempre se ejecuta con el menor nivel de prioridad, 0. Este tipo de hilos realizan tareas en segundo plano y sin importancia crítica para el sistema cuando no hay ningún otro hilo en ejecución. Con el objetivo de consumir la menor energía posible, el microcontrolador puede entrar en modo “bajo consumo” cuando se está ejecutando un proceso tipo Idle.

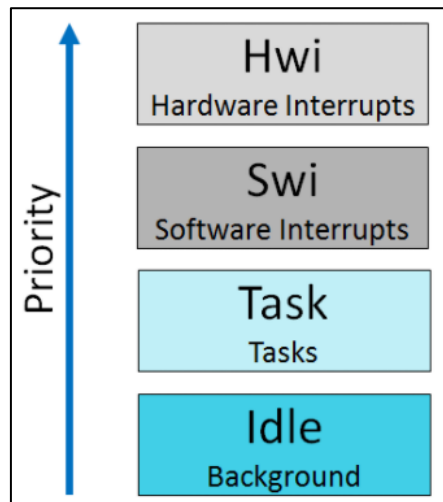


Ilustración 34. Tipos de hilos (threads) en TI-RTOS.

Métodos de comunicación entre procesos

- ❖ Semaphores (semáforos): Regula el acceso a un recurso común entre procesos. Pueden ser utilizados para sincronizar tareas o para exclusión mutua (*mutex*).
- ❖ Mailboxes (buzones): Módulo de paso de mensajes entre procesos.
- ❖ Queues (colas de mensajes): Lista doblemente enlazada sin sincronización.
- ❖ Gates (puertas): Método que protege el acceso simultáneo a estructuras de datos críticas. Es un objeto de exclusión mutua (*mutex*) reentrante.
- ❖ Events (eventos): Módulo que permite la sincronización a través de múltiples eventos.

De todas maneras, TI-RTOS es compatible con el estándar de la IEEE POSIX y el SDK de SimpleLink permite que las APIs de POSIX se ubiquen encima de la estructura de TI-RTOS.

Con el objetivo de comprender el funcionamiento del microprocesador, aprender a trabajar con esta placa que contiene un procesador de una arquitectura diferente a la LaunchPad F28069M y que posee un sistema operativo en tiempo real, se estudiarán los ejemplos de SimpleLink “Project Zero” y “Simple Central”. Además, estos programas servirán como base a la hora de desarrollar los programas que se ejecutarán en microcontroladores CC2650.

Las explicaciones de estos programas se encuentran en la documentación y se ubican en SimpleLink y en el explorador de recursos de Code Composer Studio a su vez.

Antes de comenzar a desarrollar programas en este microcontrolador, es necesario descargar e instalar la versión 2.02.01.18 de BLE-STACK, que además instala la versión 2.20.01.08 de TI-RTOS, para compilar los programas de ejemplo de SimpleLink Academy (versión 1.11.00), la cual también es necesaria descargar e instalar.

- ❖ BLE-STACK: [35].
- ❖ SimpleLink Academy: [36].

PROJECT ZERO

Para estudiar este programa de ejemplo de la versión 1.11 de SimpleLink Academy, será necesario utilizar las guías y documentación correspondiente: [37] [38] [39] [40].

Para realizar la conexión con el microcontrolador será necesario un *smartphone* con Bluetooth 4.1 como mínimo y la aplicación BLE Scanner [41] instalada en el dispositivo. Además, será necesario instalar en el PC un terminal de comunicación en serie. Se recomienda el programa PuTTY [42] o utilizar el terminal de comunicaciones integrado en Code Composer Studio.

SIMPLE CENTRAL

Este programa de prueba de SimpleLink para el microcontrolador LAUNCHXL-CC2650 se encuentra explicado en el soporte “Resource Explorer” de Texas Instruments [43]. Este programa implementa un dispositivo BLE en modo de funcionamiento central o maestro con funcionalidad de cliente GATT. Para trabajar con este ejemplo, se necesita que otro dispositivo trabaje como elemento periférico en la conexión BLE, por lo que se va a utilizar otro módulo CC2650 que tendrá cargado el ejemplo “simple_peripheral” de SimpleLink Academy.

2.4. Protocolo de comunicación BLE

En este apartado se va a explicar cómo funciona la pila de protocolo de la comunicación de Bluetooth Low Energy o BLE y como se ha implementado su uso en el microcontrolador LAUNCHXL-CC2650 por parte de Texas Instruments a través de una lista de APIs para interactuar con dicha pila. Los proyectos programados para este microcontrolador constan de dos partes (Ilustración 35), “stack” y “application”. En la parte de “stack”, el protocolo BLE está implementado como la tarea de mayor prioridad, mientras que en la parte de “application” se ubica TI-RTOS, el GAPRole y la aplicación desarrollada por el programador. Esto facilita las modificaciones a los desarrolladores de aplicaciones, puesto que tan sólo necesitaran cargar la parte de “application” al microcontrolador, dejando la parte de “stack” invariable para todo tipo de aplicaciones o cambios a un mismo programa.

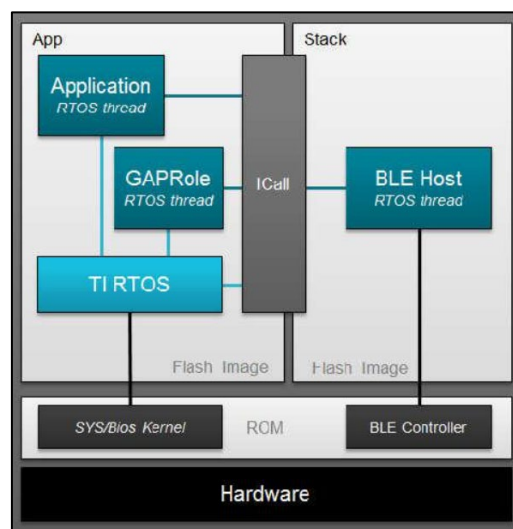


Ilustración 35. Estructura de un programa en CCS.

La entidad que se encarga de comunicar las dos imágenes del programa es ICall (Indirect Call Framework). Esta capa de software proporciona un mecanismo para que la aplicación interactúe con los servicios de la API de la pila del protocolo BLE, así como otros servicios de TI-RTOS (como por ejemplo la sincronización de hilos o el control del montículo). ICall permite que la aplicación y la pila BLE operen eficientemente, se comuniquen y compartan recursos en un entorno RTOS unificado.

El componente central de la entidad ICall es el “dispatcher” (controlador de transporte o despachador, Ilustración 36), que facilita la interfaz del programa de aplicación entre la aplicación y la pila de protocolos BLE a través de la frontera de la imagen dual.

Aunque la mayoría de las interacciones de ICall se abstraen dentro de la API de la pila BLE (como por ejemplo GAP, HCI, etc.), ICall también ofrece otros servicios relacionados con TI-RTOS tal y como se mencionó previamente, como la sincronización y comunicación entre procesos y la asignación y control del montículo.

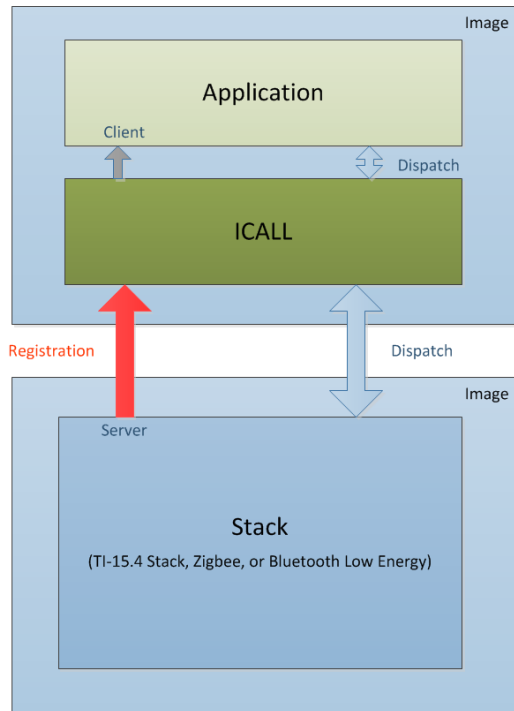


Ilustración 36. Comunicación entre las dos imágenes de un programa a través de ICALL.

Nótese que la mayor parte de los protocolos de la pila están almacenados en una única librería ya compilada, puesto que sus derechos y autoría pertenecen a Bluetooth Special Interest Group (SIG, [44]), aunque el uso de licencias BLE es totalmente gratuita.

BLE es una tecnología de red de área personal inalámbrica o WPAN destinada a aplicaciones de pequeño costo, que requieran un consumo de energía lo más bajo posible y que operen durante largos períodos de tiempo con el uso de baterías de pequeña capacidad. Sus aplicaciones más comunes están relacionadas con la asistencia sanitaria, el deporte, balizas, seguridad y el entretenimiento. BLE (también llamado Bluetooth Smart) se lanzó al mundo con la versión 4.0 de Bluetooth (Ilustración 37) y es necesario puntualizar que BLE y Bluetooth Classic son protocolos diferentes y no son compatibles entre sí, aunque normalmente los fabricantes de módulos Bluetooth suelen incluir ambas especificaciones en sus dispositivos.



Ilustración 37. Logo Bluetooth versión 4.0.

ARQUITECTURA DE LA PILA DEL PROTOCOLO BLE

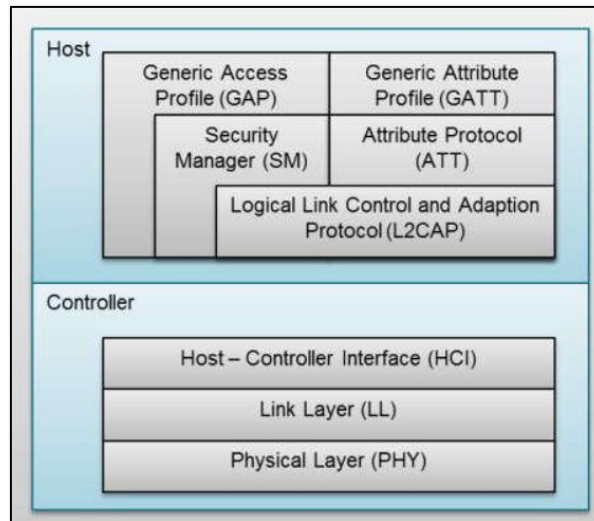


Ilustración 38. Arquitectura de la pila del protocolo BLE.

La pila del protocolo BLE [45] [46] [47] [48] [49] [50] [51] [52] [53] se divide en dos elementos, el controlador ('controller') con las capas inferiores y el servidor ('host'), con las capas superiores (Ilustración 38); ambos heredados de la implementación de la pila del protocolo Bluetooth tradicional o "classic", puesto que estas dos secciones se implementaban por separado. Por encima de la pila se sitúan los programas y aplicaciones que hacen uso del protocolo Bluetooth, llamando al último nivel "Application Layer" (App).

a) Physical Layer (PHY, nivel físico)

Esta capa, también llamada "Radio Layer" o "RF Layer" (capa de radiofrecuencia), es la primera de la pila del protocolo BLE y contiene toda la circuitería que se encarga de la comunicación analógica. Su objetivo consiste en modular o demodular las señales analógicas para convertirlas en señales digitales que puedan ser utilizadas por los niveles superiores.

La transmisión en el espectro de radiofrecuencias se produce en la banda de frecuencias libres y sin licencia ISM (Industrial, Scientific and Medical) de 2,4 GHz utilizando modulación GFSK (modulación por desplazamiento de frecuencia gaussiana), la cual reduce el pico de consumo de potencia y puede estar codificada o no codificada. El rango de alcance máximo de la señal puede ser de 30 a 50 m si no hay ningún obstáculo. La velocidad de transmisión y el tamaño máximo de los paquetes de datos transmitidos varían dependiendo de la versión Bluetooth:

- ❖ En la versión 4.0 y 4.1 la velocidad de transmisión es de 1 Mbps, aunque en las aplicaciones se obtiene una velocidad de hasta 302 kbps y el tamaño máximo de los paquetes es de 27 bytes.

- ❖ En la versión 4.2 se introdujo una extensión en el tamaño de los paquetes de datos (Data Length Extension), que amplió el tamaño máximo de los paquetes transmitidos de 27 bytes a 251 bytes, haciendo que la velocidad de transmisión real aumentase hasta los 803 kbps.
- ❖ En la versión 5.0 aumentó la velocidad de transmisión hasta los 2 Mbps, obteniendo una velocidad de transmisión en las aplicaciones de 1434 kbps.

El protocolo utiliza una técnica llamada “frequency hopping spread spectrum” (FHSS, salto de frecuencia en el espectro de radiofrecuencias), en la que la comunicación va cambiando de canales de emisión y recepción en cada evento de conexión con el objetivo de lograr una comunicación robusta, segura y minimizando interferencias. En cada conexión entre dispositivos, el orden y valor de los saltos de frecuencia son diferentes, y son establecidos al iniciar la conexión entre dos dispositivos.

El rango de frecuencias de 2,4 GHz, que se expande desde 2,402 GHz hasta 2,480 GHz se divide en 40 canales de 2 MHz de ancho de banda, tal y como muestra la Ilustración 39. El número de los canales sigue el orden de la ilustración 212, destacando que los canales 37, 38 y 39 se utilizan cuando un dispositivo está en estado “advertising” o publicador. Este estado se utiliza para descubrir dispositivos, realizar conexiones entre dispositivos o para trabajar en modo “broadcast” o emisor. El resto de canales se usan para la transmisión de datos en comunicaciones entre dispositivos que hayan establecido una conexión, aunque también pueden ser utilizados para publicaciones secundarias (secondary advertisement) en la versión 5.0.

El motivo por el cual los canales 37, 38 y 39 están tan separados entre sí y se ubican en esas frecuencias determinadas (imagen 213) es para evitar el mayor número de interferencias posibles con otros dispositivos Bluetooth y todo tipo de dispositivos que operan en esta misma frecuencia, como el Wi-Fi o los hornos de microondas, a 2,45 GHz.

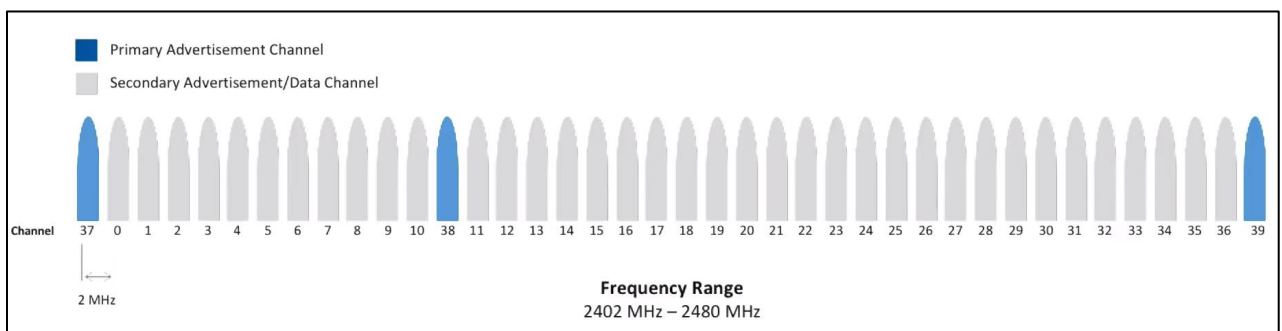


Ilustración 39. Canales RF de BLE.

b) Link Layer (LL, nivel de enlace)

Capa situada encima de la física, cuyo cometido es publicar, escanear y crear o mantener conexiones entre dispositivos. Este nivel suele implementarse como una combinación de hardware y software y es la primera capa que provee control y estructura a los datos con los que opera el nivel inferior en forma de paquetes de datos.

Cada dispositivo Bluetooth puede estar en uno de los siguientes estados definidos durante su funcionamiento (Ilustración 43). El nivel de enlace únicamente permite que un dispositivo esté en un estado determinado en cada momento, aunque dicho dispositivo puede tener varias instancias de este nivel del protocolo de comunicaciones.

- ❖ Standby (en espera): Estado por defecto. El dispositivo no está transmitiendo ni recibiendo paquetes de información. Este estado puede alcanzarse directamente desde cualquier otro estado.
- ❖ Advertising (anunciando): El dispositivo está emitiendo paquetes de datos al medio. Si el dispositivo está en modo “Advertiser”, este está a la vez a la escucha por si recibe algún tipo de respuesta por parte de otro dispositivo para realizar una conexión, pero en cambio, si el dispositivo está en modo “Broadcast”, no se realizará ningún tipo de escucha al medio.
- ❖ Scanning (escanenido): El nivel de enlace pone al dispositivo a la escucha de mensajes en el medio provenientes de otros dispositivos en estado “Advertising”. El escaneo puede ser pasivo, si el dispositivo únicamente está a la escucha o activo, si el dispositivo envía mensajes de respuesta a los dispositivos emisores solicitando más información sobre su identidad y sus servicios, con la intención de realizar una conexión en el futuro.
- ❖ Initiating (iniciando conexión): Cuando el dispositivo quiere realizar una conexión con otro dispositivo en modo “Advertiser”, tras escuchar su paquete de datos emitido, el primero responde enviando un mensaje al segundo para iniciar la conexión.

La secuencia de eventos es la siguiente (Ilustración 40): el dispositivo anunciante emite sus paquetes datos en los canales 37, 38 y 39 en unos intervalos de tiempo fijos preestablecidos guardando un periodo de tiempo entre emisiones en los canales lo suficientemente grande como para permitir que dispositivos en modo “Scanning” o “Initiating” puedan enviar sus paquetes de respuesta.



Ilustración 40. Intervalos de tiempo de emisión y espera de paquetes de respuesta.

Cuando un dispositivo en modo “Advertising” recibe un paquete de respuesta con la intención de realizar una conexión en uno de sus tres canales, finaliza inmediatamente la emisión de paquetes en estos tres canales. A continuación, se realiza una demora durante una ventana de tiempo que es la suma de 1,25 ms más un offset llamado “Transmit Window Offset”. Después, el dispositivo en modo “Advertising” (que pasará a tener el rol de esclavo), se pondrá a la escucha y deberá recibir un mensaje por parte del dispositivo iniciador de la conexión (futuro maestro) durante un periodo de tiempo llamado “Transmit Window”. Si la recepción del paquete no se produce, el dispositivo esclavo se volverá a poner a la escucha un “Connection Interval” después. Toda esta secuencia de eventos está representada en la Ilustración 41.

En el momento que el esclavo recibe el mensaje por parte del maestro y el dispositivo esclavo responde al mensaje, la conexión se establece.

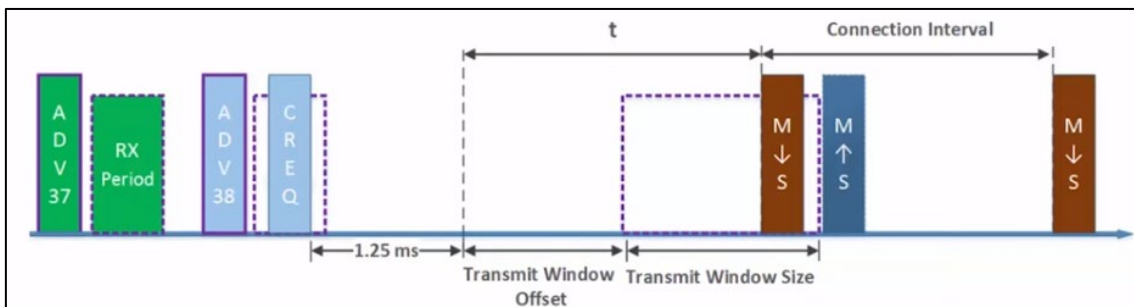


Ilustración 41. Secuencia de eventos para iniciar una conexión.

- ❖ **Connection (conectado):** Estado alcanzado desde el estado “Advertising” o desde “Initiating”. En este estado conviven dispositivos con el rol de maestro o de esclavo, ya que el equipo que haya entrado en este estado desde “Initiating” será maestro, mientras que será esclavo si proviene del lado de “Advertising”. Además, el dispositivo con el rol de maestro será el encargado de regular los parámetros de la conexión entre ambos equipos.

El dispositivo maestro tiene que enviar paquetes de datos al esclavo con una frecuencia determinada por el periodo “Connection Interval”, el cual puede ser establecido entre 7,5 ms y 4 s. Véase en la siguiente ilustración como durante un mismo intervalo de conexión pueden enviarse múltiples paquetes de datos. Además, el esclavo puede ignorar un número determinado de veces los mensajes del dispositivo maestro de manera consecutiva sin cortar la conexión con el objetivo de ahorrar energía. Este comportamiento es llamado “Slave latency”, como muestra la Ilustración 42.

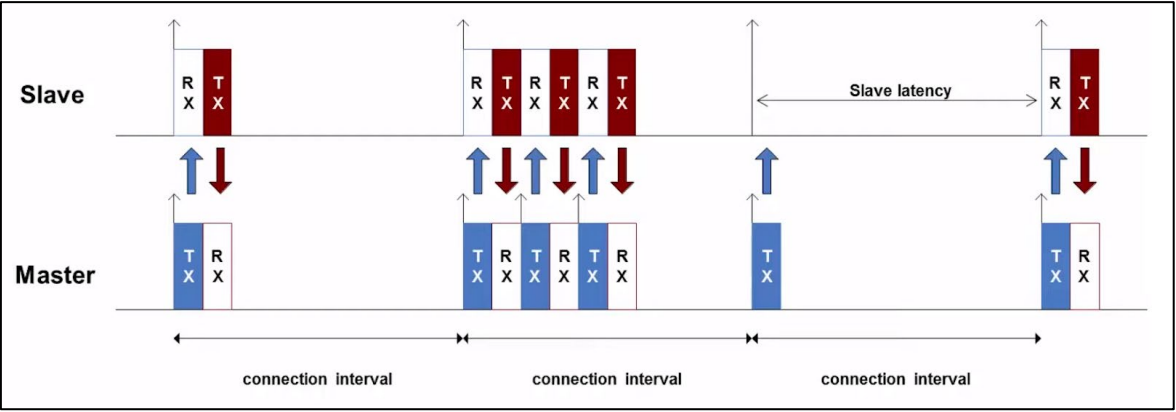


Ilustración 42. Comunicación entre dispositivos maestro y esclavo.

- ❖ Synchronization (sincronización): El equipo que esté en este estado está a la escucha de paquetes de datos transmitidos a una frecuencia o intervalo periódico por parte de otro dispositivo.

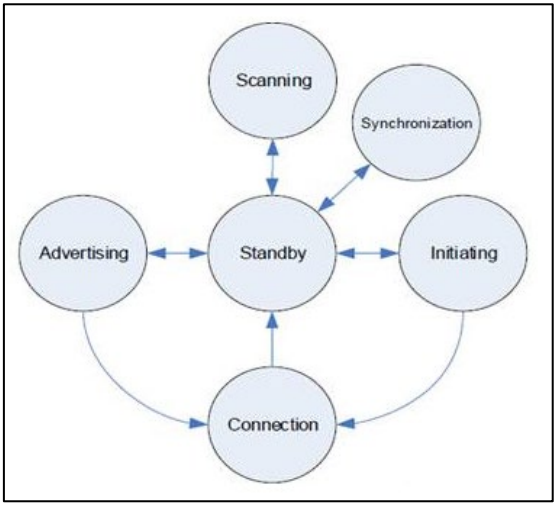


Ilustración 43. Diagrama de estados del nivel Link Layer.

Además de definir el estado de un dispositivo Bluetooth, el nivel de enlace se encarga de establecer las propias conexiones entre los dispositivos, los intervalos de tiempo entre dos eventos de conexión consecutivos y la encriptación de la información.

Por último, la capa de enlace también es responsable de asignar la dirección del dispositivo Bluetooth, conocido como BDA (Bluetooth Device Address), un número de identificación único de 48 bits. Esta dirección sería la equivalente a la MAC en el protocolo TCP/IP.

El formato de paquete de datos (Ilustración 45 e Ilustración 46) que utiliza el nivel de enlace es el mismo para los paquetes de anuncio (advertising) y transmisión de datos (connection), siendo la PDU (Protocol Data Unit) el único elemento diferenciable.

Para la versión 4.2 de BLE la estructura es la siguiente:

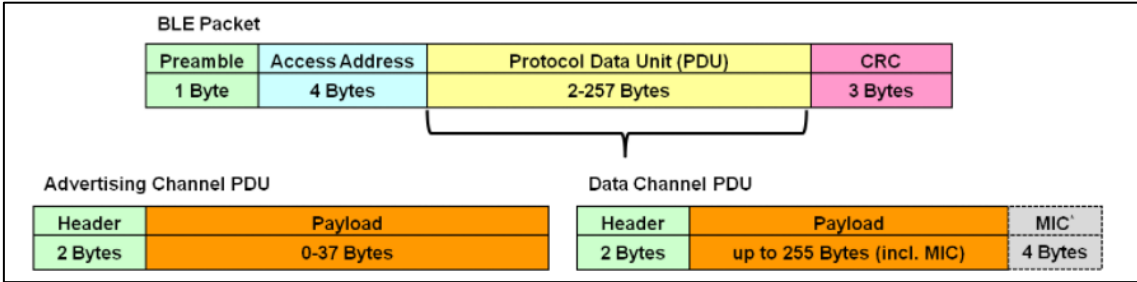


Ilustración 44. Formatos de los paquetes de datos.

En la versión 4.0 y 4.1 de BLE la PDU tiene un tamaño variable de 2 a 39 bytes.

- ❖ Las PDUs de los paquetes de datos de anuncio (advertising) tienen dos propósitos: emitir información para aplicaciones que no requieren conexión y descubrir y conectarse a dispositivos esclavos.

