
Monitorización de señales biomédicas en sistemas Android



TRABAJO DE FIN DE GRADO

Mario Michiels Toquero
Cristian Pinto Lozano

Departamento de Arquitectura de Computadores y Automática
Facultad de Informática
Universidad Complutense de Madrid

Junio 2017

Monitorización de señales biomédicas en sistemas Android

*Memoria que presentan para optar al título de Graduados en
Ingeniería del Software*

**Mario Michiels Toquero
Cristian Pinto Lozano**

Dirigida por el Doctor
Joaquín Recas Piorno

**Departamento de Arquitectura de Computadores y
Automática
Facultad de Informática
Universidad Complutense de Madrid**

Junio 2017

A nuestros padres

*Any fool can criticize, condemn and complain
- and most fools do. But it takes character
and self-control to be understanding and forgiving.
Dale Carnegie (1888 - 1955)*

Agradecimientos

*A todos los que la presente vieron y
entendieron.*

Inicio de las Leyes Orgánicas. Juan
Carlos I

Groucho Marx decía que encontraba a la televisión muy educativa porque cada vez que alguien la encendía, él se iba a otra habitación a leer un libro. Utilizando un esquema similar, nosotros queremos agradecer al Word de Microsoft el habernos forzado a utilizar \LaTeX . Cualquiera que haya intentado escribir un documento de más de 150 páginas con esta aplicación entenderá a qué nos referimos. Y lo decimos porque nuestra andadura con \LaTeX comenzó, precisamente, después de escribir un documento de algo más de 200 páginas. Una vez terminado decidimos que nunca más pasaríamos por ahí. Y entonces caímos en \LaTeX .

Es muy posible que hubiéramos llegado al mismo sitio de todas formas, ya que en el mundo académico a la hora de escribir artículos y contribuciones a congresos lo más extendido es \LaTeX . Sin embargo, también es cierto que cuando intentas escribir un documento grande en \LaTeX por tu cuenta y riesgo sin un enlace del tipo “*Author instructions*”, se hace cuesta arriba, pues uno no sabe por donde empezar.

Y ahí es donde debemos agradecer tanto a Pablo Gervás como a Miguel Palomino su ayuda. El primero nos ofreció el código fuente de una programación docente que había hecho unos años atrás y que nos sirvió de inspiración (por ejemplo, el fichero `guionado.tex` de \TeX IS tiene una estructura casi exacta a la suya e incluso puede que el nombre sea el mismo). El segundo nos dejó husmear en el código fuente de su propia tesis donde, además de otras cosas más interesantes pero menos curiosas, descubrimos que aún hay gente que escribe los acentos españoles con el `\'{\i}`.

No podemos tampoco olvidar a los numerosos autores de los libros y tutoriales de \LaTeX que no sólo permiten descargar esos manuales sin coste adicional, sino que también dejan disponible el código fuente. Estamos pensando en Tobias Oetiker, Hubert Partl, Irene Hyna y Elisabeth Schlegl, autores del famoso “The Not So Short Introduction to $\text{\LaTeX} 2_{\epsilon}$ ” y en Tomás

Bautista, autor de la traducción al español. De ellos es, entre otras muchas cosas, el entorno **example** utilizado en algunos momentos en este manual.

También estamos en deuda con Joaquín Ataz López, autor del libro “Creación de ficheros L^AT_EX con GNU Emacs”. Gracias a él dejamos de lado a WinEdt y a Kile, los editores que por entonces utilizábamos en entornos Windows y Linux respectivamente, y nos pasamos a emacs. El tiempo de escritura que nos ahorramos por no mover las manos del teclado para desplazar el cursor o por no tener que escribir `\emph` una y otra vez se lo debemos a él; nuestro ocio y vida social se lo agradecen.

Por último, gracias a toda esa gente creadora de manuales, tutoriales, documentación de paquetes o respuestas en foros que hemos utilizado y seguiremos utilizando en nuestro quehacer como usuarios de L^AT_EX. Sabéis un montón.

Y para terminar, a Donal Knuth, Leslie Lamport y todos los que hacen y han hecho posible que hoy puedas estar leyendo estas líneas.

Resumen

...

...

Abstract

...

...

Índice

Agradecimientos	IX
Resumen	XI
Abstract	XIII
1. Introducción	1
1.1. Motivación	1
1.2. Estructura de capítulos	2
2. Antecedentes	3
2.1. Electrocardiograma	3
2.1.1. Estructura de un ECG	4
2.1.2. Funcionamiento interno del corazón	5
2.1.3. Obtención del ECG	6
2.1.4. Utilidades del ECG	6
2.2. Tecnologías	6
2.2.1. BeagleBone Black	7
2.2.2. Bus SPI	9
2.2.3. Chip ADS1198	11
2.2.4. Analizador lógico Saleae	13
3. Desarrollo	15
3.1. Proyecto inicial	15
3.1.1. Objetivos	16
3.1.2. Alternativas	17
3.2. Programación en tiempo real	18
3.2.1. Decisiones de diseño	19
3.3. Configuración hardware	19
3.3.1. Device Tree	21
3.3.2. Device Tree Overlay	22
3.3.3. Conexiones físicas	23

Bibliografía	27
---------------------	-----------

Lista de acrónimos	28
---------------------------	-----------

Índice de figuras

2.1. Estructura habitual de la señal ECG durante un ciclo cardiaco.	4
2.2. Estructura anatómica del corazón humano	5
2.3. Vista general de una BeagleBone Black	7
2.4. Especificación hardware BeagleBone Black	9
2.5. Estándar de comunicación SPI	10
2.6. Configuraciones posibles de fase y polaridad en el estándar SPI	11
2.7. Diagrama de funcionamiento interno ADS1198	12
2.8. Software multiplataforma proporcionado por Saleae	14
3.1. Ejemplo de salida al ejecutar comando <code>top</code> en una terminal linux	17
3.2. Diagrama de conexiones físicas disponibles en una BBB . . .	20
3.3. Ejemplo simple de un fichero <code>.dts</code>	21
3.4. Device Tree Overlay del proyecto para la BBB	23
3.5. Conexiones físicas necesarias para el proyecto	25

Índice de Tablas

3.1. Conexiones físicas necesarias para la alimentación del chip ADS1198	24
3.2. Conexiones físicas de carácter general GPIO	24
3.3. Conexiones físicas para posibilitar la conexión SPI	24
3.4. Conexiones físicas del analizador lógico Saleae	24

Capítulo 1

Introducción

*Un comienzo no desaparece nunca,
ni siquiera con un final.*

Harry Mulisch

Motivación

La sociedad actual en la que nos encontramos está completamente informatizada podríamos decir. Es posible encontrar software en todo tipo de lugares en los que jamás hubiesemos pensado hace décadas que hubiese sido posible, como por ejemplo, neveras en las que una vez acabado un cierto tipo de refrigerio, es el propio electrodoméstico el encargado de comprarlo por nosotros, relojes con los que podemos conectarnos a internet y comunicarnos, eliminando la necesidad de llevar un teléfono encima, e incluso zapatillas que se encargan de pedir nuestra comida favorita a domicilio con tan solo pulsar un pequeño botón.

Es imposible enumerar la cantidad de aplicaciones que el software podría tener, al menos en la presente memoria, y todo esto sin olvidar los futuros usos que adquirirá. Podríamos decir que la industria del Software está en pleno auge, es más, lleva en pleno auge desde hace décadas, y no parece que vaya a decaer. Con tantos posibles sectores en los que especializarse, parece complicado elegir uno en el que sumergirse.

Sin embargo, para los autores de la presente memoria siempre tuvo cierto atractivo el desarrollo de software para dispositivos empotrados que utilizan software libre. Fue entonces cuando se nos dio la posibilidad de trabajar en el presente proyecto, utilizando este tipo de dispositivos y además íntimamente enfocado y relacionado con el cuidado de la salud. La mezcla de ambos componentes nos fascinó a primera vista.

