



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN

**ANÁLISIS PRUEBA Y UTILIZACIÓN DE UNA TARJETA DE
DESARROLLO CON PROCESADOR ARM CÓRTEX A8**

TESIS

**QUE PARA OBTENER EL TÍTULO DE:
INGENIERO MECÁNICO ELECTRICISTA**

PRESENTA:

ALBERTO DANIEL PICASSO GONZÁLEZ

ASESOR:

ING. JOSÉ LUIS BARBOSA PACHECO

Cuautitlán Izcalli, Estado de México, 2016



Universidad Nacional
Autónoma de México



UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.



UNIVERSIDAD NACIONAL
AUTÓNOMA DE
MÉXICO

**FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN
UNIDAD DE ADMINISTRACIÓN ESCOLAR
DEPARTAMENTO DE EXÁMENES PROFESIONALES**

U. N. A. M.
FACULTAD DE ESTUDIOS
SUPERIORES CUAUTITLÁN
ASUNTO: **VOTO APROBATORIO**

**M. en C. JORGE ALFREDO CUÉLLAR ORDAZ
DIRECTOR DE LA FES CUAUTITLÁN
PRESENTE**

ATN: M. en A. ISMAEL HERNÁNDEZ MAURICIO
Jefe del Departamento de Exámenes
Profesionales Cuautitlán.
DEPARTAMENTO DE EXÁMENES PROFESIONALES



Con base en el Reglamento General de Exámenes, y la Dirección de la Facultad, nos permitimos comunicar a usted que revisamos **La Tesis:**

**"ANÁLISIS PRUEBA Y UTILIZACIÓN DE UNA TARJETA DE DESARROLLO CON PROCESADOR ARM
CÓRTEX A8"**

Que presenta el pasante: **ALBERTO DANIEL PICASSO GONZÁLEZ**
Con número de cuenta: **30813556-1** para obtener el Título de: **Ingeniero Mecánico Electricista**

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el **EXAMEN PROFESIONAL** correspondiente, otorgamos nuestro **VOTO APROBATORIO**.

ATENTAMENTE
"POR MI RAZA HABLARA EL ESPÍRITU"
Cuautitlán Izcalli, Méx. a 15 de marzo de 2016.

PROFESORES QUE INTEGRAN EL JURADO

	NOMBRE	FIRMA
PRESIDENTE	M. en T.I. Jorge Buendía Gómez	
VOCAL	Ing. José Luis Barbosa Pacheco	
SECRETARIO	Ing. Jorge Ramírez Rodríguez	
1er SUPLENTE	Ing. Oscar Carmona Islas	
2do SUPLENTE	Dr. David Tinoco Varela	

NOTA: Los sinodales suplentes están obligados a presentarse el día y hora del Examen Profesional (art. 127).
En caso de que algún miembro del jurado no pueda asistir al examen profesional deberá dar aviso por anticipado al departamento.
(Art 127 REP)

HHA/Vc

ÍNDICE.

INTRODUCCIÓN.....	1
OBJETIVO.....	4
Capítulo 1. Marco teórico.....	5
1.1 Características de los procesadores ARM.....	5
1.2 Sitara AM3358 ARM Cortex-A8.....	9
1.3 Tarjetas de desarrollo ARM Cortex con sistema operativo embebido.....	12
1.4 Ventajas de utilizar la tarjeta de desarrollo Beaglebone.....	15
1.5 Características del Beaglebone.....	17
Capítulo 2. Software y hardware en el Beaglebone.....	18
2.1 Núcleo de la tarjeta.....	18
2.2 Acceso a los puertos de entrada y salida (I/O).....	21
2.3 Accesorios externos.....	23
2.4 Sistema operativo y comunicación con la tarjeta.....	27
2.5 Repositorios del sistema operativo.....	32
2.6 GITHUB.....	33
Capítulo 3. Comunicación entre el Beaglebone y diversos dispositivos.....	34
3.1 Conexión a través de una red LAN.....	34
3.1 Escritorio remoto.....	37
3.3 Conexión SSH mediante un dispositivo Android.....	44
3.4 Conexión SSH mediante un dispositivo IOS.....	49
Capítulo 4. Compilación de programas con Eclipse.....	51
4.1 Instalación de ECLIPSE.....	51
4.2 Sistema de Exploración Remoto.....	60
4.3 Programación de los GPIO.....	68
4.4 Usando BlackLib.....	73
4.5 Programación de diferentes Buses con BlackLib.....	77
Capítulo 5. Procesamiento de Imágenes.....	108
5.1 Hardware compatible para la captura de imágenes.....	108
5.2 Implementación de una aplicación para capturar video e imágenes.....	109
5.3 Captura de imágenes empleando V4L2.....	112

5.4 Procesamiento de imágenes empleando Open CV.....	113
5.5 Detección de rostros empleando OpenCV.	116
Capítulo 6. Interface para aplicaciones básicas de tiempo real.....	120
6.1 PRU-ICSS.....	120
6.2 Introducción al uso de los PRU-ICSS.	122
6.3 GPIO de tiempo real.	124
6.4 Implementación con un sensor Ultrasónico.	131
Conclusiones	137
BIBLIOGRAFÍA.....	139
ANEXOS.....	141
Apéndices.....	155

INTRODUCCIÓN.