Hay 7 tipos de PDUs de paquetes de datos de “advertising”, que se clasifican en las siguientes categorías:

- Advertising PDUs: PDUs que transmiten dispositivos en modo “Advertising”. Hay 4 tipos:

Nombre de la PDU	Tipo de “advertising”
ADV_IND	No dirigida y conectable
ADV_DIRECT_IND	Dirigida y conectable
ADV_NONCONN_IND	No dirigida y no conectable
ADV_SCAN_IND	No dirigida y escaneable

- Scanning PDUs: SCAN_REQ (scanning request) y SCAN_RSP (scanning response). Estas PDUs utilizadas por dispositivos en estado de “Scanning”.
- Initiating PDUs: CONNET_REQ (connect request). Es la PDU que se utiliza cuando un dispositivo en modo “Initiating” (futuro maestro) quiere iniciar una conexión con otro (futuro esclavo).

- ❖ Las PDUs de los paquetes que transmiten información (connection) pueden transportar como máximo 246 bytes de información debido a las cabeceras de control de los diferentes niveles de la pila.

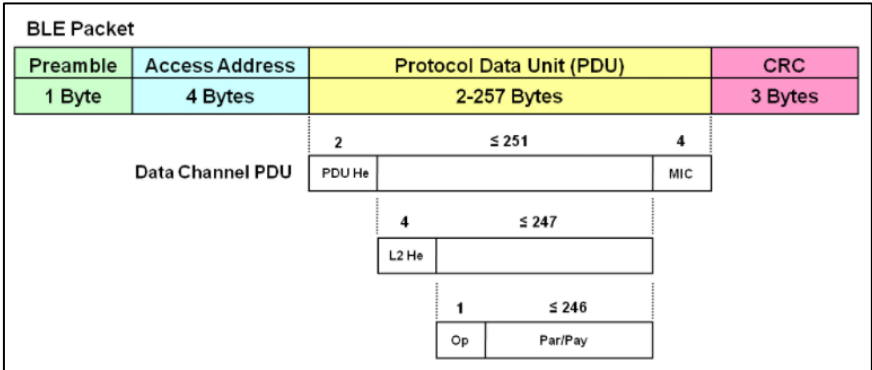


Ilustración 45. Formato de paquete de datos para transmisión de información.

Acrónimo del campo	Nombre completo del campo
PDU He	Cabecera de la PDU
MIC	Message Integrity Check
L2 He	Cabecera del nivel L2CAP
Op	Código de operación ATT
Par/Pay	Parámetros ATT y mensaje

En la versión 4.0 y 4.1 de BLE el tamaño máximo del campo de la PDU que corresponde a los parámetros ATT y al mensaje es de 22 bytes.

c) Host-Controller Interface (HCI, interfaz controlador-servidor)

Último nivel de la parte del controlador de la arquitectura del protocolo BLE. Este nivel vincula los niveles del controlador con los del servidor a través de una interfaz de comunicación estándar, la cual puede ser una función de una API o interfaces de comunicación comunes como UART, SPI o USB. Esta situación permite que un controlador pueda trabajar con diferentes instancias de “hosts” o que, en un PC, por ejemplo, el controlador esté implementado en un módulo o chip mientras que el “host” y la aplicación se ejecutan vía software en la propia CPU del ordenador.

d) Logical Link Control and Adaptation Protocol (L2CAP, nivel de control de enlace lógico y protocolo de adaptación)

Primera capa de la parte del “host” de la pila del protocolo BLE, encargada de mantener una comunicación lógica de extremo a extremo. Este nivel tiene dos funciones principales:

- ❖ Ofrece servicios de encapsulación de datos para los niveles superiores, recombina los paquetes de datos de los niveles inferiores para formar un paquete más grande que puede ser recibido y procesado por los niveles superiores. Al mismo tiempo, esta capa realiza el servicio opuesto, fragmenta los mensajes provenientes de los niveles superiores en el formato de paquete BLE estándar para los niveles inferiores, los cuales deben tener un tamaño máximo de 27 bytes.
- ❖ Se encarga de dar soporte a dos protocolos principales del nivel superior: el “Attribute Protocol”, (ATT, protocolo de atributos) y “Security Manager Protocol”, (SMP, protocolo administrador de seguridad), explicados a continuación.

e) Security Manager Protocol (SMP, protocolo administrador de seguridad)

Nivel que provee servicios de emparejamiento y distribución de claves de seguridad con el objetivo de conseguir una conexión e intercambio de datos seguro entre dispositivos BLE. El uso de los procedimientos de seguridad es recomendable, pero no obligatorio. Los dos principales sistemas de seguridad implementados en BLE son:

- ❖ Pairing (emparejamiento): Sistema de autenticación de dispositivos que establece claves compartidas, secretas y temporales que se utilizan para cifrar una conexión. Existen tres sistemas de emparejamiento:
 - Just Works (Simplemente funciona): Conexión directa. En realidad, no es ningún sistema de autenticación de seguridad.
 - Passkey Entry (Clave de acceso): Contraseña de 6 dígitos que deben introducir los dispositivos que quieren realizar una conexión.
 - Numeric Comparison (comparación numérica): Número de 6 dígitos que se muestra en los dispositivos que quieren realizar una conexión. Para iniciar la conexión, los usuarios de ambos terminales deberán aceptar la solicitud de conexión con dicho número.
 - OOB (Out of Band): Claves encriptadas transferidas a través de otros sistemas de comunicación, como NFC, por ejemplo.
- ❖ Bonding (enlazamiento o vinculación): La vinculación es el emparejamiento seguido de la distribución de claves que pueden utilizarse para cifrar el enlace en conexiones futuras entre los mismos dispositivos, resultando en una encriptación de los mensajes.

f) Attribute Protocol (ATT, protocolo de atributos)

Esta capa de la pila BLE implementa un protocolo de cliente-servidor y define como un cliente puede encontrar y acceder a los atributos de un servidor. El cliente solicita información del servidor y este le enviará la respuesta a su petición siempre y cuando no tenga ninguna petición previa pendiente.

Las operaciones que ofrece este nivel a la pareja cliente-servidor son las siguientes:

- Request (solicitud): Operación realizada por un cliente a un servidor, en la que el cliente pide realizar al servicio una acción determinada.
- Response (respuesta): Esta operación la realiza un servidor a un cliente en respuesta a una solicitud recibida previamente.

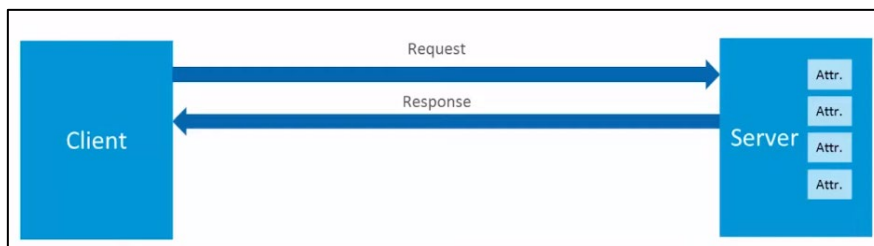


Ilustración 46. Operaciones de solicitud y respuesta.

- Command (orden): Operación similar a la solicitud, pero a diferencia de esta, una orden no necesita una respuesta inmediata por parte del servidor.

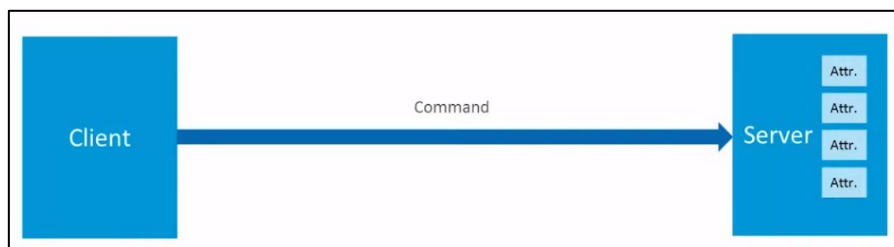


Ilustración 47. Operación de tipo orden.

- Indication (indicación): Operación que realiza un servidor a un cliente, informando a este último sobre el valor de un atributo determinado.
- Confirmation (confirmación): Mensaje de respuesta después de una indicación por parte de un cliente a un servidor. Las indicaciones y las confirmaciones tienen el sentido inverso a las solicitudes y respuestas.

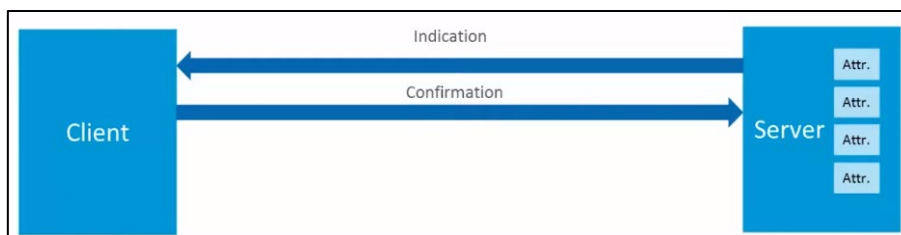


Ilustración 48. Operaciones de indicación y confirmación.

- Notification (notificación): Operación inversa a las órdenes. En este caso, es el servidor quien envía un mensaje a un cliente sin necesidad de recibir una respuesta.

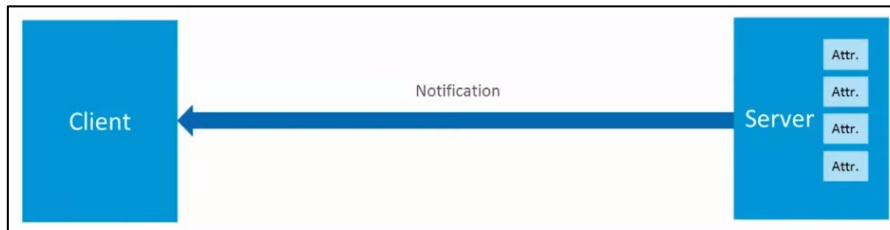


Ilustración 49. Operación de notificación.

Los servidores organizan y almacenan sus datos e información en forma de *atributos*, los cuales poseen cuatro elementos o propiedades (Ilustración 50):

- ❖ Manipulador (Handler): Método que sirve para acceder a los datos de un atributo. El manipulador identifica al atributo inequívocamente en el servidor.
- ❖ Identificador UUID (Universal Unique Identification Address), el cual especifica la naturaleza de los datos del atributo. Los UUID tienen un tamaño de 128 bits. Sin embargo, Bluetooth SIG ha definido 128 tipos de atributos estándar, llamados Bluetooth Base UUID, cuyo identificador es de 16 bits, lo que permite una comunicación más rápida y eficiente.
- ❖ Datos y valores etiquetados y direccionables, con tamaños de 0 a 512 bytes.
- ❖ Lista de permisos del atributo: Lectura, escritura o ambos a la vez.

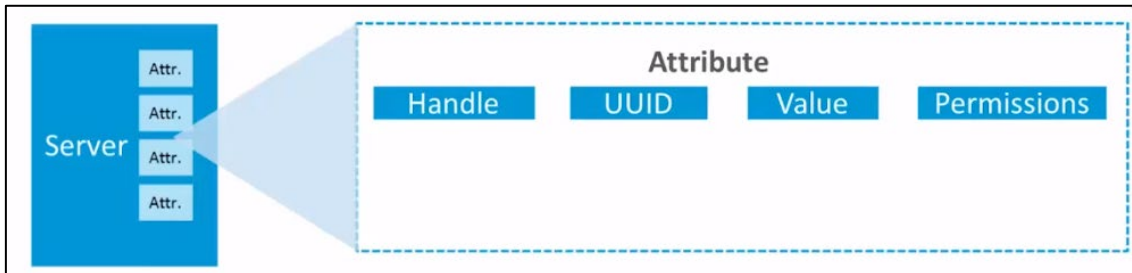


Ilustración 50. Propiedades de un atributo.

Cuando un cliente quiere leer o escribir valores en un atributo en un servidor, el primero envía una solicitud de lectura o escritura al segundo a través del manipulador. Después, el servidor envía una respuesta con el valor del atributo o con una respuesta de acuse de recibo. Si la operación es de lectura, el cliente tiene que realizar el tratamiento pertinente del valor recibido basándose en el UUID del atributo. En cambio, en una operación de escritura, el cliente es el encargado de asegurarse que los datos enviados se corresponden con el tipo de atributo. Si ocurre algún error, el servidor rechazará la operación.

g) Generic Attribute Profile (GATT, perfil de atributos genéricos)

Este es uno de los dos niveles superiores de la pila BLE (Ilustración 51), y está ubicado encima del nivel ATT. Su cometido es jerarquizar y organizar cómo se intercambia información entre diferentes aplicaciones, además de definir el procedimiento de búsqueda y acceso de atributos.

En esta capa, los datos se organizan en “Services” (servicios), los cuales contienen “Characteristics” (características), que son una combinación de datos de usuario y metadatos o información descriptiva y contienen dos o más atributos. Estos servicios se organizan en perfiles llamados “GATT profiles”, en los que cada uno puede contener varios servicios. Tanto los servicios como las características se identifican a través de un UUID de 16 bits o de 128 bits si el perfil no está predefinido por Bluetooth SIG.

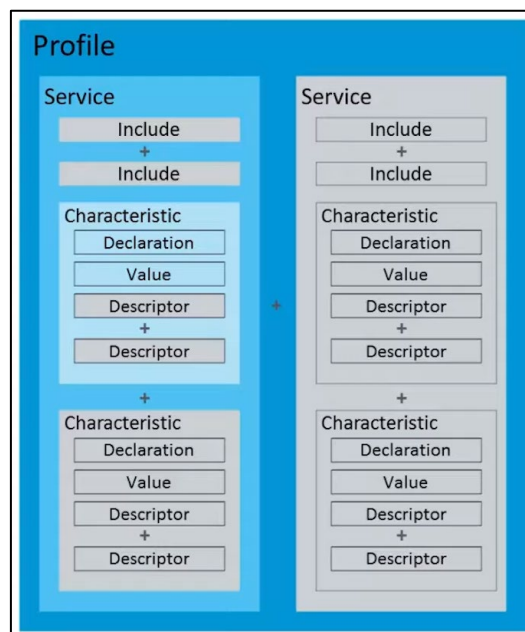


Ilustración 51. Perfil GATT.

Como ya se ha mencionado, las características (Ilustración 52) se componen de dos o más atributos, y están formadas por 3 elementos:

- ❖ Declaration (declaración), constituida a su vez por:
 - Properties (propiedades): Determina si un valor de la característica puede ser leído, escrito, notificado, indicado o emitido (broadcast).
 - Value handle (manipulador): Manipulador del atributo que controla el valor de la característica, permitiendo que la búsqueda de la característica por parte de un cliente sea rápida y eficaz, recibiendo únicamente el valor de la característica.
 - Type (UUID): Identificador del valor de la característica.

- ❖ Value (valor): Es un atributo.
- ❖ Descriptor: Los descriptores se utilizan para actualizar la información relacionada con el valor de la característica. Son los siguientes:
 - Characteristic Extended Properties Descriptor: Expande las propiedades de una característica ya definidas en la declaración.
 - Characteristic User Description Descriptor: Asocia un “string” con una característica.
 - Characteristic Presentation Format Descriptor: Se utiliza para asociar un número con una descripción estándar del valor, como por ejemplo un número decimal que representa la temperatura junto con su unidad, los grados centígrados.
 - Characteristic Aggregation Format Descriptor: Descriptor que habilita el uso de formatos más complejos.
 - Client Characteristic Configuration Descriptor (CCCD): Debe usarse si una característica está utilizando notificaciones o indicaciones. Para habilitar las notificaciones el CCCD debe tener el valor “1”, mientras que para habilitar las indicaciones el valor del CCCD tiene que ser “2”.
 - Server Characteristic Configuration Descriptor: Debe ser utilizado si la característica va a ser retransmitida (broadcast).



Ilustración 52. Elementos de una característica.

Los procedimientos que realiza este nivel son 3:

- ❖ Discovery (descubrimiento): Procedimiento que se utiliza para descubrir los servicios, características y atributos en un servidor. Hay cuatro elementos que deben ser descubiertos:
 - Primary Service Discovery: El cliente descubre primero todos los servicios o únicamente uno específico a través de su UUID.
 - Relationship Discovery: Después, el cliente puede descubrir servicios secundarios y otro tipo de servicios.
 - Characteristic Discovery: El cliente descubre las características de un servicio.
 - Characteristic Descriptor Discovery: El cliente descubre los descriptores de una característica.
- ❖ Client-initiated (iniciado por el cliente): Un cliente lee o escribe en los valores de las características de un servidor y en sus descriptores asociados.
- ❖ Server-initiated (iniciado por el servidor): Un servidor envía datos a un cliente directamente sin que este le haya hecho ninguna solicitud. El servidor puede enviar notificaciones o indicaciones de los valores de una característica. Las notificaciones no necesitan acuse de recibo, pero las indicaciones sí.

Handle	UUID	Value
0x0001	0x2800 (Service)	0x1800 (GAP Service UUID)
0x0002	0x2803 (Characteristic)	{0x02, 0x0003, 0x2A00} (Read, Value Handle, Device Name UUID)
0x0003	0x2A00	“Device Name”
0x0004	0x2803 (Characteristic)	{0x02, 0x0005, 0x2A01} (Read, Value Handle, Appearance UUID)
0x0005	0x2A01	0x0000
0x0006	0x2800 (Service)	0x1801 (GATT Service UUID)
0x0007	0x2800 (Service)	0x180F (Battery Service UUID)
0x0008	0x2803 (Characteristic)	{0x12, 0x0009, 0x2A19} (Read/Notify, Value Handle, Battery Level UUID)
0x0009	0x2A19	0x00
0x000A	0x2902	0x0000
0x000B	0x2800 (Service)	0x5AB20000-B355-4D8A-96EF-2963812DD0B8 (Proprietary Temperature Service UUID)
0x000C	0x2803 (Characteristic)	{12, 0x000D, 0x5AB20001-B355-4D8A-96EF-2963812DD0B8} (Read/Notify, Value Handle, Proprietary Temperature UUID)
0x000D	0x5AB20001-B355-4D8A-96EF-2963812DD0B8	0x0000
0x000E	0x2901	“Bedroom”
0x000F	0x2904	{0x0E, 0xFE, 0x272F, 0x01, 0x010B}
0x0010	0x2902	0x0000

Ilustración 53. Ejemplo de una tabla de datos de un atributo.

Los niveles GATT y GAP (explicado a continuación) conforman la principal interfaz de la pila del protocolo BLE.

h) Generic Access Profile (GAP, perfil de acceso genérico)

Última capa de la pila BLE, cuyo cometido es definir cómo los dispositivos se buscan, conectan y enlazan entre sí, a través de roles de funcionamiento, modos, procedimientos y enlaces seguros.

Los roles o modos de funcionamiento de los dispositivos se agrupan por parejas y son los siguientes:

- ❖ (1) Advertiser (publicador o anunciante): Dispositivo que emite paquetes de datos al medio con información relativa a los servicios que presta. Si un dispositivo, con el rol de escáner quiere recibir la información, deberá realizarse una conexión entre ambos dispositivos previamente.
- ❖ (1) Scanner (escáner): Dispositivo en escucha en el espectro RF de paquetes de datos que establece comunicaciones con dispositivos en modo “Advertiser”.
- ❖ (2) Central: Dispositivo maestro que inicia una conexión y una vez realizada, la administra. Nótese que un maestro puede realizar conexiones con varios esclavos a la vez. Un dispositivo central puede tener varios periféricos.
- ❖ (2) Peripheral (periférico): Dispositivo esclavo que acepta una conexión por parte de un dispositivo maestro y que seguirá sus instrucciones.
- ❖ (3) Broadcaster (emisor): Dispositivo que emite paquetes de datos al medio, sin necesidad de haber creado previamente ninguna conexión con otro dispositivo.
- ❖ (3) Observer (observador): Dispositivo en escucha en el espectro RF de paquetes de datos emitidos por dispositivos en modo “Broadcaster”.

i) Profiles (perfiles)

Los perfiles son especificaciones que describen casos de uso para dos o más dispositivos con varios servicios. También describen los procedimientos de cómo los dispositivos deben ser descubiertos y conectados a través de las funciones de GATT y GAP y qué servicios son necesarios en cada dispositivo.

Las aplicaciones que utilicen la pila del protocolo BLE pueden implementar los perfiles predefinidos por Bluetooth SIG o crear uno propio con las especificaciones deseadas en cuanto a comportamiento y servicios.

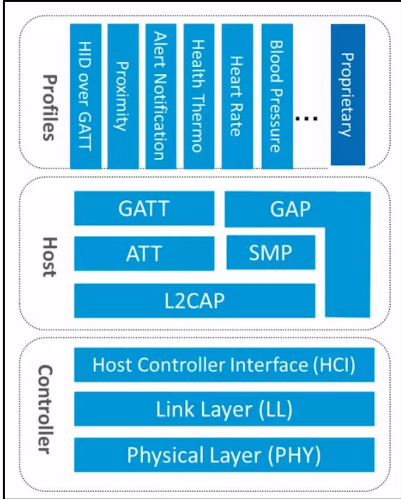


Ilustración 54. Pila del protocolo BLE con perfiles.

La mayor parte de la documentación del protocolo de comunicación BLE proviene de Texas Instruments y de Nordic Semiconductor (Ilustración 54), una de las empresas integrantes de Bluetooth SIG.

2.5. Protocolo de comunicación I2C

a) Fundamentos del protocolo I2C

El protocolo de comunicación I2C o I²C, Inter-Integrated Circuit [54], fue desarrollado por Phillips Semiconductors (hoy NXP Semiconductors, perteneciente a Qualcomm) en 1982. También fue llamado TWI (Two Wire Interface) por otras tecnológicas por motivos de licencia hasta que esta venció en 2006.



Ilustración 55. Logo del bus I2C.

Este es un bus de comunicación serie síncrono que se utiliza principalmente para comunicar diferentes partes de un mismo circuito, como un controlador y sus circuitos periféricos, por ejemplo. El protocolo [55] utiliza únicamente dos cables para la transmisión y recepción de datos, SDA (datos en serie) y SCL (reloj serie), tal y como muestra la Ilustración 56. Además, I2C es multi-maestro, multi-esclavo, single-ended, transmite los datos a través del método de conmutación de paquetes y tiene acuse de recibo, que significa que el transmisor del mensaje recibe una notificación por parte del receptor si el mensaje ha llegado correctamente.

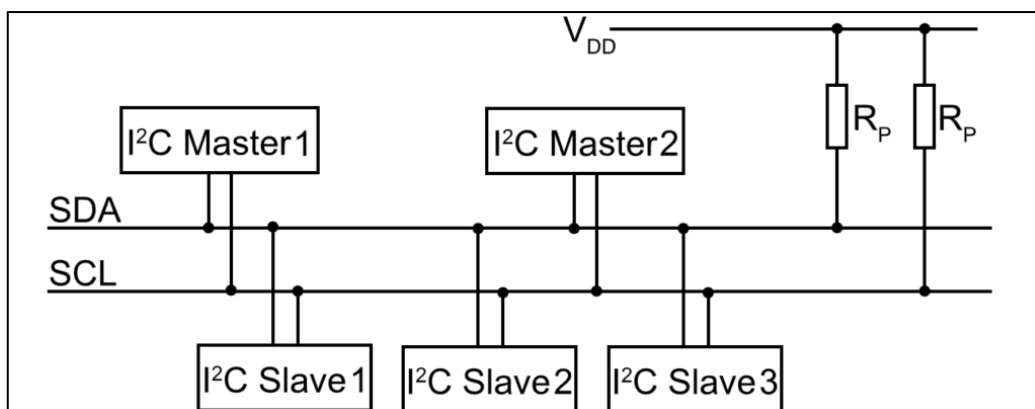


Ilustración 56. Ejemplo de implementación de comunicación entre dispositivos mediante I2C.

- ❖ El cable SDA es el medio por el cual se transmite o recibe el mensaje.
- ❖ Por otro lado, el cable SCL porta la señal de reloj gobernada por el dispositivo maestro para que todos los dispositivos del bus estén sincronizados.

Es necesario mencionar que ambos cables tienen una configuración “open drain” o drenaje abierto (Ilustración 57), es decir, los pines SDA y SCL pueden poner la señal a nivel bajo, pero no a nivel alto, por lo tanto, es necesario conectar ambos buses a un voltaje de referencia con resistencias pull-up.

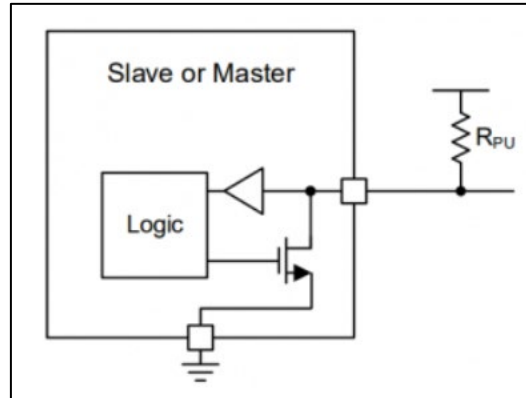


Ilustración 57. Configuración "open drain".

Un dispositivo que utilice la comunicación I2C tiene dos maneras de funcionar [56], bien en modo maestro, bien en modo esclavo. El dispositivo maestro es el encargado de iniciar la comunicación con los dispositivos esclavos y de controlar la señal de reloj enviada por el cable SCL. Sin embargo, la realidad es que los modos de operación de un dispositivo I2C son cuatro: maestro transmisor, maestro receptor, esclavo transmisor y esclavo receptor.

La velocidad de transmisión [57] del bus puede ser: “standard mode”, a 100 Kbit/s; “fast mode”, a 400 Kbit/s; “fast mode plus”, a 1Mbit/s; “high speed mode”, a 3,2 Mbit/s; o “ultra fast mode”, a 5 Mbit/s.