El desarrollo de software para dispositivos empotrados se encuentra en pleno crecimiento desde hace ya varios años, y cada vez son más populares las

comunidades que dan soporte a los desarrolladores de los mismos, facilitando así el proceso de creación del software. Asimismo también se encuentra en uno de sus mejores momentos todo tipo de *gadget* capaz de medir o monitorizar ciertas señales, como pueden ser los actuales relojes inteligentes, o las pulseras de actividad tan frecuentemente vistas. Estos dispositivos son capaces de monitorizar multitud de señales procedentes del cuerpo humano, y todo esto en un reducido espacio físico fácilmente portable.

La monitorización de ciertas señales biomédicas puede ser un factor fundamental a la hora de detectar problemas de salud de forma precoz. La presente memoria trata de ilustrar el proceso gracias al cual es posible monitorizar este tipo de señales utilizando un hardware de bajo coste y portable, y un software libre, adecuado y preciso, cuya unión pueda facilitar la práctica de este tipo de procesos en todos los contextos en los que pudiera ser necesario.

Estructura de capítulos

Capítulo 2

Antecedentes

*Cada día sabemos más
y entendemos menos.*

Albert Einstein

RESUMEN: Para la realización del presente proyecto ha sido necesario realizar una investigación que cubre desde aspectos médicos hasta aspectos técnicos propios de una ingeniería compleja. En este capítulo se expone un breve resumen de cada antecedente investigado.

Electrocardiograma

Un electrocardiograma (más popularmente conocido como ECG) es un proceso por el cual se registran las actividades eléctricas que emite el corazón durante un tiempo determinado.

El registro de esta actividad cardiaca es posible gracias a las diferencias de potencial existentes producidas por la contractilidad del corazón. El estudio de la información ilustrada por un ECG puede ser de gran utilidad a la hora de detectar multitud de enfermedades cardiovasculares, así como prevenir problemas cardiacos de forma precoz.

La ventaja de un ECG frente a otros métodos de medición para comprobar el estado del corazón, es que el ECG aporta mucha más información que los métodos más habituales como pueden ser simplemente medir el pulso o utilizar un estetoscopio.

Entre la información extra que puede obtenerse recurriendo al ECG, cabe destacar desde la medición continua durante un tiempo prolongado (días incluso si se utiliza un ECG portátil) hasta la obtención del movimiento de

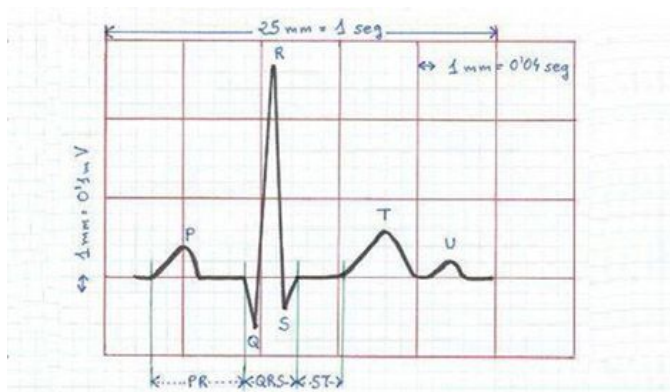


Figura 2.1: Estructura habitual de la señal ECG durante un ciclo cardiaco.

los músculos del corazón (los electrodos conectados al paciente registran esos movimientos en mV), lo cual permite saber cuando entra la sangre, cuando sale, cuanto dura cada movimiento, etc. Sin esa información sería imposible detectar ciertos tipos de arritmias y/o otras alteraciones del corazón.

Estructura de un ECG

La estructura habitual de un ECG [Fig 2.1] habitualmente está formada por un conjunto de ondas y complejos determinados:

- Onda P
- Onda Q
- Complejo QRS
- Onda T
- Onda U

Habitualmente este tipo de señales se encuentran dentro de un rango determinado de amplitudes, que generalmente abarca desde los $0.5mV$ hasta los $5mV$, existiendo además una componente continua causada por el contacto existente entre los electrodos y la piel.

Mayormente este tipo de señales suele ilustrarse en los libros de forma muy clara y reconocible, aunque no siempre es posible disponer de un entorno con las características propicias para eliminar toda existencia de ruido en la señal. Generalmente el ruido que se recoge al analizar este tipo de señales es despreciado, aunque en ciertas ocasiones puede llegar a ser tan difuso, que nos impida reconocer hasta las pautas más características de una señal ECG.

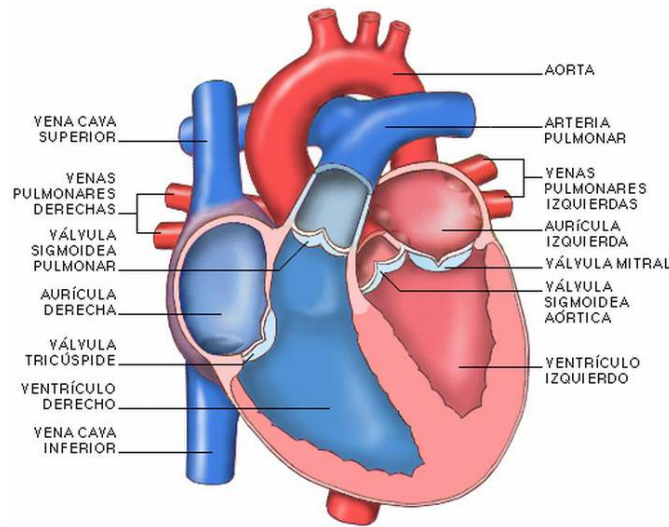


Figura 2.2: Estructura anatómica del corazón humano

Puede presentarse ruido en la señal simplemente debido a la luz tanto natural como artificial que incide indirectamente en los electrodos, así como debido a la corriente que reciben los dispositivos que nos permiten llevar a cabo la medición de la señal.

Funcionamiento interno del corazón

El corazón está constituido por cuatro cámaras diferenciadas: dos aurículas (izquierda y derecha) y dos ventrículos (izquierdo y derecho) como puede apreciarse en la [Fig 2.2]

Su funcionamiento sigue siempre un mismo ciclo que se repite una y otra vez: la aurícula derecha recibe la sangre venosa del cuerpo a través de la vena cava superior y la envía al ventrículo derecho. Para que dicha sangre pueda oxigenarse, el ventrículo derecho la envía a través de la arteria pulmonar a los pulmones, retornando al corazón, a través de la vena pulmonar, a la aurícula izquierda. Para finalizar con el ciclo, la sangre pasa de dicha aurícula al ventrículo izquierdo, el cual la distribuye por todo el cuerpo gracias a la arteria aorta, para volver posteriormente a la aurícula derecha y así cerrar el ciclo.

Para que este ciclo periódico funcione de manera correcta, síncrona y con total ausencia de errores, el corazón dispone de un sistema de conducción eléctrica constituido por fibras de músculo cardíaco cuya finalidad es la transmisión de impulsos eléctricos. El motivo por el cual no tenemos ningún tipo de control sobre los latidos del corazón es porque este sistema es autoexcitable.

Obtención del ECG

El proceso físico a través del cual se realiza el procedimiento de registro y se obtiene el ECG consiste en la medición de la actividad eléctrica del corazón entre distintos puntos corporales, ya que, mientras que el corazón pasa por los estados de polarización y despolarización, el cuerpo en su conjunto se comporta como un volumen conductor, propagando la corriente eléctrica.

El equipo de registro consta de una serie de electrodos, que se conectan a la piel del paciente y a un equipo de registro. Estos electrodos se colocan en unas posiciones predeterminadas y universales, lo que permite obtener el registro del sistema de la forma más precisa y exacta posible.

Utilidades del ECG

Tras la obtención de un ECG completo, se tendrá un registro completo acerca del tamaño, la cadencia, y la naturaleza de los impulsos eléctricos emitidos por el corazón. Gracias a esta información, es posible definir tanto el ritmo como la frecuencia cardíaca. Este tipo de información puede ser crucial a la hora de detectar ciertos problemas cardíacos. A continuación se detallan posibles utilidades que puede tener la obtención de un ECG:

- Obtener información sobre el estado físico del corazón.
- Detectar problemas cardíacos de forma precoz, o alteraciones electrolíticas de potasio, sodio, calcio, magnesio, u otros elementos químicos.
- Determinar si el corazón funciona correctamente, en caso de no encontrar ninguna anomalía.
- Adquirir información acerca de la repercusión miocárdica de diversas enfermedades cardíacas.
- Detección de anormalidades conductivas.
- Indicar bloqueos coronarios arteriales.

