UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

isa

ai2
INSTITUTO DE AUTOMÁTICA E INFORMÁTICA INDUSTRIAL

API para la interacción de sensores y actuadores de robot Bioloid, basado en BeagleBone Black

Autor: Sisternes Roses, David.

Tutor: Blanes Noguera, Juan Francisco.

Titulación: Máster Universitario en Automática e Informática Industrial.

INDICE GENERAL DEL CONTENIDO

Resumen

En la actualidad se cuenta con muchas aplicaciones realizadas para robots humanoides y gran variedad de placas de desarrollo. En éste caso se dispone del robot Bioloid, el cual dispone del controlador CM530 con un microcontrolador cuya capacidad de cómputo es limitada. Una capacidad insuficiente, por ejemplo, si se requiere de un sistema de visión dando paso a la utilización de una placa de mayor capacidad como es la BeagleBone Black. A partir de esta premisa y junto con la necesidad de controlar el Bioloid con un sistema empotrado, nace la necesidad de implementar una aplicación que permita la conexión de los dos elementos.

De este modo, se introduce el principal propósito del siguiente trabajo, cuya finalidad radica en la implementación de una API de interacción con sensores y actuadores del robot Bioloid que facilite el desarrollo de aplicaciones sobre BeagleBone Black.

En líneas generales, se trata de desarrollar tres capas de abstracción donde se solucionen las cuestiones de sensores y actuadores, comunicación TTL y por último, los movimientos del Bioloid.

El resultado final es la implementación de una aplicación, donde se plasme el uso de todas las capas que conforman el API, además de verificar su funcionalidad. Siendo este desarrollo por módulos independientes, para que aquellos interesados con conocimientos puedan contribuir aportándole nuevas funcionalidades y mejoras a cada módulo.

Palabras clave: Robot Humanoide, Comunicación Serie, API programación.

Resum

En l'actualitat existeixen moltes aplicacions realitzades per a robots humanoides y una gran varietat de plaques de desenvolupament. En aquest cas es disposa del robot Biolodid, el qual disposa del controlador CM530 amb un microcontrolador on la capacitat de còmput és llimitada. Una capacitat insuficient, per exemple, si es requereix d'un sistema de visió donant pas a la utilització d'una placa de major capacitat com es la BeagleBone Black. A partir d'aquesta premissa i junt amb la necessitat de controlar el Bioloid amb un sistema encastat, naix la necessitat de implementar una aplicació que permeta la connexió del dos elements.

D'aquesta forma, s'introdueix el principal propòsit del següent treball, on la finalitat radica en la implementació d'una API d'interacció amb sensors i actuadors del robot Bioloid que facilite el desenvolupament d'aplicacions sobre BeagleBone Black.

En línies generals, es tracta de desenvolupar tres capes d'abstracció on es dóne solució a les qüestions de sensors i actuadors, comunicació TTL y per últim, els moviments del Bioloid.

El resultat final és la implementació d'una aplicació, on es plasme l'ús de totes les capes que conformen l'API, a més de verificar la seva funcionalitat. Sent aquest desenvolupament per mòduls independents, per a que aquells interessats amb coneixements puguin contribuir aportant-li noves funcionalitats i millores per a cada mòdul.

Paraules clau: Robot Humanoide, Comunicació Sèrie, API programació.

Abstrac

Nowadays, several applications have been developed to create humanoid robots and a wide range of development boards. In this particular case, we have worked with the Bioloid robot which is formed by CM530 controller with a microcontroller that holds a limited computing capacity. In the case we need a vision system, this would result in an insufficient capacity, therefore, a higher capacity board is used, the BealgeBone Black. Based on this premise and together with the need to control the Bioloid with an embedded system, it emerges the need to implement an application that interconnects both elements.

Thus, the objective of this paper is the implementation of a Programming API connected to sensors and actuators from the Bioloid Robot. In this way, a new mechanism that makes possible the development of new applications in BeagleBone Black is presented.

In general terms, we have developed three layers of abstraction which have to be used to solve issues related to sensors and actuators, TTL communication and Bioloid movements.

As a final result, we can see the implementation of an application that shows the use of each of the layers which make up the API and its functionality proof.

A development made of independent units, which allows people who are interested to provide new and better functionalities to each unit.

Keywords: Humanoid Robot, Serial communication, Programming API.

1. Memoria

Autor: Sisternes Roses, David.

Tutor: Blanes Noguera, Juan Francisco.

Titulación: Máster Universitario en Automática e Informática Industrial.

ÍNDICE DEL CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

El siguiente proyecto forma parte del Trabajo de Fin de Máster requerido para la finalización del Máster en Automática e Informática Industrial, para la posterior obtención de la titulación en la Universidad Politécnica de Valencia.

El proyecto se titula: "*API para interacción de sensores y actuadores de robot Bioloid, basado en BeagleBone Black*".

2. OBJETO

La propuesta del siguiente trabajo se plantea ante la necesidad de posibilitar el control de un robot Bioloid desde un sistema empotrado basado en BeagleBone Black. De esta forma, siguiendo otras implementaciones existentes para otras plataformas se propone llevar a cabo la implementación de una API que permita comunicar sensores y actuadores ofreciendo un fácil manejo del mismo a las tareas de control del mismo.

El principal objetivo se centra en el diseño y desarrollo de una API de interacción con los sensores y actuadores del robot Bioloid para facilitar el desarrollo de aplicaciones sobre BeagleBone Black de control del robot.

Esto se llevará a cabo mediante:

- Diseño y desarrollo de una capa de abstracción de las comunicaciones.
- Diseño y desarrollo de una capa de abstracción de sensores y actuadores
- Diseño y desarrollo de una capa de abstracción de movimiento.
- Validación de las diferentes API desarrolladas, a través de diversos test.

3. ANTECEDENTES

En la actualidad se puede observar con facilidad el creciente protagonismo que está adquiriendo la robótica junto con el desarrollo de sistemas automáticos en el que los robots constituyen ya, una herramienta esencial para el avance y funcionamiento de estos sistemas.

No obstante, existen diversos problemas entre los que ocupa especial importancia la toma de decisiones en función de la situación del medio, ante lo que surge la necesidad de dotar a los sistemas con elementos que permitan conocer su estado y actuar en determinación de éste. Para ello se emplean los llamados sensores y actuadores, componentes en constante renovación y desarrollo, que facultan la elaboración de sistemas cada vez más complejos.

Por otra parte, resultaría imposible la puesta en marcha de sistemas robóticos sin la mediación de los microcontroladores que conforman un elemento primordial en cualquier sistema automático/robótico. Éstos, al igual que los sensores y actuadores, permanecen en continuo desarrollo ofreciendo de este modo beneficiosas prestaciones que permiten dar cabida a un alto procesamiento de la información generada por los sensores/actuadores y facilitando el uso de los mismos.

Por consiguiente, se cuenta con cualificados y desarrollados sistemas de comunicación entre sensores y actuadores en el ámbito robótico. Se habla sin embargo, de un desarrollo que nada tiene que ver con los avances en robot Bioloid y BeagleBone Black, los cuales muestran unas condiciones que demandan de nuevas y mejores investigaciones y que han acabado suponiendo un reto tecnológico al que se va a tratar de hacer frente en la elaboración y desarrollo del siguiente proyecto.

Un proyecto capaz de abarcar diversas aplicaciones que manejará herramientas open source (código libre) con el fin de facilitar su mantenimiento y permitir la realización de las modificaciones que se crean necesarias. Para ello se tratará de realizar el trabajo con la mejor modularidad posible.

4. HARDWARE UTILIZADO

En el siguiente punto se detallará el material hardware que se ha utilizado en la elaboración de éste proyecto y que ha permitido el cumplimiento de los objetivos establecidos. En este caso se trata del Kit Boiloid Premium de Robotis, junto con los sensores y actuadores incluidos en éste. Empleando además, la palca de bajo coste BeagleBone Black y ROBOcape, junto a nuevos componentes que serán posteriormente explicados.

4.1 Kit Bioloid Premium

Figura 1: Robot Bioloid

En primer lugar se muestra el robot Bioloid de la marca surcoreana Robotis. Ofrece un Kit apto para aquellos usuarios que centren su interés en la robótica humanoide, brindándoles la capacidad de armar y programar sin gran dificultad independientemente de los conocimientos que posean sobre éste.

Por ejemplo, los servomotores Dynamixel, han sido diseñados específicamente para este fin, presentando una serie de características óptimas para el desarrollo de movimientos en el robot, como la llamada retroalimentación “feedback”, que posibilita la realización de un control en bucle cerrado, tanto si es de posición, como de velocidad o Torque.

A través del protocolo TTL (half-duplex) que utilizan, admiten tanto la escritura como la lectura de parámetros situados en una tabla de control,

proporcionando así el conocimiento de la información necesaria para el usuario.

Por otro lado, el Kit ofrece la posibilidad de realizar el ensamblaje de tres tipos de androide (tipo A, B o C), ofreciendo un punto de adaptación al problema que se quiera abordar. En este caso, se va a realizar el montaje tipo A, que será posteriormente analizado.

Se dispone además, de una serie de sensores los cuales ayudan al robot a conocer ciertas características del entorno que lo rodea. Se habla entonces de los sensores de DMS-80(distancia), GiroSensor GS-12(velocidad Angular), IR-Sensor (Distancia corta) y Ir-Receiver. Para el desarrollo de la capa de abstracción de sensores y actuadores se utilizarán los sensores DMS-80 y el GiroSensro GS-12, ambos incluidos en este Kit, seguido del uso de un sensor de distancia con ultrasonidos (HC-SR04) ajeno al proyecto.

Por último, queda nombrar el controlador CM530 que se encuentra en la caja a partir del cual, se dispone de un microcontrolador de 32bits stm32f103 con un abanico de excelentes prestaciones que lo hacen apto para este tipo de sistemas. No obstante, en el caso que aquí se presenta y como se ha referido anteriormente se hará uso de la placa de bajo coste BeagleBone Black que posee las prestaciones idóneas.

A continuación se explicarán y expondrán los componentes del Kit Bioloid empleados para el desarrollo del trabajo:

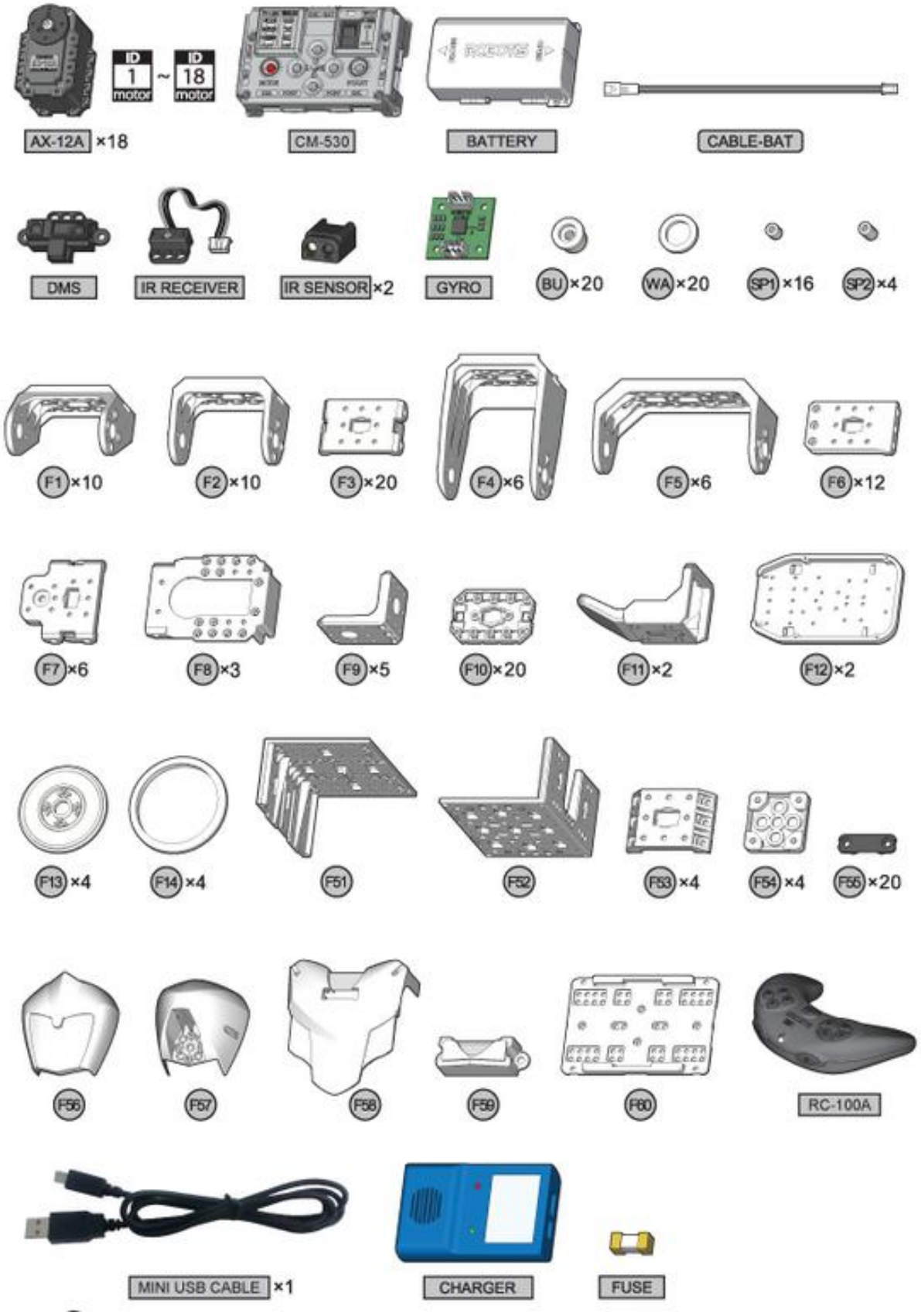


Figura 2: Componentes Kit Bioloid 1

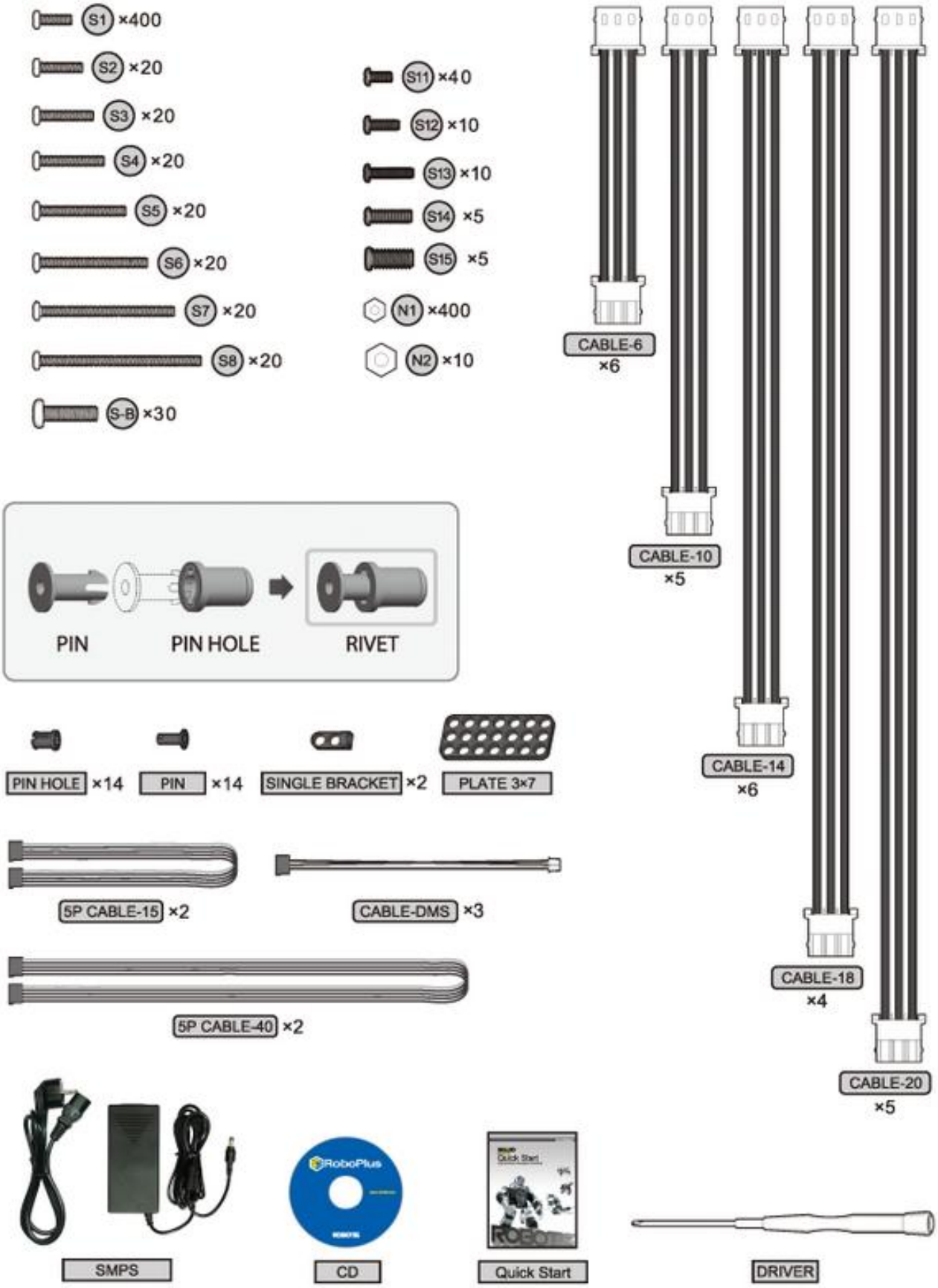


Figura 3: Componentes Kit Bioloid 2

Figura 4: Dynamixel AX-12A

El primer elemento del que se ha dispuesto en el trabajo, alude al servomotor Dynamixel AX12a, entre cuyas principales características destacan: la posibilidad de configurar cada uno individualmente con los parámetros que precisen, así como permitir el control retroalimentado de la posición, velocidad, torque y corriente del mismo.

Además cuenta con:

- Giro de 0º - 300º
- Resolución de 0.29º
- Conexión Fisica: TTL Level Multi Drop
- Tipo de protocolo: comunicación serie asíncrona “half-dúplex” (8bits, 1stop, no parity)
- Velocidad de comunicación: 7343bps - 1 Mbps
- Tensión de operación: 9V - 12V (Recomendado 11.1V)
- Permite lectura de un amplio conjunto de parámetros como Posición, Velocidad, Torque, Corriente, Temperatura, Slope y Margim.

Figura 5: Sensor DMS (Sharp 2y0A21)

Otro de los elementos utilizados de este Kit, es el sensor de distancia DMS la marca Sharp, concretamente el modelo 2y0A21. Este sensor ofrece las siguientes características:

- Distancia detección: 10 ~ 80 Cm
- Tensión de operación: 4.5 – 5.5V
- Tensión de salida: 0.4 ~ 3.3 V

La siguiente gráfica permite visualizar la curva que describe el sensor (Salida Analógica Vs Distancia en cm):

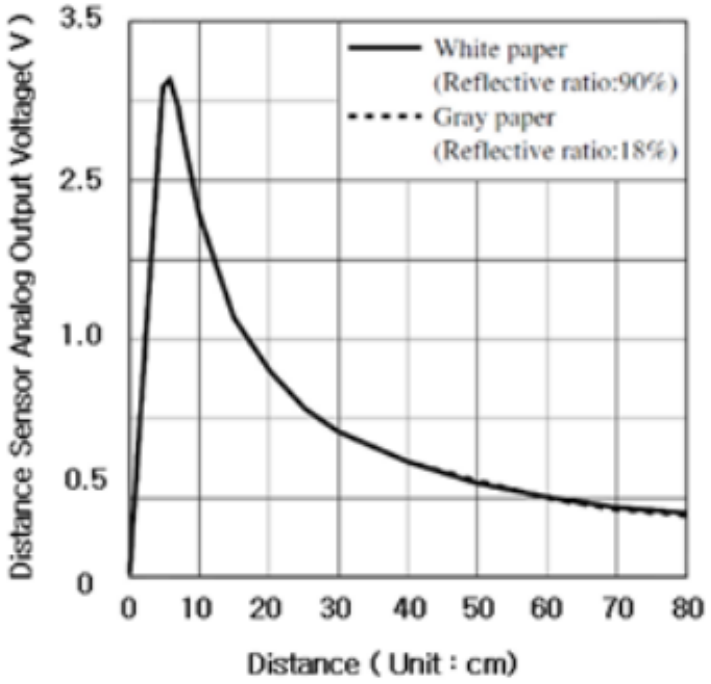


Figura 6: Gráfica Sharp 2y0A21

No obstante, la (**Figura 6**), presenta el inconveniente de requerir la realización de una linealización para obtener valores de distancia correctamente.

El tercer elemento que utilizado para el proyecto, es el sensor GyroSensor GS-12, que permite medir la velocidad angular en el eje Y y X del robot conociendo de este modo si éste sufre alguna caída. Concretamente este sensor ofrece:

- Tensión de operación: 5V
- Tensión de salida: [2.23 - 1.23 – 0.23] V

Velocidad Angular	+300 °/s	←	0 °/s	→	-300 °/s
Voltaje de Salida	2.23 V		1.23 V		0.23 V

Figura 7: Tabla GyroSensor GS-12

El cuarto y principal elemento de este Kit es el robot Bioloid para el cual se va a crear el API. En el Kit, siguiendo el libro de ensamblaje se ofrecen hasta tres tipos de robot diferentes en función de las labores que se van a llevara a cabo.

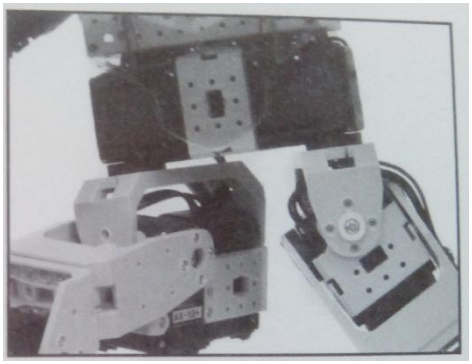


Figura 8: Distribución tipo A Bioloid

El tipo A contiene 18 servos Dynamixels, un peso de 1.7Kg y una altura de 39,7 cm. A diferencia de los otros dos tipos, éste ofrece Pasos estables y omni-direccionales.

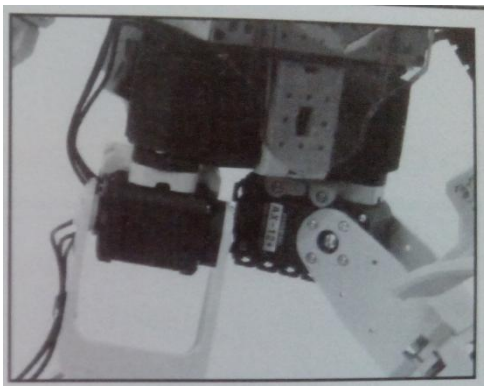


Figura 9: Distribución tipo B Bioloid

El tipo B, contiene 16 servos, un peso de 1.6Kg y una altura de 39.7Cm. La diferencia de este tipo, reside en la capacidad de cambiar de dirección mientra marcha, disponiendo además de dos Dynamixels para otra función.

Figura 10: Distribución tipo C Bioloid

El tipo C al igual que el anterior, contiene 16 servos, pesa 1.6Kg y tiene una altura de 38.6cm. En este caso la diferencia se encuentra en la capacidad de ir de lado mientras marcha y disponiendo también de dos Dynamixels para otras funciones.

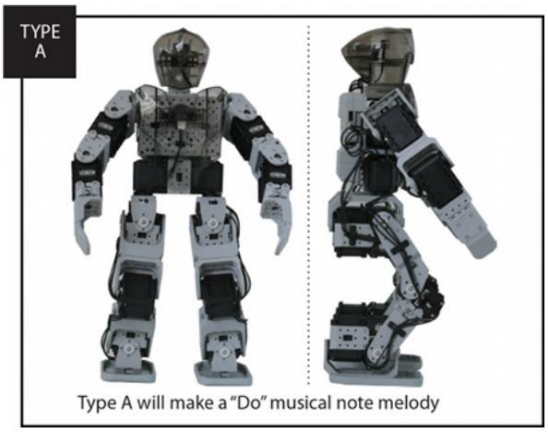


Figura 11: Robot Bioloid tipo A

Finalmente se opta por la distribución del tipo A siendo ésta la que mejor adaptabilidad muestra al tipo de movimientos que se pretenden realizar y sirviendo además para usos posteriores.

Figura 12: USB2Dynamixel.

Por consiguiente, otro de los componentes de este Kit que se utiliza, es el llamado USB2Dynamixel y que su utilidad reside en la configuración de los servos entre otras cosas mediante el software de Robotis llamado Roboplus. El USB permite la conexión RS-485, TTL y RS-232.

Figura 13: Batería Lipo 1000mAh para Bioloid y Cargador de baterías lipo.

Por último, se va a utilizar como sistema de alimentación la batería recargable lipo de dos celdas de 12V y capacidad de 1000mAh, junto con su correspondiente cargador de baterías lipo.

Figura 14: Sensor de distancia HC-SR04.

Con tal de dar por finalizando este punto se ha adoptado además, por incluir en el pack de sensores el sensor HC-SR04 siendo éste de fácil

obtención e instalación y a lo que se le suma la obtención de unos resultado favorables.

Se trata de un sensor de distancia que funciona por ultrasonidos. Un haz de ondas es lanzado y se espera el rebote de las ondas. Mediante el tiempo que se ha tardado a recibir el eco se calcula la distancia.

Algunas características que ofrece se detallan a continuación:

- Tensión de alimentación: 5V
- Rango de Operación: [0.02 ~ 4] m

4.2 BeagleBone Black

Como se ha indicado en el punto anterior se descarta el uso de CM530 disponible en pack de Bioloid Premium y se dispone al uso de la placa BeagleBone Black, tratándose de una placa programable de bajo coste que encuentra su punto fuerte en su capacidad para convertirse en un ordenador de bajo coste y consumo, además de soportar un sistema operativo en base Linux. Asimismo la placa dispone de un acelerador hardware de operaciones de punto flotante, herramienta esencial para ejecutar de forma eficiente el núcleo de Linux en un sistema embebido.

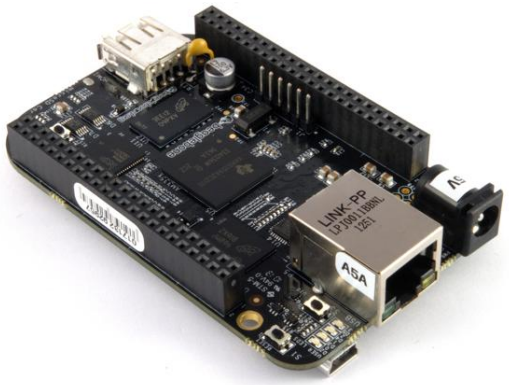


Figura 15: BeagleBone Black RevC

Compatible con los sistemas operativos Debian, Android y Ubuntu entre otros, con un procesador Sitara AM3358BZCZZ100 (ARM-A8) corriendo a 1Ghz y su memoria RAM DDR3 de 512Mb hacen de la BealgeBone Black una placa relativamente potente en relación a su tamaño.

Además, dispone de 92 pines de entrada/salida, compatibles con todo tipo de periféricos como son SPI, UART, I2C, Ethernet, USB y miniUSB,

ofreciendo una gran capacidad de adaptación y un gran potencial. Su diseño también facilita la incorporación de placas accesorias a los conectores P8 y P9, llamadas capes.

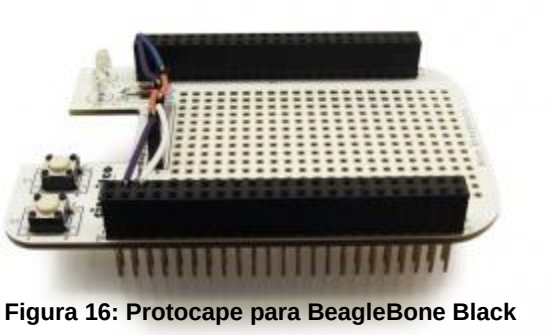


Figura 16: Protocape para BeagleBone Black

Por último, contiene salida HDMI, salida Audio, ranura para microSD, una memoria interna de 4Gb para almacenar el sistema operativo y correrlo sin necesidad de microSD o ampliar su capacidad a gusto del usuario.

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

Figura 17: Características BeagleBone Black

4.3 ROBOcape

Una de las cuestiones que se planteaban en el momento de emprender el trabajo, era la etapa de adaptación de los sensores con la placa BeagleBone Black. Para ello se dispuso desde el principio de la placa ROBOcape diseñada y fabricada en el instituto de automática e informática industrial AI2, la cual resolvía dicha cuestión y facilitando el desarrollo del proyecto.

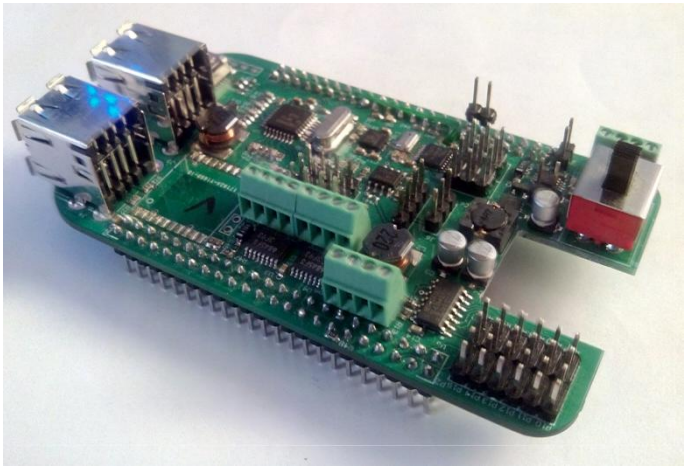


Figura 18: ROBOcape

Se trata por lo tanto de una placa electrónica cuyo diseño se centra en el uso con BeagleBone Balck permitiendo el uso de sensores en sistemas robóticos. Está enfocada a poder utilizar la mayoría de los periféricos de los que puedan necesitar los diseñadores de robots.

En cuanto a sus principales particularidades cuenta con:

- Sistema de alimentación DC/DC independiente, capaz de ofrecer hasta 3 Amperios y alimentar directamente a BBB.
- Cargador de baterías lipo de 2 células.
- 4 conexiones pwm para servomotores mediante buffer de salida.
- 4 puentes en H para control de motores.
- 2 entradas para encoder.
- 6 conectores para sensores de ultrasonido HC-SR04.
- 4 conectores para sensores tipo Sharp de salida analógica.
- 2 conectores para conexión red servos Dynamixel (RS-485 y TTL).
- Sensor IMU de 9 ejes (MPU-9150).

- Altímetro y termómetro.
- GPS integrado (FGPMMOPA6H), con toma para antena externa.
- HUB USB2.0 de 4 puertos.

Respecto a la placa se optó finalmente por la utilización de las entradas analógicas y conectores para sensor HC-SR04, ya que en un primer momento se tanteó la posibilidad de usar el conector del driver MAX-485 para el protocolo TTL, pero ante el surgimiento de diversos problemas durante el desarrollo del proyecto, esta opción quedó descartada. El problema se abordará con más detalle en el (**punto 4.5**) de este documento.

En cuanto a la ROBOcape, cabe destacar su incompatibilidad con versiones anteriores a BeagleBone Black (Ej: BeagleBone), y el hecho de que puede ser alimentada desde la BeagleBone siempre y cuando se conecte el alimentador de 5V ya que con la alimentación del USB resulta insuficiente.

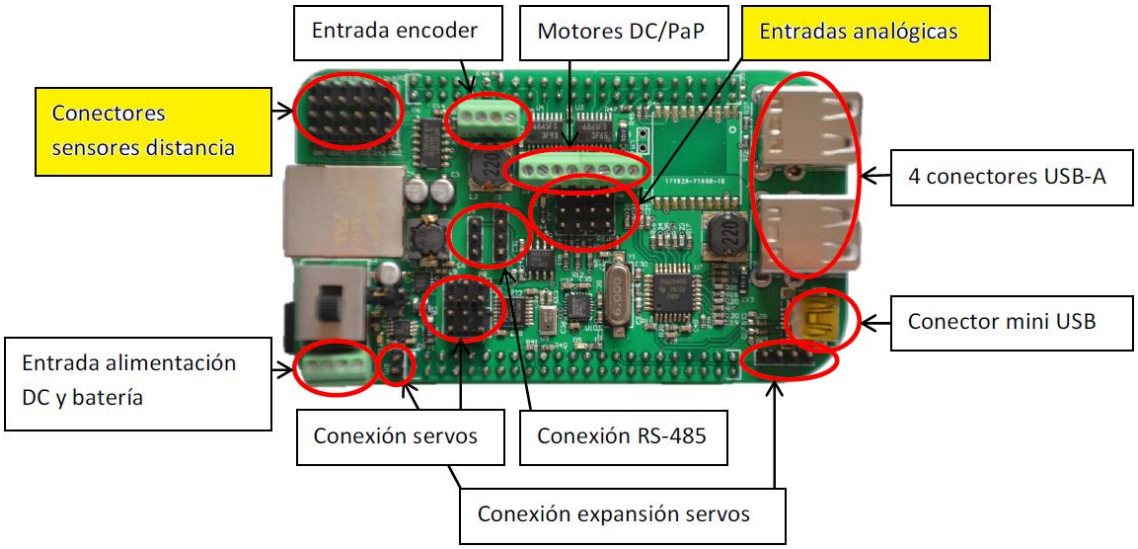


Figura 19: ROBOcape, conectores y pines

4.4 Regulador de tensión

Debido a que el sistema que se pretende realizar, es sobre un robot humanoide que va a estar moviéndose por determinados lugares, está dotado de baterías y todos los componentes no funcionan a la misma tensión de alimentación, se precisa la incorporación de un regulador para adecuar dichas tensiones y favorecer el correcto funcionamiento de todo el sistema. En concreto debido a disponer de un sistema de baterías de 12V y ser necesaria la alimentación de 5V para la BeagleBone Black.

Para ello se da con un módulo de alimentación de 12V a 5V que se compone del regulador de tensión LM2596 y el cual soluciona la cuestión de la alimentación de la BeagleBone Black.

Como se ha indicado, se elige el regulador de tensión LM2596 incorporado en un Módulo junto con otros componentes para estabilizar tensiones. El regulador mediante un potenciómetro permite alimentación de 35V ~ 1.25V de salida, con una tensión de entrada de 40V ~ 3.2V y proporcionando una corriente de hasta 3 Amperios.

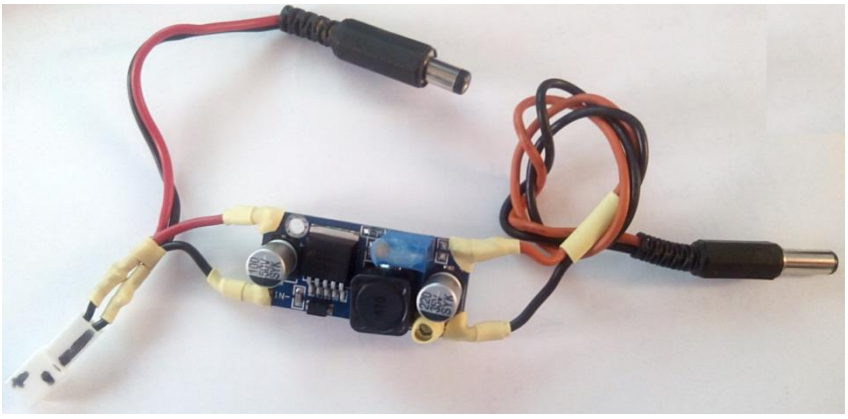


Figura 20: Módulo regulador de tensión

Así se dispone de una alimentación de 12V para los servomotores Dynamixel y otra de 5V para la alimentación de la BeagleBone Black junto con la placa ROBOcape.

4.5 Problema y solución para protocolo TTL

Durante el desarrollo del protocolo TTL para servomotores Dynamixel y BeagleBone Black, apareció cierto imprevisto entre la comunicación de ambas partes, debido al driver MAX-485 de la ROBOcape y los buffers de entrada/salida de la UART de la BeagleBone Black.

Para realizar la comunicación entre placas y Dynamixel, es necesario de un driver adaptador debido a que utilizan diferentes tipos de comunicación. La BeagleBone Black utiliza UART's con protocolo RS-232(dúplex) y los servomotores utilizan el tipo TTL (half-duplex). En este punto aparece el driver MAX-485 de la ROBOcape, cuya función permite la adaptación de los dos tipos de comunicación, haciendo cambiar un pin de control. Véase la siguiente (**Figura21**):

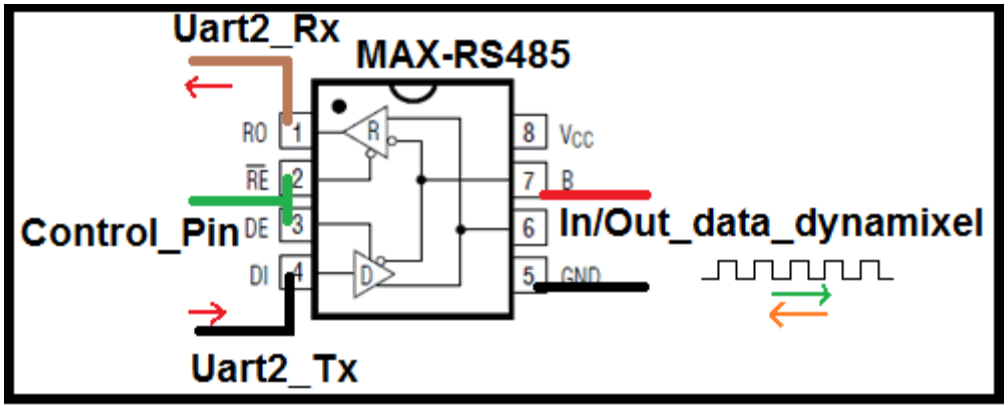


Figura 21: Diagrama MAX-485

El error en la transmisión, residía en la basculación del bit de control por parte de la BeagleBone Black, si el bit de control conmutaba a una velocidad mayor que el tiempo que se empleaba para el envío/recepción, el paquete no llegaba o se recibía correctamente. Por otro lado si la velocidad de oscilación era demasiado pequeña tampoco se enviaba/recibía el paquete entero, obteníendose entonces un resultado poco favorable.

La solución llegó de la búsqueda de drivers que hiciesen la adaptación de comunicación (dúplex-halfduplex) de forma física (hardware) sin ningún tipo de software y así evitar el problema.

En la búsqueda resultaron dos opciones interesantes que descritas a continuación, optando finalmente por aquella que mejor se adaptaba.

4.5.1 Driver USB2AX 3.2v

Figura 22: Driver USB2AX 3.2v

Este pequeño driver permite la comunicación de los Dynamixel ofreciendo un pequeño tamaño (16x36mm), una baja latencia y una velocidad de 1Mbps. Ofrece una fácil conexión tanto para la placa BeagleBone Black como a los servos Dynamixel, sin embargo presenta el inconveniente de la alimentación de los servomotores, ya que se debe realizar externamente.

4.5.2 Driver CDS55xx

Figura 23: Driver CDS55xx

Por otro lado también existe el driver CDS55xx que permite conectar el protocolo RS232 Duplex con Half-Duplex, con las mismas prestaciones anteriormente descritas. A diferencia del anterior, este driver posibilita que la alimentación sea a través del driver, facilitando de este modo el montaje. El tamaño de este es un poco más grande pero sigue siendo una buena opción para lo pretendido en el proyecto.

4.5.3 Solución adoptada para protocolo TTL

De las dos opciones mencionadas anteriormente, se decide probar las dos y quedarse con la que mejor funcione. En este caso el driver CDS55xx fue el que mejor resultado obtuvo en lo referente a los problemas de configuración y conexiones, decidiéndose por lo tanto continuar con éste el proyecto.

5. DESARROLLO DELPROYECTO

5.1 Preparación del entorno de desarrollo

Antes de realizar la explicación del software desarrollado, existe un paso previo que consiste en preparar el entorno de trabajo donde se desarrolla el API para robot Bioloid y BeagleBone Black y cuyos pasos son los siguientes:

5.1.1 Instalación Virtual Box

El primer paso llevado a cabo para la preparar el entorno de programación, es la instalación de una maquina virtual, la cual nos permita tener un sistema operativo (en este caso Linux) dentro de nuestro sistema operativo normal de trabajo (Windows10 en mi caso). Esta máquina virtual se realiza con el programa de la compañía ORACLE llamado “VirtualBox”. Dentro de este programa se instala el entrono de programación y así evitar posibles desperfectos en el sistema operativo de trabajo normal. Aparte de no dañar los cambios de la máquina virtual en el sistema no virtualizado, se tiene la posibilidad de realizar una réplica del sistema virtualizado de una manera rápida y sencilla.

Para realizar la instalación se deben seguir los siguientes pasos:

- Descargar “VirtualBox” de la página:

  - https://www.virtualbox.org/wiki/Downloads

- Realizar la instalación del mismo siguiendo los pasos que se nos indica en el proceso de instalación y quedará listo el virtualizador.

Figura 24: Instalación VirtualBox

5.1.2 Instalación máquina virtual Linux

El siguiente paso trata de instalar un sistema Linux, permitiendo así la instalación posterior del entorno IDE de Eclipse. Esta instalación es muy sencilla y fácil de realizar siguiendo los siguientes pasos, extraídos de [Sem]:

- Descargar una imagen de Linux (se recomienda la versión “Long Time Suport”), en nuestro caso se utiliza la imagen “Kubuntu 14.04” (kubuntu-14.04.2-desktop-amd64.iso).

  Link de descarga: http://www.kubuntu.org/getkubuntu/

- Instalar la imagen de Linux sobre el “VirtualBox”, asignándole recursos de memoria principal, procesadores y espacio de forma razonable ya que lo que se pretende llevar a cabo no utiliza demasiados recursos.

Una vez se tiene instalado el sistema operativo en la máquina virtual, se da paso a la instalación de los drivers y configuración necesaria para el correcto funcionamiento, para ello se siguen los siguientes pasos:

- Arrancar la máquina virtual, comprobar que se puede conectar a internet el sistema Linux y actualizar el sistema.
- Instalar Drivers y extensiones de “VirtualBox” en el sistema operativo.
- Defina algún directorio compartido si desea transferir archivos entre el computador de trabajo y la máquina virtual.
- Activar la interacción a través del “clipboard” para poder cortar/pegar elementos entres máquina virtual y computador de trabajo.

Para más detalles sobre la instalación ver y seguir los pasos del capítulo 0 del documento [Sem] que se incluye en los Anexos.

Figura 25: Escritorio Kubuntu14.04 en VirtualBox

5.1.3 Instalación Eclipse en máquina Virtual

Una vez realizados los pasos anteriores, solo queda preparar el entorno IDE de Eclipse para poder llevar a cabo el desarrollo del código de la API. Para ello se necesita la instalación de los entornos de desarrollo nativos y Eclipse. Esto es posible siguiendo los pasos detallados en [Sem] y que se detallan brevemente a continuación:

- Instalación de Java (en este caso se instala Java8), es necesario instalar el JRE para poder utilizar Eclipse.
- Instalación de C y C++ para poder desarrollar aplicaciones en el sistema Linux.
- Descarga e Instalación de Eclipse-CDT.
- Instalación del entorno de desarrollo cruzado.

El siguiente y último paso se necesita para realizar la configuración de Eclipse para el desarrollo cruzado, mediante los puntos que se detallan brevemente a continuación:

- Intalación de los componentes de eclipse necesarios: “Help->Install New Software -> Remote System Explorer”
- Configuración de un proyecto Eclipse de desarrollo cruzado: “File -> New -> Project”.
- Configuración de “Remote System Explorer”.

Todos los pasos anteriores mencionados, se encuentran con más detalles en capítulo 0 del documento [Sem] que se incluye en los Anexos.

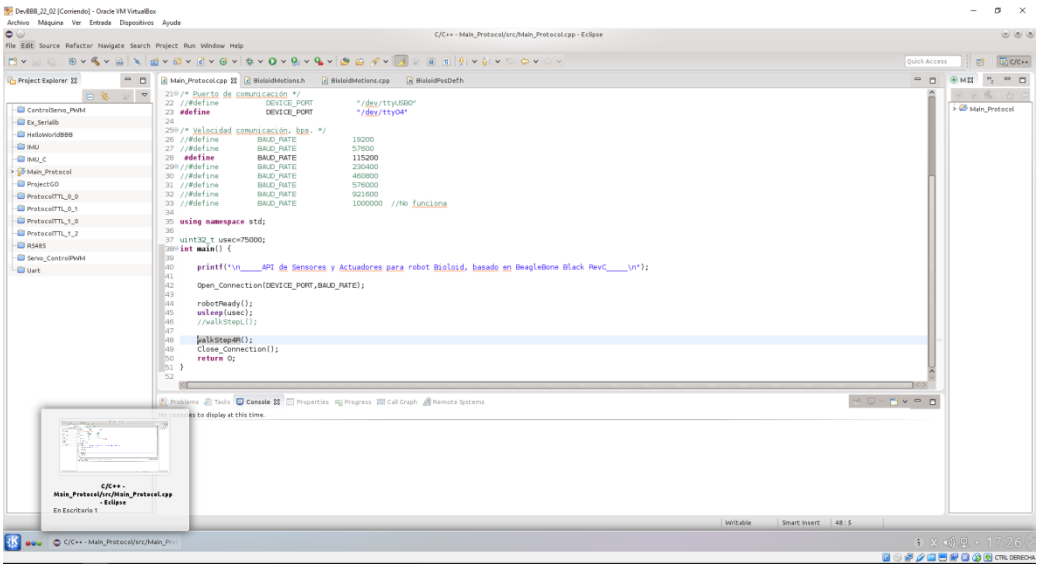


Figura 26: Entorno Eclipse en VBox y proyecto configurado

5.2 Desarrollo capas de software

En el siguiente punto, se procede a la explicación del desarrollo tramitado de la API para sensores y actuadores en robot Bioloid. Profundizando un poco, la API consta de 3 capas con un total de 5 módulos en C/C++ (4 para protocolo TTL y 1 para sensores) los cuales están estructurados de bajo a alto nivel, facilitando su utilización para el usuario que la vaya a utilizar.

5.2.1 Capa de abstracción de comunicaciones

La primera capa de la API, se centra en la capa de abstracción para la comunicación entre Dynamixel y BeagleBone Black. En ella se resuelve la cuestión de apertura/cierre del puerto y el envió/recepción de datos a través del puerto especificado.

5.2.1.1 BlackDynamixel.cpp/.h

BlackDynamixel.cpp/.h es el primer módulo desarrollado en C/C++ del protocolo y consta de 4 funciones básicas para manipular el flujo de datos de entrada y salida del puerto, aparte de abrir o cerrar el puerto que se utilize (UART_4 en este caso).

```
bool Open_Connection(const char *Device,const uint32_t Bauds);
```

Permite abrir el puerto especificado en (const char *Device) con la velocidad en baudios especificada en (uint32_t Bauds).

```
void Close_Connection(void);
```

Realiza el cierre de la conexión que estaba abierta.

```
void Send_Data(uint8_t *packetOut,uint16_t length);
```

Envía por el puerto, el número de los datos especificado en (uint16_t length) y que se encuentran contenidos en (uint8_t *packetOut).

```
uint8_t Received_Data(uint8_t *destination, uint16_t max_data);
```

Lee del puerto, el número de bytes indicado en (uint16_t max_data) y los guarda en (uint8_t *destination).

De todas estas funciones se encuentra un ejemplo de cómo utilizarlas en el archivo *BlackDynamixel.h*, situado en el apartado Anexos.

5.2.2 Capa abstracción movimiento

La segunda capa que compone el API, se llama capa de abstracción de movimientos. Esta capa depende de la anterior para comunicarse con los Dynamixel y en ella se desarrolla por una parte el módulo *InstrucDynamixel.cpp/.h* con las funciones básicas que se pueden realizar a los Dynamixel (PING, WRITE, SYN_WRITE, etc…).

Por otra parte también se desarrolla el módulo *UtilDynamixel.cpp/.h* con un escalón más de abstracción pero que depende del anterior y se puede incluir en esta capa. En este módulo se llevan a cabo las funciones necesarias para realizarse la configuración de los parámetros de la tabla de control disponible en los Dinamixel. Estos dos módulos mencionados llevan desarrollado el protocolo que utilizan los servomotores, cuyo funcionamiento se detalla en el documento [Dyn].

Finalizando la introducción de esta capa, cabe destacar el último módulo desarrollado llamado *BioloidMotions.cpp/.h* y en el que se incluyen una serie de funciones asociadas a los movimientos del robot, llegando así a la capa más externa del protocolo. Dado por introducidos los módulos que componen esta capa se procede a la explicación detallada de los mismos.

5.2.2.1 InstrucDynamixel.cpp/.h

Como se ha dicho antes, *InstrucDynamixel.cpp/.h* es el primer módulo en C/C++ y consta de 9 funciones que permiten realizar las funciones básicas realizables sobre los Dynamixel. En él se soluciona la primera cuestión del protocolo que utilizan los Dynamixel, obtenido del [Dyn] y que se procede la explicación a continuación.

El protocolo TTL para Dynamixel AX-12ª, funciona con dos tipos de paquetes llamados paquetes de instrucción (los que se envía de la unidad de control a los Dynamixels) y paquetes de estado (los que se envían de los Dynamixel a la unidad de control).

El primer tipo de paquete (Instrución) tiene la siguiente estructura y sirve para realizar acciones sobre los servos:

0xFF 0xFF 0xID 0xLENGTH 0xINSTRUCTION 0xPARAMETER1 0xPARAMETER N 0xCHECKSUM

El paquete empieza con la cabecera de dos bytes “0xFF 0xFF” avisando a todos los Dynamixel que va a llegar un paquete de datos.

Seguidamente está el byte “0xID” que contiene el identificador del servo al que se envía el paquete. Este byte puede ser individual (de 0x00 a 0xFD) o para todos los servos que haya conectados al bus de datos (0xFE) “BroadCasting”.

Luego, se encuentra el byte “0xLENGTH” que contiene el número de bytes que le sigue.

Por consiguiente, se tiene el byte “0xINSTRUCTION”, que engloba la instrucción que se quiere realizar en el/los dynamixel/s. Esta instrucción puede ser:

Instrucción	Valor	Función
PING	**0x01**	Realiza un Ping para comprobar si existe el dinamixel.
READ_DATA	**0x02**	Leer valores de la tabla de control.
WRITE_DATA	**0x03**	Escribir valores en la tabla de control
REG_WRITE	**0x04**	Escribir valores en la tabla de control, pero permanece en espera hasta que se ejecute la instrucción de ACTION
ACTION	**0x05**	Produce la ejecución establecida por REG_WRITE
RESET	**0x06**	Restablece los valores del Dynamixel a los parámetros de fábrica
SYNC_WRITE	**0x83**	Realiza la escritura en la tabla de control de todos los dynamixels conectados al bus, pero no espera que se ejecute otra instrucción.

Tabla 1: Tipo de instrucciones

Los bytes que le sigue a la instrucción, son “0xPARAMETER 1…N”. En estos bytes se deposita la información complementaria a ejecutar por la instrucción del byte anterior.

Finalmente, queda nombrar el byte “0xCHECKSUM”, el cual tiene la función de saber si el paquete se ha enviado correctamente. Para realizar el

cálculo del checksum, basta con hacer la inversa de los ocho últimos dígitos obtenidos al sumar los bytes:

0xID+0xLENGTH+0xPARAMETER1+…PARAMETER

El segundo tipo de paquete (Estado) tiene la siguiente estructura y sirve para realizar la respuesta de los servomotores Dynamixel:

0xFF 0xFF 0xLENGTH 0xERROR 0xPARAMETER1 0xPARAMETER N 0xCHECKSUM

Al igual que en el paquete de Instrucción, este también empieza con la cabecera de dos bytes “0xFF 0xFF” avisando a todos los Dynamixel que va a llegar un paquete de datos.

Luego, se encuentra el byte “0xLENGTH” que contiene el número de bytes que le sigue.

Por consiguiente, se tiene el byte “0xERROR”, que engloba el error (si se ha producido) que envía el Dynamixel. Este puede ser de diferentes valores detallados a continuación:

Bit	Nombre	Detalles
7	*No hay respuesta*	Vale "1" si no hay respuesta del Dynamixel.
6	*Instruction Error*	Vale "1" si se ha enviado una instrucción incorrecta o no definida.
5	*Overload Error*	Vale "1" si se ha superado la torsión máxima del motor.
4	*Checksum Error*	Vale "1" si el valor del checksum del paquete es incorrecto.
3	*Range Error*	Vale "1" si se ha enviado un rango más grande del que puede llegar.
2	*Overheating Error*	Vale "1" si se ha sobrepasado el valor de temperatura interior del motor.

1	*Angle Limit Error*	Vale "1" si se ha enviado una posición fuera de los limites CW y CCW.
0	*Input Voltage Error*	Vale "1" si el voltaje aplicado está fuera del rango definido en la tabla de control.

Tabla 2: Tipo de errores de respuesta

Una vez hecho un boceto sobre el funcionamiento del protocolo TTL que emplean los Dynamixel, se procede con la explicación de las funciones realizadas en la librería *InstrucDynamixel.cpp/.h*.

```
void Get_Packet(uint8_t *param,uint8_t packetOut[DATA_SIZE],uint16_t
*length);
```

La función “Get_Packet”, es la encargada de añadir al paquete que se pasa por (uint8_t *param), la cabecera “0xff 0xff” y el Checksum, guardando posteriormente el paquete en (uint8_t packetOut[DATA_SIZE]). La longitud (uint16_t *length), primeramente es la longitud de (uint8_t *param) pero luego se actualiza a la longitud de (uint8_t packetOut[DATA_SIZE]), por eso se le pasa por referencia.

```
uint8_t Get_Checksum(uint8_t *param,uint16_t length);
```

La siguiente función, “Get_Checksum”, es la que se ocupa de calcular el checksum de la trama de datos que se va a enviar. En (uint8_t *param) se le pasa el vector con los parámetros para el cálculo y en (uint16_t length) la longitud del mismo.

```
uint16_t Paralen(uint8_t *param);
```

En la función sucesiva, “Paralen”, se pretende contar el número de elementos que hay en un vector hasta localizar el carácter “}”.

```
uint8_t Ping_Servo(uint8_t id);
```

“Ping_Servo”, es la función que realiza un PING al Dynamixel especificado en (uint8_t id).

```
uint8_t Reset_Servo(uint8_t id);
```

La función “Reset_Servo” es la encargada de realizar un reset al Dynamixel especificado en (uint8_t id) y establecer los parámetros de fábrica.

```
uint8_t Write_Servo(uint8_t id,uint16_t length,uint8_t
instruction,uint8_t *param);
```

La siguiente función, “Write_Servo”, permite la escritura de parámetros en la tabla de control del servomotor. En esta función se puede hacer una escritura con sincronización (Instrucción REG_WRITE) o escritura normal (WRITE_DATA). Además se puede escribir los parámetros a un servomotor en concreto o realizar un “Broadcasting” para que cambien todos los Dynamixels conectados al bus TTL.

```
void Action_Servo(void);
```

Seguidamente, se encuentra la función “Action_Servo” cuya tarea es de dar la orden de que se ejecuten los parámetros enviados en el paquete de datos con la instrucción REG_WRITE.

```
void SyncWrite_Servo(uint8_t *id, uint16_t length, uint8_t *param);
```

Concluyendo la librería llegamos a la función “SyncWrite_Servo”, encargada de hacer una escritura a todos los servos forma sincronizada, pero sin necesidad de ejecutar ninguna instrucción después.

```
uint8_t Read_Servo(uint8_t id,uint8_t Address,uint16_t lng_reg,uint8_t
*value);
```

Finalmente llegamos a la última función llamada “Read_Servo” donde su labor trata de leer en la tabla de control del servomotor y devolver el valor correspondiente.

De todas estas funciones se encuentra un ejemplo de cómo utilizarlas en el archivo *InstrucDynamixel.h*, situado en el apartado Anexos.

5.2.2.2 UtilDynamixel.cpp/.h

Se sigue de *UtilDynamixel.cpp/.h*, un módulo que consta de 40 funciones y donde se pretende ofrecer mayor facilidad en la configuración de los servos Dynamixel al usuario que vaya a utilizarlas. Al ser un elevado número de funciones, a continuación se muestra tres tablas (Escritura en Dynamixel, Lectura en Dynamixels y Lectura en Pies) con el nombre y tarea que realiza cada función. Para más información de cómo se utilizan véase los ejemplos en *UtilDynamixel.h* del apartado Anexos.

Nombre función	Tarea
set_Id	Establece el identificador del Dynamixel
set_BaudRate	Establece la velocidad de transmisión del Dynamixel
set_ReturnDelay	Establece el tiempo de retorno del Dynamixel

set_Mode	Establece el Modo Rotación continua o Servomotor
set_HLimitTemp	Establece el límite de temperatura del Dynamixel
set_LimitVoltage	Establece el límite de tensión del Dynamixel
set_MaxTorque	Establece el Torque máximo que proporcionará el Dynamixel
set_StatusPacket	Establece cuando se quiere recibir respuesta del Dynamixel
set_AlarmLED	Establece cuando se activa la alarma y se enciende el Led
set_AlarmShutdown	Establece en que errores se auto apaga el Dynamixel
set_HoldinTorque	Activa/Desactiva el torque del Dynamixel
set_LED	Activa/Desactiva el Led del Dynamixel
set_Compliance_M_S	Establece los parámetros Slope y Margim del Dynamixel
set_Punch	Establece la corriente que consume el Dynamixel
set_Lock	Activa modo escritura reducido solo se podrá escribir en las direcciones (0x18 a 0x23). Reiniciar alimentación para desbloquear
set_TorqueLimit	Establece el Torque máximo que proporcionará el Dynamixel, pero este es volátil(se pierde la configuración al quitar la alimentación)
set_Pos	Establece la posición del Dynamixel
set_Pos_Vel	Establece la posición y velocidad del Dynamixel
set_PosVelPreload	Establece la posición y velocidad del Dynamixel con REG_WRITE, después ejecutar ACTION
set_PosSync	Establece la posición de los Dynamixel de manera sincronizada con la instrucción SYNC_WRITE
set_VelSync	Establece la velocidad de los Dynamixel de manera sincronizad con la instrucción SYNC_WRITE
set_MovSpeed	Establece la velocidad del Dynamixel
mode_Wheel	Configura velocidad y sentido de giro del Dynamixel cuando está en modo Rotación continua
mode_WheelPreload	Configura velocidad y sentido de giro del Dynamixel cuando está en modo Rotación continua con la instrucción REG_WRITE, luego espera la instrucción ACTION
set_QuestDynamixel	Comprueba cuantos servomotores Dynamixel hay conectados al bus

Tabla 3: Funciones de escritura en Dynamixel

Nombre función	Tarea
readPos	Lee y devuelve la posición del Dynamixel
readSpeed	Lee y devuelve la velocidad presente del Dynamixel
readMovSpeed	Lee y devuelve la velocidad configurada del Dynamixel
readTemperature	Lee y devuelve la temperatura configurada del Dynamixel
readVoltage	Lee y devuelve el Voltaje configurado del Dynamixel
readAngleLimit	Lee y devuelve el límite CW o CCW configurado del Dynamixel
readLoad	Lee y devuelve el toque configurado del Dynamixel
checkRegister	Lee si el Dynamixel espera la instrucción ACTION
checkMovement	Lee y devuelve si el Dynamixel se encuentra en movimiento
checkLock	Lee y devuelve si está activo el modo escritura reducido

ledState	Lee y devuelve el estado del Led

Tabla 4: Funciones de lectura en Dynamixel

Nombre función	Tarea
Read_Sharp_Foot	Lee y devuelve la distancia del pie indicado
Read_Pressure	Lee y devuelve la presión ejercida por los sensores de la planta del pie
Read_CM	Lee y devuelve el centro de masas del eje indicado
Read_CM_Axes	Lee y devuelve el centro de masas del eje X e Y

Tabla 5: Funciones de lectura en Pies Bioloid

5.2.2. BioloidMotions.cpp/.h

Por último y para terminar la segunda capa, queda explicar las funciones del módulo, *BioloidMotions.cpp/.h*. En este módulo se sube el último escalón y ofrece al usuario funciones básicas de movimiento (caminar/manotazos) del robot, consiguiendo así una manera fácil de programar un robot Bioloid con BeagleBone Black. La filosofía que se sigue en este módulo, se asemeja con la del software de RoboPlus. Se extraen una serie de matrices con las posiciones de los movimientos de los servomotores, luego son introducirlas en funciones de movimiento desarrolladas en el módulo y mediante una serie de pasos se generan las trayectorias del Bioloid. Dichas funciones se describen a continuación:

Nombre función	Tarea
roboReady	Establece la posición para ejecutar movimientos
walkNSteps	Realiza N pasos
walkStepL	Realiza 1 paso empezando con el pié izquierdo
walkStepR	Realiza 1 paso empezando con el pié derecho
walkStep4L	Realiza 4 paso empezando con el pié izquierdo
walkStep4R	Realiza 4 paso empezando con el pié derecho
Manotazos	Realiza el movimiento de brazos para defenderse

Tabla 6: Funciones de movimiento en Bioloid

Para realizar un correcto movimiento sin que se caiga el robot, previamente a todos los movimientos, se recomienda realizar la llamada a roboReady

5.2.3 Capa Abstracción sensores y actuadores

La tercera capa de la API, trata de adaptar los sensores DMS, HC-SR04 y GyroSensor, cuya finalidad radica en la obtención de la distancia de los objetos del entorno en los dos primeros y las velocidades angulares del eje X e Y en el

último. La finalidad de la presente capa, tiene el propósito de dotar al Bioloid con sensores y así poder tener un mínimo conocimiento del entorno que le rodea. El desarrollo realizado para los sensores de la presente capa, se encuentran en el módulo *BlackSensors.cpp/.h.*

5.2.3.1 Sensor Sharp DMS

Como primer punto, se tiene la adaptación del sensor de distancia Sharp 2y0A21. Como se ha explicado anteriormente en el punto de 4.1 (componentes del robo Bioloid), se trata de un sensor con un rango de distancia de 10 a 80cm. El inconveniente que muestra a primera vista, si se observa la (**figura 6**) del documento, es la curva que describe. Al no ser un sensor lineal, previamente se debe realizar una linealización para así obtener los valores de distancia correctos.

A partir de la hoja de cálculo de [JCB], utilizada como guía, se obtiene la siguiente fórmula linealizadora:

$$Distancia(Cm) = \frac{37790}{(EntradaAnalógica - 3)} - 4$$

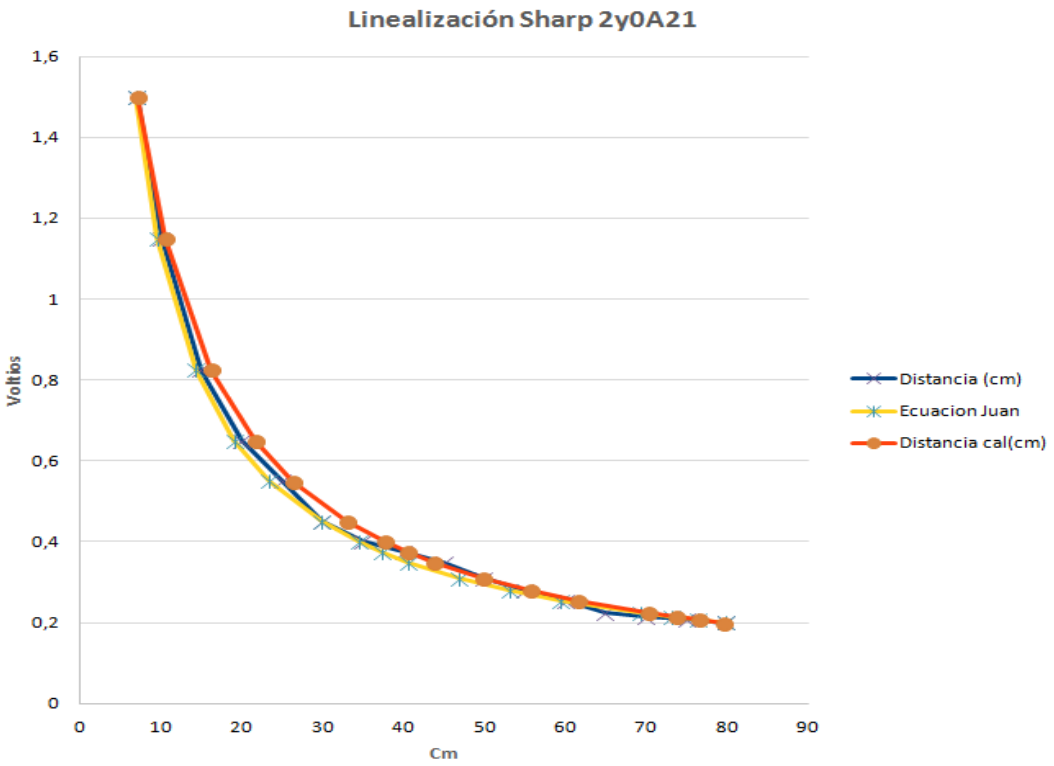


Figura 27: Curva Linealizada del sensor Sharp 2y0A21

Es con la función “readSharp” la cual se encarga de leer la entrada analógica indicada y realizar los cálculos necesarios, con la que se puede obtener la distancia de los objetos presentes delante del sensor. Con solo indicar el número de la entrada analógica donde se encuentra conectado (0, 1, 2, 3) el Sharp, esta devolverá la distancia del mismo.

```
uint8_t readSharp(uint8_t number);
```

Cabe destacar del sensor la salida analógica de la que dispone y por la cual solo se podrá conectar a los pines de entrada analógica de la ROBOcape. Asimismo no será posible la conexión de más de 4 sensores tipo Sharp, ya que la ROBOcape solo dispone de 4 entradas analógicas.

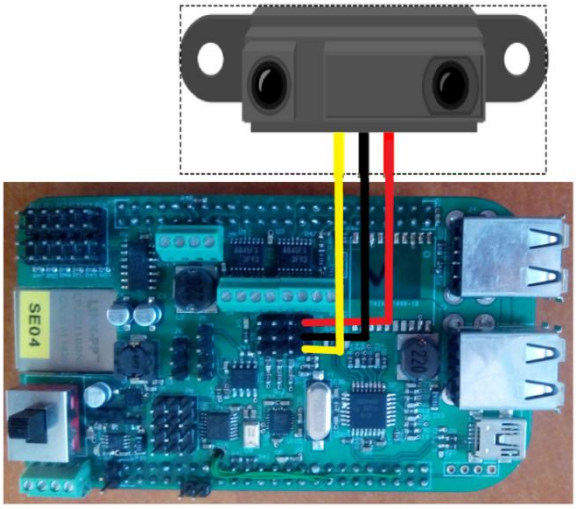


Figura 28: Conexión DMS a AI0 de la ROBOcape

5.2.3.2 GyroSensor GS-12

Como segundo elemento de la tercera capa, se dispone el Giroscopo que viene en el Kit Bioloid de Robotis. Este sensor como se ha explicado en el punto 4.1, sirve para saber en qué posición se encuentra el Robot Bioloid ya sea de pie o en el suelo. El sensor mide la velocidad angular del eje X e Y del robot y los devuelve en tres estados o casos:

Velocidad Angular	+300 °/s	←	0 °/s	→	-300 °/s
Voltaje de Salida	2.23 V		1.23 V		0.23 V

Figura 29: Sensor Giroscopo de Robotis

Como la tensión de salida del Giroscopio es analógica, al igual que el Sharp del punto anterior, este se conectará a la entrada analógica (0, 1, 2, 3)

de la Robocape. En la función “readGyro” que se muestra a continuación se debe indicar el pin donde se conecta el eje el cual se quiere saber la velocidad angular.

```
int16_t readGyro(uint8_t number);
```

Como en el caso del Sharp, la ROBOcape, dispone de 4 entradas analógica, limitando la conexión de 4 ejes o 2 GyroSensors.

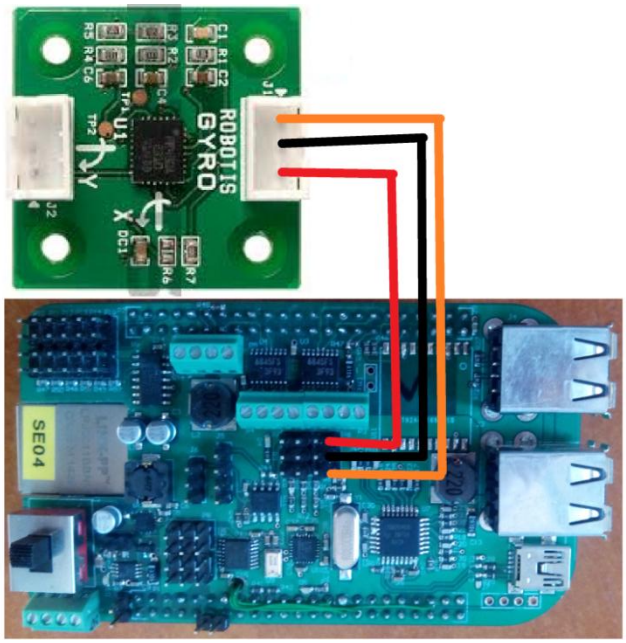


Figura 30: Conexión EjeX GyroSensor a AI0 de la ROBOcape

5.2.3.3 Sensor HC-SR04

Por último y concluyendo la explicación de la tercera capa de la API, queda nombrar el sensor de ultrasonidos HC-SR04. Sensor de distancia que como se ha nombrado en el punto 4.1, tiene un rango de [0.02 ~ 4] m.

Para este sensor se ha utilizado los conectores disponibles en la placa ROBOcape para sensores HC_SR04. En concreto se utiliza los pines 8.8 y 8.9 que se muestra en la siguiente imagen:

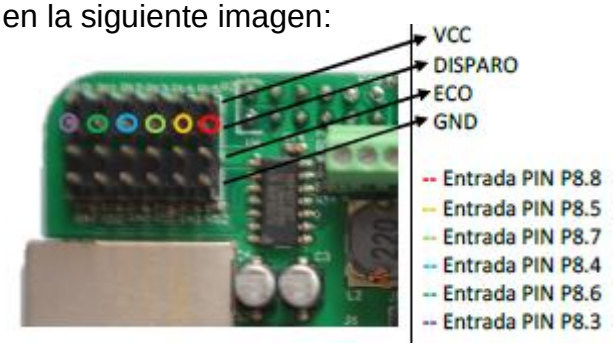


Figura 31: Conectores para sensor HC-SR04 en ROBOcape

```
int16_t readHC_SR04(void);
```

Para esta función se ha tenido en cuenta el funcionamiento del sensor que se detalla en [Rcape] y el cual se va a detallar brevemente.

Primeramente, se envía un pulso de al menos 10μs por el Pin Trigger (Pin8.8). Después, el sensor envía 8 pulsos de 40kHz y coloca su salida Echo (Pin8.9) en alto ("1"lógico). A continuación se detecta este evento e se inicia un conteo de tiempo hasta que la salida Echo (Pin9.9) vuelve a estar a bajo("0" lógico). Una vez el pin Echo esté abajo, se deja pasar 50ms y se calcula la distancia mediante la siguiente fórmula:

$$Distancia(cm) = Tiempo\ medido(\mu s)\ x\ 0.017$$

La función "readHC_SR04", SOLO está diseñada para realizar la lectura de un solo sensor de ultrasonidos en el del pin8.8 y 8.9, para poder conectar más sensores de este tipo, se tiene que reprogramar el módulo *BlackSensors.cpp/.h*. Para más detalles véase [Rcape]

5.2.4 Validación API desarrollada

Tras la explicación sobre el desarrollo de la API que se ha realizado para robot Bioloid, basada en BeagleBone Black, se da paso a la exposición de los tests elaborados y sus respectivos resultados.

5.2.4.1 Test 1: IMU-Dynamixel

El primer test que se realiza, trata de abordar la primera parte de la API (protocolo TTL) y la cual trata de escribir posiciones en dos servomotores Dynamixel en función de una posición obtenida de un sensor inercial llamado IMU, que se encuentra instalado en la placa ROBOcape [Rcape].

Explicando brevemente el funcionamiento del sensor, la IMU entre otros devuelve aceleraciones de los ejes Y, X y Z. Por tanto, mediante un ejemplo obtenido de [Sem] en el que se leen dichas aceleraciones, se adapta a la API para poder leer las aceleraciones de los ejes X e Y.

Una vez inicializada y configurada la IMU para realizar las lecturas necesarias, se da paso a escalar los valores de aceleración que vienen comprendidos entre [-1 a 1] a los valores que se usan en los Dynamixels [0 a 1023], cuya equivalencia es de [0º a 300º].

Cabe destacar de este ensayo, el uso del el driver MAX-485 para adaptar la comunicación dúplex a simplex ya que solo se enviaban posiciones a los Dynamixel sin esperar respuesta.

Finalmente se muestra un diagrama con las conexiones y un par de fotos del montaje realizado para este ensayo, el código de la aplicación realizada se adjunta en el apartado Anexos.

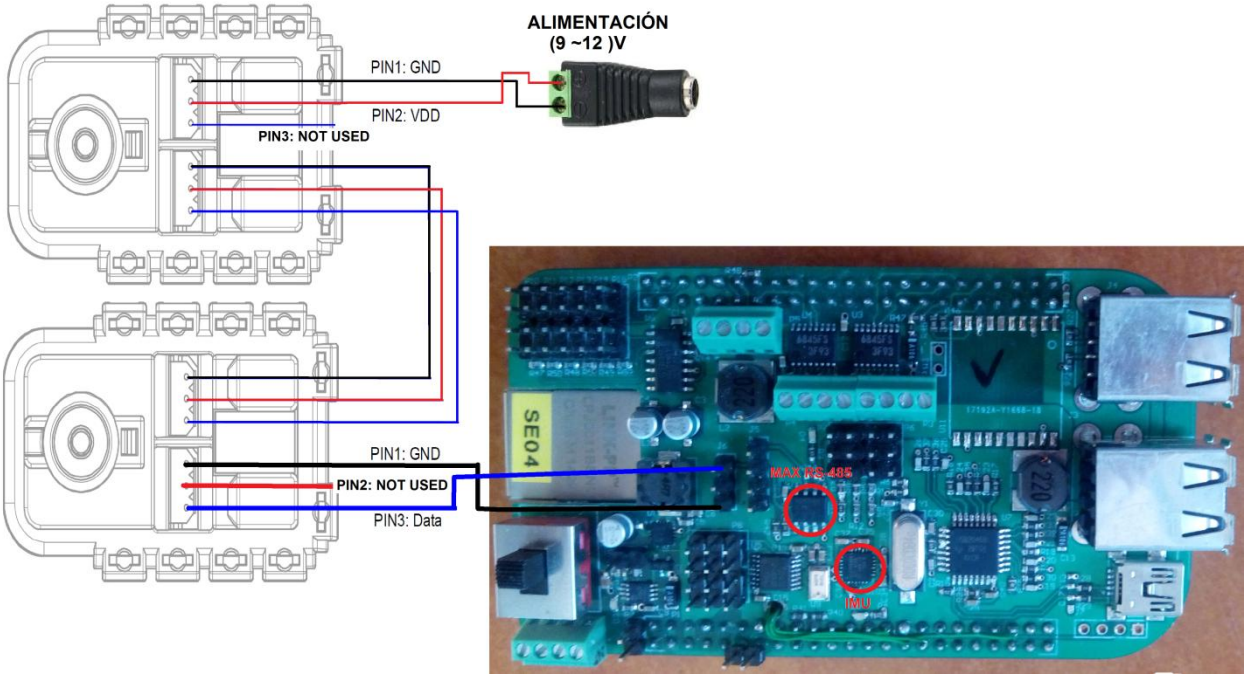


Figura 32: Diagrama Conexiones test 1 (IMU - Dynamixels)

Figura 33: Montaje realizado en test 1 (IMU-Dynamixel)

5.2.4.2 Test 2: Usuario-Dynamixel

El siguiente test ocupa parte de la capa de abstracción de movimientos del protocolo TTL para Dynamixels y BeagleBone, más concretamente los módulos *InstrucDynamixel.cpp/.h* y *UtilDynamixel.cpp/.h.* Se realiza un ejemplo cuyo funcionamiento persigue el uso del mayor número posible de funciones, incluyendo tanto las funciones de configuración de parámetros para los servomotores como las funciones de que indiquen al Dynamixel donde tiene que ir. En esta parte surge la necesidad de leer y escribir en los Dynamixel por medio del protocolo, siendo en un principio a través del driver MAX-485 de la ROBOcape. No obstante debido al imprevisto explicado en el apartado 4.5, se realiza el test con el uso del driver CDS55xx

Con el problema solucionado, se procede a la programación y comprobación con el siguiente ejemplo:

Se dispone de dos brazos derechos (brazo 1 y 2) del robot bioloid (sin las manos) conectados al driver CDS55xx. Cuando se mueve el brazo derecho 2, se leen las posiciones de los motores que lo componen y se escriben en el brazo derecho 1, así el movimiento que describe el brazo 2 también lo realiza el brazo 1. En la siguiente figura se expone el montaje realizado para este ensayo:

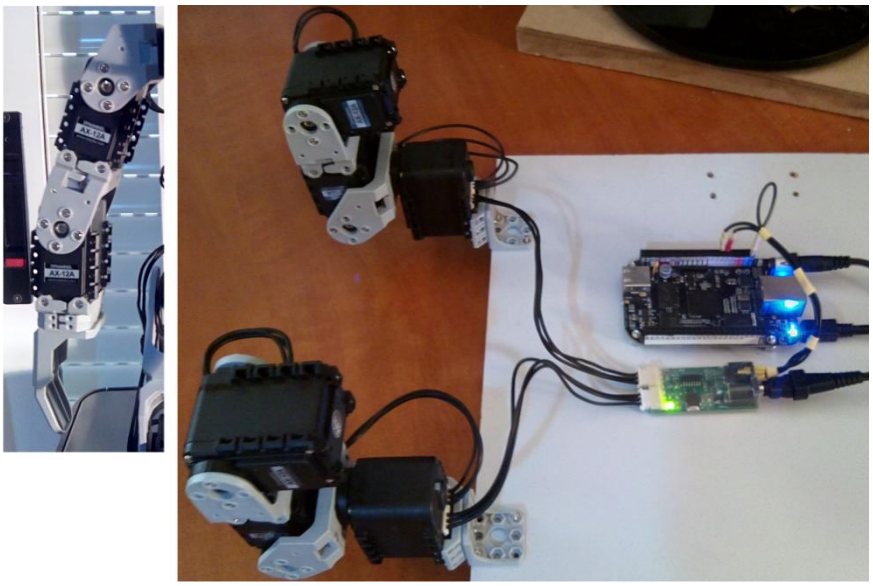


Figura 34: Montaje realizado en test 2 (Usuario-Dynamixel)

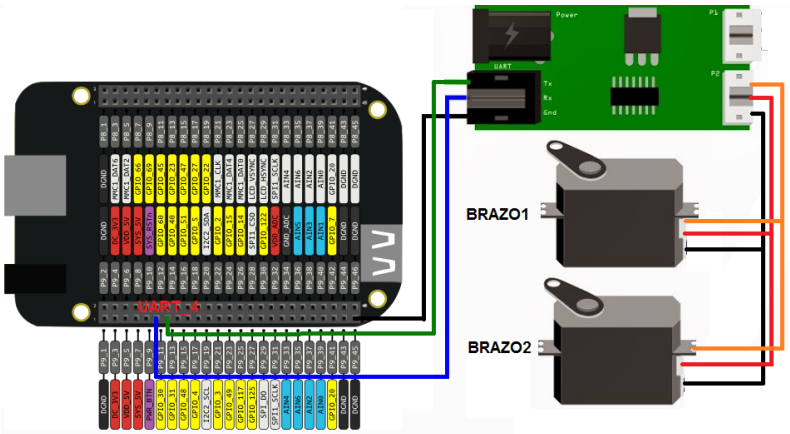


Figura 35: Diagrama de conexiones test 2 (Usuario-Dynamixel)

5.2.4.3 Test 3: Sensor Sharp

Dejando de un lado la capa de abstracción de movimiento, se procede con el examen de la tercera capa de abstracción, en cuyo foco son los sensores mencionados anteriormente.

En primer lugar se realiza un ejemplo con un sensor de distancia DMS, exactamente el sensor del pecho del Bioloid el cual debe devolver la distancia del objeto que se ubica enfrente. Para conocer su correcto valor de distancia, se señala en un tablón unas marcas con un medidor métrico, diferentes distancias a la que se puede mover el objeto. Así mediante el software se debe obtener un valor muy próximo al que se tiene en el tablón con el objeto. Se puede aclarar la idea del experimento, con las siguientes imágenes:

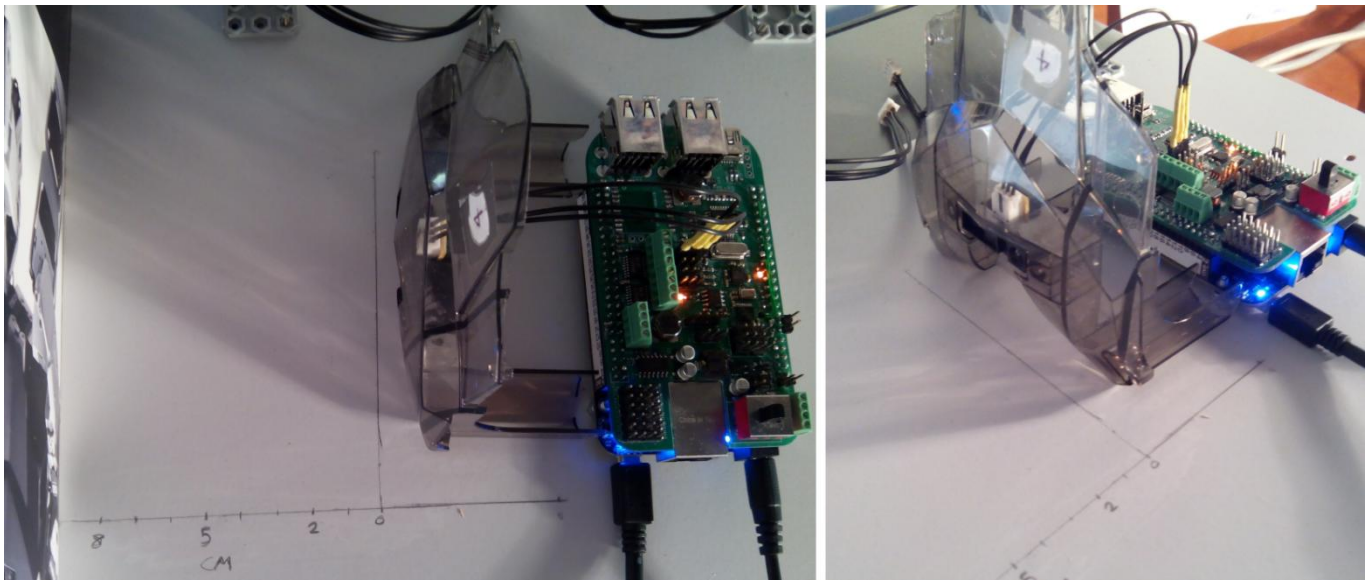


Figura 36: Montaje realizado en test 3 Sensor Sharp

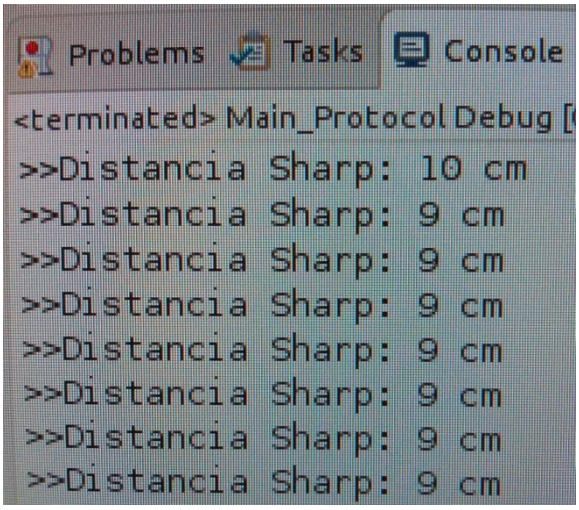


Figura 37: Resultado visualizado por consola

5.2.4.4 Test 4: GyroSensor

Otro test realizado para la validación, se realiza para el giroscopio del Kit del robot Bioloid, en cuyo objetivo es conocer la velocidad angular a la que se mueve uno de los dos ejes que se pueden medir con este sensor. En concreto para este, se realizar la lectura de la velocidad angular del ejeX y se muestra a través de la consola del PC.