Por último, se explicará la estructura básica que deben tener los mensajes [58] enviados a través del cable SDA de un bus I2C. Su estructura la esquematiza la Ilustración 58.

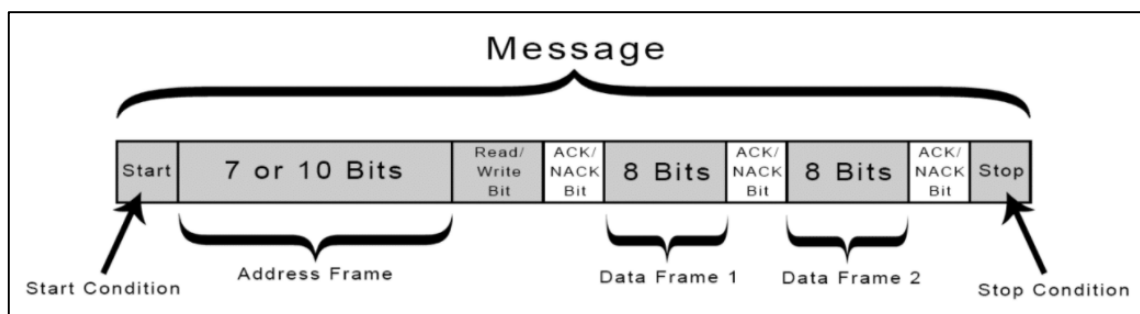


Ilustración 58. Estructura básica de los mensajes en la comunicación I2C.

- ❖ Bit de inicio (Start Condition): La línea SDA cambia de nivel alto a bajo antes de que la línea SCL cambie de nivel alto a bajo (sino no se registra el cambio).

- ❖ Dirección (Address Frame): Secuencia de 7 o 10 bits con la dirección del dispositivo esclavo con el que el dispositivo maestro quiere iniciar la comunicación. Esta secuencia debe ir a continuación de la condición de inicio.
- ❖ Bit de lectura o escritura (Read/Write Bit): El dispositivo maestro especifica si su intención es enviar o recibir un mensaje del dispositivo esclavo tras la secuencia de bits con la dirección del esclavo. Si el dispositivo maestro quiere enviar un mensaje, el bit será 0, mientras que, si quiere recibir información, el bit será 1.
- ❖ Bit de confirmación (ACK/NACK bit): Bit que envía el dispositivo receptor del mensaje para indicar al transmisor que los datos se han recibido correctamente. Este bit de confirmación debe intercalarse entre los paquetes de datos, explicados a continuación, y justo después del bit de lectura o escritura. Si el bit es 0 corresponde a un ACK, pero si la línea SDA se mantiene a nivel alto, se entiende que el mensaje no ha llegado correctamente (NACK).
- ❖ Paquete de datos (Data frame): Secuencia de 8 bits correspondiente al paquete de datos enviado o recibido, siendo el primer bit el más significativo.
- ❖ Bit de parada (Stop Condition): El dispositivo maestro finaliza la comunicación cambiando el estado de la línea SDA de nivel bajo a alto después de una transición de nivel bajo a alto en la línea SCL, manteniendo desde este momento la línea SCL a nivel alto (se deja de enviar la señal de reloj).

b) Configuración del módulo I2C en el LAUNCHXL-F28069M

La LaunchPad posee dos módulos I2C que cumplen las especificaciones de la versión 2.1 del protocolo I2C de Philips Semiconductors, pero únicamente uno de los módulos es externo y de propósito general, ya que el otro es interno y se utiliza solo para comunicarse con la EEPROM interna. El módulo I2C del microcontrolador permite velocidades de transmisión entre 10 Kbps y 400 Kbps. Este módulo permite los 4 modos de funcionamiento descritos anteriormente: maestro transmisor, maestro receptor, esclavo transmisor y esclavo receptor además de que permite generar interrupciones funcionando tanto como maestro como esclavo.

Otras dos características dignas de mención son que los paquetes de datos de los mensajes no han de tener 8 bits de tamaño de forma obligatoria, sino que su tamaño puede variar de 1 a 8 bits y que las líneas SDA están conectadas a dos estructuras de registros FIFO de 4 niveles para la recepción y la emisión de los mensajes, como muestra la Ilustración 59.

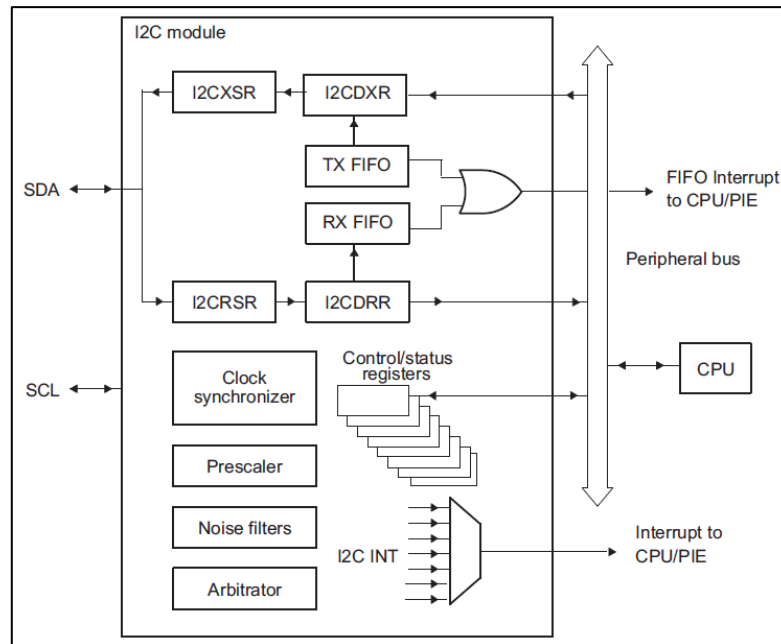


Ilustración 59. Esquema del módulo I2C del microcontrolador.

El microcontrolador permite dos métodos de transmisión de datos:

- ❖ Modo estándar: se envían los “n” paquetes de datos.
- ❖ Modo repetición: se envía únicamente un paquete de datos de forma repetida y continuada en el tiempo hasta que se produzca una condición de parada.

Tal y como se explicó en el punto anterior, ambos cables del bus (SDA y SCL) deben conectarse a +Vcc (3,3 V en este caso) con resistencias pull-up, ya que tienen una configuración de solo drenaje de corriente, no pueden ser fuentes (“open-drain configuration”). Aunque internamente pueden conectarse a resistencias pull-up embebidas en la LaunchPad, para el desarrollo del programa de prueba y el proyecto final se descartará esta opción, porque como se verá más adelante, el valor de estas resistencias internas es demasiado grande e induce a errores en la comunicación a altas velocidades de transmisión. De esta manera, deberá realizarse el cálculo de las resistencias pull-up e implementar el circuito externamente.

FUNCIONAMIENTO DE LOS REGISTROS AL ENVIAR O RECIBIR MENSAJES

La CPU mete los datos a transmitir en el registro “I2CDXR” y el registro donde la CPU lee información es el “I2CDRR”.

Si el módulo I2C está configurado como transmisor, la información ubicada en el registro “I2CDXR” se copia al registro “I2CDSR” para después ser enviada a través del pin SDA bit a bit. Sin embargo, si el módulo está configurado como receptor, los datos recibidos desde el pin SDA se meten en el registro “I2CRSR” para ser copiados al registro “I2CDRR” a continuación.

GENERACIÓN DE LA SEÑAL DE RELOJ PARA EL MÓDULO I2C Y PARA EL RELOJ MAESTRO DEL PIN SCL

La señal de entrada de reloj del módulo proviene de la señal de reloj de la CPU del microcontrolador (hay que tener en cuenta si la señal pasa por los diferentes divisores de frecuencia). Después, dentro del módulo I2C se encuentran otros dos divisores de frecuencia; tras pasar por el primero (IPSC), se obtiene la señal de reloj del módulo y, tras pasar por el segundo (ICCL e ICCH, explicados más adelante) se obtiene la señal para el reloj maestro, tal y como exhiben la Ilustración 60 y la Ilustración 61.

$$Frecuencia\ de\ reloj\ del\ módulo\ I2C = \frac{SYSCLKOUT}{(IPSC + 1)}$$

Se remarca que la señal de reloj del módulo I2C debe cumplir las especificaciones y estar configurada entre 7 y 12 MHz.

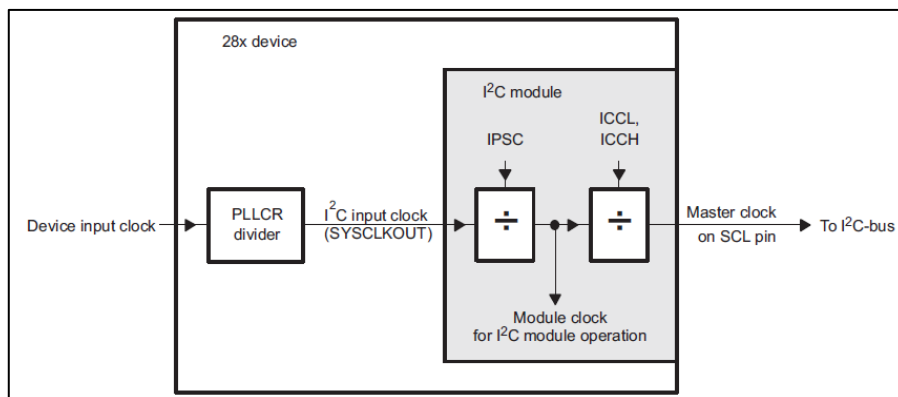


Ilustración 60. Esquema para obtener la señal de reloj del módulo I2C y del reloj SCL.

- ❖ Divisor de frecuencia “IPSC”, ubicado en el registro “I2CPSC.all”. A su salida se obtiene la frecuencia de funcionamiento del módulo I2C.
- ❖ Divisor de frecuencia “ICCL”, localizado en el registro “I2CCLKL”. Regula el tiempo que la señal está a nivel bajo.
- ❖ Divisor de frecuencia “ICCH”, localizado en el registro “I2CCLKH”. Regula el tiempo que la señal está a nivel alto.

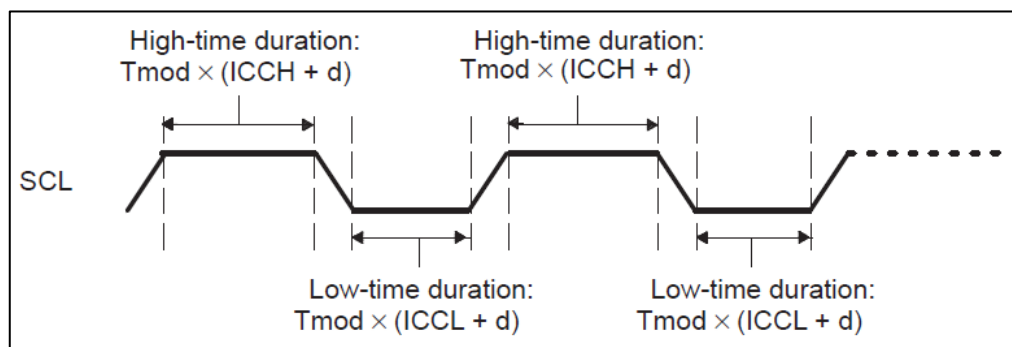


Ilustración 61. Configuración de la señal SCL.

Fórmula de la señal de reloj SCL:

$$T_{mst} = T_{mod} \times [(ICCL + d) + (ICCH + d)]$$

$$T_{mst} = \frac{(IPSC + 1) [(ICCL + d) + (ICCH + d)]}{I2C \text{ input clock frequency}}$$

Ilustración 62. Fórmula para obtener la señal de reloj SCL.

T_{mod} es el periodo de la señal de reloj del módulo I2C y el parámetro “d” puede ser 5, 6 o 7 dependiendo del valor introducido en el registro “IPSC”.

IPSC	d
0	7
1	6
Greater than 1	5

Ilustración 63. Valores de “d” en función de IPSC.

VALIDACIÓN DE DATOS

El nivel del pin SDA debe ser constante durante el periodo que dure la señal SCL en nivel alto y solo puede cambiar cuando la señal de reloj esté a nivel bajo.

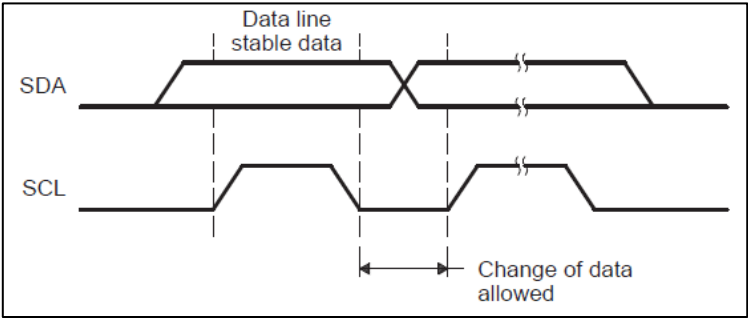


Ilustración 64. Validación de datos.

CONDICIONES DE INICIO Y PARADA

La condición de inicio se produce cuando la señal SDA cambia de nivel alto a bajo cuando la señal de reloj SCL está a nivel alto.

La condición de parada se produce cuando la señal SDA cambia de nivel bajo a alto cuando la señal de reloj SCL está a nivel alto.

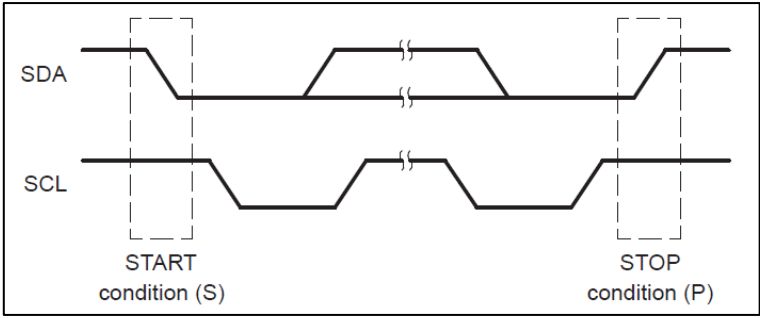


Ilustración 65. Condiciones de inicio y parada.

FORMATO DE TRANSMISIÓN DE DATOS

Los datos son transmitidos con el bit más significativo primero.

- ❖ Si la dirección del esclavo es de 7 bits (Ilustración 66), los bits “FDF” y “XA” del registro “I2CMDR” se configuran a 0. Después del bit de inicio de la trama se envían los 7 bits correspondientes a la dirección del dispositivo esclavo, un bit correspondiente a lectura o escritura y se espera el envío del bit de confirmación (ACK) por parte del esclavo. Tras la confirmación se enviarán (o recibirán) de 1 a 8 bits de datos, siempre seguido de un bit de confirmación. Una vez finalizado el envío de todos los datos, el dispositivo maestro cierra la comunicación enviando un bit de parada.

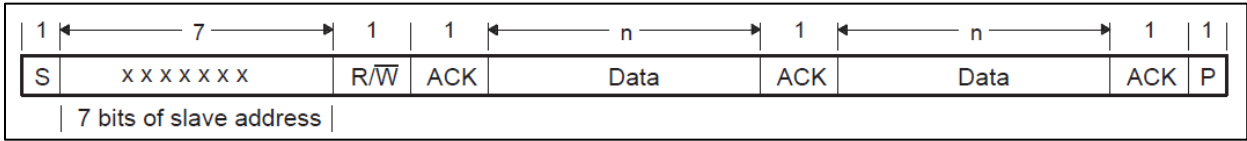


Ilustración 66. Estructura del mensaje si la dirección del esclavo son 7 bits.

- ❖ Si la dirección del esclavo es de 10 bits (Ilustración 67), el bit “FDF” deberá ser 0 mientras que el “XA” (eXtended Address) será 1. A diferencia del caso anterior, cuando la dirección del dispositivo esclavo consta de 7 bits, tras el bit de inicio de comunicación, los siguientes 7 bits siguen esta estructura: 11110xx (Ilustración 67), donde los dos últimos bits de este paquete de 7 corresponden a los bits más significativos de la dirección del dispositivo esclavo. A continuación, y al igual que antes, vienen los bits de lectura/escritura y confirmación. Después, los siguientes 8 bits (deben ser obligatoriamente 8), corresponden a los 8 bits menos significativos de la dirección del esclavo. A partir de este punto el formato de la trama es idéntico que el descrito en el párrafo anterior para direcciones de dispositivos esclavos de 7 bits.

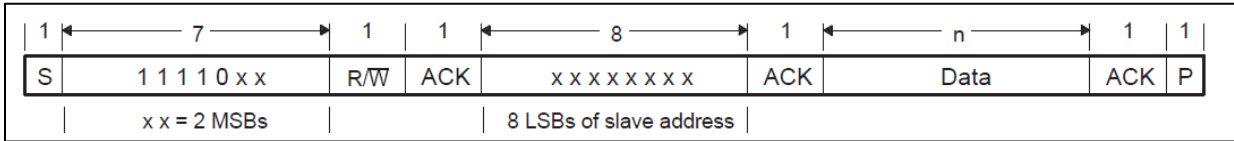


Ilustración 67. Estructura del mensaje si la dirección del esclavo son 10 bits.

- ❖ FDF: El microcontrolador tiene la opción de transmitir en el denominado “formato libre”, (free data format, Ilustración 68). Para ello, el bit “FDF” del registro “I2CMDR” debe ser 0. En este formato, el maestro no envía la dirección del esclavo al que quiere transmitir la información, sino que desde un principio envía paquetes de datos con longitud entre 1 y 8 bits.

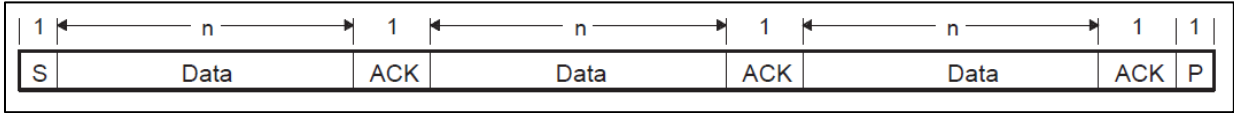


Ilustración 68. Estructura del mensaje en formato libre.

- ❖ Repetición de la condición de inicio (Ilustración 69): Al final de cada mensaje de datos y el consiguiente bit de confirmación, el dispositivo maestro puede enviar un bit de arranque y comenzar una nueva transmisión sin necesidad de enviar un bit de parada antes. Esto permite al maestro comunicarse con varios esclavos con direcciones diferentes sin tener que ceder el control del bus I2C con el bit de parada. Este modo de funcionamiento puede aplicarse con el direccionamiento de 7 bits, el de 10 bits o el de formato libre.

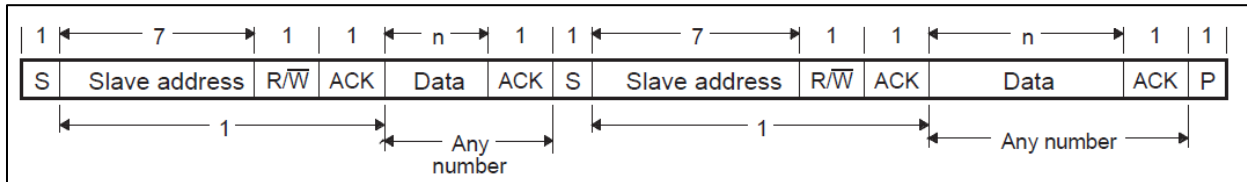


Ilustración 69. Estructura del mensaje en repetición de la condición de inicio.

Últimas consideraciones:

El bit correspondiente a lectura o escritura es 0 si el maestro quiere escribir y 1 si el maestro va a recibir información del esclavo.

El número de bits que componen el mensaje de datos a transmitir o recibir se especifica en el campo “BC” del registro “I2CMDR”.

SINCRONIZACIÓN DE LA SEÑAL DE RELOJ

En el caso de que en el bus I2C haya más de un dispositivo maestro (Ilustración 70), sus señales de reloj deben ser sincronizadas para que los datos puedan ser escritos y leídos correctamente. Como todos los pines SCL del bus I2C están conectados entre sí, el dispositivo que primero genere un semiperiodo de nivel bajo prevalece sobre los demás, obligando al resto de dispositivos a iniciar su semiperiodo de nivel bajo. La señal de reloj se mantiene a nivel bajo durante el tiempo marcado por el dispositivo con el mayor semiperiodo a nivel bajo, mientras que el resto de los dispositivos deben esperar antes de iniciar sus semiperiodos de nivel alto.

De esta manera se obtiene una señal de reloj SCL sincronizada, donde el dispositivo más lento determina la duración del semiperiodo de nivel bajo y el más rápido fija la duración del semiperiodo de nivel alto.

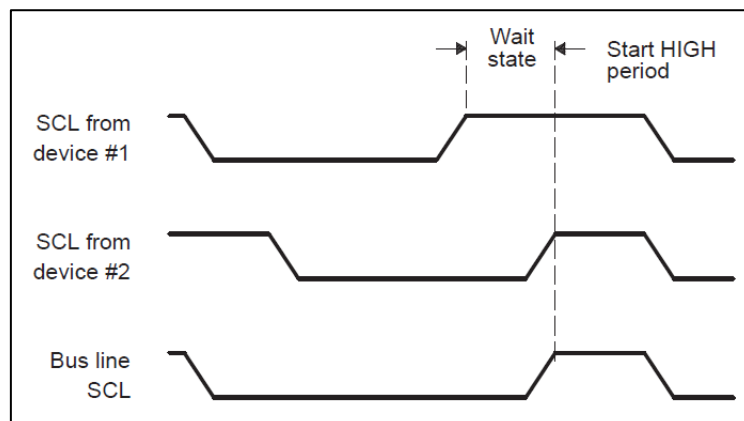


Ilustración 70. Sincronización de la señal de reloj SCL.

2.6. Electrocardiograma (ECG)

Un electrocardiograma (ECG, Ilustración 71) [59] es un procedimiento médico que detecta la actividad eléctrica del corazón en función del tiempo, puesto que cada vez que el corazón late, una señal eléctrica circula a través de él. Esta prueba [60] es ampliamente utilizada para valorar la condición del corazón de forma no invasiva y se usa para evaluar el estado del sistema de conducción del corazón a nivel muscular, pero también, de forma indirecta, la condición de este órgano como una bomba y la aparición de ritmos patológicos causados por daño al tejido de conducción de las señales eléctricas, otros trastornos no-cardíacos o para medir el valor del ritmo cardiaco.

El ECG es la representación gráfica de la actividad bioeléctrica del músculo cardíaco, por lo que un equipo de registro de ECG (electrocardiógrafo) es comparable a un voltímetro midiendo diferencias de potencial eléctrico.

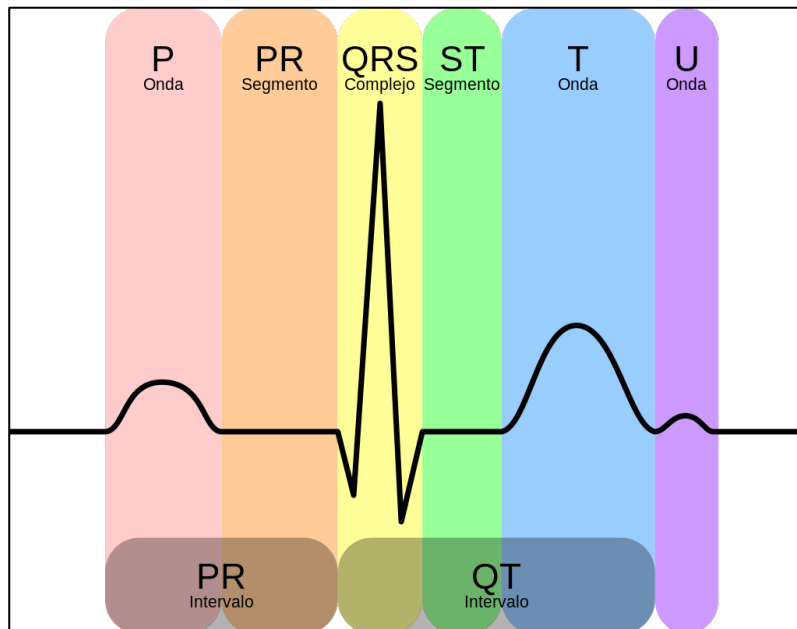


Ilustración 71. Representación teórica de un ECG.

La forma estándar [61] de un electrocardiograma registrando un latido cardíaco normal consiste en una onda “P”, un complejo “QRS” y una onda “T”, separados estos 3 eventos por 2 periodos o segmentos de inactividad. Después de la aparición de la onda “T”, existe otra pequeña onda “U”, pero esta última suele pasar desapercibida o directamente no registrarse porque es poco significativa.

- ❖ La onda P es la señal eléctrica correspondiente a la despolarización auricular.
- ❖ El complejo QRS (Ilustración 72), formado por las ondas Q, R y S, corresponde a la corriente eléctrica que causa la contracción de los ventrículos derecho e izquierdo, llamada despolarización ventricular, la cual es mucho más potente que la de las aurículas y hace trabajar a más masa muscular, produciendo de este modo una mayor deflexión en el electrocardiograma. Es debido a esto que la detección del complejo QRS es el que se suele utilizar para calcular el valor del ritmo cardiaco.

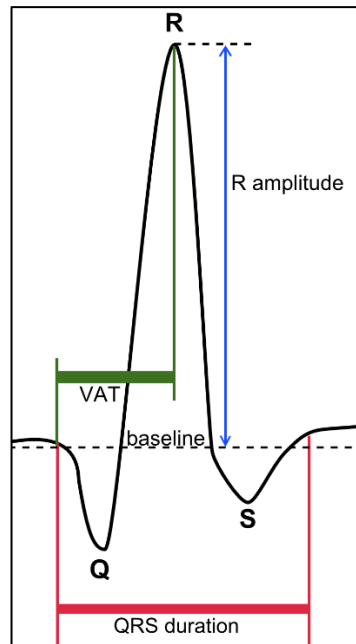


Ilustración 72. Representación del complejo QRS.