Tecnologías

Para llevar a cabo el correcto desarrollo del trabajo que se expone en la presente memoria, ha sido necesaria la investigación y el desarrollo de diversas tecnologías que se expondrán a continuación, así como el estudio y la asimilación de tecnologías ya existentes con el fin de adaptarlas y modificarlas para el proyecto desarrollado.

Como módulo de procesamiento se ha utilizado una placa BeagleBone Black (puede apreciarse una visión general de esta placa en la [Fig 2.3]), aprovechando la gran comunidad de desarrolladores existentes gracias a la

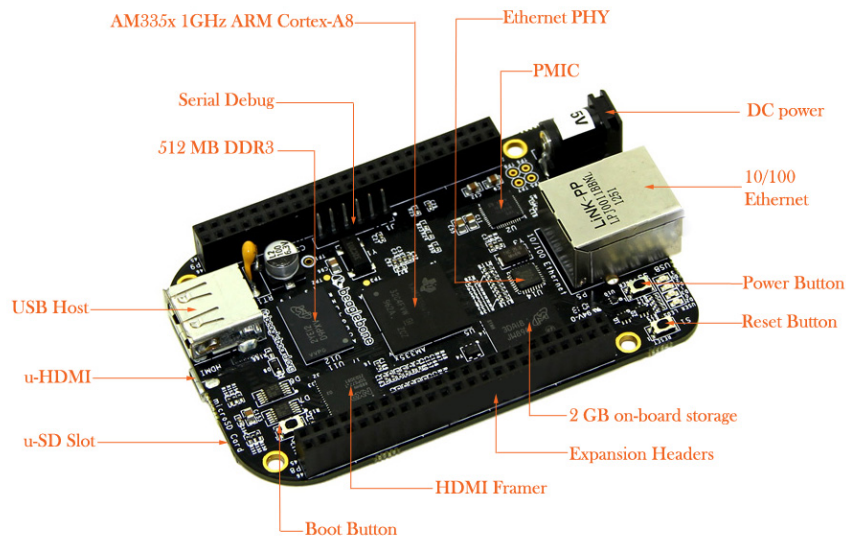


Figura 2.3: Vista general de una BeagleBone Black

cual fue posible resolver los problemas que fueron surgiendo durante el desarrollo, y aprovechando también el bajo costo de la placa así como la multitud de posibilidades que ofrece. En cuanto al tratamiento y captura de señales se ha utilizado el chip ADS1198 de Texas Instruments, que permite unas velocidades de captura suficientemente elevadas para este tipo de desarrollos además de tener un coste asequible. Con motivos de depuración durante el tratamiento de las distintas señales que se han obtenida, se ha utilizado Saleae Logic, un analizador lógico USB que cuenta con 8 canales para cada posible señal. Como método de sincronización entre los dispositivos que lo necesitasen se ha recurrido a una interfaz SPI.

BeagleBone Black

La BeagleBone es una plataforma compacta, de bajo coste y operada por software libre Linux que puede ser usada para complejas aplicaciones en las que intervenga software de alto nivel y circuitos electrónicos de bajo nivel (Molloy, Enero, 2014). Nostros concretamente nos hemos decantado por trabajar con la última versión de la plataforma BeagleBone, es decir, BeagleBone Black (BBB), ya que era la placa que más flexibilidad aportaba al presente proyecto. Las características de la BBB son las siguientes:

- Es muy potente, ya que contiene un procesador que puede realizar hasta 2 billones de instrucciones por segundo (2000 MIPS).
- Tiene un coste bastante asequible, ya que su precio oscila entre los 45\$

y los 55\$.

- Es compatible con muchos estandares de interfaces para dispositivos electronicos (SPI entre otros).
- Es muy eficiente, ya que requiere únicamente entre 1 y 2.3 W, según este inactivo, o funcionando a máxima potencia.
- Es fácilmente ampliable a través de otro tipo de placas de expansión opcionales y de dispositivos USB.
- Tiene una gran comunidad de desarrolladores innovadores y entusiastas que respaldan y dan soporte a los usuarios de la plataforma.
- Es hardware libre, y es compatible con multitud de herramientas y de aplicaciones de software libre.

La plataforma BeagleBone lleva el sistema operativo Linux, lo que significa que es posible usar todo tipo de librerías de software libre, y aplicaciones. La posibilidad de usar software libre, brinda también la oportunidad de conectar esta plataforma con todo tipo de perifericos como pueden ser cámaras USB, teclados, adaptadores Wi-Fi ...

Versiones BeagleBoneBlack

La plataforma BeagleBone y sus placas computadoras han evolucionado progresivamente con el paso de los años, tanto en prestaciones, como en reducción del coste final. A continuación se muestra el progreso de la plataforma en orden histórico.

- **(2008) BeagleBoard (125\$)** La original placa de desarrollo de hardware libre basada en ARM que tiene soporte para video HD. Tiene un Procesador ARM A8 a 720MHz pero no tiene conexión Ethernet.
- **(2010) BeagleBoard xM (149\$)** Similar a la BeagleBoard, solo que tiene un procesador ARM AM37x a 1Ghz, 512MB de memoria, cuatro puertos USB, y soporte para Ethernet. Es una placa muy popular por su núcleo C64+TMDSP para procesamiento de señales digitales.
- **(2011) BeagleBone (89\$)** Más compacta que su antecesora la BeagleBoard. Tiene un procesador a 720Mhz y 256MB de memoria, soporta Ethernet y conexiones de salida de bajo nivel, pero no tiene soporte nativo de video.
- **(2013) BeagleBone Black (45\$-55\$)** Esta placa mejora la BeagleBone con un procesador a 1GHz, 512MB de memoria DDR3, conexión Ethernet, almacenamiento eMMC y soporte para conexión de video HDMI.

	Feature	
Processor	Sitara AM3358BZCZ100	
Graphics Engine	1GHz, 2000 MIPS	
SDRAM Memory	SGX530 3D, 20M Polygons/S	
Onboard Flash	512MB DDR3L 800MHZ	
PMIC	4GB, 8bit Embedded MMC	
Debug Support	TPS65217C PMIC regulator and one additional LDO.	
Power Source	Optional Onboard 20-pin CTI JTAG, Serial Header	
PCB	miniUSB USB or DC Jack	5VDC External Via Expansion Header
Indicators	3.4" x 2.1"	6 layers
HS USB 2.0 Client Port	1-Power, 2-Ethernet, 4-User Controllable LEDs	
HS USB 2.0 Host Port	Access to USB0, Client mode via miniUSB	
Serial Port	Access to USB1, Type A Socket, 500mA LS/FS/HS	
Ethernet	UART0 access via 6 pin 3.3V TTL Header. Header is populated	
SD/MMC Connector	10/100, RJ45	
User Input	microSD , 3.3V	
Video Out	Reset Button	
Audio	Boot Button	
Expansion Connectors	Power Button	
Weight	16b HDMI, 1280x1024 (MAX)	
Power	1024x768,1280x720,1440x900 ,1920x1080@24Hz w/EDID Support	
	Via HDMI Interface, Stereo	
	Power 5V, 3.3V , VDD_ADC(1.8V)	
	3.3V I/O on all signals	
	McASP0, SPI1, I2C, GPIO(69 max), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 4 Serial Ports, CAN0, EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)	
	1.4 oz (39.68 grams)	
	Refer to Section 6.1.7	

Figura 2.4: Especificación hardware BeagleBone Black

La especificación completa de las características de la BBB se muestran con más detalle en la [Fig. 2.4] (elinux.org, 2017)

Bus SPI

El bus SPI (del inglés Serial Peripheral Interface) es un estándar de comunicación síncrono, rápido y bidireccional que permite comunicar dispositivos como la BBB con otros dispositivos en una distancia corta (Molloy, Enero, 2014). Es posible que la comunicación sea bidireccional, ya que tanto la transmisión como la recepción de datos se realizan en buses separados.