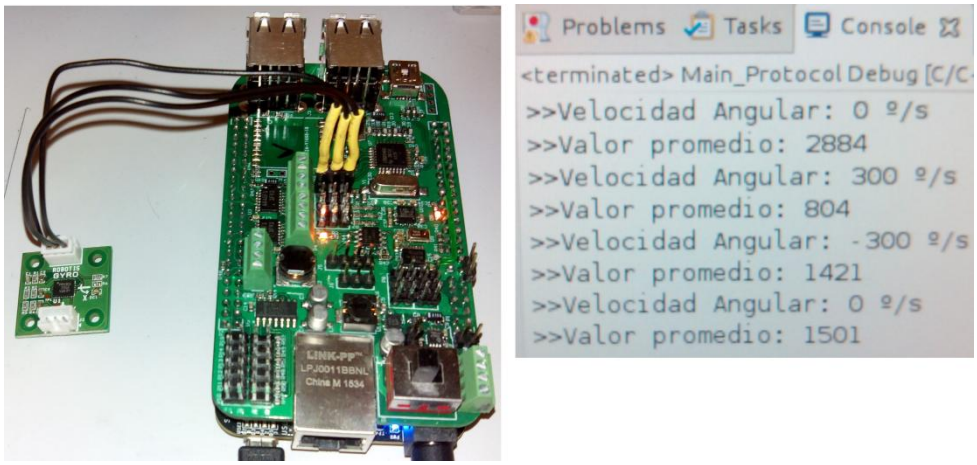


Figura 38: Montaje y resultados test4 con GyroSensor

5.2.4.4 Test 5: HC-SR04

Para el último sensor de la capa de abstracción de sensores y actuadores, se tiene el sensor de distancia HC-SR04, con el que se persigue el mismo objetivo del test (Test 3), pero siendo el protagonista el sensor de ultrasonidos. Se sitúa el sensor delante de un objeto y este debe obtener la lectura de distancia correcta. La comprobación se realiza por medio del tablón comentado también anteriormente y con los resultados que se adjuntan a continuación:

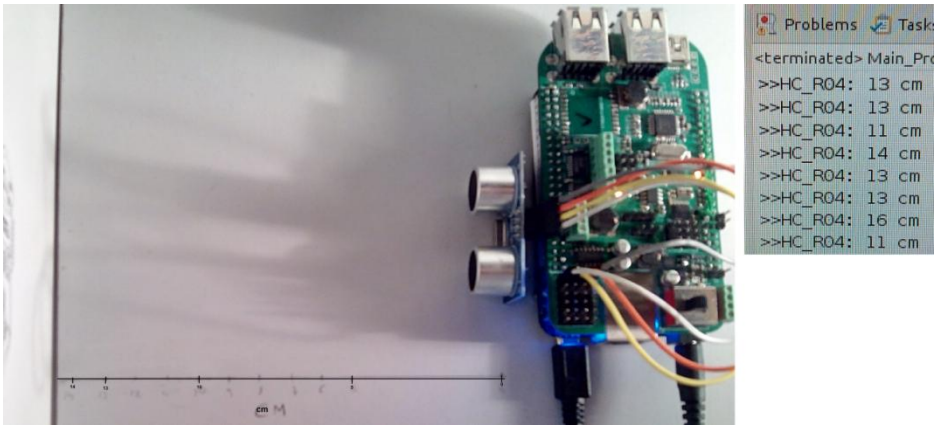


Figura 39: Montaje y resultado test5 con HC-SR04

5.2.4.5 Test 6: en RobotBioloid

Finalmente, volviendo a la capa de abstracción de movimiento y para dar por cerrado el API del proyecto, se realiza el test 6 en el cual se llevar a cabo el movimiento del androide. Detallando un poco más, se realizan las órdenes de caminar al robot hasta que el sensor de distancia DMS situado en la cabeza detecte un objeto. A partir de ese punto, el robot se dispondrá a dar manotazos como si de un combate de sumo se tratase. Una vez acabada la secuencia de los manotazos, si el robot no detecta ningún objeto, sigue con la tarea de caminar.

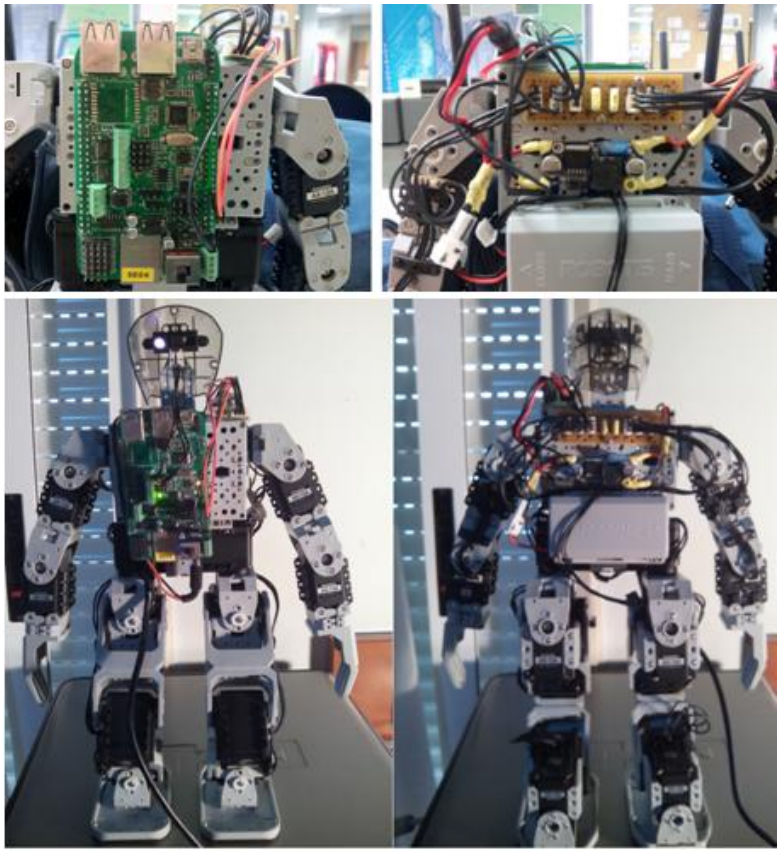


Figura 40: Montaje final Robot Bioloid

Con la realización de éste test, se da por finalizada, comprobada y validada la API de interacción de sensores y actuadores en robotBioloid, basado en BeagleBone Black y se da paso a la conclusiones del mismo. Para más información sobre el código de los test, se dispone del contendio en el apartado Anexos.

7. CONCLUSIONES

En conclusión, se puede afirmar que los objetivos han sido alcanzados con éxito a partir de la implementación de una API para interacción de sensores y actuadores en robot Bioloid, basado en BeagleBone Balck RevC mediante:

- Diseño y desarrollo de una capa de abstracción de las comunicaciones, como se ha mostrado en la sección 5.2.1 y el test1 IMU-Dynamixel.
- Diseño y desarrollo de una capa de abstracción de sensores y actuadores, expuesta en el punto 5.2.3 y test 3, 4 y 5.
- Diseño y desarrollo de una capa de abstracción de movimiento, comprobado en el test 6 Robot Bioloid y utilizado en los trabajos:

  -Diseño y desarrollo de un módulo de gestión de IMU para Bioloid sobre BeagleBone Black. Antoni Benavent

  -API de locomoción para robot Bioloid en Beaglebone black. Juan Carlos Brenes
- Validación de las diferentes API desarrolladas, a través de diversos test, así como un ejemplo de control que abarca toda la API.

La finalización del proyecto ha permitido la extracción de una serie de conclusiones globales sintetizadas a continuación:

- En primer lugar, ha proporcionado la adquisición de conocimientos acerca de los elementos que componen un sistema robótico y empotrado.
- Ha permitido la puesta en práctica de conocimientos aprendidos durante el curso de máster.
- Ha permitido usarse en otros trabajos, verificando así su funcionamiento.
- Y por último, ha servido de preparación para la búsqueda de diversas soluciones ante la aparición de un problema.

En resumen, se han obtenido unos resultados del todo favorables y además se quiere dar las gracias a todos los que de una forma u otra han contribuido a la posible realización del proyecto.

8. BIBLIOGRAFÍA

[Sem] "Guia_Uso_BeagleBoneBlackRevC.pdf". Manual de Prácticas de la asignatura Sistemas Empotrados del Máster de Automática e Informática Industrial, DISCA 2016.

[Dyn] "AX_12.pdf". Manual de usuario de Robotis, 2006.

[Rcape] "ROBOcape_for_BeagleBoneblack". Guía usuario, migalgil@gmail.com.

[JCB] "EquacionesSharp.xls", Juan Carlos Brenes.

[DM] "DerekMolloy.ie", Disponible en:

http://derekmolloy.ie/beaglebone/

[SAV] "Arduino y Dynamixel-12a", Disponible en:

http://savageelectronics.blogspot.com.es/2011/01/arduino-y-dynamixel-ax-12.html

[TIC] "Tician/CM530", Disponible en:

https://github.com/tician/cm530

2. Anexos

Autor: Sisternes Roses, David.

Tutor: Blanes Noguera, Juan Francisco.

Titulación: Máster Universitario en Automática e Informática Industrial.

ÍNDICE DEL CONTENIDO

2.1. Código fuente

Autor: Sisternes Roses, David.

Tutor: Blanes Noguera, Juan Francisco.

Titulación: Máster Universitario en Automática e Informática Industrial.

ÍNDICE DEL CONTENIDO

1. CAPA ABSTRACCIÓN DE LAS COMUNICACIONES

1.1 Archivo “BlackDynamixel.h”

```
//===================================================================================
// Name        : BlackDynamixel.h
// Author      : David Sisternes Roses
// Version     : 1.0 - 05/05/2016
// Copyright   : Free_Green_Team
// Description : Cabezera para el manejo y manipulación de la comuncación TTL
//               con servomotores Dynamixel y BeagleBone Black Rev C
//===================================================================================

#ifndef BLACKDYNAMIXEL_H
#define BLACKDYNAMIXEL_H_

#include <cstdint>
#include <stdio.h>
#include "serialib.h"
#include "InstrucDynamixel.h"

/* Declaración de Funciones */

/* Función Open_Connection *//*
  Abre y inicia la comunicación con el puerto especificado en (const char *Device)
  con una velocidad de transmisión (const uint32_t Bauds).
  Devuelve true si la conexión se ha realizado correctamente.
  Ex:
   Open_Connection("/dev/ttyO4",115200);
*/
bool Open_Connection(const char *Device,const uint32_t Bauds);

/* Función Close_Connection *//*
   Cierra la comuniación con el puerto especificado.
   No devuelve nada.
   Ex:
    Close_Connection(void);
*/
void Close_Connection(void);

/* Función Send_Data *//*
   Envía los datos de (uint8_t *packetOut), con la longitud (uint16_t length)
indicada.
   No devuelve nada.
   Ex:
    uint8_t packet_O[4]={'H','o','l','a'};
    Send_Data(packet_O,4);
*/
void Send_Data(uint8_t *packetOut,uint16_t length);
```

```
/* Función Received_Data *//*
   Lee el número de datos especificado por (uint16_t max_data) y guarda
   en (uint8_t *destination) los datos validos.
   Devuelve el número de bytes validos.
   Ex:
    uint8_t packet_I[255]="";
    uint8_t num_bytes=0;

    num_bytes = Received_Data(packet_I,255);
*/
uint8_t Received_Data(uint8_t *destination, uint16_t max_data);

#endif /* BLACKDYNAMIXEL_H_ */
```

2. CAPA DE ABSTRACCIÓN DE MOVIMIENTOS

2.1 Archivo “InstrucDynamixel.h”

```
//============================================================================
// Name        : InstrucDynamixel.h
// Author      : David Sisternes Roses
// Version     : 1.0 - 05/05/2016
// Copyright   : Free_Green_Team
// Description : Cabezera para el manejo y manupulación de instruciones con
//               servomotores Dynamixel.
//============================================================================

#ifndef INSTRUCDYNAMIXEL_H_
#define INSTRUCDYNAMIXEL_H_

#include <cstdint>
#include <stdio.h>
#include <string.h>
#include "BlackDynamixel.h"
#include "DynamixelAXDef.h"

#define BUFFER_SIZE      200 /* Tamaño del Buffer de entrada */
#define DATA_SIZE        200 /* Tamaño del Buffer de salida */

/* Declaración de Funciones */

/* Función Get Packet *//*
  Se añade al vector (uint8_t *param), la cabecera 0xff 0xff y se calcula el Get_CheckSum.
  (uint8_t packetOut[DATA_SIZE]), es el vector de salida donde quedarán guardados los
datos.
  (uint16_t *length) es el puntero con el número de elementos que contiene (uint8_t
*param).
  No devuelve nada.
  Ex:
   uint16_t lng=3;
   uint8_t  params[]={0x00,0x02,0x01,'}'};
   uint8_t  packet_O[255];
   Get_Packet(params,packet_O,&lng);
 */
void Get_Packet(uint8_t *param,uint8_t packetOut[DATA_SIZE],uint16_t *length);
```

```
/* Función Get_Checksum *//*
   Se le pasa un vector (uint8_t *param) con la cantidad de elemenots (uint16_t length) que
contiene
   y devuelve el byte de CheckSum necesario para el correcto envío de datos.
   Ex:
     uint8_t chck=0;
     uint8_t dat_Out[]={0xff,0xff,0x05,0x05,0x03,0x1e,0xff,0x03};
     chck = Get_Checksum(dat_Out,8);
 */
uint8_t Get_Checksum(uint8_t *param,uint16_t length);

/* Función Paralen *//*
   Devuelve el número de elementos contenidos en (uint8_t *param), cuenta bytes hasta
hallar '}'.
   Ex:
    uint16_t num_id=0;
    uint8_t id[]={1,2,3,10,18,'\0'};
    num_id = Paralen(id);
*/
uint16_t Paralen(uint8_t *param);

/* Función Ping_Servo *//*
  Se realiza un PING al servo especificado en (uint8_t id) para saber si está conectado y
disponible.
  Devuelve 0 si no hay error, en caso contrario devuelve el error que se ha producido.
  Ex:
   Ping_Servo(1);
   ó
   Ping_Servo(S_Dyn1);
*/
uint8_t Ping_Servo(uint8_t id);

/* Función Reset_Servo *//*
   Resetea el nº de servo Dynamixel especificado en (uint8_t id), estableciendo los
   parámetros del servo con valores de fábrica.
   Devuelve 0 si no hay error, en caso contrario devuelve el error que se ha producido.
   Ex:
    Reset_Servo(1);
    ó
    Reset_Servo(S_Dyn1);
*/
uint8_t Reset_Servo(uint8_t id);
/* Función Write_Servo *//*
   Se envía la trama la cual provocará una escritura en los parámetros de cada servo.
   (uint8_t id) corresponde con el identificador del servo.
   (uint16_t length) corresponde con la longitud de parámetros que se enviarán, puede ser
~Automatic~ si
   en el vector (uint8_t *Param), el último elemento es '}', en caso contrario se debe
especificar el tamaño
   del vecto (uint8_t *Param).
   (uint8_t instruction) corresponde con la instrucción que se quiere realizar sobre el
servo.
   (uint8_t *Param) contiene los parámetros que se le quieren enviar al servo.
   Se puede realizar un Write con Sincronización, instrucción (INST_REG_WRITE) y función
Action_Servo();
   o realizar un Write directo con la instrucción (INST_WRITE).
   Devuelve 0 si no hay error, en caso contrario devuelve el error que se ha producido.
```

```
   Ex:
     uint8_t Params[3] = {0x03,0x02,'}'};
     uint8_t error=0;
     error=Write_Servo(S_Dyn1,Automatic,INST_WRITE, Params);

     ó
     uint8_t Params[] = {0x03,0x02};
     uint8_t error=0;
     error=Write_Servo(S_Dyn3,2,INST_WRITE, Params);

     ó
     uint8_t Params[] = {0x1e,0xff,0xff,'}'};
     uint8_t error=0;
     error=Write_Servo(BROADCASTING_ID,Automatic,INST_WRITE, Params);

     ó
     uint8_t Params[3] = {0x03,0x02,'}'};
     uint8_t error=0;
     error=Write_Servo(S_Dyn1,Automatic,INST_REG_WRITE, Params);
     Action_Servo();
*/
uint8_t Write_Servo(uint8_t id,uint16_t length,uint8_t instruction,uint8_t *param);

/* Función Action_Servo *//*
  Provoca la ejecución de las instrucciones depositadas en el buffer de los servos
(enviadas
  con la instrucción "INST_REG_WRITE") de manera sincronizada.
  No devuelve nada.
  Ex:
   uint8_t Parametros[3] = {0x03,0x02,'}'};
   uint8_t error=0;
   error=Write_Servo(S_Dyn1,Automatic,INST_REG_WRITE, Parametros);
   Action_Servo();
*/
void Action_Servo(void);

/* Función SyncWrite_Servo *//*
   Se envía la trama la cual provocará una escritura en los parámetros de cada servo.
   A diferencia de la Write_Servo, permite el envío simultáneo de datos a varios servos
   dynamixel, escribiendo en los mismos registros de todos los servos.
   (uint8_t *id) es el vector que contiene los diferentes "Id's" de los servomotores.
    ADVERTENCIA!!! el vector uint8_t *id de contener el caracter '}' como elemento final.
    por ejemplo: uint8_t id[]={0x01,0x03,'}'};
   (uint16_t length) es el numero de registros "Address" que se quieren escribir en dichos
servos.
   (uint8_t *param) es el vector con los parámetros que se quieren cambiar en dichos
servos.
   No devuelve nada.
   Ex:
    uint8_t Dynamixels[5] = {1,2,3,4,5,'}'};
    uint8_t params[]={0x1e,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
    SyncWrite_Servo(Dynamixels,2,params);
*/
void SyncWrite_Servo(uint8_t *id, uint16_t length, uint8_t *param);
```

```
/* Función Read_Servo *//*
   Lee el número de registros (uint16_t lng_reg) del dynamixel (uint8_t id) y los guarda en
(uint8_t *value).
   Devuelve 0 si no hay error, en caso contrario devuelve el error que se ha producido.
   Ex:
      uint8_t error=0;
      uint8_t Read_Value[2]="";
      error = Read_Servo(2,P_PRESENT_POSITION_L,2,Read_Value);
*/
uint8_t Read_Servo(uint8_t id,uint8_t Address,uint16_t lng_reg,uint8_t *value);

#endif /* INSTRUCDYNAMIXEL_H_ */
```

2.2 Archivo “UtilDynamixel.h”

```
//============================================================================
// Name        : UtilDynamixel.h
// Author      : David Sisternes Roses
// Version     : 1.0 - 05/06/2016
// Copyright   : Free_Green_Team
// Description : Cabezera para el manejo y manupulación de utilidades en los
//               servomotores Dynamixel.
//============================================================================

#ifndef UTILDYNAMIXEL_H_
#define UTILDYNAMIXEL_H_

#include <cstdint>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "BlackDynamixel.h"
#include "InstrucDynamixel.h"
#include "DynamixelAXDef.h"

#define NUM_DYNAMIXELS  18
/* Declaración de Funciones */
//============================================================================
// FUNCIONES DE CONFIGURACIÓN O ESCRITURA EN DYNAMIXEL
//============================================================================
/*  Función set_Id *//*
  Establece el Id del dynamixel indicado en (uint8_t id_old), por el Id de (uint8_t
id_new).
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
     set_Id(S_Dyn1,S_Dyn2); -> Establece el Identificador del servo de 1 a 2.
     ó
     set_Id(1,4);                  -> Establece el Identificador del servo de 1 a 4.

*/
uint8_t set_Id(uint8_t id_old,uint8_t id_new);
```

```
/*  Función set_BauRate *//*
  Establece la velocidad actual de trasmisión del dynamixel indicado en (uint8_t id) a
la velocidad
  indicada en (uint32_t BaudRate).
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
     set_BaudRate(1,19200); -> Establece el BaudRate a la velocidad de 19200 en el
dinamixel 1.
*/
uint8_t set_BaudRate(uint8_t id,uint32_t BaudRate);

/*  Función set_ReturnDelay *//*
  Establece el tiempo de retardo de la respuesta de los paquetes de los dynamixel.
  En (uint8_t id) se indica el dynamixel sobre el que se envía la configuración.
  En (uint16_t us) se indica el tiempo de retardo en usec. Rango de operación (0 -
508)usec.
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
    set_ReturnDelay(1,400); -> Establece un Return Delay de 400us en el dynamixel 1.
*/
uint8_t set_ReturnDelay(uint8_t id,uint16_t us);

/*  Función set_Mode *//*
  Establece el modo funcionamiento del dynamixel indicado en (uint8_t id).
  El primer modo puede ser en rotación continua (WHEEL), sin limites CW y CCW.
  El otro modo es con los limites CW y CCW, en el cual el servo no sobrepasará dichos
limites.
  En (uint16_t Angle_Limit_CW) se indica el límite inferior. Rango (0 - 1023) -> (0 -
300)°
  En (uint16_t Angle_Limit_CCW) se indica el límite superior. Rango (0 - 1023) -> (0 -
300)°
  En (uint8_t type) se indica si los datos introducidos en (uint16_t Angle_Limit_CW) y
  (uint16_t Angle_Limit_CCW), son en grados o en hexadecimal.
  Nota:Para configurar el modo de rotación continua poner a O Angle_Limit_CW y
Angle_Limit_CCW.
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
    set_Mode(2,WHEEL,0,0,DEGRESS); -> Establece el dynamixel 2 con modo rotación
continua. Para poner en marcha
    ó                                    el dynamixel ver función Mode_Wheel();
detallada más abajo.
    set Mode(3,SERVO,0,1023,HEX);  -> Establece el dynamixel 3 en modo Servo, configura
el límite CW a 0 (0°)
                                     y el límie CCW a 1023 (300°), como los datos que
se introducen son hexadecimales,
                                     se usa la etiqueta HEX.
*/
uint8_t set_Mode(uint8_t id,uint8_t mode, uint16_t Angle_Limit_CW, uint16_t
Angle_Limit_CCW, uint8_t type);
```

```
/*  Función set_HLimitTemp *//*
  Establece la temperatura máxima a la que puede trabajar el dynamixel.
  En caso de sobrepasar dicho límite, saltará una alarma y el dynamixel se desactivará.
  En (uint8_t id) se indica el dynamixel a configurar.
  En (uint8_t tmp) se indica el límite de temperatura al que saltará la alarma. Rango
(0-150)°C
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
    set_HLimitTemp(5,85) -> Establece el límite de temperatura del dynamixel 5 a 85°C.
*/
uint8_t set_HLimitTemp(uint8_t id,uint8_t tmp);

/*  Función set_Voltage *//*
  Establece los valores máximo y mínimo de tensión del dynamixel.
  Si no se llega al valor mínimo o se sobrepasa el valor máximo se generará una señal
de alarma.
  En (uint8_t Low_Limit_Voltage) se indicará el límite inferior de tensión. Rango (50 -
250)
  En (uint8_T Highest_Limit_Voltage) se indicará el límite superior de tensión. Rango
(50 -250)
  Nota: El valor de tensión indicado es 10 veces más del real. Por ejemplo un valor 80
-> 8V
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
     set_LimitVoltage(1,60,190); -> Establece en el dynamixel 1, el nivel mínimo de
tensión a 6 V
                                    y el nivel máximo de tensión a 19 V.
*/
uint8_t set_LimitVoltage(uint8_t id,uint8_t Low_Limit_Voltage, uint8_t
Highest_Limit_Voltage);

/*  Función set_MaxTorque *//*
  Establece el máximo torque del dynamixel. Si se establece el parámetro a 0, el
dynamixel entra
  en modo (FreeRun).
  En (uint8_t id) se indica el dynamixel a configurar.
  En (uint16_t Mtorque) se indica el valor de torque. Rango (0 - 1023) -> (0 - 100)%
  En (uint8_t type) se indica el formato de valo que se introduce. Debe ser HEX si los
datos que
  se introducen en (uint16_t Mtorque) son en hexadecimal o PERCENT si son datos de 0% a
100% de torque.
  Nota: Para modo FreeRun Mtorque = 0.
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
    set_MaxTorque(1,1023,HEX);        -> Establece el torque del dynamixel 1 a 1023
(100%)
    ó
    set_MaxTorque(1,0, PERCENT);  -> Establece el torque del dynamixel 1 al 0%, modo
FREERUN
*/
uint8_t set_MaxTorque(uint8_t id,uint16_t Mtorque,uint8_t type);
```

```
/*  Función set_StatusPacket *//*
  Establece el modo de respuesta de los dynamixel ante el envío de paquetes.
  En (uint8_t id) se indica el dynamixel a configurar.
  En (uint8_t set) se determina el modo de respuesta de los dynamixel.
   Establecer set = NONE para no recibir respuesta del dynamixel, sea cual sea el
paquete que se envíe.
   Establecer set = READ para recibir respuesta Solo de los paquetes con la instrucción
READ.
   Establecer set = ALL  para recibir respuesta siempre y cuando no se haga un
BROADCAST_ID.
   Return 0 si no hay error, en caso de error devuelve el número de error.
   Ex:
     set_StatusPacket(1,ALL); -> Establece dynamixel 1, para que responda a todos los
paquetes sin BROADCAST.
     ó
     set_StatusPacket(1,NONE); -> Establece dynamixel 1, para que no responda a ningún
paquete.
*/
uint8_t set_StatusPacket(uint8_t id,uint8_t set);

/*  Función set_AlarmLED *//*
  Establece en que errores parpadeará el led del dynamixel.
  La función sigue la operación lógica "OR", permitiendo configurar varios errores a la
vez.
  Por ejemplo: Input_Voltage_Error | Angle_Limit_Error, configura AlarmaLED para que se
active
  cuando se produzca un error de voltage o de límite de águlo.
  En (uint8_t id) se indica el dynamixel a configurar.
  En (uint8_t errors) se indican los errores en los que saltará la alarmaLED.
  Tabla Errores:
     ->Input_Voltage_Error
     ->Angle_Limit_Error
     ->Overheating_Error
     ->Range_Error
     ->CheckSum_Error
     ->Overload_Error
     ->Instruction_Error
   Return 0 si no hay error, en caso de error devuelve el número de error.
   Ex:
     set_AlarmLED(1,Overheating_Error|Input_Voltage_Error); -> Establece en el
dynamixel 1 los errores de Overheating y de Input_Voltage_Error
*/
uint8_t set_AlarmLED(uint8_t id,uint8_t errors);
```

```
/*  Función set_AlarmShutdown *//*
  Establece en que errores se desactivará el troque del dynamixel.
  La función sigue la operación lógica "OR", permitiendo configurar varios errores a la
vez.
  Por ejemplo: Input_Voltage_Error | Angle_Limit_Error, configura AlarmShutdown para
que desactive
  el torque del dynamixel cuando se produzca un error de voltage o de límite de ángulo.
  En (uint8_t id) se indica el dynamixel a configurar.
  En (uint8_t errors) se indican los errores en los que se desactivará el dynamixel.
  Tabla Errores:
     ->Input_Voltage_Error
     ->Angle_Limit_Error
     ->Overheating_Error
     ->Range_Error
     ->CheckSum_Error
     ->Overload_Error
     ->Instruction_Error
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
    set_AlarmShutdown(1,Overheating_Error|Input_Voltage_Error); -> Establece en el
dynamixel 1 los errores de Overheating y de Input_Voltage_Error
*/
uint8_t set_AlarmShutdown(uint8_t id,uint8_t errors);

/*  Función set_HoldingTorque *//*
  Establece la activación/desactivación del torque del dynamixel.
  En caso de desactivar torque el dynamixel entra en modo FreeRun (zero torque).
  En (uint8_t id) se indica el dynamixel a configurar.
  En (uint8_t set) se indica el estado que se quiere configurar en el dynamixel.
   set = ON para activar torque.
   set = OFF para desactivar torque.
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
    set_HoldingTorque(1,ON); -> Activa el torque del dynamixel 1.
    ó
    set_HoldingTorque(1,OFF); -> Desactiva el torque del dynamixel 1.
*/
uint8_t set_HoldingTorque(uint8_t id,uint8_t set);

/*  Función set_LED *//*
  Establece la activación/desactivación del led del servomotor.
  En (uint8 t id) se indica el dynamixel a configurar.
  En (uint8_t set) se indica el estado que se quiere configurar en el LED del
dynamixel.
   set = ON para activar LED.
   set = OFF para desactivar LED.
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
    set_LED(3,OFF); -> Apaga el LED del dynamixel 3.
    ó
    set_LED(1,OFF); -> Enciende el LED del dynamixel 1.
*/
uint8_t set_LED(uint8_t id,uint8_t set);
```

```
/*  Función set_Compliance_M_S *//*
  Establece.
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
    set_Compliance_M_S(1,0,0,32,32); -> Establece en el dynamixel 1 el CW y CCW Margin
a 0 y el CW y CCW Slope a 32.
*/
uint8_t set_Compliance_M_S(uint8_t id,uint8_t CW_Margin,uint8_t CCW_Margin,uint8_t
CW_Slope, uint8_t CCW_Slope);

/*  Función set_Punch *//*
  Establece la corriente que se suministra al servo durante la operación.
  En (uint8_t id) se indica el dynamixel a configurar.
  En (uint16_t set) se indica el valor de corriente que se le suministra. Rango (0 -
1023) - (0-100)%
  En (uint8_t type) se indica el tipo de datos que se va a introducir. HEX para datos
hexadecimales,
  PERCENT para datos en %.
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
    set_Punch(1,32, HEX); -> Establece el punch del dynamixel 1 a 32 (~3%)
    ó
    set_Punch(1,29, PERCENT); -> Establece el punch del dynamixel 1 a 20 %
*/
uint8_t set_Punch(uint8_t id,uint16_t punch, uint8_t type);
/*  Función set_Lock *//*
  Activa el modo de escritura reducido. Si se activa el bit Lock, solo podrán
  ser escritas las direcciones de la (0x18 a la 0x23), el resto no podrán ser
modificadas.
  Nota: Para realizar el desbloqueo se deberá desconectar la alimentación del
dynamixel.
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
    set_Lock(2); -> Activa el modo escritura reducido en el dynamixel 2.
*/
uint8_t set_Lock(uint8_t id);
/*  Función set_TorqueLimit *//*
  Establece el máximo torque del dynamixel. Si se establece el parámetro a 0, el
dynamixel entra
  en modo (FreeRun).
  En (uint8_t id) se indica el dynamixel a configurar.
  En (uint16_t torque_limit) se indica el valor de torque. Rango (0 - 1023) - (0 -
100)%
  En (uint8_t type) se indica el formato de valo que se introduce. Debe ser HEX si los
datos que
  se introducen en (uint16_t Mtorque) son en hexadecimal o PERCENT si son datos de 0% a
100% de torque.
  Nota: Para modo FreeRun Mtorque = 0. A diferencia de la función set_MaxTorque() ver
más arriba, el valor del
  límite del torque es volátil, con lo que al quitar la alimentación se perderá la
configuración de este parámetro.
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
     set_TorqueLimit(1,1023,HEX); -> Establece el Torque Limit del dynamixel 1 al 1023
(100%)
     ó
     set_TorqueLimit(1,50,PERCENT); -> Establece el Torque Limit del dynamixel 1 al 50%
*/
uint8_t set_TorqueLimit(uint8_t id,uint16_t torque_limit, uint8_t type);
/*  Función set_Posl *//*
```

```
  Pone el dynamixel con Id(uint8_t id) en la posición indicada en (uint16_t pos). Rango
(0 - 1023) -> (0° - 300°)
  En (uint8_t type) se indica el tipo de datos que se va a introducir. HEX para datos
hexadecimales,
  DEGRESS para datos en grados.
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
    set_Pos(1,512,HEX); -> Establece el dynamixel 1 a la posición 512 (150°)
    ó
      set_Pos(1,200,DEGREES); -> Establece el dynamixel 1 a la posición 200°
*/
uint8_t set_Pos(uint8_t id,uint16_t pos, uint8_t type);

/*  Función set_PosVel *//*
  Pone el dynamixel con Id(uint8_t id) en la posición indicada en (uint16_t pos)
  a la velocidad indicadad en (uint16_t vel). Rangos: pos (0 - 1023) -> (0° - 300°)
                                                                        vel (0 -
1023) -> (0 - 114) RPM
  En (uint8_t type_pos) se indica el tipo de datos en la posición que se va a
introducir. HEX para datos hexadecimales,
  DEGREES para datos en grados.
  En (uint8_t type_vel) se indica el tipo de datos en la velocidad que se va a
introducir. HEX para datos hexadecimales,
  RPM para datos en revoluciones por minuto.
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
    set_PosVel(1,0,800,HEX,HEX); -> Establece la posición a 0 (0°) del dynamixel 1 con
una velocidad de 800 (~88 RPM)
    ó
    set_PosVel(1,210,80,DEGREES,RPM); -> Establece la posición a 210° del dynamixel 1
con una velocidad de 80 RPM
*/
uint8_t set_PosVel(uint8_t id, uint16_t pos, uint16_t vel, uint8_t type_pos, uint8_t
type_vel);

/*  Función set_PosVelPreload *//*
  Pone el dynamixel con Id(uint8_t id) en la posición indicada en (uint16_t pos)
  a la velocidad indicadad en (uint16_t vel).
  A diferencia de set_PosVel(), esta función carga los parámetros que se van a
configurar
  en el dynamixel, pero no se harán válidos hasta que se ejecute la función
Action_Servo();
  Rangos: pos (0 - 1023) -> (0° - 300°)
          vel (0 - 1023) -> (0 - 114) RPM
  En (uint8_t type_pos) se indica el tipo de datos en la posición que se va a
introducir. HEX para datos hexadecimales,
  DEGREES para datos en grados.
  En (uint8_t type_vel) se indica el tipo de datos en la velocidad que se va a
introducir. HEX para datos hexadecimales,
  RPM para datos en revoluciones por minuto.
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
    set_PosVelPreload(2,512,600,HEX,HEX); -> Precarga la posición 512 (150°) y
velocidad 600 (~66RPM) al dynamixel 2
    sleep(5);                              -> espera de 5s para el ejemplo
    Action_Servo();                        -> función para que se ejecuten los
parámetros precargados.
*/
uint8_t set_PosVelPreload(uint8_t id, uint16_t pos, uint16_t vel,uint8_t type_pos,
uint8_t type_vel);
/*  Función set_PosSync *//*
```

```
  Establece la posición de cada dynamixel contenida en una tabla o matriz.
  (uint8_t *id) debe contener los identificadores de los dynamixel.
  (const uint16_t pos[][NUM_DYNAMIXELS]) debe contener las posiciones de cada servo.
  (uint8_t column) establece que columna de posiciones se va a utilizar.
  En (uint8_t type) se indica el tipo de datos en la posición que se va a introducir.
HEX para datos hexadecimales,
  DEGREES para datos en grados.
  Ex:
   set_PosSync(Dynamixels,Robot_ready_pos,0,HEX); -> Establece a cada dynamixel la
posición contenida en la tabla Robot_ready_pos con valores en hexadecimal,
                                                    dicha tabla se encuentra en
BioloidPosDef.h al igual que el vector Dynamixels.
                                                    Column indica la columna de la
tabla que se quiere cargar, en este caso es la columna 0.
   ó
   set_PosSync(Dynamixels,Robot_ready_pos_deg,0,DEGREES); -> Establece a cada dynamixel
la posición contenida en la tabla Robot_ready_pos_deg con valores en grados,
                                                          dicha tabla se
encuentra en BioloidPosDef.h al igual que el vector Dynamixels.
                                                          Column indica la
columna de la tabla que se quiere cargar, en este caso es la columna 0.
*/
void set_PosSync(uint8_t *id,const uint16_t pos[][NUM_DYNAMIXELS],uint8_t
column,uint8_t type);

/*  Función set_VelSync *//*
  Establece la velocidad de cada dynamixel contenida en una tabla o matriz.
  (uint8_t *id) debe contener los identificadores de los dynamixel.
  (const uint16_t vel[][NUM_DYNAMIXELS]) debe contener las velocidades de cada servo.
  (uint8_t column) establece que columna de velocidades se va a utilizar.
  En (uint8_t type) se indica el tipo de datos en la velocidad que se va a introducir.
HEX para datos hexadecimales,
  RPM para datos en revoluciones por minuto.
  Ex:
   set_VelSync(Dynamixels,Robot_ready_vel_rpm,0,RPM); -> Establece a cada dynamixel la
velocidad contenida en la tabla Robot_ready_vel con valores de revoluciones por minuto,
                                                    dicha tabla se encuentra en
BioloidPosDef al igual que el vector Dynamixels.
                                                    Column indica la columna de
la tabla que se quiere cargar, en este caso es la columna 0.
   ó
   set_VelSync(Dynamixels,Robot_ready_vel_hex,0,HEX); -> Establece a cada dynamixel la
velocidad contenida en la tabla Robot ready vel hex con valores hexadecimales,
                                                    dicha tabla se encuentra en
BioloidPosDef al igual que el vector Dynamixels.
                                                    Column indica la columna de
la tabla que se quiere cargar, en este caso es la columna 0.
*/
void set_VelSync(uint8_t *id,const uint16_t vel[][NUM_DYNAMIXELS],uint8_t
column,uint8_t type);
```

```
/*  Función set_MovSpeed *//*
  Establece la velocidad (uint8_t vel) del dynamixel con Id (uint8_t id).
  Rango ( 1 - 1023 ) -> (0.111 - 114 RPM).
  Nota: en caso de establecer (uint8_t vel) = 0, el dynamixel configura su velocidad
  a la máxima que pueda ir (~114 RPM).
  En (uint8_t type) se indica el tipo de datos que se va a introducir. HEX para datos
hexadecimales,
  RPM para datos en revoluciones por minuto.
  Return 0 si no hay error, en caso de error devuelve el número de error.
  Ex:
    set_MovSpeed(1,0,HEX); -> Establece la velocidad del dynamixel 1 a 0 , configurando
la velociada a la máxima que pueda ir ~114 RPM. ó
    set_MovSpeed(1,60,RPM); -> Establece la velocidad del dynamixel 1 a 60 RPM.
*/
uint8_t set_MovSpeed(uint8_t id, uint16_t vel,uint8_t type);
/*  Función mode_Wheel *//*
   Establece en el modo rotación contínua, el sentido de giro (uint8_t rotation) en el
dynamixel (uint8_t id), con la velocidad
   especificada en (uint16_t speed).
   En (uint8_t type) se indica el tipo de datos que se va a introducir. HEX para datos
hexadecimales,
   RPM para datos en revoluciones por minuto.
   Return 0 si no hay error, en caso de error devuelve el número de error.
   Ex:
    set_Mode(2,WHEEL,0,0,DEGREES); -> Primero se establece el dynamixel 2 en modo
rotación continua.

    mode_Wheel(1,LEFT,200,HEX);    -> Establece el sentido de giro a la izquierda del
dynamixel 2 y la velocidad a 200 (~22 RPM).

*/
uint8_t mode_Wheel(uint8_t id, uint8_t rotation, uint16_t speed, uint8_t type);
/*  Función mode_WheelPreload *//*
   Establece en el modo rotación contínua, el sentido de giro (uint8_t rotation) en el
dynamixel (uint8_t id), con la velocidad
   especificada en (uint16_t speed). A diferencia de la función anterior, está carga
los parámetros para la rotación continua y
   no se ejecutarán hasta que se envíe la trama de Action_Servo().
   En (uint8_t type) se indica el tipo de datos que se va a introducir. HEX para datos
hexadecimales,
   RPM para datos en revoluciones por minuto.
   Return 0 si no hay error, en caso de error devuelve el número de error.
   Ex:
    set_Mode(2,WHEEL,0,0,DEGREES); -> Primero se establece el dynamixel 2 en modo
rotación continua.

    mode_WheelPreload(1,LEFT,200,HEX); -> Establece el sentido de giro a la izquierda
del dynamixel 2 y la velocidad a 200 (~22 RPM).
    Action_Servo();                -> Provoca la ejecución de los parámetros que
estaban a la espera de Action_Servo.
*/
uint8_t mode_WheelPreload(uint8_t id, uint8_t rotation, uint16_t speed, uint8_t type);
/*  Función set_QuestDynamixel *//*
   Realiza un Ping a los servos que haya conectados.
   Devuelve el número de dynamixels conectados.
   Ex:
    printf("\nDyna encontrados: %u",set_QuestDynamixel());
*/
uint8_t set_QuestDynamixel();
//=============================================================================
```

```
// FUNCIONES DE LECTURA EN DYNAMIXEL
//=============================================================================
/*  Función readPos *//*
   Lee la posición del dynamixel indicado en (uint8_t id) y lo devuelve en formato
especificado en (uint8_t type).
   HEX para datos en hexadecimal, DEGREES para datos en grados.
   En caso de error de lectura, devolverá el valor 4096 o 0xfff.
   Ex:
    printf("\nLa posición del dynamixel 1 es: %u",readPos(1,DEGREES));
    ó
    printf("\nLa posición del dynamixel 1 es: %u",readPos(1,HEX));
*/
int16_t readPos(uint8_t id, uint8_t type);

/*  Función readSpeed *//*
   Lee la velocidad actual que lleva el dynamixel indicado en (uint8_t id) y lo
devuelve en formato especificado en (uint8_t type).
   HEX para datos en hexadecimal, RPM para datos en revoluciones por minuto.
   En caso de error de lectura, devolverá el valor 4096 o 0xfff.
   Ex:
    printf("\nLa velocidad del dynamixel 1 es: %u",readSpeed(1,HEX));
    ó
    printf("\nLa velocidad del dynamixel 1 es: %u",readSpeed(1,RPM));
*/
int16_t readSpeed(uint8_t id, uint8_t type);

/*  Función readMovSpeed *//*
   Lee el parámetro de configuracion de la velocidad del dynamixel indicado en (uint8_t
id) y lo devuelve en formato especificado en (uint8_t type).
   HEX para datos en hexadecimal, RPM para datos en revoluciones por minuto.
   En caso de error de lectura, devolverá el valor 4096 o 0xfff.
   Ex:
    printf("\nParámetro velocidad del dynamixel 1 es: %u",readMovSpeed(1,HEX));
    ó
    printf("\nParámetro velocidad del dynamixel 1 es: %u",readMovSpeed(1,RPM));
*/
int16_t readMovSpeed(uint8_t id, uint8_t type);

/*  Función readTemperature *//*
   Lee la temperatura del dynamixel indicado en (uint8_t id). el valor de la
temperatura se devuelve en °C.
   En caso de error de lectura, devolverá el valor 4096 o 0xfff.
   Ex:
    printf("\nLa temperatura del dynamixel 1 es: %u",readTemperature(1));
*/
int16_t readTemperature(uint8_t id);

/*  Función readVoltage *//*
   Lee el voltaje del dynamixel indicado en (uint8_t id). El valor será diez veces el
valor real, es decir
   el valor 80 equivale a 8 V.
   En caso de error de lectura, devolverá el valor 4096 o 0xfff.
   Ex:
    printf("\nVoltaje del dynamixel 1 es: %u",readVoltage(1));
*/
int16_t readVoltage(uint8_t id);
```

```
/*  Función readAngleLimit *//*
   Lee el límite del ángulo del dynamixel indicado en (uint8_t id) y lo devuelve en el
formato indicado en (uint8_t type),
   HEX para datos en hexadecimal, DEGREES para datos en grados.
   En (uint8_t Limit_Select) se especifica que límite se quiere leer, (CW/CCW)
   En caso de error de lectura, devolverá el valor 4096 o 0xfff.
   Ex:
    printf("\nLímite CW del dynamixel 1 es: %u",readAngleLimit(1,CW,HEX));
    ó
    printf("\nLímite CCW del dynamixel 1 es: %u",readAngleLimit(1,CCW,DEGREES));
*/
int16_t readAngleLimit(uint8_t id, uint8_t Limit_Select, uint8_t type);

/*  Función readLoad *//*
   Lee la carga que realiza el dynamixel indicado en (uint8_t id) y lo devuelve en el
formato indicado en (uint8_t type),
   HEX para datos en hexadecimal, PERCENT para datos en %.
   En caso de error de lectura, devolverá el valor 4096 o 0xfff.
   Ex:
    printf("\nCarga dynamixel 1 es: %u",readLoad(1,PERCENT));
*/
int16_t readLoad(uint8_t id,uint8_t type);

/*  Función checkRegister *//*
   Lee el Register Instruccion del dynamixel(uint8_t id) y indica si hay o no una
instrucción a la espera de Action_Servo().
   Devuelve 1 en caso que haya instrucción esperando y 0 en caso contrario.
   En caso de error de lectura, devolverá el valor 4096 o 0xfff.
   Ex:
    set_PosVelPreload(1,512,0,HEX,HEX);
    printf("\nCheck register dynamixel 1 es: %u",checkRegister(1));
    sleep(1);
    Action_Servo();
    printf("\nCheck register dynamixel 1 es: %u",checkRegister(1));
*/
int16_t checkRegister(uint8_t id);

/*  Función checkMovement *//*
   Lee el parámetro movement del dynamixel(uint8_t id) y indica si hay o no movimiento.
   Devuelve 1 en caso que el servo este moviendose, en caso de dynamixel parado
devuelve 0.
   En caso de error de lectura, devolverá el valor 4096 o 0xfff.
   Ex:
     set_Pos(1,0,DEGREES);
     sleep(2);
     set_Pos(1,300,DEGREES);
    if(checkMovement(1)){
        printf("\nDynamixel 1 en movimiento");
       }else
         {
           printf("\nDynamixel 1 Parado");
         }
*/
int16_t checkMovement(uint8_t id);
```

```
/*  Función checkLock *//*
   Lee el parámetro lock del dynamixel(uint8_t id) y indica si se ha activado el modo
escritura reducida.
   El modo escritura reducida, límita la escritura en las "Adress" de la 0x18 a la
0x23, las demás quedan bloqueadas.
   Devuelve 1 en caso que el servo este moviendose, en caso de dynamixel parado
devuelve 0.
   En caso de error de lectura, devolverá el valor 4096 o 0xfff.
   Ex:
    checkLock(1);
    set_Lock(1);
    checkLock(1);
*/
int16_t checkLock(uint8_t id);

/*  Función ledState *//*
   Lee el parámetros LED del dynamixel(uint8_t id) y devuelve si encuentra encendido o
apagado.
   En caso de error de lectura, devolverá el valor 4096 o 0xfff.
   Ex:
     ledState(1);
      set_LED(1,ON);
     ledState(1);
      set_LED(1,OFF);
*/
int16_t ledState(uint8_t id);

//=============================================================================
// FUNCIONES DE LECTURA PLANTA DEL PIE
//=============================================================================
/*  Función Read_Sharp_Foot *//*
   Lee la distancia del sensor Sharp ubicado en la parte delantera de la planta del
pie(uint8_t id).
   En caso de error de lectura, devolverá el valor 4096 o 0xfff.
   Ex:
     printf("\nSensor Sharp del pie %u: %u mm",Left_Foot,Read_Sharp_Foot(Left_Foot));
*/
int16_t Read_Sharp_Foot(uint8_t id);

/*  Función Read_Pressure *//*
   Lee la presión del sensor MF-01 indicado en(uint8_t sensor) y que se encuentra en la
planta del pie (uint8_t id).
   En caso de error de lectura, devolverá el valor 4096 o 0xfff.
   Ex:
    printf("\nPresión del sensor %u del pie %u: %u
N",PRESSURE_SENSOR_1,Right_Foot,Read_Pressure(Right_Foot, PRESSURE_SENSOR_1));
*/
int16_t Read_Pressure(uint8_t id, uint8_t sensor);
/*  Función Read_CM *//*
   Lee el centro de masas(uint8_t axis) del pie indicado en(uint8_t id).
   En caso de error de lectura, devolverá el valor 4096 o 0xfff.
   En (uint8_t axis) se indica de que eje se quiere leer el CM. (Axis_X ó Axis_Y)
   Ex:
    printf("CM_X pie %u: %u",Left_Foot,Read_CM(Left_Foot,Axis_X));
*/
int16_t Read_CM(uint8_t id, uint8_t axis);
```

```
/*  Función Read_CM_Axes *//*
   Lee el centro de masas de los ejes X e Y del pie indicado en (uint8_t id) y los pasa
por referencia.
   Ex:
*/
void Read_CM_Axes(uint8_t id, uint16_t *axis_x, uint16_t *axis_y);

#endif /* UTILDYNAMIXEL_H_ */
```

2.3 Archivo “BioloidMotions.h”

```
//============================================================================
// Name        : BioloidMotions.h
// Author      : David Sisternes Roses
// Version     : 1.0
// Copyright   : Free_Green_Team
// Description : Cabezera para el manejo y manupulación de los movimientos en
//               robot Bioloid.
//============================================================================

#ifndef BIOLOIDMOTIONS_H_
#define BIOLOIDMOTIONS_H_

#include <cstdint>
#include <stdio.h>
#include <string.h>
#include "DynamixelAXDef.h"
#include "UtilDynamixel.h"
#include "BioloidPosDef.h"
/* Declaración de Funciones */

/*  Función robotReady *//*
  Establece la posición inicial del robot
  No retorna nada.
  Ex:
     robotReady(); -> Establece la posición inicial del robot
*/
void robotReady(void);

/*  Función walkNStep *//*
  Realiza el número de pasos indicado en (uint8_t pasos)
  No retorna nada.
  Ex:
     walkNStep(6); -> realiza 6 pasos
*/
void walkNSteps(uint8_t pasos);
/*  Función walkStepL *//*
  Realiza un paso del robot empezando con el pié izquierdo
  No retorna nada.
  Ex:
     walkStepL(); -> realiza un paso, empezando con el pié izquierdo
*/
void walkStepL(void);
```

```
/*  Función walkStepR *//*
   Realiza un paso del robot empezando con el pié derecho
  No retorna nada.
  Ex:
     walkStepR(); -> realiza un paso, empezando con el pié derecho
*/
void walkStepR(void);

/*  Función walkStep4L *//*
  Realiza cuatro pasos del robot empezando con el pié izquierdo
  No retorna nada.
  Ex:
     walkStep4L(); -> realiza cuatro pasos, empezando con el pié izquierdo
*/
void walkStep4L(void);

/*  Función walkStep4R *//*
  Realiza cuatro pasos del robot empezando con el pié derecho
  No retorna nada.
  Ex:
     walkStep4R(); -> realiza cuatro pasos, empezando con el pié derecho
*/
void walkStep4R(void);

/*  Función Manotazos *//*
  Realiza la ejecución de unos pocos manotazos
  No retorna nada.
  Ex:
     Manotazos();
*/
void Manotazos(void);
#endif /* BIOLOIDMOTIONS_H_ */
```

3. CAPA DE ABSTRACCIÓN DE SENSORES Y ACTUADORES

3.1 Archivo “BlackSensors.h”

```
//============================================================================
// Name        : BlackSensors.h
// Author      : David Sisternes Roses
// Version     : 1.0 - 05/09/2016
// Copyright   : Free_Green_Team
// Description : Cabezera para el manejo y manupulación de los sensores
//               SharpGP2Dxx y GYRO de Robotis.
//============================================================================

#ifndef BLACKSENSORS_H_
#define BLACKSENSORS_H_

#include <iostream>
#include <unistd.h>
#include <fstream>
#include <string>
#include <sstream>
#include <math.h>
#include "GPIO.h"

using namespace std;
#define ADC_PATH "/sys/bus/iio/devices/iio:device0/in_voltage"
```

```
/* Declaración de Funciones */
/*  Función readAnalog *//*
  Realiza la lectura de la entrada analógica especificada en (int number)
  ATENCIÓN!!! Sólo se dispone de 4 entradas analógicas en ROBOcape (0,1,2,3)
  Ex:
     printf("Valor AIO: %u",readAnalog(0)); Lee la entrada analógica 0
     ó
     printf("Valor A2O: %u",readAnalog(2)); Lee la entrada analógica 2
*/
uint16_t readAnalog(int number);
/*  Función readSharp *//*
  Obtiene el valor de la distancia del sensor SharpGP2Dxx conectado a la entrad
especificada en (int number)
  ATENCIÓN!!! Sólo se dispone de 4 entradas analógicas en ROBOcape (0,1,2,3), solo se
puede
  conectar 4 sensores.
  Ex:
     printf("Distancia SharpGP2Dxx__0: %u",readSharp(0)); Lee sensor SharpGP2Dxx
conectado a la entrada analógica 0
*/
uint8_t readSharp(uint8_t number);

/*  Función readGyro *//*
  Obtiene el valor de la velocidad angular del sensor Gyrosensor conectado a la entrad
especificada en (int number)
  Cada Gyrosensor dispone de dos ejes, por tanto uso mínimonimo de entradas analógicas
a utilizar son dos.
  Ex:
     printf("Vel Angl Eje X: %u",readGyro(1)); Lee Gyrosensor conectado a la entrada
analógica 1
*/
int16_t readGyro(uint8_t number);

/*  Función readHC_SR04 *//*
  Obtiene el valor de la distancia del sensor HC_SR04 conectado en el conector 0 de la
ROBOcape, solo está implementado
  para un sensor.
  Ex:
     printf("Distancia HC_SR04: %u",readHC_SR04); Lee sensor HC_SR04 y devuelve la
distancia
*/
int16_t readHC_SR04(void);

long getUsec();


#endif /* BLACKSENSORS_H_ */
```

4. CABECERAS DE LAS DEFINICIONES

4.1 Archivo “DynamixelAXDef.h”

```
/*
 *  DynamixelAXdef.h
 *
 *  Created on: 3 de Mayo de 2016
 *      Author: David Sisternes Roses
 */

#ifndef DYNAMIXELAX_DEF_H_
#define DYNAMIXELAX_DEF_H_

// Register names according to the datasheet.
// "DYNAMIXEL_AX_12 and Foots Register Map"
// Rev 1.0

//------Control Table Address------------------------------//
//EEPROM AREA
#define P_MODEL_NUMBER_L        0x00    // R
#define P_MODEL_NUMBER_H        0x01    // R
#define P_VERSION               0x02    // R
#define P_ID                    0x03    // R/W
#define P_BAUD_RATE             0x04    // R/W
#define P_RETURN_DELAY_TIME     0x05    // R/W
#define P_CW_ANGLE_LIMIT_L      0x06    // R/W
#define P_CW_ANGLE_LIMIT_H      0x07    // R/W
#define P_CCW_ANGLE_LIMIT_L     0x08    // R/W
#define P_CCW_ANGLE_LIMIT_H     0x09    // R/W
#define P_SYSTEM_DATA2          0x0A    /* Reserved */
#define P_LIMIT_TEMEPERATURE    0x0B    // R/W
#define P_DOWN_LIMIT_VOLTAGE    0x0C    // R/W
#define P_UP_LIMIT_VOLTAGE      0x0D    // R/W
#define P_MAX_TORQUE_L          0x0E    // R/W
#define P_MAX_TORQUE_H          0x0F    // R/W
#define P_RETURN_LEVEL          0x10    // R/W
#define P_ALARM_LED             0x11    // R/W
#define P_ALARM_SHUTDOWN        0x12    // R/W
#define P_OPERATING_MODE        0x13    // R/W
#define P_DOWN_CALIBRATION_L    0x14    // R
#define P_DOWN_CALIBRATION_H    0x15    // R
#define P_UP_CALIBRATION_L      0x16    // R
#define P_UP_CALIBRATION_H      0x17    // R

//RAM AREA
#define P_TORQUE_ENABLE           0x18    // R/W
#define P_LED                     0x19    // R/W
#define P_CW_COMPLIANCE_MARGIN    0x1A    // R/W
#define P_CCW_COMPLIANCE_MARGIN   0x1B    // R/W
#define P_CW_COMPLIANCE_SLOPE     0x1C    // R/W
#define P_CCW_COMPLIANCE_SLOPE    0x1D    // R/W
#define P_GOAL_POSITION_L         0x1E    // R/W
#define P_GOAL_POSITION_H         0x1F    // R/W
#define P_GOAL_SPEED_L            0x20    // R/W
#define P_GOAL_SPEED_H            0x21    // R/W
#define P_TORQUE_LIMIT_L          0x22    // R/W
#define P_TORQUE_LIMIT_H          0x23    // R/W
#define P_PRESENT_POSITION_L      0x24    // R
#define P_PRESENT_POSITION_H      0x25    // R
```

```
#define P_PRESENT_SPEED_L         0x26   // R
#define P_PRESENT_SPEED_H         0x27   // R
#define P_PRESENT_LOAD_L          0x28   // R
#define P_PRESENT_LOAD_H          0x29   // R
#define P_PRESENT_VOLTAGE         0x2A   // R
#define P_PRESENT_TEMPERATURE     0x2B   // R
#define P_REGISTERED_INSTRUCTION 0x2C   // R/W
#define P_PAUSE_TIME              0x2D   /* Reserved */
#define P_MOVING                  0x2E   // R
#define P_LOCK                    0x2F   // R/W
#define P_PUNCH_L                 0x30   // R/W
#define P_PUNCH_H                 0x31   // R/W

//INSTRUCTION
#define INST_PING              0x01
#define INST_READ              0x02
#define INST_WRITE             0X03
#define INST_REG_WRITE         0x04
#define INST_ACTION            0x05
#define INST_RESET             0x06
#define INST_DIGITAL_RESET     0x07  /* No se sabe si funciona */
#define INST_SYSTEM_READ       0x0C  /* No se sabe si funciona */
#define INST_SYSTEM_WRITE      0x0D  /* No se sabe si funciona */
#define INST_SYNC_WRITE        0x83
#define INST_SYNC_REG_WRITE    0x84  /* No se sabe si funciona */

//NUM_SERVOMOTORS
#define BROADCASTING_ID   0xfe
#define S_Dyn0            0x00
#define S_Dyn1            0x01
#define S_Dyn2            0x02
#define S_Dyn3            0X03
#define S_Dyn4            0x04
#define S_Dyn5            0x05
#define S_Dyn6            0x06
#define S_Dyn7            0x07
#define S_Dyn8            0X08
#define S_Dyn9            0x09
#define S_Dyn10           0x0a
#define S_Dyn11           0x0b
#define S_Dyn12           0x0c
#define S_Dyn13           0x0d
#define S_Dyn14           0x0e
#define S_Dyn15           0X0f
#define S_Dyn16           0x10
#define S_Dyn17           0x11
#define S_Dyn18           0x12
#define S_Dyn19                0x13
#define S_Dyn20           0x14
// To be continued...


//OTHERS
#define Automatic      0x00

#define WHEEL          0x00
#define SERVO          0x01

#define NoNe           0x00
#define READ           0x01
```

```
#define ALL             0x02
#define OFF             0x00
#define ON              0x01

#define LEFT            0x00
#define RIGHT           0x01

#define HEX             0x00
#define DEGREES         0x01
#define RPM             0x02
#define PERCENT         0x03

#define CW              0x00
#define CCW             0x01

#define Input_Voltage_Error 0x01
#define Angle_Limit_Error   0x02
#define Overheating_Error   0x04
#define Range_Error         0x08
#define CheckSum_Error      0x10
#define Overload_Error      0x20
#define Instruction_Error   0x40
#define No_Response_Error   0x80
#define Read_Error          4096

//Control Table Planta del Pie
//Foots
#define Left_Foot       0x55
#define Right_Foot      0x56

#define P_PRESENT_DISTANCE_SHARP_L      0x00
#define P_PRESENT_DISTANCE_SHARP_H      0x01
#define P_PRESENT_PRESSURE_SENSOR_1_L   0x02
#define P_PRESENT_PRESSURE_SENSOR_1_H   0x03
#define P_PRESENT_PRESSURE_SENSOR_2_L   0x04
#define P_PRESENT_PRESSURE_SENSOR_2_H   0x05
#define P_PRESENT_PRESSURE_SENSOR_3_L   0x06
#define P_PRESENT_PRESSURE_SENSOR_3_H   0x07
#define P_PRESENT_PRESSURE_SENSOR_4_L   0x08
#define P_PRESENT_PRESSURE_SENSOR_4_H   0x09
#define P_PRESENT_CM_X_L                0x0A
#define P_PRESENT_CM_X_H                0x0B
#define P_PRESENT_CM_Y_L                0x0C
#define P_PRESENT_CM_Y_H                0x0D

//Sensores Predefinidos
#define PRESSURE_SENSOR_1               0x01
#define PRESSURE_SENSOR_2               0x02
#define PRESSURE_SENSOR_3               0x03
#define PRESSURE_SENSOR_4               0x04

#define Axis_X                          0x01
#define Axis_Y                          0x02

#endif /* DYNAMIXELAX_DEF_H_ */
```

4.2 Archivo "BioloidPosDef.h"

```
/*
 *  BioloidPosDef.h
 *
 *  Created on: 3 de Agosto de 2016
 *  Author: David Sisternes Roses
 */
#ifndef BIOLOIDPOS_DEF_H_
#define BIOLOIDPOS_DEF_H_

/*
 * Declaraciones de identificadores, posiciones y velocidades para
 * el desarroyo de movimientos en robot Bioloid.
*/

/* Declaración de los identificadores de cada servomotor dynamixel */
// uint8_t Dynamixels[19] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,'}'};

/* Declaración de las configuraciones para el Margim y Slope (Dureza de los
Dynamixels)*/
const uint8_t ComplianceCW_M[18]={1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
const uint8_t ComplianceCCW_M[18]={1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
const uint8_t
ComplianceCW_S[18]={127,127,32,32,32,32,32,32,32,32,32,32,32,32,64,64,64,64};
const uint8_t
ComplianceCCW_S[18]={127,127,32,32,32,32,32,32,32,32,32,32,32,32,64,64,64,64};

/* Declaración posición Ready del robot Bioloid */
const uint16_t Robot_ready_pos[1][18] =
{{235,788,279,744,462,561,358,666,507,516,341,682,240,783,647,376,507,516}}; //Valores
en hexadecimal.
const uint16_t Robot_ready_pos_deg[1][18] =
{{68,230,81,217,135,164,105,195,149,151,99,199,70,229,190,110,149,151}}; //Valores en
grados.

/* Declaración velocidad Ready del robot Bioloid */
const uint16_t Robot_ready_vel_rpm[1][18] =
{{40,40,40,40,40,40,40,40,40,40,40,40,40,40,40,40,40,40}};
const uint16_t Robot_ready_vel_hex[1][18] = {{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}};

/* Declaración posiciones para movimientos en robot Bioloid */

const uint16_t FWALK0[7][18]=
{{235,788,279,744,462,561,358,666,513,522,342,681,241,781,646,377,513,522},
{235,788,279,744,462,561,358,666,519,528,344,678,245,776,644,380,519,528},
{235,788,279,744,462,561,358,666,524,533,346,675,250,770,642,383,524,533},
{235,788,279,744,462,561,358,666,528,537,349,672,256,763,639,386,528,537},
{235,788,279,744,462,561,358,666,521,544,352,679,262,778,636,379,531,541},
{235,788,279,744,462,561,358,666,503,555,354,696,266,811,634,362,533,544},
{235,788,279,744,462,561,358,666,494,560,355,703,268,826,633,355,534,546}};

const uint16_t FWALK1[7][18]=
{{233,786,279,744,462,561,358,666,503,555,353,725,267,807,641,386,533,544},
{248,801,279,744,462,561,358,666,521,544,357,718,264,765,648,421,531,541},
{263,816,279,744,462,561,358,666,528,537,355,713,259,744,652,437,528,537},
{276,829,279,744,462,561,358,666,524,533,352,718,252,751,655,434,524,533},
{287,840,279,744,462,561,358,666,519,528,353,720,248,760,660,427,519,528},
{295,848,279,744,462,561,358,666,513,522,357,719,246,769,666,418,513,522},
{301,854,279,744,462,561,358,666,507,516,364,716,248,775,671,409,507,516}};
```

```
const uint16_t FWALK2[7][18]=
{{300,853,279,744,462,561,358,666,501,510,363,721,254,777,675,402,501,510},
{303,856,279,744,462,561,358,666,496,505,372,715,262,775,675,398,496,505},
{302,855,279,744,462,561,358,666,490,504,373,709,262,771,677,395,491,500},
{298,851,279,744,462,561,358,666,482,510,361,702,245,766,682,394,486,496},
{290,843,279,744,462,561,358,666,473,519,341,696,222,761,685,393,482,493},
{280,833,279,744,462,561,358,666,467,527,319,691,203,757,681,392,480,491},
{268,821,279,744,462,561,358,666,465,530,299,688,196,756,669,390,479,490}};

const uint16_t FWALK3[7][18]=
{{254,807,279,744,462,561,358,666,467,527,287,686,203,757,649,387,480,491},
{239,792,279,744,462,561,358,666,473,519,283,685,222,761,627,382,482,493},
{223,776,279,744,462,561,358,666,482,510,286,684,245,766,607,376,486,496},
{209,762,279,744,462,561,358,666,490,504,291,683,262,771,595,369,491,500},
{195,748,279,744,462,561,358,666,496,505,293,680,262,775,596,363,496,505},
{184,737,279,744,462,561,358,666,501,510,293,676,254,777,605,357,501,510},
{175,728,279,744,462,561,358,666,507,516,297,669,248,775,614,352,507,516}};

const uint16_t FWALK4[7][18]=
{{169,722,279,744,462,561,358,666,513,522,302,660,246,769,621,348,513,522},
{166,719,279,744,462,561,358,666,518,527,308,651,248,761,625,348,518,527},
{167,720,279,744,462,561,358,666,519,533,314,650,252,761,628,346,523,532},
{171,724,279,744,462,561,358,666,513,541,321,662,257,778,629,341,527,537},
{179,732,279,744,462,561,358,666,504,550,327,682,262,801,630,338,530,541},
{189,742,279,744,462,561,358,666,496,556,332,704,266,820,631,342,532,543},
{201,754,279,744,462,561,358,666,493,558,335,724,267,827,633,354,533,544}};

const uint16_t FWALK5[7][18]=
{{215,768,279,744,462,561,358,666,496,556,337,736,266,820,636,374,532,543},
{230,783,279,744,462,561,358,666,504,550,338,740,262,801,641,396,530,541},
{246,799,279,744,462,561,358,666,513,541,339,737,257,778,647,416,527,537},
{260,813,279,744,462,561,358,666,519,533,340,732,252,761,654,428,523,532},
{274,827,279,744,462,561,358,666,518,527,343,730,248,761,660,427,518,527},
{285,838,279,744,462,561,358,666,513,522,347,730,246,769,666,418,513,522},
{294,847,279,744,462,561,358,666,507,516,354,726,248,775,671,409,507,516}};
const uint16_t FWALK6[7][18]=
{{303,856,279,744,462,561,358,666,501,510,373,711,254,777,674,402,501,510},
{301,854,279,744,462,561,358,666,495,504,382,705,263,775,675,398,495,504},
{297,850,279,744,462,561,358,666,490,499,389,700,272,771,673,397,490,499},
{289,842,279,744,462,561,358,666,486,495,392,697,279,764,668,400,486,495},
{278,831,279,744,462,561,358,666,479,502,376,693,258,759,673,402,482,492},
{265,818,279,744,462,561,358,666,468,520,341,686,216,756,680,398,479,490},
{251,804,279,744,462,561,358,666,463,529,310,678,197,755,668,390,477,489}};
const uint16_t FWALK7[7][18]=
{{235,788,279,744,462,561,358,666,468,520,327,669,212,757,661,389,479,490},
{235,788,279,744,462,561,358,666,479,502,344,671,245,761,644,387,482,492},
{235,788,279,744,462,561,358,666,486,495,351,674,260,767,637,384,486,495},
{235,788,279,744,462,561,358,666,490,499,348,677,253,773,640,381,490,499},
{235,788,279,744,462,561,358,666,495,504,345,679,247,778,643,379,495,504},
{235,788,279,744,462,561,358,666,501,510,342,681,242,782,646,377,501,510},
{235,788,279,744,462,561,358,666,507,516,341,682,240,783,647,376,507,516}};
const uint16_t F_S_L1[7][18]=
{{235,788,279,744,462,561,358,666,513,522,342,681,241,781,646,377,513,522},
{235,788,279,744,462,561,358,666,519,528,344,678,245,776,644,380,519,528},
{235,788,279,744,462,561,358,666,524,533,346,675,250,770,642,383,524,533},
{235,788,279,744,462,561,358,666,528,537,349,672,256,763,639,386,528,537},
{235,788,279,744,462,561,358,666,521,544,352,679,262,778,636,379,531,541},
{235,788,279,744,462,561,358,666,503,555,354,696,266,811,634,362,533,544},
{235,788,279,744,462,561,358,666,494,560,355,703,268,826,633,355,534,546}};
```

```
const uint16_t F_S_L2[7][18]=
{{235,788,279,744,462,561,358,666,503,555,353,725,267,807,641,386,533,544},
{248,801,279,744,462,561,358,666,521,544,357,718,264,765,648,421,531,541},
{263,816,279,744,462,561,358,666,528,537,355,713,259,744,652,437,528,537},
{276,829,279,744,462,561,358,666,524,533,352,718,252,751,655,434,524,533},
{287,840,279,744,462,561,358,666,519,528,353,720,248,760,660,427,519,528},
{295,848,279,744,462,561,358,666,513,522,357,719,246,769,666,418,513,522},
{301,854,279,744,462,561,358,666,507,516,364,716,248,775,671,409,507,516}};

const uint16_t F_S_R1[7][18]=
{{235,788,279,744,462,561,358,666,501,510,342,681,242,782,646,377,501,510},
{235,788,279,744,462,561,358,666,495,504,345,679,247,778,643,379,495,504},
{235,788,279,744,462,561,358,666,490,499,348,677,253,773,640,381,490,499},
{235,788,279,744,462,561,358,666,486,495,351,674,260,767,637,384,486,495},
{235,788,279,744,462,561,358,666,479,502,344,671,245,761,644,387,482,492},
{235,788,279,744,462,561,358,666,468,520,327,669,212,757,661,389,479,490},
{235,788,279,744,462,561,358,666,463,529,320,668,197,755,668,390,477,489}};

const uint16_t F_S_R2[7][18]=
{{236,789,279,744,462,561,358,666,468,520,298,670,216,756,637,382,479,490},
{221,774,279,744,462,561,358,666,479,502,305,666,258,759,602,375,482,492},
{206,759,279,744,462,561,358,666,486,495,310,668,279,764,586,371,486,495},
{193,746,279,744,462,561,358,666,490,499,305,671,272,771,589,368,490,499},
{182,735,279,744,462,561,358,666,495,504,303,670,263,775,596,363,495,504},
{174,727,279,744,462,561,358,666,501,510,304,666,254,777,605,357,501,510},
{168,721,279,744,462,561,358,666,507,516,307,659,248,775,614,352,507,516}};
const uint16_t F_M_R1[7][18]=
{{300,853,279,744,462,561,358,666,501,510,363,721,254,777,675,402,501,510},
{303,856,279,744,462,561,358,666,496,505,372,715,262,775,675,398,496,505},
{302,855,279,744,462,561,358,666,490,504,373,709,262,771,677,395,491,500},
{298,851,279,744,462,561,358,666,482,510,361,702,245,766,682,394,486,496},
{290,843,279,744,462,561,358,666,473,519,341,696,222,761,685,393,482,493},
{280,833,279,744,462,561,358,666,467,527,319,691,203,757,681,392,480,491},
{268,821,279,744,462,561,358,666,465,530,299,688,196,756,669,390,479,490}};
const uint16_t F_M_R2[7][18]=
{{254,807,279,744,462,561,358,666,467,527,287,686,203,757,649,387,480,491},
{239,792,279,744,462,561,358,666,473,519,283,685,222,761,627,382,482,493},
{223,776,279,744,462,561,358,666,482,510,286,684,245,766,607,376,486,496},
{209,762,279,744,462,561,358,666,490,504,291,683,262,771,595,369,491,500},
{195,748,279,744,462,561,358,666,496,505,293,680,262,775,596,363,496,505},
{184,737,279,744,462,561,358,666,501,510,293,676,254,777,605,357,501,510},
{175,728,279,744,462,561,358,666,507,516,297,669,248,775,614,352,507,516}};

const uint16_t F_M_L1[7][18]=
{{169,722,279,744,462,561,358,666,513,522,302,660,246,769,621,348,513,522},
{166,719,279,744,462,561,358,666,518,527,308,651,248,761,625,348,518,527},
{167,720,279,744,462,561,358,666,519,533,314,650,252,761,628,346,523,532},
{171,724,279,744,462,561,358,666,513,541,321,662,257,778,629,341,527,537},
{179,732,279,744,462,561,358,666,504,550,327,682,262,801,630,338,530,541},
{189,742,279,744,462,561,358,666,496,556,332,704,266,820,631,342,532,543},
{201,754,279,744,462,561,358,666,493,558,335,724,267,827,633,354,533,544}};
const uint16_t F_M_L2[7][18]=
{{215,768,279,744,462,561,358,666,496,556,337,736,266,820,636,374,532,543},
{230,783,279,744,462,561,358,666,504,550,338,740,262,801,641,396,530,541},
{246,799,279,744,462,561,358,666,513,541,339,737,257,778,647,416,527,537},
{260,813,279,744,462,561,358,666,519,533,340,732,252,761,654,428,523,532},
{274,827,279,744,462,561,358,666,518,527,343,730,248,761,660,427,518,527},
{285,838,279,744,462,561,358,666,513,522,347,730,246,769,666,418,513,522},
{294,847,279,744,462,561,358,666,507,516,354,726,248,775,671,409,507,516}};
```

```
const uint16_t F_E_L1[7][18]=
{{166,719,279,744,462,561,358,666,513,522,312,650,246,769,621,349,513,522},
{168,721,279,744,462,561,358,666,519,528,318,641,248,760,625,348,519,528},
{172,725,279,744,462,561,358,666,524,533,323,634,252,751,626,350,524,533},
{180,733,279,744,462,561,358,666,528,537,326,631,259,744,623,355,528,537},
{191,744,279,744,462,561,358,666,521,544,330,647,264,765,621,350,531,541},
{204,757,279,744,462,561,358,666,503,555,337,682,267,807,625,343,533,544},
{218,771,279,744,462,561,358,666,949,560,345,713,268,826,633,355,534,546}};

const uint16_t F_E_L2[7][18]=
{{235,788,279,744,462,561,358,666,503,555,354,696,266,811,634,362,533,544},
{235,788,279,744,462,561,358,666,521,544,352,679,262,778,636,379,531,541},
{235,788,279,744,462,561,358,666,528,537,349,672,256,763,639,386,528,537},
{235,788,279,744,462,561,358,666,524,533,346,675,250,770,642,383,524,533},
{235,788,279,744,462,561,358,666,519,528,344,678,245,776,644,380,519,528},
{235,788,279,744,462,561,358,666,513,522,342,681,241,781,646,377,513,522},
{235,788,279,744,462,561,358,666,507,516,341,682,240,783,647,376,507,516}};

const uint16_t F_E_R1[7][18]=
{{303,856,279,744,462,561,358,666,501,510,373,711,254,777,674,402,501,510},
{301,854,279,744,462,561,358,666,495,504,382,705,263,775,675,398,495,504},
{297,850,279,744,462,561,358,666,490,499,389,700,272,771,673,397,490,499},
{289,842,279,744,462,561,358,666,486,495,392,697,279,764,668,400,486,495},
{278,831,279,744,462,561,358,666,479,502,376,693,258,759,673,402,482,495},
{265,818,279,744,462,561,358,666,468,520,341,686,216,756,680,398,479,490},
{251,804,279,744,462,561,358,666,463,529,310,678,197,755,668,390,477,489}};

const uint16_t F_E_R2[7][18]=
{{235,788,279,744,462,561,358,666,468,520,327,669,212,757,661,389,479,490},
{235,788,279,744,462,561,358,666,479,502,344,671,245,761,644,387,482,492},
{235,788,279,744,462,561,358,666,486,495,351,674,260,767,637,384,486,495},
{235,788,279,744,462,561,358,666,490,499,348,677,253,773,640,381,490,499},
{235,788,279,744,462,561,358,666,495,504,345,679,247,778,643,379,495,504},
{235,788,279,744,462,561,358,666,501,510,342,681,242,782,646,377,501,510},
{235,788,279,744,462,561,358,666,507,516,341,682,240,783,647,376,507,516}};

const uint16_t
Manotazos0[1][18]={{234,788,279,744,461,561,358,666,507,516,311,704,123,899,732,278,507
,516}};
const uint16_t Manotazos1[2][18]=
{{520,784,180,744,496,560,357,666,507,516,311,704,121,899,731,277,507,516},
{195,527,275,853,485,529,357,666,507,516,311,704,121,899,731,277,507,516}};
const uint16_t VEL_Manotazos[4][18]=
{{1,1,1,1,1,1,1,1,1,1,52,40,206,204,150,172,1,1},
{630,5,130,1,31,1,1,1,1,1,1,1,1,1,1,1,1,1},
{630,566,130,240,62,1,1,1,1,1,1,1,1,1,1,1,1,1},
{714,566,300,240,24,68,1,1,1,1,1,1,1,1,1,1,1,1}};

#endif /* END BIOLOIDPOS_DEF_H_ */
```

5. TEST DE VALIDACIÓN

5.1 Archivo “test1.cpp”

```
//============================================================================
// Name        : test1.cpp
// Author      : David Sisternes Roses
// Version     : 1.0
// Copyright   : Free GreenTeam
// Description : Lectura posiciones en IMU e escritura en servos dynamixel
//============================================================================

#include <iostream>
#include <stdio.h>
#include <cstdint>
#include "BlackDynamixel.h"
#include "InstrucDynamixel.h"
#include "UtilDynamixel.h"
#include "DynamixelAXDef.h"
#include "BlackSensors.h"
#include "BioloidMotions.h"
#include "BlackLib.h"
#include "BlackADC.h"
#include "MPU9150.h"
/* Puerto de comunicación */
//#define           DEVICE_PORT          "/dev/ttyUSB0"
#define             DEVICE_PORT          "/dev/ttyO4"

/* Velocidad comunicación, bps. */
//#define         BAUD_RATE             19200
//#define         BAUD_RATE             57600
 #define          BAUD_RATE             115200
//#define         BAUD_RATE             1000000  //No funciona

using namespace std;
using namespace BlackLib;
using namespace exploringBB;

int y=0,y_1=0;
float accx,accy,accz;
int main() {

    i2cName iN = BlackLib::I2C_1;
      i2cAddress add = BlackLib::IMU;
      BlackLib::BlackI2C i2cBus(iN,add );
      BlackLib::MPU9150 imu(&i2cBus);

      i2cBus.open(BlackLib::ReadWrite | BlackLib::NonBlock);

      if(!i2cBus.isOpen()){
            cout << "Cannot open I2C_"<< iN << " with address 0x" << hex <<
i2cBus.getDeviceAddress()<< hex << endl;
            exit(-1);
      }
      i2cBus.printInfo();
      cout << "Get Connection: 0x" << hex << (int)imu.getConnection()  << endl;
      usleep(2000000);
      imu.initialize();
```

```
      imu.loadOffsetRegisters("accOffsets.txt");

      printf("\n_____Prueba IMU con servos dynamixel_____\n");

    Open_Connection(DEVICE_PORT,BAUD_RATE); /* Abrimos la el puerto especificado */


  while(1){

        uint8_t parameters[DATA_SIZE]="";

        accx=imu.getAccelerationX();
        accy=imu.getAccelerationY();

        printf("\n>ACC--> X: %f",accx);
        y = (int)((512*accx)+512);
        printf("\n>ACC--> X  escalado: %d",y);
        printf("\n>ACC--> X  en hexa:  %x %x",(char)0xff&(y>>8),(char)0xff&y);


        printf("\n>ACC--> Y: %f",accy);
        y_1 = (int)((512*accy)+512);
        printf("\n>ACC--> Y  escalado: %d",y_1);
        printf("\n>ACC--> Y  en hexa:  %x %x",(char)0xff&(y_1>>8),(char)0xff&y_1);

        usleep(500000);
        parameters[0]= P_GOAL_POSITION_L; //Adress
        parameters[1]= (char)0xff&y;
        parameters[2]= (char)0xff&(y>>8);
        parameters[3]= '\n';       //Si se quiere calcular la longitud automáticamente
se tiene que poner este parametro al final
                                   //En caso contrario poner SOLO el numero de
parámetros.
        Write_Servo(S_Dyn3,Automatic,INST_REG_WRITE,parameters);

     parameters[0]= P_GOAL_POSITION_L; //Adress
     parameters[1]= (char)0xff&y_1;
     parameters[2]= (char)0xff&(y_1>>8);
     parameters[3]= '\n';
     Write_Servo(S_Dyn5,Automatic,INST_REG_WRITE,parameters);

     usleep(1000000);

     Action_Servo();

   }
  return 0; /* FIN */
}
```

5.2 Archivo “test2.cpp”

```
//============================================================================
// Name        : test2.cpp
// Author      : David Sisternes Roses
// Version     : 1.0
// Copyright   : FreeGreenTeam
// Description : Aplicación para prueva brazos robot y capa abstracción
//               movimientos
//============================================================================
#include <iostream>
#include <stdio.h>
#include <cstdint>
#include "BlackDynamixel.h"
#include "InstrucDynamixel.h"
#include "UtilDynamixel.h"
#include "DynamixelAXDef.h"
#include "BlackSensors.h"
#include "BioloidMotions.h"

/* Puerto de comunicación */
//#define           DEVICE_PORT          "/dev/ttyUSB0"
#define             DEVICE_PORT          "/dev/ttyO4"
/* Velocidad comunicación, bps. */
//#define         BAUD_RATE             19200
//#define         BAUD_RATE             57600
 #define          BAUD_RATE             115200
//#define         BAUD_RATE             1000000  //No funciona
using namespace std;

const uint16_t Pos_init[1][6] = {{512,512,512,512,512,512}};
uint8_t Dyn[7] = {1,2,3,4,5,6,'}'};   //((((((RECUERDA!!! Cambiar el NUM_DYNAMIXEL DE 18
a 6 en UtilDynamixel.h)))))
uint32_t usec=10000;
uint16_t pos1=0,pos2=0,pos3=0;

int main() {

      printf("\n_____API  para la prueba del brazo del Robot_____\n");

      Open_Connection(DEVICE_PORT,BAUD_RATE); /* Abrimos la el puerto especificado */

      set_PosSync(Dyn,Pos_init,0,HEX); /* Se establece la configuración inicial */
      set_MaxTorque(BROADCASTING_ID,1023,HEX);

      set_HoldingTorque(1,ON);
      set_HoldingTorque(3,ON);
      set_HoldingTorque(5,ON);

      set_HoldingTorque(2,OFF);
    set_HoldingTorque(4,OFF);
    set_HoldingTorque(6,OFF);


      while(true){

            pos1 = readPos(2,HEX);
            pos2 = readPos(4,HEX);
            pos3 = readPos(6,HEX);
```

```
            usleep(usec);
            set_Pos(1,pos1,HEX);
            set_Pos(3,pos2,HEX);
            set_Pos(5,pos3,HEX);
            usleep(usec);
      }

      Close_Connection();
      return 0;
}
```

5.3 Archivo “test3.cpp”

```
//============================================================================
// Name        : test3.cpp
// Author      : David Sisternes Roses
// Version     : 1.0
// Copyright   : FreeGreenTeam
// Description : Lectura Distancia con sensro Sharp
//============================================================================

#include <iostream>
#include <stdio.h>
#include <cstdint>
#include "BlackDynamixel.h"
#include "InstrucDynamixel.h"
#include "UtilDynamixel.h"
#include "DynamixelAXDef.h"
#include "BlackSensors.h"
#include "BioloidMotions.h"

using namespace std;
uint32_t usec=10000;

int main() {

      printf("\n_____Lectura Distancia con Sensor Sharp_____\n");

      while(true){
            printf("Distancia leída por sensro Sharp: %u", readSharp(0));
            usleep(usec*10);
      }
      return 0;

}
```

5.4 Archivo “test4.cpp”

```
//============================================================================
// Name        : test4.cpp
// Author      : David Sisternes Roses
// Version     : 1.0
// Copyright   : FreeGreenTeam
// Description : test para la validación del sensor Gyrosensor
//============================================================================

#include <iostream>
#include <stdio.h>
#include <cstdint>
```

```
#include "BlackDynamixel.h"
#include "InstrucDynamixel.h"
#include "UtilDynamixel.h"
#include "DynamixelAXDef.h"
#include "BlackSensors.h"
#include "BioloidMotions.h"

using namespace std;
uint32_t usec=10000;

int main() {

      printf("\n_____Prueba Gyrosensor_____\n");

      while(true){
       printf("Velocidad Gyrosensor Ejex: %u",readGyro(0));
       usleep(usec*10);
      }
      return 0; }
```

5.5 Archivo “test5.cpp”

```
//============================================================================
// Name        : test5.cpp
// Author      : David Sisternes Roses
// Version     : 1.0
// Copyright   : FreeGreenTeam
// Description : test para la validación del sensor HC_SR04
//============================================================================

#include <iostream>
#include <stdio.h>
#include <cstdint>
#include "BlackDynamixel.h"
#include "InstrucDynamixel.h"
#include "UtilDynamixel.h"
#include "DynamixelAXDef.h"
#include "BlackSensors.h"
#include "BioloidMotions.h"

using namespace std;
uint32_t usec=10000;

int main() {

      printf("\n_____Lectura Distancia con Sensor HC_SR04_____\n");

      while(true){
            printf("Distancia leída por sensro HC_SR04: %u", readHC_SR04());
            usleep(usec*10);
      }
      return 0;
}
```

5.6 Archivo “test6.cpp”

```
//============================================================================
// Name        : test6.cpp
// Author      : David Sisternes Roses
// Version     : 1.0
// Copyright   : FreeGreenTeam
// Description : test realizado para la prueba de movimiento del robot
//               Bioloid con sensor Sharp
//============================================================================

#include <iostream>
#include <stdio.h>
#include <cstdint>
#include "BlackDynamixel.h"
#include "InstrucDynamixel.h"
#include "UtilDynamixel.h"
#include "DynamixelAXDef.h"
#include "BlackSensors.h"
#include "BioloidMotions.h"

/* Puerto de comunicación */
//#define           DEVICE_PORT          "/dev/ttyUSB0"
#define             DEVICE_PORT          "/dev/ttyO4"
/* Velocidad comunicación, bps. */
//#define         BAUD_RATE             19200
//#define         BAUD_RATE             57600
 #define          BAUD_RATE             115200
//#define         BAUD_RATE             1000000  //No funciona

using namespace std;

uint32_t usec=10000;
uint8_t dist=0;

int main() {
      printf("\n_____Movimientos en Robot Bioloid_____\n");

      Open_Connection(DEVICE_PORT,BAUD_RATE); /* Abrimos la el puerto especificado */

      robotReady();   /* Se pone el Robot en posición de preparado */
      usleep(usec);
      while(true){

            dist= readSharp(0); /* Lectura del senor Sharp conectado en AI0*/

            if(dist > 15 ){ /* Si no se detecta nada */
                  walkStepL(); /* Caminamos */
            }else
              {
                  Manotazos(); /* Manotazos */
              }

            printf("\ndis: %d",dist); /* Se muestra distancia por pantalla */
            usleep(200000);
      }

      Close_Connection();
      return 0;
}
```

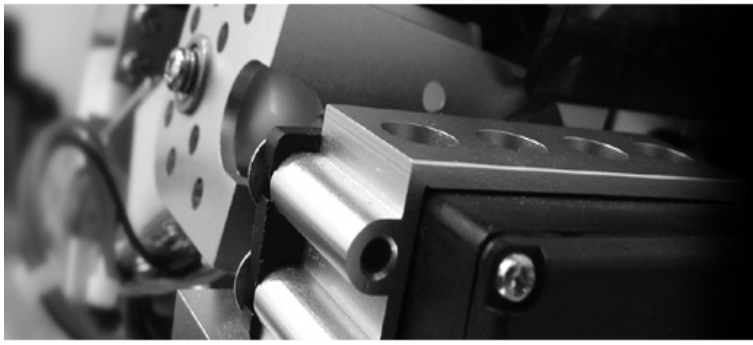
2.2. Documentos Anexados

Autor: Sisternes Roses, David.