- ❖ La onda T representa la repolarización de los ventrículos.

Estos son eventos eléctricos que no deben ser confundidos con los eventos mecánicos correspondientes, es decir, a la contracción y relajación de las aurículas y ventrículos del corazón.

De esta manera, la sístole mecánica o contracción ventricular comienza justo después del inicio del complejo QRS y culmina justo antes de terminar la onda T. La diástole, que es la relajación y relleno ventricular posterior a la sístole, que corresponde con la contracción de las aurículas, se produce justo después de iniciarse la onda P.

DERIVACIONES

En electrocardiografía, una derivación se refiere a la medida del voltaje entre dos electrodos. Los electrodos se colocan sobre el cuerpo del paciente y se conectan al aparato de detección electrocardiográfica. Las derivaciones de un ECG utilizan diferentes combinaciones de electrodos para medir distintas señales procedentes del corazón desde ángulos diferentes.

Para realizar un ECG estándar de 12 derivaciones se utilizan diez electrodos, para obtener 3 derivaciones periféricas y 9 precordiales. Sin embargo, para medir el valor del ritmo cardiaco es suficiente con utilizar las derivaciones I, II y III correspondientes a las derivaciones periféricas, las cuales se colocarán respectivamente en el brazo izquierdo, en el brazo derecho y en la pierna derecha (véase la Ilustración 73).

Esta última metodología será la aplicada en el proyecto, ya que utilizarán 3 electrodos colocados en el brazo izquierdo, en el brazo derecho y en la pierna derecha.

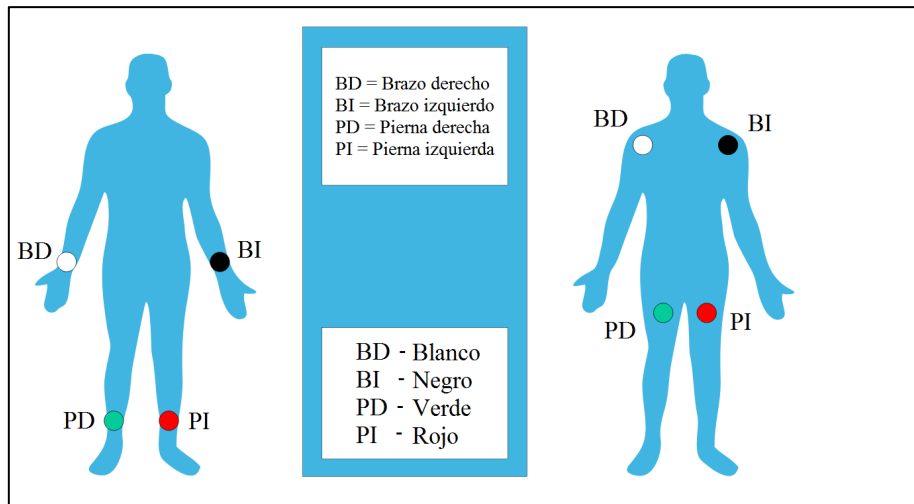


Ilustración 73. Opciones de colocación de los electrodos periféricos según la American Heart Association (AHA).

- ❖ La derivación I mide la diferencia de potencial entre el electrodo del brazo derecho y el izquierdo.
- ❖ La derivación II mide la diferencia de tensión del brazo derecho a la pierna izquierda.
- ❖ La derivación III mide la diferencia de potencial del brazo izquierdo a la pierna izquierda.

ALGORITMO DE PAN-TOMKINS PARA DETECTAR QRS

Algoritmo propuesto por Jiapu Pan y Willis J. Tompkins en 1985, en la revista IEEE "Transactions on Biomedical Engineering" [62] para detectar el complejo QRS de un electrocardiograma (ECG). Como el complejo QRS posee un pico muy representativo en la gráfica del ECG, esta característica lo hace especialmente adecuado para medir la frecuencia cardíaca.

Este algoritmo [63] aplica una serie de filtros para resaltar el contenido de frecuencia de esta despolarización rápida del corazón y elimina el ruido de fondo (Ilustración 74). A continuación, eleva al cuadrado la señal para amplificar la contribución del QRS, lo que facilita la identificación del complejo QRS. Por último, se aplican umbrales adaptativos para detectar los picos de la señal filtrada.

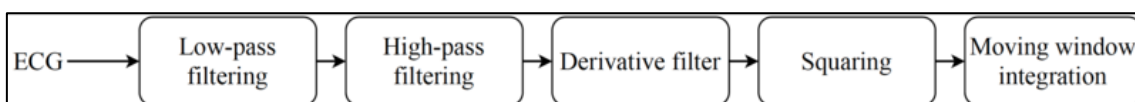


Ilustración 74. Diagrama de bloques de la fase de pre-procesado del algoritmo.

Una vez reconocido el complejo QRS, el valor de la frecuencia cardíaca se calcula en función de la distancia entre dos complejos QRS consecutivos (o picos R).

$$HR(bpm) = \frac{60}{RR(s)}$$

SENSOR DE SPARKFUN (AD8232)

El sensor que se utilizará en el proyecto para detectar las señales electrocardiográficas de los pacientes será el SparkFun Single Lead Heart Rate Monitor [64], basado en el AD8232 [65] exhibido en la Ilustración 75.

Según el fabricante, este sensor se utiliza para medir la actividad eléctrica del corazón. Esta actividad eléctrica puede ser graficada como un electrocardiograma y la salida puede ser leída por un controlador como una señal analógica. Como los ECGs pueden ser extremadamente ruidosos, el sensor actúa como un amplificador operacional para ayudar a obtener una señal clara de los intervalos PR y QT de forma sencilla.

El AD8232 es un bloque de acondicionamiento de señal integrado para ECG y otras aplicaciones de medición de marcadores. Está diseñado para extraer, amplificar y filtrar pequeñas señales eléctricas en presencia de condiciones ruidosas, como las creadas por el movimiento o la colocación de electrodos a distancia.

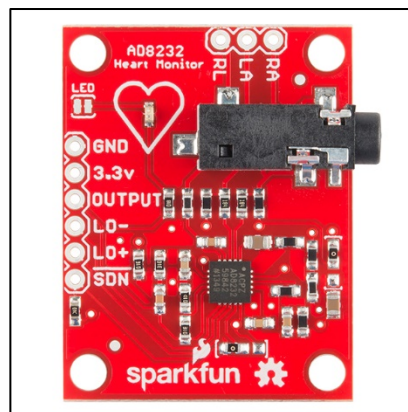


Ilustración 75. Sensor electrocardiográfico.

El sensor dispone de nueve conexiones a las que se pueden soldar pines, cables u otros conectores. /SDN, LO+, LO-, OUTPUT, 3.3V, GND proporcionan pines esenciales para operar este monitor con un Arduino u otra placa de desarrollo. También se proporcionan en esta placa los pines RA (brazo derecho), LA (brazo izquierdo) y RL (pierna derecha) para conectar y utilizar otras conexiones de electrodos.

Pin	Descripción de la conexión
3.3 V	Tensión de alimentación
GND	Tierra
OUTPUT	Salida de la señal analógica
LO-	Señal digital de desconexión -
LO+	Señal digital de desconexión +
/SDN	Señal digital de apagado
RL	Entrada del electrodo de la pierna derecha
LA	Entrada del electrodo del brazo izquierdo
RA	Entrada del electrodo del brazo derecho

Además, hay un LED embebido en la PCB que se iluminará al mismo ritmo que los latidos del corazón.

Los electrodos utilizados, expuestos en la Ilustración 76, también son del mismo fabricante [66]. Tienen una longitud de 24 pulgadas y poseen un conector jack de 3,5 mm para poder acoplarse al sensor descrito previamente.



Ilustración 76. Electrodo.

Es muy importante remarcar que la asignación correcta de los electrodos NO se corresponde con el código de colores que describe la web y el tutorial del fabricante [67] [68] (Ilustración 77), sino con el código de colores de la Ilustración 78.

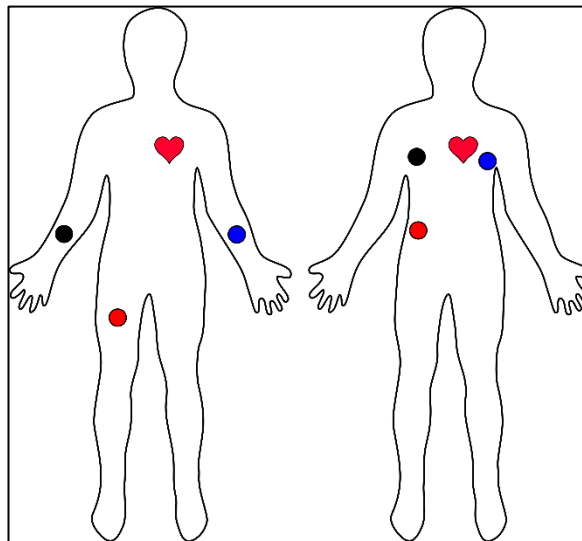


Ilustración 77. Colocación errónea de los electrodos.

Conexión CORRECTA de los electrodos:

Brazo izquierdo	Rojo
Brazo derecho	Negro
Pierna derecha	Azul

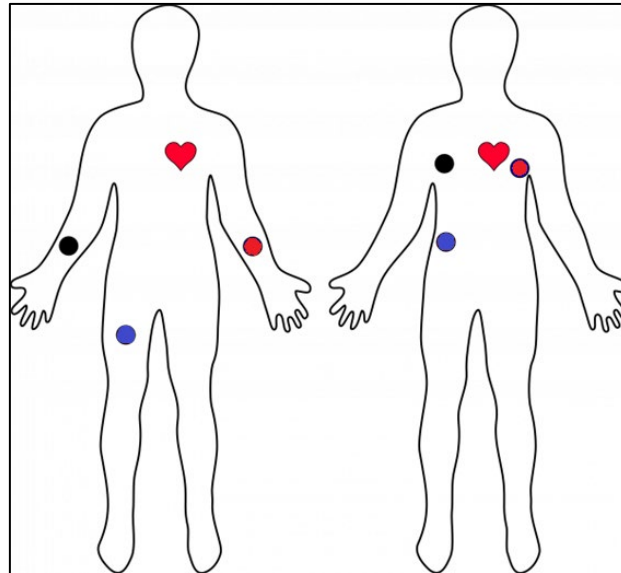


Ilustración 78. Colocación correcta de los electrodos.

Además, los electrodos deben conectarse a unos parches desechables que se adhieren a la superficie corporal del paciente, como los mostrados a continuación.



Ilustración 79. Parches desechables para electrodos.

Por último, si se realiza una prueba de testeo del sensor de SparkFun se puede comprobar en la siguiente imagen cómo se detecta la forma de onda de un electrocardiograma de forma satisfactoria, diferenciándose fácilmente los componentes del ECG sin mucho ruido eléctrico.

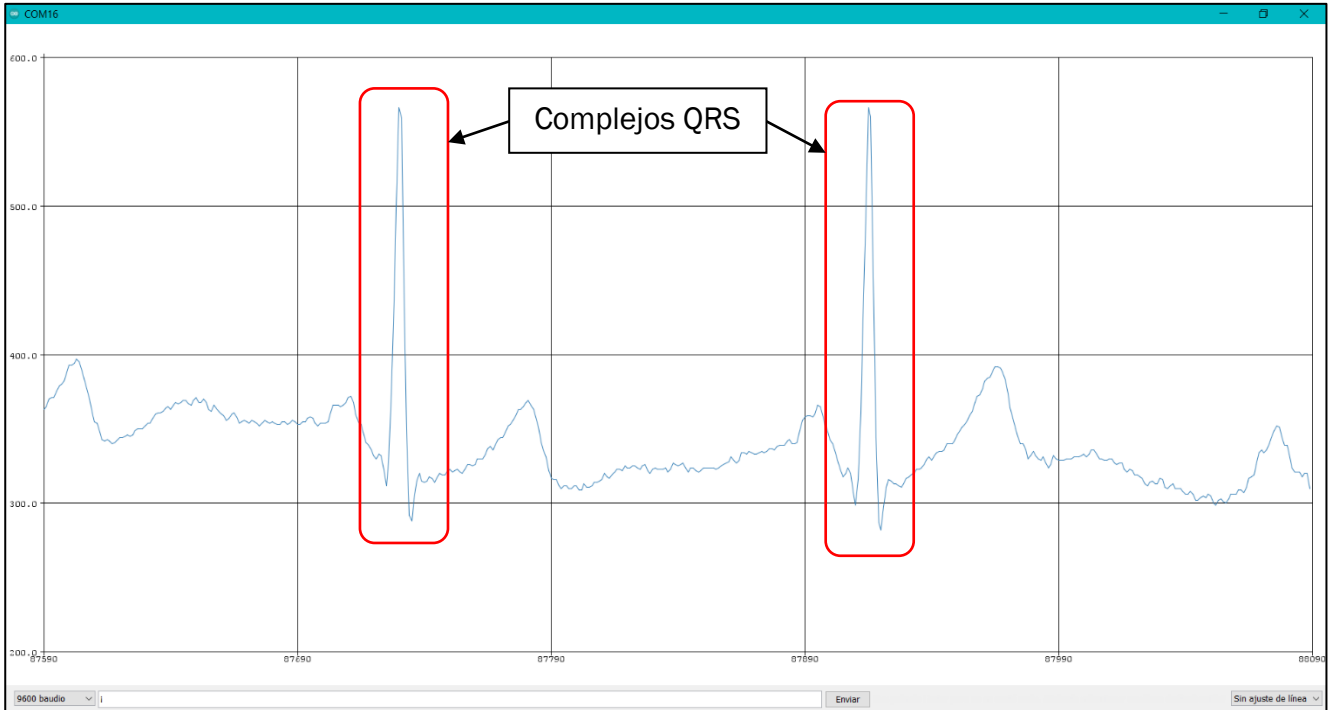


Ilustración 80. Gráfica de los valores detectados por el sensor.

3. Desarrollo y programación del proyecto

Una vez comprendido el funcionamiento de los microcontroladores F28069M y CC2650, y tras haber programado una serie de programas de prueba utilizando diferentes módulos como los GPIO, ADC, ePWM, I2C, BLE, las tareas de TI-RTOS y el tratamiento de las interrupciones; se van a desarrollar los 4 programas correspondientes a las especificaciones del trabajo definidas en el apartado de introducción y objetivos del proyecto.

3.1. Paso 1: F28069M_TX

Programa del primer módulo F28069M. Este microcontrolador se encarga de recibir las señales electrocardiográficas del sensor de Sparkfun, les aplica el algoritmo de Pan-Tomkins para hallar el valor del ritmo cardiaco o BMP (Beats Per Minute) y transmite dicho valor a un módulo CC2650 mediante comunicación I2C en modo rápido o 'fast' (400 kHz), trabajando como dispositivo esclavo en el bus.

Código significativo y explicación del programa:

Árbol del proyecto en Code Composer Studio (Ilustración 81).

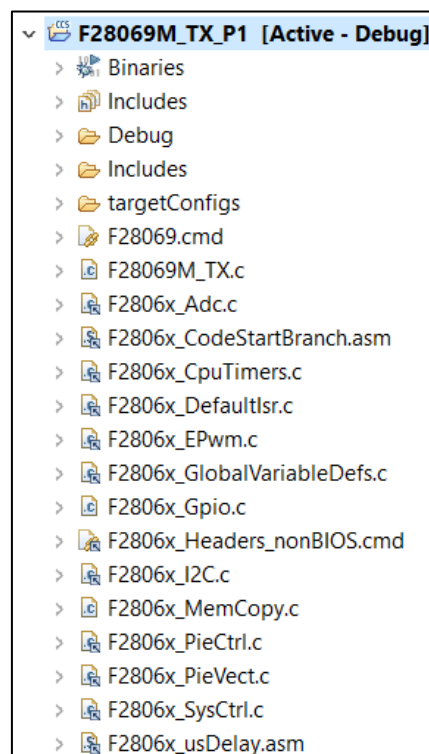


Ilustración 81. Árbol del proyecto "F28069M_TX_P1".

MÓDULO ADC

Se utiliza el módulo ADC (pin ADCINA4) para recibir los valores electrocardiográficos detectados por el sensor de Sparkfun (AD8232). Para ello se configura el ADC junto al módulo ePWM1 en la Ilustración 82 para que se genere una interrupción y poder muestrear la señal analógica a una frecuencia de 1 kHz.

```

202 void Config_ADC(void)
203 {
204     EALLOW;
205     AdcRegs.ADCCTL2.bit.ADCNONOVERLAP = 1; // Enable non-overlap mode
206
207     // ADCINT1 trips after AdcResults latch
208     AdcRegs.ADCCTL1.bit.INTPULSEPOS = 1;
209     AdcRegs.INTSEL1N2.bit.INT1E    = 1; // Enabled ADCINT1
210     AdcRegs.INTSEL1N2.bit.INT1CONT = 0; // Disable ADCINT1 Continuous mode
211
212     // Setup EOC1 to trigger ADCINT1 to fire. EOC es 'end of conversion'
213     AdcRegs.INTSEL1N2.bit.INT1SEL  = 1;
214     AdcRegs.ADCSOC1CTL.bit.CHSEL   = ECG_PIN; // set SOC0 channel select to ADCINA4
215
216     // Set SOC0 start trigger on EPWM1A. SOC es 'start of conversion'
217     AdcRegs.ADCSOC1CTL.bit.TRIGSEL = 5;
218
219     // Set SOC0 S/H Window to 7 ADC Clock Cycles, (6 ACQPS plus 1). S/H es 'sample and hold'
220     AdcRegs.ADCSOC1CTL.bit.ACQPS   = 6;
221     EDIS;
222
223     // Assumes ePWM1 clock is already enabled in InitSysCtrl();
224     EPwm1Regs.ETSEL.bit.SOCAEN = 1; // Enable SOC on A group
225     EPwm1Regs.ETSEL.bit.SOCASEL = 4; // Select SOC from CMPA on upcount
226     EPwm1Regs.ETPS.bit.SOCAPRD = 1; // Generate pulse on 1st event
227     EPwm1Regs.CMPA.half.CMPA = 0x0080; // Set compare A value. Valor del comparador A
228
229     // Para conseguir una frecuencia de muestreo de 1 kHz:
230     // En modo 'count up' f = SYSCLKOUT / ((TBPRD+1) * HSPCLKDIV * CLKDIV). SYSCLKOUT son 90 MHz
231     // f = 90 MHz / ((44999 + 1) * 2 * 1) = 1 kHz
232     EPwm1Regs.TBPRD = 0xAFC7; // Set period for ePWM1. Periodo del PWM
233     EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count Up and start. Selección de modo de funcionamiento
234     EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV2; // Clock ratio to SYSCLKOUT/2. Pre-escalador de alta precisión
235     EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1; // Clock ratio to SYSCLKOUT. Pre-escalador
236     return;
237 }

```

Ilustración 82. Código de la función "Config_ADC()".

La instrucción "AdcRegs.ADCSOC1CTL.bit.CHSEL = ECG_PIN" selecciona el pin analógico ADCINA4 como entrada de la señal analógica y "AdcRegs.ADCSOC1CTL.bit.TRIGSEL = 5" asigna la señal del módulo ePWM1 como disparador. La frecuencia de muestreo (1 kHz), se rige por la siguiente fórmula (para el modo de funcionamiento "up count") y se consigue asignando los valores de los factores en los registros correspondientes:

$$f = \frac{SYSCLKOUT}{(TBPRD + 1) * HSPCLKDIV * CLKDIV} = \frac{90 \cdot 10^6}{(44999 + 1) * 2 * 1} = 1000 \text{ Hz}$$

Una vez configurado el módulo ADC se programa la función que tratará la interrupción producida por el ADC (Ilustración 83). Así, cada vez que se produzca una interrupción (disparada por el módulo ePWM cada 1 ms), se leerá y almacenará en la variable "dato" el valor electrocardiográfico detectado por el sensor de Sparkfun.

```

297 /** Función que trata la interrupción producida por el módulo ADC */
298 __interrupt void adc_isr(void)
299 {
300     dato = AdcResult.ADCRESULT1; // Se almacena el dato enviado por el sensor de Sparkfun
301     AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; // Clear ADCINT1 flag reinitialize for next SOC
302     PieCtrlRegs.PIEACK.all = PIEACK_GROUP1; // Acknowledge interrupt to PIE
303     return;
304 }

```

Ilustración 83. Código de la función "adc_isr()".

MÓDULO I2C

Después, se inicializa el módulo I2C (código de la Ilustración 84) para funcionar como dispositivo esclavo (MST = 0) y estar listo para enviar información (TRX = 1) cuando el dispositivo maestro lo pida (XRDY = 1). Se utilizará el valor "0x05" como dirección I2C (registro "I2COAR").

La frecuencia de funcionamiento del módulo I2C serán 10 MHz, mientras que la frecuencia de la señal SCL serán 400 kHz (fast mode).

```

240 /** Configuración del módulo I2C como dispositivo esclavo que únicamente transmitirá */
241 /** mensajes pero no recibirá nada. Pines: SCL: P33 y SDA: P32. */
242 void I2C_Init(void)
243 {
244     // El módulo I2C debe trabajar a una frecuencia de entre 7 y 12 MHz
245     // f = SYSCLKOUT / (IPSC + 1) = 90 MHz / (8+1) = 10 MHz
246     // Por otro lado, la señal SCL que recibirá será de 400 kHz, por lo que hay que configurar los
247     // preescaladores. f = f(I2C) / ((ICCL + d) + (ICCH + d)) = 10 MHz / ((8+5)+(7+5)) = 400 kHz
248     // Nota: 'd' es 5 porque el valor de IPSC es mayor que 1.
249     I2caRegs.I2CPSC.all = 8; // Valor del pre-escalador del módulo I2C
250     I2caRegs.I2CCLKL = 7; // Valor del primer pre-escalador del pin SCL. NO debe ser cero
251     I2caRegs.I2CCLKH = 8; // Valor del segundo pre-escalador del pin SCL. NO debe ser cero
252     I2caRegs.I2CIER.all = 0x0; // Desabilita todas las interrupciones
253     I2caRegs.I2CMDR.all = 0x0020; // Reset del módulo I2C (lo mismo que I2caRegs.I2CMDR.bit.IRS = 1)
254
255     I2caRegs.I2COAR = I2C_SLAVE_ADDR; // Dirección I2C del microcontrolador
256     I2caRegs.I2CMDR.bit.FREE = 1; // El módulo I2C seguirá funcionando aunque haya un 'breakpoint'
257     I2caRegs.I2CMDR.bit.MST = 0; // Modo esclavo. La señal SCL será recibida desde un dispositivo maestro
258     I2caRegs.I2CMDR.bit.TRX = 1; // Funcionamiento en modo transmisor de mensajes a través del pin SDA
259     I2caRegs.I2CIER.bit.XRDY = 1; // Habilitación de la interrupción de 'listo para transmitir'
260     I2caRegs.I2CMDR.bit.IRS = 1; // Reset del módulo I2C para guardar los cambios realizados
261     return;
262 }

```

Ilustración 84. Código de la función "I2C_Init()".

La frecuencia del módulo I2C está gobernada por esta fórmula:

$$f_{I2C} = \frac{SYSCLKOUT}{IPSC + 1} = \frac{90 \cdot 10^6}{(8 + 1)} = 10 \text{ MHz}$$

Por ello, el valor del registro "I2CPSC" deberá ser 8 (línea 249).

Por otro lado, la frecuencia del bus I2C se corresponde con la siguiente fórmula:

$$f_{SCL} = \frac{f_{I2C}}{(ICCL + d) + (ICCH + d)} = \frac{10 \cdot 10^6}{(7 + 5) + (8 + 5)} = 400 \text{ kHz}$$

Como consecuencia, el valor del registro “I2CCLKL” deberá ser 7, el “I2CCLKH” será 8 y la constante “d” es 5 porque el valor del registro “I2CPSC” es mayor que 1 (consultar el manual técnico).

La última instrucción de la configuración se corresponde con el reinicio del módulo I2C para guardar y aplicar los cambios (“IRS = 1”).

Tras configurar el módulo I2C, se programa la función que tratará las interrupciones producidas por el microcontrolador maestro del bus, en la Ilustración 85.

```

307/** Función que trata la interrupción producida por el módulo I2C */
308 __interrupt void i2c_tx_isr(void)
309 {
310     int_source = I2caRegs.I2CISRC.all; // Se registra qué tipo de interrupción que se ha producido
311 // ++debug;
312     if(int_source == I2C_TX_ISRC) // Continúa si la interrupción ha sido 'datos listos para ser transmitidos'
313     {
314         I2caRegs.I2CCNT = 1; // Número de bytes a enviar
315         while(!I2caRegs.I2CSTR.bit.XRDY); // Se verifica que el bus está listo para transmitir el mensaje
316         I2caRegs.I2CDXR = bpm_media; // Se envía el mensaje
317     }
318
319     PieCtrlRegs.PIEACK.all |= PIEACK_GROUP8; // Acknowledge interrupt to PIE
320     return;
321 }

```

Ilustración 85. Código de la función "i2c_tx_isr()".

Esta función se encarga de transmitir el valor de la variable “bpm_media” (que es el valor resultante del algoritmo de Pan-Tomkins) al primer microcontrolador CC2650 cargando dicho valor en el registro “I2CDXR” si se detecta la interrupción “datos listos para ser transmitidos” o “XRDY”.