Actualmente existe una gran variedad de tarjetas de desarrollo con las que podemos implementar infinidad de proyectos, algunas de estas tarjetas están basadas en el uso de microcontroladores, en cambio otras constan de una computadora en una sola placa, estos últimos, a diferencia de los microcontroladores, suelen tener sus elementos separados en distintos circuitos integrados y generalmente son más potentes que un microcontrolador. A dichas placas se les conoce como computadoras de placa reducida o SBC por sus siglas en inglés (Single Board Computer). Las tarjetas de desarrollo compuestas por microcontroladores suelen basarse en hardware libre y generalmente cuentan con un entorno de desarrollo diseñado para facilitar el uso de la electrónica en proyectos multidisciplinarios. En cambio los SBC son pequeñas computadoras con un sistema operativo incluido (este sistema generalmente suele ser alguna distribución de Linux), y que a diferencia de las tarjetas basadas en microcontroladores, estas no cuentan con un entorno de desarrollo específico ya que por el hecho de ser pequeñas computadoras son capaces de ejecutar una gran variedad de compiladores. Para programar un SBC se pueden utilizar los lenguajes de programación más comunes actualmente como Java, JavaScript, C, C++, Python, entre otros.

Los sistemas operativos basados en Linux son muy flexibles a la hora de tener acceso al control de hardware, ya que la mayoría de sus distribuciones son de carácter libre, lo que significa que los usuarios tenemos la libertad de ejecutar el software para cualquier propósito e incluso podemos modificarlo (tener acceso al código fuente o a comandos que controlan el sistema) para que funcione del modo en que nosotros los usuarios deseemos. Otra característica destacable de la mayoría de los sistemas operativos basados en Linux es que estos sistemas fueron creados bajo la filosofía cliente-servidor, lo que significa que si hay dispositivos remotos interconectados a un solo equipo con un sistema operativo Linux, el equipo con Linux (en este caso el servidor) puede ejecutar en segundo plano y paralelamente las tareas que los equipos que están interconectados (clientes), deseen. Esta característica es de gran ayuda para los SBC.

A diferencia de las placas de desarrollo que utilizan un microcontrolador, los SBC generalmente suelen tener procesadores de tipo ARM (Advanced RISC Machine), razón por la cual tienen capacidades de procesamiento similares a las de una computadora.

Los procesadores ARM son ideales para aplicaciones donde se requiera una gran capacidad de procesamiento, es por eso que este tipo de procesadores se han convertido en los dominantes de la electrónica moderna. Estos procesadores se utilizan generalmente en tabletas, teléfonos móviles, teléfonos inteligentes, reproductores de música, videocámaras, cámaras fotográficas e incluso en periféricos de las computadoras como discos duros y routers.

Hoy en día hay una gran cantidad de sistemas operativos compatibles con procesadores ARM, en la siguiente lista mencionaremos algunos de los sistemas operativos más populares con soporte para ARM son: Android, Debian, Fedora, IOS, Windows Phone, Ubuntu, Symbian, Chrome OS.

Como se verá en el desarrollo de este trabajo, en el mercado existen actualmente varios tipos de SBC con funciones y características diversas. Sin embargo, centraremos nuestro objetivo de estudio en la tarjeta Beaglebone Black Rev C porque con esta se puede desarrollar un mayor número de actividades, ya que cuenta una gran variedad de características técnicas que en los demás SBC no poseen. Además, cuenta una distribución de Linux diseñada específicamente para desarrolladores y cuya comunidad se amplía por todo el mundo.

La tarjeta de desarrollo Beaglebone Black Rev C es un ordenador de placa reducida que cuenta con un procesador ARM y un sistema operativo basado en Linux, el cual es capaz de arrancar en tan solo unos segundos gracias a las características con las que fue creada esta tarjeta. Esta tarjeta mezcla las características de rendimiento de un ordenador y la compatibilidad de hardware que tienen la mayoría de los microcontroladores más utilizados hoy en día.

Este trabajo comenzará con un marco teórico donde se mencionan algunas de las SBC más utilizadas y sus principales características. También se mencionan las características técnicas más importantes en los procesadores ARM y sus principales ventajas frente a otras arquitecturas, haciendo especial énfasis en el procesador ARM que utiliza el Beaglebone Black rev C. En este capítulo también se mencionan las principales ventajas que presenta la tarjeta Beaglebone frente a las demás tarjetas de desarrollo, además de las características más importantes para tomar en cuenta por el diseñador.

En el segundo capítulo se habla principalmente de las características de hardware y software, centrandó nuestra atención en la distribución de pines, también se hace una referencia a los accesorios complementarios que son altamente recomendados para poder trabajar de manera óptima y facilitar algunas tareas relacionadas con el hardware. En este capítulo también se habla detalladamente de una plataforma de desarrollo colaborativo llamada GitHub y de qué forma se empleará a lo largo de este trabajo.

En el tercer capítulo se mostrarán las distintas formas de comunicación entre el Beaglebone y diferentes dispositivos como computadoras y celulares, empleando diversos sistemas operativos como Android, Windows e IOS. La finalidad de este capítulo es probar la gran diversidad de comunicación con la mayoría dispositivos más populares.