Tutor: Blanes Noguera, Juan Francisco.

Titulación: Máster Universitario en Automática e Informática Industrial.

User's Manual 2006-06-14



Closer to Real, ROBOTIS

Dynamixel AX-12

Contents

1. Dynamixel AX-12

1-1. Overview and Characteristics of AX-12

Dynamixel AX-12 The Dynamixel series robot actuator is a smart, modular actuator that incorporates a gear reducer, a precision DC motor and a control circuitry with networking functionality, all in a single package. Despite its compact size, it can produce high torque and is made with high quality materials to provide the necessary strength and structural resilience to withstand large external forces. It also has the ability to detect and act upon internal conditions such as changes in internal temperature or supply voltage. The Dynamixel series robot actuator has many advantages over similar products.

Precision Control Position and speed can be controlled with a resolution of 1024 steps.

Compliance Driving The degree of compliance can be adjusted and specified in controlling position.

Feedback Feedback for angular position, angular velocity, and load torque are available.

Alarm System The Dynamixel series robot actuator can alert the user when parameters deviate from user defined ranges (e.g. internal temperature, torque, voltage, etc) and can also handle the problem automatically (e.g. torque off)

Communication Wiring is easy with daisy chain connection, and it support communication speeds up to 1M BPS.

Distributed Control Position, velocity, compliance, and torque can be set with a single command packet, thus enabling the main processor to control many Dynamixel units even with very few resources.

Engineering Plastic The main body of the unit is made with high quality engineering plastic which enables it to handle high torque loads.

Axis Bearing A bearing is used at the final axis to ensure no efficiency degradation with high external loads.

Status LED The LED can indicate the error status to the user.

Frames A hinge frame and a side mount frame are included.

1-2. Main Specifications

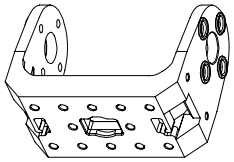
	AX-12	
Weight (g)	**55**	
Gear Reduction Ratio	**1/254**	
Input Voltage (V)	**at 7V**	**at 10V**
Final Max Holding Torque(kgf.cm)	**12**	**16.5**
Sec/60degree	**0.269**	**0.196**

Resolution	0.35°
Operating Angle	300°, Endless Turn
Voltage	7V~10V (Recommended voltage: 9.6V)
Max. Current	900mA
Operate Temperature	-5℃ ~ +85℃
Command Signal	Digital Packet
Protocol Type	Half duplex Asynchronous Serial Communication (8bit,1stop,No Parity)
Link (Physical)	TTL Level Multi Drop (daisy chain type Connector)
ID	254 ID (0~253)
Communication Speed	7343bps ~ 1 Mbps
Feedback	Position, Temperature, Load, Input Voltage, etc.
Material	Engineering Plastic

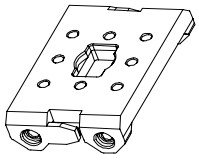
2. Dynamixel Operation

2-1. **Mechanical Assembly**

Frames Provided The two frames provided with AX-12 are shown below.

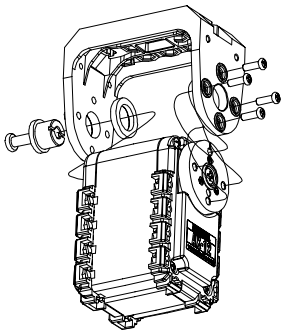


OF-12SH

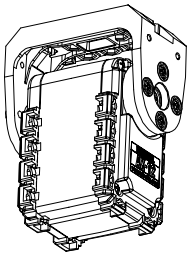


OF-12S

OF-12SH Installation The OF-12SH (hinge frame) can be installed on the AX-12 as the following.



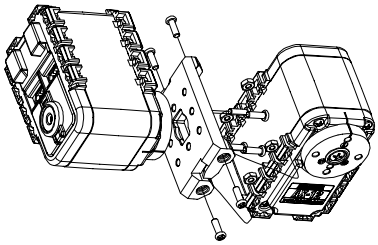
Exploded view



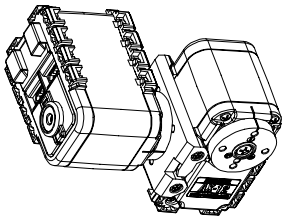
Assembled

OF-12S Installation The OF-12S (side mount frame) can be installed on the AX-12 as the following. The OF-12S can be mounted on any of the three faces (left, right, or under side) of the AX-12 body as needed.

Horn2Body

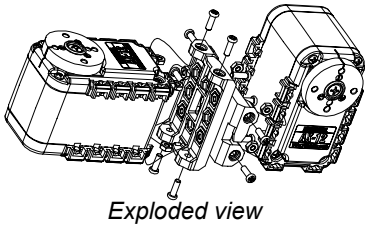


Exploded view

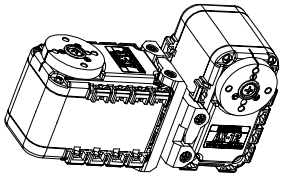


Assembled

Body2Body



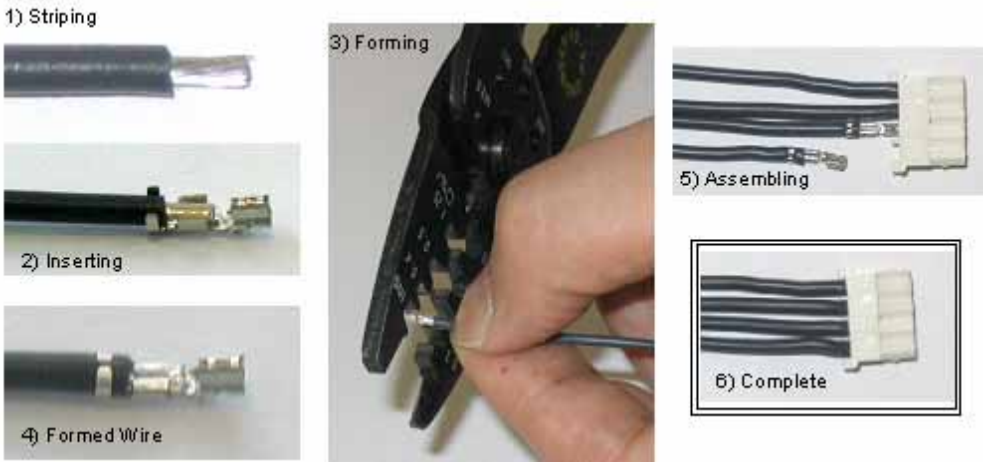
Exploded view



Assembled

2-2 . Connector Assembly

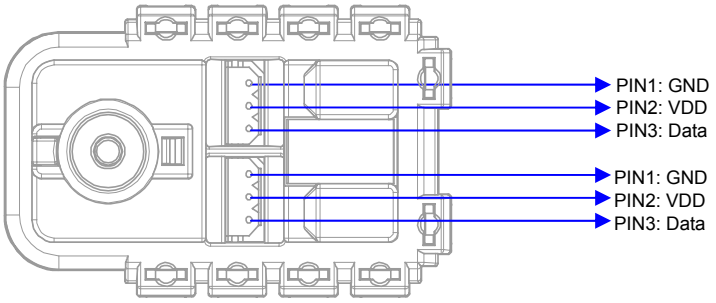
Assemble the connectors as shown below. Attach the wires to the terminals using the correct crimping tool. If you do not have access to a crimping tool, solder the terminals to the wires to ensure that they do not become loose during operation.



2-3. Dynamixel Wiring

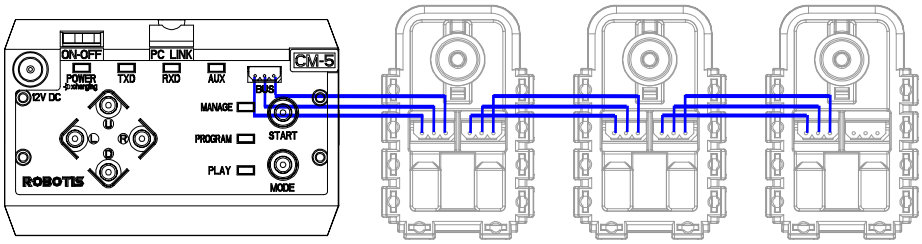
Pin Assignment

The connector pin assignments are as the following. The two connectors on the Dynamixel are connected pin to pin, thus the AX-12 can be operated with only one connector attached.



Wiring

Connect the AX-2 actuators pin to pin as shown below. Many AX-12 actuators can be controlled with a single bus in this manner.



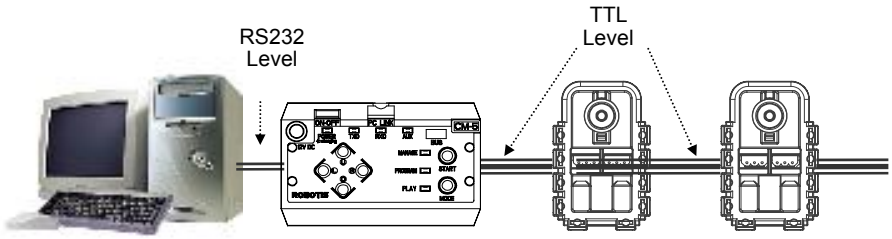
Control Box “CM-5”

Main Controller

To operate the Dynamixel actuators, the main controller must support TTL level half duplex UART. A proprietary controller can be used, but the use of the Dynamixel controller CM-5 is recommended.

PC LINK

A PC can be used to control the Dynamixel via the CM-5 controller.



PC CM-5 Dynamixels

bioloid

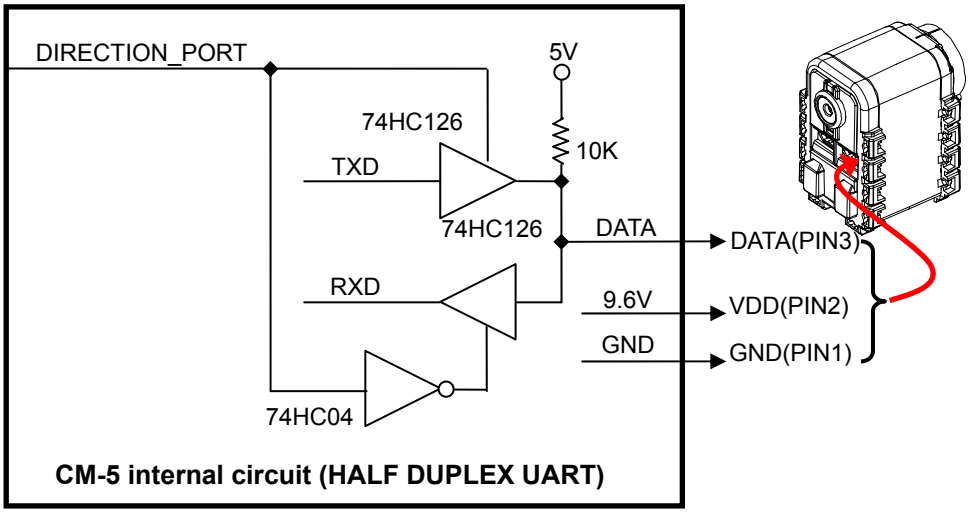
A robot can be built using only the CM-5 controller and a number of AX-12 actuators. An edutainment robotic kit named “Bioloid” is available which is based on the CM-5 controller and the AX-12 actuators.

An example of a robot built with Bioloid

For details, please refer to the Bioloid manual.

Connection to UART

To control the Dynamixel actuators, the main controller needs to convert its UART signals to the half duplex type. The recommended circuit diagram for this is shown below.



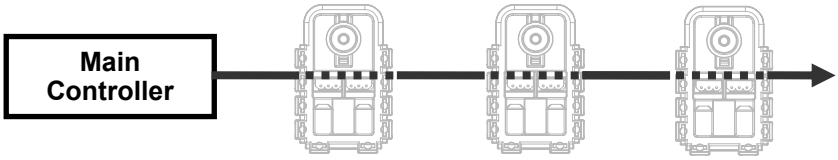
CM-5 internal circuit (HALF DUPLEX UART)

The power is supplied to the Dynamixel actuator from the main controller through Pin 1 and Pin 2 of the Molex3P connector. (The circuit shown above is presented only to explain the use of half duplex UART. The CM-5 controller already has the above circuitry built in, thus the Dynamixel actuators can be directly connected to it)

The direction of data signals on the TTL level TxD and RxD depends on the DIRECTION_PORT level as the following.

• When the DIRECTION_PORT level is High: the signal TxD is output as Data
• When the DIRECTION_PORT level is Low: the signal Data is input as RxD

Half Duplex UART A multi-drop method of connecting multiple Dynamixel actuators to a single node is possible by using the half duplex UART. Thus a protocol that does not allow multiple transmissions at the same time should be maintained when controlling the Dynamixel actuators.



[Multi Drop Link]

Caution Please ensure that the pin assignments are correct when connecting the Dynamixel actuators. Check the current consumption when powering on. The current consumption of a single Dynamixel actuator unit in standby mode should be no larger than 50mA

Connection Status Verification

When power is applied to the Dynamixel actuator, the LED blinks twice to confirm its connection.

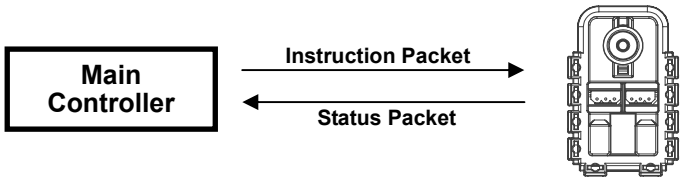
Inspection If the above operation was not successful, then check the connector pin assignment and the voltage/current limit of the power supply.

3. Communication Protocol

3-1. Communication Overview

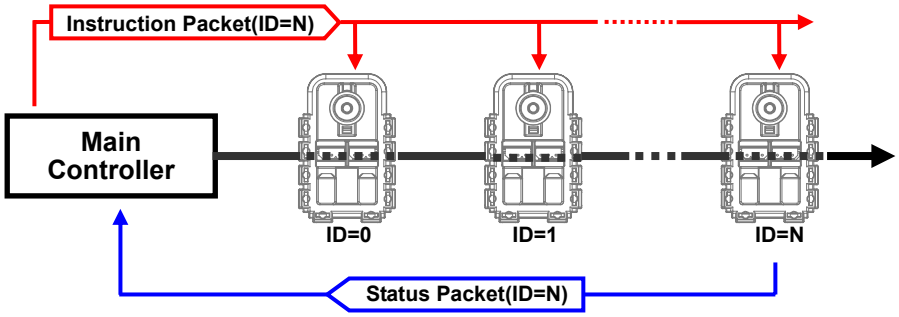
Packet

The main controller communicates with the Dynamixel units by sending and receiving data packets. There are two types of packets; the “Instruction Packet” (sent from the main controller to the Dynamixel actuators) and the “Status Packet” (sent from the Dynamixel actuators to the main controller.)



Communication

For the system connection below, if the main controller sends an instruction packet with the ID set to N, only the Dynamixel unit with this ID value will return its respective status packet and perform the required instruction.



Unique ID

If multiple Dynamixel units have the same ID value, multiple packets sent simultaneously collide, resulting in communication problems. Thus, it is imperative that no Dynamixel units share the same ID in a network node.

Protocol

The Dynamixel actuators communicate through asynchronous serial communication with 8 bit, 1 stop bit and no parity.

3-2. Instruction Packet

The Instruction Packet is the packet sent by the main controller to the Dynamixel units to send commands. The structure of the Instruction Packet is as the following.

Instruction Packet | OXFF | 0XFF | ID | LENGTH | INSTRUCTION | PARAMETER1 | PARAMETER N | CHECK SUM |

The meanings of each packet byte definition are as the following.

0XFF 0XFF The two 0XFF bytes indicate the start of an incoming packet.

ID The unique ID of a Dynamixel unit. There are 254 available ID values, ranging from 0X00 to 0XFD.

Broadcasting ID ID 0XFE is the Broadcasting ID which indicates all of the connected Dynamixel units. Packets sent with this ID apply to all Dynamixel units on the network. Thus packets sent with a broadcasting ID will not return any status packets.

LENGTH The length of the packet where its value is “Number of parameters (N) + 2”

INSTRUCTION The instruction for the Dynamixel actuator to perform.

PARAMETER0 N Used if there is additional information needed to be sent other than the instruction itself.

CHECK SUM The computation method for the ‘Check Sum’ is as the following.

Check Sum = ~ (ID + Length + Instruction + Parameter1 + ... Parameter N)
If the calculated value is larger than 255, the lower byte is defined as the checksum value.
~ represents the NOT logic operation.

3-3. Status Packet(Return Packet)

The Status Packet is the response packet from the Dynamixel units to the Main Controller after receiving an instruction packet. The structure of the status packet is as the following.

| OXFF | 0XFF | ID | LENGTH | ERROR | PARAMETER1 | PARAMETER2 | PARAMETER N | CHECK SUM |

The meanings of each packet byte definition are as the following.

0XFF 0XFF The two 0XFF bytes indicate the start of the packet.

ID The unique ID of the Dynamixel unit returning the packet. The initial value is set to 1.

LENGTH The length of the packet where its value is “Number of parameters (N) + 2”

ERROR The byte representing errors sent from the Dynamixel unit. The meaning of each bit is as the following.

Bit	Name	Details
Bit 7	0	-
Bit 6	Instruction Error	Set to 1 if an undefined instruction is sent or an action instruction is sent without a Reg_Write instruction.
Bit 5	Overload Error	Set to 1 if the specified maximum torque can't control the applied load.
Bit 4	Checksum Error	Set to 1 if the checksum of the instruction packet is incorrect.
Bit 3	Range Error	Set to 1 if the instruction sent is out of the defined range.
Bit 2	Overheating Error	Set to 1 if the internal temperature of the Dynamixel unit is above the operating temperature range as defined in the control table.
Bit 1	Angle Limit Error	Set as 1 if the Goal Position is set outside of the range between CW Angle Limit and CCW Angle Limit.
Bit 0	Input Voltage Error	Set to 1 if the voltage is out of the operating voltage range as defined in the control table.

PARAMETER0 N Used if additional information is needed.

CHECK SUM The computation method for the ‘Check Sum’ is as the following.

Check Sum = ~ (ID + Length + Instruction + Parameter1 + ... Parameter N)

If the calculated value is larger than 255, the lower byte is defined as the checksum value. ~ represents the NOT logic operation.

3-4. Control Table

Address	Item	Access	Initial Value
0(0X00)	Model Number(L)	RD	12(0x0C)
1(0X01)	Model Number(H)	RD	0(0x00)
2(0X02)	Version of Firmware	RD	?
3(0X03)	ID	RD,WR	1(0x01)
4(0X04)	Baud Rate	RD,WR	1(0x01)
5(0X05)	Return Delay Time	RD,WR	250(0xFA)
6(0X06)	CW Angle Limit(L)	RD,WR	0(0x00)
7(0X07)	CW Angle Limit(H)	RD,WR	0(0x00)
8(0X08)	CCW Angle Limit(L)	RD,WR	255(0xFF)
9(0X09)	CCW Angle Limit(H)	RD,WR	3(0x03)
10(0x0A)	(Reserved)	-	0(0x00)
11(0X0B)	the Highest Limit Temperature	RD,WR	85(0x55)
12(0X0C)	the Lowest Limit Voltage	RD,WR	60(0X3C)
13(0X0D)	the Highest Limit Voltage	RD,WR	190(0xBE)
14(0X0E)	Max Torque(L)	RD,WR	255(0XFF)
15(0X0F)	Max Torque(H)	RD,WR	3(0x03)
16(0X10)	Status Return Level	RD,WR	2(0x02)
17(0X11)	Alarm LED	RD,WR	4(0x04)
18(0X12)	Alarm Shutdown	RD,WR	4(0x04)
19(0X13)	(Reserved)	RD,WR	0(0x00)
20(0X14)	Down Calibration(L)	RD	?
21(0X15)	Down Calibration(H)	RD	?
22(0X16)	Up Calibration(L)	RD	?
23(0X17)	Up Calibration(H)	RD	?
24(0X18)	Torque Enable	RD,WR	0(0x00)
25(0X19)	LED	RD,WR	0(0x00)
26(0X1A)	CW Compliance Margin	RD,WR	0(0x00)
27(0X1B)	CCW Compliance Margin	RD,WR	0(0x00)
28(0X1C)	CW Compliance Slope	RD,WR	32(0x20)
29(0X1D)	CCW Compliance Slope	RD,WR	32(0x20)
30(0X1E)	Goal Position(L)	RD,WR	[Addr36]value
31(0X1F)	Goal Position(H)	RD,WR	[Addr37]value
32(0X20)	Moving Speed(L)	RD,WR	0
33(0X21)	Moving Speed(H)	RD,WR	0
34(0X22)	Torque Limit(L)	RD,WR	[Addr14] value
35(0X23)	Torque Limit(H)	RD,WR	[Addr15] value
36(0X24)	Present Position(L)	RD	?
37(0X25)	Present Position(H)	RD	?
38(0X26)	Present Speed(L)	RD	?
39(0X27)	Present Speed(H)	RD	?
40(0X28)	Present Load(L)	RD	?
41(0X29)	Present Load(H)	RD	?
42(0X2A)	Present Voltage	RD	?
43(0X2B)	Present Temperature	RD	?
44(0X2C)	Registered Instruction	RD,WR	0(0x00)
45(0X2D)	(Reserved)	-	0(0x00)
46[0x2E)	Moving	RD	0(0x00)
47[0x2F)	Lock	RD,WR	0(0x00)
48[0x30)	Punch(L)	RD,WR	32(0x20)
49[0x31)	Punch(H)	RD,WR	0(0x00)

EEPROM Area

RAM Area

Control Table The Control Table contains information on the status and operation of the Dynamixel actuator. The Dynamixel actuator is operated by writing values to its control table and its status is checked by reading values off its control table.

RAM and EEPROM The data values for the RAM area will be set to the default initial values whenever the power is turned on. However, the data values for the EEPROM area are non-volatile and will still remain even after the power is turned off.

Initial Value The Initial Value column on the right side of the control table shows the Factory Default Values for the case of EEPROM area data, and shows the initial value when the power is turned on for the case of RAM area data.

The following explains the meaning of data stored in each of the addresses in the control table.

Address 0x00,0x01 **Model Number**. For AX-12, this value is 0X000C (12).

Address 0x02 **Firmware Version**.

Address 0x03 **ID**. The unique ID number assigned to each Dynamixel actuators for identifying them. Different IDs are required for each Dynamixel actuators that are on the same network.

Address 0x04 **Baud Rate.** Determines the communication speed. The computation is done by the following formula.

Speed (BPS) = 2000000 / (Address4 + 1)

Data Value for each Major Baud Rate

Adress4	Hex	Set BPS	Goal BPS	Error
1	0X01	1000000.0	1000000.0	0.000%
3	0X03	500000.0	500000.0	0.000%
4	0X04	400000.0	400000.0	0.000%
7	0X07	250000.0	250000.0	0.000%
9	0X09	200000.0	200000.0	0.000%
16	0X10	117647.1	115200.0	-2.124%
34	0X22	57142.9	57600.0	0.794%
103	0X67	19230.8	19200.0	-0.160%
207	0XCF	9615.4	9600.0	-0.160%

Note A maximum Baud Rate error of 3% is within the tolerance of UART communication.

Caution The initial value of Baudrate is set to 1(1000000bps)

Address 0x05 **Return Delay Time.** The time it takes for the Status Packet to return after the Instruction Packet is sent. The delay time is given by 2uSec * Address5 value.

Address 0x06,0x07,0x08,0x09

Operating Angle Limit. Sets the Dynamixel actuator's operating angle range. The Goal Position needs to be within the range of: CW Angle Limit <= Goal Position <= CCW Angle Limit. An Angle Limit Error will occur if the Goal Position is set outside this range set by the operating angle limits.

Address 0x0B **the Highest Limit Temperature.** The upper limit of the Dynamixel actuator's operating temperature. If the internal temperature of the Dynamixel actuator gets higher than this value, the Over Heating Error Bit (Bit 2 of the Status Packet) will return the value 1, and an alarm will be set by Address 17, 18. The values are in Degrees Celsius.

Address 0x0C,0x0D **the Lowest (Highest) Limit Voltage.** The upper and lower limits of the Dynamixel actuator's operating voltage. If the present voltage (Address 42) is out of the specified range, a Voltage Range Error Bit (Bit 0 of the Status Packet) will return the value 1, and an alarm will be set by Address 17, 18. The values are 10 times the actual voltage value. For example, if the Address 12 value is 80, then the lower voltage limit is set to 8V.

Address 0x0E,0x0F, 0x22,0x23

Max Torque. The maximum torque output for the Dynamixel actuator. When this value is set to 0, the Dynamixel actuator enters the Free Run mode. There are two locations where this maximum torque limit is defined; in the EEPROM (Address 0X0E, 0x0F) and in the RAM (Address 0x22, 0x23). When the power is turned on, the maximum torque limit value defined in the EEPROM is copied to the location in the RAM. The torque of the Dynamixel actuator is limited by the values located in the RAM (Address 0x22, 0x23).

Address 0X10 **Status Return Level.** Determines whether the Dynamixel actuator will return a Status Packet after receiving an Instruction Packet.

Address16	Returning the Status Packet
0	Do not respond to any instructions
1	Respond only to READ_DATA instructions
2	Respond to all instructions

In the case of an instruction which uses the Broadcast ID (0XFE) the Status Packet will not be returned regardless of the Address 0x10 value.

Address 0X11 **Alarm LED.** If the corresponding Bit is set to 1, the LED blinks when an Error occurs.

Bit	Function
Bit 7	0
Bit 6	If set to 1, the LED blinks when an Instruction Error occurs
Bit 5	If set to 1, the LED blinks when an Overload Error occurs
Bit 4	If set to 1, the LED blinks when a Checksum Error occurs
Bit 3	If set to 1, the LED blinks when a Range Error occurs
Bit 2	If set to 1, the LED blinks when an Overheating Error occurs
Bit 1	If set to 1, the LED blinks when an Angle Limit Error occurs
Bit 0	If set to 1, the LED blinks when an Input Voltage Error occurs

This function operates following the “OR” logical operation of all bits. For example, if the value is set to 0X05, the LED will blink when an Input Voltage Error occurs or when an Overheating Error occurs. Upon returning to a normal condition from an error state, the LED stops blinking after 2 seconds.

Address 0X12 **Alarm Shutdown.** If the corresponding Bit is set to a 1, the Dynamixel actuator’s torque will be turned off when an error occurs.

Bit	Function
Bit 7	0
Bit 6	If set to 1, torque off when an Instruction Error occurs
Bit 5	If set to 1, torque off when an Overload Error occurs
Bit 4	If set to 1, torque off when a Checksum Error occurs
Bit 3	If set to 1, torque off when a Range Error occurs
Bit 2	If set to 1, torque off when an Overheating Error occurs
Bit 1	If set to 1, torque off when an Angle Limit Error occurs
Bit 0	If set to 1, torque off when an Input Voltage Error occurs

This function operates following the “OR” logical operation of all bits. However, unlike the Alarm LED, after returning to a normal condition, it maintains the torque off status. To recover, the Torque Enable (Address0X18) needs to be reset to 1.

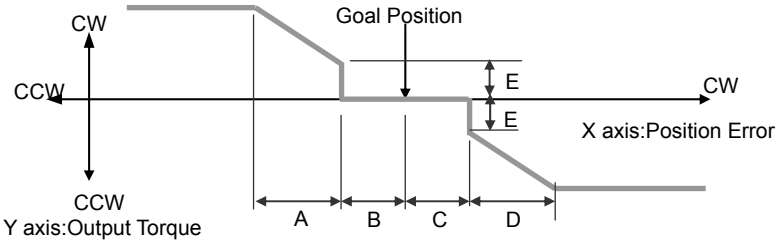
Address 0x14~0x17 **Calibration.** Data used for compensating for the differences between the potentiometers used in the Dynamixel units. The user cannot change this data.

The following (from Address 0x18) is in the RAM area.

Address 0x18 **Torque Enable.** When the power is first turned on, the Dynamixel actuator enters the Torque Free Run condition (zero torque). Setting the value in Address 0x18 to 1 enables the torque.

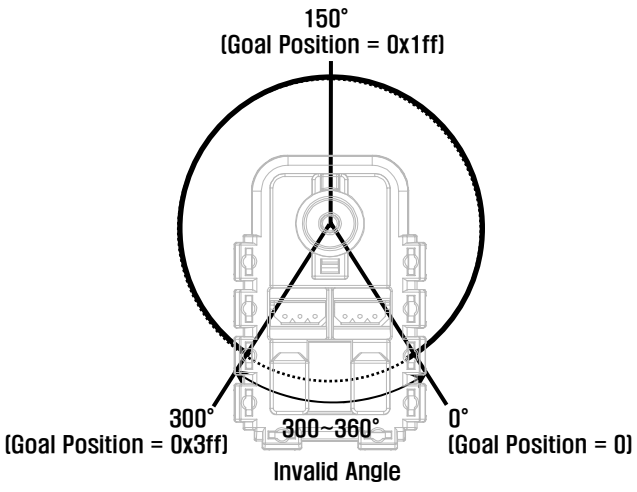
Address 0x19 **LED**. The LED turns on when set to 1 and turns off if set to 0.

Address 0x1A~0x1D **Compliance Margin and Slope.** The compliance of the Dynamixel actuator is defined by setting the compliance Margin and Slope. This feature can be utilized for absorbing shocks at the output shaft. The following graph shows how each compliance value (length of A, B, C & D) is defined by the Position Error and applied torque.



A : CCW Compliance Slope(Address0x1D)
B : CCW Compliance Margin(Address0x1B)
C : CW Compliance Margin(Address0x1A)
D : CW Compliance Slope (Address0x1C)
E : Punch(Address0x30,31)

Address 0X1E,0x1F **Goal Position** Requested angular position for the Dynamixel actuator output to move to. Setting this value to 0x3ff moves the output shaft to the position at 300°.



Address 0x20,0x21 **Moving Speed.** Sets the angular velocity of the output moving to the Goal Position. Setting this value to its maximum value of 0x3ff moves the output with an angular velocity of 114 RPM, provided that there is enough power supplied (The lowest velocity is when this value is set to 1. When set to 0, the velocity is the largest possible for the supplied voltage, e.g. no velocity control is applied.)

Address 0x24,0x25 **Present Position.** Current angular position of the Dynamixel actuator output.

Address 0x26,0x27 **Present Speed.** Current angular velocity of the Dynamixel actuator output.

Address 0x28,0x29 **Present Load.** The magnitude of the load on the operating Dynamixel actuator. Bit 10 is the direction of the load.

BIT	15~11	10	9	8	7	6	5	4	3	2	1	0
Value	0	Load Direction	Load Value									

Load Direction = 0 : CCW Load, Load Direction = 1: CW Load

Address 0x2A **Present Voltage.** The voltage currently applied to the Dynamixel actuator. The value is 10 times the actual voltage. For example, 10V is represented as 100 (0x64).

Address 0x2B **Present Temperature.** The internal temperature of the Dynamixel actuator in Degrees Celsius.

Address 0x2C **Registered Instruction.** Set to 1 when an instruction is assigned by the REG_WRITE command. Set to 0 after it completes the assigned instruction by the Action command.

Address 0x2E **Moving.** Set to 1 when the Dynamixel actuator is moving by its own power.

Address 0x2F **Lock.** If set to 1, only Address 0x18 to 0x23 can be written to and other areas cannot. Once locked, it can only be unlocked by turning the power off.

Address 0x30,0x31 **Punch.** The minimum current supplied to the motor during operation. The initial value is set to 0x20 and its maximum value is 0x3ff.

Endless Turn If both values for the CW Angle Limit and the CCW Angle Limit are set to 0, an Endless Turn mode can be implemented by setting the Goal Speed. This feature can be used for implementing a continuously rotating wheel.

Goal Speed Setting

BIT	15~11	10	9	8	7	6	5	4	3	2	1	0
Value	0	Turn Direction	Speed Value									

Turn Direction = 0 : CCW Direction Turn, Load Direction = 1: CW Direction Turn

Range

Each data has a valid minimum and maximum values. Write instructions made outside of these valid ranges will return an error. The following table summarizes the data range for each register. 16 bit data registers are indicated with two bytes (L) and (H). Both bytes need to be written at the same time as one instruction packet.

Write Address	Writing Item	Length (bytes)	Min	Max
3(0X03)	ID	1	0	253(0xfd)
4(0X04)	Baud Rate	1	0	254(0xfe)
5(0X05)	Return Delay Time	1	0	254(0xfe)
6(0X06)	CW Angle Limit	2	0	1023(0x3ff)
8(0X08)	CCW Angle Limit	2	0	1023(0x3ff)
11(0X0B)	the Highest Limit Temperature	1	0	150(0x96)
12(0X0C)	the Lowest Limit Voltage	1	50(0x32)	250(0xfa)
13(0X0D)	the Highest Limit Voltage	1	50(0x32)	250(0xfa)
14(0X0E)	Max Torque	2	0	1023(0x3ff)
16(0X10)	Status Return Level	1	0	2
17(0X11)	Alarm LED	1	0	127(0x7f)
18(0X12)	Alarm Shutdown	1	0	127(0x7f)
19(0X13)	(Reserved)	1	0	1
24(0X18)	Torque Enable	1	0	1
25(0X19)	LED	1	0	1
26(0X1A)	CW Compliance Margin	1	0	254(0xfe)
27(0X1B)	CCW Compliance Margin	1	0	254(0xfe)
28(0X1C)	CW Compliance Slope	1	1	254(0xfe)
29(0X1D)	CCW Compliance Slope	1	1	254(0xfe)
30(0X1E)	Goal Position	2	0	1023(0x3ff)
32(0X20)	Moving Speed	2	0	1023(0x3ff)
34(0X22)	Torque Limit	2	0	1023(0x3ff)
44(0X2C)	Registered Instruction	1	0	1
47(0X2F)	Lock	1	1	1
48(0X30)	Punch	2	0	1023(0x3ff)

[Control Table Data Range and Length for Writing]

4. Instruction Set and Examples

The following Instructions are available.

Instruction	Function	Value	Number of Parameter
PING	No action. Used for obtaining a Status Packet	0x01	0
READ DATA	Reading values in the Control Table	0x02	2
WRITE DATA	Writing values to the Control Table	0x03	2 ~
REG WRITE	Similar to WRITE_DATA, but stays in standby mode until the ACION instruction is given	0x04	2 ~
ACTION	Triggers the action registered by the REG_WRITE instruction	0x05	0
RESET	Changes the control table values of the Dynamixel actuator to the Factory Default Value settings	0x06	0
SYNC WRITE	Used for controlling many Dynamixel actuators at the same time	0x83	4~

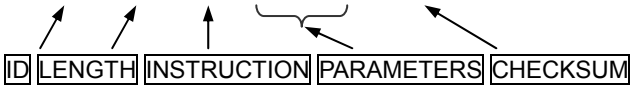
4-1. WRITE_DATA

Function To write data into the control table of the Dynamixel actuator
Length N+3 (N is the number of data to be written)
Instruction 0X03
Parameter1 Starting address of the location where the data is to be written
Parameter2 1st data to be written
Parameter3 2nd data to be written
Parameter N+1 Nth data to be written

Example 1 **Setting the ID of a connected Dynamixel actuator to 1**

Write 1 to address 3 of the control table. The ID is transmitted using the Broadcasting ID (0xFE).

Instruction Packet : 0XFF 0XFF 0XFE 0X04 0X03 0X03 0X01 0XF6`



Because it was transmitted with a Broadcast ID (0XFE), no status packets are returned.

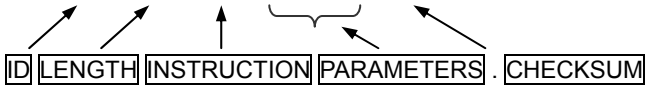
4-2. READ_DATA

Function	Read data from the control table of a Dynamixel actuator
Length	0X04
Instruction	0X02
Parameter1	Starting address of the location where the data is to be read
Parameter2	Length of the data to be read

Example 2 **Reading the internal temperature of the Dynamixel actuator with an ID of 1**

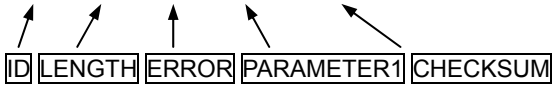
Read 1 byte from address 0x2B of the control table.

Instruction Packet : 0XFF 0XFF 0X01 0X04 0X02 0X2B 0X01 0XCC`



The returned Status Packet will be as the following.

Status Packet : 0XFF 0XFF 0X01 0X03 0X00 0X20 0XDB



The data read is 0x20. Thus the current internal temperature of the Dynamixel actuator is approximately 32°C (0X20).

4-3. REG_WRITE과 ACTION

4-3-1. REG_WRITE

Function The REG_WRITE instruction is similar to the WRITE_DATA instruction, but the

execution timing is different. When the Instruction Packet is received the values are stored in the Buffer and the Write instruction is under a standby status. At this time, the Registered Instruction register (Address 0x2C) is set to 1. After the Action Instruction Packet is received, the registered Write instruction is finally executed.

Length	N+3 (N is the number of data to be written)
Instruction	0X04
Parameter1	Starting address of the location where the data is to be written
Parameter2	1st data to be written
Parameter3	2nd data to be written
Parameter N+1	Nth data to be written

4-3-2. ACTION

Function	Triggers the action registered by the REG_WRITE instruction
Length	0X02
Instruction	0X05
Parameter	NONE

The ACTION instruction is useful when multiple Dynamixel actuators need to move simultaneously. When controlling multiple Dynamixel actuator units, slight time delays can occur between the 1st and last units to receive an instruction. The Dynamixel actuator handles this problem by using the ACTION instruction.

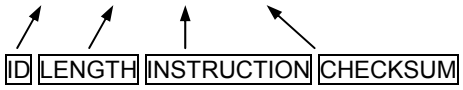
Broadcasting The Broadcast ID (0XFE) is used when sending ACTION instructions to more than two Dynamixel actuators. Note that no packets are returned by this operation.

4-4. PING

Function	Does not command any operations. Used for requesting a status packet or to check the existence of a Dynamixel actuator with a specific ID.
Length	0X02
Instruction	0X01
Parameter	NONE

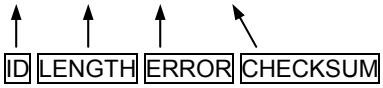
Example 3 **Obtaining the status packet of the Dynamixel actuator with an ID of 1**

Instruction Packet : 0XFF 0XFF 0X01 0X02 0X01 0XFB`



The returned Status Packet is as the following

Status Packet : 0XFF 0XFF 0X01 0X02 0X00 0XFC



Regardless of whether the Broadcasting ID is used or the Status Return Level (Address 16) is 0, a Status Packet is always returned by the PING instruction.

4-5. RESET

Function Changes the control table values of the Dynamixel actuator to the Factory Default Value settings

Length 0X02

Instruction 0X06

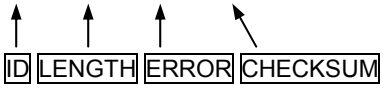
Parameter NONE

Example 4 **Resetting the Dynamixel actuator with an ID of 0**

Instruction Packet : 0XFF 0XFF 0X00 0X02 0X06 0XF7`

The returned Status Packet is as the following

Status Packet : 0XFF 0XFF 0X00 0X02 0X00 0XFD



Note the ID of this Dynamixel actuator is now changed to 1 after the RESET instruction.

4-6. SYNC WRITE

Function	Used for controlling many Dynamixel actuators at the same time. The communication time decreases by the Synch Write instruction since many instructions can be transmitted by a single instruction. However, you can use this instruction only when the lengths and addresses of the control table to be written to are the same. Also, the broadcasting ID needs to be used for transmitting.
ID	0XFE
Length	(L + 1) * N + 4 (L: Data length for each Dynamixel actuator, N: The number of Dynamixel actuators)
Instruction	0X83
Parameter1	Starting address of the location where the data is to be written
Parameter2	The length of the data to be written (L)
Parameter3	The ID of the 1st Dynamixel actuator
Parameter4	The 1st data for the 1st Dynamixel actuator
Parameter5	The 2nd data for the 1st Dynamixel actuator
Parameter L+3	The Lth data for the 1st Dynamixel actuator
Parameter L+4	The ID of the 2nd Dynamixel actuator
Parameter L+5	The 1st data for the 2nd Dynamixel actuator
Parameter L+6	The 2nd data for the 2nd Dynamixel actuator
Parameter 2L+4	The Lth data for the 2nd Dynamixel actuator

Data for the 1st Dynamixel actuator

Data for the 2nd Dynamixel actuator

.

Example 5 **Setting the following positions and velocities for 4 Dynamixel actuators**

Dynamixel actuator with an ID of 0: to position 0X010 with a speed of 0X150
Dynamixel actuator with an ID of 1: to position 0X220 with a speed of 0X360
Dynamixel actuator with an ID of 2: to position 0X030 with a speed of 0X170
Dynamixel actuator with an ID of 0: to position 0X220 with a speed of 0X380

Instruction Packet : 0XFF 0XFF 0XFE 0X18 0X83 0X1E 0X04 0X00 0X10 0X00 0X50 0X01 0X01 0X20 0X02 0X60 0X03 0X02 0X30 0X00 0X70 0X01 0X03 0X20 0X02 0X80 0X03 0X12
No status packets are returned since the Broadcasting ID was used.

5. Example

For the following examples, we assume a Dynamixel actuator with an ID of 1 in Reset status and that the Baud rate is 57142 BPS.

Example 6 **Reading the Model Number and Firmware Version of the Dynamixel actuator with an ID of 1**

Instruction Packet Instruction = READ_DATA, Address = 0x00, Length = 0x03

Communication ->[Dynamixel]:FF FF 01 04 02 00 03 F5 (LEN:008)
<-[Dynamixel]:FF FF 01 05 00 74 00 08 7D (LEN:009)

Status Packet Result Model Number = 116 (0x74) (for the case of DX-116) Firmware Version = 0x08

Example 7 **Changing the ID to 0 for a Dynamixel actuator with an ID of 1**

Instruction Packet Instruction = WRITE_DATA, Address = 0x03, DATA = 0x00

Communication ->[Dynamixel]:FF FF 01 04 03 03 00 F4 (LEN:008)
<-[Dynamixel]:FF FF 01 02 00 FC (LEN:006)

Status Packet Result NO ERROR

Example 8 **Changing the Baud Rate of a Dynamixel actuator to 1M bps**

Instruction Packet Instruction = WRITE_DATA, Address = 0x04, DATA = 0x01

Communication ->[Dynamixel]:FF FF 00 04 03 04 01 F3 (LEN:008)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

Example 9 **Resetting the Return Delay Time to 4 uSec for a Dynamixel actuator with an ID of 0**

A Return Delay Time Value of 1 corresponds to 2uSec.

Instruction Packet Instruction = WRITE_DATA, Address = 0x05, DATA = 0x02

Communication ->[Dynamixel]:FF FF 00 04 03 05 02 F1 (LEN:008)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

It is recommended to set the Return Delay Time to the minimum value allowed by the Main Controller.

Example 10 **Limiting the operating angle range to 0°~150° for a Dynamixel actuator with an ID of 0**

Since the CCW Angle Limit of 0x3ff corresponds to 300°, the angle 150° is represented by the value 0x1ff

Instruction Packet Instruction = WRITE_DATA, Address = 0x08, DATA = 0xff, 0x01

Communication ->[Dynamixel]:FF FF 00 05 03 08 FF 01 EF (LEN:009)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

Example 11 **Resetting the upper limit for the operating temperature to 80°C for a Dynamixel actuator with an ID of 0**

Instruction Packet Instruction = WRITE_DATA, Address = 0x0B, DATA = 0x50

Communication ->[Dynamixel]:FF FF 00 04 03 0B 50 9D (LEN:008)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

Example 12 **Setting the operating voltage to 10V ~ 17V for a Dynamixel actuator with an ID of 0**

10V is represented by 100 (0x64), and 17V by 170 (0xAA).

Instruction Packet Instruction = WRITE_DATA, Address = 0x0C, DATA = 0x64, 0xAA

Communication ->[Dynamixel]:FF FF 00 05 03 0C 64 AA DD (LEN:009)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

Example 13 **Setting the maximum torque to 50% of its maximum possible value for a Dynamixel actuator with an ID of 0**

Set the MAX Torque value located in the ROM area to 0x1ff which is 50% of the maximum value 0x3ff.

Instruction Packet Instruction = WRITE_DATA, Address = 0x0E, DATA = 0xff, 0x01

Communication ->[Dynamixel]:FF FF 00 05 03 0E FF 01 E9 (LEN:009)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR
To verify the effect of the adjusted Max Torque value, the power needs to be turned off and then on.

Example 14 **Set the Dynamixel actuator with an ID of 0 to never return a Status Packet**

Instruction Packet Instruction = WRITE_DATA, Address = 0x10, DATA = 0x00

Communication ->[Dynamixel]:FF FF 00 04 03 10 00 E8 (LEN:008)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR
The Status Packet is not returned starting with the following instruction.

Example 15 **Set the Alarm to blink the LED and Shutdown (Torque off) the actuator when the operating temperature goes over the set limit**

Since the Overheating Error is Bit 2, set the Alarm value to 0x04.

Instruction Packet Instruction = WRITE_DATA, Address = 0x11, DATA = 0x04, 0x04

Communication ->[Dynamixel]:FF FF 00 05 03 11 04 04 DE (LEN:009)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

Example 16 **Turn on the LED and Enable Torque for a Dynamixel actuator with an ID of 0**

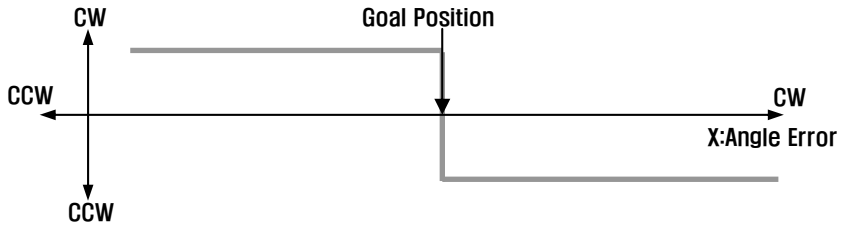
Instruction Packet Instruction = WRITE_DATA, Address = 0x18, DATA = 0x01, 0x01

Communication ->[Dynamixel]:FF FF 00 05 03 18 01 01 DD (LEN:009)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

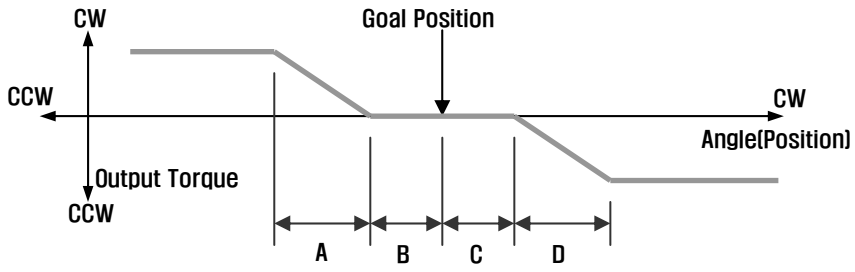
Status Packet Result NO ERROR
You can verify the Torque Enabled status by trying to move the output of the actuator by hand.

Example 17 **Setting the Compliance Margin to 1 and Compliance Slope to 0x40 for a Dynamixel actuator with an ID of 0**

Compliance The Angle Error and Torque Output can be represented with the following graph.



Even if the position deviates a little from the goal position in the CW direction, a large amount of torque is generated in the CCW direction to compensate for this. However, since inertia must be considered, a realistic implementation differs from this approach. Considering this, the given conditions can be represented by the following graph.



A : CCW Compliance Slope (Address0x1D) = 0x40 (about 18.8°)
B : CCW Compliance Margin (Address0x1B) = 0x01 (about 0.29°)
C : CW Compliance Margin (Address0x01A) = 0x01 (about 0.29°)
D : CW Compliance Slope (Address0x1C) = 0x40 (about 18.8°)

Instruction Packet Instruction = WRITE_DATA, Address = 0x1A, DATA = 0x01, 0x01, 0x40, 0x40

Communication ->[Dynamixel]:FF FF 00 07 03 1A 01 01 40 40 59 (LEN:011)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

The Compliance Slope takes effect with discrete steps of 2^n (n is integer). Thus any Compliance value between 0x11 and 0x20 has identical effects.

Example 18 **Position the output of a Dynamixel actuator with an ID of 0 to 180° with an angular velocity of 057RPM**

Set Address 0x1E (Goal Position) to 0x200 and Address 0x20 (Moving Speed) to 0x200.

Instruction Packet Instruction = WRITE_DATA, Address = 0x1E, DATA = 0x00, 0x02, 0x00, 0x02

Communication ->[Dynamixel]:FF FF 00 07 03 1E 00 02 00 02 D3 (LEN:011)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

Example 19 **Position the output of a Dynamixel actuator with an ID of 0 to 0° and Position the output of a Dynamixel actuator with an ID of 1 to 300°, and initiate the movement at the same time.**

If the WRITE_DATA is used, the movement of the two actuators cannot be initiate at the same time, thus the REG_WRITE and ACTION instructions should be used instead.

Instruction Packet ID=0, Instruction = REG_WRITE, Address = 0x1E, DATA = 0x00, 0x00
ID=1, Instruction = REG_WRITE, Address = 0x1E, DATA = 0xff, 0x03
ID=0xfe(Broadcasting ID), Instruction = ACTION,

Communication ->[Dynamixel]:FF FF 00 05 04 1E 00 00 D8 (LEN:009)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)
->[Dynamixel]:FF FF 01 05 04 1E FF 03 D5 (LEN:009)
<-[Dynamixel]:FF FF 01 02 00 FC (LEN:006)
->[Dynamixel]:FF FF FE 02 05 FA (LEN:006)
<-[Dynamixel]: //No return packet against broadcasting ID

Status Packet Result NO ERROR

Example 20 **Lock all addresses except for Address 0x18 ~ Address0x23 for a Dynamixel actuator with an ID of 0**

Set Address 0x2F (Lock) to 1.

Instruction Packet Instruction = WRITE_DATA, Address = 0x2F, DATA = 0x01

Communication
->[Dynamixel]:FF FF 00 04 03 2F 01 C8 (LEN:008)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

Once locked, the only way to unlock it is to remove the power.

If an attempt is made to access any locked data, an error is returned.

->[Dynamixel]:FF FF 00 05 03 30 40 00 87 (LEN:009)
<-[Dynamixel]:FF FF 00 02 08 F5 (LEN:006)

Range Error

Example 21 **Set the minimum power (Punch) to 0x40 for a Dynamixel actuator with an ID of 0**

Instruction Packet Instruction = WRITE_DATA, Address = 0x30, DATA = 0x40, 0x00

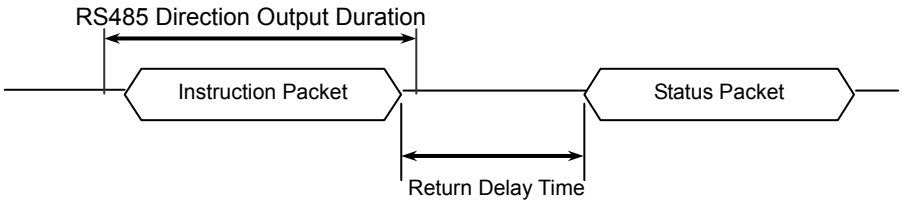
Communication
->[Dynamixel]:FF FF 00 05 03 30 40 00 87 (LEN:009)
<-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)

Status Packet Result NO ERROR

Appendix

Half duplex UART

Half duplex UART is a serial communication protocol where both TxD and RxD cannot be used at the same time. This method is generally used when many devices need to be connected to a single bus. Since more than one device are connected to the same bus, all the other devices need to be in input mode while one device is transmitting. The Main Controller that controllers the Dynamixel actuators sets the communication direction to input mode, and only when it is transmitting an Instruction Packet, it changes the direction to output mode.



Return Delay Time

The time it takes for the Dynamixel actuator to return the Status Packet after receiving an Instruction Packet. The Default Value is 160 uSec and can be changed via the Control Table at Address 5. The Main Controller needs to change the Direction Port to input mode during the Return Delay Time after sending an instruction packet.

Tx,Rx Direction

For Half Duplex UART, the transmission ending timing is important to change the direction to receiving mode. The bit definitions within the register that indicates UART_STATUS are as the following

TXD_BUFFER_READY_BIT: Indicates that the transmission DATA can be loaded into the Buffer. Note that this only means that the SERIAL TX BUFFER is empty, and does not necessarily mean that the all the data transmitted before has left the CPU.

TXD_SHIFT_REGISTER_EMPTY_BIT: Set when all the Transmission Data has completed its transmission and left the CPU.

The TXD_BUFFER_READY_BIT is used when one byte is to be transmitted via the serial communication channel, and an example is shown below.

```
TxDByte(byte bData)
{
    while(!TXD_BUFFER_READY_BIT);    //wait until data can be loaded.
    SerialTxDBuffer = bData;          //data load to TxD buffer
}
```

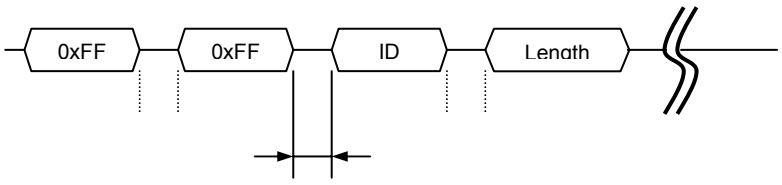
When changing the direction, the TXD_SHIFT_REGISTER_EMPTY_BIT must be checked.

The following is an example program that sends an Instruction Packet.

```
LINE 1    DIRECTION_PORT = TX_DIRECTION;
LINE 2    TxDByte(0xff);
LINE 3    TxDByte(0xff);
LINE 4    TxDByte(bID);
LINE 5    TxDByte(bLength);
LINE 6    TxDByte(bInstruction);
LINE 7    TxDByte(Parameter0);   TxDByte(Parameter1);
LINE 8    DisableInterrupt(); // interrupt should be disable
LINE 9    TxDByte(Checksum);   //last TxD
LINE 10   while(!TXD_SHIFT_REGISTER_EMPTY_BIT); //Wait till last data bit has been sent
LINE 11   DIRECTION_PORT = RX_DIRECTION; //Direction change to RXD
LINE 12   EnableInterrupt(); // enable interrupt again
```

Please note the important lines between LINE 8 and LINE 12. Line 8 is necessary since an interrupt here may cause a delay longer than the return delay time and corruption to the front of the status packet may occur.

Byte to Byte Time The delay time between bytes when sending an instruction packet. If the delay time is over 100ms, then the Dynamixel actuator recognizes this as a communication problem and waits for the next header (0xff 0xff) of a packet again.



Byte To Byte Time

The following is the source code of a program (Example.c) that accesses the Dynamixel actuator using the Atmega 128.

C Language Example : Dinamixel access with Atmega128

```
/*
 * The Example of Dynamixel Evaluation with Atmega128
 * Date : 2005.5.11
 * Author : BS KIM
 */

/*
 * included files
 */
#define ENABLE_BIT_DEFINITIONS
//#include <io.h>
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

#define cbi(REG8,BITNUM) REG8 &= ~(_BV(BITNUM))
#define sbi(REG8,BITNUM) REG8 |= _BV(BITNUM)


typedef unsigned char byte;
typedef unsigned int word;
#define ON 1
#define OFF 0
#define _ON 0
#define _OFF 1


//--- Control Table Address ---
//EEPROM AREA
#define P_MODEL_NUMBER_L      0
#define P_MODOEL_NUMBER_H     1
#define P_VERSION             2
#define P_ID                  3
#define P_BAUD_RATE           4
#define P_RETURN_DELAY_TIME   5
#define P_CW_ANGLE_LIMIT_L    6
#define P_CW_ANGLE_LIMIT_H    7
#define P_CCW_ANGLE_LIMIT_L   8
#define P_CCW_ANGLE_LIMIT_H   9
#define P_SYSTEM_DATA2        10
#define P_LIMIT_TEMPERATURE   11
#define P_DOWN_LIMIT_VOLTAGE  12
#define P_UP_LIMIT_VOLTAGE    13
#define P_MAX_TORQUE_L        14
#define P_MAX_TORQUE_H        15
#define P_RETURN_LEVEL        16
#define P_ALARM_LED           17
#define P_ALARM_SHUTDOWN      18
#define P_OPERATING_MODE      19
#define P_DOWN_CALIBRATION_L  20
#define P_DOWN_CALIBRATION_H  21
#define P_UP_CALIBRATION_L    22
#define P_UP_CALIBRATION_H    23

#define P_TORQUE_ENABLE          (24)
#define P_LED                    (25)
#define P_CW_COMPLIANCE_MARGIN   (26)
#define P_CCW_COMPLIANCE_MARGIN  (27)
#define P_CW_COMPLIANCE_SLOPE    (28)
#define P_CCW_COMPLIANCE_SLOPE   (29)
#define P_GOAL_POSITION_L        (30)
#define P_GOAL_POSITION_H        (31)
#define P_GOAL_SPEED_L           (32)
#define P_GOAL_SPEED_H           (33)
#define P_TORQUE_LIMIT_L         (34)
#define P_TORQUE_LIMIT_H         (35)
#define P_PRESENT_POSITION_L     (36)
#define P_PRESENT_POSITION_H     (37)
#define P_PRESENT_SPEED_L        (38)
#define P_PRESENT_SPEED_H        (39)
#define P_PRESENT_LOAD_L         (40)
#define P_PRESENT_LOAD_H         (41)
#define P_PRESENT_VOLTAGE        (42)
#define P_PRESENT_TEMPERATURE    (43)
#define P_REGISTERED_INSTRUCTION (44)
#define P_PAUSE_TIME             (45)
#define P_MOVING (46)
#define P_LOCK                   (47)
#define P_PUNCH_L                (48)
#define P_PUNCH_H                (49)

//--- Instruction ---
#define INST_PING           0x01
#define INST_READ           0x02
#define INST_WRITE          0x03
#define INST_REG_WRITE      0x04
#define INST_ACTION         0x05
#define INST_RESET          0x06
#define INST_DIGITAL_RESET  0x07
#define INST_SYSTEM_READ    0x0C
#define INST_SYSTEM_WRITE   0x0D
#define INST_SYNC_WRITE     0x83
#define INST_SYNC_REG_WRITE 0x84

#define CLEAR_BUFFER gbRxBufferReadPointer = gbRxBufferWritePointer
#define DEFAULT_RETURN_PACKET_SIZE 6
#define BROADCASTING_ID 0xfe

#define TxD8 TxD81
#define RxD8 RxD81

//Hardware Dependent Item
#define DEFAULT_BAUD_RATE 34   //57600bps at 16MHz

////// For CM-5
#define    RS485_TXD    PORTE    &=    ~_BV(PE3),PORTE    |=    _BV(PE2)
                       //PORT_485_DIRECTION = 1
#define    RS485_RXD    PORTE    &=    ~_BV(PE2),PORTE    |=    _BV(PE3)
                       //PORT_485_DIRECTION = 0
/*
////// For CM-2
#define RS485_TXD PORTE |= _BV(PE2); //_485_DIRECTION = 1
#define RS485_RXD PORTE &= ~_BV(PE2);//PORT_485_DIRECTION = 0
*/
//#define TXDO_FINISH  UCSR0A,6  //This bit is for checking TxD Buffer
                     in CPU is empty or not.
//#define TXD1_FINISH  UCSR1A,6

#define SET_TxD0_FINISH   sbi(UCSR0A,6)
#define RESET_TXD0_FINISH cbi(UCSR0A,6)
#define CHECK_TXD0_FINISH bit_is_set(UCSR0A,6)
#define SET_TxD1_FINISH  sbi(UCSR1A,6)
#define RESET_TXD1_FINISH cbi(UCSR1A,6)
#define CHECK_TXD1_FINISH bit_is_set(UCSR1A,6)

#define RX_INTERRUPT 0x01
#define TX_INTERRUPT 0x02
#define OVERFLOW_INTERRUPT 0x01
#define SERIAL_PORT0 0
#define SERIAL_PORT1 1
#define BIT_RS485_DIRECTION0  0x08  //Port E
#define BIT_RS485_DIRECTION1  0x04  //Port E

#define BIT_ZIGBEE_RESET              PD4  //out : default 1 //PORTD
#define BIT_ENABLE_RXD_LINK_PC        PD5  //out : default 1
#define BIT_ENABLE_RXD_LINK_ZIGBEE    PD6  //out : default 0
#define BIT_LINK_PLUGIN               PD7  //in, no pull up

void TxD81(byte bTxdData);
void TxD80(byte bTxdData);
void TxDString(byte *bData);
void TxD8Hex(byte bSentData);
void TxD32Dec(long lLong);
byte RxD81(void);
void MiliSec(word wDelayTime);
void PortInitialize(void);
void SerialInitialize(byte bPort, byte bBaudrate, byte bInterrupt);
byte TxPacket(byte bID, byte bInstruction, byte bParameterLength);
byte RxPacket(byte bRxLength);
void PrintBuffer(byte *bpPrintBuffer, byte bLength);
```

```
// --- Gloval Variable Number ---
volatile byte gbpRxInterruptBuffer[256];
byte gbpParameter[128];
byte gbRxBufferReadPointer;
byte gbpRxBuffer[128];
byte gbpTxBuffer[128];
volatile byte gbRxBufferWritePointer;

int main(void)
{
  byte bCount,bID, bTxPacketLength,bRxPacketLength;

  PortInitialize(); //Port In/Out Direction Definition
  RS485_RXD; //Set RS485 Direction to Input State.
  SerialInitialize(SERIAL_PORT0,1,RX_INTERRUPT);//RS485
                    Initializing(RxInterrupt)
  SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,0);          //RS232
                    Initializing(None Interrupt)

  gbRxBufferReadPointer  =  gbRxBufferWritePointer  =  0;    //RS485
                    RxBuffer Clearing.

  sei();  //Enable Interrupt -- Compiler Function
  TxDString("¥r¥n   [The   Example   of   Dynamixel   Evaluation   with
                    ATmega128,GCC-AVR]");

//Dynamixel Communication Function Execution Step.
// Step 1. Parameter Setting (gbpParameter[]). In case of no parameter
                    instruction(Ex.  INST_PING),  this  step  is  not
                    needed.
//   Step   2.   TxPacket(ID,INSTRUCTION,LengthOfParameter);   --Total
                    TxPacket Length is returned
// Step 3. RxPacket(ExpectedReturnPacketLength); -- Real RxPacket
                    Length is returned
// Step 4 PrintBuffer(BufferStartPointer,LengthForPrinting);

  bID = 1;
  TxDString("¥r¥n¥n Example 1. Scanning Dynamixels(0~9). -- Any Key to
                    Continue."); RxD8();
  for(bCount = 0; bCount < 0x0A; bCount++)
  {
    bTxPacketLength = TxPacket(bCount,INST_PING,0);
    bRxPacketLength = RxPacket(255);
    TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
    TxDString(", RxD:");    PrintBuffer(gbpRxBuffer,bRxPacketLength);
    if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE)
    {
      TxDString(" Found!! ID:");TxD8Hex(bCount);
      bID = bCount;
    }
  }

  TxDString("¥r¥n¥n Example 2. Read Firmware Version. -- Any Key to
                    Continue."); RxD8();
  gbpParameter[0] = P_VERSION; //Address of Firmware Version
  gbpParameter[1] = 1; //Read Length
  bTxPacketLength = TxPacket(bID,INST_READ,2);
  bRxPacketLength                                                    =
                    RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter
                    [1]);
  TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
  TxDString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);
  if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1])
  {
    TxDString("¥r¥n Return Error      : ");TxD8Hex(gbpRxBuffer[4]);
    TxDString("¥r¥n Firmware Version  : ");TxD8Hex(gbpRxBuffer[5]);
  }

  TxDString("¥r¥n¥n  Example  3.  LED  ON  --  Any  Key  to  Continue.");
                    RxD8();
  gbpParameter[0] = P_LED; //Address of LED
  gbpParameter[1] = 1; //Writing Data
  bTxPacketLength = TxPacket(bID,INST_WRITE,2);
  bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
  TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
  TxDString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);

  TxDString("¥r¥n¥n  Example  4.  LED  OFF  --  Any  Key  to  Continue.");
                    RxD8();
  gbpParameter[0] = P_LED; //Address of LED
  gbpParameter[1] = 0; //Writing Data
  bTxPacketLength = TxPacket(bID,INST_WRITE,2);
  bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
  TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
  TxDString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);

  TxDString("¥r¥n¥n  Example  5.  Read  Control  Table.  --  Any  Key  to
                    Continue."); RxD8();
  gbpParameter[0] = 0; //Reading Address
  gbpParameter[1] = 49; //Read Length
  bTxPacketLength = TxPacket(bID,INST_READ,2);
  bRxPacketLength                                                    =
                    RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter
                    [1]);

  TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
  TxDString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);
  if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1])
  {
    TxDString("¥r¥n");
    for(bCount = 0; bCount < 49; bCount++)
    {
      TxD8('[');TxD8Hex(bCount);TxDString("]:");
                    TxD8Hex(gbpRxBuffer[bCount+5]);TxD8(' ');
    }
  }

  TxDString("¥r¥n¥n Example 6. Go 0x200 with Speed 0x100 -- Any Key to
                    Continue."); RxD8();
  gbpParameter[0] = P_GOAL_POSITION_L; //Address of Firmware Version
  gbpParameter[1] = 0x00; //Writing Data P_GOAL_POSITION_L
  gbpParameter[2] = 0x02; //Writing Data P_GOAL_POSITION_H
  gbpParameter[3] = 0x00; //Writing Data P_GOAL_SPEED_L
  gbpParameter[4] = 0x01; //Writing Data P_GOAL_SPEED_H
  bTxPacketLength = TxPacket(bID,INST_WRITE,5);
  bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
  TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
  TxDString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);

  TxDString("¥r¥n¥n Example 7. Go 0x00 with Speed 0x40 -- Any Key to
                    Continue."); RxD8();
  gbpParameter[0] = P_GOAL_POSITION_L; //Address of Firmware Version
  gbpParameter[1] = 0x00; //Writing Data P_GOAL_POSITION_L
  gbpParameter[2] = 0x00; //Writing Data P_GOAL_POSITION_H
  gbpParameter[3] = 0x40; //Writing Data P_GOAL_SPEED_L
  gbpParameter[4] = 0x00; //Writing Data P_GOAL_SPEED_H
  bTxPacketLength = TxPacket(bID,INST_WRITE,5);
  bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
  TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
  TxDString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);

  TxDString("¥r¥n¥n Example 8. Go 0x3ff with Speed 0x3ff -- Any Key to
                    Continue."); RxD8();
  gbpParameter[0] = P_GOAL_POSITION_L; //Address of Firmware Version
  gbpParameter[1] = 0xff; //Writing Data P_GOAL_POSITION_L
  gbpParameter[2] = 0x03; //Writing Data P_GOAL_POSITION_H
  gbpParameter[3] = 0xff; //Writing Data P_GOAL_SPEED_L
  gbpParameter[4] = 0x03; //Writing Data P_GOAL_SPEED_H
  bTxPacketLength = TxPacket(bID,INST_WRITE,5);
  bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
  TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
  TxDString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);

  TxDString("¥r¥n¥n Example 9. Torque Off -- Any Key to Continue.");
                    RxD8();
  gbpParameter[0] = P_TORQUE_ENABLE; //Address of LED
  gbpParameter[1] = 0; //Writing Data
  bTxPacketLength = TxPacket(bID,INST_WRITE,2);
  bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
  TxDString("¥r¥n TxD:"); PrintBuffer(gbpTxBuffer,bTxPacketLength);
  TxDString("¥r¥n RxD:"); PrintBuffer(gbpRxBuffer,bRxPacketLength);

  TxDString("¥r¥n¥n End. Push reset button for repeat");
```

```
  while(1);
}

void PortInitialize(void)
{
  DDRA = DDRB = DDRC = DDRD = DDRE = DDRF = 0;  //Set all port to
                    input direction first.
  PORTB = PORTC = PORTD = PORTE = PORTF = PORTG = 0x00; //PortData
                    initialize to 0
  cbi(SFIOR,2); //All Port Pull Up ready
  DDRE  |=  (BIT_RS485_DIRECTION0|BIT_RS485_DIRECTION1);  //set  output
                    the bit RS485direction

  DDRD                                                          |=
                    (BIT_ZIGBEE_RESET|BIT_ENABLE_RXD_LINK_PC|BIT_ENA
                    BLE_RXD_LINK_ZIGBEE);

  PORTD &= ~_BV(BIT_LINK_PLUGIN); // no pull up
  PORTD |= _BV(BIT_ZIGBEE_RESET);
  PORTD |= _BV(BIT_ENABLE_RXD_LINK_PC);
  PORTD |= _BV(BIT_ENABLE_RXD_LINK_ZIGBEE);
}

/*
TxPacket() send data to RS485.
TxPacket()  needs  3  parameter;  ID  of  Dynamixel,  Instruction byte,
                    Length of parameters.
TxPacket() return length of Return packet from Dynamixel.
*/
byte TxPacket(byte bID, byte bInstruction, byte bParameterLength)
{
    byte bCount,bCheckSum,bPacketLength;

    gbpTxBuffer[0] = 0xff;
    gbpTxBuffer[1] = 0xff;
    gbpTxBuffer[2] = bID;
    gbpTxBuffer[3]                =                bParameterLength+2;
                    //Length(Paramter,Instruction,Checksum)
    gbpTxBuffer[4] = bInstruction;
    for(bCount = 0; bCount < bParameterLength; bCount++)
    {
        gbpTxBuffer[bCount+5] = gbpParameter[bCount];
    }
    bCheckSum = 0;
    bPacketLength = bParameterLength+4+2;
    for(bCount  =  2;  bCount  <  bPacketLength-1;  bCount++)  //except
                    0xff,checksum
    {
        bCheckSum += gbpTxBuffer[bCount];
    }
    gbpTxBuffer[bCount]  =  ~bCheckSum;  //Writing  Checksum  with  Bit
                    Inversion

    RS485_TXD;
    for(bCount = 0; bCount < bPacketLength; bCount++)
    {
        sbi(UCSR0A,6);//SET_TXD0_FINISH;
        TxD80(gbpTxBuffer[bCount]);
    }
    while(!CHECK_TXD0_FINISH); //Wait until TXD Shift register empty
    RS485_RXD;
    return(bPacketLength);
}

/*
RxPacket() read data from buffer.
RxPacket() need a Parameter; Total length of Return Packet.
RxPacket() return Length of Return Packet.
*/

byte RxPacket(byte bRxPacketLength)
{
#define RX_TIMEOUT_COUNT2   3000L
#define RX_TIMEOUT_COUNT1   (RX_TIMEOUT_COUNT2*10L)
  unsigned long ulCounter;
  byte bCount, bLength, bChecksum;
  byte bTimeout;

  bTimeout = 0;
  for(bCount = 0; bCount < bRxPacketLength; bCount++)
  {
    ulCounter = 0;
    while(gbRxBufferReadPointer == gbRxBufferWritePointer)
    {
      if(ulCounter++ > RX_TIMEOUT_COUNT1)
      {
        bTimeout = 1;
        break;
      }
    }
    if(bTimeout) break;
    gbpRxBuffer[bCount]                                                 =
                    gbpRxInterruptBuffer[gbRxBufferReadPointer++];
  }
  bLength = bCount;
  bChecksum = 0;

  if(gbpTxBuffer[2] != BROADCASTING_ID)
  {
    if(bTimeout && bRxPacketLength != 255)
    {
      TxDString("¥r¥n [Error:RxD Timeout]");
      CLEAR_BUFFER;
    }

    if(bLength > 3) //checking is available.
    {
      if(gbpRxBuffer[0] != 0xff || gbpRxBuffer[1] != 0xff )
      {
        TxDString("¥r¥n [Error:Wrong Header]");
        CLEAR_BUFFER;
        return 0;
      }
      if(gbpRxBuffer[2] != gbpTxBuffer[2] )
      {
        TxDString("¥r¥n [Error:TxID != RxID]");
        CLEAR_BUFFER;
        return 0;
      }
      if(gbpRxBuffer[3] != bLength-4)
      {
        TxDString("¥r¥n [Error:Wrong Length]");
        CLEAR_BUFFER;
        return 0;
      }
      for(bCount  =  2;  bCount  <  bLength;  bCount++)  bChecksum  +=
                    gbpRxBuffer[bCount];
      if(bChecksum != 0xff)
      {
        TxDString("¥r¥n [Error:Wrong CheckSum]");
        CLEAR_BUFFER;
        return 0;
      }
    }
  }
  return bLength;
}


/*
PrintBuffer() print data in Hex code.
PrintBuffer()  needs  two  parameter;  name  of  Pointer(gbpTxBuffer,
                    gbpRxBuffer)
*/
void PrintBuffer(byte *bpPrintBuffer, byte bLength)
{
    byte bCount;
    for(bCount = 0; bCount < bLength; bCount++)
    {
        TxD8Hex(bpPrintBuffer[bCount]);
        TxD8(' ');
    }
    TxDString("(LEN:");TxD8Hex(bLength);TxD8(')');
}
```

```
/*
Print value of Baud Rate.
*/
void PrintBaudrate(void)
{
  TxDString("¥r¥n
                    RS232:");TxD32Dec((16000000L/8L)/((long)UBRR1L+1
                    L) );  TxDString(" BPS,");
  TxDString("    RS485:");TxD32Dec((16000000L/8L)/((long)UBRR0L+1L)  );
                    TxDString(" BPS");
}


/*Hardware Dependent Item*/
#define TXD1_READY                  bit_is_set(UCSR1A,5)
                    //(UCSR1A_Bit5)
#define TXD1_DATA                   (UDR1)
#define RXD1_READY                  bit_is_set(UCSR1A,7)
#define RXD1_DATA                   (UDR1)

#define TXD0_READY                  bit_is_set(UCSR0A,5)
#define TXD0_DATA                   (UDR0)
#define RXD0_READY                  bit_is_set(UCSR0A,7)
#define RXD0_DATA                   (UDR0)

/*
SerialInitialize() set Serial Port to initial state.
Vide Mega128 Data sheet about Setting bit of register.
SerialInitialize() needs port, Baud rate, Interrupt value.

*/
void SerialInitialize(byte bPort, byte bBaudrate, byte bInterrupt)
{
  if(bPort == SERIAL_PORT0)
  {
    UBRR0H = 0; UBRR0L = bBaudrate;
    UCSR0A = 0x02;  UCSR0B = 0x18;
    if(bInterrupt&RX_INTERRUPT) sbi(UCSR0B,7); // RxD interrupt enable
    UCSR0C = 0x06; UDR0 = 0xFF;
    sbi(UCSR0A,6);//SET_TXD0_FINISH; // Note. set 1, then 0 is read
  }
  else if(bPort == SERIAL_PORT1)
  {
    UBRR1H = 0; UBRR1L = bBaudrate;
    UCSR1A = 0x02;  UCSR1B = 0x18;
    if(bInterrupt&RX_INTERRUPT) sbi(UCSR1B,7); // RxD interrupt enable
    UCSR1C = 0x06; UDR1 = 0xFF;
    sbi(UCSR1A,6);//SET_TXD1_FINISH; // Note. set 1, then 0 is read
  }
}

/*
TxD8Hex() print data seperatly.
ex> 0x1a -> '1' 'a'.
*/
void TxD8Hex(byte bSentData)
{
  byte bTmp;

  bTmp =((byte)(bSentData>>4)&0x0f) + (byte)'0';
  if(bTmp > '9') bTmp += 7;
  TxD8(bTmp);
  bTmp =(byte)(bSentData & 0x0f) + (byte)'0';
  if(bTmp > '9') bTmp += 7;
  TxD8(bTmp);
}

/*
TxD80() send data to USART 0.
*/
void TxD80(byte bTxdData)
{
  while(!TXD0_READY);
  TXD0_DATA = bTxdData;
}

/*
TXD81() send data to USART 1.
*/
void TxD81(byte bTxdData)
{
  while(!TXD1_READY);
  TXD1_DATA = bTxdData;
}

/*
TXD32Dex() change data to decimal number system
*/
void TxD32Dec(long lLong)
{
  byte bCount, bPrinted;
  long lTmp,lDigit;
  bPrinted = 0;
  if(lLong < 0)
  {
    lLong = -lLong;
    TxD8('-');
  }
  lDigit = 1000000000L;
  for(bCount = 0; bCount < 9; bCount++)
  {
    lTmp = (byte)(lLong/lDigit);
    if(lTmp)
    {
      TxD8(((byte)lTmp)+'0');
      bPrinted = 1;
    }
    else if(bPrinted) TxD8(((byte)lTmp)+'0');
    lLong -= ((long)lTmp)*lDigit;
    lDigit = lDigit/10;
  }
  lTmp = (byte)(lLong/lDigit);
  /*if(lTmp)*/ TxD8(((byte)lTmp)+'0');
}

/*
TxDString() prints data in ACSII code.
*/
void TxDString(byte *bData)
{
  while(*bData)
  {
    TxD8(*bData++);
  }
}

/*
RxD81() read data from UART1.
RxD81() return Read data.
*/
byte RxD81(void)
{
  while(!RXD1_READY);
  return(RXD1_DATA);
}

/*
SIGNAL() UART0 Rx Interrupt - write data to buffer
*/
SIGNAL (SIG_UART0_RECV)
{
  gbpRxInterruptBuffer[(gbRxBufferWritePointer++)] = RXD0_DATA;
}
```

Connector

Company Name : Molex

Pin Number: 4

Model Number

	Molex Part Number	Old Part Number
Male	22-03-5045	5267-03
Female	50-37-5043	5264-03

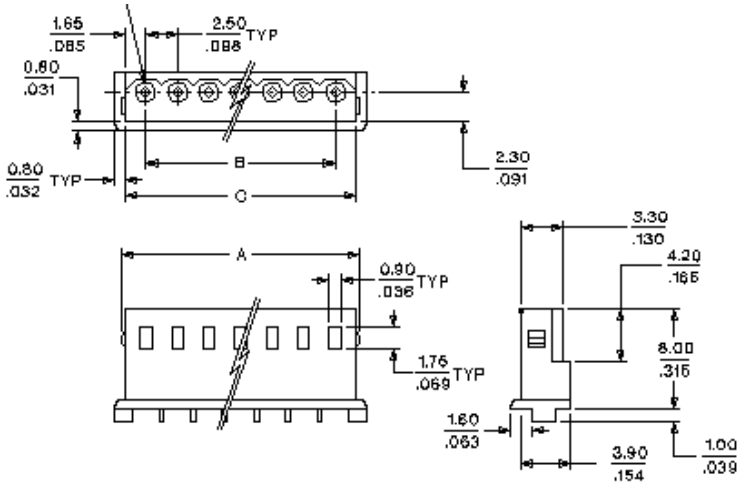
Temperature range : -40°C to +105°C

Contact Insertion Force-max : 14.7N (3.30 lb)

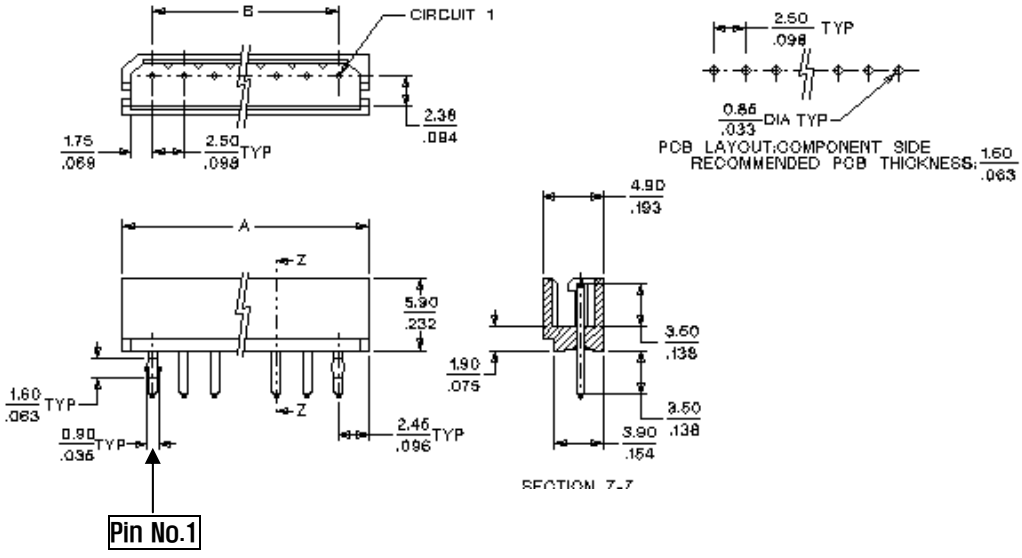
Contact Retention Force-min : 14.7N (3.30 lb)

www.molex.com or www.molex.co.jp for more detail information

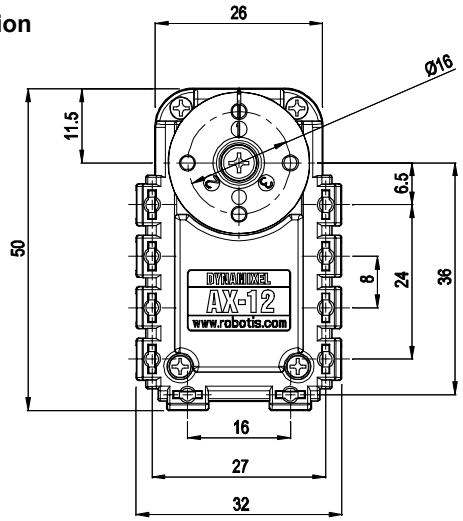
Female Connector

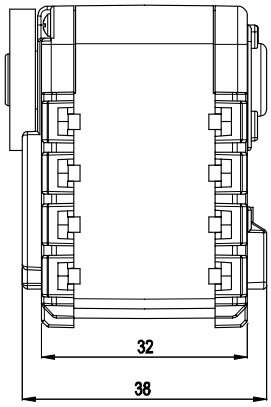


Male Connector



Dimension





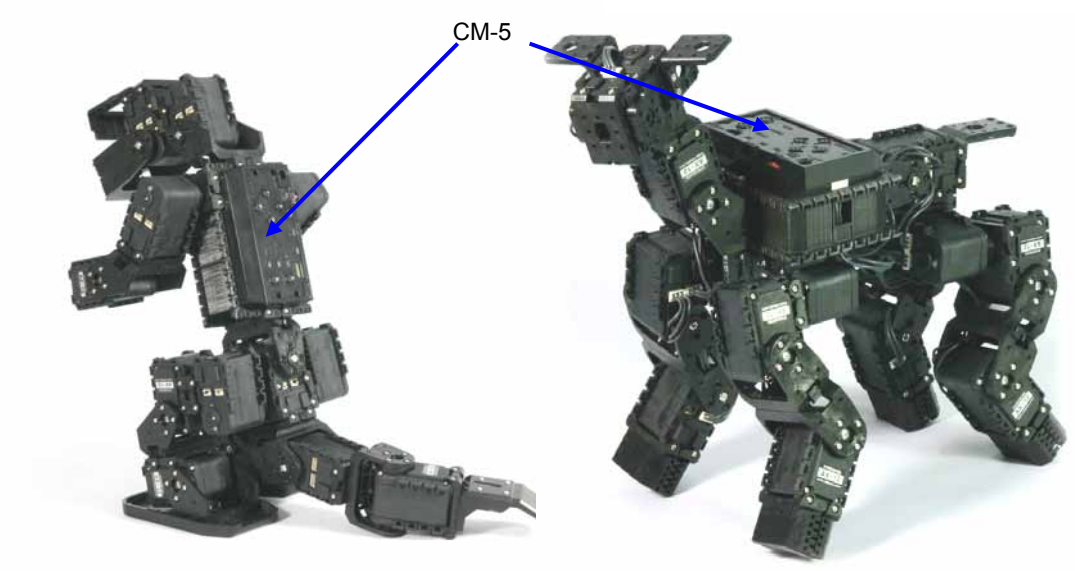
CM-5

Dedicated AX-12 control box. Able to control 30 AX-12 actuators.