En la transferencia de datos se implican distintos dispositivos que desempeñan diferentes funciones. Existe un dispositivo principal, más conocido como maestro, y otro dispositivo denominado esclavo (como mínimo debe

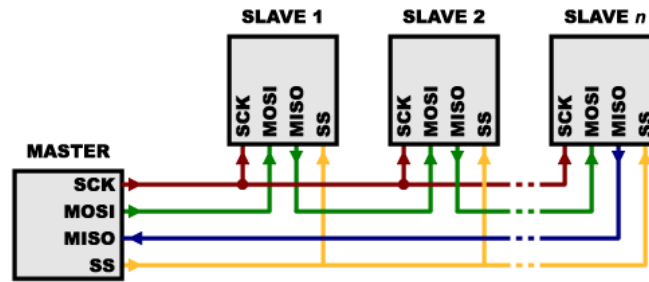


Figura 2.5: Estándar de comunicación SPI

existir uno). El maestro es responsable de enviar las señales de reloj y de seleccionar al esclavo activo en cada instante de la comunicación.

Las diferentes líneas existentes en este estándar de comunicación son las siguientes [Fig 2.5]:

- **MISO** (Master Input Slave Output): Bus de salida de datos de los dispositivos esclavos y de entrada al maestro.
- **MOSI** (Master Output Slave Input): Bus de salida de datos del maestro y entrada a los esclavos.
- **SCK** (Clock o señal de reloj): Pulso de reloj generado por el maestro.
- **SS/Select** (Chip Select): Bus de salida del maestro y entrada a los esclavos, encargado de seleccionar el dispositivo esclavo activo en la comunicación.

Con el fin de procesar una lectura desde un dispositivo esclavo, el maestro debe realizar una escritura en el bus, forzando de esta forma la generación de una señal del reloj que tendrá como consecuencia la escritura de datos por parte del dispositivo deseado.

El dispositivo maestro debe conocer las características de cada esclavo implicado en la comunación, entre las cuales, es posible destacar las siguientes:

- **Frecuencia máxima de transferencia**: La velocidad de comunicación con cada dispositivo estará limitada por este valor, haciendo imposible enviar o recibir datos a mayor velocidad.
- **Polaridad (CPOL)**: Determina la polaridad del reloj [Fig 2.6].
- **Fase (CPHA)**: Determina el flanco de reloj en el que se desencadenará la escritura [Fig 2.6].

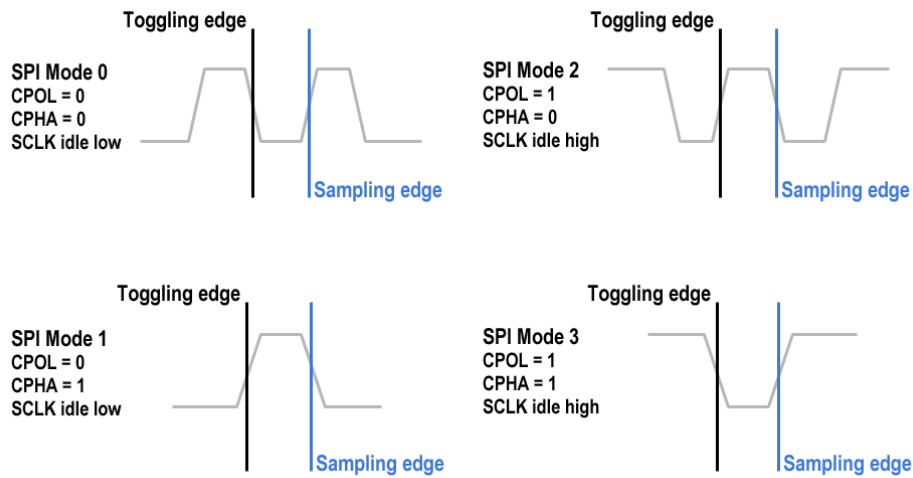


Figura 2.6: Configuraciones posibles de fase y polaridad en el estándar SPI

Chip ADS1198

El ADS1198 es un chip de Texas Instruments de bajo consumo para mediciones de señales ECG, que se caracteriza por tener ocho canales de 16 bits cada uno, una frecuencia de muestreo de hasta 8KHz, un amplificador de ganancia programable, referencia interna, y un oscilador integrado. (Instruments, 2017) Cada canal consta de un multiplexor que permite la lectura desde ocho entradas diferentes, siendo las más relevantes las entradas de temperatura, electrodos, y señal de test generada internamente.

El chip consta de una interfaz SPI para permitir la comunicación con otros dispositivos. Además de las líneas típicas de SPI se proporciona una línea adicional, **DRDY** (Data ready), que indica cuando se tienen nuevos datos válidos preparados para enviar. La lectura de datos se realiza siempre para los 8 canales, devolviendo adicionalmente una cabecera con información de la configuración del chip.

El chip permite obtener la alimentación de manera unipolar o bipolar:

- **Unipolar:** La alimentación unipolar se realiza mediante una entrada de 5V y otra de 3V.
- **Bipolar:** El modo bipolar requiere entradas desde -2.5V hasta 2.5V.

La alimentación que se ha utilizado en este proyecto ha sido unipolar, por ser más fácilmente adaptable a los pines que proporciona la BBB.

Internamente el ADS1198 cuenta con 25 registros que permiten al usuario configurar todas las características programables del chip [Fig 2.7]. Gran

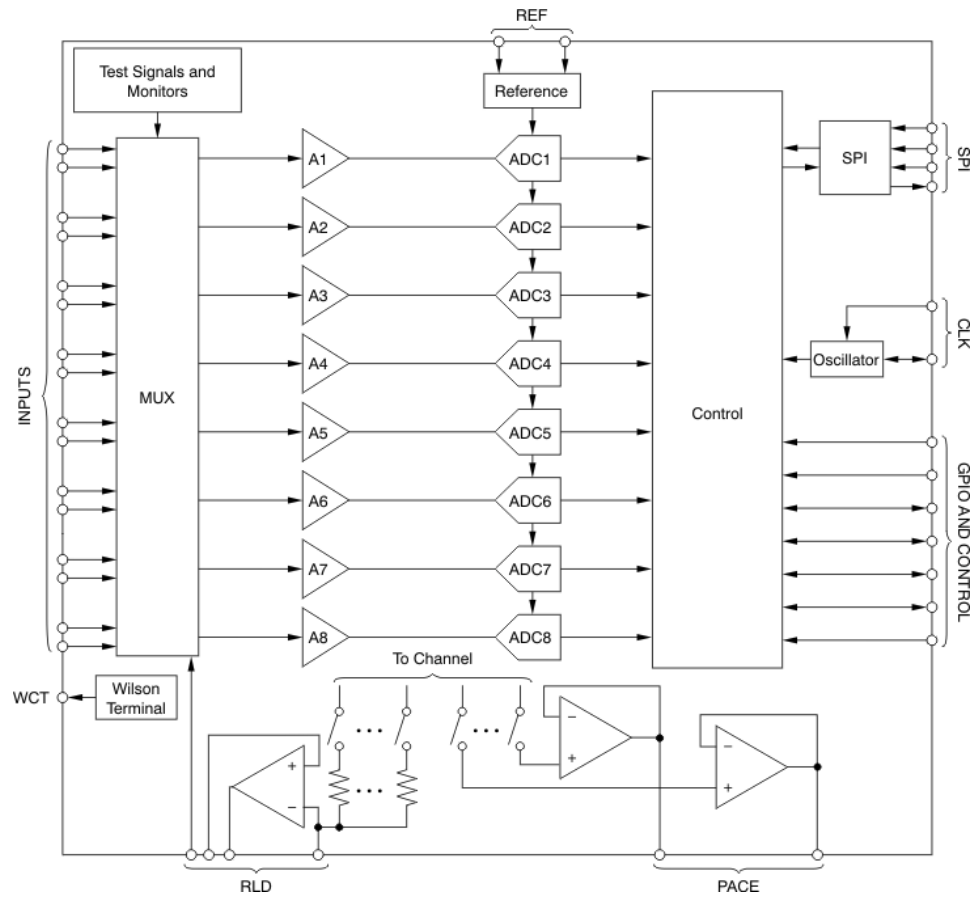


Figura 2.7: Diagrama de funcionamiento interno ADS1198

parte de la configuración permitida está relacionada con aspectos propios de la captura de señales, como las ganancias, el uso de un oscilador interno, o el voltaje de referencia utilizado para la captura. El resto de configuraciones posibles pueden variar la frecuencia de captura, las entradas de los canales o modificar las señales de test internas en amplitud y frecuencia.