MÓDULO ePWM2

La función que ejecuta el algoritmo de Pan-Tomkins necesita de la existencia de un temporizador o contador de microsegundos. Para lograrlo se va a utilizar el módulo ePWM2 para generar una señal con una frecuencia de 1 MHz, la cual irá incrementando el valor de una variable que trabajará como contador de microsegundos cuando se produzca una interrupción.

De esta manera, lo primero que hay que hacer para poder configurar los parámetros de funcionamiento del módulo ePWM (Ilustración 86) es deshabilitar la sincronización de los temporizadores con la señal de reloj (registro “TBCLKSNC”) y al final volver a habilitarla.

A continuación, se configuran los registros que establecen la frecuencia de trabajo del módulo ePWM. Para lograr una frecuencia de 1 MHz, el valor del registro “TBPRD” será 89, los valores de ambos pre-escaldores, “CLKDIV” y “HSPCLKDIV”, será 1 y se trabajará en el modo “up count”. Además, será necesario habilitar las interrupciones.

```

267 void ePWM2_Timer(void)
268 {
269     EALLOW;
270     SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;    // Deshabilitación de la sincronización de los temporizadores con la señal de reloj
271     EDIS;
272
273     // Setup Sync
274     EPwm2Regs.TBCTL.bit.SYNCSEL = TB_SYNC_IN; // Pass through
275     // Habilitación de la sincronización de los temporizadores
276     EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE;
277     EPwm2Regs.TBPHS.half.TBPHS = 100;
278
279     // Para lograr f = 1 MHz en modo 'count up': f = SYSCLKOUT / ((TBPRD+1) * HSPCLKDIV * CLKDIV)
280     // f = 90 MHz / ((89+1) * 1 * 1) = 1 MHz
281     EPwm2Regs.TBPRD = 89;                    // Registro del valor del periodo
282     EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // Registro del valor del pre-escalador de alta precisión
283     EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;    // Registro del valor del pre-escalador
284
285     EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Modo de funcionamiento 'count up'
286     EPwm2Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO; // Habilitación de la interrupción en el 'evento cero'
287     EPwm2Regs.ETSEL.bit.INTEN = 1;           // Habilitación de las interrupciones
288     EPwm2Regs.ETPS.bit.INTPRD = ET_1ST;     // Se generará la interrupción en el primer evento
289
290     EALLOW;
291     SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;    // Habilitación de la sincronización de los temporizadores con la señal de reloj
292     EDIS;
293     return;
294 }

```

Ilustración 86. Código de la función “ePWM2_Timer()”.

Véase que como la señal del ePWM no va a ser utilizada fuera del microcontrolador no es necesario hacer la llamada a la función “InitEPwm2Gpio()”.

De esta manera, una vez configurado el módulo ePWM, se producirá una interrupción cada microsegundo (código de la función que trata las excepciones de este módulo en la Ilustración 87), haciendo que la variable “EPwm2TimerIntCount” incremente su valor.

```

324 /* Función que trata la interrupción producida por el módulo ePWM2 */
325 __interrupt void epwm2_timer_isr(void)
326 {
327     EPwm2TimerIntCount++;                    // Incremento del contador de microsegundos. T = 1/f = 1/1 MHz = 1 us
328     EPwm2Regs.ETCLR.bit.INT = 1;           // Clear INT flag for this timer
329     PieCtrlRegs.PIEACK.all = PIEACK_GROUP3; // Acknowledge this interrupt to receive more interrupts from group 3
330     return;
331 }

```

Ilustración 87. Código de la función “epw2_timer_isr()”.

EJECUCIÓN DEL PROGRAMA Y ALGORITMO DE PAN-TOMKINS

Por último se explicará el funcionamiento del programa general en el bucle “for()” infinito (Ilustración 88). En cada iteración se registra el momento actual de ejecución del programa mediante el uso del contador de microsegundos y se aplica el algoritmo de Pan-Tomkins para detectar si el dato registrado por el sensor pertenece al complejo QRS o no mediante la función “detect()” (Ilustración 89).

Si el punto del electrocardiograma sí pertenece al complejo QRS, se almacenará el valor en un buffer de BPM para posteriormente, y tras sucesivas iteraciones, calcular el valor medio y registrarlo como el valor de la resultante “bpm_media”, cuyo valor será enviado por el bus I2C cuando el microcontrolador CC2650 realice una interrupción.

La adaptación del algoritmo de Pan-Tomkins en este proyecto se basa en el programa para la plataforma Arduino de “blakeMilner” en GitHub [69].

```

currentMicros = EPwm2TimerIntCount; // Almacenamiento del momento actual (en microsegundos)

// Iterar si ha pasado un periodo de tiempo
if (currentMicros - previousMicros >= PERIOD)
{
    previousMicros = currentMicros; // Se almacena el momento previo
    bool QRS_detected = false; // Booleana que indica si se ha detectado un QRS
    int next_ecg_pt = dato; // Se lee el valor del electrocardiograma (ECG)

    QRS_detected = detect(next_ecg_pt); // Se envía el dato del ECG al algoritmo

    if(QRS_detected == true) // Continúa si el algoritmo detecta un punto QRS
    {
        foundTimeMicros = EPwm2TimerIntCount; // Se guarda el momento en el que se detectó el punto QRS actual

        // Se almacena el valor en BPM en el buffer
        bpm_buff[bpm_buff_WR_idx] = (60.0 / ((float) (foundTimeMicros - old_foundTimeMicros)) / 1000000.0));
        bpm_buff_WR_idx++;
        bpm_buff_WR_idx %= BPM_BUFFER_SIZE;

        tmp = bpm_buff_RD_idx - BPM_BUFFER_SIZE + 1;
        if(tmp < 0) tmp += BPM_BUFFER_SIZE;

        bpm_buff_RD_idx++;
        bpm_buff_RD_idx %= BPM_BUFFER_SIZE;

        // Cálculo del valor medio de BPM del buffer
        bpm_media = (bpm_buff[0]+bpm_buff[1]+bpm_buff[2]+bpm_buff[3]+bpm_buff[4]) / BPM_BUFFER_SIZE;

        old_foundTimeMicros = foundTimeMicros; // Se guarda el momento en el que se detectó el último punto QRS
    }
}

```

Ilustración 88. Código del bucle for infinito del "main()".

```

368 bool detect(float new_ecg_pt)
369 {
370     // Copy new point into circular buffer, increment index
371     ecg_buff[ecg_buff_WR_idx++] = new_ecg_pt;
372     ecg_buff_WR_idx %= (M+1);
373
374     /* High pass filtering */
375     if(number_iter < M)
376     {
377         // first fill buffer with enough points for HP filter
378         hp_sum += ecg_buff[ecg_buff_RD_idx];
379         hp_buff[hp_buff_WR_idx] = 0;
380     }
381     else
382     {

```

Ilustración 89. Fragmento de la función "detect()".

3.2. Paso 2: CC2650_Peripheral_BLE

Programa del primer módulo CC2650. Este segundo microcontrolador del proyecto se encarga de recibir el valor del ritmo cardiaco a través del bus I2C, trabajando como dispositivo maestro a una velocidad de transferencia de 400 kbps, y enviar dicho dato a un segundo módulo CC2650 a través del protocolo BLE. En este caso, el microcontrolador trabajará en el rol de dispositivo periférico o esclavo, implementando la característica 'Heart Rate Measurement', contenida en el servicio 'Heart Rate', el cual es un servicio GATT predefinido por Bluetooth SIG [70], en el que sus UUIDs constan de 16 bits en vez de los 128 estándar [71] [72].

Es de vital importancia tener en cuenta que el primer octeto de la característica 'Heart Rate Measurement', correspondiente al atributo 'Heart Rate Value Format bit' debe ser 0 si el tipo de dato que almacena el ritmo cardiaco es de 8 bits (como es este caso) o debe ser 1 si el dato se guarda en un contenedor de 16 bits. Además, el atributo de la característica que guarda el valor del ritmo cardiaco es 'Heart Rate Measurement Value Field' [73].

El UUID del servicio GATT "Heart Rate Service" es 0x180D, el UUID de la característica "Heart Rate Measurement" es 0x2A37 y, por último, el identificador de las unidades BPM (beats per minute) es 0x27AF.

Para desarrollar este programa se ha tomado como base de trabajo el ejemplo "Project Zero" de SimpleLink Academy presentado en el apartado de programas de prueba con el microcontrolador CC2650, por lo que la imagen "Stack" del programa permanecerá invariable y la parte "App" se modificará.

Código significativo y explicación de la imagen “App” del programa:

Árbol del proyecto en Code Composer Studio (Ilustración 90).

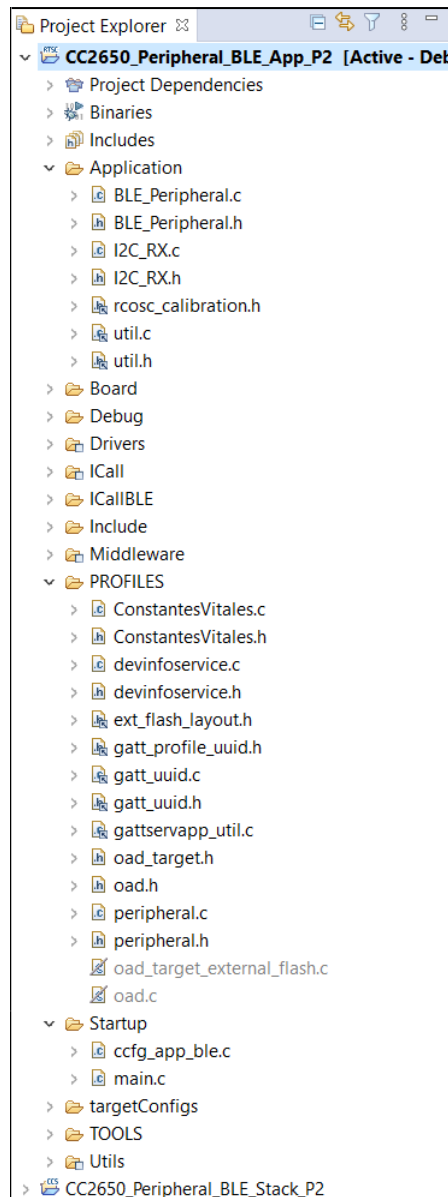
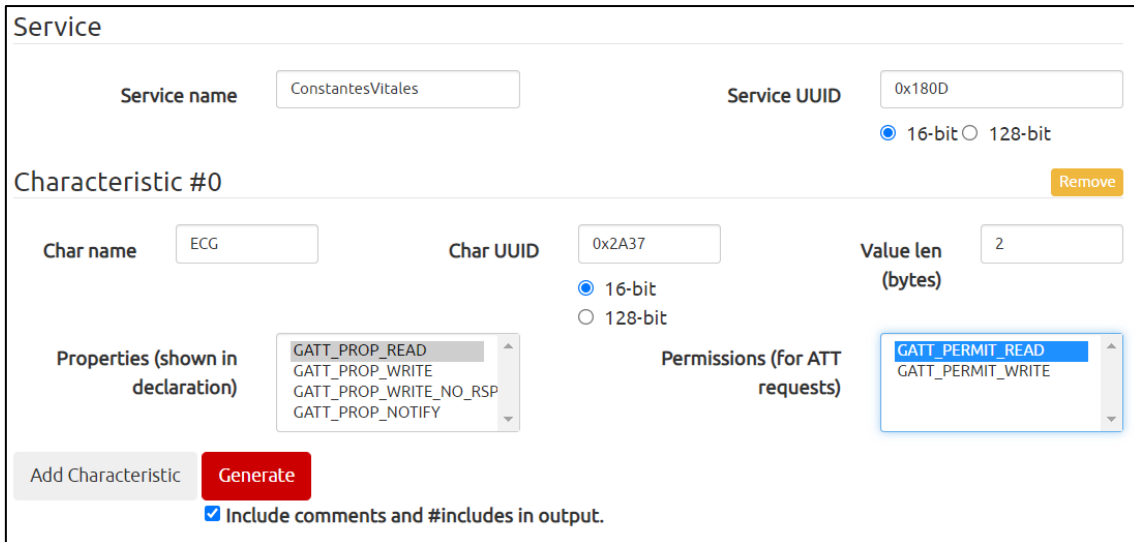


Ilustración 90. Árbol del proyecto "CC250_Peripheral_BLE_P2".

SERVICIO “HEART RATE”

El primer paso consiste en eliminar y borrar todo el código y ficheros pertenecientes a los servicios del ejemplo: LED Service, BUTTON Service y Data Service. A continuación, y con la ayuda de los tutoriales de SimpleLink Academy [74] [75] [76], se implementará el servicio GATT “Heart Rate Service”, los cuales poseen una herramienta de creación de servicios personalizados.

Utilizando dicha herramienta (Ilustración 91), se introduce el nombre del servicio a implementar, su identificador (de 16 o 128 bits) y se definen el número de características que se quieren incluir. En este caso solo se implementará una característica, y por ende será necesario introducir su nombre, identificador (de 16 o 128 bits), la longitud de la característica y sus propiedades y permisos. Los UUIDs serán los definidos previamente (ambos de 16 bits) las propiedades y permisos serán únicamente de lectura, tal y como muestra la siguiente ilustración.



The screenshot shows a web-based interface for generating Bluetooth service definitions. It is divided into two main sections: 'Service' and 'Characteristic #0'.
 In the 'Service' section, the 'Service name' is 'ConstantesVitales' and the 'Service UUID' is '0x180D'. There are radio buttons for '16-bit' (selected) and '128-bit'.
 In the 'Characteristic #0' section, the 'Char name' is 'ECG', the 'Char UUID' is '0x2A37', and the 'Value len (bytes)' is '2'. There are also radio buttons for '16-bit' (selected) and '128-bit'.
 Below these fields are two dropdown menus: 'Properties (shown in declaration)' with options GATT_PROP_READ, GATT_PROP_WRITE, GATT_PROP_WRITE_NO_RSP, and GATT_PROP_NOTIFY; and 'Permissions (for ATT requests)' with options GATT_PERMIT_READ and GATT_PERMIT_WRITE.
 At the bottom, there is an 'Add Characteristic' button, a red 'Generate' button, and a checked checkbox for 'Include comments and #includes in output'.

Ilustración 91. Captura de la herramienta generadora de servicios.

Una vez generado el servicio personalizado, se copiará el código de “your-service.h” en un nuevo fichero que se llamará “ConstantesVitales.h”, localizado en la carpeta “PROFILES”. Del mismo modo, el contenido de “your-service.c” se incluirá en un nuevo fichero “ConstantesVitales.c” ubicado en la misma carpeta.

Dentro del fichero “ConstantesVitales.h” se localizan todas las constantes y las declaraciones de las funciones de la API, tal y como muestra la Ilustración 92.

```

55 * CONSTANTS
56 */
57 // Service UUID
58 #define CONSTANTESVITALES_SERV_UUID 0x180D
59
60 // Characteristic defines
61 #define CONSTANTESVITALES_ECG_ID 0
62 #define CONSTANTESVITALES_ECG_UUID 0x2A37
63 #define CONSTANTESVITALES_ECG_LEN 2

```

Ilustración 92. Constantes definidas en “ConstantesVitales.h”.

Por otro lado, en el fichero “ConstantesVitales.c”, donde se ubica la definición y programación de todas las funciones que implementan el servicio “Heart Rate”, será necesario incluir las siguientes instrucciones (Ilustración 93 e Ilustración 94):

```
114 #include "ConstantesVitales.h"
115 #include <ti/sysbios/knl/Event.h> // Para usar eventos como método de comunicación entre procesos
```

Ilustración 93. Instrucción que añadir en "ConstantesVitales.c" (I).

```
133 // Se añade una referencia externa al manipulador de eventos localizado en I2C_RX.c
134 extern Event_Handle readEvent;
```

Ilustración 94. Instrucción que añadir en "ConstantesVitales.c" (II).

Estas instrucciones se encargan de habilitar el uso de eventos como método de comunicación entre los procesos de comunicación del bus I2C y de la modificación del atributo de la característica del servicio “Heart Rate”.

Por otro lado, dentro de la función “ConstantesVitales_ReadAttrCB()” (Ilustración 95) será necesario realizar la publicación de un evento con la instrucción “Event_post()” para indicar al proceso de comunicación I2C que el atributo de la característica ya ha sido leído por el dispositivo “central” o maestro conectado al microcontrolador CC2650. En este caso, el funcionamiento del evento es equivalente al levantamiento de un semáforo.

```
else
{
    // Se trabaja con eventos para realizar la comunicación entre procesos. En este caso,
    // se publica el evento Id_00 para indicar que el atributo de la característica ha sido leído
    // Este funcionamiento provoca que la primera vez que se lea el atributo de la característica
    // su valor será 0 porque todavía no se habrá producido la comunicación I2C
    Event_post(readEvent, Event_Id_00);
    *pLen = MIN(maxLen, CONSTANTESVITALES_ECG_LEN - offset); // Transmit as much as possible
    memcpy(pValue, pAttr->pValue + offset, *pLen);
}
```

Ilustración 95. Instrucción que añadir en "ConstantesVitales.c" (III).

BLE Peripheral

Los ficheros “ProjectZero.c” y “ProjectZero.h”, localizados en la carpeta “Application” serán renombrados a “BLE_Peripheral” (.c y .h respectivamente). Además, el fichero “.c” sufrirá las siguientes modificaciones (además de las eliminaciones de los 3 servicios de prueba mencionadas anteriormente):

- ❖ Inclusión del fichero de cabecera del servicio “Heart Rate” (Ilustración 96).

```
87 #include "ConstantesVitales.h"
```

Ilustración 96. Modificación (I) en "BLE_Peripheral.c".

- ❖ Modificación del nombre de dispositivo anunciante de servicios dentro del array “static uint8_t advertData[]” (Ilustración 97).

```

201 // Complete name
202 15,
203 GAP_ADTYPE_LOCAL_NAME_COMPLETE,
204 'R', 'i', 't', 'm', 'o', ' ', 'C', 'a', 'r', 'd', 'i', 'a', 'c', 'o',

```

Ilustración 97. Modificación (II) en "BLE_Peripheral.c".

- ❖ Cambio del nombre del dispositivo (Ilustración 98).

```

207// GAP GATT Attributes
208static uint8_t attDeviceName[GAP_DEVICE_NAME_LEN] = "Ritmo Cardiaco";

```

Ilustración 98. Modificación (II) en "BLE_Peripheral.c".

- ❖ Se añade el servicio “Heart Rate” al servidor GATT dentro de la función “ProjectZero_init()” (Ilustración 99).

```

379 // Add services to GATT server and give ID of this task for Indication acks.
380 ConstantesVitales AddService( selfEntity );

```

Ilustración 99. Modificación (III) en "BLE_Peripheral.c".

TAREA DE EJECUCIÓN DEL PROTOCOLO I2C

Una vez implementado el servicio “Heart Rate” y configurado toda la parte del protocolo BLE, se va a introducir la tarea de ejecución del protocolo I2C con la creación de los ficheros “I2C_RX.c” e “I2C_RX.h” dentro de la carpeta “Application”. El código de este hilo de ejecución se basa en la explicación del tutorial de SimpleLink Academy y en el ejemplo “tmp007”.

Primero se declara un array de dos números enteros sin signo de 8 bits (véase la Ilustración 100). El primer número siempre será 0, pues será el valor del atributo “Heart Rate Value Format bit”, indicando que el atributo “Heart Rate Measurement Value” (el segundo número del array) será de 8 bits.

Después se declara el evento “readEvent” para posibilitar la comunicación entre los hilos de ejecución del protocolo BLE y el I2C.

```

111// Declaración del array de enteros sin signo de 8 bits que almacenará en
112// su primera posición siempre un 0 (debido a que el valor de BPM se guarda
113// en un contenedor de 8 bits. La segunda posición del array guarda el valor
114// del ritmo cardiaco recibido por el bus I2C
115uint8_t msg_recibido[2] = {0, 0};
116
117// Se declara el evento para realizar la comunicación entre procesos
118Event_Handle readEvent;
119Error_Block eb;

```

Ilustración 100. Código ubicado en el fichero "I2C_RX.c".

La función “I2C_RX_createTask()” (Ilustración 101) se responsabiliza de la creación de la tarea que ejecuta el funcionamiento de la comunicación I2C.

```

143 void I2C_RX_createTask(void)
144 {
145     Task_Params taskParams;
146
147     // Create the task for the state machine
148     Task_Params_init(&taskParams);
149     taskParams.stack = I2C_RX_TaskStack;
150     taskParams.stackSize = I2C_RX_TASKSTACKSIZE;
151     taskParams.priority = I2C_RX_TASKPRIORITY;
152
153     Task_construct(&I2C_RX_Task, I2C_RX_TaskFxn, &taskParams, NULL);
154 }

```

Ilustración 101. Código de la función “I2C_RX_createTask()”.

Esta tarea únicamente ejecutará una función, “I2C_RX_TaskFxn()” (Ilustración 102 e Ilustración 103), que será la encargada de realizar la comunicación a través del protocolo I2C con el primer microcontrolador F28069M, trabajando en modo maestro y recibiendo el valor del ritmo cardiaco a una velocidad de transmisión de 400 kbps [77] [78].

```

167 static void I2C_RX_TaskFxn(UArg a0, UArg a1)
168 {
169     //     uint8_t         txBuffer[1];
170     uint8_t         rxBuffer[1];
171     I2C_Handle       i2c;
172     I2C_Params       i2cParams;
173     I2C_Transaction i2cTransaction;
174     UInt key;
175
176     I2C_init(); // Inicialización del módulo I2C
177
178     /* Create I2C for usage */
179     I2C_Params_init(&i2cParams);
180     i2cParams.bitRate = I2C_400kHz; // Velocidad de transmisión
181     i2c = I2C_open(Board_I2C, &i2cParams);
182
183     if (i2c == NULL)
184     {
185         System_abort("****ERROR*** >> FAILED TO OPEN I2C PORT");
186     }
187
188     Error_init(&eb);
189     /* Default instance configuration params */
190     // Se inicializa el evento de comunicación entre procesos
191     readEvent = Event_create(NULL, &eb);
192     if (readEvent == NULL)
193     {
194         System_abort("****ERROR*** >> EVENT CREATE FAILED");
195     }

```

Ilustración 102. Código de la función “I2C_RX_TaskFxn()” (I).

```

i2cTransaction.slaveAddress = 0x05; // Dirección I2C del dispositivo esclavo
i2cTransaction.writeBuf = NULL; // Datos a enviar (ninguno)
i2cTransaction.writeCount = 0; // Número de bytes a enviar
i2cTransaction.readBuf = rxBuffer; // Datos a recibir
i2cTransaction.readCount = 1; // Número de bytes a recibir

```

Ilustración 103. Código de la función “I2C_RX_TaskFxn()” (II).

En esta tarea se inicializa y configura el módulo I2C del microcontrolador, estableciendo la velocidad de transmisión, la dirección del dispositivo esclavo y las estructuras de datos a enviar y recibir a través del bus.

Luego, dentro del bucle infinito (Ilustración 104), se esperará a la publicación del evento definido en el apartado “BLE Peripheral” para establecer la comunicación a través del protocolo I2C y recibir el valor del ritmo cardiaco y almacenarlo en la segunda posición del array “msg_recibido”. Antes de pasar a la siguiente iteración, se llama a la función “ConstantesVitales_SetParameter()” para modificar el valor del atributo de la característica “Heart Rate Measurement” con el nuevo valor del ritmo cardiaco recién recibido por el bus I2C.

```

while (1)
{
    // Se espera a que se publique el evento (véase en 'ConstantesVitales.c')
    Event_pend(readEvent, Event_Id_NONE, Event_Id_00, BIOS_WAIT_FOREVER);

    // Se desarrolla la comunicación por el bus I2C entre los dispositivos
    if (I2C_transfer(i2c, &i2cTransaction))
    {
        key = Hwi_disable();

        // Se guarda el mensaje recibido (BPM) en el segundo valor del array
        msg_recibido[1] = rxBuffer[0];

        Hwi_restore(key);
    }

    // Después, se modifica el valor del atributo de la característica 'Heart Rate Measurement'
    // Tal y como se explicó antes, el primer octeto siempre es 0 para indicar que el valor de
    // BPM se almacena en un tipo de variable de 8 bits, siendo este byte el segundo octeto del
    // atributo de la característica y el correspondiente a msg_recibido[1].
    ConstantesVitales_SetParameter( CONSTANTESVITALES_ECG_ID, CONSTANTESVITALES_ECG_LEN, &msg_recibido );
}

```

Ilustración 104. Código de la función “I2C_RX_TaskFxn()” (III).

La razón por la cual los módulos CC2650 trabajan como dispositivos maestros en la comunicación I2C y no como esclavos es porque las funciones de la API únicamente permiten trabajar como dispositivo maestro. Si estos microcontroladores trabajaran como esclavos, harían el desarrollo del programa un grado más complejo, ya que sería necesario bajar el nivel de abstracción en la programación, y como la configuración de los microcontroladores F28069M como maestros o esclavos posee el mismo nivel de dificultad se ha decidido que los CC2650 sean maestros y los dispositivos F28069M esclavos.

MAIN

Por último, en el fichero “main.c” de la carpeta “Startup” se añade la inclusión del fichero “I2C_RX.h” (Ilustración 105) y dentro de la función “main()” se llama a la función que crea la tarea de ejecución del protocolo I2C “I2C_RX_createTask()” (Ilustración 106).

```
133 // Se incluye el fichero donde se ubica la tarea que ejecuta el protocolo de comunicación I2C
134 #include <I2C_RX.h>
```

Ilustración 105. Modificación (I) en "main.c".

```
209 // Se llama a la función que crea la tarea de ejecución del protocolo I2C
210 I2C_RX_createTask();
```

Ilustración 106. Modificación (II) en "main.c".