6 push buttons (5 for selection, 1 for reset)

Optional installable wireless devices available

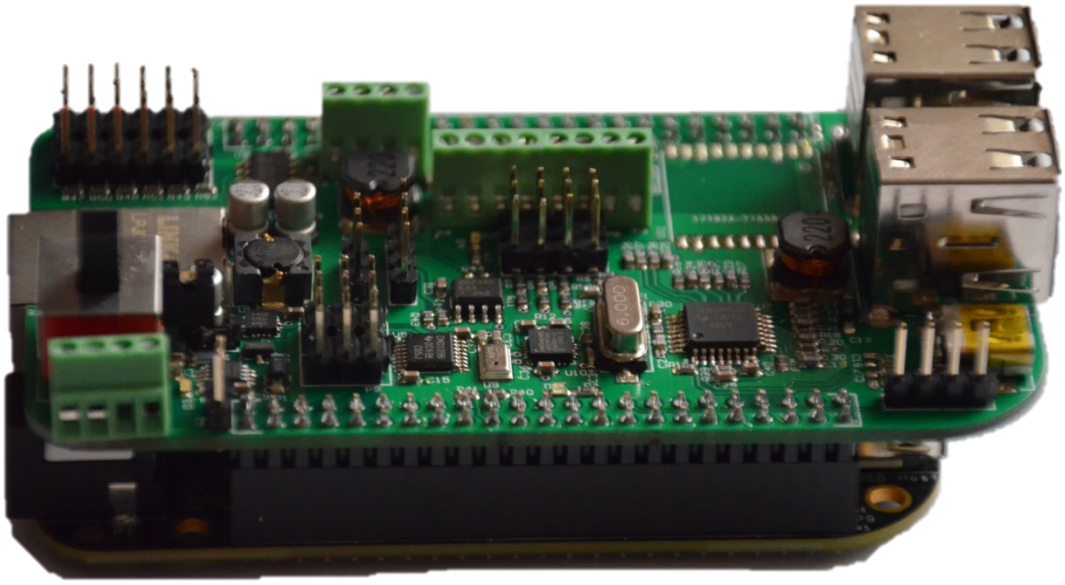
Battery compartment (AA x 8) with recharging capability (when connected to an external SMPS)



ROBOcape

for BeagleBoneBlack

rev 1f beta 5



-Guía usuario-

migalgil@gmail.com

Contenido

1 INTRODUCCION.

Robocape es una placa electrónica especialmente diseñada para su uso con BeagleBoneBlack (BBB). El uso conjunto de ambas posibilita el diseño de cualquier tipo de sistema róbotico, y su programación.

En el diseño de la Robocape, se ha tenido muy en cuenta la necesidad de ofrecer un dispositivo que contenga la gran mayoría de periféricos que los usuarios y diseñadores de robots habitualmente utilizan en sus diseños.

Con la Robocape podrás controlar una gran variedad de tipologías de robots, tales como 2wd, 4wd, antropomorfos, aéreos, etc. Ya que su diseño posibilita una inmensa variedad de configuraciones conjuntamente con la potencia que ofrece BBB.

1.1 CARACTERISTICAS TECNICAS.

- Sistema de alimentación DC/DC independiente, capaz de ofrecer hasta 3A y alimentar directamente a BBB.
- Cargador de baterías lipo de 2 células.
- 4 conexiones pwm para servos mediante buffer de salida.
- 4 puentes H para control de motores.
- 2 entradas para encoder.
- 6 conectores para sensores de ultrasonido HC-SR04.
- 4 conectores para sensores tipo Sharp de salida analógica.
- 2 conectores para conexión red servos Dynamixel (RS-485).
- 2 puertos serie con salida digital 3v3.
- IMU 9 ejes (MPU-9150).
- Altímetro y termómetro.
- GPS integrado (FGPMMOPA6H), con toma para antena externa.
- HUB USB 2.0 de 4 puertos.

1.2 REQUERIMIENTOS.

- Placa BBB, Robocape no es compatible 100% con versiones BeagleBone anteriores.
- Robocape puede ser alimentada desde la propia BBB[1].
- Robocape requiere de un valor de entrada en la alimentación de entre 6-12V.

[1] En caso de que BBB sea alimentada únicamente desde el puerto USB, esta no será capaz de poder alimentar a todo el sistema, por lo que no se recomienda el uso del puerto USB como única alimentación.

- Soporta carga de baterías de 2 celulas[2], por lo que si queremos cargar este tipo de baterías la tensión de alimentación deberá ser entre 9-12V.
- Cable USB TypeA-miniUSB para la interconexión del HUB que incorpora Robocape con BBB.
- Sensores ultrasonido tipología HC-SR04.
- Entradas analógicas max. 3V.
- Antena externa conexión uFL.

2 DESCRIPCIÓN Y USO.

Para poder trabajar con seguridad con Robocape, debemos conocer cuál es la distribución de periféricos y pines dentro del esquema que ofrece BBB. Así pues a continuación se muestran una primera descripción de la distribución física de cada uno de los componentes que integran el diseño Robocape, así como el pinout y su interacción con BBB.

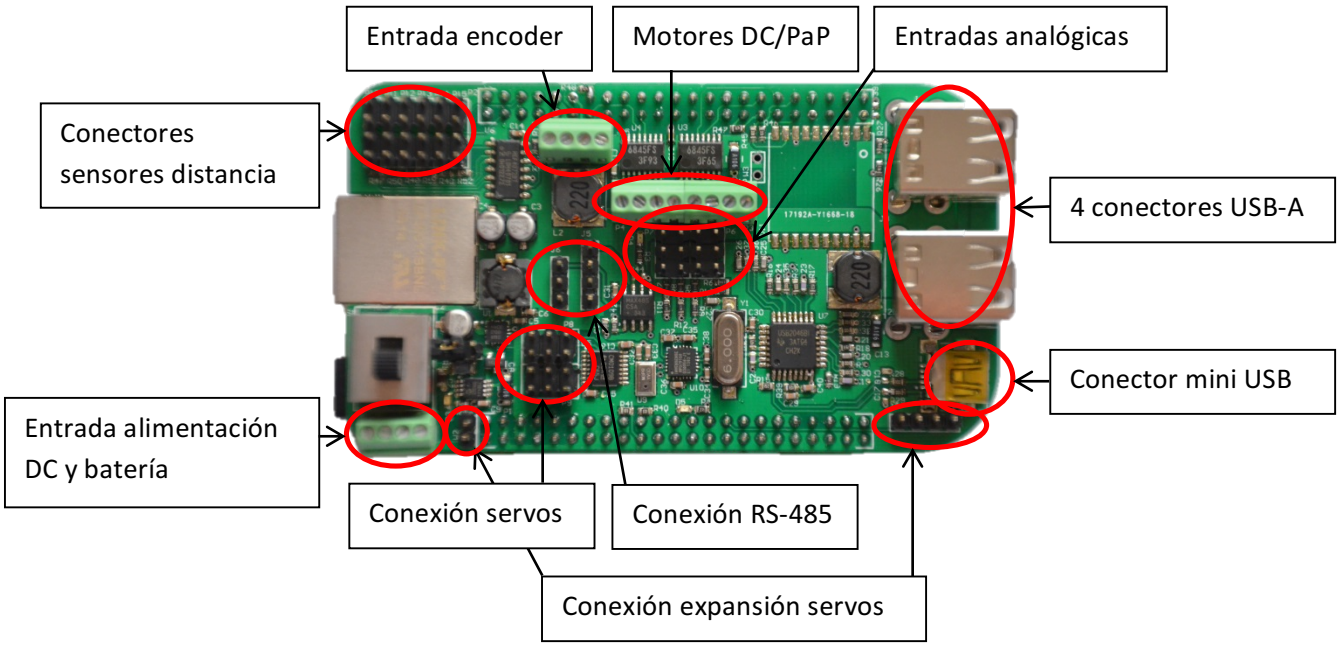


Fig.1 Descripción de conectores y su distribución.

[2] Esto no supone que la Robocape no pueda trabajar con baterías de 3 células, solamente que su cargador no será válido para este tipo de baterías.

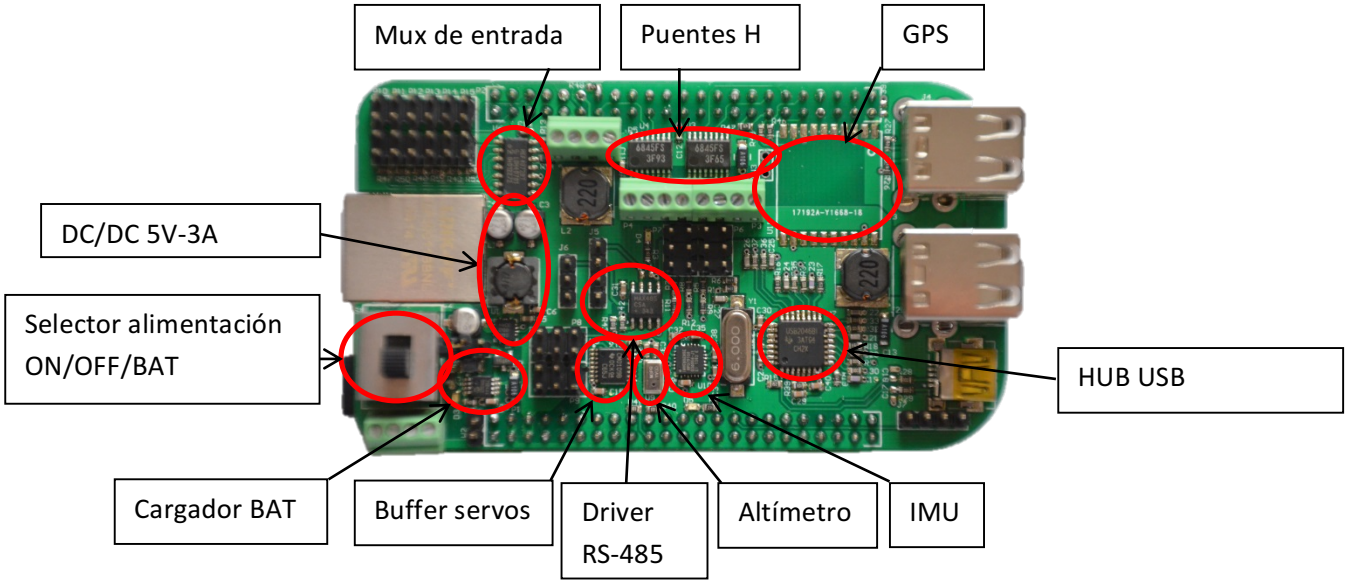


Fig.2 Descripción de los módulos y IC`s y su distribución.

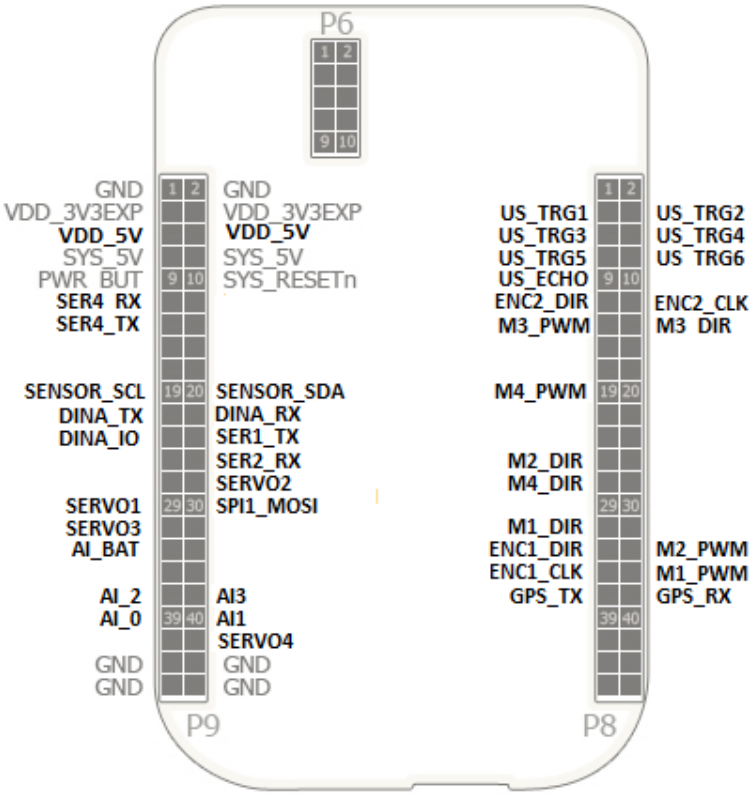


Fig.3 Descripción pinout según distribución conectores BBB.

2.1 ALIMENTACIÓN Y BATERIA.

Robocape posee un conector propio (J1) para toma de alimentación, esta deberá ser de tensión continua DC de al menos 1A en el caso de que estemos alimentando a BBB, y sin tener en cuenta el número de periféricos y motores que tenemos conectados a la placa. El valor máximo de alimentación que puede ser introducido al sistema es de 12V.

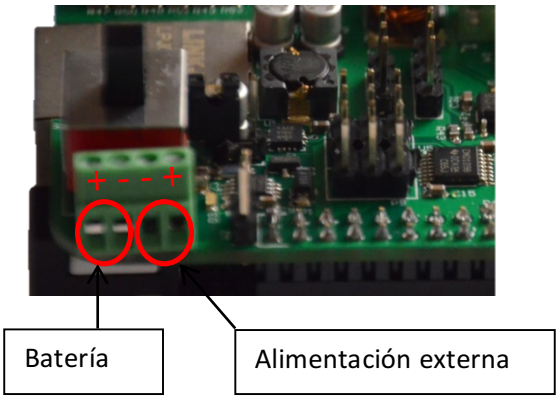


Fig. 4 Conector alimentación principal.

Deberemos llevar especial cuidado a la hora de las conexiones de alimentación y su polaridad, ya que a pesar de que Robocape posee protecciones contra sobretensión y cambio de polaridad, la propia protección podrá ser destruida, y el sistema dejará de funcionar hasta que esta sea reemplazada.

Robocape incluye un sistema de carga de batería, este sistema está diseñado para baterías de 2 celdas (2C) de 7,4V[3]. En cualquier caso podremos conectar al mismo tiempo una fuente externa y la batería de 2C y seleccionar que fuente queremos que alimente a nuestro sistema mediante el selector. Si decidimos cargar nuestra batería 2C, deberemos posicionar el selector para alimentación externa, y además insertar el jumper (W1) que existe junto al selector, de este modo la carga será iniciada y esta será indicada mediante un led (D3).

2.2 ALTIMETRO E IMU.

De la necesidad en cualquier tipo de robot, de obtener su posicionamiento en el espacio donde está realizando la tarea a desarrollar, nace la obligación del uso de sistemas de medición espacial tales como acelerómetros, giróscopos, etc. Robocape incorpora toda una serie de sensores que nos aportan datos e información acerca del estado y posicionamiento espacial en todas sus variables.

[3] Esto no supone la imposibilidad de conectar una batería de voltaje diferente, sino que esta no podrá ser cargada por Robocape, ya que el cargador esta especialmente diseñado para lipo-2C

El primero de estos sistemas es el MPU-9150, primer seguidor de posicionamiento del mundo con procesador integrado, el cual incluye 3 acelerómetros, 3 giróscopos y 3 magnetómetros. La comunicación de este sistema con la BBB se realiza mediante un bus I2C, concretamente y siguiendo la nomenclatura BBB el bus se corresponde con el I2C_2 (pines 19 y 20 del conector P9). La dirección I2C HW ofrecidas por este dispositivo pueden ser varias según la configuración, en el caso de la Robocape, esta dirección se encuentra en la 0x69, por tener el AD0=1.

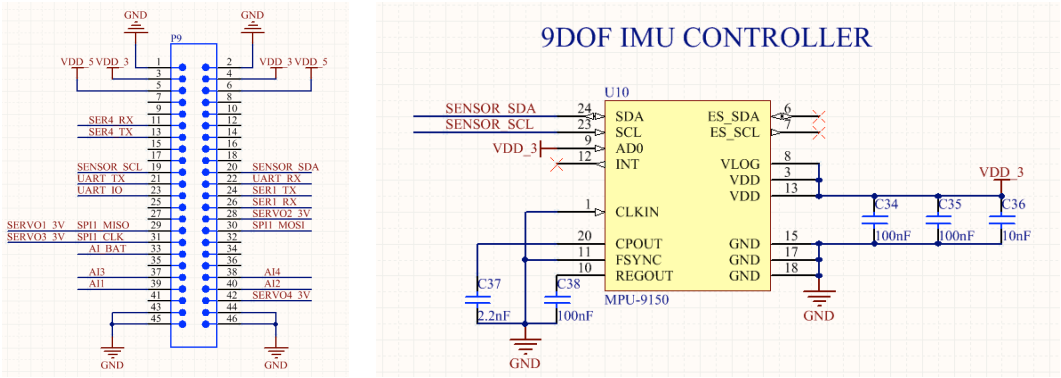


Fig. 5 Esquema de conexión MPU-9150

El mismo bus I2C_2 es compartido por el altímetro MPL3115A2, debido al gran número de periféricos que son necesarios. Esto no supone ningún problema ya que la dirección HW de sendos periféricos son diferentes. En el caso del MPL3115A2 la dirección de acceso es la 0x60. La inclusión de este dispositivo dentro de Robocape, aporta información a la BBB, referente a la presión/altitud, así como la temperatura existente en el entorno.

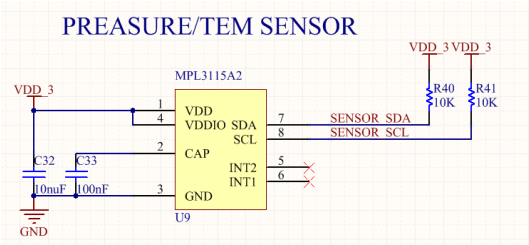


Fig. 6 Esquema de conexión MPL 3115A2

2.3 GPS

Los sistemas GPS, ampliamente utilizados en multitud de dispositivos electrónicos son una ayuda muy importante en el campo de los sistemas autónomos autoguiados. De este modo, siendo una herramienta fundamental en el desarrollo de sistemas robóticos de última generación, Robocape integra dentro de su diseño un sistema GPS, que se comunica con BBB mediante un puerto serie (uart5 según

nomenclatura BBB). Se trata del modelo FGPMMOPA6H, que integra un sistema GPS de alta sensibilidad, junto a una antena integrada, aumentando así su integración. De igual modo existe la posibilidad de conectar una antena externa en el caso de que necesitemos una mejor cobertura.

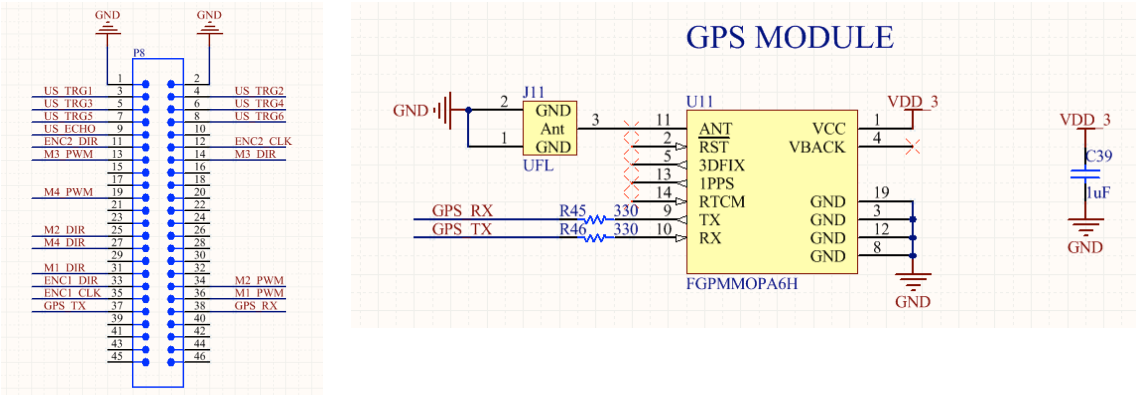


Fig. 7 Esquema de conexión GPS

2.4 CONEXIÓN USB2.0

Debido a la enorme disparidad de necesidades que existen en el mundo de la robótica en el uso de periféricos, ya sean de comunicaciones, sistemas de visión, sonido, etc., es imposible poder incluir todas estas posibilidades dentro de un mismo diseño. De esta forma, Robocape aporta una solución que posibilita la inclusión de todos estos sistemas mediante una conexión USB, conexión ampliamente usada por gran número de fabricantes. Para ello, el diseño ofrece un HUB USB (TUSB2046B) de cuatro puertos, el cual es conectado entre el conector USB-A existente en la BBB y el conector mini-USB que hay en la Robocape. De esta manera la conectividad de BBB aumenta pudiendo interconectar un mayor número de dispositivos al sistema. El uso de este dispositivo es totalmente transparente al sistema operativo, ya que se trata de HUB genérico, el cual es soportado por la gran mayoría de SO que son utilizados por BBB.

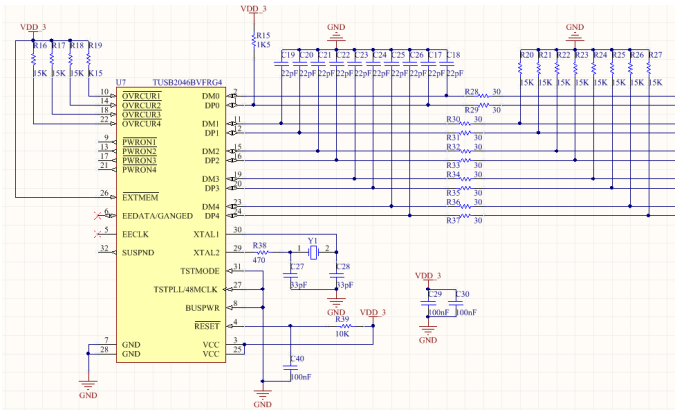
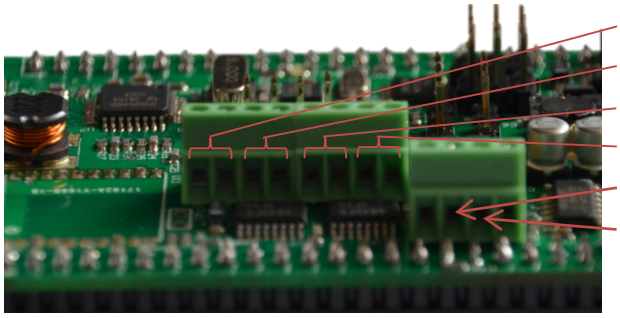


Fig. 8 Esquema de conexión del TUSB2046B.

2.5 CONTROL DE MOTORES DC Y PaP

La integración de un sistema de potencia que gestione dispositivos como motores, es esencial en un diseño integrado, de esta manera Robocape integra 4 puentes H, desde los cuales se pueden manejar o bien motores DC, o motores paso-paso. El sistema de gestión de motores de la Robocape, integra controladores (BA6845) capaces de trabajar con corrientes no superiores a 1A en régimen nominal, lo que es suficiente para gran parte de los diseños. La integración de estos controladores con BBB, se realiza mediante señales PWM y pines IO que gestionan la dirección, que provienen directamente de los controladores PWM que BBB tiene integrados. En la Fig. 9, podremos observar cual es la relación existente entre las señales PWM/dirección y las conexiones externa que ofrece Robocape.



P8

DGND	1	2	DGND
GPIO_38	3	4	GPIO_39
GPIO_34	5	6	GPIO_35
TIMER4	7	8	TIMER7
TIMER5	9	10	TIMER6
GPIO_45	11	12	GPIO_44
EHRPWM2B	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
EHRPWM2A	19	20	GPIO_63
GPIO_62	21	22	GPIO_37
GPIO_36	23	24	GPIO_33
GPIO_32	25	26	GPIO_61
GPIO_86	27	28	GPIO_88
GPIO_87	29	30	GPIO_89
GPIO_10	31	32	GPIO_11
GPIO_9	33	34	EHRPWM1B
GPIO_8	35	36	EHRPWM1A
GPIO_78	37	38	GPIO_79
GPIO_76	39	40	GPIO_77
GPIO_74	41	42	GPIO_75
GPIO_72	43	44	GPIO_73
EHRPWM2A	45	46	EHRPWM2B

MOTOR4/FASEB_M2p→ EHRPWM2A
MOTOR3/FASEA_M2p→ EHRPWM2B
MOTOR2/FASEB_M1p→ EHRPWM1B
MOTOR1/FASEA_M1p→EHRPWM1A
Encoder1
Encoder2

Fig. 9 Descripción conectores motores y encoder.

Las salidas utilizadas desde BBB para el manejo de este tipo de motores provienen de los controladores EHRPWM1 y EHRPWM2 (según configuración mostrada en tabla 1), los cuales poseen 2 salidas independientes por controlador, así de ese modo podremos gestionar los 4 motores DC o 2 motores PaP descritos anteriormente. No solamente existen las señales de PWM, las cuales indican la potencia que le va a ser transmitida al motor, sino que debemos indicarle que dirección de giro deben tener, para ello se incluyen varias señales (Mx_DIR), la cuales indicaran al controlador que sentido de giro deberá tener el motor.

ROBOBONE SIGNAL	CONECTOR-PIN	PIN MODE	BBB SIGNAL
M1_PWM	P8-36	MODE 2	EHRPWM1A
M1_DIR	P8-31	MODE 7	GPIO0[10]
M2_PWM	P8-34	MODE 2	EHRPWM1B
M2_DIR	P8-25	MODE 7	GPIO1[0]
M3_PWM	P8-13	MODE 4	EHRPWM2B
M3_DIR	P8-14	MODE 7	GPIO0[26]
M4_PWM	P8-19	MODE 4	EHRPWM2A
M4_DIR	P8-27	MODE 7	GPIO2[22]

Tabla 1.

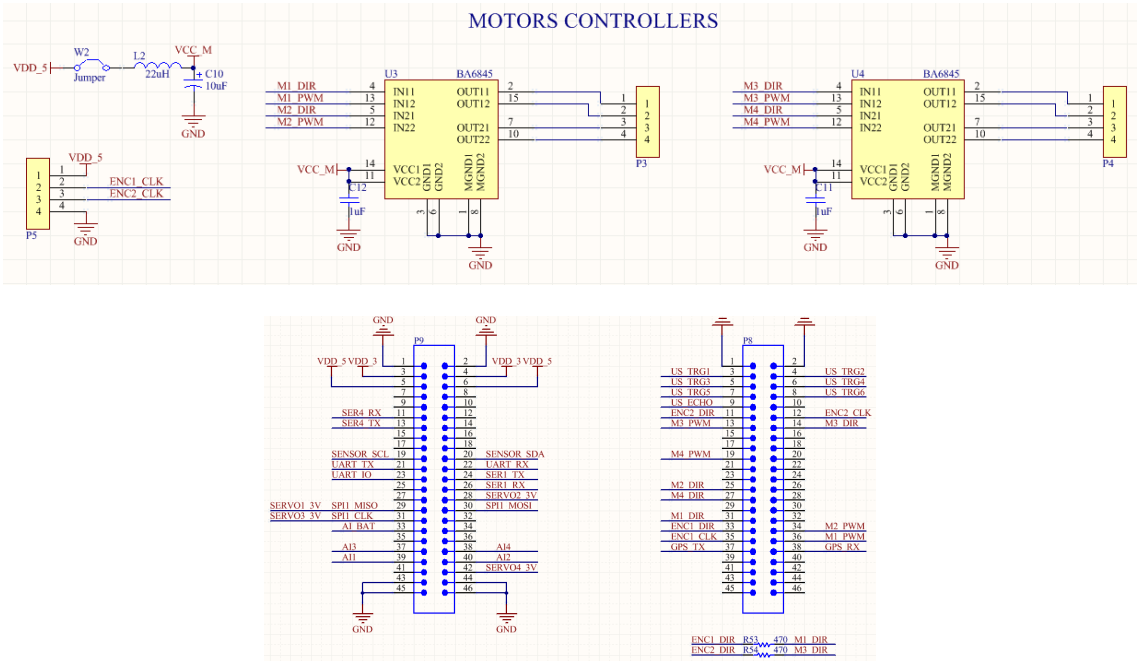


Fig. 11 Esquema de conexiones para controladores de motores y encoder.

En relación a las entradas para las señales de encoder, debemos indicar que se trata de un diseño especial, ya que si bien desde BBB podemos leer la señal de clock del encoder, la cual determina los grados girados, no así podremos determinar exactamente la dirección de giro, ya que para poder utilizar otros periféricos únicamente podía hacerse uso de 2 entradas de encoder. Por este motivo se ha llegado a un diseño de compromiso, con ciertas restricciones, y es que se han puenteado las líneas de dirección de salida de control de motores con las de entrada de dirección del encoder. Esto supone en la realidad que ante cambios en la dirección de los motores habrán errores en la lectura del valor absoluto del encoder, por lo que habrá que realizar adaptaciones SW que gestionen esta anomalía, en el caso de que deseemos un control de posición del sistema, si bien para control de velocidades esto no supone un problema.

2.6 PUERTOS SERIE Y SPI.

La proliferación de periféricos para sistemas empotrados es cada vez mayor, y estos suelen estar diseñados para una conexión serie a través de la cual enviar y recibir información desde el sistema de control. Robocape, ofrece, además de los puertos serie que sus periféricos utilizan, 3 conexiones externas para poder insertar modulos externos en nuestro diseño. Se trata de 2 conectores cuya salida incluye sendos puertos serie, además de un conector extra para conexiones de periféricos que utilicen el puerto SPI.

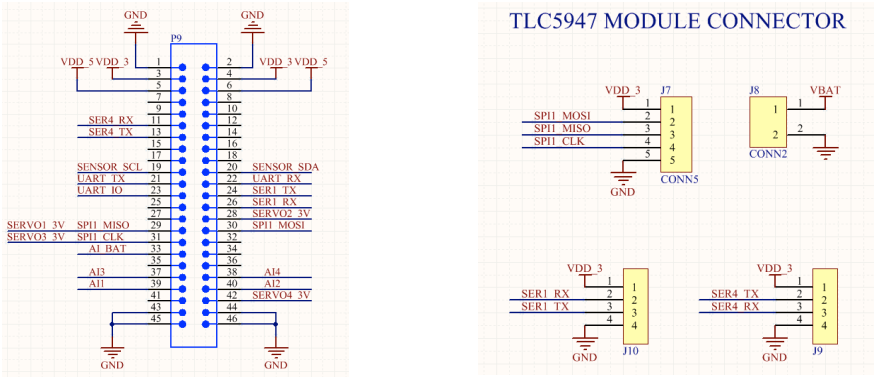


Fig. 11 Descripción conexionado puertos serie y spi.

Como podemos observar en la Fig.11, indica que el puerto SPI está especialmente indicado para su uso con un módulo especialmente diseñado para Robocape, cuyo uso está destinado a proporcionar hasta un máximo de 24 señales PWM.

2.7 ENTRADAS ANALÓGICAS PARA SENSORES SHARP.

Los sensores con salida analógica siempre han protagonizado un papel destacado en gran número de diseños, por ello Robocape, incluye 4 entradas analógicas. Estas están especialmente preparadas para poder insertar directamente los conocidos sensores de distancia Sharp (GP2Dxx) con salida analógica. Debido a que las entradas analógicas que BBB posee solamente admiten tensiones hasta 1.8V, se adapta las tensiones máximas posibles de este tipo de sensores a las entradas que soporta BBB únicamente con un divisor resistivo. Así de esta manera podremos conectar cualquier entrada analógica con tan solo puentear la resistencia de entrada, y siempre teniendo en cuenta los valores máximos soportados por BBB.

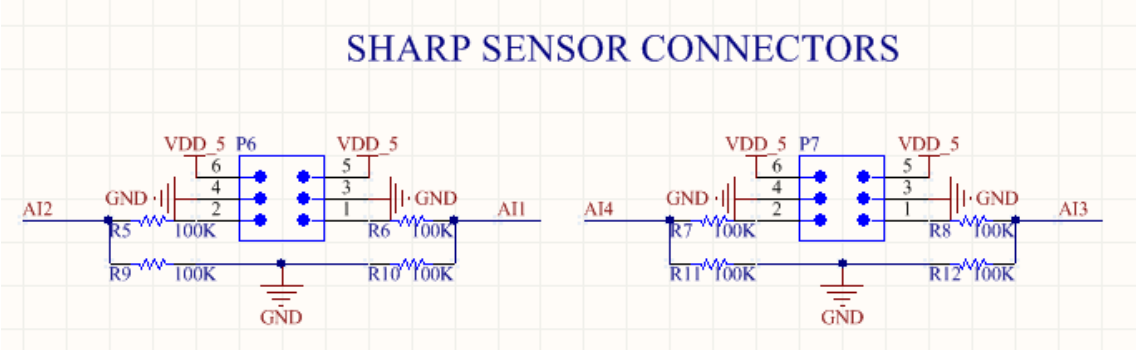


Fig. 12 Conexiones puertos de entrada ADC con sensores Sharp.

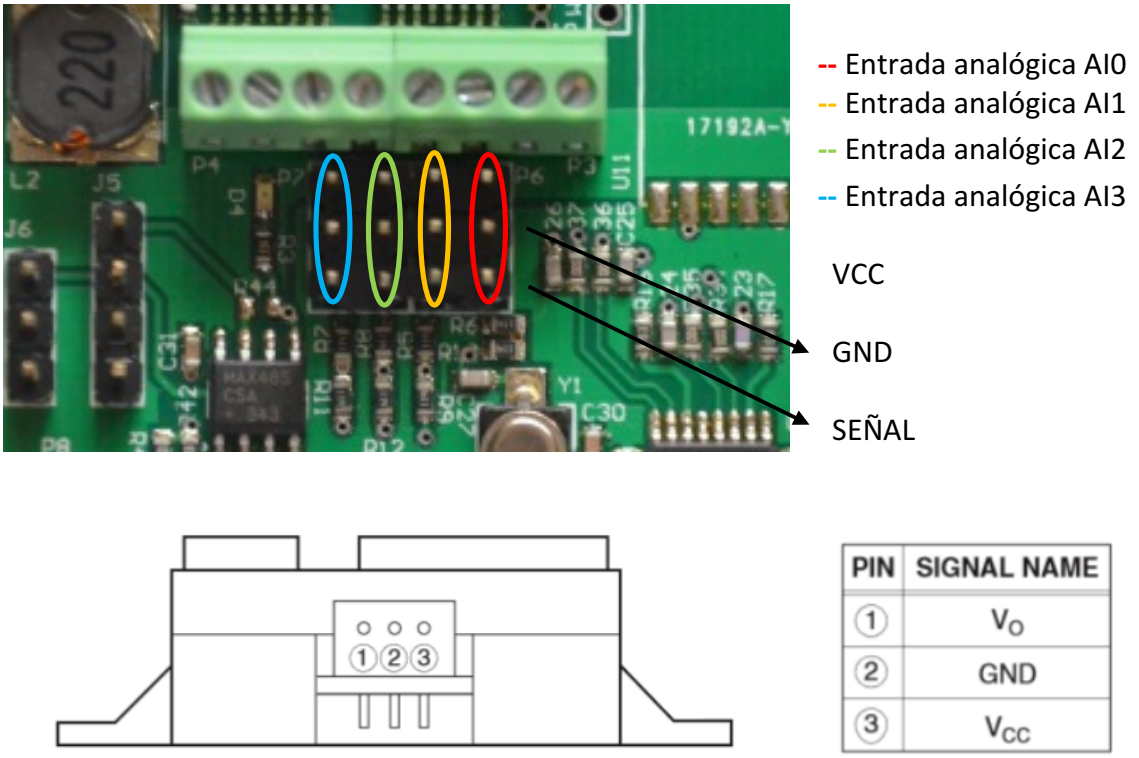


Fig. 13 Distribución de los pines de entrada analógicas

2.8 SENSORES ULTRASONIDO HC-SR04.

Ampliamente utilizados en todos los ámbitos de la robótica, son los sensores de ultrasonido, debido a su bajo coste, así como su facilidad de uso. Por este Echo Robocape incorpora hasta un total de 6 conectores para poder conectar un anillo de sensores de ultrasonido. El diseño incluye la lógica necesaria para no hacer un uso elevado de pines.

El funcionamiento de este tipo de sensores se basan en el cálculo del tiempo de vuelo de la señal transmitida, además los modulos HC-SR04 integran la lógica necesaria

para no tener que abordar el control de la onda de detección. Así pues el uso del módulo se rige por el siguiente funcionamiento:

1. Enviar un Pulso "1" de al menos de 10uS por el Pin Trigger (Disparador).
2. El sensor enviará 8 Pulsos de 40KHz (Ultrasonido) y coloca su salida Echo a alto (set), se debe detectar este evento e iniciar un conteo de tiempo.
3. La salida Echo se mantendrá en alto hasta recibir el eco reflejado por el obstáculo a lo cual el sensor pondrá su pin Echo a bajo, es decir, terminar de contar el tiempo.
4. Se recomienda dar un tiempo de aproximadamente 50ms de espera después de terminar la cuenta.
5. La distancia es proporcional a la duración del pulso y puedes calcularla con las siguiente formula (Utilizando la velocidad del sonido = 340m/s):

Distancia en cm (centímetros) = Tiempo medido en us x 0.017

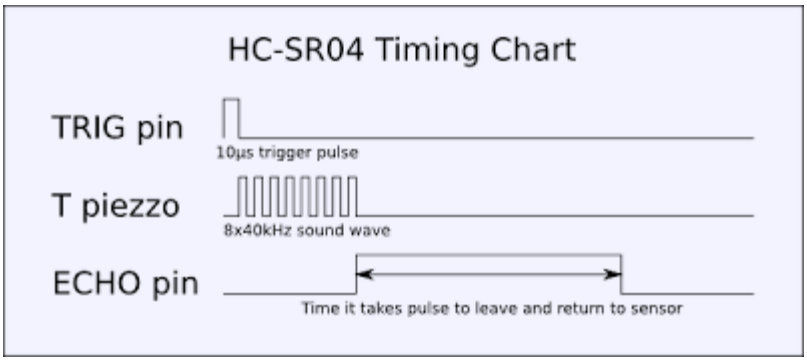


Fig. 14 Cronología de disparo en el HC-SR04.

Robocape integra en su diseño un pin de disparo por cada sensor, pero únicamente un solo pin de entrada de echo (pin BB y BBB **P8.9**), por lo que únicamente podremos disparar un sensor a la vez y esperar su respuesta. Este diseño se basa en la idea de que un disparo múltiple en un anillo de ultrasonidos puede generar interferencias entre sensores, así como rebotes no determinados, por este motivo se fuerza a realizar una lectura secuencial de los sensores de distancia.

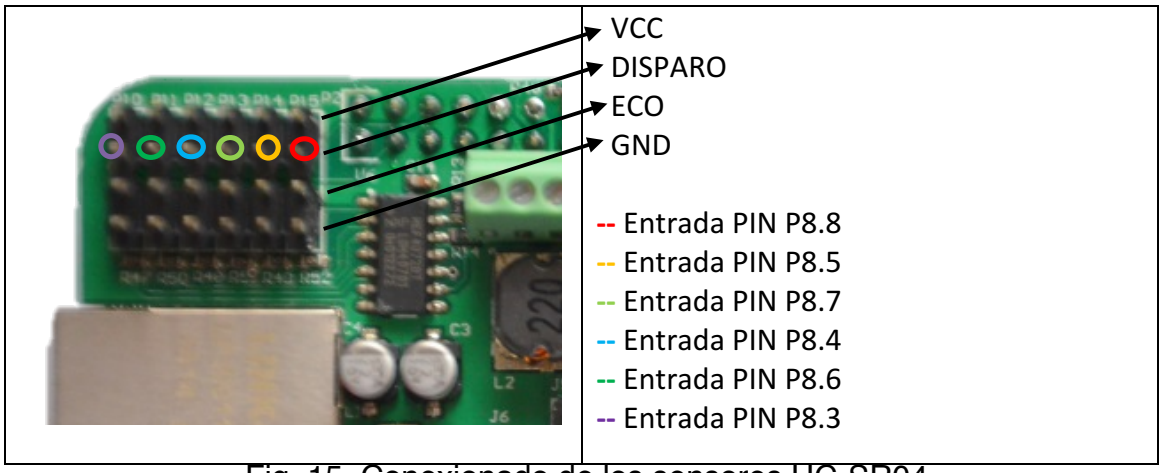


Fig. 15 Conexionado de los sensores HC-SR04

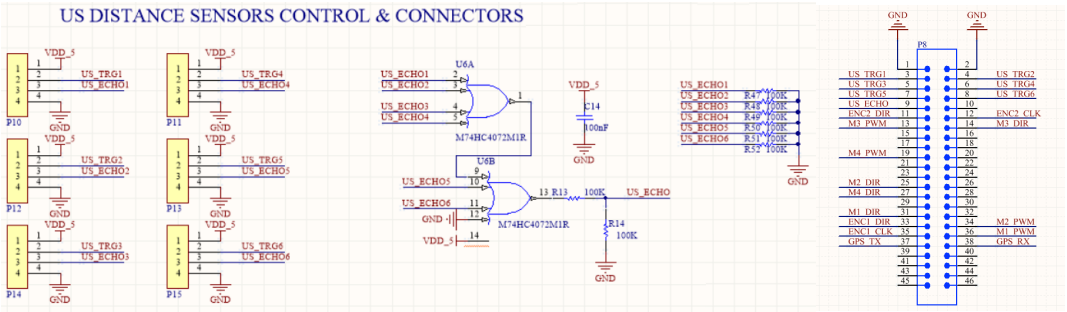


Fig. 16 Esquemático de conexiones para los sensores de ultrasonido.

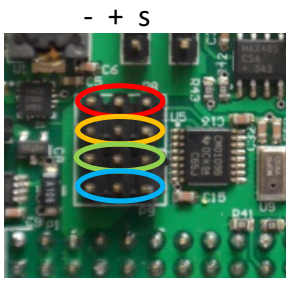
2.9 SERVOS, CONEXIÓN Y AMPLIACIÓN.

El uso de servomotores, así como sus señales en la robótica cada vez tienen una mayor aceptación, abarcando un amplio rango de posibilidades tanto de servomotores, controladores brushless, sistema de radiocontrol, etc. Por este motivo Robocape adapta las señales y buses existentes en BBB hacia la conexión de dispositivos cuya actuación viene determinada por señales de servomotores.

De forma directa, obtenemos desde la BBB 4 líneas que provienen de los controladores PWM todavía existentes en BBB, Robocape adapta estas señales a la tensión típica de 5V en las señales servo mediante un buffer que sirve de protección ante problemas eléctricos en los circuitos externos. En la Fig.15 podemos observar cual es la distribución de pines en los conectores de salida, así como su interacción con los controladores PWM de BBB, y en la Tabla 2 la distribución de dichas señales según nomenclatura y modos de programación en los pines de la BBB.

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETn
GPIO_30	11	12	GPIO_60
GPIO_31	13	14	EHRPWM1A
GPIO_48	15	16	EHRPWM1B
GPIO_4	17	18	GPIO_5
I2C2_SCL	19	20	I2C2_SDA
EHRPWM0B	21	22	EHRPWM0A
GPIO_49	23	24	GPIO_15
GPIO_117	25	26	GPIO_14
GPIO_125	27	28	ECAPPWM2
EHRPWM0B	29	30	GPIO_122
EHRPWM0A	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	ECAPPWM0
DGND	43	44	DGND
DGND	45	46	DGND



Servo4→ECAPPWM0

Servo3→EHRPWM0A

Servo2→ECAPPWM2

Servo1→ERPWM0B

Fig.17 Distribución de pines para señales PWM-Servo.

ROBOBONE SIGNAL	CONECTOR-PIN	PIN MODE	BBB SIGNAL
Servo1	P9-29	MODE 1	EHRPWM0B
Servo2	P9-28	MODE 4	ECAPPWM2
Servo3	P9-31	MODE 1	EHRPWM0A
Servo4	P9-42	MODE 0	ECAPPWM0

Tabla 2.

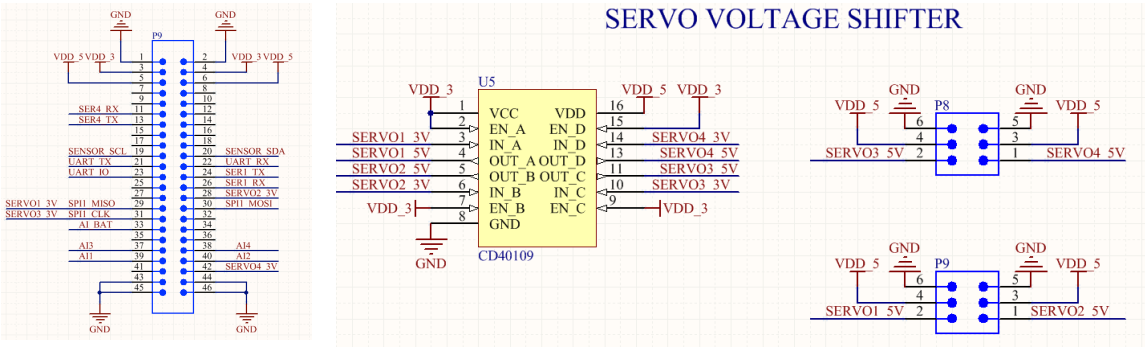


Fig. 18 Conexiones de buffers de salida para señales PWM-Servo.

Además de las salidas directas de las señales PWM, Robocape tiene la posibilidad de añadir una board de expansión[4], cuya conexión se realiza mediante los conectores J7 y J8, y que aporta 24 canales PWM configurables mediante el bus SPI. Se trata de un accesorio de expansión basado en el chip TLC5947, además de un microcontrolador y lógica que nos permitirán realizar un switch entre las señales PWM de un sistema externo o bien las señales provenientes desde el chip TLC5947, mediante una señal externa, la cual determinará que señales serán enviadas a los periféricos conectados. Este sistema de gestión es ampliamente usado por aquellos sistemas autónomos que poseen sistemas de apoyo para el manejo remoto en caso de fallo.

Este diseño supone una pérdida de 2 señales directas PWM, ya que los pines utilizados para la comunicación SPI están multiplexados con 2 de las señales PWM, tal y como se puede observar en la Fig.17. Por tanto las señales Servo1 y Servo3 quedaran inutilizadas en el caso de que estemos utilizando el puerto SPI1, el cual utiliza BBB y Robocape para la comunicación con el módulo de expansión.

[4] Todavía en desarrollo.

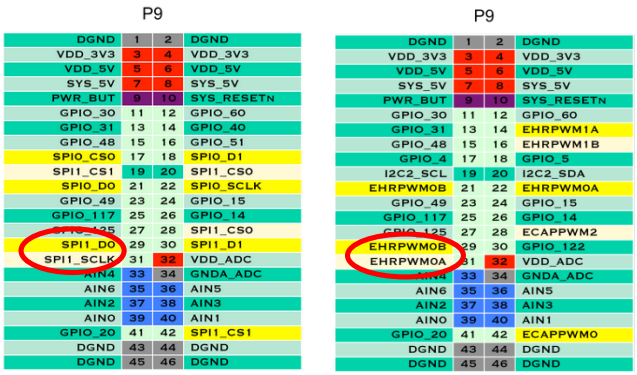
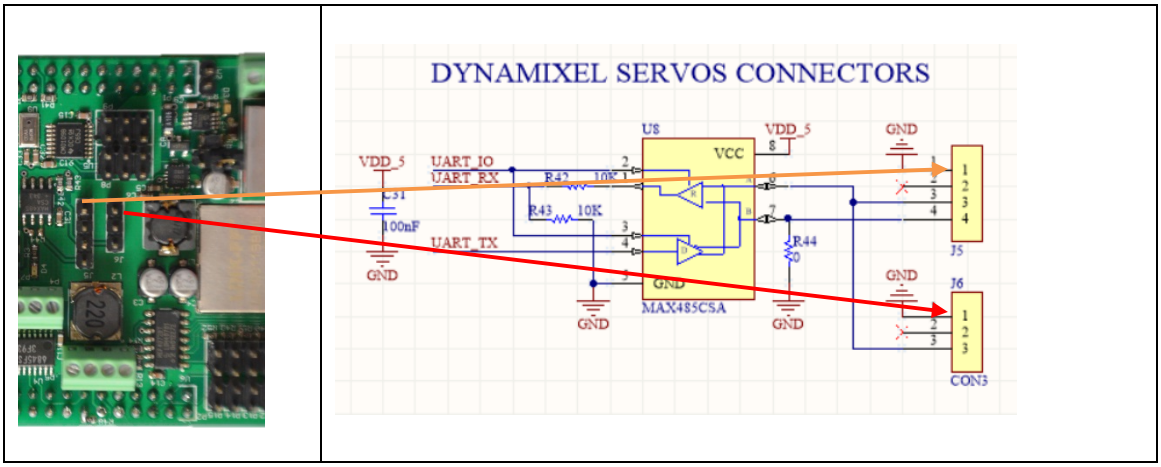


Fig. 19 Diferencias según selección de mux en BBB

2.10 RS485 Y SENSORES DYNAMIXEL.

A lo largo de los últimos años se han desarrollado alrededor del mundo de la robótica actuadores con capacidades especiales que eran requeridas en dichas áreas de la robótica. De este modo los servos Dynamixel, ampliamente utilizados en el mundo de la robótica humanoide y antropomórfica, se distinguen por los demás servos en que la conexión utilizada es en forma de bus de comunicaciones, pudiendo de esta forma minimizar cableados y numero de pines.

Así pues Robocape integra un transceiver RS485, que posibilita la interacción con este tipo de actuadores, mediante el uso de la UART2 que la BBB posee.



ANEXO A

P9 Descripción de señales y sus modos.

Pin	ZCZ Ball	Name	DT Offset	Mode 0	1	2	3	4	5	6	7
1	-	GND									
2	-	GND									
3	-	DC_3.3V									
4	-	DC_3.3V									
5	-	VDD_5V									
6	-	VDD_5V									
7	-	SYS_5V									
8	-	SYS_5V									
9	-	PWR_BUT									
10	A10	SYS_RESETn		RESET_OUT	-	-	-	-	-	-	-
11	T17	UART4_RXD	0x070	gpmc_wait0	mii2_crs	gpmc_csn4	rmii2_crs_dv	mmc1_sdcd	pr1_mii1_col	uart4_rxd	gpio0[30]
12	U18	GPIO1_28	0x078	gpmc_ben1	mii2_col	gpmc_csn6	mmc2_dat3	gpmc_dir	pr1_mii1_rxlink	mcasp0_aclkr	gpio1[28]
13	U17	UART4_TXD	0x074	gpmc_wpn	mii2_rxerr	gpmc_csn5	rmii2_rxerr	mmc2_sdcd	pr1_mii1_txen	uart4_txd	gpio0[31]
14	U14	EHRPWM1A	0x048	gpmc_a2	mii2_txd3	rgmii2_td3	mmc2_dat1	gpmc_a18	pr1_mii1_txd2	ehrpwm1A	gpio1[18]
15	R13	GPIO1_16	0x040	gpmc_a0	gmii2_txen	rmii2_tctl	mii2_txen	gpmc_a16	pr1_mii_mt1_clk	ehrpwm1_tripzone_input	gpio1[16]
16	T14	EHRPWM1B	0x04C	gpmc_a3	mii2_txd2	rgmii2_td2	mmc2_dat2	gpmc_a19	pr1_mii1_txd1	ehrpwm1B	gpio1[19]
17	A16	I2C1_SCL	0x15C	spi0_cs0	mmc2_sdwp	I2C1_SCL	ehrpwm0_synci	pr1_uart0_txd	pr1_edio_data_in1	pr1_edio_data_out1	gpio0[5]
18	B16	I2C1_SDA	0x158	spi0_d1	mmc1_sdwp	I2C1_SDA	ehrpwm0_tripzone	pr1_uart0_rxd	pr1_edio_data_in0	pr1_edio_data_out0	gpio0[4]
19	D17	I2C2_SCL	0x17C	uart1_rtsn	timer5	dcan0_rx	I2C2_SCL	spi1_cs1	pr1_uart0_rts_n	pr1_edc_latch1_in	gpio0[13]
20	D18	I2C2_SDA	0x178	uart1_ctsn	timer6	dcan0_tx	I2C2_SDA	spi1_cs0	pr1_uart0_cts_n	pr1_edc_latch0_in	gpio0[12]
21	B17	UART2_TXD	0x154	spi0_d0	uart2_txd	I2C2_SCL	ehrpwm0B	pr1_uart0_rts_n	pr1_edio_latch_in	EMU3	gpio0[3]
22	A17	UART2_RXD	0x150	spi0_sclk	uart2_rxd	I2C2_SDA	ehrpwm0A	pr1_uart0_cts_n	pr1_edio_sof	EMU2	gpio0[2]
23	V14	GPIO1_17	0x044	gpmc_a1	gmii2_rxdv	rgmii2_rxdv	mmc2_dat0	gpmc_a17	pr1_mii1_txd3	ehrpwm0_synco	gpio1[17]
24	D15	UART1_TXD	0x184	uart1_txd	mmc2_sdwp	dcan1_rx	I2C1_SCL	-	pr1_uart0_txd	pr1_pru0_pru_r31_16	gpio0[15]
25	A14	GPIO3_21	0x1AC	mcasp0_ahclkx	eQEP0_strobe	mcasp0_axr3	mcasp1_axr1	EMU4	pr1_pru0_pru_r30_7	pr1_pru0_pru_r31_7	gpio3[21]
26	D16	UART1_RXD	0x180	uart1_rxd	mmc1_sdwp	dcan1_tx	I2C1_SDA	-	pr1_uart0_rxd	pr1_pru1_pru_r31_16	gpio0[14]
27	C13	GPIO3_19	0x1A4	mcasp0_fsr	eQEP0B_in	mcasp0_axr3	mcasp1_fsx	EMU2	pr1_pru0_pru_r30_5	pr1_pru0_pru_r31_5	gpio3[19]
28	C12	SPI1_CS0	0x19C	mcasp0_ahclkr	ehrpwm0_synci	mcasp0_axr2	spi1_cs0	eCAP2_in_PWM2_out	pr1_pru0_pru_r30_3	pr1_pru0_pru_r31_3	gpio3[17]
29	B13	SPI1_D0	0x194	mcasp0_fsx	ehrpwm0B	-	spi1_d0	mmc1_sdcd	pr1_pru0_pru_r30_1	pr1_pru0_pru_r31_1	gpio3[15]
30	D12	SPI1_D1	0x198	mcasp0_axr0	ehrpwm0_tripzone	-	spi1_d1	mmc2_sdcd	pr1_pru0_pru_r30_2	pr1_pru0_pru_r31_2	gpio3[16]
31	A13	SPI1_SCLK	0x190	mcasp0_aclkx	ehrpwm0A	-	spi1_sclk	mmc0_sdcd	pr1_pru0_pru_r30_0	pr1_pru0_pru_r31_0	gpio3[14]
32	-	VADC									
33	C8	AIN4									
34	-	AGND									
35	A8	AIN6									
36	B8	AIN5									
37	B7	AIN2									
38	A7	AIN3									
39	B6	AIN0									
40	C7	AIN1									
41	D14	CLKOUT2	0x1B4	xdma_event_intr1	-	tclkin	clkout2	timer7	pr1_pru0_pru_r31_16	EMU3	gpio0[20]
#	D13	GPIO3_20	0x1A8	mcasp0_axr1	eQEP0_index	-	Mcasp1_axr0	EMU3	pr1_pru0_pru_r30_6	pr1_pru0_pru_r31_6	gpio3[20]
42	C18	GPIO0_7	0x164	eCAP0_in_PWM0_out	uart3_txd	spi1_cs1	pr1_ecap0_ecap_capin_apwm_o	spi1_sclk	mmc0_sdwp	xdma_event_intr2	gpio0[7]
@	B12	GPIO3_18	0x1A0	mcasp0_aclkr	eQEP0A_in	mcaspo_axr2	mcasp1_aclkx	mmc0_sdwp	pr1_pru0_pru_r30_4	pr1_pru0_pru_r31_4	gpio3[18]
43	-	GND									
44	-	GND									
45	-	GND									
46	-	GND									

Output
Input
I/O

#,@ - 2 Signals available on pins 41 and 42, set the unused signal to Input

P8 Descripción de señales y sus modos.

Pin	ZCZ Ball	Name	DT Offset	Mode 0	1	2	3	4	5	6	7
1	-	GND									
2	-	GND									
3	R9	GPIO1_6	0x018	gpmc_ad6	mmc1_dat6	-	-	-	-	-	gpio1[6]
4	T9	GPIO1_7	0x01C	gpmc_ad7	mmc1_dat7	-	-	-	-	-	gpio1[7]
5	R8	GPIO1_2	0x008	gpmc_ad2	mmc1_dat2	-	-	-	-	-	gpio1[2]
6	T8	GPIO1_3	0x00C	gpmc_ad3	mmc1_dat3	-	-	-	-	-	gpio1[3]
7	R7	TIMER4	0x090	gpmc_advn_ale	-	timer4	-	-	-	-	gpio2[2]
8	T7	TIMER7	0x094	gpmc_oen_ren	-	timer7	-	-	-	-	gpio2[3]
9	T6	TIMER5	0x09C	gpmc_ben0_cle	-	timer5	-	-	-	-	gpio2[5]
10	U6	TIMER6	0x098	gpmc_wen	-	timer6	-	-	-	-	gpio2[4]
11	R12	GPIO1_13	0x034	gpmc_ad13	lcd_data18	mmc1_dat5	mmc2_dat1	eQEP2B_in	pr1_mii0_txd1	pr1_pru0_pru_r30_15	gpio1[13]
12	T12	GPIO1_12	0x030	gpmc_ad12	lcd_data19	mmc1_dat4	mmc2_dat0	eQEP2A_IN	pr1_mii0_txd2	pr1_pru0_pru_r30_14	gpio1[12]
13	T10	EHRPWM2B	0x024	gpmc_ad9	lcd_data22	mmc1_dat1	mmc2_dat5	ehrpwm2B	pr1_mii0_col	-	gpio0[23]
14	T11	GPIO0_26	0x028	gpmc_ad10	lcd_data21	mmc1_dat2	mmc2_dat6	ehrpwm2_tripzone_in	pr1_mii0_txen	-	gpio0[26]
15	U13	GPIO1_15	0x03C	gpmc_ad15	lcd_data16	mmc1_dat7	mmc2_dat3	eQEP2_strobe	pr1_ecap0_ecap_capin_apwm_o	pr1_pru0_pru_r31_15	gpio1[15]
16	V13	GPIO1_14	0x038	gpmc_ad14	lcd_data17	mmc1_dat6	mmc2_dat2	eQEP2_index	pr1_mii0_txd0	pr1_pru0_pru_r31_14	gpio1[14]
17	U12	GPIO0_27	0x02C	gpmc_ad11	lcd_data20	mmc1_dat3	mmc2_dat7	ehrpwm0_synco	pr1_mii0_txd3	-	gpio0[27]
18	V12	GPIO2_1	0x08C	gpmc_clk	lcd_memory_clk	gpmc_wait1	mmc2_clk	pr1_mii1_crs	pr1_mdio_mdclk	mcasp0_fsr	gpio2[1]
19	U10	EHRPWM2A	0x020	gpmc_ad8	lcd_data23	mmc1_dat0	mmc2_dat4	ehrpwm2A	pr1_mii_mt0_clk	-	gpio0[22]
20	V9	GPIO1_31	0x084	gpmc_csn2	gpmc_be1n	mmc1_cmd	pr1_edio_data_in7	pr1_edio_data_out7	pr1_pru1_pru_r30_13	pr1_pru1_pru_r31_13	gpio1[31]
21	U9	GPIO1_30	0x080	gpmc_csn1	gpmc_clk	mmc1_clk	pr1_edio_data_in6	pr1_edio_data_out6	pr1_pru1_pru_r30_12	pr1_pru1_pru_r31_12	gpio1[30]
22	V8	GPIO1_5	0x014	gpmc_ad5	mmc1_dat5	-	-	-	-	-	gpio1[5]
23	U8	GPIO1_4	0x010	gpmc_ad4	mmc1_dat4	-	-	-	-	-	gpio1[4]
24	V7	GPIO1_1	0x004	gpmc_ad1	mmc1_dat1	-	-	-	-	-	gpio1[1]
25	U7	GPIO1_0	0x000	gpmc_ad0	mmc1_dat0	-	-	-	-	-	gpio1[0]
26	V6	GPIO1_29	0x07C	gpmc_csn0	-	-	-	-	-	-	gpio1[29]
27	U5	GPIO2_22	0x0E0	lcd_vsync	gpmc_a8	gpmc_a1	pr1_edio_data_in2	pr1_edio_data_out2	pr1_pru1_pru_r30_8	pr1_pru1_pru_r31_8	gpio2[22]
28	V5	GPIO2_24	0x0E8	lcd_pclk	gpmc_a10	pr1_mii0_crs	pr1_edio_data_in4	pr1_edio_data_out4	pr1_pru1_pru_r30_10	pr1_pru1_pru_r31_10	gpio2[24]
29	R5	GPIO2_23	0x0E4	lcd_hsync	gpmc_a9	gpmc_a2	pr1_edio_data_in3	pr1_edio_data_out3	pr1_pru1_pru_r30_9	pr1_pru1_pru_r31_9	gpio2[23]
30	R6	GPIO2_25	0x0EC	lcd_ac_bias_en	gpmc_a11	pr1_mii1_crs	pr1_edio_data_in5	pr1_edio_data_out5	pr1_pru1_pru_r30_11	pr1_pru1_pru_r31_11	gpio2[25]
31	V4	UART5_CTSN	0x0D8	lcd_data14	gpmc_a18	eQEP1_index	mcasp0_axr1	uart5_rxd	pr1_mii0_rxd3	uart5_ctsn	gpio0[10]
32	T5	UART5_RTSN	0x0DC	lcd_data15	gpmc_a19	eQEP1_strobe	mcasp0_ahclkx	mcasp0_axr3	pr1_mii0_rxdv	uart5_rtsn	gpio0[11]
33	V3	UART4_RTSN	0x0D4	lcd_data13	gpmc_a17	eQEP1B_in	mcasp0_fsr	mcasp0_axr3	pr1_mii0_rxer	uart4_rtsn	gpio0[9]
34	U4	UART3_RTSN	0x0CC	lcd_data11	gpmc_a15	ehrpwm1B	mcasp0_ahclkr	mcasp0_axr2	pr1_mii0_rxd0	uart3_rtsn	gpio2[17]
35	V2	UART4_CTSN	0x0D0	lcd_data12	gpmc_a16	eQEP1A_in	mcasp0_aclkr	mcasp0_axr2	pr1_mii0_rxlink	uart4_ctsn	gpio0[8]
36	U3	UART3_CTSN	0x0C8	lcd_data10	gpmc_a14	ehrpwm1A	mcasp0_axr0	-	pr1_mii0_rxd1	uart3_ctsn	gpio2[16]
37	U1	UART5_TXD	0x0C0	lcd_data8	gpmc_a12	ehrpwm1_tripzone_in	mcasp0_aclkx	uart5_txd	pr1_mii0_rxd3	uart2_ctsn	gpio2[14]
38	U2	UART5_RXD	0x0C4	lcd_data9	gpmc_a13	ehrpwm0_synco	mcasp0_fsx	uart5_rxd	pr1_mii0_rxd2	uart2_rtsn	gpio2[15]
39	T3	GPIO2_12	0x0B8	lcd_data6	gpmc_a6	pr1_edio_data_in6	eQEP2_index	pr1_edio_data_out6	pr1_pru1_pru_r30_6	pr1_pru1_pru_r31_6	gpio2[12]
40	T4	GPIO2_13	0x0BC	lcd_data7	gpmc_a7	pr1_edio_data_in7	eQEP2_strobe	pr1_edio_data_out7	pr1_pru1_pru_r30_7	pr1_pru1_pru_r31_7	gpio2[13]
41	T1	GPIO2_10	0x0B0	lcd_data4	gpmc_a4	pr1_mii0_txd1	eQEP2A_in	-	pr1_pru1_pru_r30_4	pr1_pru1_pru_r31_4	gpio2[10]
42	T2	GPIO2_11	0x0B4	lcd_data5	gpmc_a5	pr1_mii0_txd0	eQEP2B_in	-	pr1_pru1_pru_r30_5	pr1_pru1_pru_r31_5	gpio2[11]
43	R3	GPIO2_8	0x0A8	lcd_data2	gpmc_a2	pr1_mii0_txd3	ehrpwm2_tripzone_in	-	pr1_pru1_pru_r30_2	pr1_pru1_pru_r31_2	gpio2[8]
44	R4	GPIO2_9	0x0AC	lcd_data3	gpmc_a3	pr1_mii0_txd2	ehrpwm0_synco	-	pr1_pru1_pru_r30_3	pr1_pru1_pru_r31_3	gpio2[9]
45	R1	GPIO2_6	0x0A0	lcd_data0	gpmc_a0	pr1_mii_mt0_clk	ehrpwm2A	-	pr1_pru1_pru_r30_0	pr1_pru1_pru_r31_0	gpio2[6]
46	R2	GPIO2_7	0x0A4	lcd_data1	gpmc_a1	pr1_mii0_txen	ehrpwm2B	-	pr1_pru1_pru_r30_1	pr1_pru1_pru_r31_1	gpio2[7]

Output
Input
I/O

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Sistemas Empotrados

Master de Automática e Informática Industrial

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Guías BeagleBone Black (0)

La BeagleBone y el computador anfitrión

Rev. José Simó. Enero 2016.

Contenido

1. Objetivos

Nuestro objetivo es aprender a instalar un entorno para el desarrollo de aplicaciones que se ejecuten en la placa BeagleBone Black. El S.O. Debian instalado en la placa incluye un compilador "gcc" que nos permite generar código ejecutable trabajando directamente en la placa. Esta forma de escribir programas, aunque efectiva, no resulta cómoda. Para trabajar cómodamente, instalaremos en un computador "anfitrión" un entorno de desarrollo cruzado y un entorno gráfico IDE "Eclipse".

2. Introducción

La BeagleBone Black (BBB) es una placa programable de bajo coste que encuentra su punto fuerte en su gran versatilidad y en su capacidad para convertirse en un ordenador de bajo consumo de potencia con un sistema operativo en base Linux. El procesado de la BBB incorpora un acelerador hardware de operaciones de punto flotante. Esto es clave para ejecutar de forma eficiente el núcleo de Linux y abre un nuevo abanico de posibilidades de desarrollo de sistemas empotrado sobre Linux.

Su procesador ARM-A8 de 1GHz y su memoria RAM DDR3 de 512 MB la convierten en una unidad de proceso realmente potente en relación a su tamaño. Sin embargo, su mayor potencial se encuentra en su capacidad de conexión gracias a sus 92 pines de entrada/salida, compatibles con buses de comunicación como I2C, UART y SPI, y a sus puertos Ethernet, USB y mini HDMI.

La BBB también ofrece una unidad de memoria flash interna de 4GB y la posibilidad de ampliar su capacidad mediante una ranura para tarjetas microSD. Las principales características de la placa podemos observarlas en la siguiente Figura 1:

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

Figura 1. Características del hardware BeagleBone Black.

La placa puede soportar lo siguientes sistema operativos:

- Amstrong (distribución Linux propia para BeagleBone)
- Ubuntu/Debian (la instalada en las versiones de que se dispone)
- Android

Para una información más completa sobre el hardware y software disponible consultar beaglebone.org.

3. Preparación del computador anfitrión

Para la configuración de la BBB y el desarrollo de programas necesitamos un "computador anfitrión" al que conectar la BBB y en el que instalaremos todas las herramientas necesarias para las tareas de desarrollo: compilador cruzado, bibliotecas, entorno Eclipse, etc.

Como es habitual y recomendable en general, utilizaremos una máquina virtual (en nuestro caso sobre "VirtualBox") que hará las veces de "computador anfitrión". El uso de una máquina virtual ofrece diferentes ventajas:

- La instalación del entorno de desarrollo para la BBB en el computador anfitrión no interfiere con el software instalado en el "computador de trabajo" que usemos como plataforma (versiones de compiladores, bibliotecas y demás dependencias de software).
- Usaremos siempre Linux como sistema operativo del computador anfitrión. El "computador de trabajo" puede tener instalado cualquier sistema operativo que soporte la ejecución del software de virtualización, en nuestro caso "VirtualBox".
- Podemos guardar y replicar la instalación del computador anfitrión de una forma simple.

La configuración de los elementos del entorno de desarrollo es la que se muestra en la Figura 2.

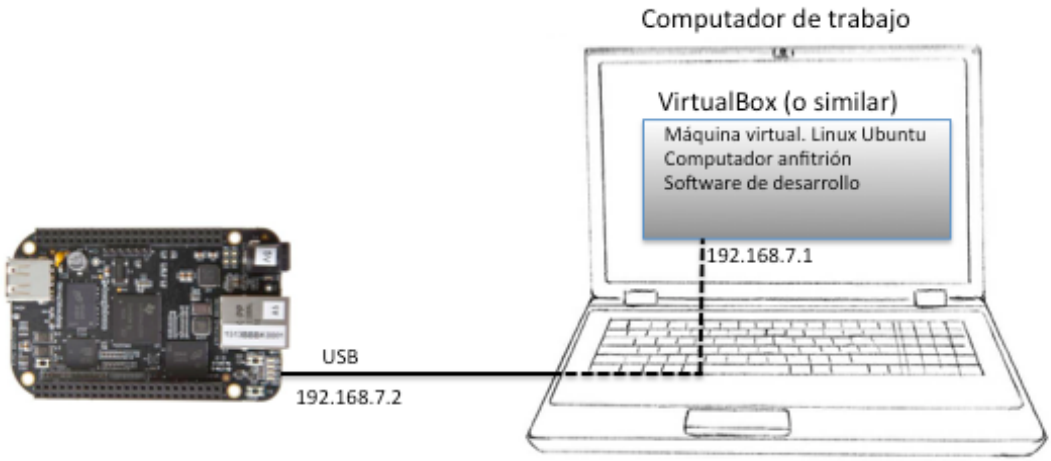


Figura 2. Computador anfitrión en una máquina virtual.

3.1. Instalación de una máquina virtual Linux

En esta sección se describe el procedimiento general para la instalación del software de virtualización "Oracle VirtualBox" en el computador de trabajo e instalación de un Linux Kubuntu en una máquina virtual sobre VirtualBox.

1. **Descargue una imagen de instalación de Linux**. Se recomienda utilizar una versión "Long Time Support" (LTS). En este manual hemos utilizado "Kubuntu 14.04" (kubuntu-14.04.2-desktop-amd64.iso).
2. **Descargue e instale en el computador de trabajo el software de virtualización**. En este manual hemos utilizado "Oracle VirtualBox".
3. **Instale la imagen de Linux descargada sobre VirtualBox**. Asigne a esta máquina virtual recursos de memoria principal, procesadores y espacio de disco de una forma generosa pero razonable con los recursos disponibles en el computador de trabajo.
4. **Arranque la máquina virtual**, compruebe que puede acceder a internet y actualice el sistema (K->Aplicaciones->Sistema->Gestor de actualizaciones)
5. **Instale drivers y extensiones de VirtualBox en la máquina virtual**

   5.1 Instale los componentes para compilar el kernel. Esto es necesario para poder instalar las "Guest Additions" en la máquina virtual.

   ```
   $ sudo apt-get install dkms
   ```

   5.2 Instale desde VirtualBox las "Guest additions".

   VirtualBox -> Devices -> Insert Guest Additions CD Image…

   5.3 Una vez montada la imagen, vaya a directorio donde se ha montado la imagen y ejecute el script de instalación (como root).

   ```
   $ sudo su –
   $ sh ./VBoxLinuxAdditions.run
   ```

6. **Defina algún directorio compartido** entre el computador de trabajo y la máquina virtual para poder trasferir archivos entre ellas con facilidad.

   6.1 Defina en VirtualBox algún "shared folder" p.e. /home/BBBuser

   VirtualBox -> Devices -> Shared Folders -> Shared Folders Settings…

   6.2 Monte el directorio compartido en la máquina virtual. Hágase un script para el futuro ya que esta operación tendrá que hacerla cada vez que reinicie la máquina virtual y quiera usar el directorio compartido.
   Ver: https://help.ubuntu.com/community/VirtualBox/SharedFolders

   ```
   $ sudo mkdir /mnt/BBBuser
   $ sudo mount -t vboxsf -o uid=$UID,gid=$(id -g) BBBUser /mnt/BBBuser
   ```

7. También es conveniente que active el la interacción a través del "clipboard" para poder cortar/pegar elementos entre la máquina virtual y el computador de trabajo.

   VirtualBox -> Devices -> Shared Clipboard -> Bidirectional

3.2. Instalación de los entornos de desarrollo nativos y Eclipse

1. **Instalación de Java** (en este caso se instala Java 8). Es necesario tener instalado un JRE para poder ejecutar Eclipse. En nuestro caso instalaremos directamente un JDK por si en un futuro queremos desarrollar aplicaciones Java en el computador anfirión.

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java8-installer
//y las variables de entorno:
$ sudo apt-get install oracle-java8-set-default
```

2. **Instalación de C y C++** para desarrollar aplicaciones que se ejecuten en el computador anfitrión.

```
$ sudo apt-get install build-essential
```

3. Instalación de Eclipse-CDT

3.1 Visite la página de descargas de Eclipse (p.e. http://www.eclipse.org/downloads /packages/eclipse-ide-cc-developers/mars1) y descargue la distribución de eclipse-cdt adecuada. P.e. eclipse-cpp-XXX.tar.gz

3.2 Descomprima el archivo descargado en /opt/eclipse_cdt

```
$ cd /opt
$ sudo tar -zxvf ~/Downloads/eclipse-XXX.tar.gz
$ sudo mv eclipse eclipse_cdt
```

3.3 Cree una entrada en el menú de “aplicaciones"

Instale la utilidad para lanzar aplicaciones como root (gksudo).

```
$ sudo apt-get install gksu
```

Lance un editor de texto para crear el archivo “/usr/share/applications /eclipse_cdt.desktop”

```
$ gksudo kate /usr/share/applications/eclipse_cdt.desktop
```

Con el contenido...

```
[Desktop Entry]
Name=Eclipse4-CDT
Type=Application
Exec=/opt/eclipse_cdt/eclipse
Icon=/opt/eclipse_cdt/icon.xpm
Comment=Eclipse IDE for C and C++
NoDisplay=false
Categories=Development;IDE;
Name[en]=Eclipse
```

Ya se puede arrancar eclipse desde el menú de aplicaciones. En un futuro puede usar el mismo procedimiento para instalar otros entornos de desarrollo de Eclipse (p.e. J2EE) descomprimiendo el archivo de distribución en otro subdirectorio de "/opt" que cree para tal efecto.

3.3. Instalación del entorno de desarrollo cruzado

Un entorno de desarrollo cruzado es el conjunto de compiladores y bibliotecas necesarios para generar código ejecutable para una determinada arquitectura que es diferente de la arquitectura del computador donde realizamos la compilación. En nuestro caso queremos desarrollar código para procesadores "ARM", utilizando los lenguajes C y C++ en una máquina que se ejecuta sobre x86.

1. Añada los repositorios necesarios (editar el fichero sources.list)

```
$ gksudo kate /etc/apt/sources.list
```

Añadir al final:

```
#Emdebian entries
deb http://www.emdebian.org/debian unstable main
deb http://ftp.us.debian.org/debian unstable main contrib non-free
```

2. instale las claves de firma para producir archivos "release" y que Debian los maneje fácilmente.