Para el propósito del presente proyecto, se ha utilizado un kit de desarrollo, en el que se incluye el chip integrado en una placa que permite configurar la forma de alimentación o tener acceso a las entradas de datos de forma más cómoda y sencilla.

Características técnicas

Dado su elevado rendimiento, alto nivel de integración y bajo consumo, el ADS1198 permite el desarrollo de instrumentación médica de prestaciones elevadas, tamaño reducido y bajo coste. Entre las características técnicas

más destacables, podemos encontrar las siguientes:

- 8 canales de alta resolución.
- Bajo consumo: 0.55mW/canal.
- Frecuencia de muestreo: desde 125Hz hasta 8KHz.
- Ganancia programable.
- Alimentación Unipolar o Bipolar.
- Oscilador y referencia internos.
- Señales de test integradas.
- Comunicación mediante interfaz SPI.
- Rango de temperatura operativo: 0 °C a 70 °C.

Analizador lógico Saleae

Se ha utilizado un analizador lógico, en este caso Saleae, con motivos de depuración durante el desarrollo del proyecto. Gracias al uso de este dispositivo ha sido posible detectar fallos muy concretos en períodos de tiempo relativamente pequeños, ya que de otra forma, en caso de no haberlo utilizado hubiese sido muy complicado el desarrollo, o al menos, hubiese llevado muchísimo más tiempo del esperado.

La idea de funcionamiento de este dispositivo es muy sencilla. Cuenta con 8 canales que actúan como analizadores lógicos, por lo que basta con conectar cada analizador al canal que se desee analizar.

Una vez conectado cada canal, entra en juego el software que proporciona el propio Saleae, que en este caso es multiplataforma ¹. Tras descargarlo e instalarlo bastará con conectar el analizador lógico por usb y configurarlo para comenzar a analizar cada una de las señales.

Entre sus principales características, cabe destacar las siguientes:

- Flexible, software fácil de usar.
- Grandes buffers de muestras.
- Muy portable, conexión USB disponible.
- Analiza 24 protocolos distintos de comunicación.
- Decodificación de protocolos personalizada.

¹Software disponible en <https://www.saleae.com/downloads>

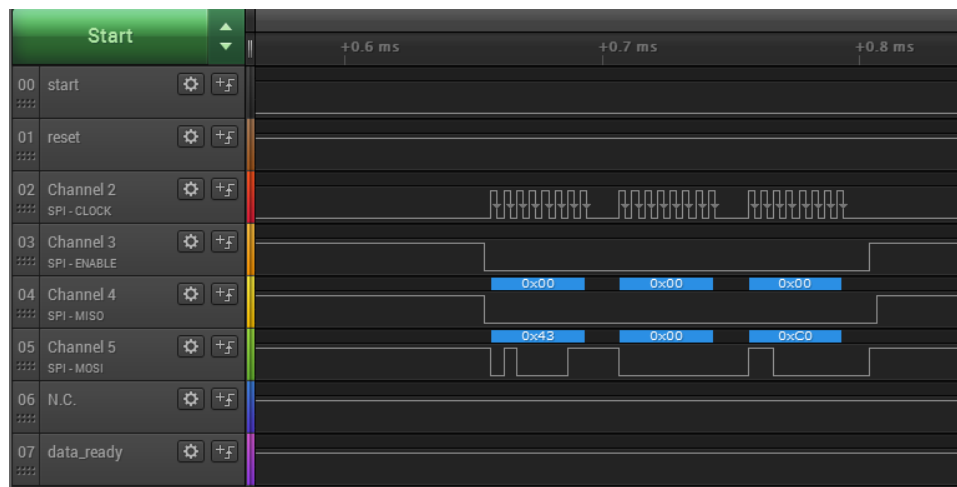


Figura 2.8: Software multiplataforma proporcionado por Saleae

- Formatos de exportación de datos: CSV, Binario, VCD y Matlab.
- Marcadores de medidas, señaladores y temporizadores.

Capítulo 3

Desarrollo

*Quisiera vivir para estudiar,
no estudiar para vivir.*

Sir Francis Bacon

RESUMEN: Aquí va el resumen del capítulo 3

Proyecto inicial

Tras haber realizado la elección del tema en el que se basaría nuestro trabajo de fin de grado, así como tras la elección del director que dirigiría dicho trabajo, estábamos ansiosos de empezar lo antes posible e introducirnos en la materia. Nos pareció apasionante la idea de poder desarrollar software para dispositivos empujados, y aun más, sabiendo que el campo con el que estaría íntimamente relacionado sería la medicina. Tras los sabios consejos que nos transmitió Joaquín sobre qué placa concreta usar, de qué hardware podríamos disponer para la realización del proyecto, así como el software que se nos podría facilitar, estábamos decididos a comenzar. La placa hardware sobre la que desarrollaríamos finalmente se trataba de una BeagleBone Black.

Inicialmente, Joaquín nos proporcionó un pequeño proyecto sobre el que poder comenzar a trabajar y sentar las bases de un proyecto mucho mayor. Este proyecto además podía funcionar correctamente con diversos chips de Texas Instruments, concretamente nosotros escogimos el chip ADS1198. Este chip generaba simulaciones de muestras de forma continua. El código proporcionado era relativamente sencillo y fácil de entender. Se trataba de un programa realizado en C capaz de leer en espacio de usuario del dispositivo (en nuestro caso la BBB). Es decir, era capaz de recibir y tratar toda la información recibida de forma continua por el chip ADS1198. Hasta aquí todo

es relativamente sencillo, gracias al protocolo SPI y al programa de usuario en C era posible gestionar toda la información recibida de las muestras simuladas y realizar el tratamiento pertinente. El único problema hasta aquí es que realmente no era posible gestionar toda la información, si no más bien, casi toda la información.

Objetivos

En ciertos contextos, la pérdida de cierto número de muestras puede ser despreciable y no requerir mayor atención. Sin embargo, en este contexto concreto, al tratarse de procesamiento de señales en tiempo real, la pérdida de un cierto número de muestras, por pequeña que sea, puede ser decisiva a la hora de interpretar la información. El programa inicial escrito en C registraba una pérdida de muestras, que se podía determinar fácilmente hallando la diferencia entre el número de muestras generadas y el número de muestras leídas en el espacio de usuario.

El conjunto total de muestras que se perdía durante la transmisión de información era variable según distintas condiciones y factores externos que influyesen en el momento de la recopilación de información. Generalmente, si únicamente se ejecutaba el programa de usuario sin tener otras aplicaciones que consumiesen muchos recursos tanto de memoria como de cálculo computacional en la BBB, la suma total de muestras perdidas podía encontrarse entre el 1 y el 2% del total de muestras generadas. Sin embargo, si mientras se ejecutaba el programa en C, se ejecutaba algún otro programa que necesitase algún tipo de recurso para ser ejecutado, el número de muestras perdidas por la lectura en espacio de usuario podía dispararse.