3.3. Paso 3: CC2650_Central_BLE

Programa del segundo módulo CC2650. Este microcontrolador funciona en el rol de dispositivo maestro dentro del protocolo BLE, con el objetivo de buscar, descubrir y conectarse automáticamente al primer microcontrolador CC2650 para leer la característica 'Heart Rate Measurement' del servicio 'Heart Rate' cuando el usuario pulse la señal digital de entrada BUTTON1 (botón derecho de la PCB del microcontrolador).

Únicamente se almacenará el segundo octeto de la característica, el cual corresponde al atributo “Heart Rate Measurement Value”, donde se localiza el valor del ritmo cardiaco, que es el dato que se quiere enviar al último microcontrolador del proyecto a través del bus I2C en modo maestro.

Este programa se basa en el ejemplo de SimpleLink 'simple_central', explicado en el apartado de programas de prueba del microcontrolador CC2650 y se ha desarrollado con la ayuda del foro de Texas Instruments [79]. De esta manera, y al igual que con el programa para el primer módulo CC2650, la imagen “Stack” del programa permanecerá invariable y la parte “App” se modificará.

Código significativo y explicación de la imagen “App” del programa:

Árbol del proyecto en Code Composer Studio (Ilustración 107).

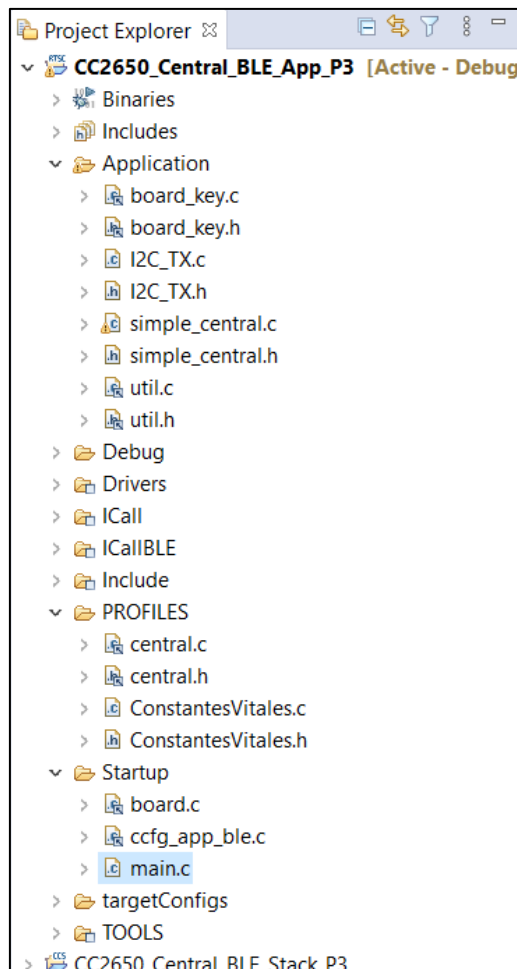


Ilustración 107. Árbol del proyecto "CC2650_Central_BLE_P3".

SERVICIO “HEART RATE”

Se añade una copia de los ficheros desarrollados en el programa anterior, “ConstantesVitales.c” y “ConstantesVitales.h”, en la carpeta “PROFILES”.

BLE Central

En el fichero “simple_central.c”, ubicado en la carpeta “Applications” se deberá modificar e incluir las siguientes líneas de código:

- ❖ Se añade el módulo de eventos para habilitar la comunicación entre procesos (Ilustración 108).

```
135 // Se incluye el módulo de eventos de TI-RTOS
136 #include <ti/sysbios/knl/Event.h>
137
138 // Se añade una referencia externa al manipulador de eventos localizado en I2C_TX.c
139 extern Event_Handle readEvent;
```

Ilustración 108. Modificación (I) del fichero "simple_central.c".

- ❖ Se cambia el valor de la constante booleana “DEFAULT_LINK_WHITE_LIST” para utilizar la lista blanca de direcciones predefinidas de dispositivos para conectar el microcontrolador central al periférico automáticamente (Ilustración 109).

```
170 // TRUE to use white list during discovery
171 #define DEFAULT_DISCOVERY_WHITE_LIST FALSE
172
173 // TRUE to use high scan duty cycle when creating link
174 #define DEFAULT_LINK_HIGH_DUTY_CYCLE FALSE
175
176 // TRUE to use white list when creating link
177 // Cambiar de FALSE a TRUE
178 #define DEFAULT_LINK_WHITE_LIST TRUE//FALSE
```

Ilustración 109. Modificación (II) del fichero "simple_central.c".

- ❖ Se cambia el valor de la constante booleana (Ilustración 110) “DEFAULT_DEV_DISC_BY_SVC_UUID” con el objetivo de descubrir solamente los dispositivos que implementan el servicio que nos interese (en este caso el “Heart Rate”).

```
232 // TRUE to filter discovery results on desired service UUID
233 // Se cambia a FALSE si se quiere buscar todos los dispositivos BLE en modo 'advertising'
234 // que se encuentren dentro del rango de alcance. Sin embargo, en este proyecto el dispositivo
235 // central buscará y se conectará únicamente al dispositivo BLE 'peripheral' de Ritmo Cardíaco
236 #define DEFAULT_DEV_DISC_BY_SVC_UUID TRUE //FALSE
```

Ilustración 110. Modificación (III) del fichero "simple_central.c".

- ❖ Modificación para la legibilidad del código (Ilustración 111). Se sustituye “MENU_ITEM_READ_WRITE” por “MENU_ITEM_READ”, ya que solo va a ser posible leer la característica del servicio.

```

288 // Menu item state
289 typedef enum {
290     MENU_ITEM_CONN_PARAM_UPDATE,
291     MENU_ITEM_RSSI,
292     MENU_ITEM_READ,           // Únicamente se va a leer, no a leer y escribir
293     MENU_ITEM_DISCONNECT
294 } MenuItem_t;
  
```

Ilustración 111. Modificación (IV) del fichero "simple_central.c".

- ❖ Nombre del dispositivo (Ilustración 112).

```

364 // GAP GATT Attributes
365 static const uint8_t attDeviceName[GAP_DEVICE_NAME_LEN] = "Simple BLE Central";
  
```

Ilustración 112. Modificación (V) del fichero "simple_central.c".

- ❖ Se comenta la variable booleana que permite escribir el valor de la característica (Ilustración 113).

```

402 // Value read/write toggle
403 //static bool doWrite = FALSE;
  
```

Ilustración 113. Modificación (VI) del fichero "simple_central.c".

- ❖ Dentro de la función “SimpleBLECentral_init()” (Ilustración 114) se introduce la dirección Bluetooth del dispositivo esclavo o periférico, la cual se deberá conocer de antemano y se añade dicha dirección a la “White List” después de haber sido vaciada.

Es muy importante remarcar que la dirección Bluetooth debe ser introducida al revés, con el formato Little Endian, es decir, los bits están ordenados del menos significativo al más significativo.

```

// Añadimos la dirección Bluetooth del dispositivo periférico al que nos queremos conectar (debemos conocerlo de antemano),
// pero si no la conocemos, dentro de la función ProjectZero_init() del programa del dispositivo periférico se puede modificar
// su dirección con la función HCI_EXT_SetBDADDRcmd(), como por ejemplo:
// uint8 bdAddr[] = {0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA};
// HCI_EXT_SetBDADDRcmd(bdAddr);
// Es muy importante que aquí la dirección Bluetooth se introduzca al revés
uint8 bdPeerAddress[B_ADDR_LEN] = {0x83, 0xB6, 0x9A, 0x2D, 0x07, 0x98}; // Dirección a derechas: {0x98, 0x07, 0x2D, 0x9A, 0xB6, 0x83};
HCI_LE_ClearWhitelistCmd(); // Se borran todas las direcciones Bluetooth previas en la White List
HCI_LE_AddWhitelistCmd(ADDRTYPE_PUBLIC, bdPeerAddress); // Se añade la dirección Bluetooth de nuestro dispositivo periférico en la White List
  
```

Ilustración 114. Fragmento de código de la función "SimpleBLECentral_init()".

- ❖ En la función “SimpleBLECentral_processRoleEvent()” (Ilustración 115 e Ilustración 116) se comentan las siguientes líneas de código y se introducen unas nuevas con el objetivo de lograr el establecimiento automático de la conexión BLE buscando el identificador del servicio “Heart Rate”.

```

833 case GAP_DEVICE_INFO_EVENT:
834 {
835     // Se comenta el siguiente código porque vamos a conectarnos directamente al dispositivo periférico
836     // Find peer device address by UUID
837     if ( (DEFAULT_DEV_DISC_BY_SVC_UUID == FALSE) ||
838         SimpleBLECentral_findSvcUuid(CONSTANTESVITALES_SERV_UUID,
839                                     pEvent->deviceInfo.pEvtData,
840                                     pEvent->deviceInfo.dataLen))
841     {
842         SimpleBLECentral_addDeviceInfo(pEvent->deviceInfo.addr,
843                                       pEvent->deviceInfo.addrType);
844     }
845
846     // Check if the discovered device is already in scan results
847     uint8_t i;
848     for (i = 0; i < scanRes; i++)
849     {
850         if (memcmp(pEvent->deviceInfo.addr, devList[i].addr , B_ADDR_LEN) == 0)
851         {
852             // Check if pEventData contains a device name
853             if (SimpleBLECentral_findLocalName(pEvent->deviceInfo.pEvtData,
854                                               pEvent->deviceInfo.dataLen))
855             {
856                 // Update deviceInfo entry with the name
857                 SimpleBLECentral_addDeviceName(i, pEvent->deviceInfo.pEvtData,
858                                               pEvent->deviceInfo.dataLen);
859             }
860         }
861     }

```

Ilustración 115. Fragmento de código (I) de la función “SimpleBLECentral_processRoleEvent()”.

```

// if filtering device discover results based on service UUID
if (DEFAULT_DEV_DISC_BY_SVC_UUID == TRUE)
{
    // Se busca el UUID del Servicio de ConstantesVitales.h
    if(SimpleBLECentral_findSvcUuid(CONSTANTESVITALES_SERV_UUID, pEvent->deviceInfo.pEvtData, pEvent->deviceInfo.dataLen))
    {
        SimpleBLECentral_addDeviceInfo(pEvent->deviceInfo.addr, pEvent->deviceInfo.addrType);
    }
}

```

Ilustración 116. Fragmento de código (II) de la función “SimpleBLECentral_processRoleEvent()”.

- ❖ Ahora, dentro de la función “SimpleBLECentral_handleKeys()” (Ilustración 117, Ilustración 118, Ilustración 119 e Ilustración 120) se comentan todas las líneas de código correspondientes a la escritura del valor de la característica del servicio.

```

case MENU_ITEM_RSSI:
    selectedMenuItem = MENU_ITEM_READ;
    Display_print0(dispatchHandle, ROW_SEVEN, 0, ">Read req");
    break;

```

Ilustración 117. Fragmento de código (I) de la función “SimpleBLECentral_handleKeys()”.

```

case MENU_ITEM_READ:
  if (state == BLE_STATE_CONNECTED &&
      charHdl != 0 &&
      procedureInProgress == FALSE)
  {
    uint8_t status;
    // Como únicamente se requiere leer la característica y no escribir,
    // todo el código referido a la escritura está comentado.
    // Do a read or write as long as no other read or write is in progress
    if (doWrite)
    {
      // Do a write
      attWriteReq_t req;
      req.pValue = GATT_bm_alloc(connHandle, ATT_WRITE_REQ, 1, NULL);
      if ( req.pValue != NULL )
      {
        Display_print0(dispHandle, ROW_SIX, 0, "Write req sent");
        req.handle = charHdl;
        req.len = 1;
        req.pValue[0] = charVal;
        req.sig = 0;
        req.cmd = 0;
        status = GATT_WriteCharValue(connHandle, &req, selfEntity);
        if ( status != SUCCESS )
        {
          GATT_bm_free((gattMsg_t *)&req, ATT_WRITE_REQ);
        }
      }
    }
  }

```

Ilustración 118. Fragmento de código (II) de la función "SimpleBLECentral_handleKeys()".

```

    else
    {
      status = bleMemAllocError;
    }
  }
  else
  {
    // Do a read
    attReadReq_t req;
    req.handle = charHdl;
    status = GATT_ReadCharValue(connHandle, &req, selfEntity);
    Display_print0(dispHandle, ROW_SIX, 0, "Read req sent");
  }

```

Ilustración 119. Fragmento de código (III) de la función "SimpleBLECentral_handleKeys()".

```

    if (status == SUCCESS)
    {
      procedureInProgress = TRUE;
      doWrite = !doWrite;
    }
  }
  break;

```

Ilustración 120. Fragmento de código (IV) de la función "SimpleBLECentral_handleKeys()".

- ❖ En la función “SimpleBLECentral_processGATTMsg()” (Ilustración 121), donde se leerá el segundo byte de la característica, el correspondiente al atributo “Heart Rate Measurement Value”, se almacena dicho valor en la variable “dato_ble” y se guarda en una variable global con la función “recibe_BLE()” para ser enviado posteriormente por el módulo I2C del microcontrolador. Además, será necesario publicar el evento “readEvent” para indicar al hilo de ejecución de la tarea del I2C que la variable ya ha sido leída y almacena y que está lista para ser enviada.

```

else
{
    // After a successful read, display the read value
    // Se lee el segundo octeto de la característica, como se ha mencionado al principio
    // y se almacena en la variable 'bpm'
    uint8_t bpm = pMsg->msg.readRsp.pValue[1];
    // Se muestra el valor del ritmo cardiaco en el monitor serie
    Display_print1(dispatchHandle, ROW_SIX, 0, "Read BPM: %d", bpm);
    // El valor del ritmo cardiaco se guarda en una variable global para ser enviado
    // posteriormente por el módulo I2C del microcontrolador
    recibe_BLE(&bpm);
    // Se trabaja con eventos para realizar la comunicación entre procesos. En este caso,
    // se publica el evento Id_00 para indicar que la variable global 'dato_ble' ya almacena
    // el valor del ritmo cardiaco recibido por Bluetooth. En este caso, el evento funcionaria
    // de una manera casi idéntica a un semáforo.
    Event_post(readEvent, Event_Id_00);
}

```

Ilustración 121. Fragmento de código de la función “SimpleBLECentral_processGATTMsg()”.

- ❖ Por último, en la función “SimpleBLECentral_processGATTDiscEvent()” (Ilustración 122 e Ilustración 123) hay que introducir el identificador del servicio “Heart Rate” para que el dispositivo BLE pueda buscarlo entre el resto de servicios del dispositivo periférico cuando se realice la conexión (aunque en este caso solo hay un servicio).

```

1633 static void SimpleBLECentral_processGATTDiscEvent(gattMsgEvent_t *pMsg)
1634 {
1635     if (discState == BLE_DISC_STATE_MTU)
1636     {
1637         // MTU size response received, discover simple BLE service
1638         if (pMsg->method == ATT_EXCHANGE_MTU_RSP)
1639         {
1640             // UUID del Heart Rate Service
1641             uint8_t uuid[ATT_BT_UUID_SIZE] = { LO_UINT16(CONSTANTESVITALES_SERV_UUID),
1642                                               HI_UINT16(CONSTANTESVITALES_SERV_UUID) };

```

Ilustración 122. Fragmento de código (I) de la función “SimpleBLECentral_processGATTDiscEvent()”.

```

if (svcStartHdl != 0)
{
    attReadByTypeReq_t req;

    // Discover characteristic
    discState = BLE_DISC_STATE_CHAR;

    req.startHandle = svcStartHdl;
    req.endHandle = svcEndHdl;
    req.type.len = ATT_BT_UUID_SIZE;
    // UUID de la característica Heart Rate Measurement
    req.type.uuid[0] = LO_UINT16(CONSTANTESVITALES_ECG_UUID);
    req.type.uuid[1] = HI_UINT16(CONSTANTESVITALES_ECG_UUID);

    VOID GATT_ReadUsingCharUUID(connHandle, &req, selfEntity);
}

```

Ilustración 123. Fragmento de código (II) de la función "SimpleBLECentral_processGATTDiscEvent()".

TAREA DE EJECUCIÓN DEL PROTOCOLO I2C

Tras configurar toda la parte del protocolo BLE, se va a introducir la tarea de ejecución del protocolo I2C, al igual que en el programa anterior. Se crearán los ficheros "I2C_TX.c" e "I2C_TX.h" dentro de la carpeta "Application". El código de este hilo de ejecución se basa en la explicación del tutorial de SimpleLink Academy y en el ejemplo "tmp007". Como la programación de estos ficheros es idéntica a los del programa anterior tan sólo se mostrarán las diferencias entre uno y otro.

Primero, en el fichero "I2C_TX.h" (Ilustración 124), se añadirá la declaración de la función externa "recibe_BLE()" para que pueda ser llamada en el fichero "simple_central.c".

```

35 * Task creation function for the I2C_TX writing task.
36 */
37 extern void I2C_TX_createTask(void);
38
39 extern void recibe_BLE(uint8_t*);

```

Ilustración 124. Código localizado en el fichero "I2C_TX.h".

Después, en el fichero "I2C_TX.c" (Ilustración 125) se localiza la definición de la función "recibe_BLE()":

```

222 // Función que hace que la variable global 'dato_ble' se actualice cuando se reciba
223 // el valor del ritmo cardiaco a través de la comunicación Bluetooth
224 void recibe_BLE(uint8_t *num)
225 {
226     dato_ble = *num;
227     return;
228 }

```

Ilustración 125. Código de la función "recibe_BLE()" en el fichero "I2C_TX.c".

Nótese que ahora la variable que almacenará el valor del ritmo cardiaco será un único entero sin signo de 8 bits y no un array, llamada “dato_ble” (Ilustración 126).

```
111// Variable global del programa que almacena el valor del BMP para enviarlo por el bus I2C
112uint8_t dato_ble = 0;
113//int debug = 0;
```

Ilustración 126. Declaración de la variable global "dato_ble".

Sin embargo, la mayor diferencia con el fichero del programa del primer módulo CC2650 se encuentra en la configuración del módulo I2C y en la ejecución del bucle infinito de la tarea, puesto que mientras el primero únicamente recibía datos, el segundo los envía. Véase en la Ilustración 127 que la dirección del dispositivo esclavo ahora es 0x10.

```
// Buffer de datos a enviar por el bus I2C
txBuffer[0] = dato_ble;

i2cTransaction.slaveAddress = 0x10; // Dirección I2C del dispositivo esclavo
i2cTransaction.writeBuf = txBuffer; // Datos a enviar
i2cTransaction.writeCount = 1; // Número de bytes a enviar
i2cTransaction.readBuf = NULL; // Datos a recibir (ninguno)
i2cTransaction.readCount = 0; // Número de bytes a recibir

while (1)
{
    // Se espera a que se publique el evento (véase en 'simple_central.c')
    Event_pend(readEvent, Event_Id_NONE, Event_Id_00, BIOS_WAIT_FOREVER);
    ++debug;
    txBuffer[0] = dato_ble;
    i2cTransaction.writeBuf = txBuffer;

    // Se desarrolla la comunicación por el bus I2C entre los microcontroladores
    if (I2C_transfer(i2c, &i2cTransaction))
    {
        key = Hwi_disable();

        Hwi_restore(key);
    }
}
```

Ilustración 127. Fragmento de código de la función "I2C_TX_TaskFxn()".

MAIN

Para acabar con este programa, en el fichero “main.c” de la carpeta “Startup” se añade la inclusión del fichero “I2C_TX.h” (Ilustración 128) y dentro de la función “main()” (Ilustración 129) se llama a la función que crea la tarea de ejecución del protocolo I2C “I2C_TX_createTask()”, de forma idéntica al programa del microcontrolador anterior.

```
127// Se incluye el fichero donde se ubica la tarea que ejecuta el protocolo de comunicación I2C
128#include "Application/I2C_TX.h"
```

Ilustración 128. Código del fichero “main.c”.

```
229 // Se llama a la función que crea la tarea de ejecución del protocolo I2C
230 I2C_TX_createTask();
```

Ilustración 129. Fragmento de código de la función “main()”.

3.4. Paso 4: F28069M_RX

Programa del segundo módulo F28069M, el cual se encarga de recibir el valor del ritmo cardiaco (BPM) a través del bus I2C en modo rápido o 'fast mode' (400 kbps).

Código significativo y explicación del programa:

Árbol del proyecto en Code Composer Studio (Ilustración 130).

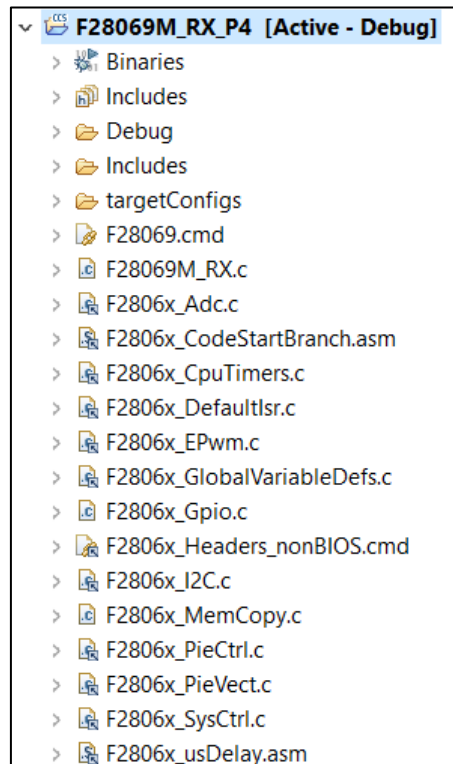


Ilustración 130. Árbol del programa "F28069M_RX_P4".

Declaraciones de las variables globales de las funciones auxiliares del programa (Ilustración 131):

```

50 __interrupt void i2c_rx_isr(void);
51
52 /** Declaraciones de las funciones auxiliares */
53
54 void I2C_Init(void);
55
56 /** Constantes y variables globales */
57
58 #define I2C_SLAVE_ADDR 0x10 // Dirección I2C del microcontrolador
59
60 int int_source = 100; // Variable que almacena el tipo de interrupción producida por el I2C
61 //Uint16 debug = 0; // Variable bandera de depuración
62 Uint8 resultado = 0; // Variable que almacena el valor recibido por el bus I2C

```

Ilustración 131. Declaraciones de las variables globales y de las funciones auxiliares del programa.

MÓDULO I2C

Para realizar este programa, solamente será necesario configurar e inicializar el módulo I2C del microcontrolador de forma casi idéntica al otro F28069M (Ilustración 132). Este trabajará como dispositivo esclavo (MST = 0) y deberá estar listo para recibir información (TRX = 0) cuando el dispositivo maestro quiera transmitir (RRDY = 1). Se utilizará el valor “0x10” como dirección I2C (registro “I2COAR”).

La frecuencia de funcionamiento del módulo I2C serán 10 MHz, mientras que la frecuencia de la señal SCL serán 400 kHz (fast mode).

```

120 /** Configuración del módulo I2C como dispositivo esclavo que únicamente recibirá */
121 /** mensajes pero no transmitirá nada. Pines: SCL: P33 y SDA: P32. */
122 void I2C_Init(void)
123 {
124     // El módulo I2C debe trabajar a una frecuencia de entre 7 y 12 MHz
125     // f = SYSCLKOUT / (IPSC + 1) = 90 MHz / (8+1) = 10 MHz
126     // Por otro lado, la señal SCL que recibirá será de 400 kHz, por lo que hay que configurar los
127     // preescaladores. f = f(I2C) / ((ICCL + d) + (ICCH + d)) = 10 MHz / ((8+5)+(7+5)) = 400 kHz
128     // Nota: 'd' es 5 porque el valor de IPSC es mayor que 1.
129     I2caRegs.I2CPSC.all = 8; // Valor del pre-escalador del módulo I2C
130     I2caRegs.I2CCLKL = 7; // Valor del primer pre-escalador del pin SCL. NO debe ser cero
131     I2caRegs.I2CCLKH = 8; // Valor del segundo pre-escalador del pin SCL. NO debe ser cero
132     I2caRegs.I2CIER.all = 0x0; // Desabilita todas las interrupciones
133     I2caRegs.I2CMDR.all = 0x0020; // Reset del módulo I2C (lo mismo que I2caRegs.I2CMDR.bit.IRS = 1)
134
135     I2caRegs.I2COAR = I2C_SLAVE_ADDR; // Dirección I2C del microcontrolador
136     I2caRegs.I2CMDR.bit.FREE = 1; // El módulo I2C seguirá funcionando aunque haya un 'breakpoint'
137     I2caRegs.I2CMDR.bit.MST = 0; // Modo esclavo. La señal SCL será recibida desde un dispositivo maestro
138     I2caRegs.I2CMDR.bit.TRX = 0; // Funcionamiento en modo receptor de mensajes a través del pin SDA
139     I2caRegs.I2CIER.bit.RRDY = 1; // Habilitación de la interrupción de 'listo para recibir'
140     I2caRegs.I2CMDR.bit.IRS = 1; // Reset del módulo I2C para guardar los cambios realizados
141     return;
142 }

```

Ilustración 132. Código de la función "I2C_Init()".

Para lograr una frecuencia del módulo de 10 MHz el valor del registro “I2CPSC” deberá ser 8. Por otro lado, la frecuencia del bus I2C deberá ser 400 kbps, por lo que el valor del registro “I2CCLKL” deberá ser 7, el “I2CCLKH” será 8 y la constante “d” es 5 porque el valor del registro “I2CPSC” es mayor que 1 (consultar el manual técnico).

$$f_{I2C} = \frac{SYSCLKOUT}{IPSC + 1} = \frac{90 \cdot 10^6}{(8 + 1)} = 10 \text{ MHz}$$

$$f_{SCL} = \frac{f_{I2C}}{(ICCL + d) + (ICCH + d)} = \frac{10 \cdot 10^6}{(7 + 5) + (8 + 5)} = 400 \text{ kHz}$$

La última instrucción de la configuración se corresponde con el reinicio del módulo I2C para guardar y aplicar los cambios (“IRS = 1”).