```
$ sudo apt-get install emdebian-archive-keyring
$ sudo apt-get update
```

3. Instale las biblioteca de "ARM Hard Floating Point"

```
$ sudo apt-get install build-essential //este puede que ya esté instalado
$ sudo apt-get install libc6-armhf-cross
$ sudo apt-get install libc6-dev-armhf-cross
$ sudo apt-get install binutils-arm-linux-gnueabihf
$ sudo apt-get install linux-libc-dev-armhf-cross
$ sudo apt-get install libstdc++6-armhf-cross
```

4. Busque e instale los compiladores cruzados...

Busque la ultima versión del compilador cruzado mediante la orden:

```
$ sudo apt-cache search gnueabihf
```

….. En nuestro caso, encontramos gcc-4.8-arm-linux-gnueabihf y g++-4.8-arm-linux-gnueabihf y los instalamos

```
$ sudo apt-get install gcc-4.8-arm-linux-gnueabihf
$ sudo apt-get install g++-4.8-arm-linux-gnueabihf
```

5. Compruebe que funciona:

```
$ arm-linux-gnueabihf-gcc-4.8 -v
y/o
$ arm-linux-gnueabihf-g++-4.8 -v
```

4. Conexión de la BBB al computador anfitrión

Una vez instalada la maquina virtual pasamos a conectar la placa a uno de los conectores USB del ordenador con el cable proporcionado (Ver Figura 2). Veremos como el computador de trabajo inicia un servicio de visualización de los ficheros del sistema de archivos de la placa. NO aceptar dicha acción. En su lugar conecte mediante el menú de la aplicación Virtual Box , el dispositivo al sistema Linux virtual (computador anfitrión).

Conecte la BBB al puerto USB
VirtualBox -> Machine -> Settings
Ports -> Pestaña USB
Botón “Add”
Seleccione “Circuitco BeagleBoneBlack”
Aceptar.

Veremos como el sistema Linux virtual nos permite acceso al sistema de archivos de la placa al tiempo que los leds de la misma se iluminan intermitentemente. Esto se realiza mediante un driver “network over USB” que está instalado de forma nativa en los sistemas Linux/Ubuntu recientes, y que asigna una IP local a un dispositivo conectado por USB. En otros sistemas operativos (Mac OS, Windows) habrá que instalar el driver de forma especifica.

Para comprobar que realmente se ha conectado correctamente la placa abrir un visor web, y conectarse con la dirección IP 192.168.7.2. Si se abre una pagina web con información de la Beaglebone, la conexión ha sido correcta.

5. Probando el compilador cruzado

1. En el computador anfitrión abra un editor de text (p.e. kate), escriba el siguiente programa y guárdelo con el nombre “hello.c”.

```
#include <stdio.h>
int main() {
   printf("Hello BBB\n");
}
```

2. Compile el programa

```
$ arm-linux-gnueabihf-gcc-4.8 -o hello hello.c
$ ls
hello  hello.c
$ ./hello
bash: ./hello: cannot execute binary file: Exec format error
```

Si intenta ejecutar el programa, obtendrá un error porque el código ejecutable generado no se corresponde con la arquitectura subyacente al computador anfitrión.

3. Conecte la BBB al computador anfitrión.

4. Copie el ejecutable a la BBB y pruébelo

```
$ ls
hello  hello.c
$ scp hello root@192.168.7.2:/root
…
$ ssh -l root 192.168.7.2
# cd /root
# ./hello
Hello BBB
```

6. Configuración de Eclipse para el desarrollo cruzado

Eclipse es un entorno de desarrollo (Integrated Development Environment) desarrollado en Java (y por tanto multiplataforma), ampliable, expansible, modular y que abarca muchas fases del desarrollo de sistemas, desde el diseño hasta el despliegue (en ingles deployment), en gran cantidad de lenguajes y para gran cantidad de sistemas (desde sistemas distribuidos, en la nube, web services, hasta sistemas empotrados como el que manejamos). Esta versatilidad y potencia lo hacen extremadamente útil, al tiempo que requiere una cuidadosa configuración para que haga lo que queremos y no nos frustre el trabajo.

6.1. Instalación de los componentes de eclipse necesarios

Lanzaremos la aplicación Eclipse de la que dispondremos de un icono en el menú de aplicaciones.

A continuación aceptaremos el *workspace* por defecto que nos proporciona. El *workspace* es el directorio de trabajo donde se guardaran los proyectos y ficheros de trabajo.

Antes de iniciar el desarrollo de un nuevo proyecto debemos instalar un pluging adicional. Los plugings son complementos que nos permiten incluir alguna funcionalidad nueva a Eclipse. En concreto nos interesa el pluging RSE (**Remote System Explorer**).

Para instalar nuevo software acudir al menú de "**Help -> Install New Software**" (Figura 3). En la ventana en la que indica las palabras de búsqueda incluir el repositorio donde buscarlo, en nuestro caso es "Mars - http://download.eclipse.org/releases/mars"

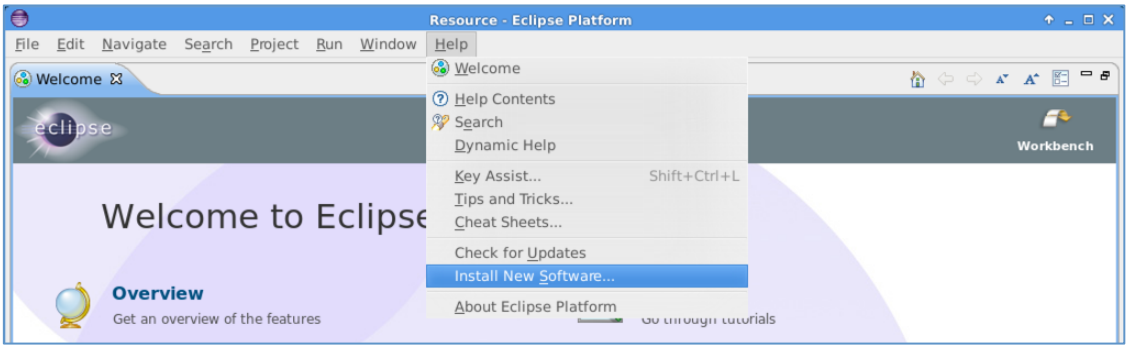


Figura 3.

Una vez localizados los plugings que coinciden con las palabras clave (ver imagen inferior), seleccionar el primero de ellos (**General Purpose Tools -> Remote System Explorer End-User Runtime**), aceptar la instalación (ver Figura 4).

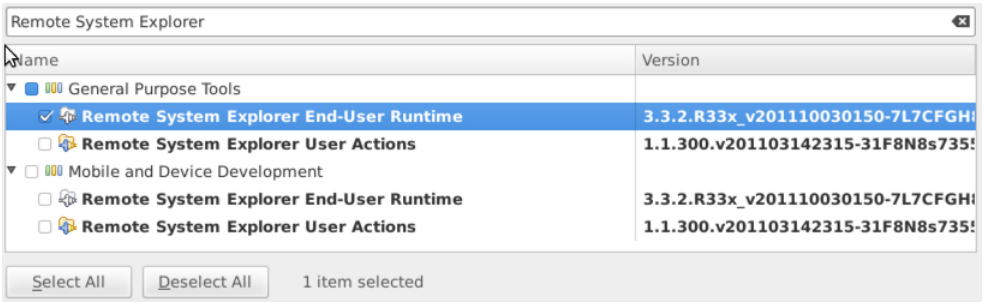


Figura 4.

Una vez terminada la instalación del mismo deberemos aceptar el **reinicio del entorno Eclipse** para permitir su habilitación.

6.2. Configuración de un proyecto Eclipse de desarrollo cruzado

Realizados todos estos pasos podremos pasar a iniciar un proyecto. Para ello seleccionamos la opción “new Project” de Eclipse

Para ello vamos a menú **File -> New->Project** el cual desplegará un menú en el que vemos toda la relación de proyectos que podemos crear con las herramientas instaladas. En nuestro caso queremos un proyecto C/C++

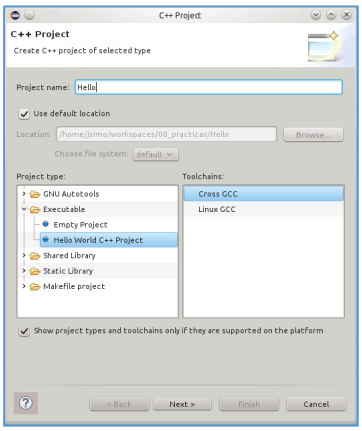


Figura 5.

Durante el proceso de creación del proyecto hay que configurar una serie de propiedades para conseguir que el código fuente que hemos creado (en este caso de forma automática), genere el código ejecutable para el target que deseamos.

En “Cross GCC Command” poner

- “Cross settings prefix:” a `arm-linux-gnueabihf-`
- “Cross compiler path” a `/usr/arm-linux-gnueabihf` (se puede buscar en el sistema de archivos con el botón “Browse”).

Una vez creado el proyecto, tiene que cambiar algunas propiedades más seleccionando el proyecto y buscando la opción de propiedades (**Project->Properties ; C/C++ Build/Settings**) para obtener el menú que nos permite configurarlas.

En **C/C++ Build/Settings…** pestaña "Tool Settings" (Ver Figura 6)

- Seleccione "**Cross Settings**" si quiere cambiar las propiedades que definió durante el proceso de creación del proyecto (prefix y path)
- Seleccione "**Cross GCC Compiler**" y establezca el valor de "Command" a "gcc-4.8" (teniendo en cuenta la versión del compilador que hemos instalado en la sección anterior)
- Seleccione "**Cross GCC Compiler/Includes**" y establezca un valor del path como "/usr/arm-linux-gnueabihf/include"
- Seleccione "**Cross G++ Compiler**" y establezca el valor de "Command" a "g++-4.8"
- Seleccione "**Cross G++ Compiler/Includes**" y establezca un valor del path como "/usr/arm-linux-gnueabihf/include"
- Seleccione "**Cross G++ Linker**" y establezca el valor de "Command" a "g++-4.8"
- Seleccione "**Cross G++ Linker/Libraries**" y establezca el valor de "Library search path" a "/usr/arm-linux-gnueabihf/lib"

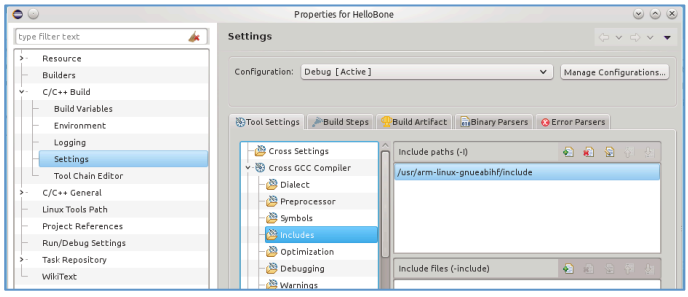
Añada al proyecto un archivo de código fuente "hello.c" con un contenido similar al que se ha usado para probar el compilador en la sección anterior.

Si todo es correcto al ejecutar la opción de menú "**Project -> Build All**" deberíamos obtener en la consola un mensaje de que se ha compilado todo sin errores.

Nota: La orden de compilación que utiliza Eclipse se genera concatenando el valor de la propiedad "prefix" con los valores de "Command" para cada caso. Tenga en cuenta qué entorno de desarrollo ha instalado para establecer estos valores (revise la sección de este documento "Instalación del entorno de desarrollo cruzado").

Nota: Posiblemente, para no tener que especificar el valor "Command" para cada nuevo proyecto que cree, prefiera establecer los siguientes enlaces simbólicos:

```
$ sudo ln -s /usr/bin/arm-linux-gnueabihf-gcc-4.8 /usr/bin/arm-linux-gnueabihf-gcc
$ sudo ln -s /usr/bin/arm-linux-gnueabihf-g++-4.8 /usr/bin/arm-linux-gnueabihf-g++
```



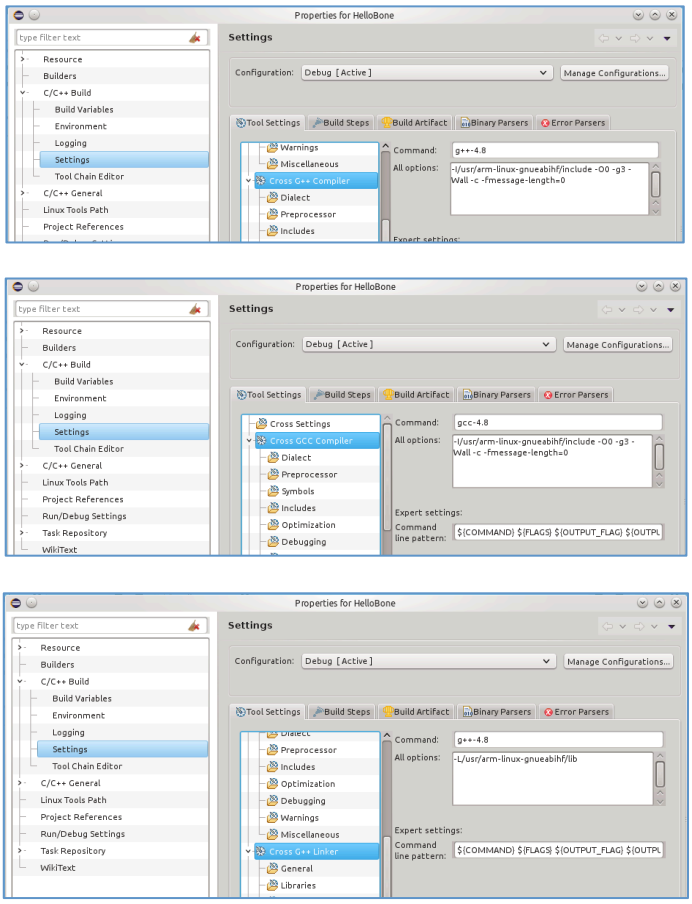


Figura 6: Algunos ejemplos de las pantallas de configuración del proyecto para la compilación cruzada.

6.3. Configuración de "Remote System Explorer"

EL RSE (Remote System Explorer) es un pluging que permite abrir conexiones de comunicaciones sobre el target (en nuestro caso la placa Beaglebone) utilizando un canal de comunicación (en este caso IP). Para comenzar a trabajar con él,

- Seleccione **Window -> Show View -> Other**
- Escriba "Remote" en el cuado de filtro
- Seleccione "Remote Systems" y pulse "OK" (Ver Figura 7)

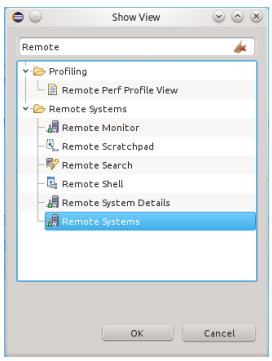


Figura 7:

Se abrirá en la parte baja del entorno una nueva pestaña para definir un nuevo RSE. En el árbol de conexiones aparecerá la del sistema local. Sobre ella pulsando el botón derecho del ratón se desplegará un menú que nos permite definir una nueva conexión (Ver Figura 8).

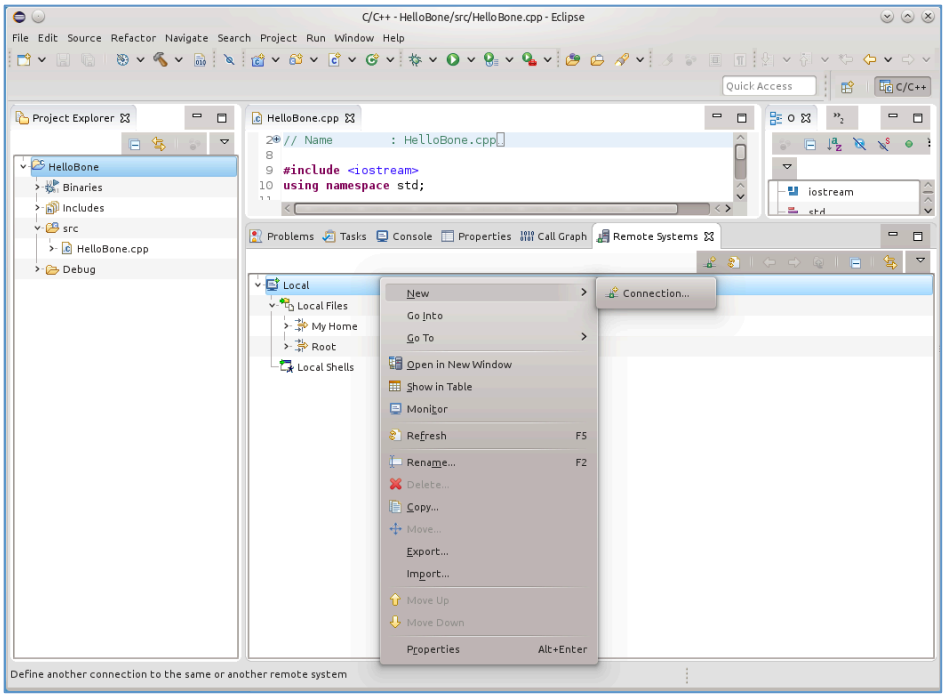


Figura 8:

Los parámetros que hay que establecer en la nueva conexión son:

1. **Remote System Type**: SSH Only
2. **Remote SSH Only System Connection**:
   Host name: 192.168.7.2
   Connection Name: BeagleBone Black USB
3. **Sftp Files**: ssh.files
4. **Ssh Shells:** ssh.shells
5. **Ssh Terminals:** ssh.terminals

Una vez definidos estos parámetros podemos ir a la opción de menú para configurar las opciones de ejecución de la aplicación. Básicamente hay que seleccionar un perfil basado en el RSE que acabamos de definir.

Para hacer esto seleccionar "**Run -> Run Configurations…**" seleccionar "C/C++ Remote Application" de la lista, y pulsar el icono "**New launch configuration**". Con esto obtendremos una ventana de configuración como la siguiente (Figura 9):

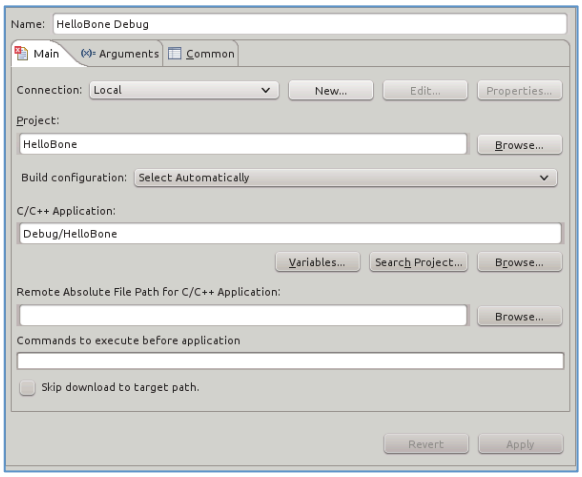


Figura 9:

- Seleccione en el campo "**Connection**" el valor "BeagleBone Black USB" (o en su defecto el nombre que le haya asignado a la conexión) .
- Especifique una **ruta absoluta** en el sistema de archivos de la BBB para ubicar programa, por ejemplo "/root/HelloBone"
- Pulse "Apply" para guardar la configuración.
- Pulse "Run" para ver por la consola la ejecución a través de la conexión SSH al sistema remoto (BBB).

La primera vez que hace una conexión, el sistema le solicitará el usuario y la clave para acceder al sistema remoto (ver Figura 10).

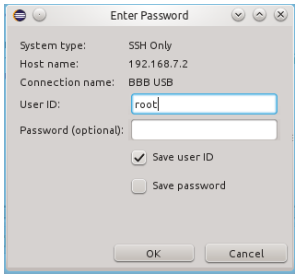


Figura 10:

El resultado de la ejecución se presenta en la pestaña "Console" en la que se abre una conexión SSH contra la BBB (ver Figura 11).

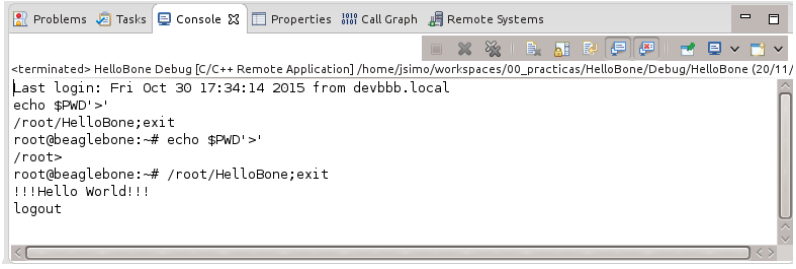


Figura 11:

7. Conclusiones

Siguiendo las explicaciones e indicaciones de este documento habrá conseguido instalar un Sistema Operativo “anfitrión” para la BBB que incluye un entorno gráfico “Eclipse-CDT” que permite el desarrollo cruzado de aplicaciones en lenguaje “C” para la BBB. Habrá aprendido a configurar proyectos de desarrollo cruzado en “Eclipse” y a ejecutarlos en la BBB sin salir del entorno gráfico.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Sistemas Empotrados

Master de Automática e Informática Industrial

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Guías BeagleBone Black (I)

Conectar con la Consola del Sistema

Rev. José Simó. Enero 2016.

Contenido

1. Objetivos

En algunos casos es necesario acceder a la información que la "BeagleBone Black" (BBB) nos ofrece durante el proceso de arranque y poder interactuar con ella a través de la consola de sistema.

Nuestro objetivo es aprender a establecer una conexión de consola de sistema en la BBB y configurar correctamente los programas de emulación de terminal del computador anfitrión para poder interactuar con el Sistema Operativo de la BBB.

2. Introducción

La consola de sistema la incluyen todos los sistemas Linux. Es el dispositivo por el que el Sistema Operativo comunica los mensajes de error y, en general, cualquier información sobre su funcionamiento. Independientemente del estado del computador, si el SO está funcionando acepta el inicio de sesiones por esta consola (login). Cuando todo falla este suele ser el último recurso que tiene el administrador del sistema para acceder a él y manipularlo.

La BBB ofrece la consola de sistema a través de un puerto serie TTL que se encuentra en el conjunto de pines "Serial Debug" (Ver Figura 1) . Por defecto, en la BBB este puerto está configurado con los siguientes parámetros: **Velocidad: 115200; Número de bits: 8; Paridad: ninguna; Bits de stop: 1.** Estos valores son los que deberemos usar al configurar nuestro programa de emulación de terminal más adelante.

Para conectar un computador a este puerto necesitaremos: Cable USB-Serial TTL y un programa de emulación de terminal en el computador anfitrión (host).

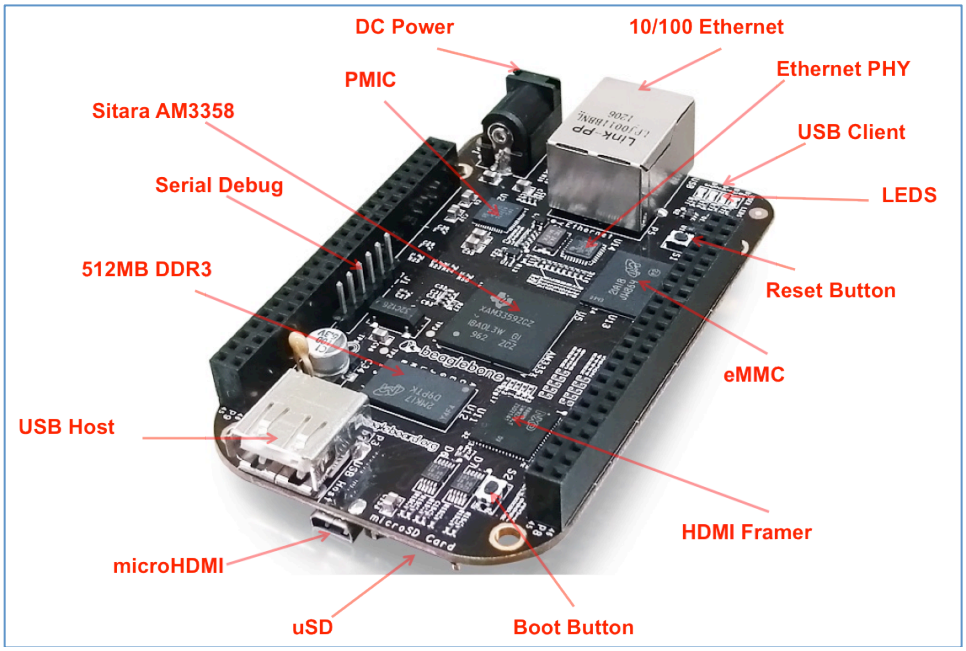


Figura 1: Vista rápida de la placa BeagleBone Black.

3. El Cable

En nuestro caso disponemos de un cable “Embest UART8000-U” USB a TTL 3-pin con el chip FTDI FT230X que soporta un rango de velocidades entre 300bps a 3Mbps. El cable se conecta a los terminales JTAG de la BBB de la siguiente manera (Ver Figura 2).

Pin 1….. Negro…….Masa (0V)
Pin 2….. No conectado
Pin 3….. No conectado
Pin 4….. Rojo……..Transmisión (Tx)
Pin 5….. Verde……..Recepción (Rx)
Pin 6….. No conectado

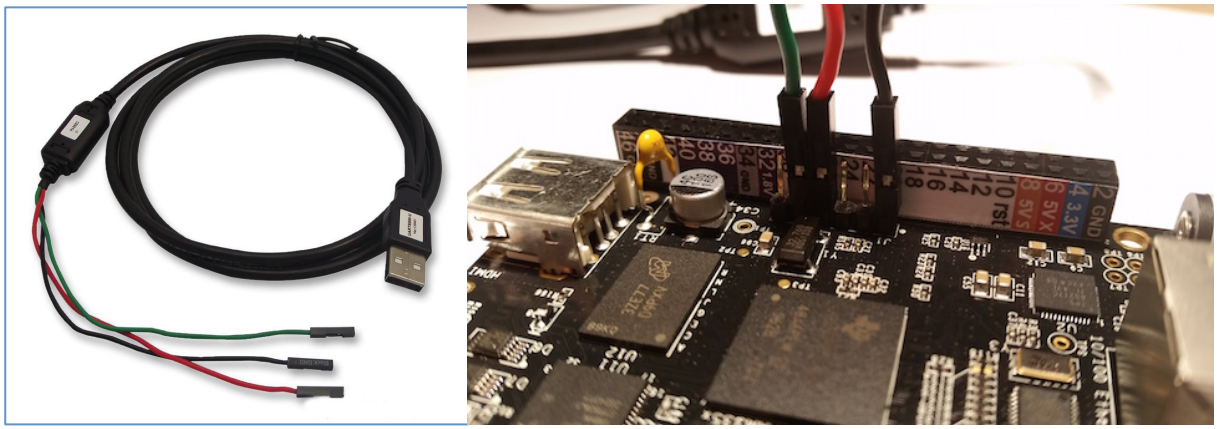


Figura 2: Cable FTDI “Embest UART8000-U USB TO 3 PIN TTL“ y detalle de conexión a la BBB.

4. La terminal en el anfitrión

El computador anfitrión debe tener instalados los manejadores de dispositivos correspondientes al cable que estemos usando (los drivers FTDI). Para instalar el manejador de dispositivo puede obtener una gama completa de drivers para "Virtual COM Port" (VCP) en la página web de "Future Technology Devices International” (FTDI) (http://www.ftdichip.com/Drivers/VCP.htm).

Escoja el que corresponda al Sistema Operativo del computador que use como “host” e instálelo. Por ejemplo, en MacOS la versión 2.3 del “FTDIUSBSerialDriver” soporta los dispositivos: FT8U232AM, FT8U245AM, FT232BM, FT245BM, FT2232, FT232R, FT245R, FT2232H, FT4232H, FT232H y FT X Series. En Linux Ubuntu más modernos que 11.10, todos los dispositivos FTDI están soportados de forma nativa. **En Linux, encontraremos el puerto serie-USB en /dev/ttyUSB0**. En MacOS, los dispositivos en el sistema de ficheros empiezan por /dev/tty.usbserial, por ejemplo, /dev/tty.usbserial-AD01U7TH.


Si está usando como computador host una máquina virtual VirtualBox (p.e. Linux Ubuntu), debe configurar la máquina virtual para asignarle el uso del dispositivo USB-Serie. Esto se hace en VirtualBox así:

- Conecte el cable USB-Serie a un puerto USB.
- Arranque VirtualBox y seleccione la máquina virtual.
- ***Machine*** -> ***Settings*** > ***Ports*** -> ***USB***
- Añadir un filtro (icon "+" de la barra lateral) e indique el dispositivo en cuestión. Ver Figura 3
- Arranque el SO Linux anfitrión de la máquina virtual.

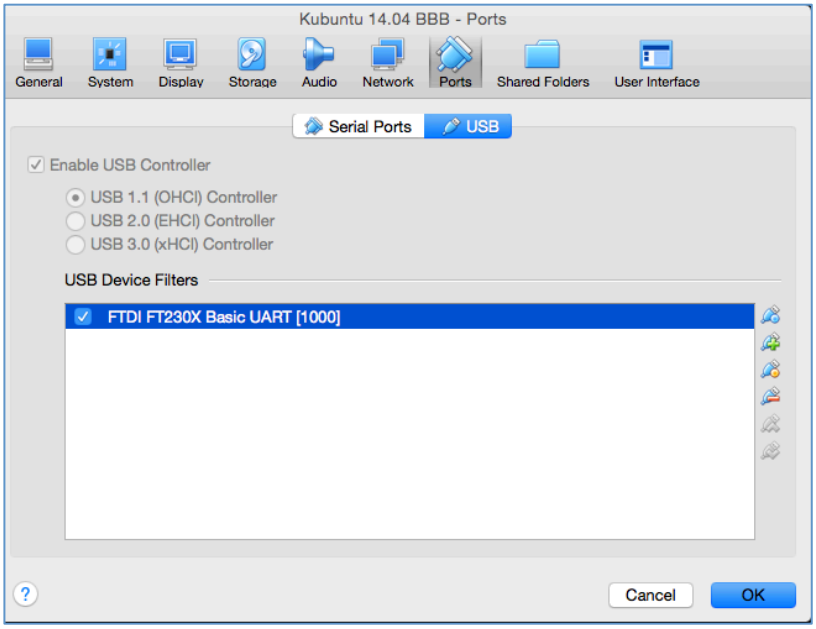


Figura 3: Si usa VirtualBox debe asignar a la VM el dispositivo USB.

Una vez haya arrancado la MV, inicie sesión, abra una consola y compruebe que el dispositivo /dev/ttyUSB0 se encuentra en el su lugar mediante la orden:

```
$ ls –l /dev/ttyU*
```

Para que, en el computador anfitrión, cualquier usuario pueda acceder al puerto USB-serie, debemos modificar los permisos del archivo "dev", por ejemplo así:

```
$ sudo chmod a+rw /dev/ttyUSB0
```

Si desconectamos el cable USB y lo volvemos a conectar, los permisos se restablecen y debemos volver a cambiarlos. La alternativa es lanzar el programa de emulación de terminal con "sudo".

En Linux puede usar diferentes emuladores de terminal. El más clásico es "screen" que puede instalarlo con la orden:

```
$ sudo apt-get install screen
```

y arrancar una sesión contra la BBB así:

$ screen /dev/ttyUSB0 115200 8N1

Consulte las páginas de manual de “screen”. Por ejemplo, para salir del programa pulse Crtl-a y teclee “:quit” (sin las comillas).

También puede utilizar aplicaciones específicas como “gtkterm” que se instalan desde el “Gestor de aplicaciones de Ubuntu (Ver Figura 4).

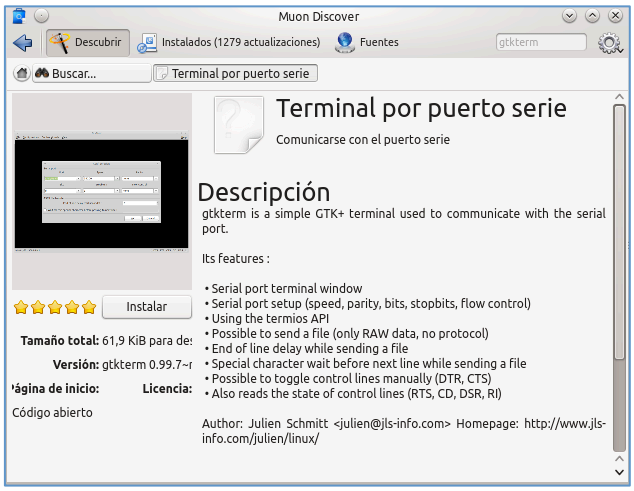


Figura 4: Puede instalar el programa “gtkterm”.

5. Conectándose

- Con la BBB apagada, conecte el cable correctamente al los pines “Serial Debug”.
- Si no estaba ya arrancado, arranque el computador anfitrión.
- Conecte el cable a un puerto USB del computador anfitrión (si usa una máquina virtual, el dispositivo USB tiene que estar asignado a ella).
- En el computador anfitrión, cambie los permisos del dispositivo con la orden:
  $ sudo chmod a+rw /dev/ttyUSB0
- Arranque correctamente configurado el programa de emulación de terminal. Por ejemplo, si usa “screen” en Linux, introduciendo la orden:
  $ screen /dev/ttyUSB0 115200 8N1
- Conecte ahora la alimentación de la BBB (“DC Power”, Figura 1). Si todo ha ido bien, podrá observar los mensajes de la BBB en la terminal semejantes a los que se ven en la Figura 5.

Supongamos ahora un caso hipotético:

Hemos conectado la BBB por el puerto “10/100 Ethernet” a una red con servicio DHCP y no sabemos qué dirección IP se le ha asignado a la BBB. Necesitamos saber la IP para poder conectarnos a la BBB mediante “ssh”. ¿Cómo averiguamos la IP?... Siempre podemos conectarnos al router y consultar las tablas de asignación, pero lo más cómodo es:

- En la consola de sistema de la BBB, lacemos “login” como “root” (por defecto no tiene clave)
- Una vez el sistema nos haya dado el “Shell”, introducimos la orden:
  $ ifconfig

El resultado es el que se muestra en la Figura 6. Comprobamos que la IP que tenemos asignada es 192.168.1.56.

Podemos seguir operando desde la consola de sistema. Seguro que en algún momento futuro lo considerará imprescindible, pero a partir de ahora, como la BBB está conectada a la red, lo más cómodo será usar “ssh”.

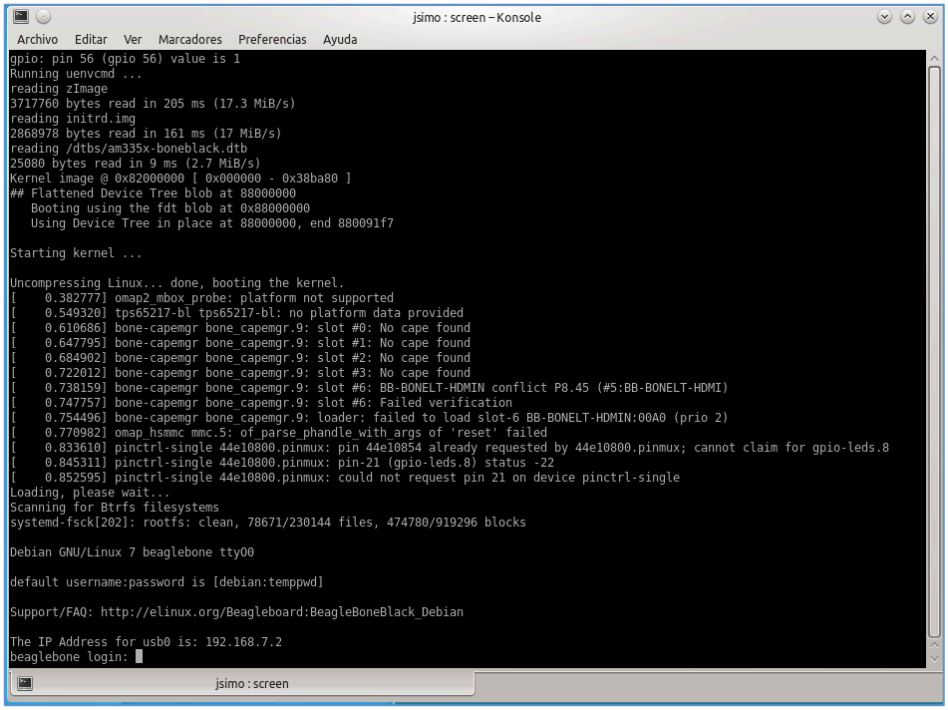


Figura 5: Terminal conectada a la consola de sistema de la BBB.

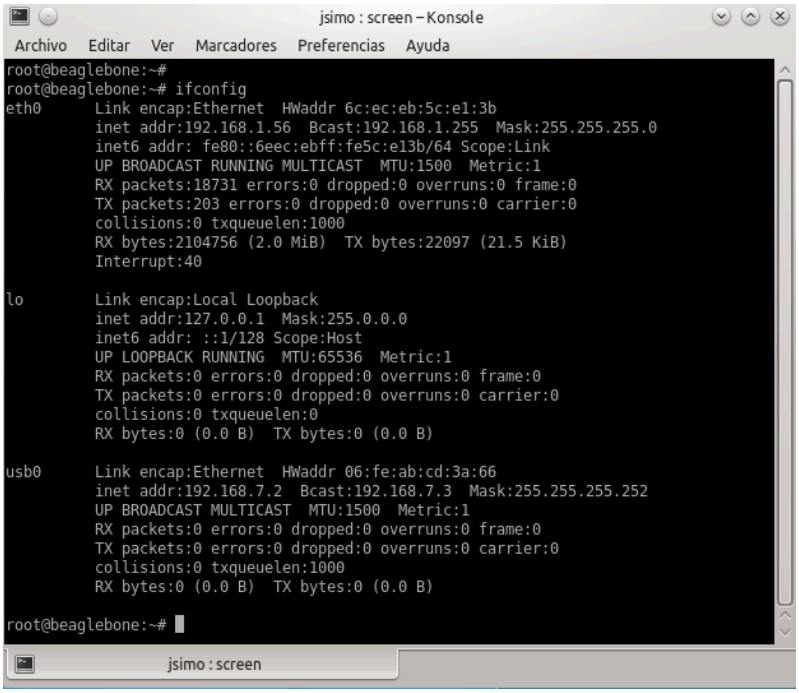


Figura 6: Consultar la dirección IP asignada.

6. Anexo I. Mensajes de arranque

Esto es un ejemplo de los mensajes de arranque del SO Debian capturado desde la consola de sistema.

```
U-Boot SPL 2014.04-00014-g47880f5 (Apr 22 2014 - 13:23:54)
reading args
spl_load_image_fat_os: error reading image args, err - -1
reading u-boot.img
reading u-boot.img


U-Boot 2014.04-00014-g47880f5 (Apr 22 2014 - 13:23:54)

I2C:   ready
DRAM:  512 MiB
NAND:  0 MiB
MMC:   OMAP SD/MMC: 0, OMAP SD/MMC: 1
*** Warning - readenv() failed, using default environment

Net:   <ethaddr> not set. Validating first E-fuse MAC
cpsw, usb_ether
Hit any key to stop autoboot:  0
gpio: pin 53 (gpio 53) value is 1
Card did not respond to voltage select!
mmc0(part 0) is current device
Card did not respond to voltage select!
gpio: pin 56 (gpio 56) value is 0
gpio: pin 55 (gpio 55) value is 0
gpio: pin 54 (gpio 54) value is 0
mmc1(part 0) is current device
gpio: pin 54 (gpio 54) value is 1
SD/MMC found on device 1
reading uEnv.txt
1430 bytes read in 4 ms (348.6 KiB/s)
gpio: pin 55 (gpio 55) value is 1
Loaded environment from uEnv.txt
Importing environment from mmc ...
Checking if uenvcmd is set ...
gpio: pin 56 (gpio 56) value is 1
Running uenvcmd ...
reading zImage
3717760 bytes read in 206 ms (17.2 MiB/s)
reading initrd.img
2868978 bytes read in 160 ms (17.1 MiB/s)
reading /dtbs/am335x-boneblack.dtb
25080 bytes read in 9 ms (2.7 MiB/s)
Kernel image @ 0x82000000 [ 0x000000 - 0x38ba80 ]
## Flattened Device Tree blob at 88000000
   Booting using the fdt blob at 0x88000000
   Using Device Tree in place at 88000000, end 880091f7

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
[    0.382606] omap2_mbox_probe: platform not supported
[    0.549451] tps65217-bl tps65217-bl: no platform data provided
[    0.614615] bone-capemgr bone_capemgr.9: slot #0: No cape found
[    0.651723] bone-capemgr bone_capemgr.9: slot #1: No cape found
[    0.688830] bone-capemgr bone_capemgr.9: slot #2: No cape found
[    0.725940] bone-capemgr bone_capemgr.9: slot #3: No cape found
[    0.742093] bone-capemgr bone_capemgr.9: slot #6: BB-BONELT-HDMIN conflict P8.45 (#5:BB-BONELT-HDMI)
[    0.751701] bone-capemgr bone_capemgr.9: slot #6: Failed verification
[    0.758451] bone-capemgr bone_capemgr.9: loader: failed to load slot-6 BB-BONELT-HDMIN:00A0 (prio 2)
[    0.774946] omap_hsmmc mmc.5: of_parse_phandle_with_args of 'reset' failed
[    0.837546] pinctrl-single 44e10800.pinmux: pin 44e10854 already requested by 44e10800.pinmux; cannot claim for gpio-leds.8
[    0.849259] pinctrl-single 44e10800.pinmux: pin-21 (gpio-leds.8) status -22
[    0.856542] pinctrl-single 44e10800.pinmux: could not request pin 21 on device pinctrl-single
Loading, please wait...
Scanning for Btrfs filesystems
systemd-fsck[202]: rootfs: clean, 78668/230144 files, 474512/919296 blocks

Debian GNU/Linux 7 beaglebone ttyO0

default username:password is [debian:temppwd]

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

The IP Address for usb0 is: 192.168.7.2
beaglebone login:
```

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Sistemas Empotrados

Master de Automática e Informática Industrial

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Guías BeagleBone Black (II)

Arranque y Sistema Operativo

Rev. José Simó. Enero 2016.

Contenido

1. Objetivos

El objetivo de este manual es aprender a actualizar la imagen del sistema operativo de la BBB con las imágenes más actuales que descargaremos de Internet.

También aprenderemos preparar la BBB para que arranque desde una tarjeta microSD y a transferir y manejar las imágenes del sistema operativo tanto si estas residen en la MMC interna como en la microSD externa.

2. Introducción

La BBB normalmente se usa como un "Sistema Linux Empotrado". Como no existe una versión del núcleo de Linux especial para sistemas empotrados, cuando nos referimos a un "Sistema Linux Empotrado" queremos decir simplemente un Linux instalado en un sistema empotrado.

El SO que utilizaremos es Linux Debian para ARM con aceleración hardware para operaciones de punto flotante ("hf").

La BBB, tal y como la encontramos al desembalarla, incluye una imagen del SO Debian instalada en la memoria flash interna "eMMC". Preparando adecuadamente el contenido de una tarjeta de memoria microSD, podemos sobrescribir el contenido de la eMMC y así instalar versiones más recientes del SO. Por otro lado, también existe la posibilidad de instalar un SO en la microSD y configurar la BBB para que arranque desde ella.

Antes de describir estas operaciones, veamos cómo se produce el arranque de la BBB.

Para ejecutar la secuencia de arranque, los computadores de escritorio disponen de un programa almacenado en una memoria flash interna y parámetros de configuración almacenados en una memoria alimentada por una pila de botón. Antiguamente este programa era el BIOS (*Basic Input/Output System*), actualmente es el UEFI (*Unified Extensible Firmware Interface*). En ambos casos, la secuencia de arranque consiste básicamente en: tomar el control del procesador, inicializar el hardware, cargar la imagen del SO y pasar el control al SO.

Los sistemas empotrados en general y la BBB en particular, no tienen UEFI/BIOS. Utilizan una combinación de programas "cargadores" (bootloader).

Los procesadores de Texas Instruments ARM Cortex (AM335x) disponen de un pequeño programa cargador almacenado en una memoria ROM en el interior del procesador. Llamaremos a este programa "Boot ROM", "cargador de primera etapa" o "cargador interno".

Cuando se arranca el sistema, la CPU salta al vector de interrupción de "reset", que está definido para apuntar a la dirección de memoria de inicio del "Boot ROM".

El “Boot ROM” tiene información de cómo acceder a la eMMC, la microSD, la UART, etc. Realiza una configuración mínima de los periféricos y accede a ellos en busca del “cargador de secunda etapa”, el “x-loader” (o MLO), lo carga y la pasa el control.

El “x-loader” o MLO, configura la asignación de pines, inicializa la memoria y el reloj, carga el “cargador de tercera etapa”, el “U-Boot”, y le pasa el control.

El “U-Boot” especifica el sistema de ficheros raíz y, atendiendo a la configuración establecida en el fichero “uEnv.txt”, inicializa el sistema, carga el núcleo de Linux y le pasa el control.

El núcleo de Linux, se descomprime en memoria, inicializa el resto de periféricos, monta el sistema de fichero raíz y arranca el primer proceso en espacio de usuario: el “init”. Con esto concluye el arranque del sistema. La Figura 1 ilustra este proceso.

Tanto el código del “BootROM” como el “x-loader” los proporciona Texas Instruments. El “u-boot” (Universal Bootloader) es un proyecto de código abierto (“Das u-boot”) adaptado para la arquitectura ARM de la BBB. Las múltiples opciones de arranque que puede ofrecer la BBB se consiguen especificando las opciones y órdenes correspondientes en el fichero “uEnv.txt”.

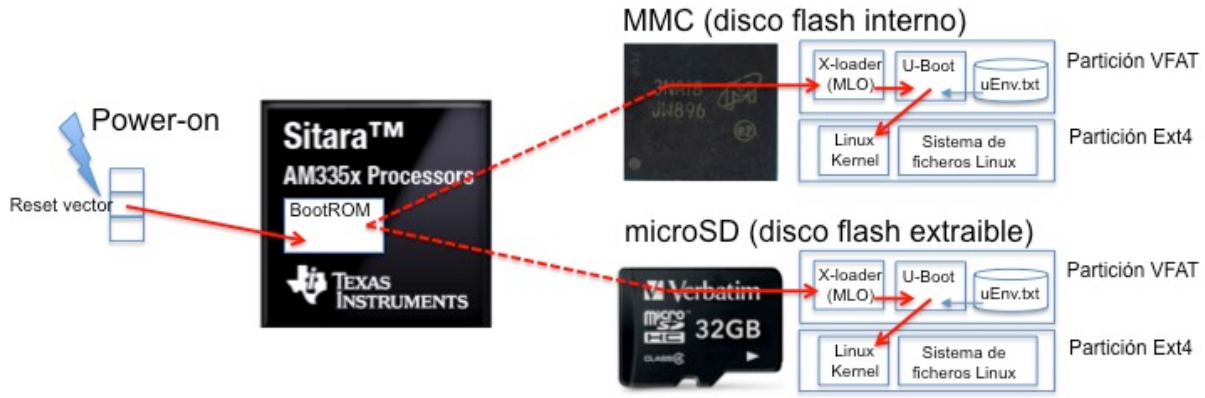


Figura 1: Secuencia de arranque de la BBB

La BBB puede arrancar desde el disco flash interno (MMC) o bien desde una tarjeta microSD que insertemos en la ranura correspondiente. En cualquier caso, un disco de arranque debe contener dos particiones: una en formato VFAT que alberga los cargadores de segunda y tercera etapa y otra en formato Linux (Ext4) con la imagen del sistema operativo. Esta estructura está definida por Texas Instruments y documentada en el capítulo 26 del manual “*AM335x SitaraTM Processors. Technical Reference Manual*”.

Podemos ver el particionado de los discos así:

```
$ fdisk -l
Disk /dev/mmcblk0: 3867 MB, 3867148288 bytes
4 heads, 16 sectors/track, 118016 cylinders, total 7553024 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

```
Disk identifier: 0x00000000

Device          Boot     Start         End      Blocks   Id  System
/dev/mmcblk0p1   *        2048      198655       98304    e  W95 FAT16 (LBA)
/dev/mmcblk0p2          198656     7553023     3677184   83  Linux

Disk /dev/mmcblk0boot1: 2 MB, 2097152 bytes
4 heads, 16 sectors/track, 64 cylinders, total 4096 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000
Disk /dev/mmcblk0boot1 doesn't contain a valid partition table

Disk /dev/mmcblk0boot0: 2 MB, 2097152 bytes
4 heads, 16 sectors/track, 64 cylinders, total 4096 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000
Disk /dev/mmcblk0boot0 doesn't contain a valid partition table
```

Nota: para observar los discos presentes en el sistema, sus particiones y puntos de montaje, pruebe a utilizar también las órdenes:

```
$ df -T
$ df -h
$ lsblk
$ mount -l
$ cat /proc/partitions
```

En versiones anteriores de la Beaglebone, si queríamos arrancar desde la unidad microSD, teníamos que mantener pulsado el botón “Power” mientras conectábamos la alimentación de la placa.

Actualmente (distribuciones con Linux Debian 7.5 o superiores), es mucho más simple: Si hay insertada una tarjeta microSD, el “BootROM” arranca desde ella, si no hay tarjeta o no encuentra la cadena de cargadores, pasa a arrancar desde la MMC interna.

Se puede preparar una imagen del disco de arranque válida tanto para ubicarla en la microSD como en la MMC interna. Para hacerlo es necesario seguir las especificaciones de Texas Instruments y utilizar los cargadores específicos (parcheados) MLO y U-Boot, un entorno de compilación cruzado y una imagen del Sistema Operativo que se quiera instalar. Se puede encontrar una explicación de este proceso en https://eewiki.net/display/linuxonarm/BeagleBone+Black

En nuestro caso, no prepararemos la imagen de arranque, nos limitaremos a descargar las imágenes que prepara y mantiene la comunidad de usuarios y fabricantes. Esta comunidad es muy activa y prepara imágenes mejoradas y actualizadas prácticamente todos los meses.

3. Imágenes del disco de arranque

Se pueden encontrar las últimas imágenes de arranque para la BBB en:

https://rcn-ee.com/rootfs/bb.org/release/

y su explicación en:

http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

visite también la página oficial de la Beaglebone:

http://beagleboard.org/latest-images

Todas las imágenes están pensadas para ubicarlas en una tarjeta microSD y arrancar la BBB desde ella. Básicamente, con diferentes variantes, hay dos tipos de imágenes:

- **Imagen "Flasheable":** Cuando arrancamos la BBB desde la microSD con una imagen de este tipo, la imagen de arranque se copia en la MMC interna. Se usa para actualizar el software de la BBB.
- **Imagen de arranque:** Se usan para arrancar la BBB desde la microSD, usar el sistema de ficheros raíz de la microSD y trabajar directamente con el sistema operativo instalado en esta unidad.

Durante la fase de desarrollo es recomendable la segunda opción: arrancar la BBB desde la microSD y trabajar con el sistema de ficheros de la microSD. Las ventajas son muchas: no estamos limitados a la capacidad de almacenamiento de la MMC interna, podemos usar tarjetas microSD de alta velocidad que superan con mucho el rendimiento de la MMC interna y, si durante la fase de desarrollo dañamos el sistema de ficheros, siempre podemos sustituir la tarjeta microSD, restaurar su imagen desde un PC o arrancar desde la MMC para realizar labores de mantenimiento y reparación.

Una vez finalizada la fase de desarrollo de nuestro sistema empotrado, para ahorrar costes, podemos usar la imagen de la microSD para volcarla en la MMC interna (siempre que no exceda del límite de almacenamiento de la MMC interna).

3.1. Preparación de la tarjeta microSD

El procedimiento de descarga y volcado de una imagen es sencillo y se puede realizar en un computador anfitrión Linux realizando los siguientes pasos:

1. Descargar la imagen que necesitemos:

```
$ wget https://rcn-ee.com/rootfs/bb.org/release/2015-10-30/lxde-4gb/BBB-eMMC-
flasher-debian-7.9-lxde-4gb-armhf-2015-10-30-4gb.img.xz
```

En este caso estamos descargando una imagen completa, con entorno de ventanas "light X11" que ocupa 4Gb y "flasheable", es decir, que al arrancarla se usará para actualizar la MMC interna.

2. Descomprimir la imagen

```
$ xz -d BBB-eMMC-flasher-debian-7.9-lxde-4gb-armhf-2015-10-30-4gb.img.xz
```

Nota: si no tiene instalada la utilidad “xz”, instalela con “$ sudo apt-get install xz-utils”

Esto generará el correspondiente fichero “.img”

3. Identificar la unidad de disco correspondiente a la tarjeta microSD.

Para ello, en el computador anfitrión, antes de insertar la tarjeta microSD, listamos los dispositivos de disco detectados en el sistema así:

```
$ ls -l /dev/sd*
brw-rw---- 1 root disk 8, 0 nov  3 16:04 /dev/sda
brw-rw---- 1 root disk 8, 1 nov  3 16:04 /dev/sda1
brw-rw---- 1 root disk 8, 2 nov  3 16:04 /dev/sda2
brw-rw---- 1 root disk 8, 5 nov  3 16:04 /dev/sda5
```

Ahora introducimos la tarjeta microSD en el lector conectado al computador anfitrión y repetimos la operación.

```
$ ls -l /dev/sd*
brw-rw---- 1 root disk 8,  0 nov  3 16:04 /dev/sda
brw-rw---- 1 root disk 8,  1 nov  3 16:04 /dev/sda1
brw-rw---- 1 root disk 8,  2 nov  3 16:04 /dev/sda2
brw-rw---- 1 root disk 8,  5 nov  3 16:04 /dev/sda5
brw-rw---- 1 root disk 8, 16 nov  3 16:07 /dev/sdb
brw-rw---- 1 root disk 8, 17 nov  3 16:07 /dev/sdb1
```

Observamos que el dispositivo que aparece al insertar la tarjeta es “/dev/sdb”.

Antes de realizar el volcado, hay que asegurarse de que todas las particiones que tenga definidas el dispositivo están desmontadas. En el caso del ejemplo, el dispositivo tiene definida una partición (/dev/sdb1) que podemos desmontar así:

```
$ umount /dev/sdb1
```

4. Volcar la imagen a la tarjeta microSD usando como destino el dispositivo identificado (el “raw” no las particiones que pudiera tener)

```
$ dd if=BBB-eMMC-flasher-debian-7.9-lxde-4gb-armhf-2015-10-30-4gb.img of=/dev/sdb
bs=1M
```

Esta operación de volcado tarda unos minutos. Una vez finalizada la operación podemos extraer la tarjeta y volverla a introducir para observar el resultado de la operación (ahora el disco /dev/sdb tiene dos particiones)

```
$ ls -l /dev/sd*
brw-rw---- 1 root disk 8,  0 nov  3 16:04 /dev/sda
brw-rw---- 1 root disk 8,  1 nov  3 16:04 /dev/sda1
brw-rw---- 1 root disk 8,  2 nov  3 16:04 /dev/sda2
brw-rw---- 1 root disk 8,  5 nov  3 16:04 /dev/sda5
brw-rw---- 1 root disk 8, 16 nov  3 16:18 /dev/sdb
brw-rw---- 1 root disk 8, 17 nov  3 16:18 /dev/sdb1
brw-rw---- 1 root disk 8, 18 nov  3 16:18 /dev/sdb2

$ lsblk
```

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0    15G  0 disk
├─sda1   8:1    0    11G  0 part /
├─sda2   8:2    0     1K  0 part
└─sda5   8:5    0     4G  0 part [SWAP]
sdb      8:16   1   3,7G  0 disk
├─sdb1   8:17   1    96M  0 part
└─sdb2   8:18   1   3,2G  0 part
sr0     11:0    1    56M  0 rom
```

Podemos repetir las instrucciones anteriores para generar una tarjeta microSD con una imagen de arranque descargando el archivo correspondiente:

```
$ wget https://rcn-ee.com/rootfs/bb.org/release/2015-10-30/lxde-4gb/bone-debian-
7.9-lxde-4gb-armhf-2015-10-30-4gb.img.xz
$ xz -d bone-debian-7.9-lxde-4gb-armhf-2015-10-30-4gb.img.xz
$ dd if= bone-debian-7.9-lxde-4gb-armhf-2015-10-30-4gb.img of=/dev/sdb bs=1M
```

Si hemos seguido todas estas operaciones, ahora disponemos de dos tarjetas microSD:

- **microSD-Flasher**: Arranca la BBB y actualiza el sistema operativo de la MMC interna.
- **microSD-Boot**: Arranca la BBB y usa el sistema de ficheros de la microSD.

Etiquete ambas tarjetas y téngalas a mano.

Nota: para la "microSD-Flasher", es suficiente con que utilice una tarjeta microSD de 4Gb de capacidad. Si tiene tarjetas de mayor capacidad, es mejor que las use para la "microSD-Boot".

3.2. Actualización de la imagen del disco MMC interno de la BBB

Para actualizar la MMC interna de la BBB necesitaremos una "microSD-Flasher" con la imagen del sistema que queramos instalar.

1. Con la BBB apagada, inserte la tarjeta "microSD-Flasher" en la ranura microSD de la BBB.
2. Para observar los mensajes del sistema durante el proceso, puede acceder a la "consola de sistema" usando un cable USB-SerialTTL conectado al computador anfitrión y un programa de emulación de terminal. Consulte el manual correspondiente.
3. Conecte la alimentación de la BBB
4. La BBB arranca desde la microSD, si ha conectado la consola de sistema, podrá leer los mensajes de progreso. El arranque inicia automáticamente el proceso de actualización de la MMC interna. Durante la actualización, los leds de usuario de la BBB (del cero al cuatro) se iluminan secuencialmente de forma ascendente y descendente siguiendo el patrón conocido como "visor de Cylon" o "frontal de Kit".
5. Cuando termina el volcado de la nueva imagen en la MCC interna, la BBB se apaga automáticamente.
6. Extraiga la tarjeta "microSD-Flasher" y guárdela.
7. Conecte la BBB y observe que tiene un nuevo sistema operativo (`$ uname -a`) y que todo lo que tenía antes ha desaparecido.

4. Arrancar el sistema desde la microSD

Tal y como se ha comentado antes, es conveniente que en la fase de desarrollo del sistema empotrado, arranque desde la microSD. Como el sistema de ficheros también estará ubicado en la microSD, podrá configurar el sistema e instalar componentes a voluntad. Tener el software de nuestro sistema empotrado en un medio extraíble como la microSD nos permite realizar copias de seguridad cuando lo creamos conveniente y replicar el sistema con facilidad.

Para arrancar la BBB desde la microSD necesitamos una "microSD-Boot". Siempre que la tarjeta esté introducida en la ranura microsSD de la BBB, esta arrancará desde ella.

4.1. Expandir el sistema de ficheros de la microSD

Al volcar la imagen "microSD-Boot" habrá observado que, independientemente de la capacidad original de la tarjeta SD, el tamaño de la unidad de disco resultante es de 4Gb. Este tamaño coincide con el tamaño de la MMC interna de la BBB y esa es la razón por la que las imágenes que descargamos están preparadas para generar una tarjeta microSD de ese tamaño.

Si hemos utilizado para generar la "microSD-Boot" una microSD de mayor capacidad, por ejemplo 32Gb, lo más normal es que queramos utilizar toda la capacidad de la tarjeta en el sistema de ficheros del Linux de la BBB.

Para expandir el sistema de ficheros de la partición Linux de la "microSD-Boot" seguiremos los siguientes pasos:

1. Arrancar desde la SD: Inserte la "microSD-Boot" en la ranura microSD de la BBB y conecte la alimentación. Abra una terminal (la consola con el cable USB-SerialTTL o con ssh) e ingrese en el sistema con el usuario "root".
2. Identifique la unidad y partición de arranque

```
$ ls -l /dev/mmcblk*
brw-rw---T 1 root floppy 179,  0 Mar  1 21:06 /dev/mmcblk0
brw-rw---T 1 root floppy 179,  1 Mar  1 21:06 /dev/mmcblk0p1
brw-rw---T 1 root floppy 179,  2 Mar  1 21:06 /dev/mmcblk0p2
brw-rw---T 1 root floppy 179,  8 Mar  1 21:06 /dev/mmcblk1
brw-rw---T 1 root floppy 179, 16 Mar  1 21:06 /dev/mmcblk1boot0
brw-rw---T 1 root floppy 179, 24 Mar  1 21:06 /dev/mmcblk1boot1
brw-rw---T 1 root floppy 179,  9 Mar  1 21:06 /dev/mmcblk1p1
brw-rw---T 1 root floppy 179, 10 Mar  1 21:06 /dev/mmcblk1p2
```

Al listar los dispositivos observamos que aparecen tanto la unidad microSD como la MMC interna con idéntica estructura. La unidad de arranque (la microSD) es "mmcblk0" (puede usar la orden "mount" para comprobarlo)

3. Con "fdisk", listar las particiones de la unidad, borrar la partición de Linux y crear una nueva usando todo el disco:

```
$ fdisk /dev/mmcblk0
```

pulsar "p" para ver las particiones.

pulsar “d” para eliminar una partición.
pulsar “2” para seleccionar la partición 2 (Linux) a eliminar.
pulsar “n” para crear una nueva partición.
pulsar “p” para que sea primaria.
pulsar “2” para que sea la partición 2.
seleccionar los valores de defecto para usar todo el espacio de disco en la partición
pulsar “w” para guardar la tabla de particiones.

4. Reiniciar la BBB

```
$ reboot
```

5. Ver el tamaño del disco, redimensionarlo y ver el resultado:

```
$  df -h
Filesystem      Size  Used Avail Use% Mounted on
rootfs          3.5G  1.8G  1.6G  54% /
udev             10M     0   10M   0% /dev
tmpfs           100M  440K   99M   1% /run
/dev/mmcblk0p2  3.5G  1.8G  1.6G  54% /
tmpfs           249M     0  249M   0% /dev/shm
tmpfs           249M     0  249M   0% /sys/fs/cgroup
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           100M     0  100M   0% /run/user
```

Observamos que el tamaño del disco sigue siendo el original (en este caso 3.5Gb). Tenemos que redimensionar el sistema de ficheros para que se ajuste a la partición que hemos editado así:

```
$ resize2fs /dev/mmcblk0p2
resize2fs 1.42.5 (29-Jul-2012)
Filesystem at /dev/mmcblk0p2 is mounted on /; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 2
The filesystem on /dev/mmcblk0p2 is now 7622144 blocks long.
```

Ahora podemos comprobar que el tamaño del disco ha cambiado.

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
rootfs           29G  1.8G   26G   7% /
udev             10M     0   10M   0% /dev
tmpfs           100M  616K   99M   1% /run
/dev/mmcblk0p2   29G  1.8G   26G   7% /
tmpfs           249M     0  249M   0% /dev/shm
tmpfs           249M     0  249M   0% /sys/fs/cgroup
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           100M     0  100M   0% /run/user
/dev/mmcblk1p1   96M   75M   22M  78% /media/boot
/dev/mmcblk0p1   96M   66M   31M  69% /media/BEAGLEBONE
/dev/mmcblk1p2  3.4G  1.8G  1.5G  55% /media/rootfs
```

5. Arrancar el sistema desde la MMC interna

Para arrancar el sistema desde la MMC interna simplemente conecte la alimentación de la BBB asegurándose de que la ranura microSD no tiene ninguna tarjeta insertada.

En esta situación puede usar una tarjeta microSD como espacio de almacenamiento adicional. Para ello formatee la tarjeta microSD en formato FAT32, insértela en la ranura microSD de la BBB, cree un punto de montaje (p.e. `$ mkdir /media/store`), identifique el dispositivo correspondiente a la unidad microSD (`$ ls /dev/mmc* , $ mount`) y monte la unidad (`$ mount /dev/mmcblkXpX`).

El dispositivo asignado a la microSD es diferente si arranca la BBB con la tarjeta microSD insertada o la inserta en caliente cuando la BBB ya ha arrancado.

El procedimiento es similar para montar unidades de memoria USB.

6. Replicar la imagen del sistema

Tanto si está trabajando durante la fase de desarrollo arrancando desde la MMC interna como si lo hace desde la unidad microSD, habrá instalado bibliotecas, software de terceros y software propio. También habrá configurado servicios y otros detalles del Sistema Operativo para que el sistema empotrado BBB se comporte como desea.

En esta situación puede necesitar transferir la imagen de instalación completa entre unidades en una misma BBB o copiarla para que funcione en otras BBB.

Si ha estado trabajando arrancando el sistema con la "microSD-Boot", puede replicarla de forma sencilla insertando la tarjeta en un lector conectado al computador anfitrión y extrayendo la imagen de la unidad con la orden "dd". Así podrá realizar tantas copias como quiera o simplemente hacer copias de seguridad de las diferentes fases de un proyecto en desarrollo.

Para realizar diferentes operaciones entre la MMC y la microSD, la instalación de Debian para la BBB contiene en el directorio "/opt/scripts/tools/eMMC" *scripts* específicos. Para encontrar una descripción de los scripts disponibles, lea el archivo "/opt/scripts/tools/eMMC/readme.md".

6.1. Copiar la imagen actual de la "microSD-Boot" a la MMC interna.

Esta operación permite copiar el sistema que tengamos en la "microSD-Boot" a la MMC. Una vez copiada, podremos extraer la tarjeta microSD y arrancar nuestro sistema desde la memoria interna. Es útil para desplegar aplicaciones en la BBB una vez hemos terminado la fase de desarrollo ya que nos ahorramos el coste de la tarjeta de memoria en el sistema final.

En la BBB, ejecutaremos el script "init-eMMC-flasher-v3.sh" en modo "single user" desmontando previamente todas las particiones excepto la del sistema de ficheros raíz con el que hayamos arrancado.

Realice las siguientes operaciones desde la consola de sistema conectándose a la BBB con un cable USB-SerialTTL y utilizando un programa de emulación de terminal.

1. Pasamos a modo “single user”:

```
$ /sbin/init 1
```

2. Comprobamos el nivel de ejecución actual

```
$ runlevel
```

3. Desmontamos todas las particiones excepto la de sistema

```
$ umount /dev/mmcblk0p1
$ umount /dev/mmcblk1p1
$ umount /dev/mmcblk1p2
```

4. Ejecutamos el script

```
$ cd /opt/scripts/tools/eMMC
$ ./init-eMMC-flasher-v3.sh
```

Puede observar el progreso de proceso siguiendo los mensajes que imprimen por la consola de sistema.

6.2. Crear una “microSD-Flasher” con la imagen actual de la MMC interna

De una forma similar al proceso anterior y utilizando el script “beaglebone-black-make-microSD-flasher-from-eMMC.sh” puede crear una “microSD-Flasher” que contenga la imagen actual de la MMC.

7. Conclusiones

Después de haber trabajado con esta guía habrá aprendido a instalar el Sistema Operativo que mejor se ajuste a sus necesidades tanto en una tarjeta de memoria externa microSD como en la memoria interna de MMC de la BBB. Como las operaciones de instalación y volcado de imágenes del SO no se hacen con frecuencia, este documento le resultará útil para futuras referencias.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Sistemas Empotrados

Master de Automática e Informática Industrial

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Guías BeagleBone Black (III)
Configuración de Red

Rev. José Simó. Enero 2016.

Contenido

1. Objetivos

En este documento se supone que el SO instalado en la BBB. Si no es así, siga el procedimiento de instalación de Linux Debian en la "BeagleBone Black" (BBB). En muchos casos la BBB viene con un SO Linux Debian pre-instalado.

Cuando encendemos la BBB, se inicia el proceso de arranque del sistema. Podemos ver los mensajes de arranque e ingresar en el sistema (login) conectándonos a la consola de la BBB que se ofrece en un puerto serie TTL cuyos pines están en los terminales "Serial Debug" (ver Figura 1). Con la consola de sistema podemos realizar operaciones básicas, pero durante el desarrollo de aplicaciones es necesario que la BBB esté conectada en red con el computador anfitrión y, como no, a Internet.

Nuestro objetivo es configurar el sistema de la BBB en red, con acceso a internet y controlando los servicios que se inician durante el arranque.

2. Introducción

La BBB nos ofrece tres posibilidades para conectarse a la red:

- A través del interfaz estándar de red de cable RJ-45 (ver Figura 1 – 10/100 Ethernet).
- Mediante el controlador de red integrado en el puerto USB cliente (ver Figura 1 - USB Client).
- Utilizando un adaptador WiFi conectado al puerto USB.(ver Figura 1 - USB Host).

La primera configuración, la de red estándar, es la que normalmente se utiliza en los sistemas durante la explotación. La segunda configuración, la del controlador de red integrado en el puerto USB cliente, es muy cómoda y se utiliza normalmente durante la fase de desarrollo. La configuración WiFi es adecuada para redes domésticas, no es posible la conexión a redes WPA2-Enterprise (como UPVNET).

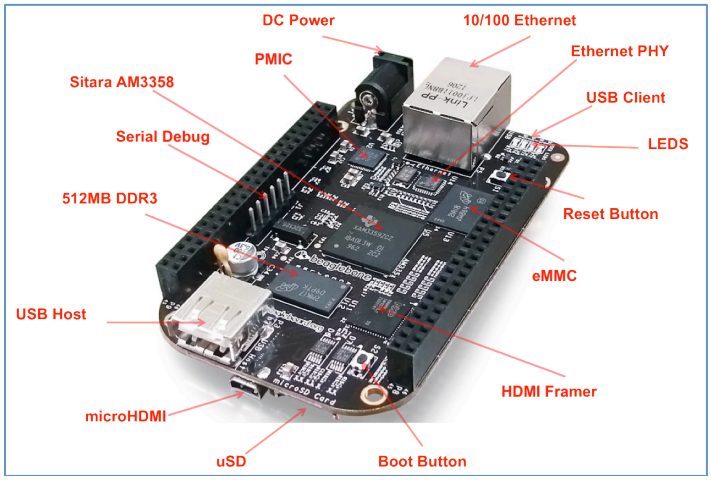


Figura 1: Vista rápida de la placa BeagleBone Black.

3. Conectando la BBB a una red con router

La primera opción de conexión es la que se muestra en la Figura 2. En este caso, conectaremos el puerto “10/100 Ethernet” de la BBB al router (o a un switch que esté conectado al router). La BBB está configurada por defecto para obtener una dirección IP de forma dinámica (DHCP) y suponemos que el router tiene el servicio DHCP activado. En este escenario, cuando alimentamos la BBB, arranca y obtiene una dirección IP de forma dinámica. Si el router está conectado a Internet, la BBB tendrá acceso a Internet.

El problema es que tendremos que averiguar qué dirección IP tiene asignada la BBB. Esto se puede hacer accediendo a la configuración del router y consultando las tablas de asignación o accediendo a la BBB a través de la “consola de sistema” con un cable SerieTTL-USB. Una vez ingresado en el sistema con el usuario “root”, ejecutar la orden “ifconfig”. Puede encontrar una descripción detallada de este procedimiento en el documento “BeagleBone Black. Conectar con la consola del sistema”.

Una vez conozcamos la dirección IP asignada a la BBB (supongamos que es 192.168.1.56), podemos abrir una terminal “ssh” desde el computador de trabajo con la siguiente orden:

```
$ ssh -l root 192.169.1.56
```

También puede copiar archivos entre la BBB y el computador de trabajo con la orden “scp”. Por ejemplo:

```
$ scp FicheroComputadorAnfitrion root@192.168.1.56:/cualquier/Directorio/De/La/BBB
```

Consulte el manual de “scp” para obtener detalles de la sintaxis (puede usar la opción “-r” para copiar de forma recursiva).

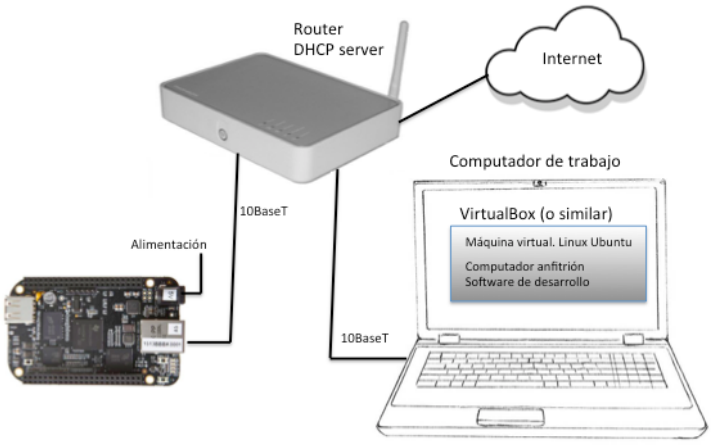


Figura 2: Conectando la BBB directamente a una red con router.

3.1. Asignación de una IP estática

Cuando estamos configurando una red de dispositivos empotrados (para el caso que nos ocupa, dispositivos BBB), en muchas ocasiones es conveniente asignar direcciones IP fijas a los dispositivos. En esta sección se describe el procedimiento para hacerlo.

En primer lugar, para averiguar la pasarela de defecto (default Gateway) de la red podemos usar la orden "route –n". Por ejemplo, para aislar la información que buscamos, podemos usar:

```
$ route -n | grep "^0.0.0.0" | tr -s " " | cut -f2 -d" "
```

Supongamos que esta orden nos ha informado que la pasarela por defecto es "192.168.1.1".

Nota: con el objetivo de familiarizarse con las ordenes "shellscript", revise las páginas de manual de "grep" (filtrar las líneas que contengan el patrón especificado, aquí el carácter "^" indica el principio de línea), "tr" (operación "translate", cambia unos caracteres por otros, la opción –s indica "que aparezca sólo una vez consecutiva el patrón especificado), "cut" ("cortar", obtener una sub-cadena, en el ejemplo obtenemos el segundo campo usando el delimitador "espacio en blanco").

Para averiguar la dirección IP asignada por el servidor DHCP, puede usar la orden:

```
$ ifconfig
```

Observe la información que proporciona esta orden y busque la dirección correspondiente al interfaz "eth0". Anote también la dirección MAC (HWaddr) para futuras referencias.

Supongamos que esta orden nos ha informado que la dirección IP asignada por el servidor DHCP es "192.168.1.56".

Nuestro objetivo es asignar una dirección IP estática. En primer lugar tenemos que decidir qué dirección vamos a utilizar. El criterio a aplicar es de "evitar colisiones", es decir, que tenemos que hacer una lista de las IP que queremos asignar a nuestros dispositivos y no repetir ninguna. Además, supongamos que nuestra BBB tiene que convivir en la red con otros dispositivos configurados para obtener su dirección por DHCP. Es posible que si asignamos de forma estática la dirección "192.168.1.56" o una próxima, el servidor DHCP, asigne nuestra dirección a otro dispositivo. Esto producirá una colisión. Tenemos que escoger una dirección IP alejada del rango que el servidor DHCP usa (p.e. 192.168.1.200). Para ser estrictos, deberíamos acceder a la configuración de nuestro router e indicar al servidor DHCP que no use el rango de direcciones que vamos a usar para asignaciones estáticas. Con todo esto, hemos decidido configurar la BBB con los siguientes parámetros:

IP: 192.168.1.200
Mask: 255.255.255.0
Gateway: 192.168.1.1

La configuración en sí es simple. Tenemos que editar el archivo **/etc/network/interfaces**

```
$ more /etc/network/interfaces
# This file describes the network interfaces available on your system
…
# The primary network interface
#allow-hotplug eth0
```

```
iface eth0 inet dhcp
# Example to keep MAC address between reboots
#hwaddress ether DE:AD:BE:EF:CA:FE
…
```

Utilizando un editor de terminal como “vi” ó “nano” modifique el archivo para que contenga esta información:

```
…
# The primary network interface
#iface eth0 inet dhcp
auto eth0
iface eth0 inet static
address 192.168.1.200
netmask 255.255.255.0
gateway 192.168.1.1
# Example to keep MAC address between reboots
#hwaddress ether DE:AD:BE:EF:CA:FE
…
```

Si fuera necesario añadir algún DNS, edite el archivo **/etc/resolv.conf**. Por ejemplo, un contenido razonable para este achivo es:

```
$ more /etc/resolv.conf
nameserver 8.8.8.8
```

La dirección 8.8.8.8 corresponde al servidor de nombres de google que debería funcionar de una forma fiable en todas las configuraciones de red.

Adicionalmente, puede cambiar el nombre de la máquina así:

```
$ more /etc/hostname
beaglebone
$ echo "BBB01" > /etc/hostname
$ more /etc/hostname
BBB01
```

Al reiniciar el sistema, los cambios tendrán efecto

```
$ shutdown –h now
```

Escriba en una etiqueta la dirección IP, la dirección MAC y el nombre de la máquina. Pegue esta etiqueta a la BBB para distinguirla de las demás.

Arranque la BBB y acceda a ella usando la nueva IP:

```
$ ssh –l root 192.169.1.200
$ uname -n
BBB01
```

3.2. Conexión a una red WiFi

Para conectar a una red WiFi necesita un adaptador WiFi USB conectado a la BBB. Debido a la energía que consume este tipo de adaptadores es conveniente conectar la alimentación de 5V de la BBB o conectar el adaptador a través de un repetidor USB con alimentación independiente. En

cualquier caso es recomendable usar un cable extensor USB para evitar que la propia placa de la BBB dificulte la recepción de las ondas de radio.

Utilice la versión del núcleo de Linux para la BBB lo más moderna que pueda. Las nuevas versiones incluyen manejadores para más variedad de dispositivos y mejoran la estabilidad.

Una vez conectado el adaptador correctamente, compruebe que lo ha reconocido el SO mediante la orden:

```
$ iwconfig
wlan0     IEEE 802.11bgn  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=0 dBm
          Retry  long limit:7   RTS thr=2347 B   Fragment thr:off
          Encryption key:off
          Power Management:on

lo        no wireless extensions.

eth0      no wireless extensions.

usb0      no wireless extensions.
```

En este caso, el nombre del adaptador es wlan0,

Edite el archivo **/etc/network/interfaces** y elimine los comentarios las líneas de la sección “WiFi Example” de manera que queden así:

```
# WiFi Example
auto wlan0
iface wlan0 inet dhcp
    wpa-ssid "essid"
    wpa-psk  "password"
```

Utilice los valores del “ssid” y “password” correspondientes a la red a la que se quiera conectar.

Active la conexión manualmente así:

```
$ ifup wlan0
```

Compruebe al funcionamiento de la red

```
$ ifconfig wlan0
$ ping 8.8.8.8
```

También de puede desactivar la conexión manualmente así:

```
$ ifdown wlan0
```

4. Conectando la BBB a una red por el USB cliente

La conexión de la BBB utilizando el interfaz “Remote Network Driver Interface Specification” (RNDIS) de Ethernet sobre USB es la que puede apreciarse en la Figura 3. Como hemos dicho antes, esta configuración es la más adecuada para las fases de desarrollo.

En este escenario, consideraremos que el computador de trabajo está ejecutando una máquina virtual (sin perdida de generalidad, supondremos que se usa Oracle VirtualBox) con el S.O. Linux KUbuntu instalado. Esta máquina virtual es la que llamaremos “Computador Anfitrión” porque en ella tendremos instaladas todas las herramientas necesarias para realizar, compilar y desplegar programas para la BBB huésped.

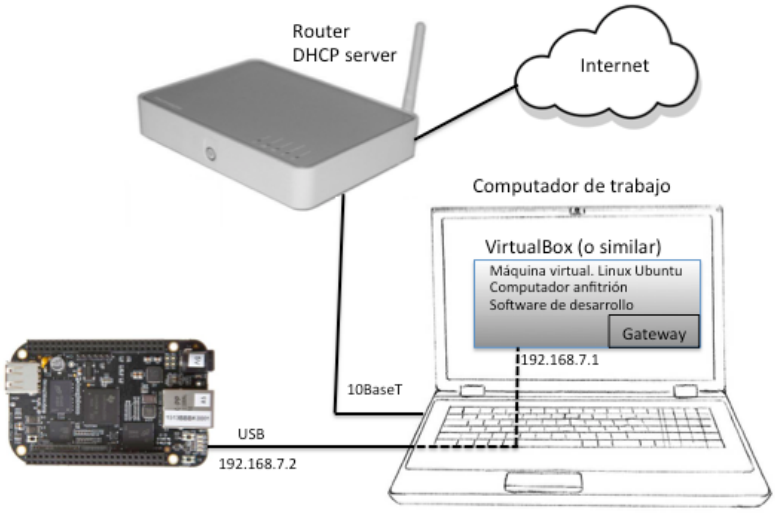


Figura 3: Conectando la BBB mediante el controlador de red integrado en el puerto USB cliente

En primer lugar, tendremos que asignar el dispositivo USB a la máquina virtual (computador anfitrión). Esta operación se hace en VirtualBox así:

- Conecte un cable MicroUSB-USB entre el USB cliente de la BBB y el computador de trabajo.
- Arranque VirtualBox y seleccione la máquina virtual.
- ***Machine*** -> ***Settings*** > ***Ports*** -> ***USB***
- Añadir un filtro (icon “+” de la barra lateral) e indique el dispositivo en cuestión. Ver Figura 4
- Arranque el SO Linux anfitrión de la máquina virtual.
- Compruebe que puede acceder a la BBB desde la máquina virtual (por defecto, la IP de la BBB es 192.168.7.2)

```
$ ssh -l root 192.169.7.2
```

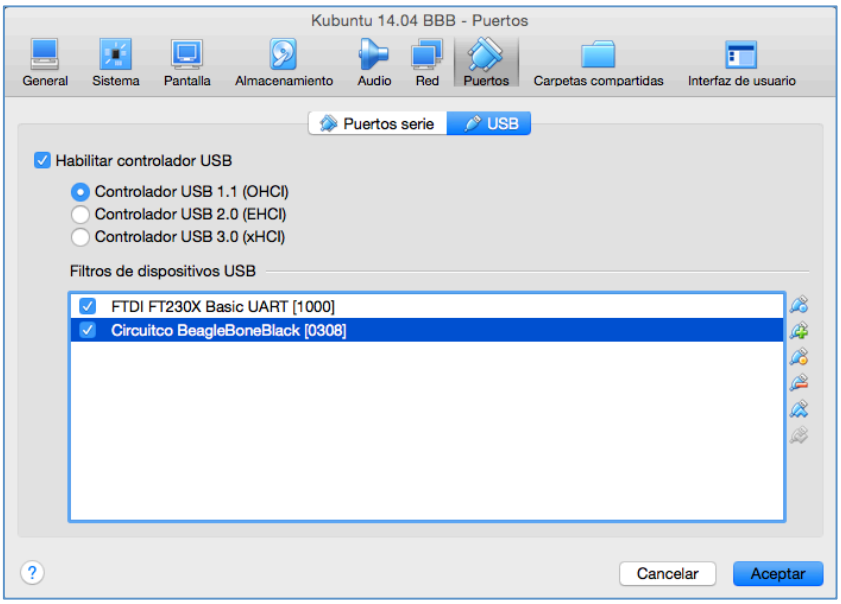


Figura 4: Si usa VirtualBox debe asignar a la VM el dispositivo USB.

4.1. Conectar la BBB a Internet a través del computador anfitrión

Ahora tenemos que configurar la máquina virtual para que actúe de pasarela de red de la conexión USB.

En una configuración normal, el computador de trabajo está conectado a internet a través de alguna infraestructura existente (p.e. un router). El computador anfitrión (una máquina virtual ejecutándose en el computador de trabajo) tiene acceso a internet porque el computador de trabajo está configurado para que actúe de pasarela de las máquinas virtuales. Pues bien, ahora la BBB conectada por USB al computador anfitrión (la MV) accederá a Internet en la medida en que configuremos el computador anfitrión para que actúe de pasarela.