El cuarto capítulo está dedicado a la programación con un entorno de programación llamado ECLIPSE, con la finalidad de implementar una herramienta de desarrollo avanzada para diseñadores, se mostrarán los pasos para poder compilar programas compatibles con arquitectura ARM desde una PC y copiarlos al Beaglebone para poder ejecutarlos, además se mostraran diversos códigos de programas en C++ que serán capaces de utilizar los puertos más utilizados en proyectos electrónicos.

En el quinto capítulo se mostrará cómo utilizar el Beaglebone con una cámara web para poder capturar fotos, video y mostrar ejemplos de procesamiento de imágenes a través de una librería llamada OpenCV. Con este capítulo se probará la gran capacidad de procesamiento que tiene esta tarjeta, mostrando las ventajas e inconvenientes que podrían presentarse al utilizar procesamiento de video e imagen.

En el último capítulo se mostrará cómo trabajar con aplicaciones de tiempo real utilizando un subsistema que contiene el procesador del Beaglebone llamado PRU-ICSS (Programmable Real-Time Unit Subsystem and Industrial Communication SubSystem). Para poder implementar estos subsistemas emplearemos un sensor de ultrasonido, el cual trabaja en tiempo real y es ideal para ejemplificar el funcionamiento de este subsistema.

OBJETIVO.

Hoy en día la gran mayoría de ingenieros dedicados al tema de sistemas embebidos que desean desarrollar un proyecto con dispositivos electrónicos en donde necesite de una unidad de procesamiento, opta por la utilización de tarjetas de desarrollo basadas en un microcontrolador como Arduino o los PIC desarrollados por la compañía Microchip, sin embargo, este tipo de dispositivos suelen ser obsoletos e incapaces de ejecutar varios programas simultáneamente (paralelismo) o cuando se desea utilizar un sistema operativo para adaptar a nuestro proyecto características propias de una computadora.

La gran mayoría de estudiantes de electrónica y carreras relacionadas a sistemas embebidos utiliza Arduino o algún PIC para sus actividades y proyectos sin saber que hay otras alternativas que son más potentes y pueden desempeñar mucho mejor las mismas tareas, incluso pueden darnos la opción de hacer más sofisticado y profesional nuestro proyecto.

Arduino fue diseñado para principiantes, simplificando al máximo su lenguaje de programación, el cual está basado en C y C++, mientras que los PIC más utilizados actualmente por estudiantes y aficionado en electrónica deberían considerarse dispositivos obsoletos debido a sus pobres características técnicas en comparación con los AVR de Atmel empleados en las tarjetas Arduino. Cabe mencionar que hay microcontroladores PIC más modernos que sí compiten con los AVR, pero éstos no son muy utilizados por estudiantes y aficionados.

Este trabajo tiene como finalidad dar a conocer las principales características de esta tarjeta y su utilización, facilitando la información y mostrarla de manera organizada a usuarios que no tienen conocimientos profundos sobre sistemas operativos basados en Linux.

Con este trabajo también se demostrará a los lectores que el uso de esta tarjeta es mucho más conveniente para ingenieros que tienen un perfil avanzado en el uso de sistemas embebidos, además se dará a conocer a las personas menos experimentadas una alternativa para enriquecer más sus conocimientos.

Otro punto importante que se abordará en este trabajo es mostrar la importancia de la implementación de programas escritos en C y C++ como lenguajes, cada vez más usados, para la programación de sistemas embebidos. Finalmente se hace especial mención de que el trabajo presentará la información de una manera organizada y completa de aquella que encontramos dispersa y con escollos en diversas fuentes escritas.

Capítulo 1. Marco teórico.

1.1 Características de los procesadores ARM.

ARM es el acrónimo de Advanced Risc Machines. La arquitectura ARM es el conjunto de instrucciones de 32 y 64 bits más ampliamente utilizado en unidades producidas, principalmente en el ámbito de dispositivos móviles. La relativa simplicidad de los procesadores ARM los hace ideales para aplicaciones de ingeniería eléctrica o de computo [1].

Los procesadores ARM son licenciados por la empresa ARM holdings, es decir, ARM holdings da permiso para que otra empresa pueda implementar sus diseños y arquitecturas. Las empresas más populares que son titulares de licencias ARM incluyen a Alcatel-Lucent, Apple Inc, Atmel, Broadcom, Intel, LG, Microsemi, Microsoft, NEC, Nintendo, Nokia, Nuvoton, Nvidia, Sony, NXP Oki, Qualcomm, Samsung, Sharp, STMicroelectronics, Texas Instruments y Yamaha.

Hoy en día los procesadores más utilizados en computadoras portátiles y de escritorio son los que cuentan con arquitectura x86, estos microprocesadores son capaces de correr instrucciones tanto de 32 como de 64 bits, es por esta razón que la mayoría de computadoras personales modernas son capaces de correr sistemas operativos con ambas arquitecturas. Estos tipos de procesadores son de tipo CISC (Complex instruction set computing) con soporte para instrucciones complejas, simultáneas y de ejecución más lenta, su principal característica es simplificar la estructura de programación lo que se traduce en un alto desempeño de velocidad, que desafortunadamente viene acompañado de un mayor consumo de energía y también de la necesidad de más espacio físico.