Tras configurar el módulo I2C, se programa la función que tratará las interrupciones producidas por el microcontrolador maestro del bus (Ilustración 133).

```
145 /** Función que trata la interrupción producida por el módulo I2C */  
146 __interrupt void i2c_rx_isr(void)  
147 {  
148     int_source = I2caRegs.I2CISRC.all; // Se registra qué tipo de interrupción que se ha producido  
149     if(int_source == I2C_RX_ISR) // Continúa si la interrupción ha sido 'módulo listo para recibir datos'  
150     {  
151         I2caRegs.I2CCNT = 1; // Número de bytes a enviar  
152         while(!I2caRegs.I2CSTR.bit.RRDY); // Se verifica que el bus está listo para recibir el mensaje  
153         resultado = I2caRegs.I2CDRR; // Se recibe el mensaje y se almacena en la variable 'resultado'  
154 //         ++debug;  
155     }  
156  
157     PieCtrlRegs.PIEACK.all |= PIEACK_GROUP8; // Acknowledge interrupt to PIE  
158     return;  
159 }
```

Ilustración 133. Código de la función "i2c_rx_isr()".

Esta función se encarga de recibir el valor del ritmo cardiaco del segundo microcontrolador CC2650. Para ello, cuando se produzca la interrupción "RRDY" o "datos listos para ser recibidos", el dato transmitido se almacenará en el registro "I2CDRR", cuyo valor será asignado a la variable "resultado".

3.5. Tabla de conexiones del proyecto

Parches	Electrodos
Brazo izquierdo	Rojo
Brazo derecho	Negro
Pierna derecha	Azul

Sensor SparkFun	MCU 1 (F28069M)
3.3 V	3.3 V
GND	GND
OUTPUT	ADCINA4

MCU 1 (F28069M)	MCU 2 (CC2650)
P33 (SCL)	DI04 (SCL)
P32 (SDA)	DI05 (SDA)
GND	GND

MCU 3 (CC2650)	MCU 4 (F28069M)
DI04 (SCL)	P33 (SCL)
DI05 (SDA)	P32 (SDA)
GND	GND

3.6. Comprobación de resultados y captura del montaje final

Utilizando las herramientas de depuración de Code Composer Studio, un analizador lógico digital y la aplicación para Android “BLE Scanner” se van a mostrar los resultados y el contenido de las variables que almacenan el valor del ritmo cardiaco en cada microcontrolador.

- ❖ MCU 1 (F28069M): En la siguiente imagen se visualiza el buffer de BPM, así como el contador de microsegundos “EPwm2TimerIntCount” y la variable que almacena el valor del ritmo cardiaco que será transmitido a través del bus I2C, “bpm_media”.

▼ bpm_buff	unsigned long[5]	[150,149,20,10,56]
[0]	unsigned long	150
[1]	unsigned long	149
[2]	unsigned long	20
[3]	unsigned long	10
[4]	unsigned long	56
bpm_media	int	77
EPwm2TimerIntCount	unsigned long	36481032

Ilustración 134. Captura (l) del depurador de CCS

Conectando un analizador lógico digital en paralelo a los pines que forman el bus I2C se puede comprobar como la comunicación entre el primer F28069M y el primer módulo CC2650 se realiza de manera satisfactoria cada vez que se realiza una lectura de la característica en el MCU 3 (Ilustración 135). La dirección del dispositivo esclavo (el MCU 1) es 0x05 y el siguiente byte de la trama se corresponde con el valor del ritmo cardiaco.

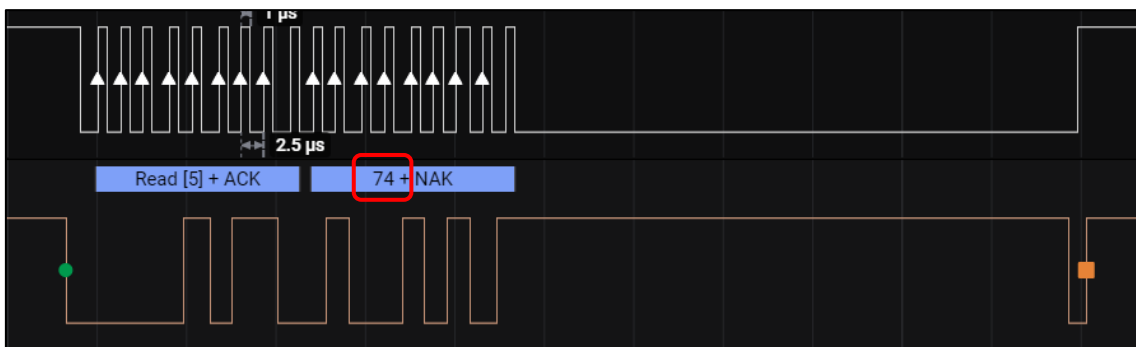


Ilustración 135. Captura (l) del programa Logic 2.

- ❖ MCU 2 (CC2650): Cada vez que el segundo módulo CC2650 (MCU 3) realice una lectura de la característica del servicio “Heart Rate”, este microcontrolador, trabajando como dispositivo periférico en el protocolo BLE, mandará al MCU 1 la orden de que se envíe el valor BPM a través del bus I2C y lo almacenará en el segundo número del array “msg_recibido”.

A continuación, se actualizarán los atributos “Heart Rate Value Format bit” y “Heart Rate Measurement Value” con los valores del array “msg_recibido”, tal y como se puede verificar con los valores del array “ConstantesVitales_ECGVal” (Ilustración 136). El valor del primer atributo siempre será 0 (msg_recibido[0]) porque eso significa que el tamaño del segundo atributo son 8 bits, y el valor del segundo atributo se corresponde con el valor del ritmo cardiaco (msg_recibido[1]).

msg_recibido	unsigned char[2]	[0 '\x00',78 'N']
[0]	unsigned char	0 '\x00'
[1]	unsigned char	78 'N'
ConstantesVitales_ECGVal	unsigned char[2]	[0 '\x00',78 'N']
[0]	unsigned char	0 '\x00'
[1]	unsigned char	78 'N'

Ilustración 136. Captura (II) del depurador de CCS.

Si se utiliza un smartphone con la aplicación “BLE Scanner” como dispositivo central o periférico para conectarse al CC2650 que trabaja como dispositivo periférico se comprueba que el microcontrolador tiene implementado el servicio “Heart Rate” y que la lectura del atributo de la característica “Heart Rate Measurement” se realiza satisfactoriamente.

Descubrimiento y conexión al dispositivo CC2650 llamado “Ritmo Cardiaco” (Ilustración 137):

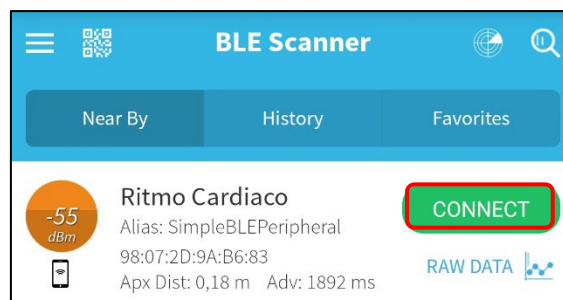


Ilustración 137. Captura (I) de la app “BLE Scanner”.

Se despliega el servicio “Heart Rate” y se pulsa en el botón virtual “R” (Ilustración 138) para realizar una lectura del atributo de la característica “Heart Rate Measurement”:

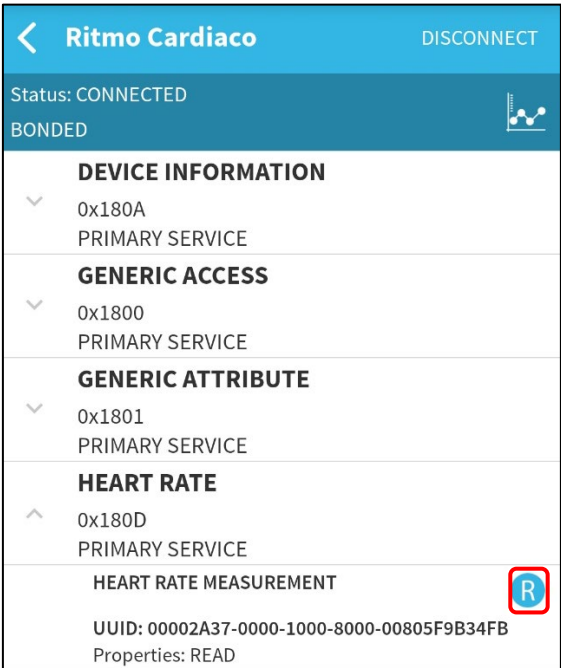


Ilustración 138. Captura (II) de la app "BLE Scanner".

El dispositivo CC2650 enviará el valor del ritmo cardiaco a la aplicación “BLE Scanner” (Ilustración 139):

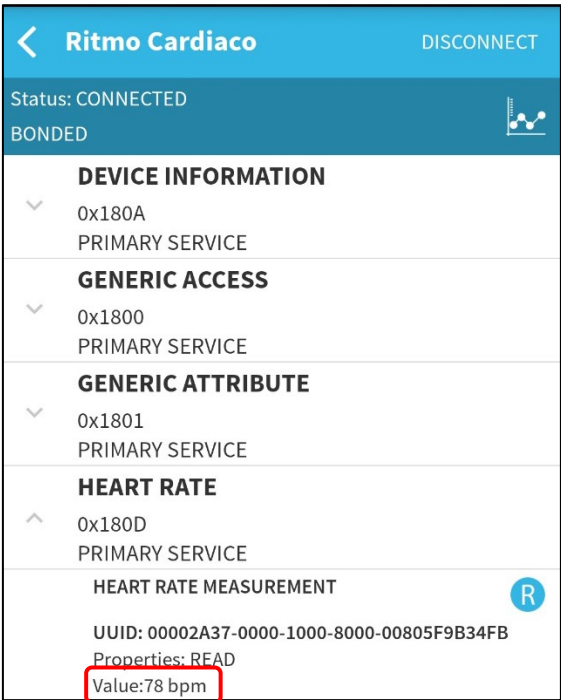
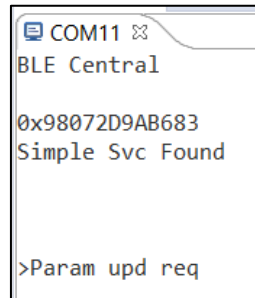


Ilustración 139. Captura (III) de la app "BLE Scanner".

- ❖ MCU 3 (CC2650). Se abre un terminal de comunicación serie para visualizar la navegación por el menú de opciones cuando el dispositivo maestro se conecte al periférico.

Cuando se abre el terminal serie (Ilustración 140) se observa que el dispositivo ya ha encontrado al CC2650 esclavo y se ha realizado la conexión de manera automática.



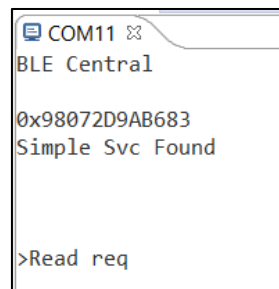
```
COM11 ☒
BLE Central

0x98072D9AB683
Simple Svc Found

>Param upd req
```

Ilustración 140. Captura (I) del terminal serie.

Se navega por el menú de opciones con el botón izquierdo (BUTTON1, Ilustración 141) hasta encontrar la opción “Read req”.



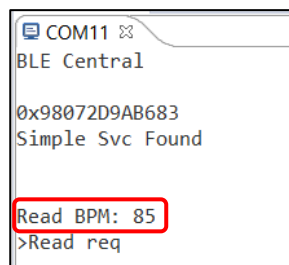
```
COM11 ☒
BLE Central

0x98072D9AB683
Simple Svc Found

>Read req
```

Ilustración 141. Captura (II) del terminal serie.

Se pulsa el botón derecho (BUTTON2, Ilustración 142) para enviar una solicitud de lectura del atributo de la característica que almacena el valor del ritmo cardiaco. Una vez recibido el mensaje, el valor BPM se muestra por el terminal serie.



```
COM11 ☒
BLE Central

0x98072D9AB683
Simple Svc Found

Read BPM: 85
>Read req
```

Ilustración 142. Captura (III) del terminal serie.

Puede comprobarse en el depurador de expresiones del IDE CCS (Ilustración 143) como la variable global “dato_ble” guarda el valor del ritmo cardiaco recibido a través del protocolo BLE. Este será el valor que se envía a través del bus I2C al último microcontrolador del proyecto.

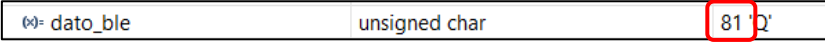


Ilustración 143. Captura (III) del depurador de CCS.

- ❖ MCU 4 (CC2650). En este último dispositivo se recibe por el bus I2C el valor del ritmo cardiaco cada vez que el MCU 3 realiza una lectura del atributo de la característica “Heart Rate Measurement”.

La variable “resultado” (Ilustración 144) es la encargada de almacenar el valor del ritmo cardiaco, tal y como muestra la siguiente imagen:



Ilustración 144. Captura (IV) del depurador de CCS.

Por último, se puede verificar que la comunicación a través del bus I2C es correcta con la ayuda del analizador lógico digital. En la siguiente ilustración se muestra una trama del mensaje enviado desde el MCU 3 al MC4. En ella se puede observar la dirección I2C del dispositivo esclavo (0x10) y en el segundo byte de la trama, el valor del ritmo cardiaco en ese momento, 76 BPM.

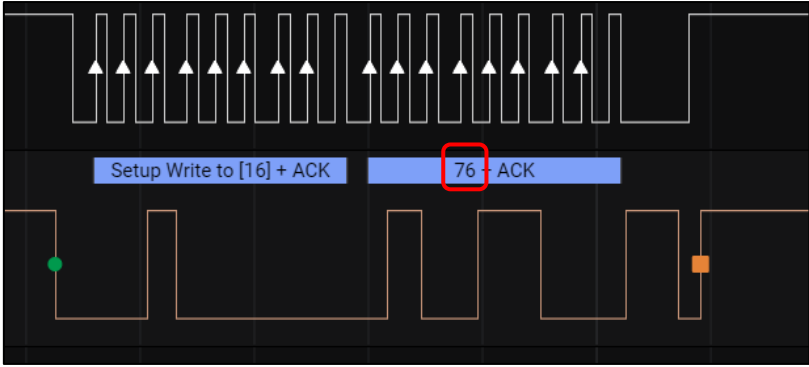


Ilustración 145. Captura (II) del programa Logic 2.

Foto de la implementación final del proyecto (Ilustración 146):

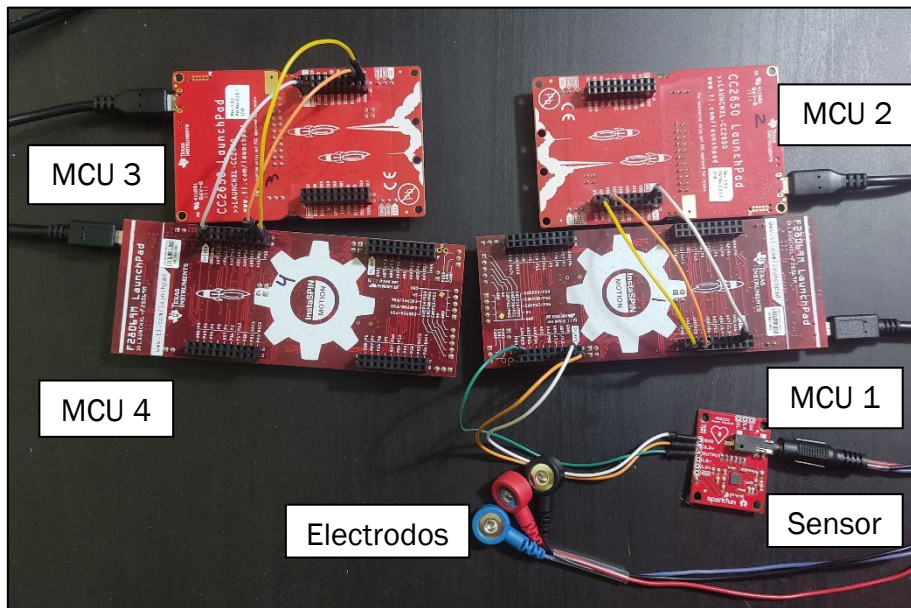


Ilustración 146. Foto del montaje final del proyecto.

Las placas de desarrollo LAUNCHXL-CC2650 pueden conectarse directamente a las LAUNCHXL-F28069M tanto por encima como por debajo, ya que la disposición de sus pines lo hace posible (formato “Booster Pack”), pero en la imagen anterior se muestran conectados mediante cables individuales por motivos de claridad y presentación.

4. Conclusiones

El proyecto, consistente en el desarrollo de una infraestructura de software que habilite la comunicación inalámbrica (BLE) entre dos parejas de microcontroladores F28069M y CC2650, con el objetivo de transmitir el valor del ritmo cardiaco de un paciente, ha sido realizado de manera satisfactoria cumpliendo todas las especificaciones definidas en el apartado de introducción y objetivos del proyecto. Cada vez que se envíe la orden (pulsando el botón 1 del microcontrolador CC2650 que trabaja como dispositivo maestro en el protocolo BLE), el valor del ritmo cardiaco calculado en el primer módulo F28069M con el algoritmo de Pan-Tomkins en base a los valores del electrocardiograma detectados por el sensor de SparFun, será enviado a través del bus I2C al primer módulo CC2650. Después, el segundo microcontrolador CC2650, trabajando en modo maestro en el protocolo BLE, leerá el atributo de la característica “Heart Rate Measurement” del primer CC2650 (funcionando en modo periférico o esclavo). Una vez leído y almacenado el valor del ritmo cardiaco, este será enviado a través del bus I2C al segundo F28069M y último microcontrolador del proyecto, que guardará en su memoria dicho dato, tal y como marcan las especificaciones del trabajo.

Como este trabajo es un primer acercamiento a la implementación de la comunicación inalámbrica en la plataforma de rehabilitación neuromotora RobHand, las líneas futuras de desarrollo deberían seguir los siguientes pasos:

- ❖ En el programa del primer módulo F28069M habría que sustituir el algoritmo de Pan-Tomkins por los nuevos datos a enviar y, si su longitud difiere de 1 byte, será necesario aumentar el número de paquetes a enviar a través del bus I2C.
- ❖ Del mismo modo, al programa del módulo CC2650 que trabaja como dispositivo periférico en el protocolo BLE, también será necesario introducir un número significativo de cambios. Estos estarán relacionados con el servicio y las características del protocolo BLE que se quieran implementar y, al igual que en el punto anterior, con el tamaño del mensaje a recibir a través del bus I2C.
- ❖ Por otro lado, el programa del segundo microcontrolador CC2650 no requeriría de tantos cambios como en los dos programas previos. Solamente habría que modificar el tamaño de la característica a leer (y qué octetos), además del número de paquetes a transmitir a través del protocolo I2C si el mensaje excede en longitud a 1 byte.
- ❖ Para acabar, en el último programa del proyecto, correspondiente al segundo microcontrolador F28069M, únicamente sería necesario modificar el número de paquetes de datos a recibir a través del bus I2C si el mensaje tiene un tamaño mayor a 1 byte.

Como puede observarse, los programas del segundo microcontrolador F28069M y del segundo módulo CC2650 serían los menos propensos a sufrir un gran número de modificaciones, mientras que los programas del primer módulo F28069M y del primer CC2650 necesitarían un mayor trabajo de adaptación con el cambio de las especificaciones.

5. Bibliografía

- [1] «ITAP,» [En línea]. Available: <https://www.itap.uva.es/>. [Último acceso: Febrero 2021].
- [2] Guk, K., Han, G., Lim, J., Jeong, K., Kang, T., Lim, E. K., & Jung, J. (2019)., «Evolution of wearable devices with real-time disease monitoring for personalized healthcare. *Nanomaterials*, 9(6), 813».
- [3] Villar, B. F., Ana, C., Vargas, M. M., Turiel, J. P., & Marinero, J. C. F. (2021)., «A Low Cost IoT Enabled Device for the Monitoring, Recording and Communication of Physiological Signals. In *BIODEVICES* (pp. 135-143).».
- [4] Schönle, P., Schulthess, F., Fateh, S., Ulrich, R., Huang, F., Burger, T., & Huang, Q. (2013, September)., «A DC-connectable multi-channel biomedical data acquisition ASIC with mains frequency cancellation. In *2013 Proceedings of the ESSCIRC (ESSCIRC)* (pp. 149-152). IEEE.».
- [5] Cui, M., & Tang, N. (2016)., «The Research and Design About Portable and Multi-parameter Health Care System Based on An Mcu of Pso4 Family. *DEStech Transactions on Engineering and Technology Research*, (iect).».
- [6] A. Cisnal, V. Moreno-SanJuan, D. Sierra, J.P. Turiel and J.C. Fraile, «An embedded implementation of EMG-driven control for assisted bilateral therapy».
- [7] Texas Instruments, «Microcontrolador LAUNCHXL-F28069M,» [En línea]. Available: <https://www.ti.com/tool/LAUNCHXL-F28069M>. [Último acceso: Febrero 2021].
- [8] Texas Instruments, «Microcontrolador LAUNCHXL-F28069M,» [En línea]. Available: https://software-dl.ti.com/C2000/docs/cla_software_dev_guide/intro.html. [Último acceso: Febrero 2021].
- [9] Texas Instruments, «Microcontrolador LAUNCHXL-F28069M,» [En línea]. Available: https://www.ti.com/en/download/mcu/C2000_Day_Presentation.pdf. [Último acceso: Febrero 2021].
- [10] Texas Instruments, «TMS320F28069M LaunchPad Development Kit Quick Start Guide».
- [11] Texas Instruments, «TMS320F2806x Piccolo Microcontrollers».
- [12] Texas Instruments, «LAUNCHXL-F28069M Overview User's Guide».
- [13] Texas Instruments, «Code Composer Studio,» [En línea]. Available: <https://www.ti.com/tool/CCSTUDIO>. [Último acceso: Febrero 2021].
- [14] Robert Wessel (Open Source), «Energia,» [En línea]. Available: <https://energia.nu/>. [Último acceso: Febrero 2021].
- [15] Texas Instruments, «controlSUITE,» [En línea]. Available: <https://www.ti.com/tool/CONTROLSUITE>. [Último acceso: Febrero 2021].

- [16] Texas Instruments, «C2000WARE,» [En línea]. Available: <https://www.ti.com/tool/C2000WARE>. [Último acceso: Febrero 2021].
- [17] Texas Instruments, «TMS320x2806x Piccolo Technical Reference Manual».
- [18] Texas Instruments, «Aprendizaje,» [En línea]. Available: <https://www.youtube.com/c/TexasInstruments/playlists>. [Último acceso: Febrero 2021].
- [19] Texas Instruments, «C28x Workshop».
- [20] Texas Instruments, *Programas de ejemplo de C2000Ware*.
- [21] Texas Instruments, «Microcontrolador LAUNCHXL-CC2650,» [En línea]. Available: <https://www.ti.com/tool/LAUNCHXL-CC2650>. [Último acceso: Mayo 2021].
- [22] Texas Instruments, «SimpleLink,» [En línea]. Available: <https://www.ti.com/wireless-connectivity/applications.html>. [Último acceso: Mayo 2021].
- [23] Texas Instruments, «TI BLE-Stack,» [En línea]. Available: <https://www.ti.com/tool/BLE-STACK>. [Último acceso: Mayo 2021].
- [24] Texas Instruments, «CC2650 LaunchPad Development Kit Quick Start Guide».
- [25] Texas Instruments, «CC2650 SimpleLink MCU».
- [26] Texas Instruments, «CC26x0 SimpleLink Developer's Guide».
- [27] Texas Instruments, «CC26x0 SimpleLink Technical Reference Manual».
- [28] IAR Systems, «IAR,» [En línea]. Available: <https://www.iar.com/>. [Último acceso: Mayo 2021].
- [29] Texas Instruments, «SmartRF Studio,» [En línea]. Available: <https://www.ti.com/tool/SMARTRFTM-STUDIO>. [Último acceso: Mayo 2021].
- [30] Texas Instruments, «SmartRF Flash Programmer 2,» [En línea]. Available: <https://www.ti.com/tool/FLASH-PROGRAMMER>. [Último acceso: Mayo 2021].
- [31] Texas Instruments, «TI-RTOS,» [En línea]. Available: http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/tirtos/index.html. [Último acceso: Mayo 2021].
- [32] Texas Instruments, «BLE SDK TI,» [En línea]. Available: <https://www.ti.com/tool/BLE-STACK-ARCHIVE>. [Último acceso: Mayo 2021].
- [33] Texas Instruments, «TI-RTOS,» [En línea]. Available: https://dev.ti.com/tirex/explore/node?node=AMtnZk2LTvHnhgseIRtw_w_krol.2c__LATEST. [Último acceso: Mayo 2021].
- [34] Texas Instruments, «TI-RTOS,» [En línea]. Available: https://dev.ti.com/tirex/explore/node?node=AHxv--RBRKmeZio00JEGGA__krol.2c__LATEST. [Último acceso: Mayo 2021].
- [35] Texas Instruments, «BLE Stack Archive,» [En línea]. [Último acceso: Mayo 2021].
- [36] Texas Instruments, «SimpleLink Academy,» [En línea]. Available: http://software-dl.ti.com/lprf/simplelink_academy/overview.html. [Último

- acceso: Mayo 2021].
- [37] Texas Instruments, «Project Zero,» [En línea]. Available: https://software-dl.ti.com/lprf/simplelink_cc2640r2_latest/docs/blestack/ble_user_guide/html/ble-stack-3.x-guide/get-started.html. [Último acceso: Mayo 2021].
 - [38] Texas Instruments, «Project Zero,» [En línea]. Available: https://dev.ti.com/tirex/explore/node?node=AMS03tMeJ9.PlrZjNs3L-A__pTTHBmu__LATEST. [Último acceso: Mayo 2021].
 - [39] Texas Instruments, «Project Zero,» [En línea]. Available: https://dev.ti.com/tirex/explore/node?node=AJ7jWuST9RweXKhWHqlyWg__FUz-xrs__LATEST. [Último acceso: Mayo 2021].
 - [40] Texas Instruments, «Project Zero,» [En línea]. Available: <https://training.ti.com/simplelink-academy-develop-your-bluetooth-low-energy-project>. [Último acceso: Mayo 2021].
 - [41] Bluepixel Technologies, «BLE Scanner (Android),» [En línea]. Available: https://play.google.com/store/apps/details?id=com.macdom.ble.blescanner&hl=en_US&gl=US. [Último acceso: Mayo 2021].
 - [42] S. Tatham, A. Lanes, B. Harris y J. Nevins, «PuTTY,» [En línea]. Available: <https://www.chiark.greenend.org.uk/~sgtatham/putty/>. [Último acceso: Mayo 2021].
 - [43] Texas Instruments, «Resource Explorer - Simple Central,» [En línea]. Available: https://dev.ti.com/tirex/explore/content/simplelink_cc2640r2_sdk_5_10_0_0_02/examples/rtos/CC2640R2_LAUNCHXL/blestack/simple_central/README.html. [Último acceso: Junio 2021].
 - [44] Bluetooth SIG, «Bluetooth,» [En línea]. Available: <https://www.bluetooth.com/>. [Último acceso: Mayo 2021].
 - [45] Texas Instruments, «Bluetooth Low Energy Software Developer's Guide,» [En línea]. Available: http://software-dl.ti.com/lprf/simplelink_cc2640r2_sdk/1.00.00.22/exports/docs/blestack/html/ble-stack/index.html. [Último acceso: Mayo 2021].
 - [46] Texas Instruments, «BLE-Stack User's Guide for Bluetooth 4.2,» [En línea]. Available: <http://software-dl.ti.com/lprf/sdg-latest/html/cc2640/architecture.html>. [Último acceso: Mayo 2021].
 - [47] Ellisys, «Ellisys Bluetooth Video Series,» [En línea]. Available: <https://www.youtube.com/watch?v=eZGixQzBo7Y&list=PLYj4Cw17Aw7ypuXt7mDFWyy6P661TD48&index=1>. [Último acceso: Mayo 2021].
 - [48] Nordic Semiconductor, «Introduction to Bluetooth Low Energy,» [En línea]. Available: <https://www.youtube.com/watch?v=5TxUnbsHsR8&t=884s>. [Último acceso: Mayo 2021].
 - [49] Nordic Semiconductor, «Nordic Semiconductor - Introduction to Bluetooth Smart,» [En línea]. Available: <https://www.youtube.com/watch?v=BZwOrQ6zkzE>. [Último acceso: Mayo 2021].