Observemos la configuración de red de del computador anfitrión

```
jsimo@DevBBB:~$ uname -a
Linux DevBBB 3.16.0-50-generic #66~14.04.1-Ubuntu SMP Thu Sep 10 17:05:00 …
jsimo@DevBBB:~$ ifconfig
eth0      Link encap:Ethernet  direcciónHW 08:00:27:7d:1e:fc
          Direc. inet:10.0.2.15  Difus.:10.0.2.255  Másc:255.255.255.0
          Dirección inet6: fe80::a00:27ff:fe7d:1efc/64 Alcance:Enlace
          ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:13461 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:5570 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1000
          Bytes RX:11003739 (11.0 MB)  TX bytes:390673 (390.6 KB)
eth1      Link encap:Ethernet  direcciónHW 6c:ec:eb:5c:e1:3d
          Direc. inet:192.168.7.1  Difus.:192.168.7.3  Másc:255.255.255.252
          Dirección inet6: fe80::6eec:ebff:fe5c:e13d/64 Alcance:Enlace
          ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:696 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:825 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1000
          Bytes RX:131526 (131.5 KB)  TX bytes:113422 (113.4 KB)
lo        Link encap:Bucle local
          Direc. inet:127.0.0.1  Másc:255.0.0.0
```

```
        Dirección inet6: ::1/128 Alcance:Anfitrión
        ACTIVO BUCLE FUNCIONANDO  MTU:65536  Métrica:1
        Paquetes RX:368 errores:0 perdidos:0 overruns:0 frame:0
        Paquetes TX:368 errores:0 perdidos:0 overruns:0 carrier:0
        colisiones:0 long.colaTX:0
        Bytes RX:31799 (31.7 KB)  TX bytes:31799 (31.7 KB)
jsimo@DevBBB:~$ route
Tabla de rutas IP del núcleo
Destino      Pasarela     Genmask           Indic Métric Ref    Uso Interfaz
default      10.0.2.2     0.0.0.0            UG    0      0      0      eth0
10.0.2.0      *           255.255.255.0      U     1      0      0      eth0
192.168.7.0   *           255.255.255.252    U     1      0      0      eth1
```

Vemos que el computador anfitrión se conecta al computador de trabajo (10.0.2.2) mediante el interfaz eth0 (10.0.2.15). El computador anfitrión está configurado (pasarela por defecto) de modo que encamina los paquetes de red al computador de trabajo. En el computador anfitrión, el interfaz de red USB (eth1) tiene la dirección 192.168.7.1.

Observemos ahora la configuración de red de la BBB:

```
root@BBB01:~# uname -a
Linux BBB01 3.8.13-bone47 #1 SMP Fri Apr 11 01:36:09 UTC 2014 armv7l GNU/Linux
root@BBB01:~# ifconfig
eth0       Link encap:Ethernet  HWaddr 6c:ec:eb:5c:e1:3b
        UP BROADCAST MULTICAST  MTU:1500  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
        Interrupt:40
lo            Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:65536  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
usb0      Link encap:Ethernet  HWaddr 9a:17:98:eb:f3:97
        inet addr:192.168.7.2  Bcast:192.168.7.3  Mask:255.255.255.252
        inet6 addr: fe80::9817:98ff:feeb:f397/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:797 errors:0 dropped:0 overruns:0 frame:0
        TX packets:669 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:64068 (62.5 KiB)  TX bytes:166492 (162.5 KiB)
root@BBB01:~# route
Kernel IP routing table
Destination     Gateway         Genmask             Flags Metric Ref    Use Iface
default         DevBBB.local    0.0.0.0             UG     0      0      0   usb0
192.168.7.0     *               255.255.255.252 U    0      0      0          usb0
```

Comprobamos que la BBB no tiene conectado el interfaz de red “10/100 Ethernet” (eth0) y que el interfaz USB (usb0) tiene la dirección 192.168.7.2. Este interfaz es “local”, es decir, que no tiene definida la pasarela por defecto y, por lo tanto, no tiene acceso a Internet.

Pues bien, para conseguir nuestro objetivo, que la BBB tenga acceso a Internet, tenemos que realizar las siguientes operaciones.

1. Que el computador anfitrión sea la pasarela de defecto de la BBB, es decir, en la BBB definir la pasarela por defecto como 192.168.7.1. Esto se hace en la BBB introduciendo la orden:

```
$ route add default gw 192.168.7.1
```

2. Que el computador anfitrión encamine los paquetes que reciba por el interfaz eth1 (192.168.7.1) hacia el interfaz eth0, su conexión de Internet. Esto se hace en el computador anfitrión introduciendo las órdenes:

```
$ iptables --table nat --append POSTROUTING --out-interface eth0 -j MASQUERADE
$ iptables --append FORWARD --in-interface eth1 -j ACCEPT
$ echo 1 > /proc/sys/net/ipv4/ip_forward
```

Estas operaciones no causan un efecto permanente y deben repetirse cada vez que reinicia el sistema. Para ello, por el momento, es conveniente que escriba los scripts que se listan en el "Anexo I. Scripts" de este documento.

Resumiendo: la secuencia de arranque y recomendada es:

- Arrancar la MV del computador anfitrión. Abrir una terminal y ejecutar el script "netUSB_enable_gateway.sh" (necesita permisos de root).

```
$ sudo netUSB_enable_gateway.sh
```

- Conectar la BBB al puerto USB del computador de trabajo. El computador anfitrión (que es el que tiene asignado el dispositivo USB BBB) reacciona mostrando una unidad de disco (ignorar) e informando del nuevo interfaz de red conectado.
- Abrir una terminal contra la BBB

```
$ ssh –l root 192.168.7.2
```

- Ejecutar el script ""netUSB_enable_internet_access.sh"

```
$ netUSB_enable_internet_access.sh
```

- Comprobar que tenemos acceso a internet

```
$ ping www.google.es
```

4.2. Problemas comunes

En esta sección se ha descrito cómo conectar una placa BBB a internet a través de la conexión de red USB y un computador anfitrión (máquina virtual) correctamente configurado. En una situación normal todo ocurre como se ha descrito, pero se pueden dar ciertas situaciones que hagan variar el escenario descrito:

1.- Si trabajamos con varias placas BBB distintas conectándolas de forma alternativa al computador anfitrión, cada vez que cambie de placa tendrá que configurar la asignación del dispositivo a la máquina virtual en "VirtualBox", tal y como se describía en la Figura 4. Tendrá que eliminar la asignación del dispositivo (botón con el signo menos en rojo) y añadir el nuevo que ha

detectado el computador de trabajo, aunque el antiguo y el nuevo se llamen igual. Después de realizar esta operación detenga el SO de la BBB, desconéctela del puerto USB del computador de trabajo y vuélvala a conectar al puerto USB.

Si está en este caso, usando diferentes BBB en el mismo computador anfitrión, la primera placa que conectemos aparecerá en el computador anfitrión como el interfaz de red "eth1", la segunda como "eth2" y así sucesivamente. El sistema operativo del computador anfitrión siempre asignará el mismo dispositivo (p.e. eth2) a una determinada placa. Tenga en cuenta este cambio en las operaciones que se han descrito y/o edite los scripts que aparecen en el anexo para que se ajusten a esta eventualidad.

Si quiere eliminar la historia de conexión de diferentes BBB en el computador anfitrión, de forma que la siguiente placa que conecte tenga asignado el interfaz "eth1", tendrá que eliminar las reglas que el SO almacena en el archivo "**/etc/udev/rules.d/70-persistent-net.rules**"

```
$ cd /etc/udev/rules.d
$ more 70-persistent-net.rules
# This file was automatically generated by the /lib/udev/write_net_rules
# program, run by the persistent-net-generator.rules rules file.
#
# You can modify it, as long as you keep each rule on a single
# line, and change only the value of the NAME= key.
# USB device 0x:0x (rndis_host)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="6c:ec:eb:5c:e1:3d",
ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="eth*", NAME="eth1"
# USB device 0x:0x (rndis_host)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="d0:5f:b8:fe:f6:39",
ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="eth*", NAME="eth2"
```

En este caso tendría que eliminar las líneas marcadas en negrita.

2.- Si trabajamos con una BBB que puede arrancar desde la microSD o desde la memoria MMC interna. En este caso es posible que el computador anfitrión no consiga conectarse la red que le ofrece la conexión USB de la BBB. El problema está en que, al cambiar la identidad del sistema operativo de la BBB, el cliente DHCP del computador anfitrión no consigue obtener su dirección IP de forma automática. Esto se resuelve en el computador anfitrión asignando estáticamente la dirección IP del interfaz de red USB de la BBB. Para ello edite el archivo "/etc/network/interfaces" del computador anfitrión para que su contenido sea:

```
$ more /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet static
   address 192.168.7.1
   netmask 255.255.255.0
   network 192.168.7.0
```

Cuando configura la red del computador anfitrión con dirección IP estática, es posible que se ralentice el proceso de arranque mostrando un mensaje "waiting for network configuration". Para solucionar este problema en el computador anfitrión, edite (con permisos de root) el archivo **"/etc/init/failsafe.conf**" y comente todas las líneas en las que aparezca la orden "sleep". Después de hacer esto observará que el computador anfitrión arranca rápidamente otra vez.

5. Conexión X11 con la BBB

El sistema de ventanas X es un software orientado a proporcionar un interfaz gráfico simple sobre un sistema operativo Unix. Se desarrolló inicialmente para que varios clientes pudieran iniciar sesiones de usuario con interfaz gráfico contra el mismo computador. Esto dio lugar a lo que se llama "conexión con cliente ligero", que proporciona un terminal gráfico virtual mediante una conexión cliente-servidor. La interacción entre cliente y servidor se realiza mediante el "Xprotocol" que consiste en órdenes básicas codificadas para crear, posicionar y controlar eventos en ventanas. Existen implementaciones de X11 para la mayoría de sistemas operativos y es extremadamente portable.

En nomenclatura X11, el "servidor X11" es el programa que interpreta las órdenes "Xprotocol" y las convierte en gráficos para el usuario, por lo tanto se ejecuta en el cliente ligero. El "cliente X11" es una aplicación que se ejecuta normalmente en un computador con gran potencia de cómputo y convierte el resultado gráfico de su ejecución en órdenes "Xprotocol".

En nuestro caso, queremos que el computador anfitrión muestre las ventanas gráficas de aplicaciones que se están ejecutando en la BBB. Siguiendo la nomenclatura, el computador anfitrión alojará el "servidor X11" y las aplicaciones ejecutándose en la BBB serán "clientes X11".

Como base de la interacción X11 entre el computador anfitrión y la BBB, usaremos la conexión ssh. El reenvío X11 se tiene que habilitar tanto en el computador anfitrión como en la BBB.

En la BBB (cliente X11) el reenvío X11 debe especificarse en el archivo "**/etc/ssh/sshd_config**" añadiendo las líneas:

```
X11Forwarding yes
X11UseLocalhost no
```

Hay que asegurarse de que el programa "xauth" esté instalado en la BBB. Observe que no es necesario establecer variables de entorno como $DISPLAY o $XAUTHORITY, ssh las establece automáticamente. Después de realizar los cambios, reinicie el servicio ssh así:

```
$ service ssh restart
```

o simplemente reinicie la BBB.

En el computador anfitrión basta con aplicar la opción "-X" (X mayúscula) en la orden ssh que usemos para establecer la conexión con la BBB. Por ejemplo:

```
$ ssh -X -l root 192.168.7.2
```

Con una conexión establecida de esta manara, ya podemos arrancar aplicaciones gráficas en la BBB y ver las ventanas en el computador anfitrión. Por ejemplo:

- Con la aplicación "**leafpad**" obtenemos un editor de textos gráfico para editar los archivos de texto de la BBB de forma cómoda.
- Con la aplicación "**pcmanfm**" obtenemos un administrador de archivos para visualizar e interactuar con el sistema de ficheros de la BBB.
- Con "**startlxde**" podemos lanzar el escritorio completo de la BBB.

6. Actualización de la hora del sistema en la BBB

La BBB no dispone de reloj permanente, es decir, la placa aunque dispone de un reloj hardware, este no está alimentado por una batería que mantenga la fecha y hora reales mientras el sistema está desconectado. La consecuencia directa es que, aunque actualicemos la hora del sistema, cuando apagamos la BBB la hora se pierde. Esto lo podemos observar si abrimos una terminal en la BBB e introducimos la orden "date":

```
$ date
Sat Jan 1 01:00:09 CET 2000
```

La forma más cómoda de actualizar la hora del sistema es utilizar el "Network Time Protocol" (NTP). La práctica totalidad de los sistemas operativos utilizan este protocolo para mantener la hora actualizada. A escala mundial existe una jerarquía de servidores NTP que se sincronizan entre ellos y con relojes atómicos para ofrecer servicio de tiempo en la red. Dependiendo del país en el que nos encontremos habrá que configurar el NTP para que use los servidores de tiempos más cercanos. En el sistema Debian de la BBB esto se hace editando (o creando) el archivo "**/etc/ntp.conf** ".

Considerando que estamos en España, el contenido recomendado para el fichero "/etc/ntp.conf " es el siguiente:

```
# /etc/ntp.conf, configuration for ntpd; see ntp.conf(5) for help
# You do need to talk to an NTP server or two (or three).
server 2.es.pool.ntp.org
server 1.europe.pool.ntp.org
server 3.europe.pool.ntp.org
```

Una vez definidos correctamente los servidores NTP, podemos actualizar la hora del sistema introduciendo la siguiente orden:

```
$ ntpd -q -g –x
ntpd: time set +46559481.380206s
```

Nota: consulte el manual de "ntpd" para comprender el significado de las opciones.

El programa "ntpd" debe encontrarse en el directorio "/usr/sbin". Si el sistema de la BBB no la incluye, instale el paquete correspondiente así:

```
$ apt-get install ntp
```

Podemos comprobar que la hora se ha actualizado utilizando otra vez la orden “date”. Tras actualizar la hora del sistema, actualizaremos el reloj hardware para que tenga la misma hora que el sistema así:

```
$ hwclock –systohc
```

Las operaciones descritas en esta sección se realizan en el script “time_NTP_sync.sh” incluido en el “Anexo I. Scripts” de este documento.

7. Ejecución de servicios en la BBB

Durante el desarrollo de aplicaciones para la BBB utilizaremos normalmente un entorno de desarrollo de compilación cruzada y el IDE “Eclipse” desde el computador anfitrión. Esto permitirá escribir programas en el computador anfitrión y probarlos en la BBB de una forma cómoda.

Cuando ya hemos desarrollado un programa y queremos desplegarlo en la BBB de forma que se ejecute al arrancar la BBB, tenemos que instalar un “servicio de sistema”. Por comodidad, instalaremos un servicio que ejecute un script “my_startup_service.sh” ubicado en el directorio “/root”. Si en un futuro queremos modificar la secuencia de programas que se ejecutan en el arranque, simplemente tendremos que editar este archivo.

1. En primer lugar, tenemos que crear el archivo “/root/my_startup_service.sh” con un contenido simple, por ejemplo:

   ```
   #!/bin/sh
   echo "----- Starting My Startup Service script -----" > /root/my_startup.log
   echo `date`  >> /root/my_startup.log
   ##### Write here custom processes start
   ###############
   echo "----- My Startup Service launched. -----" >> /root/my_startup.log
   while true; do sleep 30; done
   echo "--- My Startup Service killed. Childs also killed. ---" >> /root/my_startup.log
   ```

   En este script hay algunas cosas importantes que observar.

   - Los servicios se ejecutan sin consola, es decir, cualquier salida por pantalla se pierde. Por esta razón, tenemos que redirigir cualquier salida por pantalla a un fichero que podamos visualizar cuando el sistema haya arrancado. En nuestro caso, el fichero que usaremos es “/root/my_startup.log”.
   - Si lanzamos un proceso desde el script (proceso hijo), si el script termina, también termina el proceso que hemos lanzado. La terminación del script implica la terminación de todos sus procesos hijos. Para evitar que esto ocurra, podemos hemos introducido la línea “while true; do sleep 30; done”. En esta situación la ultima línea del script no debería ejecutarse nunca.

2. Asigne permisos de ejecución al script:

   ```
   $ chmod a+x my_startup_service.sh
   ```

3. Creamos el archivo “/lib/systemd/myStartup.service” con el siguiente contenido

```
[Unit]  Description=Starts user-defined daemons
After=syslog.target network.target

[Service]
Type=simple  ExecStart=/root/my_startup_service.sh

[Install]
WantedBy=multi-user.target
```

4. Hacemos el siguiente enlace simbólico:

```
$ ln -s /lib/systemd/myStartup.service /etc/systemd/system/myStartup.service
```

5. Ahora arrancamos y habilitamos el servicio así.

```
$ systemctl daemon-reload
$ systemctl start myStartup.service
$ systemctl enable myStartup.service
```

6. Reinicie la BBB, abra una nueva terminal ssh y observe el contenido del fichero “my_startup.log”.

```
$ more my_startup.log
----- Starting My Startup Service script -----
Sat Jan 1 01:00:09 CET 2000
----- My Startup Service launched. -----
```

8. Conclusiones

Tal y como ya se ha comentado, para modificar las órdenes que queremos que se ejecuten durante el arranque, simplemente tendremos que modificar el contenido del archivo “/root/my_startup_service.sh”. Copie en la BBB los scripts del “Anexo I. Scripts” de este documento con la estructura que se muestra en la Figura 5.

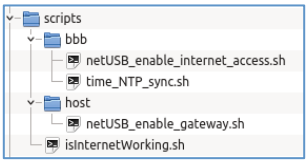


Figura 5: Estructura de directorios de los scripts suministrados.

En la BBB edite el contenido del archivo “/root/my_startup_service.sh” para que sea el que se incluye en el “Anexo I. Scripts”. Apague la BBB.

Ahora, para tener una configuración como la de la Figura 3 con acceso a Internet y reloj sincronizado, simplemente:

1. Arranque el computador anfitrión
2. En el computador anfitrión ejecutar el script /scripts/host/netUSB_enable_gateway.sh
3. Conectar la BBB al puerto USB. Abrir una terminal de la BBB.
4. Comprobar que todo funciona correctamente (mirar el contenido de “my_startup.log”, hacer ping,…)

9. Anexo I. Scripts

9.1. netUSB_enable_internet_access.sh (para la BBB)

Este script comprueba si el interfaz de red “usb0” está presente y activo. En el caso en que no esté presente y activo, realiza hasta 15 intentos de comprobación espaciados por un segundo. Cuando comprueba que el interfaz está presente y activo, añade la pasarela por defecto. Deberá editar este fichero si el interfaz de red en cuestión tiene un nombre diferente de “usb0” o si la pasarela de red tiene una IP diferente de “192.168.7.1”.

```
#!/bin/bash
interface=usb0
opfile="/sys/class/net/"$interface"/operstate"
gateway="192.168.7.1"
wellKnownDNS="8.8.8.8"
for i in {1..15}
do
if [ -f $opfile ]
then
   isInterfaceUp=`cat $opfile | grep -i up | wc -l`
   if [ $isInterfaceUp = "1" ]
   then
      echo "The interface "$interface" is UP. Adding default gateway "$gateway
      route add default gw $gateway
      #Check if the wellKnownDNS is configured in the system. If not, define it.
      configuredNameServer=`cat /etc/resolv.conf | grep "^nameserver" | grep
$wellKnownDNS | wc -l`
      if [ $configuredNameServer = 0 ]
      then
         echo "nameserver "$wellKnownDNS >> /etc/resolv.conf
      fi
      exit 0
   else
      echo "The interface "$interface" is DOWN"
      sleep 1
   fi
else
   echo "The interface "$interface" is not present"
   sleep 1
fi
done
```

9.2. time_NTP_sync.sh (para la BBB)

Sincroniza el tiempo del sistema con los servidores NTP definidos y, posteriormente, actualiza el reloj hardware con los valores del sistema. Si la red no está accesible, el programa “ntpd” se queda bloqueado. Para evitar esto, se lanza con un timeout de 60 segundos.

```
#!/bin/bash
timeout 60 /usr/sbin/ntpd -q -g -x
if [ $? -eq 0 ]; then
    echo "Time server Online. Syncing Hw Clock."
    /sbin/hwclock --systohc
else
    echo "Time servers Offline. WARNING: System Time is out of sync."
    exit 0
fi
```

9.3. netUSB_enable_gateway.sh (para el computador anfitrión)

```
#!/bin/bash
external_interface=eth0
internal_interface=eth1
iptables --table nat --append POSTROUTING --out-interface $external_interface -j
MASQUERADE
iptables --append FORWARD --in-interface $internal_interface -j ACCEPT
echo 1 > /proc/sys/net/ipv4/ip_forward
```

9.4. isInternetWorking.sh (para el computador anfitrión y la BBB)

```
#!/bin/bash
#Check the Internet access by pining a well known host
wellKnownHost="www.google.com"
responses=`ping -c 1 $wellKnownHost | grep "bytes from" | wc -l`
if [ $responses = 0 ]
then
    connected="FALSE"
else
    connected="TRUE"
fi
######################## Return the state
echo $connected
########################
```

9.5. my_startup_service.sh (para la BBB)

Configura el acceso a Internet por el interfaz USB y, si hay conexión a Internet, sincroniza el reloj.

```
#!/bin/sh
echo "----- Starting My Startup Service script -----" > /root/my_startup.log
echo `date` >> /root/my_startup.log

#Try to enable the internet access throght USB interface
echo "Enabling Internet access" >> /root/my_startup.log
/root/scripts/bbb/netUSB_enable_internet_access.sh >>  /root/my_startup.log

if [ `/root/scripts/isInternetWorking.sh` = TRUE ]
then
    echo "Sync Clock (NTP)" >> /root/my_startup.log
    /root/scripts/bbb/time_NTP_sync.sh >> /root/my_startup.log
    echo `date` ": Current Time." >>  /root/my_startup.log
fi

##### Write here custom processes start
###############

echo "----- My Startup Service launched. -----" >> /root/my_startup.log
while true; do sleep 30; done
echo "----- My Startup Service killed. Child processes also killed. -----" >>
/root/my_startup.log
```

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Sistemas Empotrados

Master de Automática e Informática Industrial

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Guías BeagleBone Black (IV)

Entrada y Salida Digital

Rev. José Simó. Enero 2016.

Contenido

1. Objetivos

Con la ayuda de este documento aprendrá a manejar los pines GPIO y a hacer programas en C++ que incluyan el acceso a estos pines.

2. Introducción

Bajo la denominación GPIO ("General Purpose Input Output") encontramos los mecanismos de E/S de señales digitales. Con estas señales el procesador puede implementar buses de comunicaciones y cualquier comunicación general con periféricos externos. La placa de desarrollo BeagleBone Black (BBB) incluye, además de otros periféricos integrados, 4 controladores GPIO. Cada controlador GPIO gestiona 32 señales digitales, es decir, potencialmente con la BBB podemos gestionar un total de 32x4 = 128 señales digitales. Obviamente como los pines del encapsulado del procesador comparten funciones (pinmux) y no todos los pines del procesador están accesibles en las cabezas de expansión (P8 y P9) de la BBB, para nuestros proyectos tendremos acceso sólo a un conjunto limitado de señales digitales de E/S. En la Figura 1 podemos ver las funciones por defecto asignadas a los pines de las cabezas de expansión de la BBB.

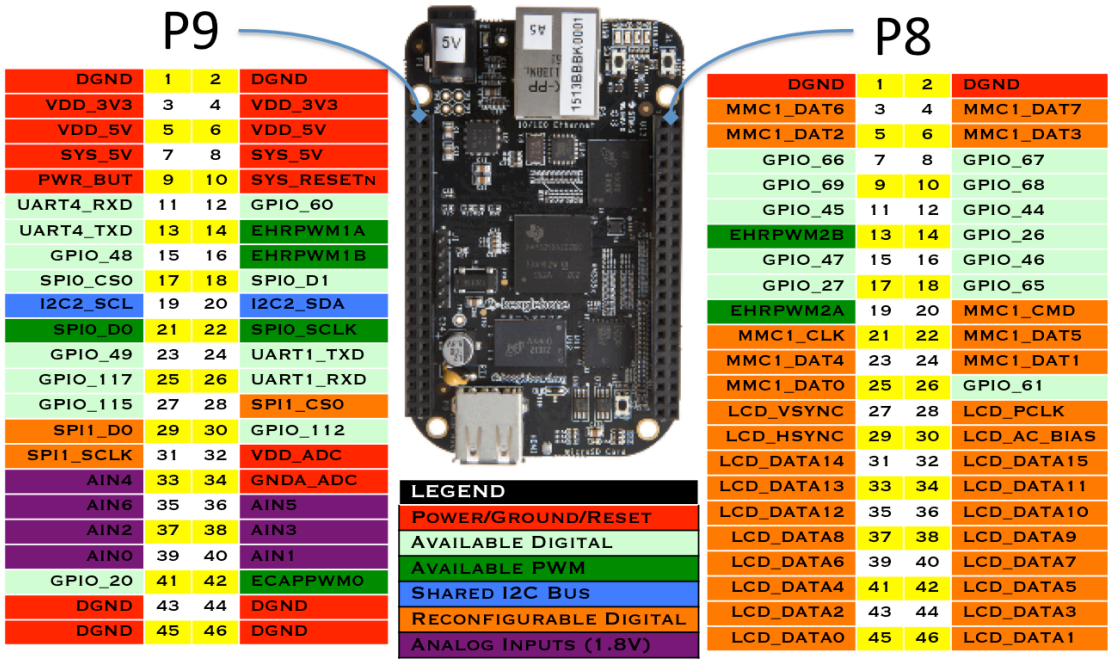


Figura 1: Asignación de los pines de expansión.

En general, cada pin del procesador puede desempeñar diferentes funciones ("pinmux") y, en particular, los pines del procesador accesibles a través de las cabezas de expansión. Las funciones por defecto de los pines que se muestran en la Figura 1 se pueden cambiar accediendo al "Módulo de Control" del procesador instalando "Device Tree Overlays" desde el Linux Debian que se ejecuta en la BBB utilizando el "Cape Manager". Este no es el objetivo de este documento, aquí nos limitaremos a controlar los pines GPIO accesibles por defecto en las cabezas de expansión P8 y P9.

3. Identificación de los pines

Observando la Figura 1 encontramos que tenemos accesibles directamente 19 pines GPIO. En la Figura 2 aparece resumida la información sobre la denominación de cada uno de estos pines. Es importante entender las diferentes denominaciones para desenvolverse en el sistema y con los manuales técnicos.

- **Pin de expansión:** es el número de pin de la cabeza de expansión. Por ejemplo, el P9_12 es el pin número 12 del conector P9.
- **GPIO interno:** Hay cuatro controladores GPIO (GPIO0..GPIO3) y cada uno de ellos tiene capacidad para 32 pines (0..31). Internamente cada pin se identifica con su número y su controlador. Por ejemplo, el GPIO1_28 hace referencia al pin "28" del controlador "1".
- **Nº de GPIO:** Es el número que corresponde a un pin si consideramos que los pines de los cuatro controladores GPIO (del 0 al 3) se numeran secuencialmente. Por ejemplo, el GPIO_60 corresponde al GPIO1_28 porque 1*32+28 = 60 (antes del controlador 1, hay 32 pines que corresponden al controlador 0).
- **PIN:** Las funciones de los pines del procesador se pueden configurar accediendo al vector "pinmux" del "Control Module" del procesador. El PIN es el índice del pin en cuestión dentro de este vector. Cada elemento de este vector tiene 4 bytes.
- **Offset:** Es el offset de la palabra de configuración del pin en cuestión dentro del vector "pinmux" del "Control Module". Observe que PIN = Offset / 4. Por ejemplo 0x78 = 120d y 120/4 = 30 (observe la línea P9_12 de la Figura 2).

Conector P9				
Pin de expansión	**nº de GPIO**	**GPIO interno**	**PIN**	**offset**
P9_12	GPIO_60	GPIO1_28	30	0x078
P9_15	GPIO_48	GPIO1_16	16	0x040
P9_23	GPIO_49	GPIO1_17	17	0x044
P9_25	GPIO_117	GPIO3_21	107	0x1AC
P9_27	GPIO_115	GPIO3_19	105	0x1A4
P9_30	GPIO_112	GPIO3_16	102	0x198
P9_41	GPIO_20	GPIO0_20	109	0x1B4

Conector P8				
P8_7	GPIO_66	GPIO2_2	36	0x090
P8_8	GPIO_67	GPIO2_3	37	0x094
P8_9	GPIO_69	GPIO2_5	39	0x09C
P8_10	GPIO_68	GPIO2_4	38	0x098
P8_11	GPIO_45	GPIO1_13	13	0x034
P8_12	GPIO_44	GPIO1_12	12	0x030
P8_14	GPIO_26	GPIO0_26	10	0x028
P8_15	GPIO_47	GPIO1_15	15	0x03C
P8_16	GPIO_46	GPIO1_14	14	0x038
P8_17	GPIO_27	GPIO0_27	11	0x02C
P8_18	GPIO_65	GPIO2_1	35	0x08C
P8_26	GPIO_61	GPIO1_29	31	0x07C

Figura 2. Información sobre los pines de expansión configurados por defecto como GPIO.

4. Conexión de componentes a los pines

Como es natural, estamos interesados en conectar algún circuito eléctrico a los pines GPIO para, por ejemplo, encender un LED, poner en funcionamiento un motor o detectar cuándo se cierra un interruptor. Para realizar estas operaciones sin dañar la BBB, tenemos que tener en cuenta las limitaciones eléctricas de los diferentes pines.

- **El nivel de tensión de 0V (tierra)** está disponible en los pines P9_1, P9_2, P9_{43,44,45,46}, P8_1, P8_2.
- **El nivel de tensión de 5V**, disponible siempre que la BBB esté en funcionamiento, lo podemos tomar de los pines SYS_5V (P9_7 y P9_8). De estos pines podemos drenar una corriente máxima de 250mA.
- **El nivel de tensión de 5V**, disponible siempre que la BBB tenga conectado el alimentador externo, lo podemos tomar de VDD_5V (P9_5 y P9_6). De estos pines podemos drenar una corriente máxima de 1A.
- **El nivel de tensión de 3.3V**, disponible siempre que la BBB esté en funcionamiento, lo podemos tomar de los pines VDD_3V3 (P9_3 y P9_4). De estos pines podemos drenar una corriente máxima de 400mA.
- **El nivel de tensión de 1.8V**, disponible siempre que la BBB esté en funcionamiento, lo podemos tomar de VDD_ADC (P9_32). Esta tensión se mide con respecto a la referencia (tierra) accesible en el pin GNDA_ADC (P9_34). De este pin podemos drenar una corriente máxima de 250mA.

- Los pines GPIO funcionan a una tensión máxima de 3.3V (NO CONECTAR TENSIONES 5V A PINES DE LA BBB)
- Los pines GPIO pueden proporcionar una corriente máxima entre 4 y 6 mA (configurado como salida).
- Los pines GPIO pueden absorber una corriente máxima de 8mA (configurado como entrada).

Como los pines GPIO tienen muy limitada la corriente que pueden drenar, tendemos que utilizar siempre en nuestros circuitos resistencias limitadoras, transistores opto-acopladores o cualquier otro método que nos asegure cumplir con las limitaciones.

Por ejemplo: **NO** PODEMOS ENCENDER UN LED DIRECTAMENTE CON UNA SEÑAL GPIO. Hay que ponerle una resistencia limitadora en serie. Ver Figura 3.

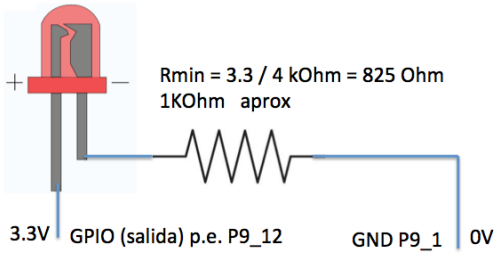


Figura 3. Resistencia limitadora

El montaje de la Figura 3 nos permite utilizar un LED para monitorizar el estado de una salida GPIO, pero en realidad no resulta muy útil para controlar una carga eléctrica. En general, para utilizar de forma práctica una salida GPIO debemos aislar el circuito de conmutación del circuito de carga utilizando un transistor. Lo ideal es utilizar un transistor MOSFET como el Toshiba 2SK2961 que drenando por el terminal "gate" una corriente del orden de microamperios, puede controla cargas de hasta 60V y 2A (consulte la hoja de características del componente). El montaje sería como el de la Figura 4.

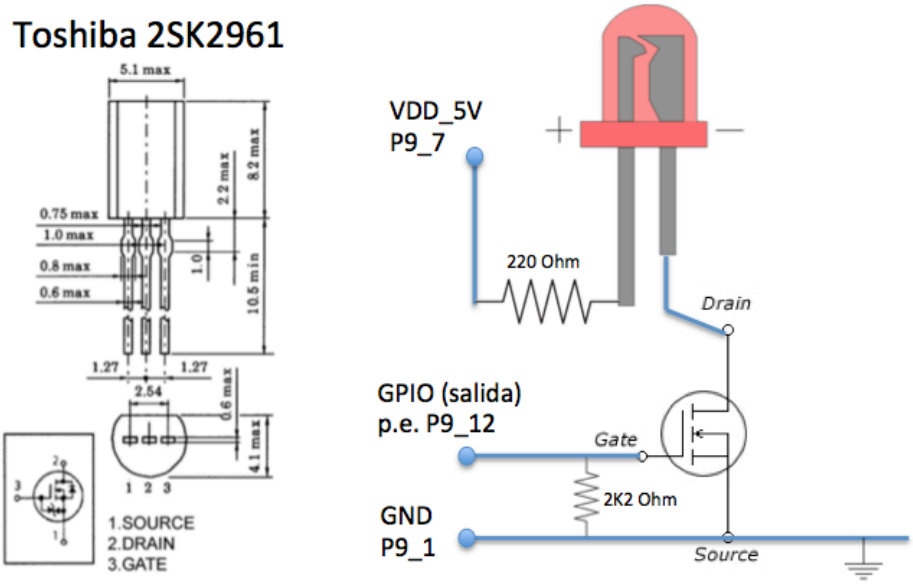


Figura 4. Montaje de conmutación de carga con MOSFET.

Si lo que queremos encender o apagar es un motor, el montaje sería el mismo que el de la Figura 4 pero añadiendo un diodo en sentido inverso entre los terminales del motor para evitar que por el circuito de conmutación circulen corrientes inversas cuando se apaga el motor. Ver Figura 5.

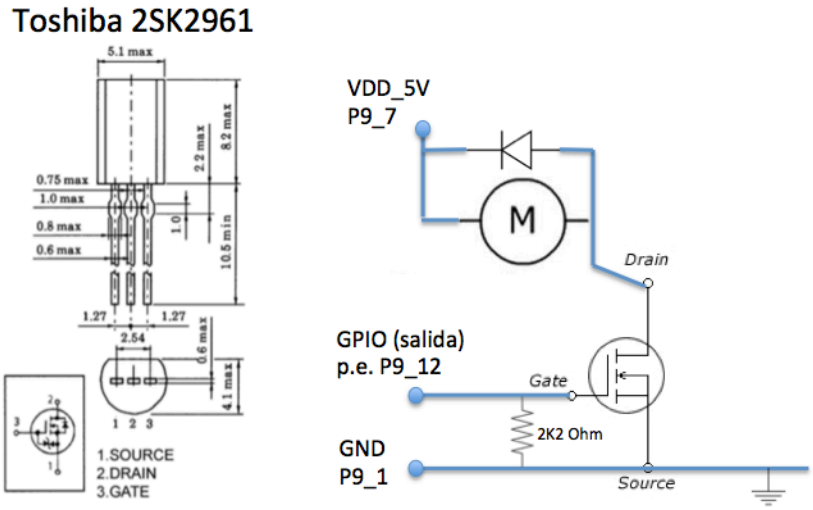


Figura 5. Si la carga a conmutar es un motor, no debemos olvidar el diodo de circulación inversa.

5. Control de los pines GPIO desde el Shell de Linux

El SO Linux Debian de la BBB incluye los manejadores de dispositivo necesarios para manejar cómodamente los pines GPIO. El usuario interactúa con el manejador de dispositivo escribiendo y leyendo de archivos "sysfs" localizados en la ruta /sys/class/gpio.

```
root@beaglebone:~# cd /sys/class/gpio
root@beaglebone:/sys/class/gpio# ls -l
total 0
--w------- 1 root root 4096 Oct 30 17:28 export
lrwxrwxrwx 1 root root    0 Jan  1  2000 gpiochip0 -> ../../devices/virtual/gpio/gpiochip0
lrwxrwxrwx 1 root root    0 Jan  1  2000 gpiochip32 -> ../../devices/virtual/gpio/gpiochip32
lrwxrwxrwx 1 root root    0 Jan  1  2000 gpiochip64 -> ../../devices/virtual/gpio/gpiochip64
lrwxrwxrwx 1 root root    0 Jan  1  2000 gpiochip96 -> ../../devices/virtual/gpio/gpiochip96
--w------- 1 root root 4096 Jan  1  2000 unexport
```

Para utilizar un pin, primero tenemos que exportarlo escribiendo el "nº de GPIO" en el archivo "export". Por ejemplo, para usar el P9_12, que es el GPIO60 haremos:

```
root@beaglebone:/sys/class/gpio# echo 60 > export
root@beaglebone:/sys/class/gpio# ls -l
total 0
--w------- 1 root root 4096 Oct 30 23:07 export
lrwxrwxrwx 1 root root    0 Oct 30 23:07 gpio60 -> ../../devices/virtual/gpio/gpio60
lrwxrwxrwx 1 root root    0 Jan  1  2000 gpiochip0 -> ../../devices/virtual/gpio/gpiochip0
lrwxrwxrwx 1 root root    0 Jan  1  2000 gpiochip32 -> ../../devices/virtual/gpio/gpiochip32
lrwxrwxrwx 1 root root    0 Jan  1  2000 gpiochip64 -> ../../devices/virtual/gpio/gpiochip64
lrwxrwxrwx 1 root root    0 Jan  1  2000 gpiochip96 -> ../../devices/virtual/gpio/gpiochip96
--w------- 1 root root 4096 Jan  1  2000 unexport
```

Esto crea un enlace simbólico a un directorio (en nuestro caso "gpio60") que contiene el "sysfs" para manejar el pin:

```
root@beaglebone:/sys/class/gpio# cd gpio60
root@beaglebone:/sys/class/gpio/gpio60# ls -l
total 0
-rw-r--r-- 1 root root 4096 Oct 30 23:11 active_low
-rw-r--r-- 1 root root 4096 Oct 30 23:11 direction
-rw-r--r-- 1 root root 4096 Oct 30 23:11 edge
drwxr-xr-x 2 root root    0 Oct 30 23:11 power
lrwxrwxrwx 1 root root    0 Oct 30 23:07 subsystem -> ../../../../class/gpio
-rw-r--r-- 1 root root 4096 Oct 30 23:07 uevent
-rw-r--r-- 1 root root 4096 Oct 30 23:11 value
```

Si realizamos el montaje de la Figura 4 (LED con MOSFET) y lo conectamos tal y como se indica al pin P9_12, podremos encender y apagar el pin así:

Primero definimos el pin como salida:

```
root@beaglebone:/sys/class/gpio/gpio60# cat direction
in
root@beaglebone:/sys/class/gpio/gpio60# echo out > direction
root@beaglebone:/sys/class/gpio/gpio60# cat direction
out
```

Ahora podemos escribir valores:

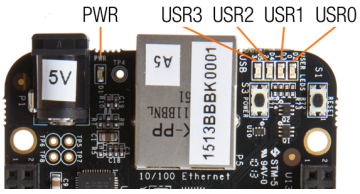
```
root@beaglebone:/sys/class/gpio/gpio60# cat value
0
root@beaglebone:/sys/class/gpio/gpio60# echo 1 > value
root@beaglebone:/sys/class/gpio/gpio60# echo 0 > value
```

Los valores de interés del "sysfs" de control de un pin GPIO son:

- **direction**: Define la dirección del pin, si es de entrada o salida. Los valores posibles son “in” y “out”. Se puede leer el valor de la dirección (haciendo $ cat direction) o escribir un nuevo valor. Cuanod se escribe el valor “out” para configurarlo como salida, el pin se inicializa a valor bajo.
- **value**: Es el valor del pin. 0 (nivel bajo) ó 1 (nivel alto). Si el pin se configura como entrada, indica el valor actual. Si el pin se configura como salida, se puede escribir el valor. Un pin GPIO configurado como entrada se puede usar como generador de interrupciones de forma que genere una interrupción cuando cambie de estado. Esto se hace aplicado la llamada a sistema POSIX “poll” al archivo sysfs “value” (consultar la página de manual de Linux “poll(2)”).
- **edge**: Si el pin se usa como interrupción, indica bajo que condiciones se genera el evento. Los posibles valores son: “none”, “rising”, “falling” y “both”. Este fichero existe sólo si el pin se puede configurar como una entrada generadora de interrupciones.
- **active_low**: Puede tomar los valores 0 (falso) ó 1 (verdadero). Indica que el valor del pin está invertido, es decir, 0 (nivel alto 3.3V) y 1 (nivel bajo 0V).

5.1. Los LEDs integrados en la placa

La BBB dispone de cuatro LEDs de usuario integrados en la placa conectados al los pines GPIO que se indican en la Figura 6.



USER LED	GPIO Interno	nº de GPIO
User led 0	GPIO1_21	GPIO_53
User led 1	GPIO1_22	GPIO_54
User led 2	GPIO1_23	GPIO_55
User led 3	GPIO1_24	GPIO_56

Figura 6. LEDs de usuario de la BBBB

Si pretendemos usar estos pines tal y como hemos aprendido obtendremos el siguiente error:

```
root@beaglebone:/sys/class/gpio# echo 53 > export
-bash: echo: write error: Device or resource busy
```

Esto se debe a que estos pines están reservados porque los usa otro manejador de dispositivos que nos ofrece un interfaz “sysfs” específico para su manejo. Este interfaz se encuentra en la ruta **“/sys/devices/ocp.3/gpio-leds.8/leds**” que contiene un directorio por cada LED.

```
root@beaglebone:/sys/class/gpio# cd /sys/devices/ocp.3/gpio-leds.8/leds
root@beaglebone:/sys/devices/ocp.3/gpio-leds.8/leds# ls -l
total 0
drwxr-xr-x 3 root root 0 Jan  1  2000 beaglebone:green:usr0
drwxr-xr-x 3 root root 0 Jan  1  2000 beaglebone:green:usr1
drwxr-xr-x 3 root root 0 Jan  1  2000 beaglebone:green:usr2
drwxr-xr-x 3 root root 0 Jan  1  2000 beaglebone:green:usr3
```

En el interior del directorio asociado a un LED en particular encontramos los archivos “sysfs” que nos permiten controlarlo:

```
root@beaglebone:/sys/devices/ocp.3/gpio-leds.8/leds# cd beaglebone\:green\:usr0
root@beaglebone:/sys/devices/ocp.3/gpio-leds.8/leds/beaglebone:green:usr0# ls -l
total 0
-rw-r--r-- 1 root root 4096 Oct 30 23:59 brightness
lrwxrwxrwx 1 root root    0 Oct 30 23:59 device -> ../../../gpio-leds.8
-r--r--r-- 1 root root 4096 Oct 30 23:59 max_brightness
drwxr-xr-x 2 root root    0 Oct 30 23:59 power
lrwxrwxrwx 1 root root    0 Jan  1  2000 subsystem -> ../../../../../class/leds
-rw-r--r-- 1 root root 4096 Oct 30 23:59 trigger
```

```
-rw-r--r-- 1 root root 4096 Jan  1  2000 uevent
```

El archivo “trigger” nos permite establecer un patrón de encendido para un LED en función de determinado eventos de sistema, por ejemplo , actividad en la red, actividad en disco o actividad en la CPU. También nos permite establecer diferentes patrones de parpadeo. Podemos ver los valores disponibles así:

```
root@beaglebone:/sys/devices/ocp.3/gpio-leds.8/leds/beaglebone:green:usr0# cat trigger
none nand-disk mmc0 mmc1 timer oneshot [heartbeat] backlight gpio cpu0 default-on transient
```

Esto nos indica que el LED usr0 tiene establecido un patrón de parpadeo de “latido de corazón”. Podemos cambiarlo así:

```
root@beaglebone:/sys/devices/ocp.3/gpio-leds.8/leds/beaglebone:green:usr0# echo none > trigger
root@beaglebone:/sys/devices/ocp.3/gpio-leds.8/leds/beaglebone:green:usr0# echo 1 > brightness
root@beaglebone:/sys/devices/ocp.3/gpio-leds.8/leds/beaglebone:green:usr0# echo 0 > brightness
```

Haga pruebas con los diferentes valores posibles de “trigger” para ver las posibilidades de uso de los LEDS de usuario. Por ejemplo, si establecemos el patrón “timer” observamos que nos aparecen dos nuevos archivos con los que podemos controlas el tiempo de encendido y apagado del patrón de parpadeo:

```
# echo timer > trigger
# ls -l
total 0
-rw-r--r-- 1 root root 4096 Oct 31 00:11 brightness
-rw-r--r-- 1 root root 4096 Oct 31 00:13 delay_off
-rw-r--r-- 1 root root 4096 Oct 31 00:13 delay_on
lrwxrwxrwx 1 root root    0 Oct 30 23:59 device -> ../../../gpio-leds.8
-r--r--r-- 1 root root 4096 Oct 30 23:59 max_brightness
drwxr-xr-x 2 root root    0 Oct 30 23:59 power
lrwxrwxrwx 1 root root    0 Jan  1  2000 subsystem -> ../../../../../class/leds
-rw-r--r-- 1 root root 4096 Oct 31 00:13 trigger
# cat delay_on
500
# cat delay_off
500
```

6. Control de la GPIO desde programas en C

La forma más utilizada, la más sencilla y que se deriva de lo que hemos aprendido en secciones anteriores de este documento de acceder a los pines GPIO desde programas que se ejecutan en el Linux de la BBB es escribir/leer en los archivos “sysfs” que nos ofrece el sistema.

En C++la lectura/escritura de archivos es trivial. En el Anexo “Código” se incluye el código de cuatro archivos que realizan de forma muy bien organizada estas operaciones y nos ofrecen funciones para el manejo directo del GPIO.

- **util.h**: cabecera de las funciones de lectura/escritura de ficheros
- **util.cpp**: implementación de las funciones de lectura/escritura de ficheros
- **GPIO.h**: definición de constantes, tipos y cabeceras de las funciones de acceso directo a la GPIO.
- **GPIO.cpp**: implementación del as funciones de acceso directo a la GPIO.

Lea y familiarícese con el código de estos cuatro archivos.

Cree en el entorno de desarrollo cruzado “Eclipse” de su computador anfitrión un nuevo proyecto “HelloGPIO” en el que copiará los cuatro archivos del anexo. Como los archivos del anexo utilizan “hilos de ejecución” usando la librería estándar POSIX “phtread”, hay que incluirla en el proyecto. Como el código incluye una utilidad para medir diferencias de tiempo con precisión, también hay que incluir la biblioteca “rt”. En las opciones del proyecto incluya las bibliotecas tal y como se indica en la Figura 7.

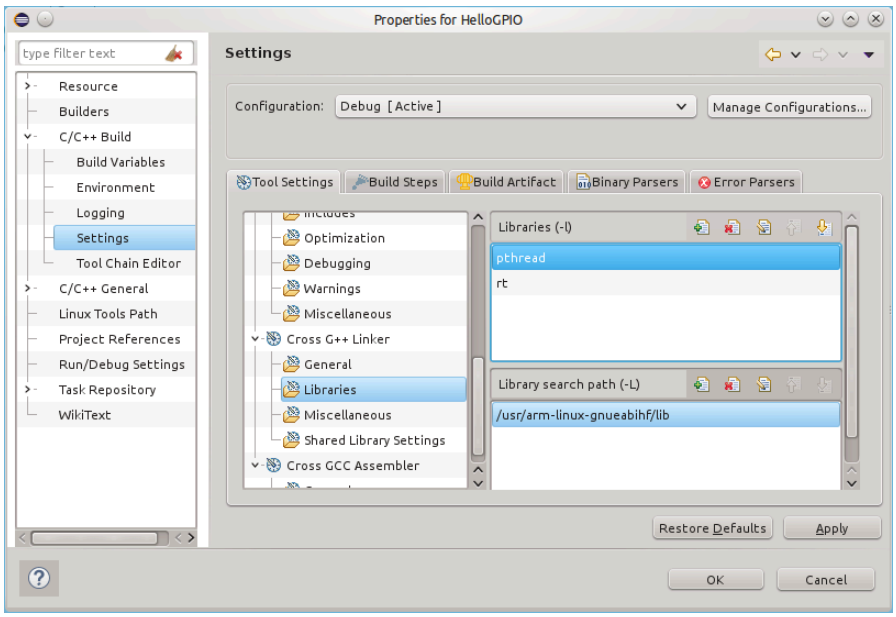


Figura 7. Incluya en el proyecto la biblioteca “pthread”

Cree también en el proyecto el archivo de programa principal (“helloGPIO.cpp”) con el siguiente contenido:

```
#include <stdio.h>
#include <unistd.h>
#include "GPIO.h"

using namespace exploringBB;

int main() {

	printf("Hello BBB GPIO.\n");

	GPIO outGPIO(60);
	outGPIO.setDirection(GPIO::OUTPUT);

	for (int i=0; i<10; i++) {
		outGPIO.setValue(GPIO::HIGH);
		usleep(500000);
		outGPIO.setValue(GPIO::LOW);
		usleep(500000);
	}
}
```

Este programa hace parpadear 10 veces el LED conectado al P9_12 (GPIO_60) suponiendo que tiene conectado el montaje de la Figura 4 usado en secciones anteriores. Como novedad, hemos utilizado la función “usleep” para suspender la ejecución del proceso una cantidad de microsegundos. El valor 500000 se corresponde a medio segundo. Para usar esta función hemos tenido que incluir en el proyecto las cabeceras “unistd.h”.

Haga pruebas modificando los retardos. En particular, si elimina los retardos por completo y aumenta el número de iteraciones (para que no acabe enseguida el programa):

```
#include <stdio.h>
#include <unistd.h>
#include "GPIO.h"

using namespace exploringBB;

int main() {

	printf("Hello BBB GPIO.\n");

	GPIO outGPIO(60);
	outGPIO.setDirection(GPIO::OUTPUT);

	for (int i=0; i<100000; i++) {
		outGPIO.setValue(GPIO::HIGH);
		outGPIO.setValue(GPIO::LOW);
	}
}
```

Podrá observas la velocidad máxima de conmutación que proporciona este método si conecta un osciloscopio a la salida P9_12. El resultado se puede ver en la Figura 8.

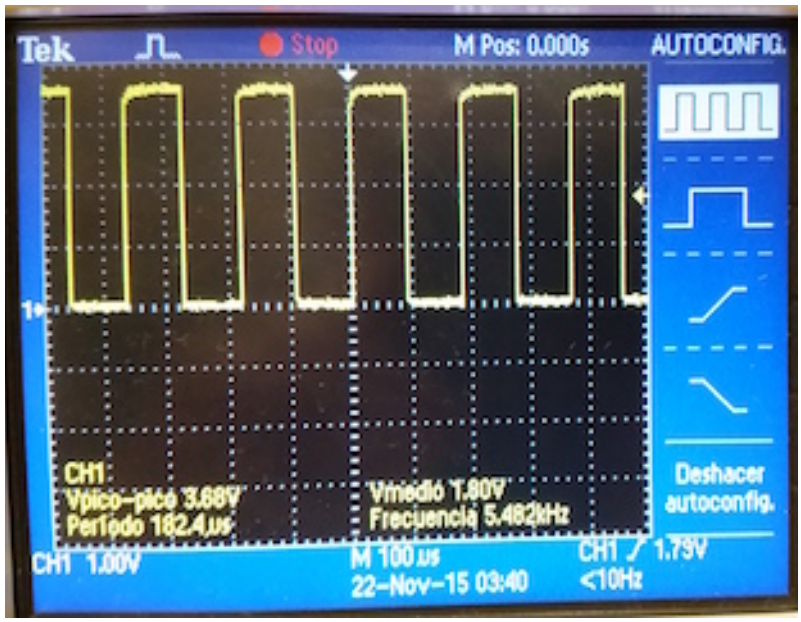


Figura 8. Velocidad de conmutación máxima obtenida con las rutinas “setValue”

El osciloscopio nos indica un periodo de unos 182 microsegundos. Rápido pero no tanto. Las rutinas “setValue” que hemos usado abren y cierran el archivo “sysfs” en cada operación y esto cuesta su tiempo.

En las rutinas incluidas en los archivos del anexo se incluyen funciones que evitan la sobrecarga de abrir y cerrar el archivo cada vez.

Pruebe el siguiente código:

```
#include <stdio.h>
#include <unistd.h>
#include "GPIO.h"
```

```
using namespace exploringBB;

int main() {

	printf("Hello BBB GPIO.\n");

	GPIO outGPIO(60);
	outGPIO.setDirection(GPIO::OUTPUT);

	outGPIO.streamOpen();
	for (int i=0; i<1000000; i++) {
		outGPIO.streamWrite(GPIO::HIGH);
		outGPIO.streamWrite(GPIO::LOW);
	}
	outGPIO.streamClose();
}
```

En este caso la imagen del osciloscopio es la que muestra la Figura 9. La el periodo de la señal es de 10 microsegundos, casi 20 veces más rápido.

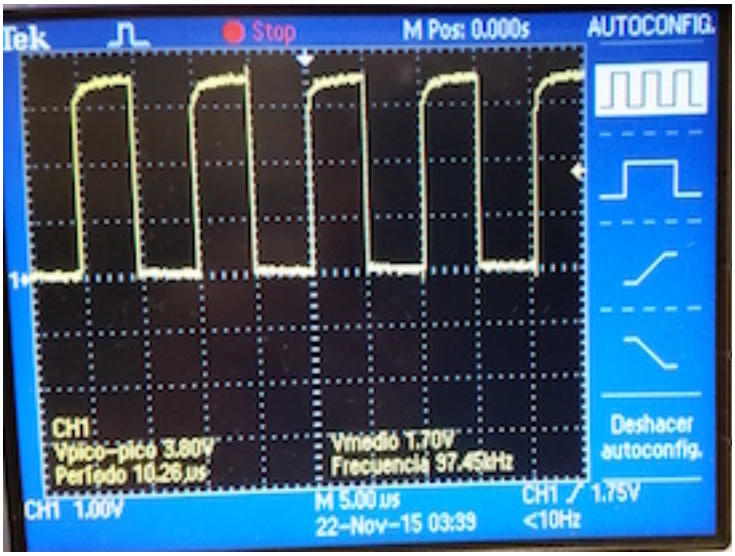


Figura 9. Velocidad de conmutación máxima obtenida con las rutinas “streamWrite”

Por último, establezca una conexión ssh desde el computador anfitrión contra la BBB y ejecuta la orden “top”. Vuelva a Eclipse y ejecute cualquiera de los dos programas en los que ha eliminado los retardos. Podrá observar que la ejecución del programa ocupa prácticamente el 100% de CPU. Ver Figura 10.

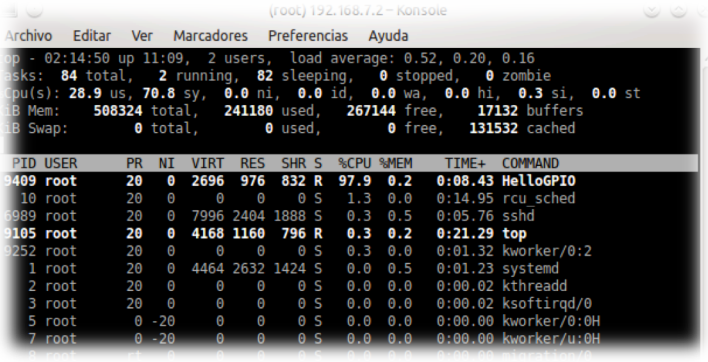


Figura 10. Uso de la CPU del programa “HelloGPIO”.

Obviamente esto no es forma de generar con la BBB una señal cuadrada de frecuencia relativamente alta. Además de ocupar completamente el procesador lo estamos haciendo sobre el SO Linux que no es de tiempo real, es decir, que la ejecución del programa se puede ver interrumpida en cualquier momento si el SO decide expulsar al proceso para ejecutar otro durante algunas decenas de milisegundos.

7. Lectura de entradas GPIO

Los pines GPIO también se pueden utilizar como entradas y así detectar niveles de tensión alto o bajo de señales binarias como pulsadores, finales de carrera o líneas de un bus digital.

IMPORTANTE: El nivel de tensión máximo que soporta una entrada GPIO es de 3.3V. No conectar nunca una entrada GPIO a la línea de tensión de 5V.

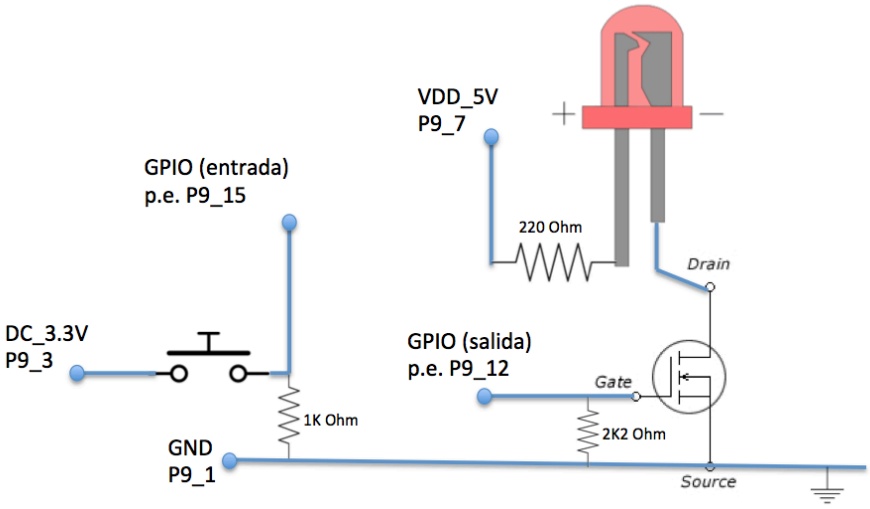


Figura 11. Montaje de un pulsador como entrada y una salida LED controlada por MOSFET.

En a Figura 11 se puede ver un ejemplo de conexión de un pulsador para generar una entrada GPIO. Para asegurar que la entrada no se queda en un valor flotante, se conecta a una resistencia de "pull-down" de 1KOhm. Cuando se acciona el pulsador, la tensión de la entrada es de 3.3V y cuando no está pulsado de 0V.

El valor de la entrada se puede consultar desde el shell de Linux de la BBB accediendo a los "sysfs" ubicados en "/sys/class/gpio" (tenga en cuenta que el pin P9_15 es el pin GPIO_48, ver Figura 2):

```
root@beaglebone:~# cd /sys/class/gpio
root@beaglebone:/sys/class/gpio# echo 48 > export
root@beaglebone:/sys/class/gpio# cd gpio48
root@beaglebone:/sys/class/gpio/gpio48# echo in > direction
root@beaglebone:/sys/class/gpio/gpio48# cat value
0
```

De forma programática podemos muestrear el valor de una entrada GPIO utilizando las rutinas que se incluyen en el código que se suministra en el anexo. Por ejemplo, modifique el archivo de programa principal ("helloGPIO.cpp") del proyecto para que contenga el siguiente código:

```
#include <stdio.h>
#include <unistd.h>
#include "GPIO.h"
```

```
using namespace exploringBB;

int main() {

	printf("Hello BBB GPIO.\n");

	GPIO outGPIO(60);
	GPIO inGPIO(48);
	outGPIO.setDirection(GPIO::OUTPUT);
	inGPIO.setDirection(GPIO::INPUT);
	inGPIO.setActiveHigh();
	for(;;) {
		GPIO::VALUE value = inGPIO.getValue();
		if (value == GPIO::HIGH) {
			outGPIO.setValue(GPIO::HIGH);
			usleep(5000000);
			outGPIO.setValue(GPIO::LOW);
		}
		usleep(100000);
	}
}
```

En este programa, se muestrea el valor del pin P9_15 (GPIO_48) cada 100ms en un bucle infinito. Si se detecta que se ha pulsado el pulsador, enciende el LED conectado a la salida P9_12 durante 5 segundos.

Esta forma de tratar la lectura GPIO puede ser adecuada en muchas aplicaciones pero tiene dos inconvenientes:

- Para tener lectura de un cambio en la entrada GPIO tenemos que estar leyendo su valor de forma continua (muestreo o “polling”) consumiendo ciclos de CPU. Cuanto mayor sea el periodo de muestreo menos será la sobrecarga de CPU introducida. En nuestro caso, como el procesador de la BBB es muy potente, el código del ejemplo que muestrea el pulsador 10 veces por segundo introduce una sobrecarga imperceptible del 0.2% (orden “top”).
- Como estamos muestreando, es posible que el cambio en la entrada GPIO que leemos se haya producido un tiempo (periodo de muestreo) antes del momento de percibirlo, con el consiguiente retraso en la posible toma de acciones en nuestro programa. Esto se puede solucionar reduciendo el tiempo de muestreo (sobrecarga de CPU). Observe que si el periodo de muestreo es muy grande (p.e. 1 segundo) podemos accionar el pulsador y soltarlo sin que nuestro programa detecte el cambio.

Cuando en nuestro programa necesitamos detectar el cambio en una entrada GPIO con precisión y corrección (sin retraso y asegurándonos de detectar cualquier cambio aunque vuelva muy rápido a su estado original), podemos usar la detección de eventos en el sistema de ficheros que nos ofrece el API POSIX. Concretamente la llamada “epoll”. Esta técnica está implementada en el código que se suministra en el anexo y consiste en crear un hilo de ejecución independiente que se queda suspendido en una llamada “epoll_wait” que desbloquea el hilo cuando se produce un evento (p.e. escritura) en el archivo “sysfs” “value” del GPIO de entrada en cuestión.

La forma de usar estas rutinas es la que aparece en el siguiente código:

```
#include <stdio.h>
#include <unistd.h>
#include "GPIO.h"
```

```
using namespace exploringBB;
int eventCounter;
GPIO outGPIO(60);
GPIO inGPIO(48);

int eventHandler(int arg) {
      eventCounter++;
      printf("Boton pulsado %d veces\n", eventCounter);
      outGPIO.setValue(GPIO::HIGH);
      usleep(500000);
      outGPIO.setValue(GPIO::LOW);
      return(0)
}

int main() {

      printf("Hello BBB GPIO.\n");

      outGPIO.setDirection(GPIO::OUTPUT);
      inGPIO.setDirection(GPIO::INPUT);
      inGPIO.setActiveHigh();
      inGPIO.setEdgeType(GPIO::RISING);
      inGPIO.setDebounceTime(300);

      inGPIO.waitForEdge(eventHandler);

      for(;;) usleep(30000000);

}
```

Básicamente este programa instala un manejador para el evento de GPIO y se queda ejecutando un bucle infinito en el que cada iteración es una suspensión de 30 segundos. Este bucle infinito, cuyo consumo de CPU es nulo, se introduce para evitar que el programa termine.

Con la función “waitForEvent” instalamos el manejador que ejecutará un hilo interno a la clase “GPIO” cuando se produzca el evento de cambio a nivel alto (RISING) en el pin. La propiedad “debounceTime” se usa para implementar internamente un filtro anti-rebote, en nuestro caso se descartarán los eventos que se produzcan con una separación temporal inferior a 300 milisegundos.

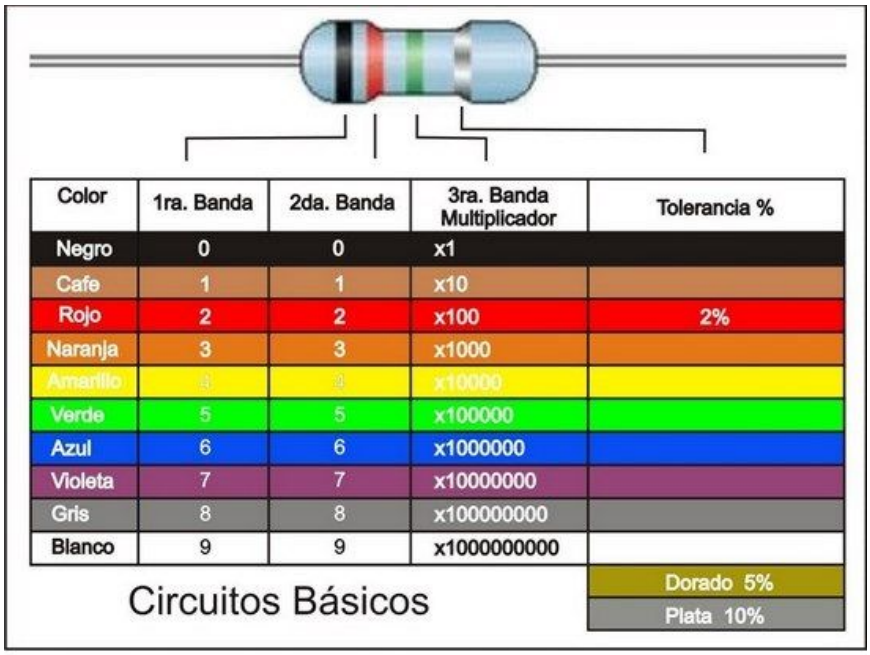
Observe detenidamente el código que se incluye en el anexo para comprender perfectamente el funcionamiento de estas rutinas de control del GPIO.

8. Conclusiones

Después de trabajar con este documento habrá aprendido a manejar la GPIO de la BBB utilizando los manejadores de dispositivo que incluye el SO Linux Debian “embedded” y su interfaz de ficheros “sysfs”. A partir de ahora podrá incluir el acceso a la GPIO en los programas “C++” que desarrolle.

9. Anexo I. Tablas

9.1. Tabla de códigos de color de resistencias



Color	1ra. Banda	2da. Banda	3ra. Banda Multiplicador	Tolerancia %
Negro	0	0	x1	
Cafe	1	1	x10	
Rojo	2	2	x100	2%
Naranja	3	3	x1000	
Amarillo	4	4	x10000	
Verde	5	5	x100000	
Azul	6	6	x1000000	
Violeta	7	7	x10000000	
Gris	8	8	x100000000	
Blanco	9	9	x1000000000	
				Dorado 5%
				Plata 10%

Circuitos Básicos

10. Anexo II. Código

Utilidades de manejo del GPIO tomadas del código ofrecido por Derek Molloy (www.derekmolloy.ie) y modificadas para su adaptación a esta guía.

10.1. Archivo "util.h"

```
#ifndef UTIL_H_
#define UTIL_H_
#include<string>
using std::string;


namespace exploringBB {

int write(string path, string filename, string value);
int write(string path, string filename, int value);
string read(string path, string filename);
long getCurrentMicroseconds();

} /* namespace exploringBB */

#endif /* UTIL_H_ */
```

10.2. Archivo "util.cpp"

```
#include "util.h"
#include<iostream>
#include<fstream>
#include<sstream>
using namespace std;

namespace exploringBB {
/**
 * Helper write function that writes a single string value to a file in the path provided
 * @param path The sysfs path of the file to be modified
 * @param filename The file to be written to in that path
 * @param value The value to be written to the file
 * @return
 */
int write(string path, string filename, string value){
   ofstream fs;
   fs.open((path + filename).c_str());
   if (!fs.is_open()){
           perror("GPIO: write failed to open file ");
           return -1;
   }
   fs << value;
   fs.close();
   return 0;
}
/**
 * Helper read function that reads a single string value to a file from the path provided
 * @param path The sysfs path of the file to be read
 * @param filename Filename The file to be written to in that path
 * @return
 */
string read(string path, string filename){
   ifstream fs;
   fs.open((path + filename).c_str());
   if (!fs.is_open()){
           perror("GPIO: read failed to open file ");
    }
   string input;
   getline(fs,input);
   fs.close();
   return input;
}

/**
 * Private write method that writes a single int value to a file in the path provided
 * @param path The sysfs path of the file to be modified
 * @param filename The file to be written to in that path
 * @param value The int value to be written to the file
 * @return
```

```
 */
int write(string path, string filename, int value){
   stringstream s;
   s << value;
   return write(path,filename,s.str());
}

long getCurrentMicroseconds(){
        struct timespec currentTime;
        clock_gettime(CLOCK_MONOTONIC, &currentTime);
        return (currentTime.tv_sec)*1000000 + (currentTime.tv_nsec) / 1000;
}

} /* namespace exploringBB */
```

10.3. Archivo “GPIO.h”

```
#ifndef GPIO_H_
#define GPIO_H_
#include<string>
#include<fstream>
using std::string;
using std::ofstream;

#define GPIO_PATH "/sys/class/gpio/"

namespace exploringBB {

typedef int (*CallbackType)(int);

/**
 * @class GPIO
 * @brief GPIO class for input and output functionality on a single GPIO pin
 */
class GPIO {
public:
        enum DIRECTION{ INPUT, OUTPUT };
        enum VALUE{ LOW=0, HIGH=1 };
        enum EDGE{ NONE, RISING, FALLING, BOTH };

private:
        int number;             /**< The GPIO number of the object */
        int debounceTime;       /**< The debounce time in milliseconds */
        string name;            /**< The name of the GPIO e.g. gpio50 */
        string path;            /**< The full path to the GPIO e.g. /sys/class/gpio/gpio50/ */

public:
        GPIO(int number);
        virtual int getNumber() { return number; } /**< Returns the GPIO number as an int. */

        // General Input and Output Settings
        virtual int setDirection(GPIO::DIRECTION);
        virtual GPIO::DIRECTION getDirection();
        virtual int setValue(GPIO::VALUE);
        virtual int toggleOutput();
        virtual GPIO::VALUE getValue();
        virtual int setActiveLow(bool isLow=true);  //low=1, high=0
        virtual int setActiveHigh(); //default
        //software debounce input (ms) - default 0
        virtual void setDebounceTime(int time) { this->debounceTime = time; }

        // Advanced OUTPUT: Faster write by keeping the stream alive (~20X)
        virtual int streamOpen();
        virtual int streamWrite(GPIO::VALUE);
        virtual int streamClose();

        virtual int toggleOutput(int time); //threaded invert output every X ms.
        virtual int toggleOutput(int numberOfTimes, int time);
        virtual void changeToggleTime(int time) { this->togglePeriod = time; }
        virtual void toggleCancel() { this->threadRunning = false; }

        // Advanced INPUT: Detect input edges; threaded and non-threaded
        virtual int setEdgeType(GPIO::EDGE);
        virtual GPIO::EDGE getEdgeType();
        virtual int waitForEdge(); // waits until button is pressed
        virtual int waitForEdge(CallbackType callback); // threaded with callback
        virtual void waitForEdgeCancel() { this->threadRunning = false; }
```

```
        virtual ~GPIO();  //destructor will unexport the pin

private:
        //int write(string path, string filename, string value);
        //int write(string path, string filename, int value);
        //string read(string path, string filename);
        int exportGPIO();
        int unexportGPIO();
        ofstream stream;
        pthread_t thread;
        CallbackType callbackFunction;
        long lastEventTime;
        bool threadRunning;
        int togglePeriod;  //default 100ms
        int toggleNumber;  //default -1 (infinite)
        friend void* threadedPoll(void *value);
        friend void* threadedToggle(void *value);
};

void* threadedPoll(void *value);
void* threadedToggle(void *value);

} /* namespace exploringBB */

#endif /* GPIO_H_ */
```

10.4. Archivo “GPIO.cpp”

```
#include "GPIO.h"
#include "util.h"
#include<iostream>
#include<fstream>
#include<string>
#include<sstream>
#include<cstdlib>
#include<cstdio>
#include<fcntl.h>
#include<unistd.h>
#include<sys/epoll.h>
#include<pthread.h>

using namespace std;

namespace exploringBB {
/**
 * The constructor will set up the states and export the pin.
 * @param number The GPIO number to be exported
 */

GPIO::GPIO(int number) {
        this->number = number;
        this->debounceTime = 0;
        this->togglePeriod=100;
        this->toggleNumber=-1; //infinite number
        this->callbackFunction = NULL;
        this->thread = (long unsigned int)NULL;
        this->threadRunning = false;
        this->lastEventTime = getCurrentMicroseconds();;
        ostringstream s;
        s << "gpio" << number;
        this->name = string(s.str());
        this->path = GPIO_PATH + this->name + "/";
        this->exportGPIO();
        // need to give Linux time to set up the sysfs structure
        usleep(250000); // 250ms delay
}

/**
 * Private method to export the GPIO
 * @return int that describes if the operation fails
 */
int GPIO::exportGPIO(){
   return write(GPIO_PATH, "export", this->number);
}
```

```
int GPIO::unexportGPIO(){
   return write(GPIO_PATH, "unexport", this->number);
}

int GPIO::setDirection(GPIO::DIRECTION dir){
   switch(dir){
   case INPUT: return write(this->path, "direction", "in");
      break;
   case OUTPUT:return write(this->path, "direction", "out");
      break;
   }
   return -1;
}

int GPIO::setValue(GPIO::VALUE value){
   switch(value){
   case HIGH: return write(this->path, "value", "1");
      break;
   case LOW: return write(this->path, "value", "0");
      break;
   }
   return -1;
}

int GPIO::setEdgeType(GPIO::EDGE value){
   switch(value){
   case NONE: return write(this->path, "edge", "none");
      break;
   case RISING: return write(this->path, "edge", "rising");
      break;
   case FALLING: return write(this->path, "edge", "falling");
      break;
   case BOTH: return write(this->path, "edge", "both");
      break;
   }
   return -1;
}

int GPIO::setActiveLow(bool isLow){
   if(isLow) return write(this->path, "active_low", "1");
   else return write(this->path, "active_low", "0");
}

int GPIO::setActiveHigh(){
   return this->setActiveLow(false);
}

GPIO::VALUE GPIO::getValue(){
        string input = read(this->path, "value");
        if (input == "0") return LOW;
        else return HIGH;
}

GPIO::DIRECTION GPIO::getDirection(){
        string input = read(this->path, "direction");
        if (input == "in") return INPUT;
        else return OUTPUT;
}

GPIO::EDGE GPIO::getEdgeType(){
        string input = read(this->path, "edge");
        if (input == "rising") return RISING;
        else if (input == "falling") return FALLING;
        else if (input == "both") return BOTH;
        else return NONE;
}

int GPIO::streamOpen(){
        stream.open((path + "value").c_str());
        return 0;
}
int GPIO::streamWrite(GPIO::VALUE value){
        stream << value << std::flush;
        return 0;
}
int GPIO::streamClose(){
        stream.close();
        return 0;
```

```
}

int GPIO::toggleOutput(){
        this->setDirection(OUTPUT);
        if ((bool) this->getValue()) this->setValue(LOW);
        else this->setValue(HIGH);
    return 0;
}

int GPIO::toggleOutput(int time){ return this->toggleOutput(-1, time); }
int GPIO::toggleOutput(int numberOfTimes, int time){
        this->setDirection(OUTPUT);
        this->toggleNumber = numberOfTimes;
        this->togglePeriod = time;
        this->threadRunning = true;
    if(pthread_create(&this->thread, NULL, &threadedToggle, static_cast<void*>(this))){
        perror("GPIO: Failed to create the toggle thread");
        this->threadRunning = false;
        return -1;
    }
    return 0;
}

// This thread function is a friend function of the class
void* threadedToggle(void *value){
        GPIO *gpio = static_cast<GPIO*>(value);
        bool isHigh = (bool) gpio->getValue(); //find current value
        while(gpio->threadRunning){
                if (isHigh)    gpio->setValue(GPIO::HIGH);
                else gpio->setValue(GPIO::LOW);
                usleep(gpio->togglePeriod * 500);
                isHigh=!isHigh;
                if(gpio->toggleNumber>0) gpio->toggleNumber--;
                if(gpio->toggleNumber==0) gpio->threadRunning=false;
        }
        return 0;
}

// Blocking Poll - based on the epoll socket code in the epoll man page

int GPIO::waitForEdge(){
        this->setDirection(INPUT); // must be an input pin to poll its value
        int fd, epollfd;
        //int count=0;
        struct epoll_event ev;
        epollfd = epoll_create1(0);
    if (epollfd == -1) {
           perror("GPIO: Failed to create epollfd");
           return -1;
    }
    if ((fd = open((this->path + "value").c_str(), O_RDONLY | O_NONBLOCK)) == -1) {
       perror("GPIO: Failed to open file");
       return -1;
    }

    //ev.events = read operation | edge triggered | urgent data
    ev.events = EPOLLIN | EPOLLET | EPOLLPRI;
    ev.data.fd = fd;  // attach the file file descriptor

    //Register the file descriptor on the epoll instance, see: man epoll_ctl
    if (epoll_ctl(epollfd, EPOLL_CTL_ADD, fd, &ev) == -1) {
       perror("GPIO: Failed to add control interface");
       return -1;
    }
    //Wait for event
    long currentTime;
    do {
       int retval, i;
       for (i=0; i<2; i++) { //Discard the first event
               retval = epoll_wait(epollfd, &ev, 1, -1);
       };
       if ( retval == -1) {
               perror("GPIO: Poll Wait fail");
           close(fd);
           close(epollfd);
               return -1;
       }
       currentTime = getCurrentMicroseconds();
    } while(currentTime - lastEventTime < this->debounceTime * 1000);
```

```
    lastEventTime = currentTime;
    close(fd);
    close(epollfd);
        return 0;
}

// This thread function is a friend function of the class
void* threadedPoll(void *value){
        GPIO *gpio = static_cast<GPIO*>(value);
        while(gpio->threadRunning){
                gpio->callbackFunction(gpio->waitForEdge());
        }
        return 0;
}

int GPIO::waitForEdge(CallbackType callback){
        this->threadRunning = true;
        this->callbackFunction = callback;
    // create the thread, pass the reference, address of the function and data
    if(pthread_create(&this->thread, NULL, &threadedPoll, static_cast<void*>(this))){
        perror("GPIO: Failed to create the poll thread");
        this->threadRunning = false;
        return -1;
    }
    return 0;
}

GPIO::~GPIO() {
        this->unexportGPIO();
}

} /* namespace exploringBB */
```

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Sistemas Empotrados

Master de Automática e Informática Industrial

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Guías BeagleBone Black (V)

Accediendo a los dispositivos

Rev. José Simó. Enero 2016.

Contenido

1. Objetivos

En este documento revisaremos los mecanismos de uso de los pines externos del procesador AM335x y su conexión a las cabezas de expansión de la BBB. Aprenderemos a configurar los pines y a observar las funciones que puede desempeñar cada uno utilizando los mecanismos que nos ofrecen los nuevos sistemas Linux: “device tree” y “capemgr”.

2. Introducción

La placa de desarrollo BeagleBone Black (BBB) es una plataforma de desarrollo de bajo coste, libre y soportada por la comunidad de desarrolladores de código abierto. El corazón de la BBB es el procesador de Texas Instruments “Sitara AM3358B-ZCZ100” que contiene un procesador “ARM Cortex-A8” con una velocidad de procesamiento de 1GHz y 512MB de memoria principal (Figura 1).

Muchos de los pines de E/S del procesador están accesibles a través de dos conectores hembra de 2x23 pines (cabezas de expansión P8 y P9) incluyendo 4 puertos UART, 7 conversores A/D de entrada, 4 salidas PWM y una gran cantidad de puertos digitales. La documentación de referencia para el uso de los periféricos es el documento de Texas Instruments “AM335x Technical Reference Manual”. Descárguelo (http://www.ti.com/product/am3358) y téngalo a mano.

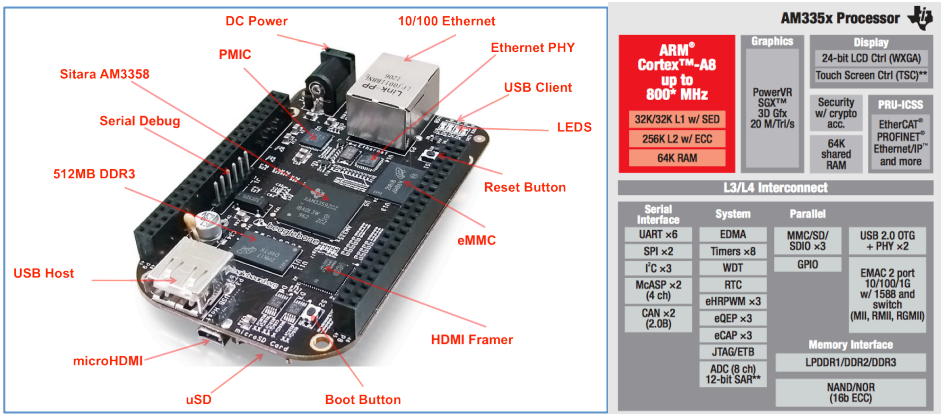


Figura 1: Vista rápida de la placa BeagleBone Black y del procesador AM335x.

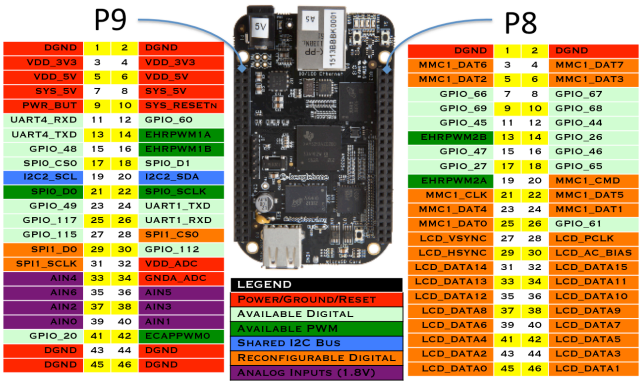


Figura 2: Asignación de los pines de expansión.

3. Configuración de los pines de expansión

En la sección 2.1 del "AM335x TRM" se encuentra la tabla que describe el mapa de memoria del procesador. En particular, observamos que el "Control Module" se encuentra ubicado a partir de la dirección 0x44e1_0000 y ocupa 7Kb. Estos registros definen el comportamiento del procesador y el significado detallado de cada uno se puede consultar en la sección 9.3.1 del TRM.

ADC_TSC	0x44E0_D000	0x44E0_EFFF	8KB	ADC_TSC Registers
	0x44E0_F000	0x44E0_FFFF	4KB	Reserved
Control Module	0x44E1_0000	0x44E1_1FFF	128KB	Control Module Registers
DDR2/3/mDDR PHY	0x44E1_2000	0x44E1_23FF		DDR2/3/mDDR PHY Registers
Reserved	0x44E1_2400	0x44E3_0FFF	4KB	Reserved

Una de las primeras cosas que tenemos que aprender el procesador AM335x es que tiene muchas más entradas y salidas en sus módulos periféricos internos que pines físicos en el encapsulado. En realidad, cada pin físico es la salida de un multiplexor con 8 modos posibles (tres bits de selección). Esta función conocida como "**pinmuxing**" es el mecanismo mediante los modernos SoC ofrecen múltiples funciones de los periféricos internos usando un número limitado de pines físicos. El modo de cada pin físico (su asignación a un periférico interno en concreto) se maneja asignando valores a registros del "Control Module". Tal y como ya hemos dicho, muchos de los pines físicos del procesador están accesibles para su uso en las cabezas de expansión P8 y P9 de la BBB, por lo tanto, antes de utilizar un pin de la cabeza de expansión tenemos que asegurarnos que está asignado correctamente al periférico que pretendamos utilizar.

En la Sección 9.3.1 ("CONTROL_MODULE Registers") del AM335x TRM (Tabla 9-10) observamos que la configuración de los pines comienza en el offset 0x800 (con respecto a la dirección de inicio del "Control Module" 0x44E1_0000), es decir, a partir de la dirección 0x44E1_0800 encontraremos un registro de 32 bits para cada uno de los pines.