Las máquinas CISC presentan un amplio juego de instrucciones complejas, que tardan varios ciclos de CPU en ser completadas, habiendo muchos formatos distintos de instrucciones, cabe destacar que también es más fácil crear un compilador CISC que RISC, el código de un programa CISC también posee menos líneas que el mismo programa para una máquina RISC. En máquinas CISC hay varios métodos para direccionar la memoria y muchas instrucciones poseen la capacidad de acceder a memoria. La arquitectura dificulta el paralelismo por lo que hoy en día se han desarrollado sistemas que convierten el código CISC en muchas instrucciones simples (RISC), las cuales se conocen como microinstrucciones.

Por otro lado, los procesadores ARM son de tipo RISC (Reduced Instruction Set Computer), cuyas propiedades son; que poseen instrucciones de tamaño fijo con pocos formatos y que sólo las instrucciones de carga y almacenamiento acceden a la memoria de datos. El objetivo de diseñar máquinas con esta arquitectura es facilitar el paralelismo en la ejecución de instrucciones y permitir realizar tareas menores con procesos más cortos lo que al final conlleva una disminución de la energía empleada y a la posibilidad de conseguir un procesamiento extremadamente bueno si se utilizan procesadores con dos o más núcleos de procesamiento.

Si una máquina RISC recibe una instrucción, debido al formato, puede empezar a ejecutar ésta casi instantáneamente al poder asignar rápidamente registros, espacio, etc. Mientras que una máquina CISC debe comprobar el formato de instrucción, decodificarlo y entonces realizar la asignación de

circuitos, esto lleva tiempo y por lo tanto la instrucción se ejecutará más lentamente que en una máquina RISC.

El diseño del primer ARM comenzó en 1983 y desde esa fecha hasta el día de hoy, se han creado una gran variedad de procesadores de este tipo, cada uno con características diferentes y siendo mejorados cada vez más con el paso del tiempo. Existen varios tipos de familias de procesadores ARM, sin embargo, la mayoría son obsoletas ya que las familias más modernas son mucho más potentes. Las familias de procesadores ARM que fueron y son usadas en diversos dispositivos electrónicos incluyen ARM1, ARM2, ARM3, ARM6, ARM7, ARM7TDMI, StrongARM, ARM8, ARM9TDMI, ARM9E, ARM10E, XScale, ARM11 y Cortex [2].

Cada familia de procesadores ARM cuenta con una o más versiones de arquitectura de tipo ARM, las cuales no deben ser confundidas con el tipo de familia del microprocesador. Por ejemplo, la familia de procesadores ARM11 cuenta con tres tipos versiones de arquitectura, ARMv5TE, ARMv5TEJ, ARMv5TE.

Hoy en día la familia de procesadores ARM utilizada en casi cualquier dispositivo electrónico es la ARM Cortex, en este trabajo solo hablaremos de esta familia, ya que es la que domina en la mayoría de dispositivos electrónicos modernos, gracias a que posee las características técnicas necesarias para la demanda tecnológica actual. La familia Cortex está compuesta por tres subfamilias que se mencionan en el siguiente apartado

Serie cortex-A.

La serie de procesadores Cortex-A es capaz de proporcionar soluciones para los dispositivos que llevan a cabo tareas de cómputo complejas tales como poder funcionar con un sistema operativo y sus múltiples aplicaciones. Este tipo de procesadores suelen ser utilizados en teléfonos inteligentes, plataformas informáticas móviles, televisores digitales, módems, entre otros. Además, la eficiencia energética de este tipo de procesadores proporciona un gasto de energía significativamente bajo.

La serie de procesadores Cortes-A cuenta con una gran variedad de versiones de arquitecturas ARM, tales como: Cortex-A72, Cortex-A57, Cortex-A53, Cortex-A17, Cortex-A15, Cortex-A9, Cortex-A8, Cortex-A7 y Cortex-A5.

Serie Cortex-R.

Esta serie de procesadores está destinada para aplicaciones de tiempo real, ofreciendo soluciones de alto rendimiento para sistemas embebidos, estos procesadores combinan las características esenciales de los procesadores ARM y la fiabilidad de respuesta en tiempo real que se requiere en el mundo de los sistemas embebidos. Cabe mencionar que el procesador con el que está equipado el Beaglebone no es de esta serie, sino de una modificación a la arquitectura A8.

Este tipo de procesadores suele ser utilizado en sistemas que necesitan control de dispositivos en tiempo real. Por ejemplo, un motor de un disco duro el cual se controla mediante modulación PWM, impresoras con motores de pasos o servomotores, y gran cantidad de dispositivos que

cuentan con actuadores. La serie de procesadores Cortex-R cuenta con las siguientes versiones de arquitecturas: Cortex-R7, Cortex-R5 y Cortex-R4.

Serie Cortex-M.

La serie Cortex-M está optimizada para trabajar conjuntamente con dispositivos que suelen consumir una gran cantidad de potencia, a diferencia de la serie Cortex-R, estos dispositivos no están destinados para sistemas embebidos sino para aplicaciones de tipo industrial o para aplicaciones en el hogar que requieren un consumo de potencia mayor como un refrigerador o una lavadora. La serie de procesadores Cortex-M cuenta con las siguientes versiones de arquitecturas: Cortex-M7, Cortex-M4, Cortex-M3, Cortex-M1, Cortex-M0+ y Cortex-M0.