- [50] Microchip, «Bluetooth® Low Energy Packet Types,» [En línea]. Available: <https://www.microchipdeveloper.com/wireless:ble-link-layer-packet-types>. [Último acceso: Mayo 2021].
- [51] MIKROE, «Bluetooth Low Energy - Part 1: Introduction To BLE,» [En línea]. Available: <https://www.mikroe.com/blog/bluetooth-low-energy-part-1-introduction-ble>. [Último acceso: Mayo 2021].
- [52] RF Wireless World, «BLE Protocol Stack | BLE System Architecture,» [En línea]. Available: <https://www.rfwireless-world.com/Terminology/BLE-Protocol-Stack-Architecture.html>. [Último acceso: Mayo 2021].
- [53] RF Wireless World, «BLE Advertising packet format | BLE Data packet format,» [En línea]. Available: <https://www.rfwireless-world.com/Terminology/BLE-Advertising-and-Data-Packet-Format.html>. [Último acceso: Mayo 2021].
- [54] «Protocolo I2C,» [En línea]. Available: <https://hetprostore.com/TUTORIALES/i2c/>. [Último acceso: Abril 2021].
- [55] S. Campbell, «Circuit Basics,» [En línea]. Available: <https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/>. [Último acceso: Abril 2021].
- [56] Descubre Arduino, «Protocolo I2C,» [En línea]. Available: <https://descubrearduino.com/i2c/>.
- [57] «Velocidades en un sistema de transmisión,» [En línea]. Available: <https://www.textoscientificos.com/redes/comunicaciones/velocidades>. [Último acceso: Abril 2021].
- [58] «Protocolo I2C,» [En línea]. Available: <https://www.i2c-bus.org/>. [Último acceso: Abril 2021].
- [59] Fundación española del corazón, «Fundación del corazón,» [En línea]. Available: <https://fundaciondelcorazon.com/informacion-para-pacientes/metodos-diagnosticos/electrocardiograma.html>. [Último acceso: Julio 2021].
- [60] Medline Plus, «Medline Plus,» [En línea]. Available: <https://medlineplus.gov/spanish/pruebas-de-laboratorio/electrocardiograma/>. [Último acceso: Julio 2021].
- [61] Wikipedia, «Wikipedia,» [En línea]. Available: <https://es.wikipedia.org/wiki/Electrocardiograma>. [Último acceso: Junio 2021].
- [62] IEEE, «IEEE Explore,» [En línea]. Available: <https://ieeexplore.ieee.org/document/4122029>. [Último acceso: Junio 2021].
- [63] Wikipedia, «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Pan%E2%80%93Tompkins_algorithm. [Último acceso: Junio 2021].
- [64] SparkFun, «SparkFun,» [En línea]. Available: <https://www.sparkfun.com/products/12650>. [Último acceso: Junio 2021].

- [65] SparkFun, «Single-Lead, Heart Rate Monitor Front End (AD8232),» 2021.
- [66] SparkFun, «SparFun,» [En línea]. Available:
<https://www.sparkfun.com/products/12970>. [Último acceso: Junio 2021].
- [67] SparkFun, «Learn SparkFun,» [En línea]. Available:
<https://learn.sparkfun.com/tutorials/ad8232-heart-rate-monitor-hookup-guide>. [Último acceso: Julio 2021].
- [68] SparkFun, «GitHub - SparkFun,» [En línea]. Available:
https://github.com/sparkfun/AD8232_Heart_Rate_Monitor. [Último acceso: Julio 2021].
- [69] GitHub, «GitHub - real_time_QRS_detection,» [En línea]. Available:
https://github.com/blakeMilner/real_time_QRS_detection. [Último acceso: Junio 2021].
- [70] Bluetooth SIG, «Bluetooth SIG - 16-bit UUID Numbers,» [En línea]. Available:
<https://specificationrefs.bluetooth.com/assigned-values/16-bit%20UUID%20Numbers%20Document.pdf>. [Último acceso: Junio 2021].
- [71] Bluegiga Technologies, «Bluegiga Technologies,» [En línea]. Available:
http://www.mt-system.ru/sites/default/files/documents/ble_application_note_hrp_sensor.pdf. [Último acceso: Junio 2021].
- [72] Axodyne, «Axodyne - BLE UUIDs,» [En línea]. Available:
<https://axodyne.com/2020/08/ble-uuids/>. [Último acceso: Junio 2021].
- [73] Bluetooth SIG, «Heart Rate Service».
- [74] Texas Instruments, «SimpleLink Academy,» [En línea]. Available:
https://dev.ti.com/tirex/content/simplelink_academy_cc2640r2sdk_2_20_03_05/modules/blestack/ble_01_basic/ble_01_basic.html. [Último acceso: Junio 2021].
- [75] Texas Instruments, «SimpleLink Academy,» [En línea]. Available:
https://dev.ti.com/tirex/content/simplelink_academy_cc2640r2sdk_2_20_03_05/modules/blestack/ble_02_thermostat/ble_02_thermostat.html. [Último acceso: Junio 2021].
- [76] Texas Instruments, «Texas Instruments - Resource explorer,» [En línea]. Available:
https://dev.ti.com/tirex/explore/node?node=AALRuBvnOq9sws.aXdTEQw__krol.2c__LATEST. [Último acceso: Julio 2021].
- [77] Texas Instruments, «Texas Instruments - SimpleLink,» [En línea]. [Último acceso: Junio 2021].
- [78] Texas Instruments, «Texas Instruments - SimpleLink,» [En línea]. Available:
http://software-dl.ti.com/lprf/simplelink_cc2640r2_sdk/1.35.00.33/exports/docs/tidrivers/doxygen/html/_i2_c_8h.html. [Último acceso: Junio 2021].
- [79] Texas Instruments, «Foro Texas Instruments,» [En línea]. Available:
<https://e2e.ti.com/support/wireless-connectivity/bluetooth->

group/bluetooth/f/bluetooth-forum/491182/connecting-device-automatically-when-switched-on . [Último acceso: Julio 2021].

- [80] «Cálculo de la resistencia pull-up para I2C,» [En línea]. Available: <https://rheingoldheavy.com/i2c-pull-resistors/>. [Último acceso: Abril 2021].
- [81] Texas Instruments, «I2C Bus Pull-Up Resistor Calculation».

6. Anexos

Adjunto a este documento se encuentran dos carpetas. La carpeta “Code Composer Studio” almacena los 4 programas desarrollados para los microcontroladores del proyecto y en “Documentación” se encuentran todos los manuales técnicos y otros documentos digitales a los que se hace referencia a lo largo de este trabajo.

6.1. Cálculo resistencia pull-up para el bus I2C

RESISTENCIA PULL-UP INTERNA DEL MICROCONTROLADOR F28069M

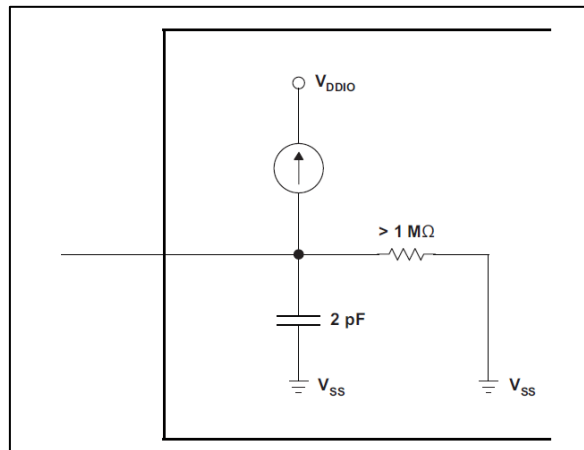


Ilustración 147. Esquema de la resistencia pull-up embebida en la LaunchPad.

Primero se va a observar el funcionamiento de la señal de reloj SCL del módulo I2C utilizando la resistencia pull-up embebida en la LaunchPad F28069M (Ilustración 147) a diferentes frecuencias de trabajo.

❖ Frecuencia SCL de 100 KHz:

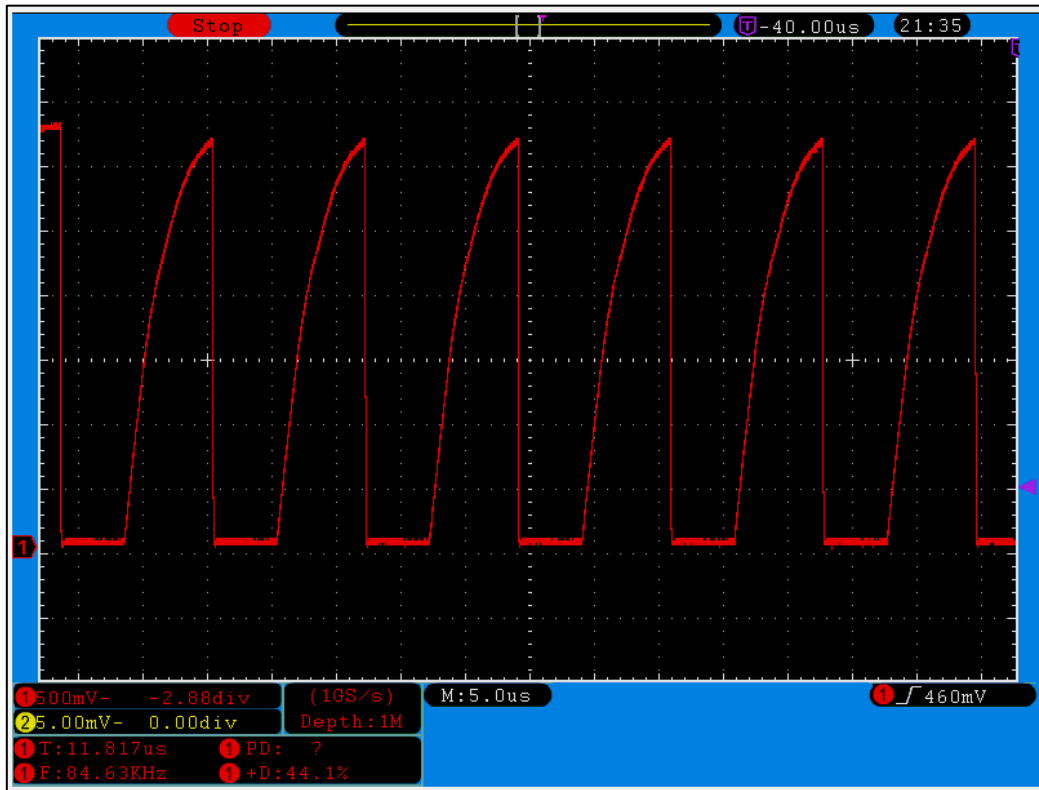


Ilustración 148. Frecuencia señal SCL a 100 KHz.

❖ Frecuencia de SCL de 400 KHz:

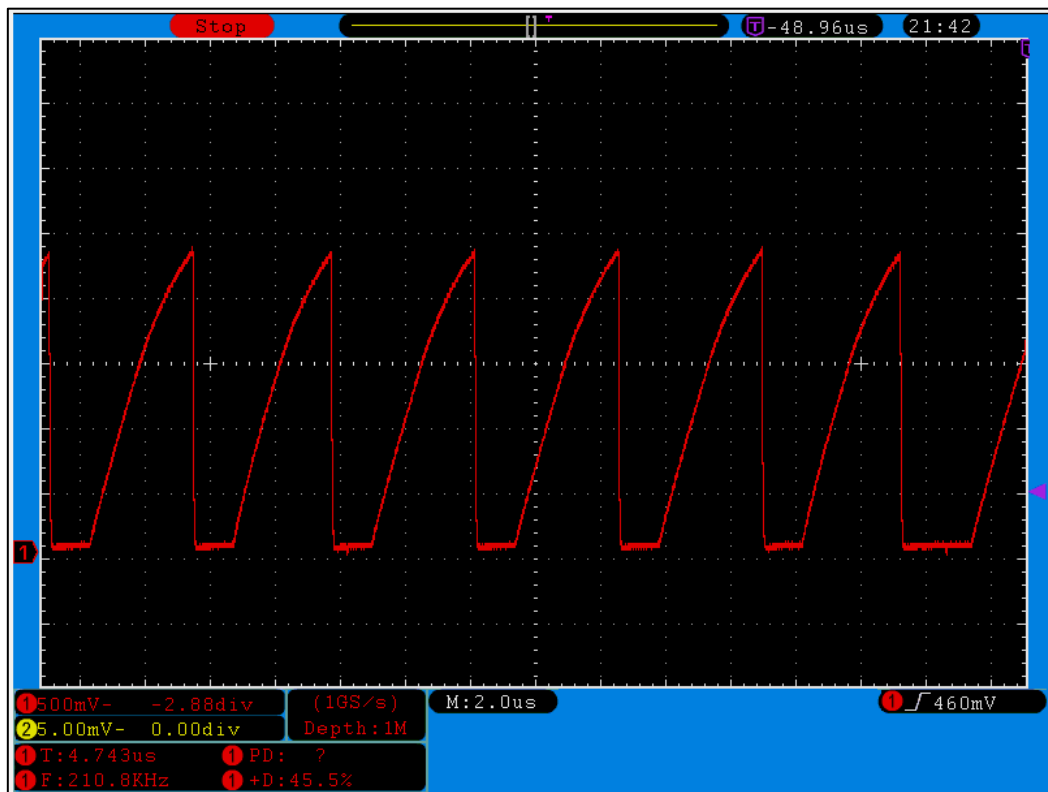


Ilustración 149. Frecuencia señal SCL a 400 KHz.

Como puede observarse (Ilustración 148 e Ilustración 149), la señal de reloj tiene una forma deficiente y muy diferente a la ideal. Si se trabaja a 100 KHz con esta señal de reloj la comunicación I2C funciona correctamente, pero a frecuencias más altas, como a 400 KHz, por ejemplo, la comunicación es del todo imposible, puesto que los dispositivos esclavos detectan una frecuencia de reloj con un valor que se corresponde a la mitad del real.

Para solucionar este contratiempo se va a desactivar la resistencia pull-up interna y se va a calcular e implementar una externa [80].

CÁLCULO RESISTENCIA PULL-UP

El valor mínimo de la resistencia se calcula de forma muy sencilla con la siguiente ecuación:

$$R_p(\text{mín}) = \frac{V_{CC} - V_{OL}(\text{máx})}{I_{OL}}$$

V_{OL} es el máximo valor de tensión que es considerado como nivel lógico bajo.

Por otro lado, el cálculo de la resistencia máxima es algo más complejo. Este valor máximo está limitado por la capacitancia del bus, C_b , debido a las especificaciones del protocolo I2C en cuanto a tiempo de subida. Si la resistencia es demasiado grande, la señal no llegará a nivel alto antes de que llegue el momento de volver a cambiar el nivel lógico. La ecuación [81] que caracteriza este comportamiento en la siguiente:

$$V(t) = V_{CC} \cdot \left(1 - e^{-\frac{t}{RC}}\right)$$

$$\text{Para } V_{IH} = 0,7 \cdot V_{CC} = V_{CC} \cdot \left(1 - e^{-\frac{t_1}{RC}}\right)$$

$$\text{Para } V_{IL} = 0,3 \cdot V_{CC} = V_{CC} \cdot \left(1 - e^{-\frac{t_2}{RC}}\right)$$

De esta manera, el tiempo de subida será $t_r = t_2 - t_1 = 0,8473 \cdot R_p \cdot C_b$, por lo que la resistencia máxima es:

$$R_p(\text{máx}) = \frac{t_r}{0,8473 \cdot C_b}$$

Ahora se consultan las especificaciones de los parámetros del protocolo I2C (Ilustración 150) para obtener los datos de las ecuaciones a resolver:

Parameter		Standard Mode (Max)	Fast Mode (Max)	Fast Mode Plus (Max)	Unit
t_r	Rise time of both SDA and SCL signals	1000	300	120	ns
C_b	Capacitive load for each bus line	400	400	550	pF
V_{OL}	Low-level output voltage (at 3 mA current sink, $V_{CC} > 2 V$)	0.4	0.4	0.4	V
	Low-level output voltage (at 2 mA current sink, $V_{CC} \leq 2 V$)	–	$0.2 \times V_{CC}$	$0.2 \times V_{CC}$	V

Ilustración 150. Especificaciones del protocolo I2C. Tabla de Texas Instruments.

Se halla el rango de resistencias adecuado para los siguientes datos:

$$V_{CC} = 3,3 V; \quad t_r = 300 \cdot 10^{-9} s \text{ (queremos 400 Kbit/s);} \quad C_b = 200 \cdot 10^{-12} F$$

$$R_p(\text{mín}) = \frac{3,3 - 0,4}{3 \cdot 10^{-3}} = 966,67 \Omega$$

$$R_p(\text{máx}) = \frac{300 \cdot 10^{-9}}{0,8473 \cdot 200 \cdot 10^{-12}} = 1,77 K\Omega$$

Tras calcular el rango de resistencias hay que decidirse por un valor concreto. Cuanto más pequeña sea la resistencia mejor para la velocidad de conmutación entre niveles lógicos, pero el consumo de corriente será mayor; mientras que cuanto mayor sea la resistencia se consumirá menos corriente, pero la velocidad de cambio será menor.

De esta manera, se eligen dos resistencias pull-up de 1 K Ω , una para cada línea del bus, SDA y SCL.

Por último, se comprueba la forma de la señal y su funcionamiento con la ayuda del osciloscopio digital y se observa que los resultados (Ilustración 151 e Ilustración 152) son mucho mejores que con la resistencia interna del microcontrolador, además de que la frecuencia que detecta el osciloscopio digital de la señal de reloj es prácticamente idéntica a la teórica.

RESULTADOS

❖ Frecuencia de SCL de 100kHz:

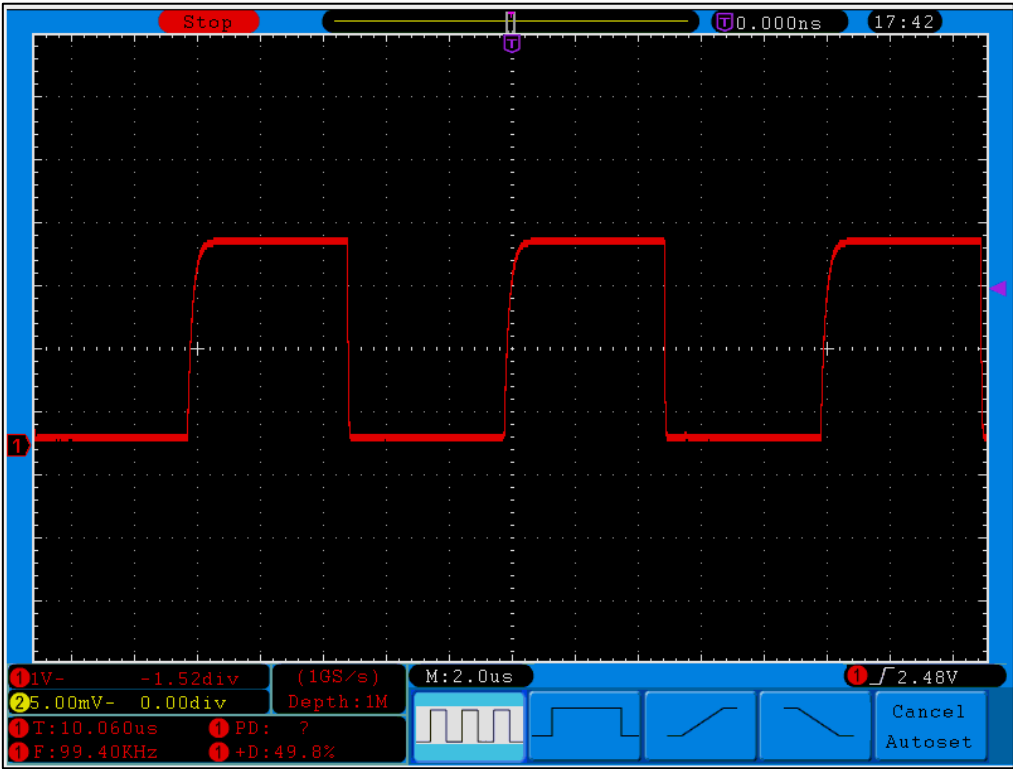


Ilustración 151. Frecuencia señal SCL a 100 KHz.

❖ Frecuencia de SCL de 400kHz:

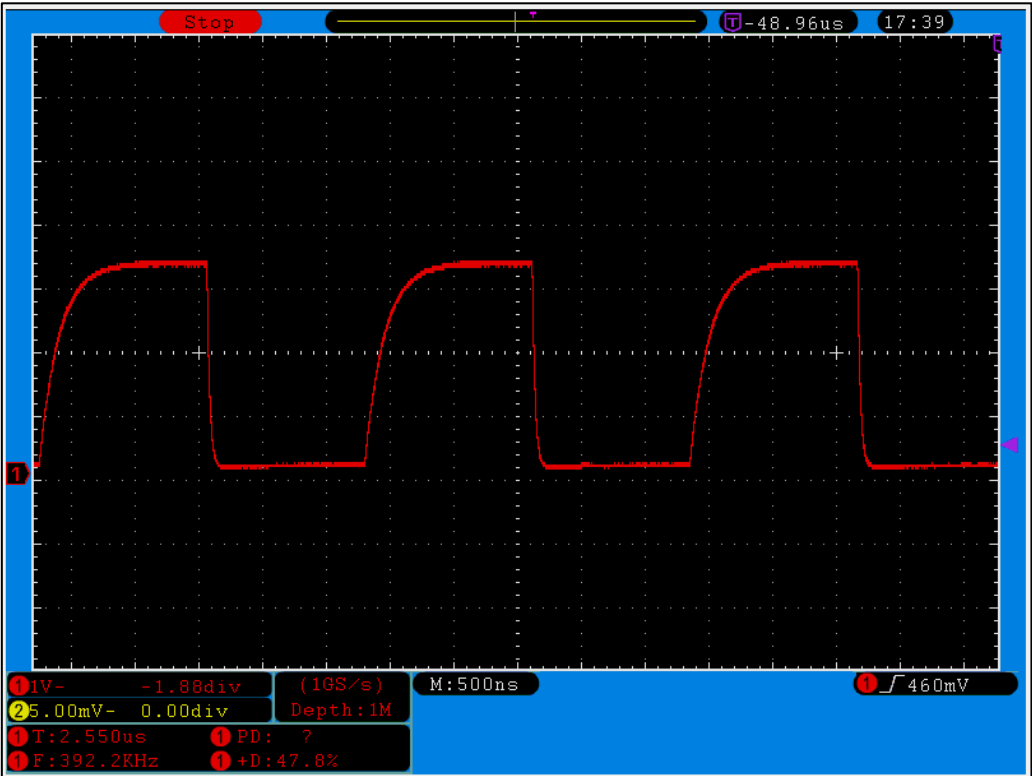


Ilustración 152. Frecuencia señal SCL a 400 KHz.