31.. 7	6	5	4	3	2	1	0
Reserved	conf_<module>_<pin>_slewctrl	conf_<module>_<pin>_rxactive	conf_<module>_<pin>_putypesel	conf_<module>_<pin>_puden	conf_<module>_<pin>_mmode		
R-0h	R/W-0h	R/W-1h	R/W-0h	R/W-0h	R/W-0h		

LEGEND: R/W = Read/Write; R = Read only; W1toCl = Write 1 to clear bit; *-n* = value after reset

Table 9-60. conf_<module>_<pin> Register Field Descriptions

Bit	Field	Type	Reset	Description
31-20	Reserved	R	0h	
19-7	Reserved	R	0h	
6	conf_<module>_<pin>_slewctrl	R/W	X	Select between faster or slower slew rate 0: Fast 1: Slow Reset value is pad-dependent.
5	conf_<module>_<pin>_rxactive	R/W	1h	Input enable value for the PAD 0: Receiver disabled 1: Receiver enabled
4	conf_<module>_<pin>_putypesel	R/W	X	Pad pullup/pulldown type selection 0: Pulldown selected 1: Pullup selected Reset value is pad-dependent.
3	conf_<module>_<pin>_puden	R/W	X	Pad pullup/pulldown enable 0: Pullup/pulldown enabled 1: Pullup/pulldown disabled Reset value is pad-dependent.
2-0	conf_<module>_<pin>_mmode	R/W	X	Pad functional signal mux select. Reset value is pad-dependent. MODE: 0..7

Figura 3: Valores para la configuración de los pines mediante los registros de control. Tabla extraída del documento de Texas Instruments "AM335x Technical Reference Manual".

El significado de cada uno de los bits de este registro es el que aparece en la Figura 3. Podemos configurar por supuesto el modo (bits 0..2), si el pin es de entrada o salida, si el flanco de subida ha de ser rápido o lento y si se enclava internamente con una resistencia de "pull-up" o de "pull-down". Algunos ejemplos de valores de este registro son:

0x27 (010 0111) Fast-Input-Pulldown-Enabled-MuxMode7
0x37 (011 0111) Fast-Input-Pullup-Enabled-MuxMode7
0x07 (000 0111) Fast-Output-Pulldown-Enabled-MuxMode7
0x17 (001 0111) Fast-Output-Pullup-Enabled-MuxMode7

Observe que en la Tabla 9-10 del AM335x TRM el acrónimo del registro de configuración se corresponde con el texto "conf_" seguido del nombre del pin. Así, por ejemplo, podemos ver que el registro de configuración del pin "lcd_data0" se encuentra a partir del offset 0x8A0.

8A0h	conf_lcd_data0	Section 9.3.1.49
8A4h	conf_lcd_data1	Section 9.3.1.49

Utilizando las facilidades de Linux de "mapeo de ficheros en memoria" (mmap) sobre el dispositivo "/dev/mem" (imagen de memoria hardware), se han escrito las rutinas de los archivos listados en el anexo ("phymem.h" y "phymem.c") que permiten acceder a la memoria hardware de una manera simple. El siguiente programa, que hace uso de estas rutinas, permite leer la configuración del pin "lcd_data0".

```
#include "phymem.h"

int main(int argc, char **argv) {
	int i;

	char buffer[4];
	readPhyMem(buffer, 4, 0x44e108a0);
	for (i = 0; i< 4; i++) {
		printf("buffer[%d] = 0x%X\n", i, buffer[i]);
	}
}
```

Al ejecutar el programa, observamos que el valor de configuración es 0x0008, que corresponde a "Fast-Output-Pulldown-Disabled-MuxMode0"

Nota: Para poder acceder de una forma cómoda a la configuración de los pines de la BBB, el Linux Debian instalado ofrece un "Sysfs" en la ruta "/sys/kernel/debug/pinctrl/44e10800.pinmux/pins". Volcando el contenido de este fichero accedemos al vector de registros de configuración ubicado en 0x44e10800. Cada posición del vector ocupa 4 bytes. Para calcular el índice (número de pin en el vector) de un determinado pin, simplemente dividimos el offset del pin por 4. Por ejemplo, el pin "lcd_data0" tiene un offset 0x0A0 = 160d, luego su número de pin será 160/4 = 40. Comprobémoslo:

```
root@beaglebone:~# export PINS=/sys/kernel/debug/pinctrl/44e10800.pinmux/pins
root@beaglebone:~# cat $PINS | grep "pin 40"
pin 40 (44e108a0) 00000008 pinctrl-single
```

que coincide con el valor que hemos leído directamente mapeando la memoria hardware.

Para localizar todas las funciones que puede soportar un pin, tendremos que acudir al los esquemas de la BBB (descargar desde http://elinux.org/Beagleboard:BeagleBoneBlack#Hardware_Files) y localizar el pin como indica la Figura 4.

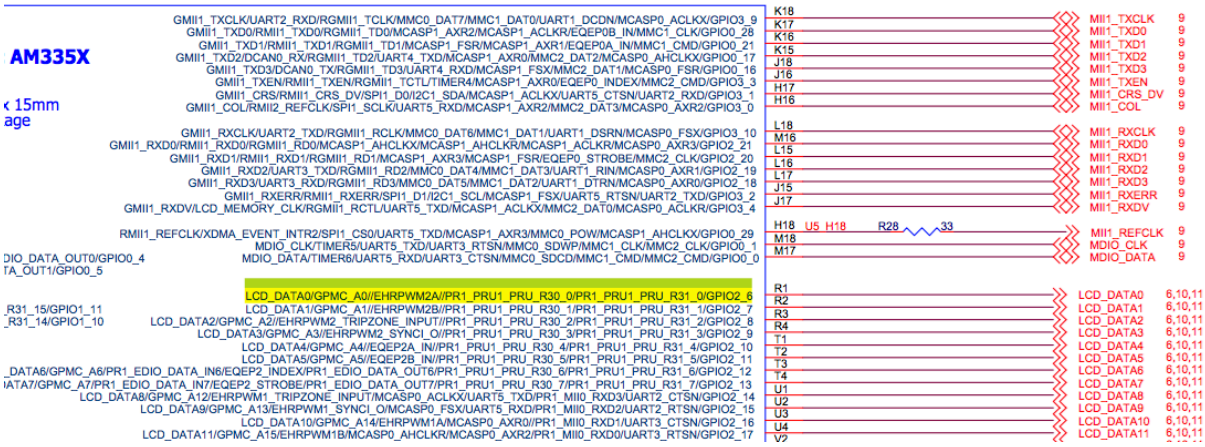


Figura 4: Localización de un pin en los esquemas de la BBB. Cada pin físico tiene asociado un conjunto de etiquetas que identifican los modos de operación que soporta (pinmux). La primera etiqueta es el nombre del pin.

En la lista de funciones (o nombres) que tiene cada pin físico, el primero es el que se considera en los manuales el nombre del pin, en nuestro caso “LCD_DATA0”. Para localizar este pin en las cabezas de expansión, hay que acudir también a los esquemas de la BBB y buscar la parte correspondiente a los conectores P8 y P9, como en la Figura 5 en la que comprobamos que el pin es el P8_45.

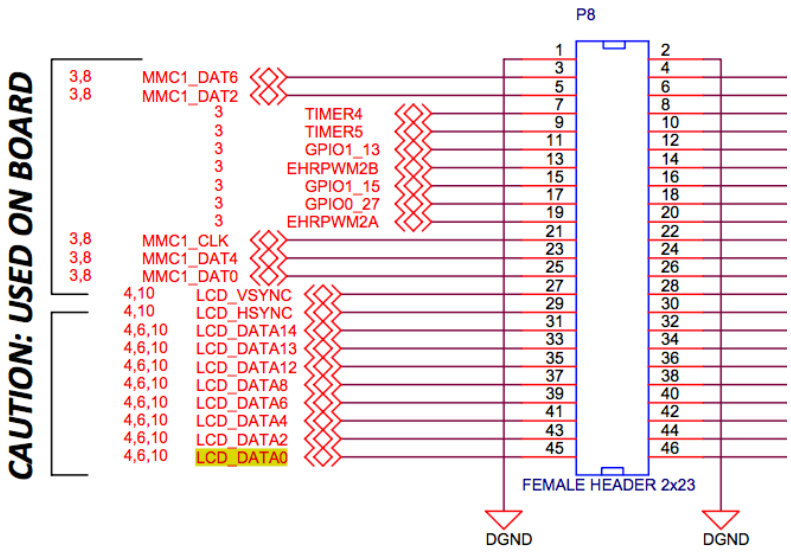


Figura 5: Localización de un pin en el conector de expansión en los esquemas de la BBB.

Para saber qué modo se corresponde con cada función, hay que consultar la Tabla 4.1 del “data sheet” del AM335x (se puede descargar desde http://www.ti.com/product/AM3358) tal y como se ve en la Figura 6.

ZCE BALL NUMBER [1]	ZCZ BALL NUMBER [1]	PIN NAME [2]	SIGNAL NAME [3]	MODE [4]	TYPE [5]	BALL RESET STATE [6]	BALL RESET REL. STATE [7]	RESET REL. MODE [8]	ZCE POWER / ZCZ POWER [9]	HYS [10]	BUFFER STRENGTH (mA) [11]	PULLUP /DOWN TYPE [12]	I/O CELL [13]
U1	R1	LCD_DATA0	lcd_data0	0	I/O	Z	Z	7	VDDSHV6 / VDDSHV6	Yes	6	PU/PD	LVCMOS
			gpmc_a0	1	O								
			pr1_mii_mt0_clk	2	I								
			ehrpwm2A	3	O								
			pr1_pru1_pru_r30_0	5	O								
			pr1_pru1_pru_r31_0	6	I								
			gpio2_6	7	I/O								

Figura 6: Localización de los atributos de un pin en el “data sheet (Tabla 4-1)” del procesador AM335x.

En la Figura 6 observamos que el modo 0 se corresponde con la función “lcd_dat0” (que coincide con el nombre del pin). También vemos que este pin puede usarse con la función “gpio2_6” (pin 6 de la GPIO 2) cambiando su modo a “modo 7”.

Nota: toda esta información repartida entre diferentes manuales está recopilada para su uso cómodo en las tablas de Derek Molloy (ver anexo) que se pueden descargar desde: https://github.com/derekmolloy/boneDeviceTree/tree/master/docs descárguelas y téngalas siempre a mano (ver anexo).

Los registros del “Control module” sólo se pueden escribir si el procesador está en modo de ejecución privilegiado (modo núcleo). Esto quiere decir que un programa de usuario NO puede escribir sobre estos registros y los programas que escriba para modificar estos registros con las rutinas “phymem” del anexo no funcionarán.

Versiones antiguas de Linux (anteriores a la 3.8) incluían un driver (cuya ejecución obviamente se realiza en modo núcleo) que permitía configurar los pines físicos del procesador escribiendo/leyendo sobre archivos del directorio “/sys/kernel/debug/omap_mux/”. Esta solución, ya abandonada por la proliferación de procesadores de la familia ARM, obligaba a incluir muchas líneas de código en el núcleo para distinguir el tipo de procesador y , por lo tanto, sus características hardware. Las nuevas versiones del núcleo de Linux para ARM utilizan un mecanismo llamado “device tree” que describe los periféricos del procesador y que se carga con el núcleo en tiempo de arranque.

4. El Device Tree

En sistemas anteriores al uso del “device tree”, el núcleo contiene la descripción completa del hardware. El programa cargador (normalmente U-Boot) carga en memoria RAM la imagen binaria del núcleo (uImage) y la ejecuta. El programa cargador prepara también un conjunto de información (ATAGS) necesaria para el arranque (localización de la memoria, parámetros de arranque del núcleo…) en una zona de memoria cuya dirección se pasa al núcleo a través del registro R2. El núcleo distingue sobre qué hardware se está ejecutando porque el programa cargador le comunica un identificador “hardware type” a través del registro R1. Ver Figura 7 (a).

En los sistemas actuales que utilizan el “device tree”, el núcleo no contiene la descripción del hardware. Esta descripción se encuentra en un bloque binario separado del núcleo, el “Device Tree Blob” (DTB). El programa cargador (U-Boot) carga en memoria dos bloques binarios, la imagen del núcleo (uImage) y el DTB. Los fuentes de Linux, contienen un DT por cada placa compatible con la versión en cuestión. Podemos encontrar estos ficheros en: “https://github.com/beagleboard/

linux/tree/3.8/arch/arm/boot/dts". El cargador pasa la dirección de ubicación del DTB al núcleo mediante el registro R2. Observe que este mecanismo NO utiliza el "tipo de máquina" para informar al núcleo sobre cual es la plataforma subyacente. Ver Figura 7 (b).

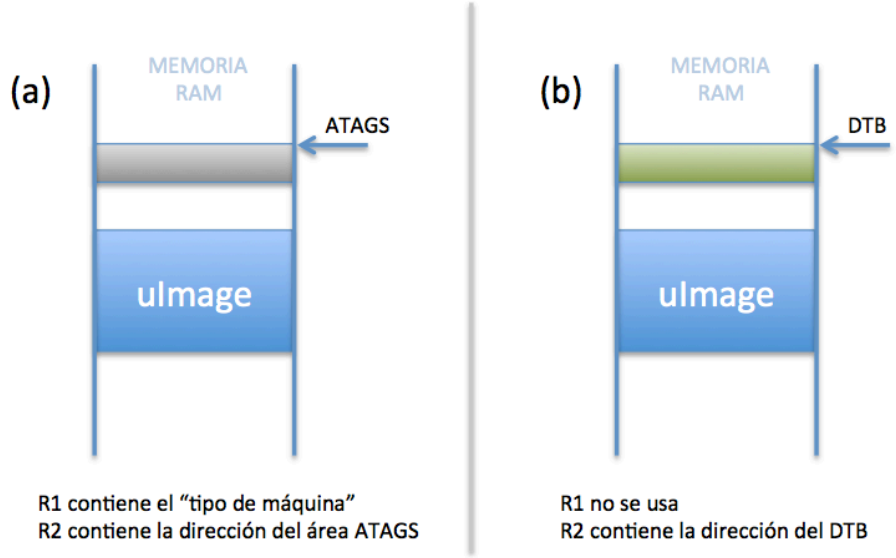


Figura 7: Carga de Linux. (a) sistemas antiguos sin Device Tree y (b) sistemas con Device Tree.

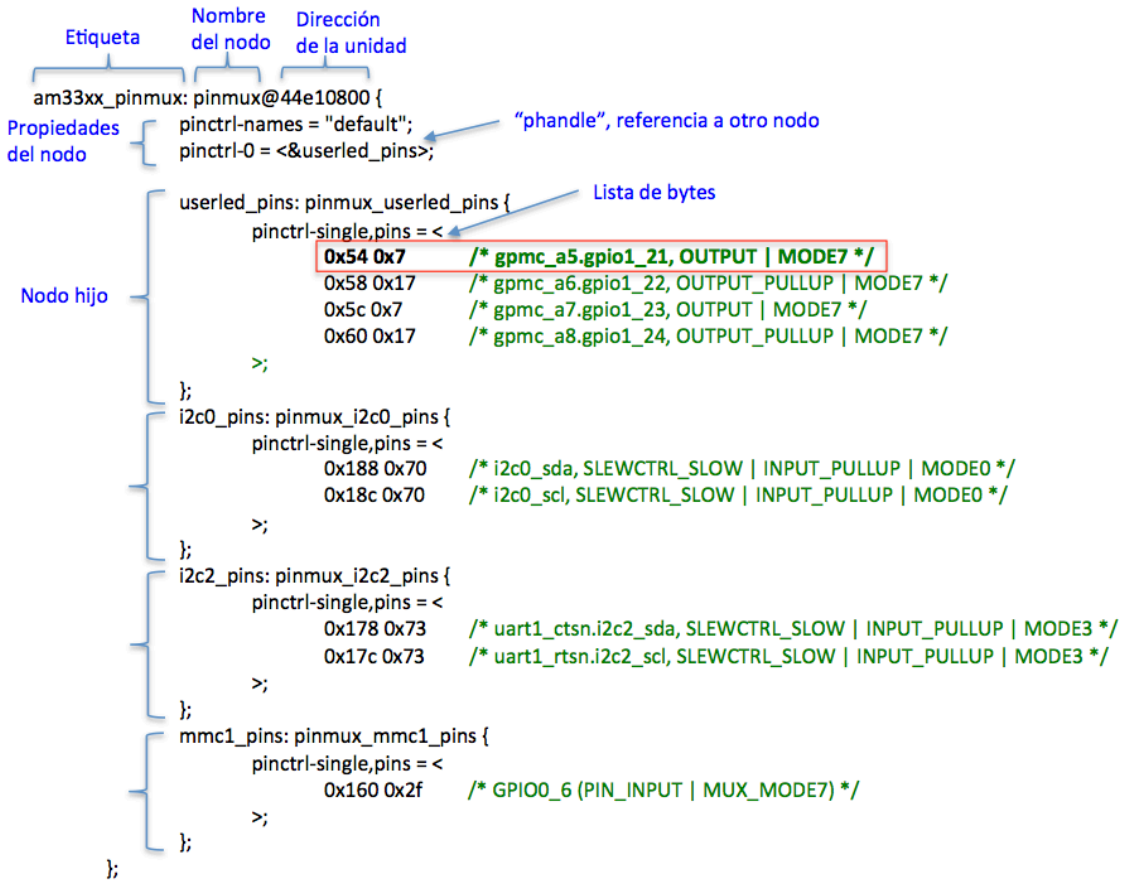
El "device tree" (DT) permite configurar una plataforma sin tener que recompilar el núcleo. El modelo resultante está completamente dirigido por datos, la lógica que antes se programaba en código del núcleo específico para la plataforma, se ha trasladado a una combinación de entornos genéricos: manejadores de dispositivo abstractos y una especificación de los datos que configuran y concretan el software genérico.

El "Flattened Device Tree" (FDT) es una estructura de datos simple que describe el hardware con el que cuenta la placa. La estructura consiste en un árbol de nodos que contienen propiedades especificadas en la forma de pares nombre-valor. La especificación del "device tree" se realiza mediante archivos de texto (formato "*.dts" ó "*.dtsi" para archivos "include") utilizando una sintaxis descrita en "http://devicetree.org/Device_ Tree_Usage".

Estos ficheros se compilan para obtener su representación binaria utilizando el "Device Tree Compiler" (DTC) . La imagen binaria (archivo *.dtb) hay que ubicarla en el directorio "/boot/dtbs/3.8.13-bone79". Estos ficheros se usan durante el proceso de arranque.

Para observar el contenido de los archivos "dts" de la distribución actual de Linux para diferentes plataformas, visite la página: https://github.com/beagleboard/linux/tree/3.8/arch/arm/boot/dts en particular encontrará los archivos que especifican el DT para la BBB.

A modo de ejemplo, observe el contenido del archivo "am335x-bone-common.dtsi". En él podrá encontrar la siguiente sección de código que configura los pines GPIO dedicados a los cuatro LEDs de usuario incluidos en la placa de la BBB.



En la línea resaltada del código puede ver que el pin "gpmc_a5", cuyo registro de configuración se encuentra ubicado en el offset 0x854 del "control module" (offset **0x54** respecto a la dirección de inicio de los registros de configuración "pinmux"), se ha configurado como salida (**0x07** (000 0111) Fast-Output-Pulldown-Enabled-MuxMode7).

En plataformas como la BBB, el DT que se carga en el arranque describe la configuración de los elementos que se encuentran integrados en la placa y la configuración por defecto de los pines que se exhiben en las cabezas de expansión P8 y P9. Esto resulta problemático precisamente por la naturaleza de los conectores de expansión, a los que se pueden conectar placas ("cape") con elementos hardware adicionales que necesiten una configuración concreta de los pines de P8 y P9.

Como el DT se carga durante el arranque, para conectar una placa de expansión "cape" a la BBB tendríamos que escribir y compilar un fragmento de DT, incluirlo en el sistema de ficheros (directorio "/boot/dtbs/3.8.13-bone79"), configurar el arranque para que cargara el DT correctamente y, finalmente reiniciar la BBB. Este proceso es especialmente molesto cuando cambiamos la "cape" de nuestra BBB con frecuencia. Para solucionar este problema, se ha ideado un método que permite modificar las características de la plataforma cargando fragmentos de descripciones DT en tiempo de ejecución: los "Device Tree Overlays" (DTO).

4.1. Device Tree Overlays

Para los casos en que no es conveniente definir completamente la plataforma mediante un solo FDT se ha ideado un sistema que permite superponer fragmentos de DT en modo usuario y, por lo tanto

en tiempo de ejecución o arranque. La definición de los fragmentos de DT (DT overlays) se hace con la misma sintaxis que el FDT, para obtener la imagen binaria de estos fragmentos también se usa el mismo compilador, el "dtc". Para gestionar la carga y descarga de "overlays" existe una utilidad, el "Cape Manager" (capemgr). El "Cape Manager" está implementado en el núcleo de forma que el usuario accede a sus funcionalidades a través de unos ficheros específicos.

Los fuentes de Linux para la BBB contienen la descripción (fragmentos de DT para usar como "overlays") de multitud de placas "cape" bien conocidas en https://github.com/beagleboard/linux/tree/3.8/firmware/capes. Cuando se compila el núcleo, estos fragmentos se compilan con el compilador "dtc" para generar su imagen binaria (Device Tree Blob Object "dtbo") y cuando se instala el Sistema Operativos, estos fragmentos binarios se copian en el directorio "/lib/firmware" de manera que quedan listos para que "Cape Manager" los pueda cargar.

4.1.1. Uso del "Cape Manager"

Cuando el "Cape Manager" carga un "overlay", identifica los nodos declarados en el código y los añade al DT. Si se ha declarado un nodo ya existente, añade las nuevas propiedades que se hayan declarado y sobre-escribe las propiedades declaradas que ya existieran.

Desde el punto de vista de usuario, podemos escribir fragmentos "overlay", compilarlos, copiarlos a /lib/firmware y cargarlos con capemgr. La carga y descarga dinámica de "overlays" se realiza a través del "sysfs" localizado en: "/sys/devices/bone_capemgr.9/slots" (el número 9 en la ruta puede cambiar entre versiones del capemgr). Por ejemplo:

Para ver los "overlays"

```
root@beaglebone:~# export SLOTS=/sys/devices/bone_capemgr.9/slots
root@beaglebone:~# cat $SLOTS
 0: 54:PF---
 1: 55:PF---
 2: 56:PF---
 3: 57:PF---
 4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
 5: ff:P-O-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
```

Para cargar un "overlay" que tenga "part-number" asignado como "miOverlay" (suponemos que ya tenemos el archivo *.dtbo en /lib/firmware):

```
root@beaglebone:~# echo miOverlay > $SLOTS
root@beaglebone:~# cat $SLOTS
 0: 54:PF---
 1: 55:PF---
 2: 56:PF---
 3: 57:PF---
 4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
 5: ff:P-O-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
 7: ff:P-O-L Override Board Name,00A0,Override Manuf, miOverlay
```

Para eliminar un "overlay" cargado dinámicamente, usamos el índice donde el "capemgr" lo ha ubicado:

```
root@beaglebone:~# echo -7 > $SLOTS
```

4.1.2. Ejemplo de overlay

Volviendo al ejemplo anterior, el pin “lcd_data0” tiene las siguientes características:

- Su registro de control se encuentra en la dirección 0x44E108A0 (offset 0x0A0 a partir de 0x44E10800)
- Es el pin 40. Este número es el índice que ocupa en el vector de configuración de pines cuya dirección de inicio es 0x44e10800. (0xA0 = 160d , 160/4 = 40)
- Está conectado al pin P8_45 de las cabezas de expansión
- Está configurado por defecto en MODO 0 y reservado para usarse en la “cape” LCD y controlador HDMI.
- En MODO 7 correspondería al GPIO2_6, que es el #GPIO 70. (32*2+6)

Supongamos que tenemos que usar este pin con su función GPIO2_6. Veamos qué operaciones tendremos que realizar:

1. Escribir el código fuente de un *overlay* que configure el pin
2. Compilar el *overlay* y copiarlo en /lib/firmware
3. Instalar el *overlay* resolviendo los conflictos que pudieran aparecer
4. Probar el funcionamiento del pin como GPIO

Para escribir el código de un *overlay*, tome como ejemplo y punto de partida alguno de los numerosos ejemplos de “https://github.com/beagleboard/linux/tree/3.8/firmware/capes”

1.- En nuestro caso, para cambiar simplemente la configuración de un pin, creamos un fichero con nombre “pinmuxGPIO206.dts” y el siguiente código (ver anexo)

```
/dts-v1/;
/plugin/;

/ {
    compatible = "ti,beaglebone", "ti,beaglebone-black";

    /* identification */
    part-number = "pinmuxGPIO206";
    version = "00A0";
        exclusive-use =
                /* the pin header uses */
                "P8.45";

    fragment@0 {
        target = <&am33xx_pinmux>;
        __overlay__ {

            gpio2_6_pin: pinmux_gpio2_6_pin {
                pinctrl-single,pins = <
                    0x0a0 0x07     /* P8_45 Fast-Output-Pulldown-Enabled-MuxMode7 */
                >;
            };

        };
    };

    fragment@1 {
        target = <&ocp>;
        __overlay__ {
            gpio_helper {
                compatible = "bone-pinmux-helper";
                status = "okay";
                pinctrl-names = "default";
                pinctrl-0 = <&gpio2_6_pin>;
            };
        };
    };
};
```

Observe que en el código se declara el "part-number" que es el nombre del *overlay* que usaremos para referirnos a él en el sistema.

2.- Compilamos el *overlay* y lo incorporamos al directorio de sistema:

```
root@beaglebone:~# dtc -O dtb -o pinmuxGPIO206-00A0.dtbo -b 0 -@ pinmuxGPIO206.dts
root@beaglebone:~# cp pinmuxGPIO206-00A0.dtbo /lib/firmware/
```

3.- Activamos el *overlay* escribiendo su "part-number" en el archivo "slots" del capemgr.:

```
root@beaglebone:~# export SLOTS=/sys/devices/bone_capemgr.9/slots
root@beaglebone:~# cat $SLOTS
 0: 54:PF---
 1: 55:PF---
 2: 56:PF---
 3: 57:PF---
 4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
 5: ff:P-O-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
root@beaglebone:~# echo pinmuxGPIO206 > $SLOTS
-bash: echo: write error: File exists
```

El error se produce porque el pin que intentamos utilizar ya está en uso por parte de otro *overlay*. En concreto, tal y como la función por defecto del pin indicaba, el pin lo usa el ovelay LCD/HDMI. Antes de poder usar el pin con la función GPIO que deseamos, tenemos que inhabilitar el *overlay* de LCD/HDMI en el proceso de arranque quitando el comentario de la línea correspondiente del archivo **/boot/uEnv.txt** para que quede así:

```
##Disable HDMI (v3.8.x)
cape_disable=capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN
```

y reiniciar la BBB.

Ahora podremos activar nuestro *overlay* sin problemas:

```
root@beaglebone:~# export SLOTS=/sys/devices/bone_capemgr.9/slots
root@beaglebone:~# export PINS=/sys/kernel/debug/pinctrl/44e10800.pinmux/pins
root@beaglebone:~# echo pinmuxGPIO206 > $SLOTS
root@beaglebone:~# cat $SLOTS
 0: 54:PF---
 1: 55:PF---
 2: 56:PF---
 3: 57:PF---
 4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
 5: ff:P-O-- Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
 6: ff:P-O-- Bone-Black-HDMIN,00A0,Texas Instrument,BB-BONELT-HDMIN
 7: ff:P-O-L Override Board Name,00A0,Override Manuf,pinmuxGPIO206
root@beaglebone:~/dts# cat $PINS | grep "pin 40"
pin 40 (44e108a0) 00000007 pinctrl-single
```

4.- Para probar el funcionamiento del pin, ponemos conectar un LED en P8_45 o medir su tensión con un multímetro y realizar las siguientes operaciones:

Nota: Si opta por conectar un LED, tendrá que poner en serie una resistencia de un valor alto (p.e 2K2) o utilizar un MOSFET para conmutar. Es muy IMPORTANTE que no conecte el LED directamente a la cabeza de expansión de la BBB. Debido a la limitación de intensidad que pueden drenar los pines de expansión puede dañar la placa.

El P8_45 es el #PIN 40, GPIO2_6 es el #GPIO 70.

```
root@beaglebone:~# cd /sys/class/gpio
root@beaglebone:/sys/class/gpio# echo 70 > export
root@beaglebone:/sys/class/gpio# cd gpio70
root@beaglebone:/sys/class/gpio/gpio70# echo out > direction
root@beaglebone:/sys/class/gpio/gpio70# echo 1 > value
root@beaglebone:/sys/class/gpio/gpio70# echo 0 > value
```

Nota: Si quiere que durante el proceso de arranque se cargue automáticamente un determinado *overlay* (por ejemplo el que hemos desarrollado en esta sección), simplemente tendrá que incluir el "part-number" del *overlay* en cuestión en el fichero "/etc/default/capemgr" así:

```
# Default settings for capemgr. This file is sourced by /bin/sh from
# /etc/init.d/capemgr.sh
# Options to pass to capemgr
CAPE=pinmuxGPIO206
```

5. Accediendo directamente a los registros GPIO

En la sección 2.1 del "AM335x TRM" se encuentra la tabla que describe el mapa de memoria del procesador. Ya sabemos que el "Control Module" se encuentra ubicado a partir de la dirección 0x44e1_0000 y que sólo podemos acceder a él cuando el procesador se encuentra en "modo privilegiado" (modo núcleo). También sabemos cómo solucionar esto y modificar el "pinmux" del "Control Module" aplicando *ovelays* al "Device Tree" del sistema.

Si seguimos observando la sección 2.1 del "AM335x TRM", encontramos la ubicación de los registros de control de las diferentes GPIO. Estos registros nos permiten configurar, encender y apagar los diferentes pines de cada GPIO y podemos acceder a ellos en modo usuario.

Siguiendo el ejemplo anterior, vamos a controlar el pin GPIO2_6 (encenderlo y apagarlo) escribiendo valores en los registros de control utilizando el manejador de dispositivo de mapeo de memoria física (/dev/mem ver Anexo phymem.*)

En la sección 2.1 del "AM335x TRM", Tablas 2.2 y 2.3, podemos localizar las direcciones de inicio de los registros de control de las GPIO.

UART5	0x481A_A000	0x481A_AFFF	4KB	UART5 Registers
	0x481A_B000	0x481A_BFFF	4KB	Reserved
GPIO2	0x481A_C000	0x481A_CFFF	4KB	GPIO2 Registers
	0x481A_D000	0x481A_DFFF	4KB	Reserved
GPIO3	0x481A_E000	0x481A_EFFF	4KB	GPIO3 Registers

Obtenemos los siguientes valores (usaremos el de GPIO2):

```
GPIO0 = 0x44E07000
GPIO1 = 0x4804C000
GPIO2 = 0x481AC000
GPIO3 = 0x481AE000
```

La disposición y los offsets de los diferentes registros de control que se ubican en esta área de memoria la podemos encontrar en la tabla 25.5 del TRM (ver Figura 8):

Offset	Acronym	Register Name	Section
0h	GPIO_REVISION		Section 25.4.1.1
10h	GPIO_SYSCONFIG		Section 25.4.1.2
20h	GPIO_EOI		Section 25.4.1.3
24h	GPIO_IRQSTATUS_RAW_0		Section 25.4.1.4
28h	GPIO_IRQSTATUS_RAW_1		Section 25.4.1.5
2Ch	GPIO_IRQSTATUS_0		Section 25.4.1.6
30h	GPIO_IRQSTATUS_1		Section 25.4.1.7
34h	GPIO_IRQSTATUS_SET_0		Section 25.4.1.8
38h	GPIO_IRQSTATUS_SET_1		Section 25.4.1.9
3Ch	GPIO_IRQSTATUS_CLR_0		Section 25.4.1.10
40h	GPIO_IRQSTATUS_CLR_1		Section 25.4.1.11
44h	GPIO_IRQWAKEN_0		Section 25.4.1.12
48h	GPIO_IRQWAKEN_1		Section 25.4.1.13
114h	GPIO_SYSSTATUS		Section 25.4.1.14
130h	GPIO_CTRL		Section 25.4.1.15
134h	GPIO_OE		Section 25.4.1.16
138h	GPIO_DATAIN		Section 25.4.1.17
13Ch	GPIO_DATAOUT		Section 25.4.1.18
140h	GPIO_LEVELDETECT0		Section 25.4.1.19
144h	GPIO_LEVELDETECT1		Section 25.4.1.20
148h	GPIO_RISINGDETECT		Section 25.4.1.21
14Ch	GPIO_FALLINGDETECT		Section 25.4.1.22
150h	GPIO_DEBOUNCENABLE		Section 25.4.1.23
154h	GPIO_DEBOUNCINGTIME		Section 25.4.1.24
190h	GPIO_CLEARDATAOUT		Section 25.4.1.25
194h	GPIO_SETDATAOUT		Section 25.4.1.26

Figura 8: Tabla 25.5 del TRM. Disposición y offset de los registros de control del GPIO.

En nuestro caso vamos a usar los siguientes registros:

- **GPIO_OE** (offset **0x134**) para habilitar el pin GPIO2_6 como salida.
- **GPIO_CLEARDATAOUT** (offset **0x190**) para apagar el pin.
- **GPIO_SETDATAOUT** (offset **0x194**) para encender el pin.

Cada bit de estos registros de 32 bits se refiere a un pin del GPIO en cuestión. Por ejemplo, para encender el GPIO2_6 tendremos que poner a “1” el bit número 6 (empieza desde cero, sería el valor **0x00000040**) del registro ubicado en **0x481AC194**. Para apagar el pin, escribiríamos el mismo valor en la dirección **0x481AC190**. Previamente a estas operaciones. Tendremos que habilitar el pin como salida usando el registro GPIO_OE. En este registro, la salida se habilita a nivel bajo, es decir, tendremos que escribir un “0” en el bit sexto del GPIO_OE (**0x481AC134**). Ver Figuras 9 y 10.

Figure 25-32. GPIO_SETDATAOUT Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTLINE[n]																															
R/W-0h																															

LEGEND: R/W = Read/Write; R = Read only; W1toCl = Write 1 to clear bit; *-n* = value after reset

Table 25-31. GPIO_SETDATAOUT Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	INTLINE[n]	R/W	0h	Set Data Output Register 0h = No effect 1h = Set the corresponding bit in the GPIO_DATAOUT register.

Figura 9: Tabla 25.32 y 31 del TRM. Formato del registro GPIO_SETDATAOUT, análogo al GPIO_CLEARDATAOUT.

Figure 25-22. GPIO_OE Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OUTPUTEN[n]																															
R/W-FFFFFFFFh																															

LEGEND: R/W = Read/Write; R = Read only; W1toCl = Write 1 to clear bit; -*n* = value after reset

Table 25-21. GPIO_OE Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	OUTPUTEN[n]	R/W	FFFFFFFFh	Output Data Enable 0h = The corresponding GPIO port is configured as an output. 1h = The corresponding GPIO port is configured as an input.

Figura 10: Tabla 25.22 y 21 del TRM. Formato del registro GPIO_OE.

Con esto ya deberíamos ser capaces de encender y apagar el pin escribiendo un programa en C que accediera a la memoria física a través del manejar de dispositivo /dev/mem. Y así sería si estuviéramos trabajando con la GPIO0 o la GPIO1.

Nuestro ejemplo es deliberadamente más complejo: Por defecto, el reloj del GPIO2 no está habilitado y cualquier operación de acceso a sus registros da lugar a un error ("Bus error").

Tenemos que habilitar el reloj del GPIO2. Esto se hace con los "Clock Module Peripheral Registers" (CM_PER) cuya dirección de inicio la encontramos en el TRM y es **0x44E0000**.

Reserved	0x44D8_3000	0x44DF_FFFF	500KB	Reserved
CM_PER	0x44E0_0000	0x44E0_3FFF	1KB	Clock Module Peripheral Registers
CM_WKUP	0x44E0_0400	0x44E0_04FF	256 Bytes	Clock Module Wakeup Registers

Dentro de esta zona, encontramos el registro de control del reloj del GPIO2 (CM_PER_GPIO2_CLKCTRL) con el offset **0xB0**.

ACh	CM_PER_GPIO1_CLKCTRL	Section 8.1.12.1.29
B0h	CM_PER_GPIO2_CLKCTRL	Section 8.1.12.1.30
B4h	CM_PER_GPIO3_CLKCTRL	Section 8.1.12.1.31

Los valores que hay que escribir en este registro para habilitar el reloj los encontramos en la Tabla 8-59 del TRM (ver Figura 11). Hay que escribir el valor **0x00000002** en la dirección **0x44E000B0**

Table 8-59. CM_PER_GPIO2_CLKCTRL Register Field Descriptions

Bit	Field	Type	Reset	Description
31-19	Reserved	R	0h	
18	OPTFCLKEN_GPIO_2_GDBCLK	R/W	0h	Optional functional clock control. 0x0 = FCLK_DIS : Optional functional clock is disabled 0x1 = FCLK_EN : Optional functional clock is enabled
17-16	IDLEST	R	3h	Module idle status. 0x0 = Func : Module is fully functional, including OCP 0x1 = Trans : Module is performing transition: wakeup, or sleep, or sleep abortion 0x2 = Idle : Module is in Idle mode (only OCP part). It is functional if using separate functional clock 0x3 = Disable : Module is disabled and cannot be accessed
15-2	Reserved	R	0h	
1-0	MODULEMODE	R/W	0h	Control the way mandatory clocks are managed. 0x0 = DISABLED : Module is disable by SW. Any OCP access to module results in an error, except if resulting from a module wakeup (asynchronous wakeup). 0x1 = RESERVED_1 : Reserved 0x2 = ENABLE : Module is explicitly enabled. Interface clock (if not used for functions) may be gated according to the clock domain state. Functional clocks are guarantied to stay present. As long as in this configuration, power domain sleep transition cannot happen. 0x3 = RESERVED : Reserved

Figura 11: Tabla "8-59. CM_PER_GPIO2_CLKCTRL Register Field Descriptions" del TRM

Con todo esto, para encender y apagar el pin GPIO2_6, tendremos que realizar las siguientes operaciones:

- Habilitar el reloj del GPIO2 escribiendo el valor 0x00000002 en la dirección 0x44E000B0
- Habilitar el pin GPIO2_6 como salida escribiendo el valor 0x0 (sólo necesitamos a cero el bit sexto) en la dirección 0x481AC134
- Encender el pin GPIO2_6 escribiendo el valor 0x0000_0040 en la dirección 0x481AC194
- Esperar un tiempo
- Apagar el pin GPIO2_6 escribiendo el valor 0x0000_0040 en la dirección 0x481AC190

Estas operaciones son las que realiza el siguiente programa que utiliza las rutinas de acceso a memoria física listadas en el Anexo.

```
#include "phymem.h"

int main(int argc, char **argv) {
	int i;
	char buffer[4];

	buffer[0]=0x02; //(enable clock)
	buffer[1]=0;
	buffer[2]=0;
	buffer[3]=0;
	writePhyMem(buffer, 4, 0x44E000B0);


	buffer[0]=0; //configure as output
	buffer[1]=0;
	buffer[2]=0;
	buffer[3]=0;
	writePhyMem(buffer, 4, 0x481AC134);

	buffer[0]=0x40;
	buffer[1]=0;
	buffer[2]=0;
	buffer[3]=0;
	writePhyMem(buffer, 4, 0x481AC194); //switch on the led
	usleep(5000000);
	writePhyMem(buffer, 4, 0x481AC190); //switch off the led

}
```

6. Conclusiones

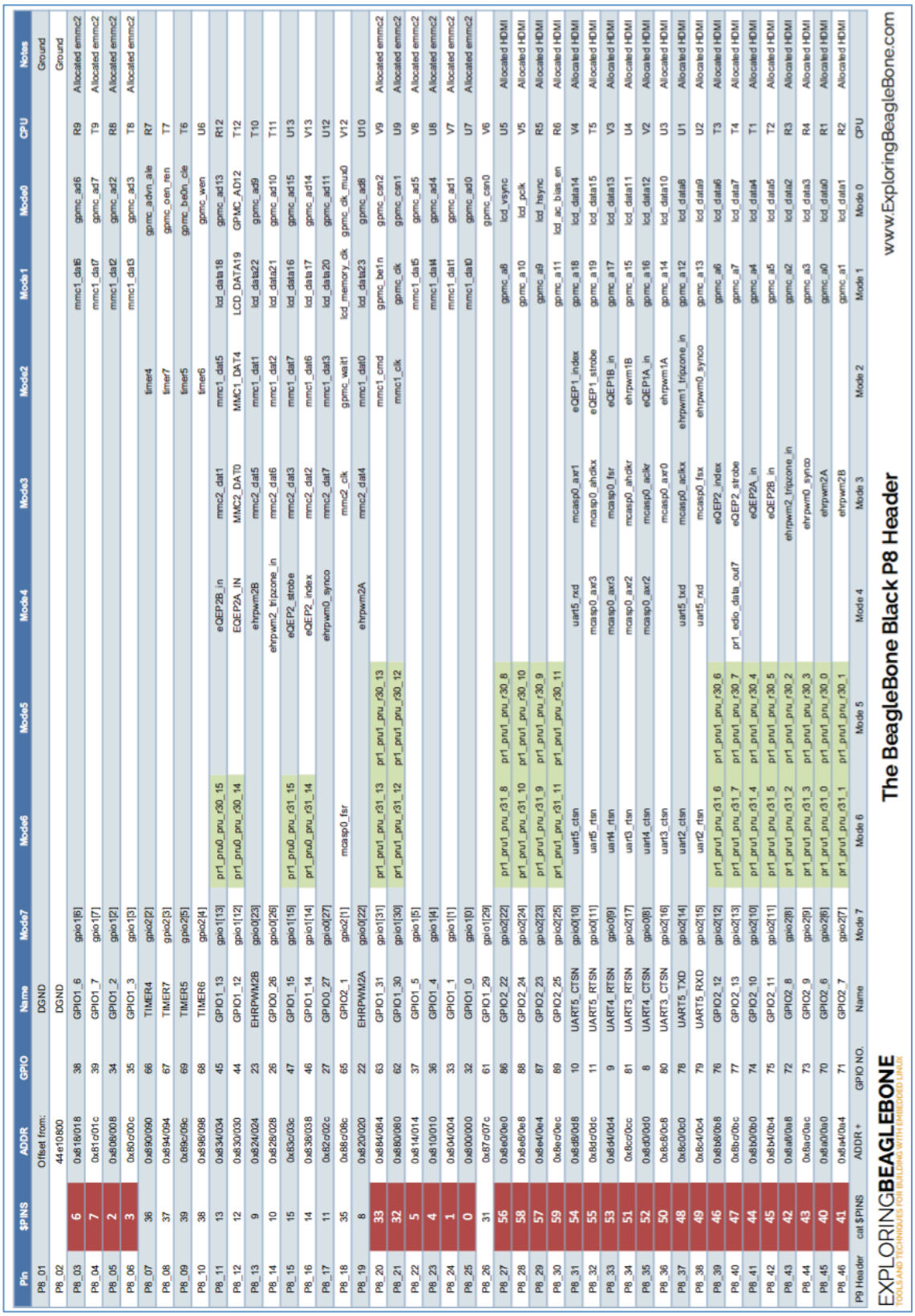
Después de trabajar con este documento habrá aprendido a configurar los pines de expansión de la BBB utilizando el “Cape Manager” y a buscar las diferentes funciones de cada pin utilizando la documentación del procesador AM335x y los esquemáticos de la BBB. A modo de resumen, recuerde que el interfaz de manejo del “Cape Manager” es un conjunto de archivos “sysfs” cuyas rutas conviene que establezca en variables de entorno así:

```
root@beaglebone:~# export SLOTS=/sys/devices/bone_capemgr.9/slots
root@beaglebone:~# export PINS=/sys/kernel/debug/pinctrl/44e10800.pinmux/pins
root@beaglebone:~# export PINMUX=/sys/kernel/debug/pinctrl/44e10800.pinmux/pinmux-pins
root@beaglebone:~# export PINGROUPS=/sys/kernel/debug/pinctrl/44e10800.pinmux/pingroups
```

7. Anexo I. Tablas

7.1. Tabla de funcionalidad de los pines de expansión (P8)

Descargue en alta resolución: https://github.com/derekmolloy/boneDeviceTree/tree/master/docs



Pin	$PINS	ADDR	GPIO	Name	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	Mode0	CPU	Notes
P8_01		Offset from:		DGND										Ground
P8_02		44e10800		DGND										Ground
P8_03	6	0x818/018	38	GPIO1_6	gpio1[6]						mmc1_dat6	gpmc_ad6	R9	Allocated emmc2
P8_04	7	0x81c/01c	39	GPIO1_7	gpio1[7]						mmc1_dat7	gpmc_ad7	T9	Allocated emmc2
P8_05	2	0x808/008	34	GPIO1_2	gpio1[2]						mmc1_dat2	gpmc_ad2	R8	Allocated emmc2
P8_06	3	0x80c/00c	35	GPIO1_3	gpio1[3]						mmc1_dat3	gpmc_ad3	T8	Allocated emmc2
P8_07	36	0x890/090	66	TIMER4	gpio2[2]				timer4			gpmc_advn_ale	R7	
P8_08	37	0x894/094	67	TIMER7	gpio2[3]				timer7			gpmc_oen_ren	T7	
P8_09	39	0x89c/09c	69	TIMER5	gpio2[5]				timer5			gpmc_be0n_cle	T6	
P8_10	38	0x898/098	68	TIMER6	gpio2[4]				timer6			gpmc_wen	U6	
P8_11	13	0x834/034	45	GPIO1_13	gpio1[13]	pr1_pru0_pru_r30_15		eQEP2B_in	mmc2_dat1	mmc1_dat5	lcd_data18	gpmc_ad13	R12	
P8_12	12	0x830/030	44	GPIO1_12	gpio1[12]	pr1_pru0_pru_r30_14		EQEP2A_IN	MMC2_DAT0	MMC1_DAT4	LCD_DATA19	GPMC_AD12	T12	
P8_13	9	0x824/024	23	EHRPWM2B	gpio0[23]			ehrpwm2B	mmc2_dat5	mmc1_dat1	lcd_data22	gpmc_ad9	T10	
P8_14	10	0x828/028	26	GPIO0_26	gpio0[26]		ehrpwm2_tripzone_in		mmc2_dat6	mmc1_dat2	lcd_data21	gpmc_ad10	T11	
P8_15	15	0x83c/03c	47	GPIO1_15	gpio1[15]	pr1_pru0_pru_r31_15		eQEP2_strobe	mmc2_dat3	mmc1_dat7	lcd_data16	gpmc_ad15	U13	
P8_16	14	0x838/038	46	GPIO1_14	gpio1[14]	pr1_pru0_pru_r31_14		eQEP2_index	mmc2_dat2	mmc1_dat6	lcd_data17	gpmc_ad14	V13	
P8_17	11	0x82c/02c	27	GPIO0_27	gpio0[27]			ehrpwm0_synco	mmc2_dat7	mmc1_dat3	lcd_data20	gpmc_ad11	U12	
P8_18	35	0x88c/08c	65	GPIO2_1	gpio2[1]	mcasp0_fsr			mmc2_clk	gpmc_wait1	lcd_memory_clk	gpmc_clk_mux0	V12	
P8_19	8	0x820/020	22	EHRPWM2A	gpio0[22]			ehrpwm2A	mmc2_dat4	mmc1_dat0	lcd_data23	gpmc_ad8	U10	
P8_20	33	0x884/084	63	GPIO1_31	gpio1[31]	pr1_pru1_pru_r31_13	pr1_pru1_pru_r30_13			mmc1_cmd	gpmc_be1n	gpmc_csn2	V9	Allocated emmc2
P8_21	32	0x880/080	62	GPIO1_30	gpio1[30]	pr1_pru1_pru_r31_12	pr1_pru1_pru_r30_12			mmc1_clk	gpmc_clk	gpmc_csn1	U9	Allocated emmc2
P8_22	5	0x814/014	37	GPIO1_5	gpio1[5]						mmc1_dat5	gpmc_ad5	V8	Allocated emmc2
P8_23	4	0x810/010	36	GPIO1_4	gpio1[4]						mmc1_dat4	gpmc_ad4	U8	Allocated emmc2
P8_24	1	0x804/004	33	GPIO1_1	gpio1[1]						mmc1_dat1	gpmc_ad1	V7	Allocated emmc2
P8_25	0	0x800/000	32	GPIO1_0	gpio1[0]						mmc1_dat0	gpmc_ad0	U7	Allocated emmc2
P8_26	31	0x87c/07c	61	GPIO1_29	gpio1[29]							gpmc_csn0	V6	
P8_27	56	0x8e0/0e0	86	GPIO2_22	gpio2[22]	pr1_pru1_pru_r31_8	pr1_pru1_pru_r30_8				gpmc_a8	lcd_vsync	U5	Allocated HDMI
P8_28	58	0x8e8/0e8	88	GPIO2_24	gpio2[24]	pr1_pru1_pru_r31_10	pr1_pru1_pru_r30_10				gpmc_a10	lcd_pclk	V5	Allocated HDMI
P8_29	57	0x8e4/0e4	87	GPIO2_23	gpio2[23]	pr1_pru1_pru_r31_9	pr1_pru1_pru_r30_9				gpmc_a9	lcd_hsync	R5	Allocated HDMI
P8_30	59	0x8ec/0ec	89	GPIO2_25	gpio2[25]	pr1_pru1_pru_r31_11	pr1_pru1_pru_r30_11				gpmc_a11	lcd_ac_bias_en	R6	Allocated HDMI
P8_31	54	0x8d8/0d8	10	UART5_CTSN	gpio0[10]	uart5_ctsn		uart5_rxd	mcasp0_axr1	eQEP1_index	gpmc_a18	lcd_data14	V4	Allocated HDMI
P8_32	55	0x8dc/0dc	11	UART5_RTSN	gpio0[11]	uart5_rtsn		mcasp0_axr3	mcasp0_ahclkx	eQEP1_strobe	gpmc_a19	lcd_data15	T5	Allocated HDMI
P8_33	53	0x8d4/0d4	9	UART4_RTSN	gpio0[9]	uart4_rtsn		mcasp0_axr3	mcasp0_fsr	eQEP1B_in	gpmc_a17	lcd_data13	V3	Allocated HDMI
P8_34	51	0x8cc/0cc	81	UART3_RTSN	gpio2[17]	uart3_rtsn		mcasp0_axr2	mcasp0_ahclkr	ehrpwm1B	gpmc_a15	lcd_data11	U4	Allocated HDMI
P8_35	52	0x8d0/0d0	8	UART4_CTSN	gpio0[8]	uart4_ctsn		mcasp0_axr2	mcasp0_aclkr	eQEP1A_in	gpmc_a16	lcd_data12	V2	Allocated HDMI
P8_36	50	0x8c8/0c8	80	UART3_CTSN	gpio2[16]	uart3_ctsn			mcasp0_axr0	ehrpwm1A	gpmc_a14	lcd_data10	U3	Allocated HDMI
P8_37	48	0x8c0/0c0	78	UART5_TXD	gpio2[14]	uart2_ctsn		uart5_txd	mcasp0_aclkx	ehrpwm1_tripzone_in	gpmc_a12	lcd_data8	U1	Allocated HDMI
P8_38	49	0x8c4/0c4	79	UART5_RXD	gpio2[15]	uart2_rtsn		uart5_rxd	mcasp0_fsx	ehrpwm0_synco	gpmc_a13	lcd_data9	U2	Allocated HDMI
P8_39	46	0x8b8/0b8	76	GPIO2_12	gpio2[12]	pr1_pru1_pru_r31_6	pr1_pru1_pru_r30_6		eQEP2_index		gpmc_a6	lcd_data6	T3	Allocated HDMI
P8_40	47	0x8bc/0bc	77	GPIO2_13	gpio2[13]	pr1_pru1_pru_r31_7	pr1_pru1_pru_r30_7	pr1_edio_data_out7	eQEP2_strobe		gpmc_a7	lcd_data7	T4	Allocated HDMI
P8_41	44	0x8b0/0b0	74	GPIO2_10	gpio2[10]	pr1_pru1_pru_r31_4	pr1_pru1_pru_r30_4		eQEP2A_in		gpmc_a4	lcd_data4	T1	Allocated HDMI
P8_42	45	0x8b4/0b4	75	GPIO2_11	gpio2[11]	pr1_pru1_pru_r31_5	pr1_pru1_pru_r30_5		eQEP2B_in		gpmc_a5	lcd_data5	T2	Allocated HDMI
P8_43	42	0x8a8/0a8	72	GPIO2_8	gpio2[8]	pr1_pru1_pru_r31_2	pr1_pru1_pru_r30_2		ehrpwm2_tripzone_in		gpmc_a2	lcd_data2	R3	Allocated HDMI
P8_44	43	0x8ac/0ac	73	GPIO2_9	gpio2[9]	pr1_pru1_pru_r31_3	pr1_pru1_pru_r30_3		ehrpwm0_synco		gpmc_a3	lcd_data3	R4	Allocated HDMI
P8_45	40	0x8a0/0a0	70	GPIO2_6	gpio2[6]	pr1_pru1_pru_r31_0	pr1_pru1_pru_r30_0		ehrpwm2A		gpmc_a0	lcd_data0	R1	Allocated HDMI
P8_46	41	0x8a4/0a4	71	GPIO2_7	gpio2[7]	pr1_pru1_pru_r31_1	pr1_pru1_pru_r30_1		ehrpwm2B		gpmc_a1	lcd_data1	R2	Allocated HDMI
P9 Header	cat $PINS	ADDR +	GPIO NO.	Name	Mode 7	Mode 6	Mode 5	Mode 4	Mode 3	Mode 2	Mode 1	Mode 0	CPU	

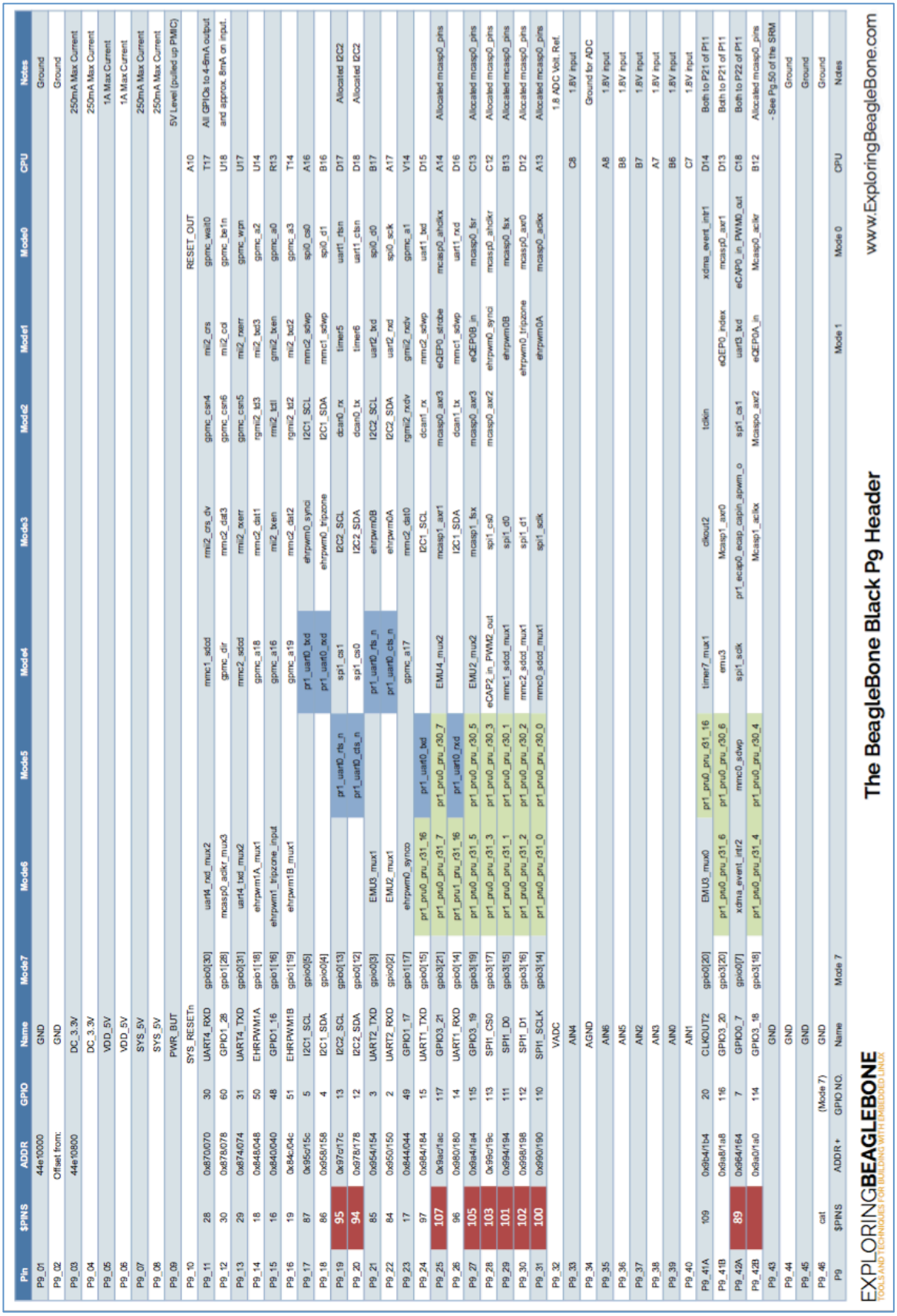
EXPLORINGBEAGLEBONE
TOOLS AND TECHNIQUES FOR BUILDING WITH EMBEDDED LINUX

The BeagleBone Black P8 Header

www.ExploringBeagleBone.com

7.2. Tabla de funcionalidad de los pines de expansión (P9)

Descargue en alta resolución: https://github.com/derekmolloy/boneDeviceTree/tree/master/docs



EXPLORINGBEAGLEBONE
TOOLS AND TECHNIQUES FOR BUILDING WITH EMBEDDED LINUX

The BeagleBone Black P9 Header

www.ExploringBeagleBone.com

Pin	$PINS	ADDR	GPIO	Name	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	Mode0	CPU	Notes
P9_01		44e10000		GND										Ground
P9_02		Offset from:		GND										Ground
P9_03		44e10800		DC_3.3V										250mA Max Current
P9_04				DC_3.3V										250mA Max Current
P9_05				VDD_5V										1A Max Current
P9_06				VDD_5V										1A Max Current
P9_07				SYS_5V										250mA Max Current
P9_08				SYS_5V										250mA Max Current
P9_09				PWR_BUT										5V Level (pulled up PMIC)
P9_10				SYS_RESETn								RESET_OUT	A10	
P9_11	28	0x870/070	30	UART4_RXD	gpio0[30]	uart4_rxd_mux2		mmc1_sdcd	mmi2_crs_dv	gpmc_csn4	mii2_crs	gpmc_wait0	T17	All GPIOs to 4-6mA output
P9_12	30	0x878/078	60	GPIO1_28	gpio1[28]	mcasp0_aclkr_mux3		gpmc_dir	mmi2_dat3	gpmc_csn6	mii2_col	gpmc_be1n	U18	and approx. 8mA on input.
P9_13	29	0x874/074	31	UART4_TXD	gpio0[31]	uart4_txd_mux2		mmc2_sdcd	mmi2_pxerr	gpmc_csn5	mii2_rxerr	gpmc_wpn	U17	
P9_14	18	0x848/048	50	EHRPWM1A	gpio1[18]	ehrpwm1A_mux1		gpmc_a18	mmc2_dat1	rgmii2_td3	rmii2_txd3	gpmc_a2	U14	
P9_15	16	0x840/040	48	GPIO1_16	gpio1[16]	ehrpwm1_tripzone_input		gpmc_a16	mii2_txen	rmii2_txd1	gmii2_txen	gpmc_a0	R13	
P9_16	19	0x84c/04c	51	EHRPWM1B	gpio1[19]	ehrpwm1B_mux1		gpmc_a19	mmc2_dat2	rgmii2_td2	mii2_txd2	gpmc_a3	T14	
P9_17	87	0x95c/15c	5	I2C1_SCL	gpio0[5]			pr1_uart0_txd	ehrpwm0_synci	I2C1_SCL	mmc2_sdwp	spi0_cs0	A16	
P9_18	86	0x958/158	4	I2C1_SDA	gpio0[4]			pr1_uart0_rxd	ehrpwm0_tripzone	I2C1_SDA	mmc1_sdwp	spi0_d1	B16	
P9_19	95	0x97c/17c	13	I2C2_SCL	gpio0[13]		pr1_uart0_rts_n	sp1_cs1	I2C2_SCL	dcan0_rx	timer5	uart1_rtsn	D17	Allocated I2C2
P9_20	94	0x978/178	12	I2C2_SDA	gpio0[12]		pr1_uart0_cts_n	sp1_cs0	I2C2_SDA	dcan0_tx	timer6	uart1_ctsn	D18	Allocated I2C2
P9_21	85	0x954/154	3	UART2_TXD	gpio0[3]	EMU3_mux1		pr1_uart0_rts_n	ehrpwm0B	I2C2_SCL	uart2_txd	spi0_d0	B17	
P9_22	84	0x950/150	2	UART2_RXD	gpio0[2]	EMU2_mux1		pr1_uart0_cts_n	ehrpwm0A	I2C2_SDA	uart2_rxd	spi0_sclk	A17	
P9_23	17	0x844/044	49	GPIO1_17	gpio1[17]	ehrpwm0_synco		gpmc_a17	mmc2_dat0	rgmii2_rxdv	gmii2_rxdv	gpmc_a1	V14	
P9_24	97	0x984/184	15	UART1_TXD	gpio0[15]	pr1_pru0_pru_r31_16	pr1_uart0_txd		I2C1_SCL	dcan1_rx	mmc2_sdwp	uart1_txd	D15	
P9_25	107	0x9ac/1ac	117	GPIO3_21	gpio3[21]	pr1_pru0_pru_r31_7	pr1_pru0_pru_r30_7	EMU4_mux2	mcasp1_axr1	mcasp0_axr3	eQEP0_strobe	mcasp0_ahclkx	A14	Allocated mcasp0_pins
P9_26	96	0x980/180	14	UART1_RXD	gpio0[14]	pr1_pru1_pru_r31_16	pr1_uart0_rxd		I2C1_SDA	dcan1_tx	mmc1_sdwp	uart1_rxd	D16	
P9_27	105	0x9a4/1a4	115	GPIO3_19	gpio3[19]	pr1_pru0_pru_r31_5	pr1_pru0_pru_r30_5	EMU2_mux2	mcasp1_fsx	mcasp0_axr3	eQEP0B_in	mcasp0_fsr	C13	Allocated mcasp0_pins
P9_28	103	0x99c/19c	113	SPI1_CS0	gpio3[17]	pr1_pru0_pru_r31_3	pr1_pru0_pru_r30_3	eCAP2_in_PWM2_out	spi1_cs0	mcasp0_axr2	eHRPWM_synci	mcasp0_ahclkr	C12	Allocated mcasp0_pins
P9_29	101	0x994/194	111	SPI1_D0	gpio3[15]	pr1_pru0_pru_r31_1	pr1_pru0_pru_r30_1	mmc1_sdcd_mux1	spi1_d0		ehrpwm0B	mcasp0_fsx	B13	Allocated mcasp0_pins
P9_30	102	0x998/198	112	SPI1_D1	gpio3[16]	pr1_pru0_pru_r31_2	pr1_pru0_pru_r30_2	mmc2_sdcd_mux1	spi1_d1		ehrpwm0_tripzone	mcasp0_axr0	D12	Allocated mcasp0_pins
P9_31	100	0x990/190	110	SPI1_SCLK	gpio3[14]	pr1_pru0_pru_r31_0	pr1_pru0_pru_r30_0	mmc0_sdcd_mux1	spi1_sclk		ehrpwm0A	mcasp0_aclkx	A13	Allocated mcasp0_pins
P9_32				VADC									C8	1.8 ADC Volt. Ref.
P9_33				AIN4										1.8V input
P9_34				AGND										Ground for ADC
P9_35				AIN6									A8	1.8V input
P9_36				AIN5									B8	1.8V input
P9_37				AIN2									B7	1.8V input
P9_38				AIN3									A7	1.8V input
P9_39				AIN0									B6	1.8V input
P9_40				AIN1									C7	1.8V input
P9_41A	109	0x9b4/1b4	20	CLKOUT2	gpio0[20]	EMU3_mux0	pr1_pru0_pru_r31_16	timer7_mux1	clkout2	tclkin		xdma_event_intr1	D14	Both to P21 of P11
P9_41B		0x9a8/1a8	116	GPIO3_20	gpio3[20]	pr1_pru0_pru_r31_6	pr1_pru0_pru_r30_6	emu3	Mcasp1_axr0		eQEP0_index	mcasp0_axr1	D13	Both to P21 of P11
P9_42A	89	0x964/164	7	GPIO0_7	gpio0[7]	xdma_event_intr2	mmc0_sdwp	spi1_sclk	pr1_ecasp0_ecasp_apwm_o	spi1_cs1	uart3_txd	eCAP0_in_PWM0_out	C18	Both to P22 of P11
P9_42B		0x9a0/1a0	114	GPIO3_18	gpio3[18]	pr1_pru0_pru_r31_4	pr1_pru0_pru_r30_4		Mcasp1_aclkx	Mcasp_axr2	eQEP0A_in	Mcasp0_aclkr	B12	Allocated mcasp0_pins
P9_43				GND										- See Pg.50 of the SRM
P9_44				GND										Ground
P9_45				GND										Ground
P9_46	cat		(Mode 7)	GND										Ground
P9	$PINS	ADDR+	GPIO NO.	Name	Mode 7						Mode 1	Mode 0	CPU	Notes

8. Anexo II. Código

8.1. Archivo “phymem.h”

```
#ifndef PHYMEM_H_
#define PHYMEM_H_

// Reads a physical memory chunk from "phyaddr" address of "size" bytes
// and copy the contents into "buff".
// The passed buffer must be pre-allocated to be "size" bytes long.
int readPhyMem(char *buff, unsigned long size, unsigned long phyaddr);
// The corresponding write operation.
int writePhyMem(char *buff, unsigned long size, unsigned long phyaddr);

#endif /* PHYMEM_H_ */
```

8.2. Archivo “phymem.c”

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <signal.h>
#include <fcntl.h>
#include <ctype.h>
#include <termios.h>
#include <sys/types.h>
#include <sys/mman.h>


#define PAGE_SIZE 4096UL
#define PAGE_MASK (PAGE_SIZE - 1)

int readPhyMem(char *buff, unsigned long size, unsigned long phyaddr) {
        int fd;
        void *map_base;

        if (size < 1) {
                fprintf(stderr, "Error: the buffer size must be non zero positive.\n");
                fflush(stderr);
                return -1;
        }
        if((fd = open("/dev/mem", O_RDONLY | O_SYNC)) == -1) {
                fprintf(stderr, "Error opening physical memory device \"/dev/mem\"\n");
                fflush(stderr);
                return -1;
        }
    /* Compute the alignment and the size of the physical memory to map. */
        unsigned long aligned_phyaddr = phyaddr & ~PAGE_MASK;
        unsigned long last_phyaddr = phyaddr + size - 1;
        unsigned long size_to_alocate = last_phyaddr - aligned_phyaddr + 1;
        unsigned long npages =  size_to_alocate / PAGE_SIZE +
                                ((size_to_alocate % PAGE_SIZE > 0) ? 1 : 0);

    /* Map "npages" of physical memory */
    map_base = mmap(0, PAGE_SIZE * npages, PROT_READ, MAP_SHARED, fd, aligned_phyaddr);
    if(map_base == (void *) -1) {
        fprintf(stderr, "Error mapping physical memory device into virtual address space.\n");
        fflush(stderr);
        return -1;
    }
    /* Copy the memory contents */
    unsigned long vaddr_start = (unsigned long)map_base + (phyaddr & PAGE_MASK);
    memcpy(buff,(char*)vaddr_start,size);
    /* Unmap the physical memory */
    if(munmap(map_base, PAGE_SIZE * npages) == -1) {
        fprintf(stderr, "Error unmapping physical memory device.");
        fflush(stderr);
        return -1;
    }
    close(fd);
```

```
    return size;
}

int writePhyMem(char *buff, unsigned long size, unsigned long phyaddr) {
        int fd;
        void *map_base;

        if (size < 1) {
                fprintf(stderr, "Error: the buffer size must be non zero positive.\n");
                fflush(stderr);
                return -1;
        }
        if((fd = open("/dev/mem", O_RDWR | O_SYNC)) == -1) {
                fprintf(stderr, "Error opening physical memory device \"/dev/mem\"\n");
                fflush(stderr);
                return -1;
        }
    /* Compute the alignment and the size of the physical memory to map. */
        unsigned long aligned_phyaddr = phyaddr & ~PAGE_MASK;
        unsigned long last_phyaddr = phyaddr + size - 1;
        unsigned long size_to_alocate = last_phyaddr - aligned_phyaddr + 1;
        unsigned long npages =  size_to_alocate / PAGE_SIZE +
                                ((size_to_alocate % PAGE_SIZE > 0) ? 1 : 0);

    /* Map "npages" of physical memory */
    map_base = mmap(0, PAGE_SIZE * npages, PROT_READ | PROT_WRITE, MAP_SHARED,
                    fd, aligned_phyaddr);
    if(map_base == (void *) -1) {
        fprintf(stderr, "Error mapping physical memory device into virtual address space.\n");
        fflush(stderr);
        return -1;
    }
    /* Copy the memory contents */
    unsigned long vaddr_start = (unsigned long)map_base + (phyaddr & PAGE_MASK);
    memcpy((char*)vaddr_start, buff, size);

    /* Unmap the physical memory */
        if(munmap(map_base, PAGE_SIZE * npages) == -1) {
                fprintf(stderr, "Error unmapping physical memory device.");
                fflush(stderr);
                return -1;
    }
    close(fd);
    return size;
}
```

8.1. Archivo “pinmuxGPIO206.dts”

```
/dts-v1/;
/plugin/;

/ {
    compatible = "ti,beaglebone", "ti,beaglebone-black";

    /* identification */
    part-number = "pinmuxGPIO206";
    version = "00A0";
        exclusive-use =
                /* the pin header uses */
                "P8.45";

    fragment@0 {
        target = <&am33xx_pinmux>;
        __overlay__ {

            gpio2_6_pin: pinmux_gpio2_6_pin {
                pinctrl-single,pins = <
                    0x0a0 0x07     /* P8_45 Fast-Output-Pulldown-Enabled-MuxMode7 */
                >;
            };

        };
    };

    fragment@1 {
```

```
        target = <&ocp>;
        __overlay__ {
            gpio_helper {
                 compatible = "bone-pinmux-helper";
                 status = "okay";
                 pinctrl-names = "default";
                 pinctrl-0 = <&gpio2_6_pin>;
             };
        };
    };
};
```

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Sistemas Empotrados

Master de Automática e Informática Industrial

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Guías BeagleBone Black (VI)

Entrada y Salida Analógica

Rev. José Simó. Enero 2016.

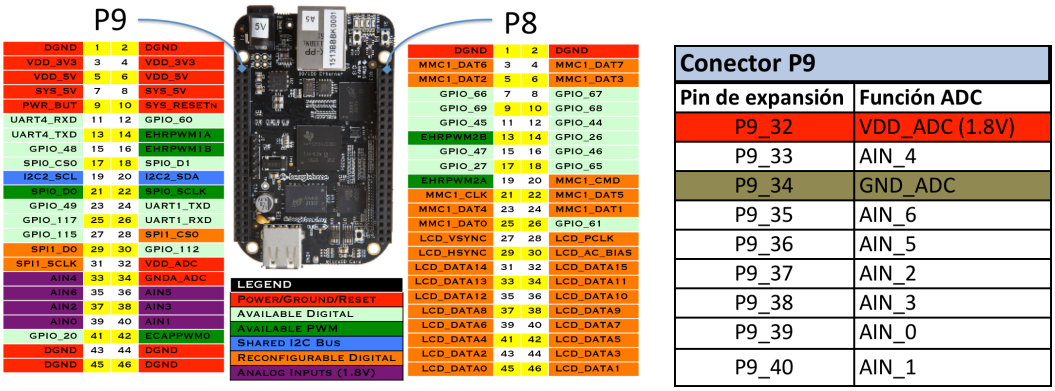
Contenido

1. Objetivos

En este documento revisamos los mecanismos de muestreo de señales analógicas (entradas) que pueden usarse para leer medidas de cualquier sensor que funcione por nivel de tensión (termómetros, sensores de distancia, resistencia variables, dinamos…). Por otro lado también se aborda la generación (salidas) de señales PWM (modulación por ancho de pulso) que pueden usarse para codificar información o para obtener una señal analógica con el valor eficaz instantáneo que queramos. Las señales PWM se usan ampliamente en el control de velocidad de motores, luminosidad de LEDs, posición de servo-motores etc.

2. Los pines de entrada a los conversores A/D

La placa de desarrollo BeagleBone Black (BBB) es una plataforma de desarrollo de bajo coste, libre y soportada por la comunidad de desarrolladores de código abierto. Los pines que conectan los periféricos de E/S integrados se ofrecen al usuario en la forma de dos cabezas de expansión (P8 y P9) cuya configuración por defecto aparece en la Figura 1.



Conector P9	
Pin de expansión	Función ADC
P9_32	VDD_ADC (1.8V)
P9_33	AIN_4
P9_34	GND_ADC
P9_35	AIN_6
P9_36	AIN_5
P9_37	AIN_2
P9_38	AIN_3
P9_39	AIN_0
P9_40	AIN_1

Figura 1: Asignación de los pines de expansión de la conversión Analógico/Digital.

En la Figura 1 observamos la disposición de los pines dedicados a entrada analógica (desde el P9_32 al P9_40). Estos pines soportan únicamente la función (modo) de conversión analógico/digital, es decir, no hay asignadas diferentes funciones en el mismo pin, por lo que no será necesario configurar el modo en el "pinmux" (multiplexado de los pines) para su utilización aunque sí que es necesario cargar un "overlay" específico para que los pines estén disponibles y los conversores A/D habilitados.

La BBB tiene 7 conversores analógico/digital de 12 bits que trabajan en el **rango 0V y 1.8V**. Esto quiere decir que para representar una señal analógica disponemos de 2^12 = 4096 posibilidades numéricas, que en el rango de operación corresponde a una resolución de unos 0.44mV.

La BBB, a través del pin de expansión P9_32 (y su tierra asociada en el pin P9_34), proporciona el nivel de tensión máximo de referencia para las entradas analógicas. Disponer de esta señal es especialmente útil cuando pretendemos leer valores de sensores basados en resistencia variable.

Es **IMPORTANTE** observar las siguientes restricciones eléctricas cuando operemos con los pines "AIN":

NO conectar a una entrada analógica una tensión fuera del rango de operación (0-1.8V).

NO debe hacer circular corriente por las entradas analógicas.

En general, para cumplir estas restricciones, es conveniente utilizar diodos de protección, amplificadores operacionales (configurados como seguidores de tensión) y/o divisores resistivos de tensión como interfaz en los pines de entrada analógica. Ver Figura 2.

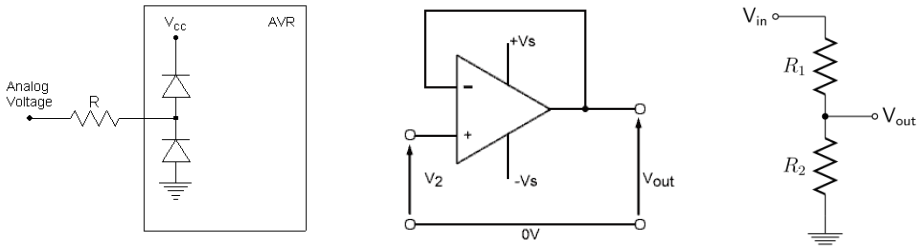


Figura 2: Circuitos útiles para proteger las entradas analógicas.

3. Lectura de valores A/D

Para probar la lectura de entradas analógicas montaremos el circuito simple de la Figura 3 que consiste en un pontenciómetro de 10kOhm conectado a la referencia de 1.8V.

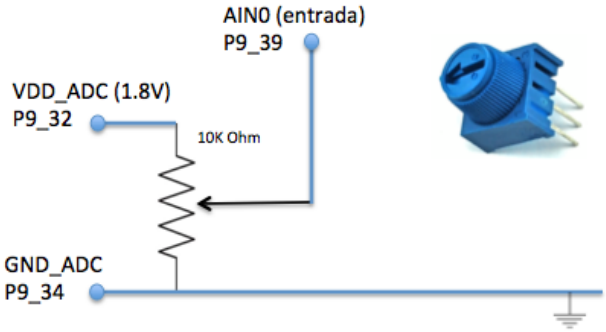


Figura 3: Circuito simple para la obtención de una tensión variable.

El Sistema Operativo Debian 3.8 proporciona unos manejadores de dispositivo que permiten interactuar con los conversores a través de archivos "sysfs".

En primer lugar, cargaremos el overlay (que está incluido en el S.O.) para habilitar los pines y conversores:

```
root@beaglebone:/# export PINS=/sys/kernel/debug/pinctrl/44e10800.pinmux/pins
root@beaglebone:/# export SLOTS=/sys/devices/bone_capemgr.9/slots

root@beaglebone:/# cd /sys/bus/iio
root@beaglebone:/sys/bus/iio# ls
devices  drivers  drivers_autoprobe  drivers_probe  uevent
root@beaglebone:/sys/bus/iio# cd devices
```

```
root@beaglebone:/sys/bus/iio/devices# ls
iio_sysfs_trigger
root@beaglebone:/sys/bus/iio/devices# echo BB-ADC > $SLOTS
root@beaglebone:/sys/bus/iio/devices# cat $SLOTS
 0: 54:PF---
 1: 55:PF---
 2: 56:PF---
 3: 57:PF---
 4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
 5: ff:P-O-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
 7: ff:P-O-L Override Board Name,00A0,Override Manuf,BB-ADC
root@beaglebone:/sys/bus/iio/devices# ls
iio:device0  iio_sysfs_trigger
```

Ahora, una vez instalado el dispositivo, podemos acceder a los valores de los conversores A/D utilizando el circuito de la Figura 3 conectado a la entrada AIN0 (in_voltage0_raw) leyendo el valor en diferentes posiciones del potenciómetro:

```
root@beaglebone:/sys/bus/iio/devices# cd iio:device0
root@beaglebone:/sys/bus/iio/devices/iio:device0# ls
dev             in_voltage1_raw  in_voltage3_raw  in_voltage5_raw  in_voltage7_raw  power      uevent
in_voltage0_raw  in_voltage2_raw  in_voltage4_raw  in_voltage6_raw  name             subsystem
root@beaglebone:/sys/bus/iio/devices/iio:device0#
root@beaglebone:/sys/bus/iio/devices/iio:device0# cat in_voltage0_raw
1505
root@beaglebone:/sys/bus/iio/devices/iio:device0# cat in_voltage0_raw
531
root@beaglebone:/sys/bus/iio/devices/iio:device0# cat in_voltage0_raw
0
root@beaglebone:/sys/bus/iio/devices/iio:device0# cat in_voltage0_raw
2754
root@beaglebone:/sys/bus/iio/devices/iio:device0# cat in_voltage0_raw
3460
root@beaglebone:/sys/bus/iio/devices/iio:device0# cat in_voltage0_raw
4094
```

Por otro lado, de una forma programática, en C++ podemos acceder a los valores de los conversores con este sencillo código:

```
#include <iostream>
#include <unistd.h>
#include <fstream>
#include <string>
#include <sstream>
using namespace std;

#define ADC_PATH "/sys/bus/iio/devices/iio:device0/in_voltage"

int readAnalog(int number){
   stringstream ss;
   ss << ADC_PATH << number << "_raw";
   fstream fs;
   fs.open(ss.str().c_str(), fstream::in);
   fs >> number;
   fs.close();
   return number;
}

int main() {
       cout << "ADC running" << endl;
       for (;;) {
          int value = readAnalog(0);
          cout << "The ADC value is " << value << " out of 4095." << endl;
          usleep(100000);
       }
       return 0;
}
```

3.1. Conexión de un sensor de proximidad

Utilizando las entradas analógicas de la BBB podemos conectar un sensor de distancia SHARP GP2Y0A21YK cuya salida es un nivel de tensión proporcional a la proximidad de un obstáculo. La curva característica y los detalles de conexión del sensor son los que aparecen en la Figura 4.

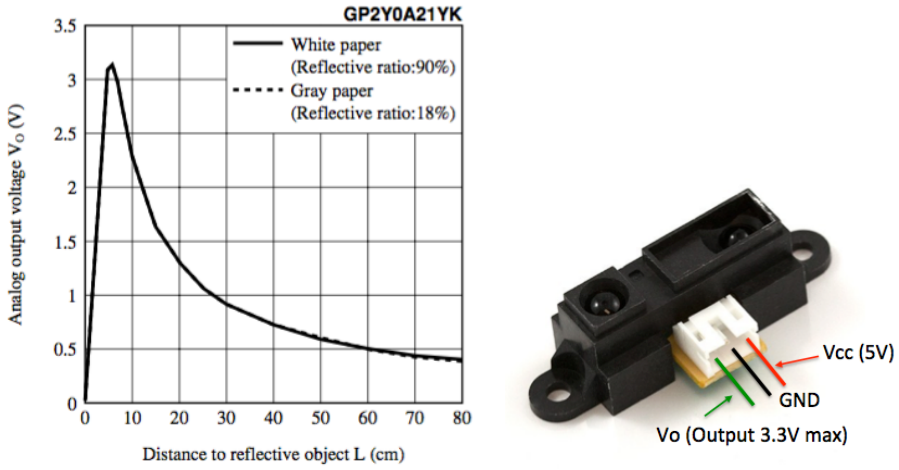


Figura 4: Características del sensor SHARP GP2Y0A21YK.

Como se aprecia en la curva característica del sensor, la respuesta es aproximadamente cuadrática con un valor máximo de unos 3.3V. Como el valor máximo aplicable a una entrada analógica es 1.8V, bastará con, aproximadamente, aplicar un divisor resistivo de tensión con relación ½. El circuito de conexión es el que se muestra en la Figura 5.

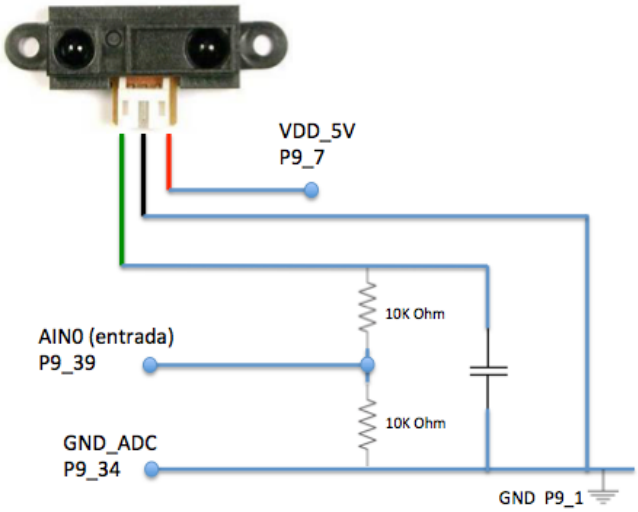


Figura 5: Conexión del sensor SHARP GP2Y0A21YK a la BBB.

En la Figura 5 el condensador es opcional y la capacidad del mismo depende del nivel de filtrado (estabilización) que deseemos para la señal a muestrear. En la Figura 6 se muestra el efecto que tiene la inclusión del condensador (izquierda sin condensador y derecha con él). Tenga en cuenta que al estabilizar la señal con un condensador también está afectando al tiempo de respuesta (tiempo de subida) del sensor.

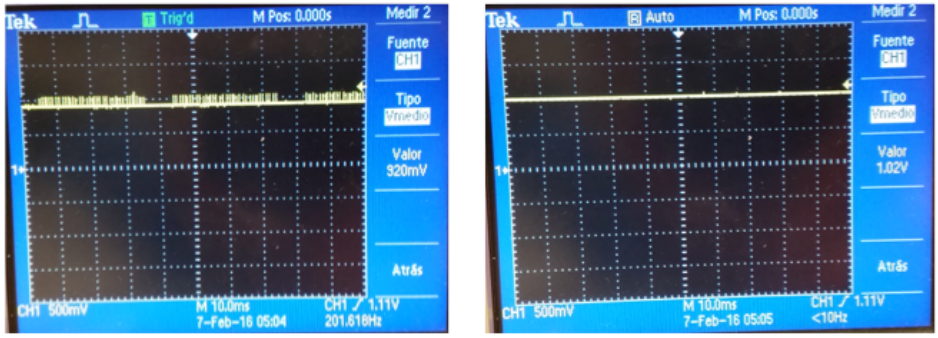


Figura 6: Señal filtrada con un condensador.

Utilice el mismo código que ha usado antes para leer el valor del sensor en la BBB.

4. Salidas PWM

Una señal PWM (modulación por ancho de pulso) es una señal cuadrada que se caracteriza por una frecuencia base y un "ciclo de trabajo" (duty cycle) que es el porcentaje del periodo en el que la señal está a nivel alto. Regulando el valor del ciclo de trabajo se puede obtener una señal con el valor eficaz que deseemos. Estos parámetros se ilustran en la Figura 7 así como un ejemplo de generación de una señal senoidal. Como regla general, la frecuencia base del PWM debe ser muy superior a la frecuencia de la señal que deseemos generar.

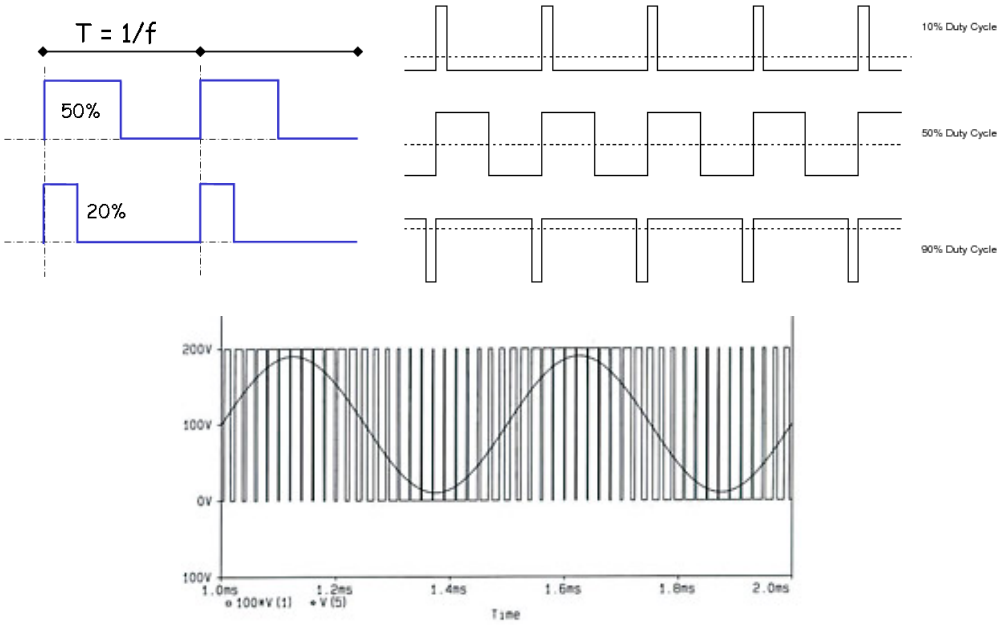


Figura 7: Parámetros característicos de una señal PWM.

La BBB dispone de 8 controladores PWM (chips) con los que puede generar 8 señales PWM independientes. Para poder generar una señal PWM es necesario:

- Configurar el multiplexado del pin para la función (modo) deseado, en este caso PWM. Esto se hace cargando los "overlays" correspondientes.
- Configurar el periférico (chip PWM) para la función del pin.

Los pines de expansión que podemos usar como salidas PWM son los que se relacionan en la Figura 8.

Número "Export"	Nombre del PIN	Pines de expansión
0	EHRPWM0A	P9.22,P9.31
1	EHRPWM0B	P9.21,P9.29
2	ECAPPWM0	P9.42
3	EHRPWM1A	P9.14,P8.36
4	EHRPWM1B	P9.16,P8.34
5	EHRPWM2A	P8.19,P8.45
6	EHRPWM2B	P8.13,P8.46
7	ECAPPWM2	P9.28

Figura 8: Controladores y pines PWM.

Como de costumbre, para poder usar el periférico PWM asociado a un determinado pin, tenemos que cargar el "overlay" correspondiente que se encuentra disponible en la distribución de Debian.