Hasta este punto ya hemos hablado brevemente de las distintas series (subfamilias) y versiones de arquitecturas que componen toda la familia Cortex, sin embargo, el procesador que más nos interesa es el ARM Cortex A8 ya que la tarjeta Beaglebone lleva una versión de este microprocesador modificada por Texas Instruments para poder ejecutar aplicaciones de tiempo real. En la siguiente figura podemos observar el diagrama de bloques de la estructura interna de la arquitectura A8.

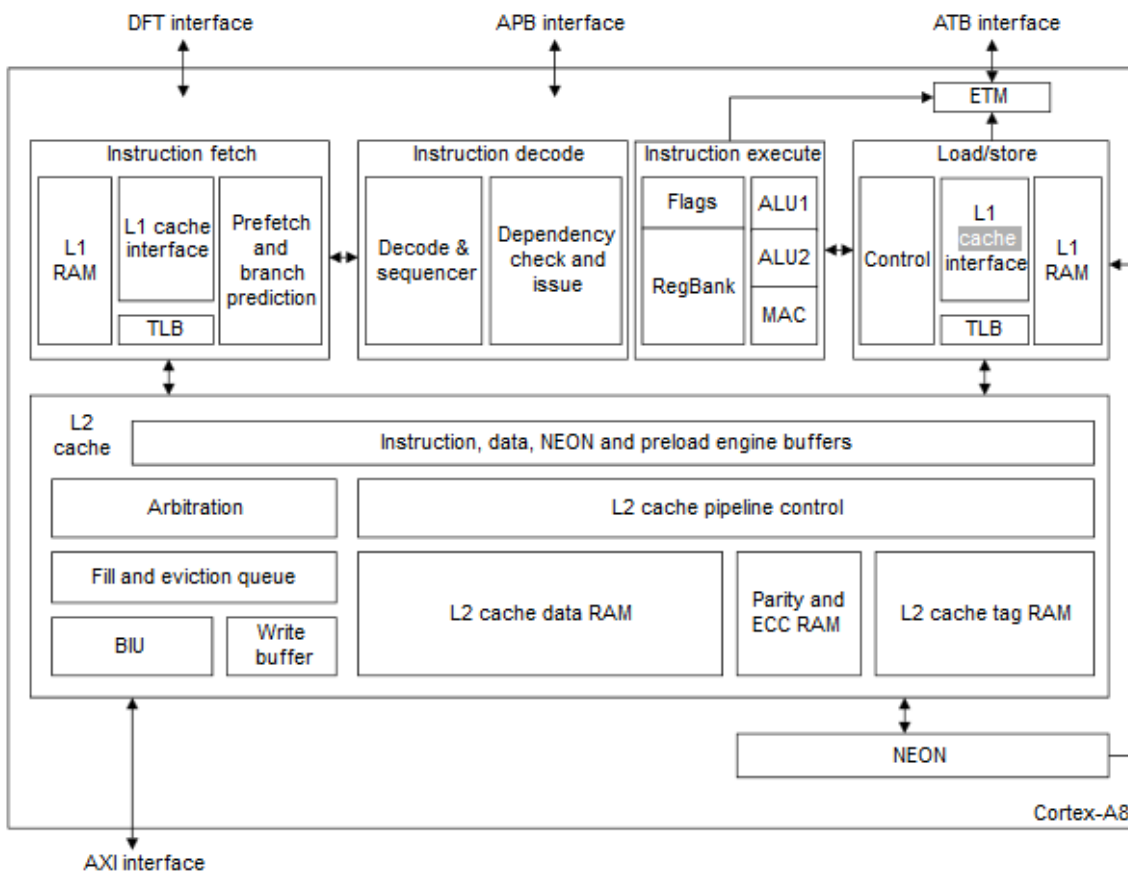


Figura 1-1 Diagrama de bloques del procesador ARM Cortex A8 [3]

Extractor de instrucciones (Instruction fetch).

Es el período que tarda la unidad central de proceso (CPU) en ejecutar una instrucción de lenguaje máquina. Comprende una secuencia de acciones determinada que debe llevar a cabo la CPU para ejecutar cada instrucción en un programa. Cada instrucción del juego de instrucciones de una CPU, puede requerir diferente número de ciclos de instrucción para su ejecución[4].

Decodificador de instrucciones (Instruction decode).

Esta unidad es la encargada de la secuencia y decodifica todas las instrucciones de procesador, incluyendo las instrucciones destinadas a los coprocesadores encargados de administrar las tareas de los demás módulos. También se encarga de secuenciar otro tipo de tareas como eventos de depuración (debug events), los cuales sirven para depurar los datos e instrucciones que ya no son necesarias para el procesador.

Otra tarea importante que desempeña este módulo es establecer la secuencia de inicio y reinicio del sistema, así como los procesos que debe seguir una interrupción ya sea interna o externa.

Ejecución de instrucciones. (Instruction execute).

Esta unidad cuenta con dos unidades aritméticas lógicas o ALU (arithmetic logic unit por sus siglas en inglés) y un generador de direcciones, el cual se encarga de cargar las instrucciones asignándoles una dirección virtual que corresponde a cada ALU.