Observamos que al ejecutar incluso ciertos comandos de consola, podría llegar a darse la situación anteriormente nombrada. Suponiendo que la ejecución del programa se realiza sobre un sistema operativo anfitrión con alguna distribución linux ejecutándose (en nuestro caso se trataba de una distribución Debian especial para la placa BBB), si tratábamos de ejecutar un comando como `top`¹ (que muestra las tareas que esten ejecutándose en la máquina anfitriona en tiempo real y actualizadolas en intervalos cortos de tiempo como puede apreciarse en la [Fig 3.1]), el número de muestras perdidas aumentaba de forma exponencial. Es decir, para que las pérdidas realmente no fuesen significativas, el dispositivo sobre el que se ejecutase el programa, no debería estar ejecutando nada adicional, o al menos no muchas más aplicaciones. Siendo conscientes de la baja probabilidad de que esta situación se diese en un entorno real, sabíamos que algo teníamos que hacer para evitar esa pérdida de muestras en cualquier situación. Esta pérdida era básicamente producida por interrupciones que lanzaba el propio sistema operativo anfitrión cuando fuese preciso, otorgando la CPU a otros procesos

¹Más información sobre el comando `top` disponible en <https://linux.die.net/man/1/top>

```
top - 10:59:20 up 2 days, 34 min, 1 user, load average: 0,84, 0,64, 0,58
Tasks: 223 total, 2 running, 221 sleeping, 0 stopped, 0 zombie
%Cpu(s): 4,9 us, 2,3 sy, 0,0 ni, 92,9 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem : 5534180 total, 138844 free, 3423732 used, 1971604 buff/cache
KiB Swap: 9705464 total, 9705464 free, 0 used. 1605796 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
32618	cristian	20	0	1252848	458816	129536	R	11,9	8,3	18:07.28	chrome
1456	cristian	20	0	582004	13500	9968	S	5,6	0,2	40:36.96	pulseaudio
32657	cristian	20	0	407872	41732	21632	S	5,3	0,8	7:05.62	chrome
31706	cristian	20	0	1600488	247408	107088	S	2,0	4,5	17:21.52	chrome
1417	cristian	20	0	1613004	288636	76352	S	1,7	5,2	69:10.86	compiz
774	message+	20	0	44520	5416	3468	S	0,3	0,1	0:45.32	dbus-daemon
967	root	20	0	417964	82484	52828	S	0,3	1,5	76:39.37	Xorg
3101	root	20	0	0	0	0	S	0,3	0,0	0:00.86	kworker/3:0
3330	root	20	0	0	0	0	S	0,3	0,0	0:00.18	kworker/0:2
3404	root	20	0	43020	3944	3292	R	0,3	0,1	0:00.31	top
1	root	20	0	185416	6048	3976	S	0,0	0,1	0:07.06	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.05	kthreadd
3	root	20	0	0	0	0	S	0,0	0,0	0:04.33	ksoftirqd/0
7	root	20	0	0	0	0	S	0,0	0,0	1:53.40	rcu_sched
8	root	20	0	0	0	0	S	0,0	0,0	0:00.00	rcu_bh
9	root	rt	0	0	0	0	S	0,0	0,0	0:00.07	migration/0
10	root	0	-20	0	0	0	S	0,0	0,0	0:00.00	lru-add-dr+

Figura 3.1: Ejemplo de salida al ejecutar comando `top` en una terminal linux

que no fuesen el programa que recogía las muestras durante cierto tiempo, suficiente para significar una pérdida representativa.

Sabiendo que el sistema operativo no tenía por qué garantizarnos la CPU para el programa de usuario durante la mayor parte del tiempo que este se encuentra en ejecución, tuvimos que investigar la mejor forma de poder solucionar este problema.

Alternativas

Teniendo en mente el problema anteriormente nombrado, era momento de buscar alternativas para solventarlo. La solución más inmediata que pasó por nuestra cabeza, fue la de sustituir la distribución Debian que utilizabamos en la BBB por una distribución que estuviese pensada para trabajar con análisis y tratamientos de datos en tiempo real. De esta forma, el sistema operativo no interrumpiría el proceso de análisis de muestras otorgando la CPU a otras tareas, y por consiguiente, presumiblemente no se produciría la pérdida de ninguna muestra. Sin embargo esta solución parecía poco sostenible, ya que el hecho de recurrir a una distribución tan específica podría no ser accesible para todo tipo de usuarios y dispositivos, sin olvidar la dependencia que se generaría hacia ese sistema operativo concreto, impidiendo con la más absoluta de las certezas que funcionase exitosamente en otros tipo de sistemas operativos (ya que por decirlo de algún modo, sería como un traje a medida). Sin embargo, si se utilizase una distribución genérica, como el Debian que usabamos en aquel momento, no habría problemas, ya que se trata de una distribución de proposito general a la que cualquier usuario podría recurrir

de manera relativamente sencilla.

Una vez estábamos decididos a mantener el sistema operativo, la solución tenía que encontrarse explorando otras posibles vías. Fue entonces, cuando se nos presentó la posibilidad de usar dos microprocesadores que tenía la BBB, denominadas PRUs (programmable real-time unit). Estos dos microprocesadores con los que contábamos, podían ser programados específicamente para tratar procesos de análisis y recopilación de muestras en tiempo real. De esta forma, aunque el sistema operativo decidiese ceder el uso de la CPU a otro proceso, las PRUs podrían seguir tratando los procesos en tiempo real en segundo plano, sin necesidad de requerir la CPU principal, por lo que la pérdida de muestras se reduciría totalmente gracias a estos pequeños procesadores que incorpora la placa. Se trata de dos procesadores de alta frecuencia (200-MHz) de arquitectura de 32 bits que ofrecen la posibilidad a los desarrolladores de tratar con operaciones en tiempo real.

A continuación se enumeran los motivos por los que escogimos recurrir a las PRUs:

- Tienen acceso a los pins, así como a la memoria interna de la BBB y a los periféricos del principal procesador principal que incorpora.
- Están diseñados para proveer software específico para periféricos como parte del sistema PRU-ICSS (Programmable Real-time Unit Industrial Control SubSystem).
- Son capaces de implementar soluciones relativamente simples a problemas complejos.
- Consta de un gran ancho de banda para comunicarse con la CPU principal y sus controladores, por lo que es improbable que se produzca el fenómeno conocido como cuello de botella.²
- Existen multitud de recursos y proyectos en los que se utilizan estos procesadores, que pueden ser consultados de forma libre y gratuita en la red.

Programación en tiempo real

Abordar la decisión de trabajar con las PRUs no es una tarea trivial. Requiere una gran labor de investigación previa, así como una sólida base en ciertos lenguajes de programación. Además, han sido necesarias tomar ciertas decisiones que han resultado cruciales para el correcto avance y desarrollo

²Se denomina cuello de botella a la situación que se presenta cuando en un proceso productivo, una fase de producción es más lenta que otras, lo que ralentiza el proceso de producción global.

del proyecto, como ha podido ser en qué lenguaje programar las PRUs, o la configuración hardware para trabajar en el desarrollo.

Decisiones de diseño

Cuando comenzó el desarrollo de este proyecto, existían dos posibles alternativas a escoger a la hora de elegir lenguaje con el que trabajar en las PRUs:

- **Posibilidad de programar en C:** Texas instruments introdujo en una de las últimas versiones de Code Composer Studio (Es un entorno de desarrollo integrado que soporta microcontroladores de Texas Instruments y otro tipo de procesadores empujados) ³ la posibilidad de programar en C para programar las PRU. Las herramientas que proporciona Texas Instruments para ello se denominan CGT (Code Generation Tools) y son relativamente sencillas de utilizar.
- **Posibilidad de programar en ensamblador:** Hay dos tipos de ensamblador disponibles para la PRU, `pasm` y `clpru`.
 - `Pasm` es el ensamblador original para la PRU. Este ensamblador soporta una unidad de traducción simple y se monta directamente a una imagen binaria (u otros formatos compatibles).
 - `Clpru` es actualmente una herramienta de compilación completa que incluye ensamblador para la programación de la PRU. Soporta multitud de unidades de traducción y se monta directamente sobre ficheros objetos, los cuales deben ser enlazados al ejecutable final. ⁴

Ante las alternativas que se nos plantearon, la decisión final fue la de programar en ensamblador, concretamente en `pasm`, ya que resultó ser una de las opciones más interesantes para este proyecto (aunque somos conscientes de que en la actualidad no es frecuente encontrarse con una situación en la que se requiera programar en ensamblador). Se optó por esta decisión de diseño, ya que investigar cual de las posibilidades era la que más flexibilidad podía aportar al contexto en el que se situaba el proyecto, `pasm` resultó aportar un mayor control y una valiosa posibilidad de optimizar el código final.

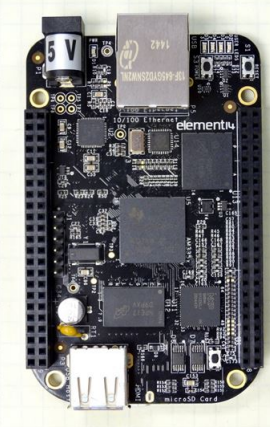
Configuración hardware

Tal y como indicábamos en la sección 2.2.1, la BBB es un dispositivo que aporta una gran flexibilidad a los desarrollares, gracias a la facilidad

³Más información sobre este entorno de desarrollo en <http://www.ti.com/tool/ccstudio>

⁴Cabe destacar que tanto `pasm` como `clpru` soportan prácticamente la misma sintaxis.

Beaglebone Black Pinout Diagram									
P9					P8				
Function	Physical Pins		Function		Function	Physical Pins		Function	
DGND	1	2	DGND		DGND	1	2	DGND	
VDD 3.3 V	3	4	VDD 3.3 V		MMC1_DAT6	3	4	MMC1_DAT7	
VDD 5V	5	6	VDD 5V		MMC1_DAT2	5	6	MMC1_DAT3	
SYS 5V	7	8	SYS 5V		GPIO_66	7	8	GPIO_67	
PWR_BTN	9	10	SYS_RESET		GPIO_69	9	10	GPIO_68	
UART4_RXD	11	12	GPIO_60		GPIO_45	11	12	GPIO_44	
UART4_TXD	13	14	EHRPWM1A		EHRPWM2B	13	14	GPIO_26	
GPIO_48	15	16	EHRPWM1B		GPIO_47	15	16	GPIO_46	
SPI0_CS0	17	18	SPI0_D1		GPIO_27	17	18	GPIO_65	
I2C2_SCL	19	20	I2C_SDA		EHRPWM2A	19	20	MMC1_CMD	
SPI0_DO	21	22	SPI0_SCLK		MMC1_CLK	21	22	MMC1_DAT5	
GPIO_49	23	24	UART1_TXD		MMC1_DAT4	23	24	MMC1_DAT1	
GPIO_117	25	26	UART1_RXD		MMC1_DATA0	25	26	GPIO_61	
GPIO_115	27	28	SP11_CS0		LCD_VSYNC	27	28	LCD_PCLK	
SP11_DO	29	30	GPIO_112		LCD_HSYNC	29	30	LCD_AC_BIAS	
SP11_SCLK	31	32	VDD_ADC		LCD_DATA14	31	32	LCD_DATA15	
AIN4	33	34	GND_ADC		LCD_DATA13	33	34	LCD_DATA11	
AIN6	35	36	AIN5		LCD_DATA12	35	36	LCD_DATA10	
AIN2	37	38	AIN3		LCD_DATA8	37	38	LCD_DATA9	
AIN0	39	40	AIN1		LCD_DATA6	39	40	LCD_DATA7	
GPIO_20	41	42	ECAPWMO		LCD_DATA4	41	42	LCD_DATA5	
DGND	43	44	DGND		LCD_DATA2	43	44	LCD_DATA3	
DGND	45	46	DGND		LCD_DATA0	45	46	LCD_DATA1	



LEGEND	
	Power, Ground, Reset
	Digital Pins
	PWM Output
	1.8 Volt Analog Inputs
	Shared I2C Bus
	Reconfigurable Digital

Figura 3.2: Diagrama de conexiones físicas disponibles en una BBB

que supone ampliar su funcionalidad con multitud de tipos de placas de expansión opcionales.

Cada uno de los pines físicos que incorpora la placa tiene una función determinada como puede apreciarse en la [Fig 3.2]. La BBB dispone de dos cabeceras completas (conocidas como P8 y P9) con multitud de pines disponibles, que permiten realizar conexiones físicas mediante cableado. La leyenda de la [Fig 3.2] muestra las funciones, o mejor dicho, las posibles funciones de los distintos pines:

- Para empezar, se han destacado en color rojo los pines de 5, 3 y 1.8 *Voltios*, así como los pines de tierra (DGND). Hay que tener en cuenta que VDD_ADC es un pin de 1.8 V que se usa para proporcionar una referencia para las funciones de lectura analógica.
- Los pines de propósito general de entrada y salida (GPIO) están destacados en color verde. Cabe destacar que algunos de estos pines se pueden usar para comunicación serie (UART) ⁵.

⁵Más información sobre UART e I2C disponible en <https://geekytheory.com/puertos-y-buses-1-i2c-y-uart>