```
root@beaglebone:/# export SLOTS=/sys/devices/bone_capemgr.9/slots
root@beaglebone:~# echo bone_pwm_P9_22 > $SLOTS
root@beaglebone:~# echo am33xx_pwm > $SLOTS
```

Ahora ya podemos operar el PWM del pin P9_22 a través de los "sysfs".

```
root@beaglebone:~# cd /sys/devices/ocp.3/pwm_test_P9_22.*/
root@beaglebone:/sys/devices/ocp.3/pwm_test_P9_22.15# ls
driver  duty  modalias  period  polarity  power  run  subsystem  uevent
root@beaglebone:/sys/devices/ocp.3/pwm_test_P9_22.15# echo 0 > polarity
root@beaglebone:/sys/devices/ocp.3/pwm_test_P9_22.15# echo 20000000 > period  //en nanoseg.
root@beaglebone:/sys/devices/ocp.3/pwm_test_P9_22.15# echo 1 > run
root@beaglebone:/sys/devices/ocp.3/pwm_test_P9_22.15# echo 570000 > duty
root@beaglebone:/sys/devices/ocp.3/pwm_test_P9_22.15# echo 1460000 > duty
root@beaglebone:/sys/devices/ocp.3/pwm_test_P9_22.15# echo 2350000 > duty
root@beaglebone:/sys/devices/ocp.3/pwm_test_P9_22.15# echo 0 > run
```

Nota: el nombre del fichero "`pwm_test_P9_22.15`" puede cambiar (los últimos dos dígitos) dependiendo de cómo se hayan instalado los *overlays* durante el arranque y ejecución del sistema.

4.1. Control de posición de un servomotor

Los servomotores cuya posición se control con una señal PWM son muy populares en modelismo y robótica ligera. Simplemente el eje del servomotor se posiciona en un ángulo codificado mediante el ciclo de trabajo de una señal PWM. El montaje a realizar para conectarlo a la BBB es el de la Figura 9.

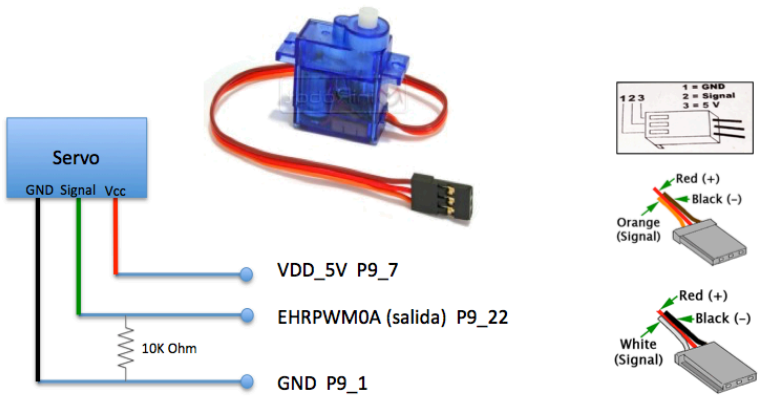


Figura 9: Conexión de un servomotor.

Antes de comenzar identifique correctamente los terminales del conector del servomotor. Ver Figura 9 en la que se muestran los casos más comunes. Es conveniente, para no dejar la salida PWM flotante, conectar una resistencia de “pull-down” en la salida PWM (es adecuado un valor elevado de resistencia p.e. 10KOhm). Si no lo hace, es posible que la BBB se comporte de manera no predecible (fallo en algún dispositivo o resets inesperados).

Tenga en cuenta que para el control de un servomotor, la frecuencia base debe ser de 50Hz (periodo de 20ms) y que el rango del ciclo de trabajo para mover el servo desde -90º a 90º es muy estrecho, de unos 2.4ms, tal y como puede apreciarse en la Figura 10.

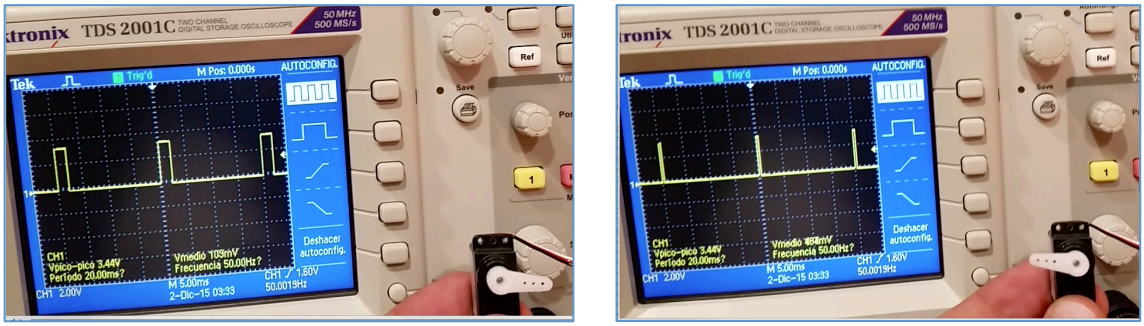


Figura 10: Rango del ciclo de trabajo del PWM para posicionar un servomotor.

Haga pruebas con el servo conectado al pin PWM P9_22 configurándolo tal y como se ha hecho anteriormente e introduciendo diferentes valores del ciclo de trabajo para averiguar los límites del ciclo de trabajo adecuados para el servo que está utilizando.

```
root@beaglebone:/sys/devices/ocp.3/pwm_test_P9_22.15# echo 20000000 > period
root@beaglebone:/sys/devices/ocp.3/pwm_test_P9_22.15# echo 1 > run
root@beaglebone:/sys/devices/ocp.3/pwm_test_P9_22.15# echo 570000 > duty
root@beaglebone:/sys/devices/ocp.3/pwm_test_P9_22.15# echo 1460000 > duty
root@beaglebone:/sys/devices/ocp.3/pwm_test_P9_22.15# echo 2350000 > duty
root@beaglebone:/sys/devices/ocp.3/pwm_test_P9_22.15# echo 0 > run
```

Desde un punto de vista programático, puede utilizar el código que se incluye en el anexo de este documento (PWM.h y PWM.cpp) para mover alternativamente el servo con el siguiente programa (ajuste los valores de los límites del ciclo de trabajo con los adecuado para el servo que esté utilizando).

Nota: Para compilar el programa necesitará también los archivos “util.cpp” y “util.h” incluidos en el anexo de la guía referente al GPIO. Tenga en cuenta que, como en el archivo “util.cpp” se utiliza la función “clock_gettime()”, para compilar el código tendrá que usar la opción “-lrt” que enlaza el código con la biblioteca correspondiente.

```
#include <unistd.h>
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
using namespace std;

#include"PWM.h"

using namespace exploringBB;
```

```
int main() {
	cout << "PWM running" << endl;

	PWM pwm("pwm_test_P9_22.15");
	pwm.setPeriod(20000000);		//50Hz
	pwm.setDutyCycle((unsigned)510000);
	pwm.setPolarity(PWM::ACTIVE_LOW);

	pwm.run();
	for (int i=0; i<100; i++) {
		pwm.setDutyCycle((unsigned)2350000);
		usleep(1000000);
		pwm.setDutyCycle((unsigned)500000);
		usleep(1000000);
	}
	pwm.stop();
	cout << "PWM exit" << endl;
	return 0;
}
```

4.2. Control de velocidad de un motor DC

Un motor DC gira a una velocidad proporcional al valor eficaz de la tensión con la que se alimenta. Para controlar la velocidad de giro de un motor DC con una señal PWM podemos realizar el típico montaje que utiliza un transistor MOSFET para hacer la conmutación, tal y como se indica en la Figura 11.

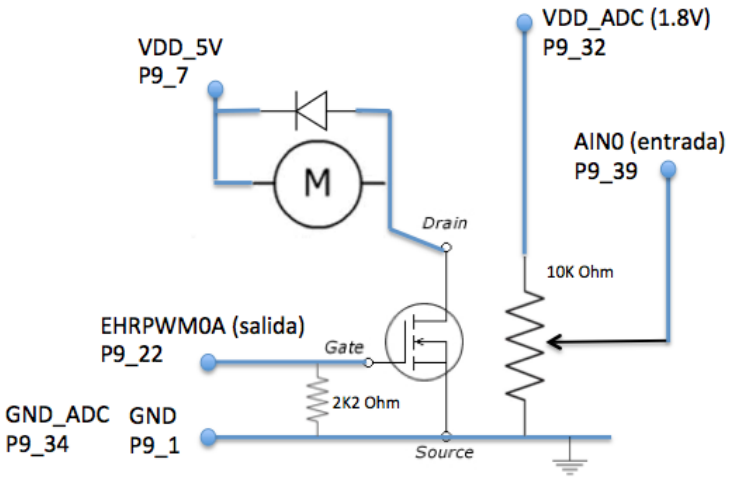


Figura 11: Variación de velocidad de un motor DC con señal PWM.

En este caso es importante que la frecuencia base del PWM sea al menos de 1kHz. Si prueba con frecuencias inferiores, podrá oír la conmutación de la señal PWM en los devanados del motor.

Con el montaje de la Figura 11 pruebe el siguiente programa para controlar la velocidad de un motor DC en función de la posición de un potenciómetro.

```
#include <unistd.h>
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
using namespace std;

#include"PWM.h"
```

```
using namespace exploringBB;
using namespace std;

#define LDR_PATH "/sys/bus/iio/devices/iio:device0/in_voltage"

int readAnalog(int number){
   stringstream ss;
   ss << LDR_PATH << number << "_raw";
   fstream fs;
   fs.open(ss.str().c_str(), fstream::in);
   fs >> number;
   fs.close();
   return number;
}

int main() {
    cout << "Start DC motor control..." << endl;
    unsigned int sval_min = 0;
      unsigned int sval_max = 1000000;
      PWM pwm("pwm_test_P9_22.15");
      pwm.setPolarity(PWM::ACTIVE_HIGH);
      pwm.setPeriod(1000000);  //1kHz
      pwm.setDutyCycle((unsigned)0);
      pwm.run();
      for (int i=0; i<100000; i++) {
            int value = readAnalog(0);
            double aval_pu = (double)value/4095.0;
            unsigned duty_ref = sval_min + aval_pu * (sval_max - sval_min);
            pwm.setDutyCycle(duty_ref);
            usleep(20000);
      }
      pwm.stop();
      return 0;
}
```

5. Ejercicio: Base “pan/tilt” para puntero láser

Si dispone de la base que aparece en la Figura 12, realice un programa que permita posicionar los dos grados de libertad de la base con sendos potenciómetros y encender el láser con un interruptor.

Si no dispone del montaje de la figura, simplemente controle un servo con la posición de un potenciómetro.

Figura 12: Base para apuntar puntero láser.

6. Conclusiones

Con esta guía ha aprendido el manejo de entradas y salidas analógicas con la BBB. Ha configurado el multiplexado de los pines y habilitado los controladores necesarios para la conversión AD y la generación de señales PWM.

7. Anexo I. Universal Cape Overlay

Una forma alternativa de manejar los pines PWM consiste en usar en primer lugar el "Universal Cape Overlay" para configurar los modos de los pines y posteriormente exportar el controlador PWM adecuado en el sysfs "/sys/class/pwm/".

El "Universal Cape Overlay" permite exportar los pines para posteriormente consultar su estado o manipular su modo con la utilidad "config-pin". El sistema Debian incluye el overlay con las siguientes variantes:

- **cape-universal**: Exporta todos los pines no usados por HDMIN o por eMMC (incluyendo audio)
- **cape-universaln**: Exporta todos los pines no usados por HDMI o por eMMC (los pines de autio no se exportan)
- **cape-univ-emmc**: Exporta los pines usados por eMMC. Se carga si la eMMC está deshabilitada.
- **cape-univ-hdmi**: Exporta los pines usados por HDMI. Se carga si el HDMI está deshabilitado.
- **cape-univ-audio**: Exporta los pines usados por HDMI audio

Nota: El código de este overlay se encuentra en:
https://github.com/cdsteinkuehler/beaglebone-universal-io/blob/master/cape-universal-00A0.dts
Si lo considera conveniente puede actualizar el código fuente de su sistema e instalar la versión más actual del overlay así:

```
$ cd /opt/source/beaglebone-universal-io
$ git pull origin master
$ make install
```

En nuestro caso, lo podemos usar de la siguiente forma:

```
root@beaglebone:~# export SLOTS=/sys/devices/bone_capemgr.9/slots
root@beaglebone:~# echo cape-universaln > $SLOTS
```

Esto crea archivos en el directorio "/sys/devices/ocp.*" para cada uno de los pines de expansión exportados. Estos archivos (en realidad son directorios) tienen la extensión "pinmux" y pueden usarse para definir el multiplexado de los pines usando la utilidad "config-pin" que se encuentra en el directorio "/opt/source/beaglebone-universal-io". Así:

```
root@beaglebone:~# cd /opt/source/beaglebone-universal-io
root@beaglebone:/opt/source/beaglebone-universal-io# config-pin -l P9.22
default gpio gpio_pu gpio_pd spi uart i2c pwm
root@beaglebone:/opt/source/beaglebone-universal-io# config-pin P9.22 pwm
root@beaglebone:/opt/source/beaglebone-universal-io# cd /sys/devices/ocp.3/P9_22_pinmux.*
root@beaglebone:/sys/devices/ocp.3/P9_22_pinmux.35# ls
driver  modalias  power  state  subsystem  uevent
root@beaglebone:/sys/devices/ocp.3/P9_22_pinmux.35# cat state
pwm
```

Una vez configurado el pin, tenemos que habilitar el controlador PWM adecuado (el que corresponde al pin que queremos usar). Para ello usaremos el sysfs "/sys/class/pwm/" y los ficheros "export" y "unexport" que contiene:

```
root@beaglebone:~# cd /sys/class/pwm/
root@beaglebone:/sys/class/pwm# ls -l
total 0
--w------- 1 root root 4096 Jan  1  2000 export
lrwxrwxrwx 1 root root    0 Oct 30 15:41 pwmchip0 -> ../../devices/ocp.3/48300000.epwmss/48300200.ehrpwm/pwm/pwmchip0
lrwxrwxrwx 1 root root    0 Oct 30 15:41 pwmchip2 -> ../../devices/ocp.3/48300000.epwmss/48300100.ecap/pwm/pwmchip2
lrwxrwxrwx 1 root root    0 Oct 30 15:41 pwmchip3 -> ../../devices/ocp.3/48302000.epwmss/48302200.ehrpwm/pwm/pwmchip3
lrwxrwxrwx 1 root root    0 Oct 30 15:41 pwmchip5 -> ../../devices/ocp.3/48304000.epwmss/48304200.ehrpwm/pwm/pwmchip5
lrwxrwxrwx 1 root root    0 Oct 30 15:41 pwmchip7 -> ../../devices/ocp.3/48304000.epwmss/48304100.ecap/pwm/pwmchip7
--w------- 1 root root 4096 Jan  1  2000 unexport
```

Siguiendo la pista de las direcciones de memoria que aparecen en los enlaces simbólicos y con ayuda del "Technical Reference Manual" de Texas Instruments, podemos averiguar qué número tenemos que utilizar en el "export" para un determinado pin. Esta información está recogida en la tabla de la Figura 8. Consultando esta tabla, para el pin P9_22, el controlado PWM que tenemos que exportar es el "0".

Habilitaremos el controlador así:

```
root@beaglebone:/sys/class/pwm# echo 0 > export
root@beaglebone:/sys/class/pwm# ls
export  pwm0  pwmchip0  pwmchip2  pwmchip3  pwmchip5  pwmchip7  unexport
```

Observamos que ha aparecido el sysfs "pwm0" que podemos usar configurar la señal PWM del pin P9_22 tal y como se ha hecho anteriormente.

```
root@beaglebone:/sys/class/pwm# cd pwm0
root@beaglebone:/sys/class/pwm/pwm0# ls
device  duty_ns  period_ns  polarity  power  run  subsystem  uevent
root@beaglebone:/sys/class/pwm/pwm0# echo 1 > run
```

8. Anexo II. Código

8.1. Archivo “PWM.h”

```
#ifndef PWM_H_
#define PWM_H_
#include<string>
using std::string;

#define PWM_PATH "/sys/devices/ocp.3/"
#define PWM_PERIOD "period"
#define PWM_DUTY "duty"
#define PWM_POLARITY "polarity"
#define PWM_RUN "run"

namespace exploringBB {

class PWM {
public:
	enum POLARITY{ ACTIVE_HIGH=0, ACTIVE_LOW=1 };

private:
	string name, path;
	float analogFrequency;  //defaults to 100,000 Hz
	float analogMax;        //defaults to 3.3V

public:
	PWM(string pinName);

	virtual int setPeriod(unsigned int period_ns);
	virtual unsigned int getPeriod();
	virtual int setFrequency(float frequency_hz);
	virtual float getFrequency();
	virtual int setDutyCycle(unsigned int duration_ns);
	virtual int setDutyCycle(float percentage);
	virtual unsigned int getDutyCycle();
	virtual float getDutyCyclePercent();

	virtual int setPolarity(PWM::POLARITY);
	virtual void invertPolarity();
	virtual PWM::POLARITY getPolarity();

	virtual void setAnalogFrequency(float frequency_hz) {
			this->analogFrequency = frequency_hz;
			}
	virtual int calibrateAnalogMax(float analogMax); //must be between 3.2 and 3.4
	virtual int analogWrite(float voltage);

	virtual int run();
	virtual bool isRunning();
	virtual int stop();

	virtual ~PWM();
private:
	float period_nsToFrequency(unsigned int);
	unsigned int frequencyToPeriod_ns(float);
};

} /* namespace exploringBB */

#endif /* PWM_H_ */
```

8.2. Archivo “PWM.cpp”

```
#include "PWM.h"
#include "util.h"
#include <cstdlib>

namespace exploringBB {

PWM::PWM(string pinName) {
	this->name = pinName;
	this->path = PWM_PATH + this->name + "/";
```

```
        this->analogFrequency = 100000;
        this->analogMax = 3.3;
}

int PWM::setPeriod(unsigned int period_ns){
        return write(this->path, PWM_PERIOD, period_ns);
}

unsigned int PWM::getPeriod(){
        return atoi(read(this->path, PWM_PERIOD).c_str());
}

float PWM::period_nsToFrequency(unsigned int period_ns){
        float period_s = (float)period_ns/1000000000;
        return 1.0f/period_s;
}

unsigned int PWM::frequencyToPeriod_ns(float frequency_hz){
        float period_s = 1.0f/frequency_hz;
        return (unsigned int)(period_s*1000000000);
}

int PWM::setFrequency(float frequency_hz){
        return this->setPeriod(this->frequencyToPeriod_ns(frequency_hz));
}

float PWM::getFrequency(){
        return this->period_nsToFrequency(this->getPeriod());
}

int PWM::setDutyCycle(unsigned int duty_ns){
        return write(this->path, PWM_DUTY, duty_ns);
}

int PWM::setDutyCycle(float percentage){
        if ((percentage>100.0f)||(percentage<0.0f)) return -1;
        unsigned int period_ns = this->getPeriod();
        float duty_ns = period_ns * (percentage/100.0f);
        this->setDutyCycle((unsigned int) duty_ns );
        return 0;
}

unsigned int PWM::getDutyCycle(){
        return atoi(read(this->path, PWM_DUTY).c_str());
}

float PWM::getDutyCyclePercent(){
        unsigned int period_ns = this->getPeriod();
        unsigned int duty_ns = this->getDutyCycle();
        return 100.0f * (float)duty_ns/(float)period_ns;
}

int PWM::setPolarity(PWM::POLARITY polarity){
        return write(this->path, PWM_POLARITY, polarity);
}

void PWM::invertPolarity(){
        if (this->getPolarity()==PWM::ACTIVE_LOW) this->setPolarity(PWM::ACTIVE_HIGH);
        else this->setPolarity(PWM::ACTIVE_LOW);
}

PWM::POLARITY PWM::getPolarity(){
        if (atoi(read(this->path, PWM_POLARITY).c_str())==0) return PWM::ACTIVE_LOW;
        else return PWM::ACTIVE_HIGH;
}

int PWM::calibrateAnalogMax(float analogMax){ //must be between 3.2 and 3.4
        if((analogMax<3.2f) || (analogMax>3.4f)) return -1;
        else this->analogMax = analogMax;
        return 0;
}

int PWM::analogWrite(float voltage){
        if ((voltage<0.0f)||(voltage>3.3f)) return -1;
        this->setFrequency(this->analogFrequency);
        this->setPolarity(PWM::ACTIVE_LOW);
        this->setDutyCycle((100.0f*voltage)/this->analogMax);
        return this->run();
}
```

```
int PWM::run(){
        return write(this->path, PWM_RUN, 1);
}

bool PWM::isRunning(){
        string running = read(this->path, PWM_RUN);
        return (running=="1");
}

int PWM::stop(){
        return write(this->path, PWM_RUN, 0);
}

PWM::~PWM() {}

} /* namespace exploringBB */
```

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Sistemas Empotrados

Master de Automática e Informática Industrial

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Guías BeagleBone Black (VII)
Comunicaciones I2C y UART

Rev. Francisco Blanes. Febrero 2016.

Contenido

1. Objetivos

Frecuentemente en el desarrollo de sistemas empotrados, es necesario conectar nuestras aplicaciones a otros sistemas por medio de buses de comunicación. Estos buses nos permiten la obtención de información sensorial (en el caso de buses de instrumentación y sensores), o también la cooperación y coordinación con otros nodos de computo, como es el caso de los sistemas distribuidos.

Con esta visión en mente, en esta guía se van a desarrollar los conocimientos necesarios para comunicar la placa BBB a sensores por medio de un bus I2C (Inter Integrated Circuit), así como con un nodo de computo (el host Linux de la MV) para el intercambio de datos del sistema por medio un bus UART. Ambos son ejemplo de buses serie orientados a comunicaciones en un entorno relativamente cercano físicamente. Capítulo a parte requiere el caso de los sistemas distribuidos basados en redes de área local, mediante protocolo de alto nivel, aspecto este que no será tratado en esta guía.

2. La RoboCape

La placa de desarrollo BeagleBone Black (BBB) es una plataforma de desarrollo de bajo coste, libre y soportada por la comunidad de desarrolladores de código abierto. Los pines que conectan los periféricos de E/S integrados se ofrecen al usuario en la forma de dos cabezas de expansión (P8 y P9) cuya configuración por defecto aparece en la Figura 1.

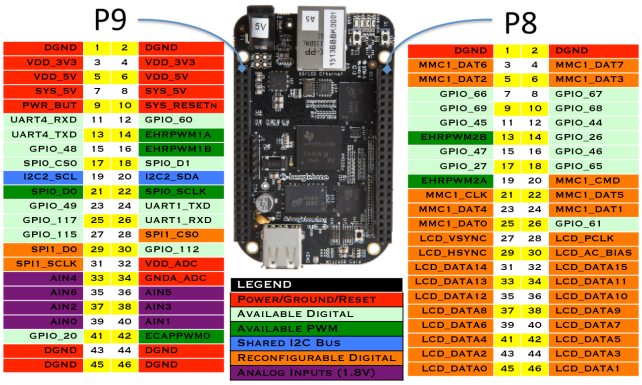


Figura 1: Asignación de los pines de expansión, en esta practica serán de relevancia los de los buses I2C y UART.

Una de las características mas relevantes del universo de la BBB es la facilidad con la que terceros desarrolladores de HW pueden diseñar placas accesorias que se conecten al sistema por medio de los conectores P8 y P9. Un ejemplo de este tipo de desarrollo es la Robocape.

Robocape es una placa electrónica especialmente diseñada para su uso con BeagleBoneBlack (BBB). El uso conjunto de ambas posibilita el diseño de cualquier tipo de sistema róbotico, y su programación. En el diseño de la Robocape, se ha tenido muy en cuenta la necesidad de ofrecer un dispositivo que contenga la gran mayoría de periféricos que los usuarios y diseñadores de robots habitualmente utilizan en sus diseños.

Sus características principales son:

- Sistema de alimentación DC/DC independiente, capaz de ofrecer hasta 3A y alimentar directamente a BBB.
- Cargador de baterías lipo de 2 células.
- 4 conexiones pwm para servos mediante buffer de salida.
- 4 puentes H para control de motores.
- 2 entradas para encoder.
- 6 conectores para sensores de ultrasonido HC-SR04.
- 4 conectores para sensores tipo Sharp de salida analógica.
- 2 conectores para conexión red servos Dynamixel (RS-485).
- 2 puertos serie con salida digital 3v3.
- IMU 9 ejes (MPU-9150).
- Altímetro y termómetro.
- GPS integrado (FGPMMOPA6H), con toma para antena externa.
- HUB USB 2.0 de 4 puertos

Requerimientos:

- La Robocape no es compatible 100% con versiones BeagleBone anteriores.
- Robocape puede ser alimentada desde la propia BBB, si bien no es suficiente con la alimentación de USB de la placa.
- Robocape requiere de un valor de entrada en la alimentación de entre
- 6-12V.
- Soporta carga de baterías de 2 celulas2, por lo que si queremos cargar
- este tipo de baterías la tensión de alimentación deberá ser entre 9-12V.
- Cable USB TypeA-miniUSB para la interconexión del HUB que incorpora
- Robocape con BBB.
- Sensores ultrasonido tipología HC-SR04.
- Entradas analógicas max. 3V.
- Antena externa conexión uFL.

2.1. Descripción de la placa

Para poder trabajar con seguridad con Robocape, debemos conocer cuál es la distribución de periféricos y pines dentro del esquema que ofrece BBB. Así pues a continuación se muestran una primera descripción de la distribución física de cada uno de los componentes que integran el diseño Robocape, así como el pinout y su interacción con BBB.

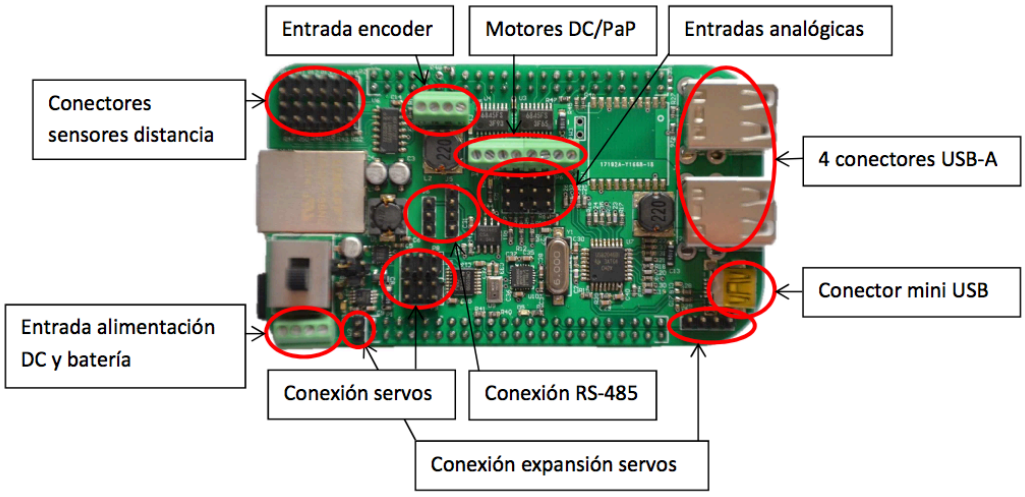


Figura 2: Descripción de conectores y su distribución.

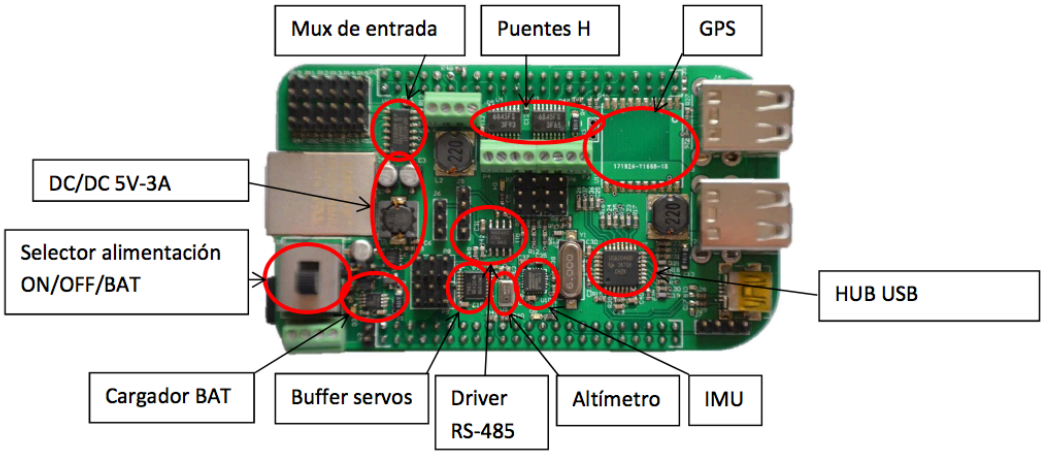


Figura 3: Descripción de los módulos y IC`s y su distribución.

2.2. Alimentación y batería

Robocape posee un conector propio (J1) para toma de alimentación, esta deberá ser de tensión continua DC de al menos 1A en el caso de que estemos alimentando a BBB, y sin tener en cuenta el número de periféricos y motores que tenemos conectados a la placa. El valor máximo de alimentación que puede ser introducido al sistema es de 12V.

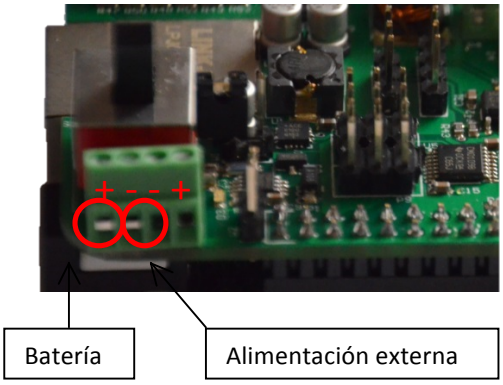


Figura 4: Conector alimentación principal.

Deberemos llevar especial cuidado a la hora de las conexiones de alimentación y su polaridad, ya que a pesar de que Robocape posee protecciones contra sobretensión y cambio de polaridad, la propia protección podrá ser destruida, y el sistema dejará de funcionar hasta que esta sea reemplazada.

Robocape incluye un sistema de carga de batería, este sistema está diseñado para baterías de 2 celdas (2C) de 7,4V[1]. En cualquier caso podremos conectar al mismo tiempo una fuente externa y la batería de 2C y seleccionar que fuente queremos que alimente a nuestro sistema mediante el selector. Si decidimos cargar nuestra batería 2C, deberemos posicionar el selector para alimentación externa, y además insertar el jumper (W1) que existe junto al selector, de este modo la carga será iniciada y esta será indicada mediante un led (D3).

2.3. La IMU

De la necesidad en cualquier tipo de robot, de obtener su posicionamiento en el espacio donde está realizando la tarea a desarrollar, nace la obligación del uso de sistemas de medición espacial tales como acelerómetros, giróscopos, etc. Robocape incorpora toda una serie de sensores que nos aportan datos e información acerca del estado y posicionamiento espacial en todas sus variables.

El primero de estos sistemas es el MPU-9150, primer seguidor de posicionamiento del mundo con procesador integrado, el cual incluye 3 acelerómetros, 3 giróscopos y 3 magnetómetros. La comunicación de este sistema con la BBB se realiza mediante un bus I2C, concretamente y siguiendo la nomenclatura BBB el bus se corresponde con el I2C_2 (pines 19 y 20 del conector P9). La dirección I2C HW ofrecidas por este dispositivo pueden ser varias según la configuración, en el caso de la Robocape, esta dirección se encuentra en la 0x69, por tener el AD0=1. Los detalles sobre sus características y modo de funcionamiento lo podemos encontrar en el manual de referencia.

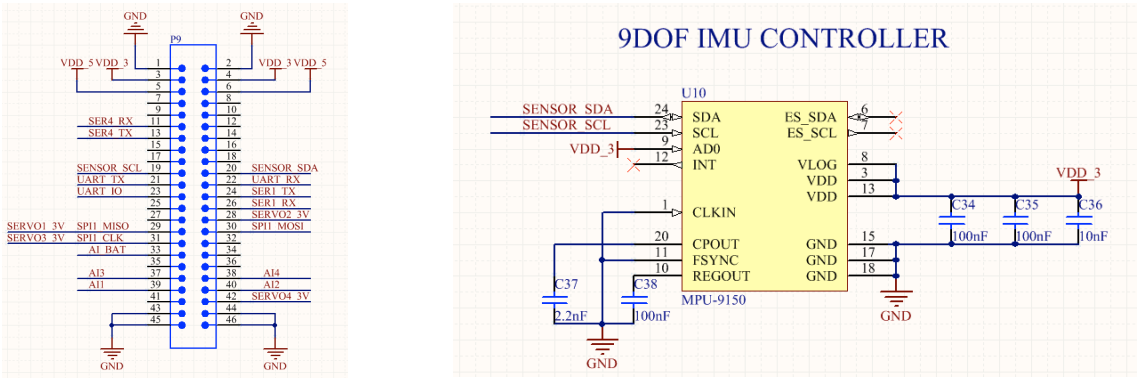


Figura 5: Descripción de los módulos y IC`s y su distribución

[1] Esto no supone la imposibilidad de conectar una batería de voltaje diferente, sino que esta no podrá ser cargada por Robocape, ya que el cargador esta especialmente diseñado para lipo-2C

3. El Bus I2C

I2C (Inter-Integrated Circuit) es un bus de comunicación muy utilizado para comunicar circuitos integrados, uno de sus usos mas comunes es la comunicación entre un microcontrolador y sensores periféricos. El I2C es un bus multi-maestro (el que inicia la comunicación) es decir permite que haya múltiples maestros y múltiples esclavos en el mismo bus (aunque en este caso se establecen mecanismos de arbitraje de acceso al bus).

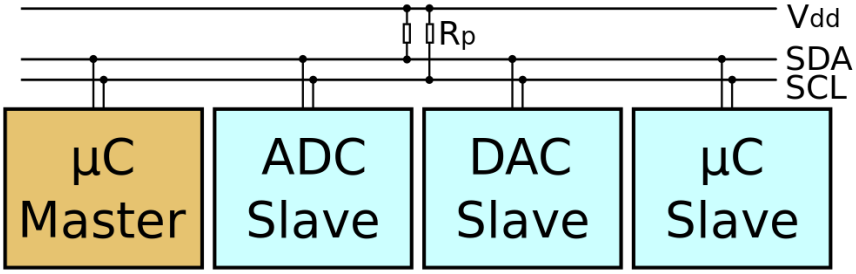


Figura 6: bus I2C

El bus I2C cuenta con dos líneas SDA(datos) y SCL(clock) además de masa. Cada flanco de SCL, marca la aparición de un bit de datos en la línea SDA. Además hay que tener en cuenta que la línea SDA solo puede cambiar de valor en caso de que la línea SCL este a 0.

En I2C hay un bit dominante (0) y otro recesivo (1), al igual que pasaba con el bus CAN. Esto es así porque el 0 se consigue forzando la línea a esa tensión pero por contra el 1 se consigue con pull-up, por lo tanto en caso de que alguien transmita un 0 y otro un 1, en la línea solo se vera reflejado el 0. Por lo tanto se define como el valor de reposo del bus como el 1, ya que si alguien quiere empezar a comunicar siempre podrá modificar el estado del bus y los demás se darán cuenta.

Otro aspecto a tener en cuenta en I2C es que los maestros o maestro son los únicos que pueden controlar la línea de SCL, eso implica que solo un maestro puede iniciar una transmisión por lo que un Slave tendrá que esperar a que un maestro le pregunte por un dato para poder enviarlo.

Así en I2C cada dispositivo tiene una dirección de 7 bits, es decir se pueden tener hasta 128 dispositivos conectados al mismo bus, hay que tener en cuenta que existen versiones extendidas de I2C con direccionamiento a 8,10 y 12 bits.

Una trama I2C esta comprendida por los siguientes campos.

1. Bit de start: Este es un bit especial ya que como hemos dicho antes la línea SDA no puede cambiar a menos que SCL este a 0. Este bit rompe dicha norma y provoca un cambio de 1 a 0 cuando SCL esta a nivel alto.
2. Address: El primer byte enviado empieza con 7 bits de dirección, el cual indica a quien enviamos o solicitamos el dato.
3. R/W (Read/Write): El siguiente bit indica si vamos a realizar una operación de lectura o escritura.
4. ACK: Este bit esta presente al final de cada byte que enviamos y nos permite asegurarnos que el byte ha llegado a su destino. De este modo el que envía deja el bit a 1 y si alguien a recibido el mensaje fuerza ese bit a 0. De esta manera confirma que le ha llegado el byte y la transmisión puede continuar.
5. 1º Byte de datos: Este es el primer byte de datos propiamente dicho ya que lo anterior no lo podemos elegir nosotros y nos viene impuesto por el protocolo. Aquí podemos poner el dato que queramos en caso

de comunicación con sensores remotos un uso habitual es poner el numero de registro al que queremos escribir o leer. Después del byte de datos se espera otro ACK del receptor.

6. Se repite el paso 5 tantas veces como sea necesario.
7. Bit de Stop. En este caso ocurre lo contrario al bit de Start, se pasa de 0 a 1 cuando la linea SCL se encuentra en alto. Esto termina la transmisión y deja el bus libre para que otro puede empezar a transmitir.

3.1. El bus I2C en la Beaglebone

La Beaglebone Black tiene tres buses I2C, de los cuales olo 2 están accesible en pines.

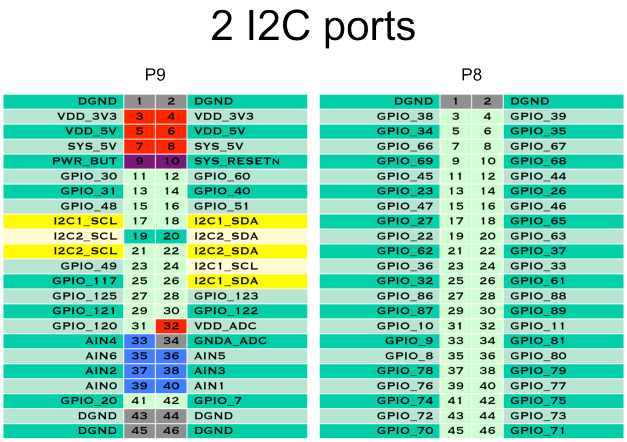


Figura 7: Pines I2C

- i2c0: No esta disponible en los conectores, uso interno para el HDMI
- i2c1: pines P9 17,18 (alternativamente 24,26)
- i2c2: pines P9 19,20 (alternativamente 21,22)

No todos está exportados por defecto, pero además hay otra complicación añadida y es que en el sistema de archivos están numerados de forma diferente. Teniendo en cuenta que sus direcciones de dispositivo son:

- i2c0: 0x44E0_B000
- i2c1: 0x4802_A000
- i2c2: 0x4819_C000

Lo mejor es ejecutar un comando ls para conocer cuales disponemos y que ficheros son los que los implementan:

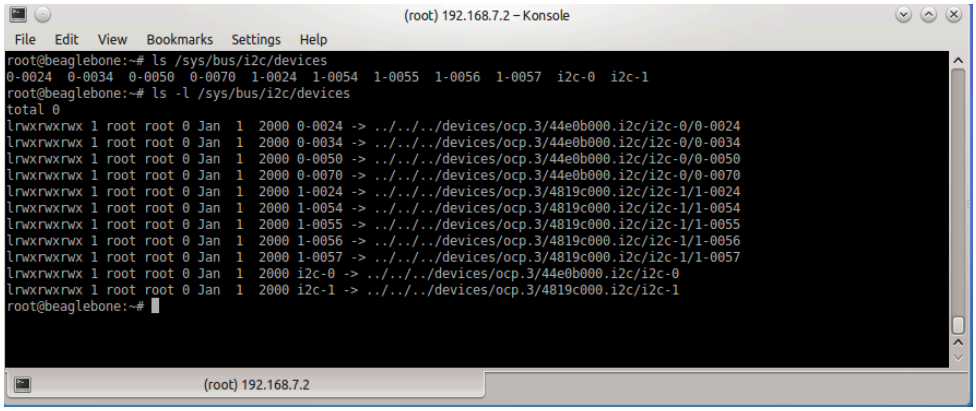
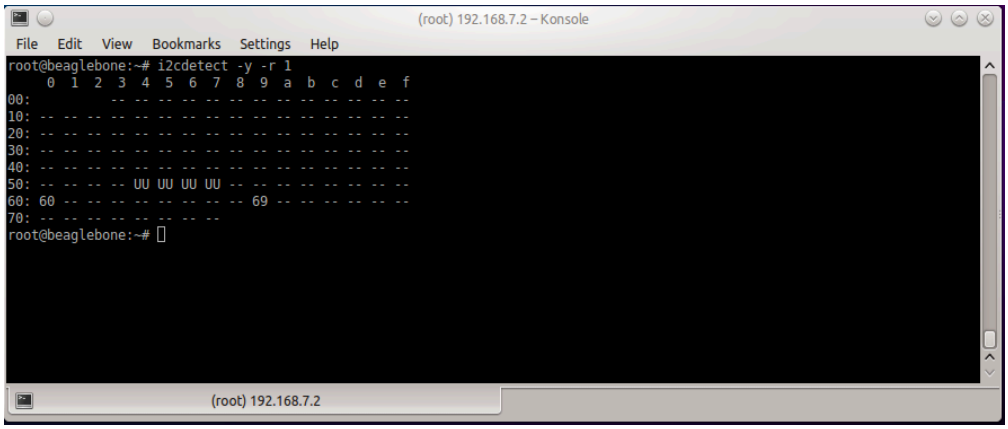


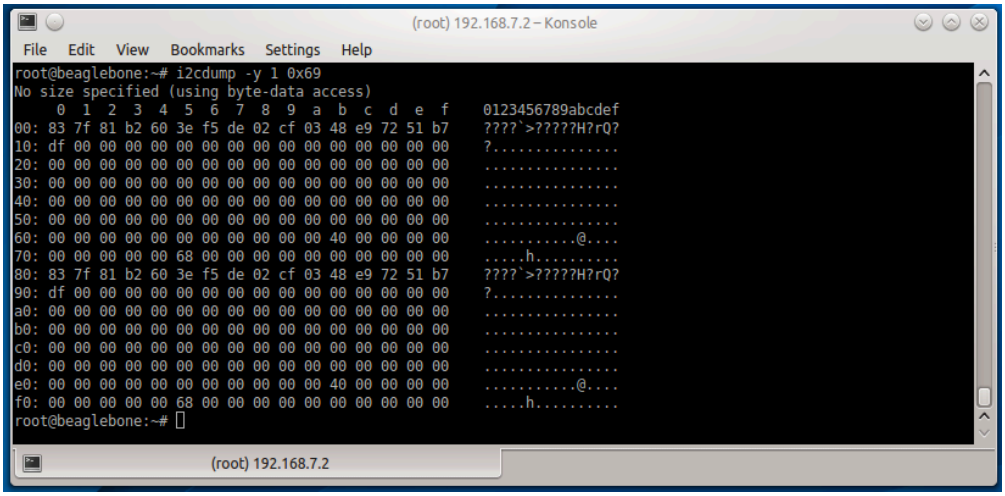
Figura 7: buses I2C presentes en el sistema, el I2C0 y el I2C2 (físico) como I2C1 en el sysfys

Una vez conocemos los buses que tenemos disponibles necesitamos conocer los dispositivos conectados a los mismos. Para ello utilizaremos el comando i2cdetect con las opciones –y (deshabilita el modo interactivo de uso) –r (usa el comando SMBus "read byte" para test) 1 (bus i2c a escanear)



Como vemos en el bus i2c1 tenemos dos dispositivos en las direcciones 60 y 69. Estos corresponden al altímetro MPL3115A2 y a la IMU MPU-9150 que es en la que tenemos interés. También vemos que existen direcciones marcadas con "UU". Esto significa que se están usando ya por un driver (ver salida anterior del comando ls de /sys/bus/i2c/devices), y por tanto se saltan del chequeo. Las marcadas con "--" significa que no tienen dispositivo asociado.

Además de conocer los buses disponibles y los dispositivos de cada uno de ellos, el sistema operativo nos permite con la orden i2cdump averiguar el contenido de los registros accesible de un dispositivo i2c. En nuestro caso podríamos hacer



Lo cual a todas luces sin conocer el significado de los registros (vía lectura del manual), es poco útil, máxime si los datos ocupan varios octetos y por tanto el dato requiere ser compuesto para su uso. Podemo fijarnos por ser común a todos los dispositivos en el primer registro, el cual contiene el valor 0x83, el cual es el identificador del dispositivo.

De igual forma también existe un comando i2cset, el cual nos permite escribir un valor en un dispositivo i2c, que debe ser utilizado con conocimiento de donde escribir, y los valores adecuados.

3.2. Acceso al bus I2C desde código

Para acceder a un dispositivo i2c en la Beaglebone Black usaremos un conjunto de rutinas definida en la librería Blacklib (http://blacklib.yigityuce.com/index.html), las cuales nos simplifican el manejo de los ficheros asociados a los buses en el sysfys, así como trasiego de datos en lectura y escritura.

Simplemente para tener un concepto aproximado de lo que implica el acceso sin este tipo de librería, el acceso tendrá que pasar por las siguientes etapas, comentadas en el siguiente código ejemplo.

Ejercicio 1: implementar este código en un proyecto C sobre la BBB

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/ioctl.h>
#include<linux/i2c.h>
#include<linux/i2c-dev.h>
#define DEVID 0x00
#define BUFFER_SIZE 40
int main(){
    int file;
    printf("Prueba de acceso a la IMU\n");
//Primero abrir el fichero
    if((file=open("/dev/i2c-1", O_RDWR)) < 0){
        perror("Error al abrir el bus\n");
        return 1;
    }
// configurar en dicho archivo un dispositivo en modo esclavo en la dirección 0x69
    if(ioctl(file, I2C_SLAVE, 0x69) < 0){
        perror("Error al conectar al sensor\n");
        return 1;
    }
    char writeBuffer[1] = {0x00};
// escribir en la dirección 0x00
    if(write(file, writeBuffer, 1)!=1){
        perror("Error al escribir en el sensor\n");
        return 1;
    }
    char readBuffer[BUFFER_SIZE];
// leer lo que nos devuelve
    if(read(file, readBuffer, BUFFER_SIZE)!=BUFFER_SIZE){
        perror("Error al leer en el sensor\n");
        return 1;
    }
// si todo va bien debería ser 0x69
    printf("El identificador del dispositivo es: 0x%02x\n", readBuffer[DEVID]);
    close(file);
    return 0;
}
```

En nuestro caso y dada la complejidad de la IMU se proporciona un proyecto con las definiciones necesarias, así como el programa principal listo para obtener los valores del acelerómetro de la IMU. Hay que recalcar que dichos deberían proporcionar en principio un valor de aceleración en Z cercano a 1 (por la gravedad). Podréis comprobar que esto es poco probable debido a la disposición del integrado de la IMU en la placa, la posición de la placa en el momento de la prueba, y posible derivas del sensor. Pero si debería comprobarse (y no está incluido en el código) que el módulo del vector aceleración con sus tres componentes tiene valor 1 (o cerca de 1).

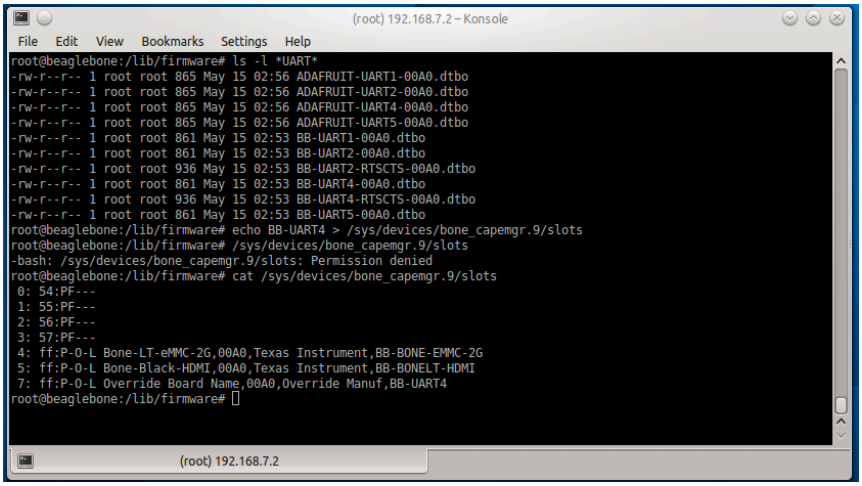
4. La UART

La universal asynchronous receiver/transmitter (UART) de la BBB, es un periférico bastante común en los sistemas empotrados, que históricamente ha sido usado para conectar sensores y dispositivos en una comunicación punto a punto. De este modo una UART no es estrictamente un bus de comunicaciones ya que solo conecta a dos nodos. Además al no compartir ambos nodos una señal de reloj, estamos ante el caso de comunicación síncrona, lo cual restringe su uso.

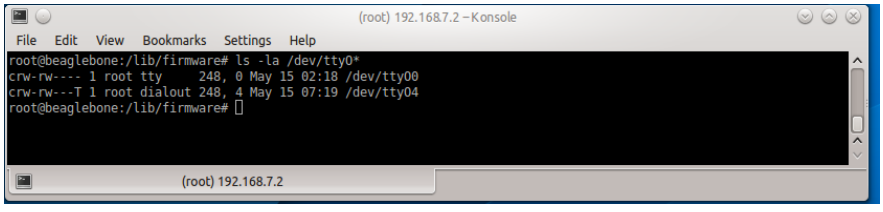
Sobre una UART se pueden desplegar diferentes interfaces de comunicación en función de la señalización que se utilice. RS-232, RS-485 son alguno de los tipos disponibles. No ahondaremos más profundamente en estos aspectos ya que han sido tratados en la asignatura de Computadores y Redes, centrándonos en a capacidad de comunicación en la BBB.

4.1. La RoboCAPE y las UART

La RoboCAPE no dispone de pines accesibles para la comunicación con las UART de la placa. En parte porque se han usado para otros canales de comunicación (UART2 y UART5), y el resto porque no están accesibles (UART1 y UART4). La forma más sencilla de poder conectar la UART con la placa RoboCAPE es por medio de un par de hilos sujetos en el conector P9 en los pines 11 y 13. De esta forma el pin 11 correspondiente a la señal UART4_RXD, lo conectaremos al cable TXD del convertidor USB-Serie disponible en el kit del laboratorio. De forma similar el pin 13 correspondiente a UART4_TXD se conecta al cable RXD del cable. El cable correspondiente a GND, puede conectarse al pin 1 de conector J% o J& de la RoboCAPE, ya que ambas son señales de GND. De esta forma ya tendremos disponible el cableado necesario para la señalización. A continuación habrá que hailitar el overlay correspondiente a la UART4. Esto como ya sabemos se puede hacer localizando dicho overlay en el directorio /lib/firmware. En la siguiente imagen podemos ver la secuencia de comandos



De esta forma ya tenemos disponible en el sysfys los ficheros correspondientes al canal de comunicación por la UART4.

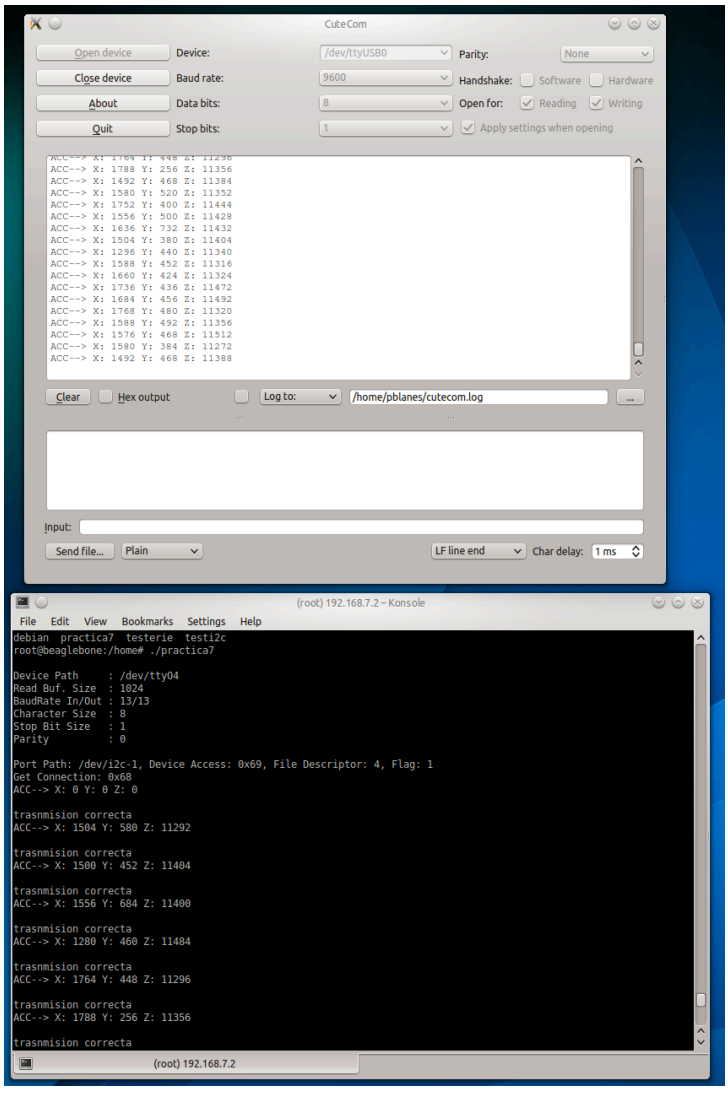


4.2. Comunicaciones UART desde código

Al igual que sucede con la comunicación I2C, aunque existe una forma directa de acceder a los puertos UART por medio de la apertura de ficheros, y la configuración de los mismos vía llamas al sistema Linux, existen librerías que simplifican dicho proceso y abstraen gran parte de esos detalles. En nuestro caso la librería Blacklib ofrece un objeto BlackUART, el cual nos permite tanto la apertura del puerto, como el acceso al mismo en modo lectura y escritura de caracteres y de strings. En el ejemplo proporcionado se puede ver como abrir la UART4 para escribir en ella un string.

5. Ejercicio: pasarela de datos IMU

Se propone como ejercicio de la practica implementar un programa que recoja los valores de aceleración en los ejes X,Y y Z de la IMU , y los envíe por el puerto serie al host Linux de la MV. Al otro extremo de la comunicación abriremos el puerto con alguna de las utilidades disponibles (minicom, cutecom, o gtkterm), de forma que podamos visualizar los mensajes enviados desde la BBB, con los datos de los acelerómetros. En la figura podemos ver un caso de ejecución



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Sistemas Empotrados

Master de Automática e Informática Industrial

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Guías BeagleBone Black (VII)

Proyecto "Simon"

Rev. José Simó. Enero 2016.

Contenido

1. Objetivos

En este documento se propone la realización de un proyecto de sistema empotrado con la BBB. El proyecto consiste en una máquina simple del juego “Simon”. El objetivo de este proyecto es combinar y ejercitar los conocimientos que ha adquirido sobre programación concurrente y E/S digital y analógica.

2. Introducción

Ralph Baer (Rodalben, Alemania, 1922 - New Hampshire, EEUU, 2014) está considerado el padre de los videojuegos porque fue el primero en convertir la televisión en un centro de entretenimiento interactivo. De origen judío, a los 12 años, escapó junto a su familia de la Alemania nazi a Estados Unidos en el otoño de 1938 poco antes de la famosa “noche de los cristales rotos”. Experto en electrónica y audiovisuales inventó en 1969 la “caja marrón”, el juego de tenis electrónico que lograba convertir a cualquier televisor en lo que hoy conocemos como una video-consola de juegos.

Figura 1: Ralph Baer y Bill Harrison mostrando la “caja marrón”.

En 2006 El presidente de los EEUU George Bush le entrega la “Medalla Nacional de Tecnología” y en 2014, poco antes de su muerte, recibió la “Medalla Edison” de la IEEE. Donó todos sus prototipos y documentos a museo “Smithsonian”.

Cambió el mundo con sus inventos, todos ellos marcados por un objetivo común: el desarrollo de la mente humana mediante la tecnología y los juegos.

Entre sus creaciones destaca “Simon” (1978), el famoso juego de 4 pulsadores de colores en el que el jugador tiene que repetir la secuencia que la máquina le propone.

Figura 2: Ralph Baer con “Simon”.

3. Descripción del trabajo

Se propone construir una máquina "Simon" simplificada. Para ello deberá programar la lógica del juego en un sistema empotrado basado en la BBB conectado a las luces, pulsadores y demás periféricos que necesite.

3.1. Hardware necesario

Recopile los componentes que se listan a continuación y móntelos sobre una chapa de plástico. Monte la electrónica necesaria para encender los LEDs en pequeñas placas de prototipo o "al aire" asegurándose de que los contactos son firmes, ordenando bien los cables e identificando cada terminal que tenga que conectar a la BBB. Organice bien todo el conjunto adhiriéndolo a la chapa de plástico.

- 1 Pulsador de "Inicio" con su correspondiente resistencia para conectarlo a la entrada GPIO.
- 4 Pulsadores de juego, con sus correspondientes resistencias.
- 4 LEDs de colores diferentes, transistores y resistencias para conectar los LEDs a salidas GPIO.
- 2 diales para configurar la dificultad del juego.

Cada dial consiste en:

- 1 potenciómetro sobre el que actúa el jugador
- 1 servomotor que mueve una manecilla que indica el nivel seleccionado

3.2. Lógica de juego

Para comenzar a jugar, el jugador debe pulsar el botón "Inicio" y el juego se inicia en el nivel "1". En el nivel "n" el juego ilumina, en una secuencia aleatoria de longitud "n", las luces de colores. Cuando el juego termina de mostrar la secuencia, el jugador debe introducir, pulsando los botones correspondientes, la secuencia mostrada en el orden correcto. Si acierta, el juego pasa a nivel n+1 y si falla, ha perdido y el juego se detiene.

Hay dos parámetros que indican el nivel de dificultad:

- La velocidad con la que se muestra la secuencia patrón
- El tiempo que tiene el usuario para introducir cada color de la secuencia patrón.

Estos dos parámetros se configurarán mediante diales.

Los estados básicos por los que pasa el juego aparecen resumidos en la Figura 3.

3.3. Actividades

Parte básica: Identifique las tareas a ejecutar en cada estado y las causas que hacen que el sistema cambie de estado. Programe el sistema completo descrito utilizando "POSIX threads" y monitores para la gestión de los estados. No olvide configurar los parámetros de dificultad incorporando los dos diales.

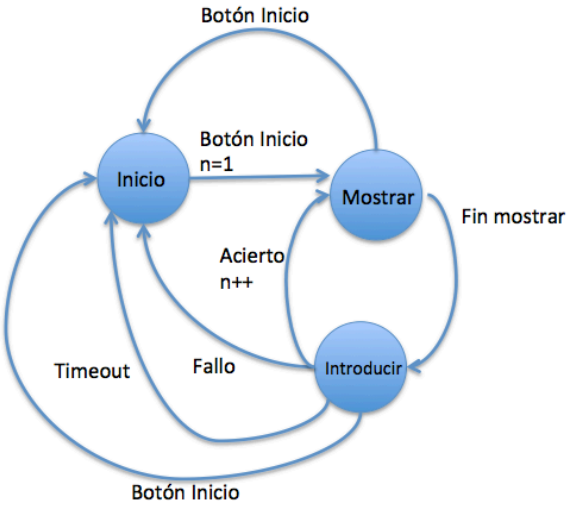


Figura 3: Posibles estados y transiciones del juego

Ampliación 1: Introduzca un nuevo botón "Pausa" para que el usuario pueda parar el juego en cualquier momento y reanudarlo con el mismo botón. Modifique el grafo de estados para incluir esta funcionalidad.

Ampliación 2: Introduzca sonido (zumbido de diferentes frecuencias) con cada pulsación o iluminación de LED. Introduzca también un tono de "inicio", uno de "acierto" y otro para el "fallo". Para generar los sonidos necesitará un zumbador conectado a una salida GPIO. Tendrá que encender y apagar el zumbador con diferente tiempos de espera para conseguir diferentes tonos.

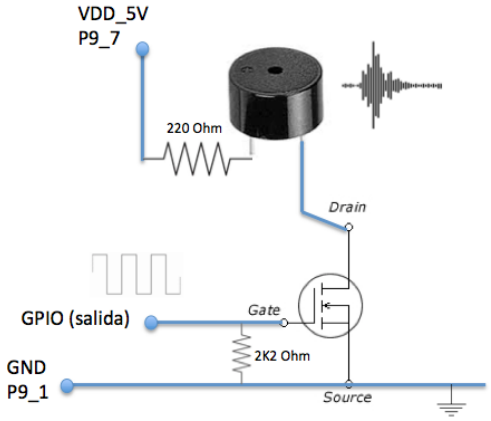


Figura 4: Zumbador

4. Conclusiones

Con este proyecto ha puesto a prueba sus conocimientos de programación concurrente y acceso a la Entrada/Salida, tanto analógica como digital en la BBB. Ahora ponga a prueba su memoria jugando un rato con Simon!

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Sistemas Empotrados

Master de Automática e Informática Industrial

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Guías BeagleBone Black (IX)

Comunicación en Red. Sockets.

Rev. José Simó. Enero 2016.

Contenido

1. Objetivos

En este documento se revisa la comunicación entre procesos a través de Internet utilizando el modelo ampliamente difundido de "Sockets BSD" i el API incluida en el estándar POSIX.

2. Introducción

El API (*Applications Programmer Interface*) "Berkeley Sockets 4.4" es un conjunto estándar de funciones disponibles en el nivel de aplicación (ver modelo OSI) que permiten comunicar procesos a través de Internet.

La primera implementación data del año 1983 y fue incluida en el código de LinuxBSD4.2. Se introdujeron mejoras hasta que en 1989, junto con LinuxBSD4.4 se distribuyó el código libre del API tal y como lo conocemos hoy en día. La permisividad de la licencia BSD hace que este código lo podamos encontrar prácticamente en todos los sistemas operativos, tanto libres como propietarios, especialmente en aquellos basados en Unix, ya que el API SocketsBSD ha evolucionado hasta convertirse en parte del estándar POSIX (POSIX 1-2008 26/07/2012).

El API está escrita originalmente en lenguaje C aunque existen bibliotecas que ofrecen envolventes para poder acceder al API desde prácticamente cualquier lenguaje.

3. Modelo de comunicación

El modelo de comunicaciones con sockets es Cliente/Servidor. La comunicación se inicia de forma asimétrica, uno de los procesos adopta el rol de Servidor y acepta las peticiones de conexión de cualquier proceso Cliente. El proceso que adopta el rol de Cliente toma la iniciativa de conectarse con el proceso servidor y, para ello, necesita conocer la dirección de Internet del computador donde se ejecuta el proceso Servidor y el puerto por el que acepta conexiones. Una vez iniciada la comunicación, queda establecido un canal de comunicación bidireccional por el que tanto el cliente como el servidor pueden enviar o recibir datos.

La principal abstracción del modelo es el "socket" o "punto final de comunicaciones" que puede ser de diferentes tipos y soportar diferentes protocolos de comunicación.

3.1. Funciones estándar

El API estándar de sockets BSD consta de las siguientes funciones:

- **`socket()`** : Crea un nuevo socket y retorna un descriptor de fichero (-1 si error).

- Prototipo: `int socket(int domain, int type, int protocol);`
- Toma tres parámetros:
  - **domain**: especifica la familia de protocolos
    - AF_INET protocolo IPv4
    - AF_INET6 protocolo IPv6.
    - AF_UNIX para socket local (IPC).
  - **type**:
    - SOCK_STREAM (Stream Sockets)
    - SOCK_DGRAM (Datagram Sockets)
    - SOCK_SEQPACKET (servicio de paquetes secuenciados)
    - SOCK_RAW (protocolo crudo de la capa de red).
  - **protocol**: especifica el protocolo de transporte. IPPROTO_TCP, IPPROTO_SCTP, IPPROTO_UDP, IPPROTO_DCCP. (Ver "netinet/in.h"). El valor 0 indica que se use el protocolo por defecto del dominio y tipo especificado.

- **`bind()`** : Vincula un socket a una dirección. Retorna 0 si éxito y -1 si error.
  - Prototipo: `int bind(int sockfd, const struct sockaddr *my_addr, socklen_t addrlen);`
  - Toma tres parámetros
    - **sockfd**: es el descriptor del socket que se quiere manipular.
    - **my_addr**: un puntero a "struct sockaddr" que representa la dirección a vincular.
    - **addrlen**: una variable del tipo "socklen_t" que especifica la longitud del "struct sockaddr".

- **`listen()`** : Prepara el socket para recibir nuevas peticiones de conexión. Se usa sólo en los modelos de intercambio de datos orientados a conexión (SOCK_STREAM y SOCK_SEQPACKET).
  - Prototipo: `int listen(int sockfd, int backlog);`
  - Toma dos parámetros
    - **sockfd**: es el descriptor del socket que se quiere manipular.
    - **backlog**: es un entero que representa el número máximo de peticiones que se pueden encolar.

- **`accept()`** : Acepta una nueva conexión (la primera que encuentre en la cola) la inicializa y crea un nuevo socket a través del cual se realizará la comunicación de datos. Se usa después de `listen()`. Retorna un descriptor de fichero que representa el nuevo socket creado (-1 si error).
  - Prototipo: `int accept(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);`
  - Toma tres parámetros
    - **sockfd**: es el descriptor del socket que se quiere manipular.
    - **cliaddr**: un puntero a "struct sockaddr" que representa la dirección del cliente que ha solicitado la conexión.
    - **addrlen**: una variable del tipo "socklen_t" que especifica la longitud del "struct sockaddr" del cliente que se ha obtenido en el parámetro anterior.

- **`connect()`** : Utiliza el socket para establecer un canal de comunicación con un host remoto identificado por su dirección. Si se está usando un protocolo basad en conexión, establece una nueva conexión. Si se usa con un protocolo sin conexión (datagramas) define la dirección remota

donse enviar o recibir paquetes de datos mediante las funciones send() y recv(). Retorna 0 si éxito y -1 si error.

- Prototipo: `int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen)`
- Toma tres parámetros
  - **sockfd**: es el descriptor del socket que se quiere manipular.
  - **serv_addr**: un puntero a "struct sockaddr" que representa la dirección del servidor con el que se quiere conectar.
  - **addrlen**: una variable del tipo "socklen_t" que especifica la longitud del "struct sockaddr" del servidor.
  -

3.2. Uso de las funciones

La secuencia de uso de estas funciones es la que se puede observar en la Figura 1.

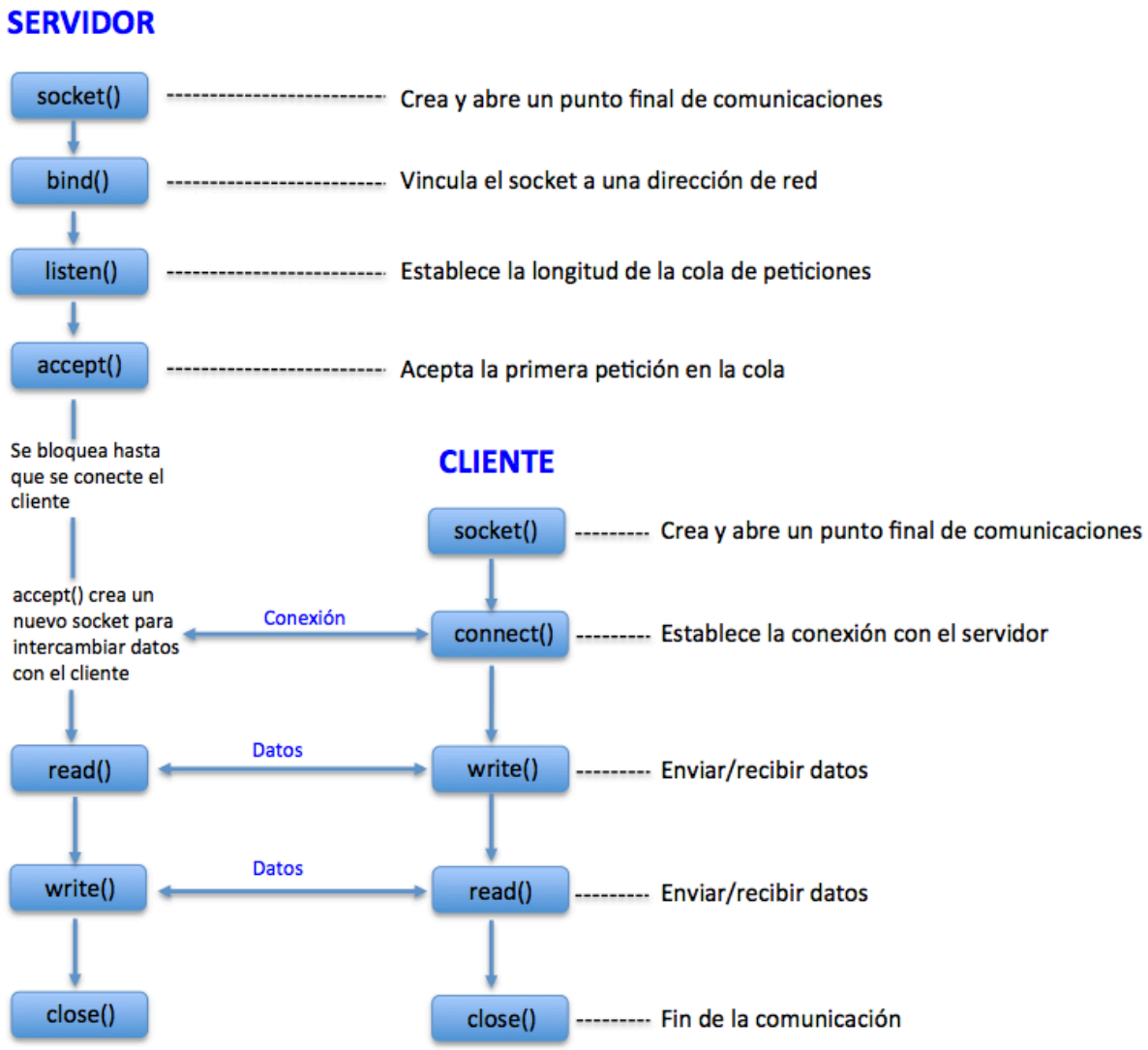


Figura 1: Interacción cliente/servidor.

3.3. Funciones auxiliares

Otras funciones útiles para el manejo de comunicaciones con sockets son:

- **`gethostbyname()`** y **`gethostbyaddr()`:** Se utilizan para resolver el nombre de host y dirección utilizando el mecanismo DNS del sistema. Retorna un puntero a "struct hostent" que describe el host. Estas funciones son auxiliares y no forman parte del estándar y se consideran anticuadas porque han sido sustituidas por "getaddrinfo()" y "getnameinfo()" junto con "struct addrinfo".

- **`getsockopt()`** y **`setsockopt()`** se utilizan para manipular las opciones de un socket. Para ver las opciones disponibles, visite la página de manual "socket(7)" de Linux. Sus prototipos son:

```
int getsockopt(int sockfd, int level, int optname, void *optval, socklen_t *optlen);
int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen);
```

Por ejemplo, para desactivar el algoritmo "Nagle" de retención de pequeños paquetes, se usaría así:

```
int result = setsockopt(sock,            /* socket affected */
                        IPPROTO_TCP,     /* set option at TCP level */
                        TCP_NODELAY,     /* name of option (disable Nagle)*/
                        (char *) &flag,  /* the cast is historical cruft */
                        sizeof(int));    /* length of option value */
```

- **`htonl(), ntohl(), htons(),ntohs()`:** Cuando utilizamos números enteros para especificar propiedades de red (típicamente pata especificar puertos) encontramos que la representación binaria en el host puede ser diferente que la representación en red (cambia el "endianness"). Para cambiar un "unsigned short int" ("s") de la representación del host ("h") a la de red ("n"), usaríamos "htons()". Lo mismo para un "unsigned integer" (hostlong) htonl().

3.4. Estructuras de datos

Un "`struct sockaddr_in`" es una estructura que contiene una dirección de internet. Está definida en el archivo "netinet/in.h".

```
struct sockaddr_in
{
  short   sin_family; /* must be AF_INET */
  u_short sin_port;
  struct  in_addr sin_addr;
  char    sin_zero[8]; /* Not used, must be zero */
};
```

"`sin_family`" indica qué familia de protocolo debe usar. "`sin_port`" indica el número de puerto asociado con el socket.

El "`struct in_addr`" contiene sólo un campo del tipo "`unsigned long`" llamado "`s_addr`" y que contiene la dirección IP que se asocia con el socket.

```
struct in_addr
{
    u_long s_addr;
};
```

El "`struct sockaddr_in`" es una modificación del estándar "`struct sockaddr`":

```
struct sockaddr
{
    u_char sa_len;
    u_char sa_family;
    char   sa_data[14];
};
```

Como las llamadas a socket esperan un "`struct sockaddr`", para comunicaciones IPv4, es apropiado pasar estructuras "`struct sockaddr_in`" mediante un cast a "`struct sockaddr`".

4. Ejercicios

Observe el uso de las funciones descritas en la sección anterior que se hace en el código de ejemplo del cliente y del servidor que se incluye en el anexo de este documento.

Ejercicio 1: Modifique el código que se incluye en el anexo ("socketServer.c " y "socketClient.c") para que el servidor pueda aceptar múltiples conexiones y atenderlas concurrentemente. Para ello deberá crear un hilo de ejecución por cada conexión aceptada y pasar en la creación del hilo una referencia al socket creado en la función "accept" (socket por el que se intercambian los datos).

Ejercicio 2: Como punto de partida el código realizado en el Ejercicio 1, modifique el código para que cumpla con el siguiente comportamiento: El programa servidor "socketServer02.c ", que se ejecutará en la BBB, enviará al programa cliente, que se ejecutará en otro computador de la red (p.e. el computador anfitrión) un mensaje cuyo formato incluirá al menos el valor de dos entradas GPIO y el valor de dos entradas analógicas. El servidor sólo enviará valores cuando estos difieran del último valor enviado y lo hará con una separación temporal nunca inferior a un valor preestablecido.

5. Conclusiones

Con la ayuda de este documento ha revisado las bases del manejo de las comunicaciones a través de Internet utilizando el estándar POSIX disponible en la BBB. A partir de ahora podrá incluir esta funcionalidad en sus proyectos con la BBB y explorar el desarrollo de sistemas empotrados distribuidos.

6. Anexo I. Código

6.1. Archivo “socketServer.c”

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

void error(const char *msg) {perror(msg); exit(0);}

int main(int argc, char *argv[])
{
     int serverSockfd, comSockfd;
     int portno;
     char buffer[256];
     int n, retval;
     struct sockaddr_in serv_addr, cli_addr;
     socklen_t clilen = sizeof(cli_addr);

     //Retrieve the listening port from the command line or use a default one.
     if (argc < 2) {
         fprintf(stdout,"Using default port (2000) to accept connections.\n");
         portno = 2000;
     } else {
       portno = atoi(argv[1]);
     }
     //Create a socket (Internet domain ipv4, stream-oriented, TCP protocol)
     serverSockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
     if (serverSockfd < 0) error("ERROR opening socket");
     //Prepare the local address (server address) structure
     bzero((char *) &serv_addr, sizeof(serv_addr));
     serv_addr.sin_family = AF_INET;
     serv_addr.sin_addr.s_addr = INADDR_ANY;
     serv_addr.sin_port = htons(portno);
     //Associate the socket to the local address
     retval = bind(serverSockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr));
     if (retval < 0) error("ERROR on binding");
     //Prepare the socket to accept incoming connections (max queue size 5)
     listen(serverSockfd,5);
     //Accept one connection. When accepted, create a new socket to transfer data.
     comSockfd = accept(serverSockfd, (struct sockaddr *) &cli_addr, &clilen);
     if (comSockfd < 0) error("ERROR on accept");
     //Connection established!
     //Now we can transfer data from/to the client
     bzero(buffer,256);
     n = read(comSockfd,buffer,255);
     if (n < 0) error("ERROR reading from socket");
     printf("Read message: %s\n",buffer);
     n = write(comSockfd,"Message ACK",11);
     if (n < 0) error("ERROR writing to socket");
     //Close communication
     close(serverSockfd);
     close(comSockfd);
     return 0;
}
```

6.2. Archivo “socketClient.c”

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void error(const char *msg) {perror(msg); exit(0);}
```

```c
int main(int argc, char *argv[])
{
    int sockfd;
    int portno;
    struct sockaddr_in serv_addr;
    struct hostent *server;
    char buffer[256];
    int n, retval;

    //Retrieve the server host and port from the command line or use default values.
    if (argc >= 3) {
        server = gethostbyname(argv[1]);
               portno = atoi(argv[2]);
    }
    if (argc == 2) {
        server = gethostbyname(argv[1]);
               portno = 2000;
               fprintf(stdout,"Usage %s hostname port\n", argv[0]);
               fprintf(stdout,"Using default port (2000)\n");
    }
    if (argc <= 1) {
        server = gethostbyname("localhost");
               portno = 2000;
               fprintf(stdout,"Usage %s hostname port\n", argv[0]);
               fprintf(stdout,"Using default host (localhost)\n");
               fprintf(stdout,"Using default port (2000)\n");
    }
    //Check for a valid server host value.
    if (server == NULL) error("ERROR, no such host");
    //Create a socket (Internet domain ipv4, stream-oriented, TCP protocol)
    sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sockfd < 0) error("ERROR opening socket");
    //Prepare the server address structure
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr, (char *)&serv_addr.sin_addr.s_addr, server->h_length);
    serv_addr.sin_port = htons(portno);
    //Connect to the server
    retval = connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr));
    if (retval < 0) error("ERROR connecting");
    //Connection established!
    //Now we can transfer data from/to the server
    printf("Please enter the message: ");
    bzero(buffer,256);
    fgets(buffer,255,stdin);
    n = write(sockfd,buffer,strlen(buffer));
    if (n < 0) error("ERROR writing to socket");
    bzero(buffer,256);
    n = read(sockfd,buffer,255);
    if (n < 0) error("ERROR reading from socket");
    printf("%s\n",buffer);
    //Close communication
    close(sockfd);
    return 0;
}
```