Carga / almacenamiento (Load / Store).

Está constituida por una serie de módulos encargados de administrar los datos de la memoria caché L1 a través de sus diferentes elementos. La memoria caché L1 es la encargada de almacenar los datos más importantes y de uso más frecuente para las ALU.

L2 caché.

Esta unidad incluye la memoria cache L2 la cual se encarga de almacenar los datos más utilizados provenientes de la memoria RAM. Además, presta servicio para fallos y funcionamiento de la memoria cache L1.

NEON SMID (Single instruction, multiple data).

Esta unidad capaz de acelerar el procesamiento de señales multimedia como codificación y decodificación de video en 2D y 3D, gráficos 2d y 3d, audio, imágenes entre otras cosas.

ETM (Embedded Trace Macrocell).

Esta unidad permite la reconstrucción de un programa o flujo de instrucciones que fueron ejecutados con anterioridad con la finalidad de no acceder nuevamente a las memorias de datos y ahorrar tiempo de ejecución.

1.2 Sitara AM3358 ARM Cortex-A8.

La línea Sitara es una de las familias de procesadores más popular y de mayor rendimiento de la compañía Texas Instruments, es por esta razón que la tarjeta de desarrollo Beaglebone rev C cuenta con uno de estos procesadores como unidad principal de procesamiento.

La línea Sitara AM335x es una familia de procesadores basado en ARM Cortex A8, pero mejorado para su mejor rendimiento en procesamiento gráfico; sin embargo, necesita de una tarjeta de video externa para aplicaciones que requieran mayor velocidad. También poseen periféricos de uso común, y opciones de interfaces industriales tales como EtherCAT¹ y PROFIBUS². También fueron mejorados para poder correr sistemas operativos como Android y Linux, incluso hay distribuciones gratis de Linux y Android en la página de Texas Instruments para toda la línea Sitara.

Algunos Procesadores de la línea Sitara AM335x también cuentan con un subsistema de programación en tiempo real para la comunicación de dispositivos industriales (Programmable Real-Time Unit Subsystem and Industrial Communication Subsystem). Las PRU-ICSS (por sus siglas en inglés) están separadas del núcleo ARM permitiendo una operación independiente para obtener mayor eficiencia y flexibilidad. Las unidades PRU-ICSS nos permiten tener comunicación con interfaces y protocolos de tiempo real que actualmente se usan masivamente en las industrias modernas. Ejemplos de estas interfaces y protocolos son: EtherCAT, PROFINET, EtherNet/IP, PROFIBUS, Ethernet Powerlink, Sercos. En la siguiente figura podemos observar el diagrama de bloques que describe cómo está estructurado internamente el AM335x.

¹ EtherCAT es un protocolo informático de código abierto y alto rendimiento que pretende utilizar protocolos de Ethernet en un entorno industrial [5].

² Profibus es un estándar de comunicaciones para bus de campo [6].

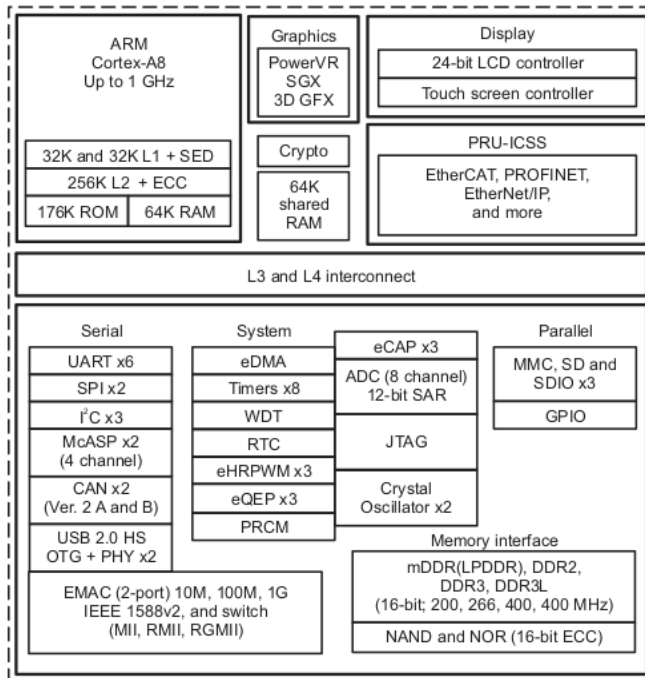


Figura 1-2 Diagrama de bloques Funcional del AM335x [7].

En el diagrama de bloques de la figura anterior podemos observar y darnos una idea de cada uno de los elementos que conforma el AM355x. Texas Instruments ha modificado este procesador para ofrecer a los usuarios un dispositivo que, además de ser capaz de poner en marcha un sistema operativo gracias a su velocidad de reloj entre otras cosas, puede ofrecernos varias posibilidades de comunicación con otros dispositivos ya que posee periféricos de comunicación muy usados por ingenieros. Los periféricos que posee este procesador incluyen UART, SPI, I2C y GPIO³. También cuenta con un controlador LCD de 24 bits para poder tener comunicación con monitores y pantallas, además posee un procesador gráfico 3D, un par de osciladores de cristal y multiplicadores de frecuencia para lograr una velocidad de reloj de 1Ghz.