```

/dts-v1/;
/include/ "common.dtsi";

/ {
    node1 {
        a-string-property = "A string";
        a-string-list-property = "first string", "second string";
        a-byte-data-property = [0x01 0x23 0x34 0x56];
        cousin: child-node1 {
            first-child-property;
            second-child-property = <1>;
            a-string-property = "Hello, world";
        };
        child-node2 {
        };
    };
    node2 {
        an-empty-property;
        a-cell-property = <1 2 3 4>; /* each number (cell) is a uint32 */
        child-node1 {
            my-cousin = <&cousin>;
        };
    };
};

```

Figura 3.3: Ejemplo simple de un fichero .dts

- Si se desea simular una salida analógica comprendida entre 0 y 3.3 V, se pueden usar los pines PWM destacados en morado.
- Los pines destacados en color azul pueden ser utilizados como entradas analógicas ⁶.
- Los pines en color naranja clarito pueden ser usados para I2C ⁵.
- Los pines destacados en color naranja oscuro son fundamentalmente usados para aplicaciones de pantalla LCD.

Device Tree

Un Device Tree (DT) es una descripción del hardware de un sistema. Debería incorporar el nombre de la CPU, la configuración de memoria, y cualquier periférico (interno y externo). Un DT no debería ser usado para describir software, a pesar de que listar los módulos hardware pueda causar que estos se carguen. Es preciso recordar que los DTs son neutros en lo que se refiere al sistema operativo, es decir, no deberían incluir nada específico de Linux por ejemplo.

Un DT representa la configuración hardware como si de una jerarquía de nodos se tratase. Cada nodo puede contener propiedades y subnodos.

⁶Estas entradas analógicas toleran voltajes comprendidos entre los 0 y los 1.8 Voltios, no soportan voltajes superiores a los 1.8 V

Las propiedades se denominan arrays de bytes, los cuales pueden contener strings, numeros (big-endian), secuencias arbitrarias de bytes, y cualquier combinación de estos. Por analogía con un sistema de ficheros, los nodos son directorios, y las propiedades son ficheros.

Los DT generalmente se encuentran en un formato textual conocido como Device Tree Source (DTS) y son almacenados en ficheros con extensión `.dts`. La sintaxis DTS es como la de C, con llaves para agrupar código y punto y coma para concluir cada línea. En la [Fig 3.3] puede apreciarse un ejemplo muy simple de este tipo de ficheros.

Cabe destacar que los DTS requieren de punto y coma tras el cierre de una llave. El formato del binario compilado es denominado Flattened Device Tree (DFT) o Device Tree Blob (DTB), y se almacena en ficheros con extensión `.dtb`.

Device Tree Overlay

Un SoC (System-on-Chip moderno) moderno es un dispositivo muy complejo; un DT completo podría suponer cientos de líneas de código. Situar un SoC en una placa junto a otros componentes solo hace que las cosas sean aun más complejas. Para que sea relativamente manejable, especialmente si hay dispositivos relaciones que comparten componentes, tiene sentido diferenciar los elementos comunes en ficheros con extensión `.dtsi`, para que sean incluidos desde los ficheros `.dts` que los requiriesen.

Pero cuando una placa como la BBB es compatible con accesorios expansibles, el problema puede ser aun más complejo. En última instancia, cada configuración posible requiere un DT para ser descrita, pero una vez que se tenga en cuenta el hardware de base y las expansiones que requieren el uso de unos pines GPIO determinados que pueden ser compartidos por otras configuraciones diferentes, el número de combinaciones posibles comienza a multiplicarse rápidamente.

Lo que se necesita es una forma de describir estos componentes opcionales utilizando un DT parcial, y luego poder construir un árbol completo tomando un DT como base y añadiendo una serie de elementos opcionales. Estos elementos opcionales son denominados *overlays*. Al DT que se forma siguiendo este proceso se le denomina Device Tree Overlay (DTO).

Para nuestro proyecto, fue necesaria la creación y personalización de un DTO; la base sobre la empezamos esta creación fue la que se utiliza en el capítulo 13 de Exploring BBB (Molloy, Enero, 2014). A partir de ahí se realizaron diversas modificaciones y expansiones para obtener un DTO hecho a medida acorde con las necesidades del proyecto. El resultado final del DTO que se obtuvo finalmente y que es utilizado actualmente en la BBB se ilustra en la [Fig 3.4]. El código final fue comentado de modo que pudiese ser interpretado y comprendido de forma sencilla. Como puede apreciarse en