El procesador AM335x tiene varias versiones, cada una con una configuración base, a la cual se le añaden o no, algunas características. Sin embargo, cada una de estas versiones están destinadas para usos diferentes. Las distintas versiones del procesador AM355x son: AM3552, AM3554, AM3556 AM3557, AM3558 y AM3559.

La frecuencia de reloj de este procesador es una de las características en donde podemos notar grandes diferencias entre las diferentes versiones, las velocidades varían desde 300Khz hasta 1Ghz. Otra característica que sólo las versiones AM3554, AM3558 y AM3559 tienen, es el acelerador gráfico, el cual es de gran utilidad para aplicaciones multimedia en donde se utiliza procesamiento de video e imágenes.

Una característica que sólo las versiones AM3556 AM3557, AM3558 y AM3559 tienen, es que estas versiones incorporan los subsistemas PRU-ICSS, lo que significa que sólo estas versiones de procesadores son capaces de ejecutar aplicaciones de tiempo real.

³ UART(Universal Asynchronous Receiver-Transmitter), SPI(Serial Peripheral Interface), I2C(Inter-Integrated Circuit), GPIO(General Purpose Input/Output).

Cabe mencionar que la tarjeta de desarrollo que empleamos en este trabajo incorpora un procesador AM3358 como unidad principal de procesamiento. En la siguiente tabla podemos observar una comparación entre el AM3358 y los demás procesadores de la misma familia siendo el AM3358 el más potente de toda la familia de procesadores Sitara AM335x.

FUNCTION	AM3352	AM3354	AM3356	AM3357	AM3358	AM3359
ARM Cortex-A8	Yes	Yes	Yes	Yes	Yes	Yes
Frequency	300 MHz 600 MHz 800 MHz 1000 MHz	600 MHz 800 MHz 1000 MHz	300 MHz 600 MHz 800 MHz	300 MHz 600 MHz 800 MHz	600 MHz 800 MHz 1000 MHz	600 MHz 800 MHz
MIPS	600 1200 1600 2000	1200 1600 2000	600 1200 1600	600 1200 1600	1200 1600 2000	1200 1600
On-chip L1 cache	64 KB	64 KB	64 KB	64 KB	64 KB	64 KB
On-chip L2 cache	256 KB	256 KB	256 KB	256 KB	256 KB	256 KB
Graphics accelerator (SGX530)	—	3D	—	—	3D	3D
Hardware acceleration	Crypto accelerator	Crypto accelerator	Crypto accelerator	Crypto accelerator	Crypto accelerator	Crypto accelerator
Programmable real-time unit subsystem and industrial communication subsystem (PRU-ICSS)	—	—	Features including basic Industrial protocols	Features including all Industrial protocols	Features including basic Industrial protocols	Features including all Industrial protocols
On-chip memory	128 KB	128 KB	128 KB	128 KB	128 KB	128 KB
Display options	LCD	LCD	LCD	LCD	LCD	LCD
General-purpose memory	1 16-bit (GPMC, NAND flash, NOR flash, SRAM)	1 16-bit (GPMC, NAND flash, NOR flash, SRAM)	1 16-bit (GPMC, NAND flash, NOR flash, SRAM)	1 16-bit (GPMC, NAND flash, NOR flash, SRAM)	1 16-bit (GPMC, NAND flash, NOR flash, SRAM)	1 16-bit (GPMC, NAND flash, NOR flash, SRAM)
DRAM	1 16-bit (LPDDR-400, DDR2-532, DDR3-800)	1 16-bit (LPDDR-400, DDR2-532, DDR3-800)	1 16-bit (LPDDR-400, DDR2-532, DDR3-800)	1 16-bit (LPDDR-400, DDR2-532, DDR3-800)	1 16-bit (LPDDR-400, DDR2-532, DDR3-800)	1 16-bit (LPDDR-400, DDR2-532, DDR3-800)
Universal serial bus (USB)	ZCE: 1 port ZCZ: 2 ports	ZCE: 1 port ZCZ: 2 ports	ZCE: 1 port ZCZ: 2 ports	No ZCE Available ZCZ: 2 ports	No ZCE Available ZCZ: 2 ports	No ZCE Available ZCZ: 2 ports
Ethernet media access controller (EMAC) with 2-port switch	10/100/1000	10/100/1000	10/100/1000	10/100/1000	10/100/1000	10/100/1000
Multimedia card (MMC)	3	3	3	3	3	3
Controller-area network (CAN)	2	2	2	2	2	2
Universal asynchronous receiver and transmitter (UART)	6	6	6	6	6	6
Analog-to-digital converter (ADC)	8-ch 12-bit	8-ch 12-bit	8-ch 12-bit	8-ch 12-bit	8-ch 12-bit	8-ch 12-bit
Enhanced high-resolution PWM modules (eHRPWM)	3	3	3	3	3	3
Enhanced capture modules (eCAP)	3	3	3	3	3	3
Enhanced quadrature encoder pulse (eQEP)	3	3	3	3	3	3
Real-time clock (RTC)	1	1	1	1	1	1

Figura 1-3 Tabla comparativa de las diferentes versiones de procesador Sitara AM335x [7].