```

/dts-v1/;
/plugin/;
/ {
    compatible = "ti,beaglebone", "ti,beaglebone-black";
    part-number = "EBB-PRU-ADC2";
    version = "00A0";
    /* This overlay uses the following resources */
    exclusive-use =
        "P9.24", "P9.25", "P9.27", "P9.28", "P9.29", "P9.30", "P9.31", "P8.46", "pru0", "pru1";
    fragment@0 {
        target = <&am33xx_pinmux>;
        __overlay__ {
            pru_pru_pins: pinmux_pru_pru_pins { // The PRU pin modes
                pinctrl-single,pins = <
                    0x184 0x2e // DATA_READY P9_24 pr1_pru0_pru_r31_16, MODE6 | INPUT | DIS 00101110=0x2e
                    0x1ac 0x0d // START P9_25 pr1_pru0_pru_r30_7, MODE5 | OUTPUT | DIS 00001101=0x0d
                    0x190 0x0d // RESET P9_31 pr1_pru0_pru_r30_0, MODE5 | OUTPUT | DIS 00001101=0x0d
                    0x1a4 0x0d // CS P9_27 pr1_pru0_pru_r30_5, MODE5 | OUTPUT | DIS 00001101=0x0d
                    0x19c 0x2e // MISO P9_28 pr1_pru0_pru_r31_3, MODE6 | INPUT | DIS 00101110=0x2e
                    0x194 0x0d // MOSI P9_29 pr1_pru0_pru_r30_1, MODE5 | OUTPUT | DIS 00001101=0x0d
                    0x198 0x0d // CLK P9_30 pr1_pru0_pru_r30_2, MODE5 | OUTPUT | DIS 00001101=0x0d
                    // This is for PRU1, the sample clock -- debug only
                    0x0a4 0x0d // SAMP P8_46 pr1_pru1_pru_r30_1, MODE5 | OUTPUT | DIS 00001101=0x0d
                >;
            };
        };
    };
    fragment@1 { // Enable the PRUSS
        target = <&pruss>;
        __overlay__ {
            status = "okay";
            pinctrl-names = "default";
            pinctrl-0 = <&pru_pru_pins>;
        };
    };
};

```

Figura 3.4: Device Tree Overlay del proyecto para la BBB

la imagen, el primer paso es indicar cuales van a ser los pines que se van a utilizar de forma exclusiva. A continuación se le asigna un modo a cada uno de los pines anteriormente indicado. El modo que se asigna, la señal que va a ir asociada a ese pin concreto y otra información complementaria puede ser encontrada en el código comentado. Cabe destacar que este tipo de ficheros `.dts` sigue siempre una misma estructura en lo que al código respecta, aunque las posibles combinaciones de configuraciones hardware son prácticamente infinitas.

Conexiones físicas

Una vez diseñada la configuración hardware, era momento de realizar las conexiones físicas mediante cableado. Entre los objetivos que han de satisfacer estas conexiones físicas, cabe destacar los siguientes:

- Alimentar el ADS1198, ya que se alimenta a través de la BBB.
- Posibilitar la conexión SPI entre ambos dispositivos (ADS1198 y BBB).
- Garantizar el uso de pines con fines GPIO (para cada una de las distintas señales).
- Conectar el analizador lógico Saleae con cada una de las señales que se desee depurar.

Tabla 3.1: Conexiones físicas necesarias para la alimentación del chip ADS1198

BBB Pin	ADS1198 Pin	Uso de la conexión
P9 Pin 1	J4 Pin 5	GND
P9 Pin 3	J4 Pin 9	3.3V
P9 Pin 5	J4 Pin 10	5V

Tabla 3.2: Conexiones físicas de carácter general GPIO

BBB Pin	ADS1198 Pin	Señal asociada a la conexión
P9 Pin 23	J3 Pin 15	DATA_READY
P9 Pin 25	J3 Pin 8	RESET
P9 Pin 27	J3 Pin 14	START

Tabla 3.3: Conexiones físicas para posibilitar la conexión SPI

BBB Pin	ADS1198 Pin	Señal asociada a la conexión
P9 Pin 29	J3 Pin 13	SPI_10
P9 Pin 30	J3 Pin 11	SPI_11
P9 Pin 31	J3 Pin 3	SPI_SCLK
P9 Pin 28	J3 Pin 1	SPI_CS0

Tabla 3.4: Conexiones físicas del analizador lógico Saleae

Canal	ADS1198 Pin	Señal asociada a la conexión
CH0	J3 Pin 15	DATA_READY
CH1	J3 Pin 3	SCLK
CH2	J3 Pin 13	?
CH3	J3 Pin 11	?
CH4	J3 Pin 1	?
CH5	J3 Pin ?	?
CH6	J3 Pin ?	?
CH7	J3 Pin ?	?

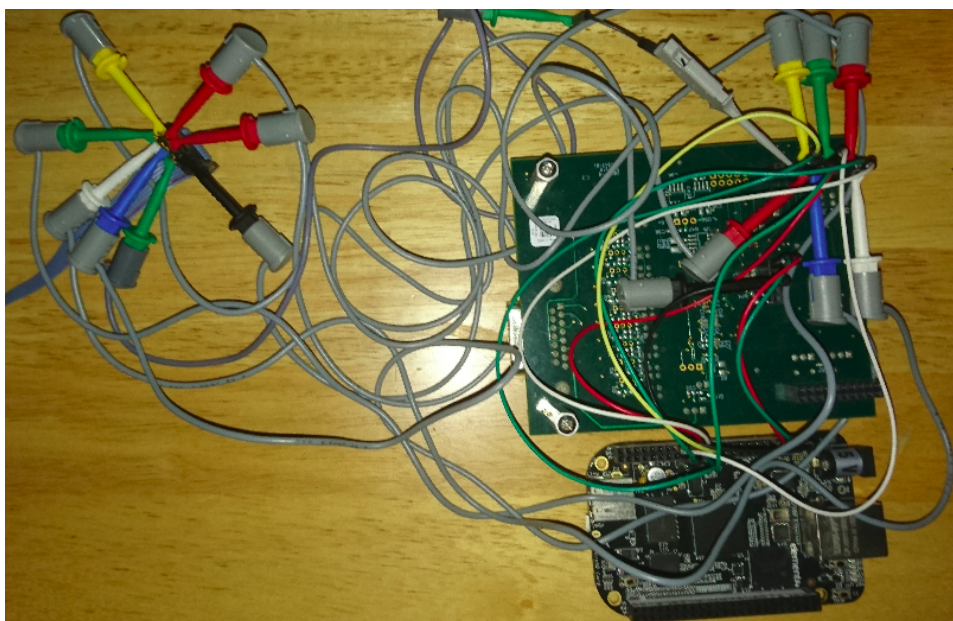


Figura 3.5: Conexiones físicas necesarias para el proyecto

Una vez se han realizado las conexiones indicadas en las tablas 3.1, 3.2, 3.3 y 3.4, puede considerarse que la configuración hardware ha concluido con éxito. En nuestro caso, el resultado de completar todas las conexiones físicas puede apreciarse en la [Fig 3.5].

Bibliografía

*Y así, del mucho leer y del poco dormir,
se le secó el cerebro de manera que vino
a perder el juicio.*

Miguel de Cervantes Saavedra

ELINUX.ORG. *BeagleBone Black*. Versión electrónica, 2017. Disponible en <http://elinux.org/Beagleboard:BeagleBoneBlack> (último acceso, Mayo, 2017).

INSTRUMENTS, T. *ADS1198*. Versión electrónica, 2017. Disponible en <http://www.ti.com/product/ADS1198> (último acceso, Mayo, 2017).

MOLLOY, D. *Exploring BeagleBone*. Versión electrónica, Enero, 2014. Disponible en <http://exploringbeaglebone.com/> (último acceso, Mayo, 2017).

Lista de acrónimos

*–¿Qué te parece desto, Sancho? – Dijo Don Quijote –
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.*

*Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes*

*–Buena está – dijo Sancho –; fírmela vuestra merced.
–No es menester firmarla – dijo Don Quijote–,
sino solamente poner mi rúbrica.*

*Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes*