1.3 Tarjetas de desarrollo ARM Cortex con sistema operativo embebido.

Hoy en día existen varias tarjetas de desarrollo basadas en un circuito con todo lo necesario para arrancar un sistema operativo, estas tarjetas son llamadas "SBC".

Un SBC es una computadora completa en un sólo circuito. El diseño de este tipo de dispositivos se basa en un sólo microprocesador que cuenta con su memoria RAM, puertos de E/S (Entrada y Salida) y una unidad de almacenamiento, la cual puede ser fija o extraíble. Estas placas cuentan con todas las características de un computador funcional en una sola tarjeta que suele ser de tamaño reducido, y que tiene todo lo que necesita en la placa base.

Debido a los grandes niveles de integración y reducción de componentes y conectores, los computadores en una tarjeta suelen ser más pequeños, livianos, más confiables y con un mejor manejo de la potencia eléctrica que los computadores de múltiples tarjetas.

Por otro lado, esto implica que actualizar uno de estos sistemas es normalmente imposible a nivel de hardware. Si hay un fallo o se necesita una actualización, es normal que tengamos que reemplazar la tarjeta completa.

Hoy en día existe una gran variedad de tarjetas de desarrollo de placa reducida, todas tienen diferentes características que las hacen más útiles para ciertas tareas. Una característica que tienen en común es que todas las tarjetas de este tipo emplean como unidad principal de procesamiento arquitectura de tipo ARM.

A continuación mencionaremos algunos de los ordenadores de placa reducida más utilizados en el diseño.

Beaglebone

Beaglebone black es una de las tarjetas con mayor capacidad de procesamiento que existen en su tipo, además cuenta con puertos de E/S que su competencia carece o tiene un número muy reducido.

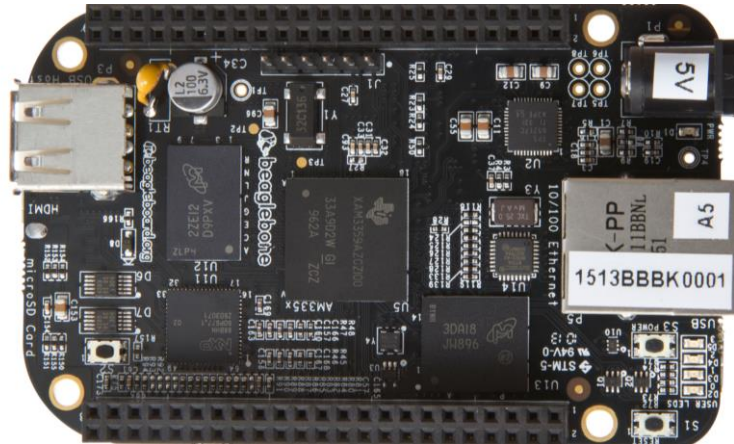


Figura 1-3 Beaglebone black rev C [8].

Raspberry Pi

Raspberry Pi es una tarjeta muy famosa, tal vez la más famosa de los SBC, sin embargo, en cuanto a potencia y cantidad de periféricos de E/S deja mucho que desear. Una característica muy buena de este dispositivo es su capacidad para desempeñar tareas relacionadas con audio y video ya que cuenta con un procesador que fue diseñado específicamente para cumplir tareas de este tipo.

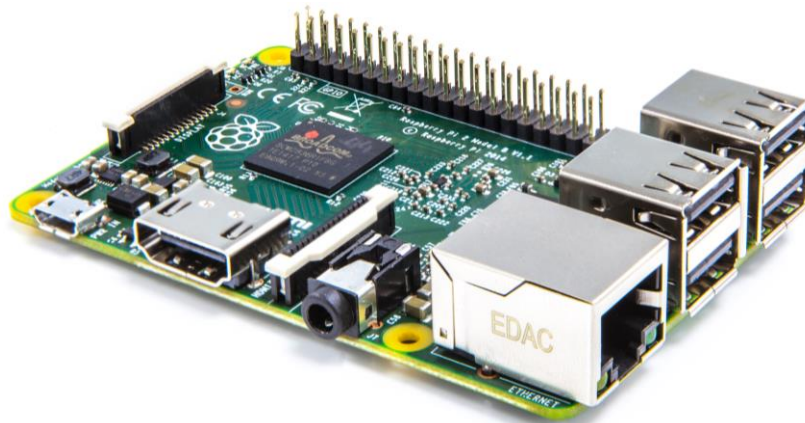


Figura 1-4 Raspberry Pi modelo B [9].

Intel Galileo

Intel Galileo es una tarjeta desarrollada conjuntamente por Intel y Atmel, una diferencia muy notable en esta tarjeta es que utiliza un procesador con arquitectura X86, a diferencia de las tarjetas de desarrollo con las que compete, las cuales utilizan procesadores con arquitectura ARM. Una característica destacable de esta tarjeta es su compatibilidad con las tarjetas Arduino, sin embargo, es demasiado costosa y deja mucho que desear en cuanto a puertos de E/S.



Figura 1-5 Intel Galileo [10].

Cubieboard

Cubieboard es una tarjeta de desarrollo con características muy similares al Beaglebone incluso hasta cuenta con dos puertos USB y 1GB en memoria RAM, sin embargo, no tiene una comunidad tan grande que la respalde por lo que carece de información sobre su uso y desarrollo de programas o librerías.

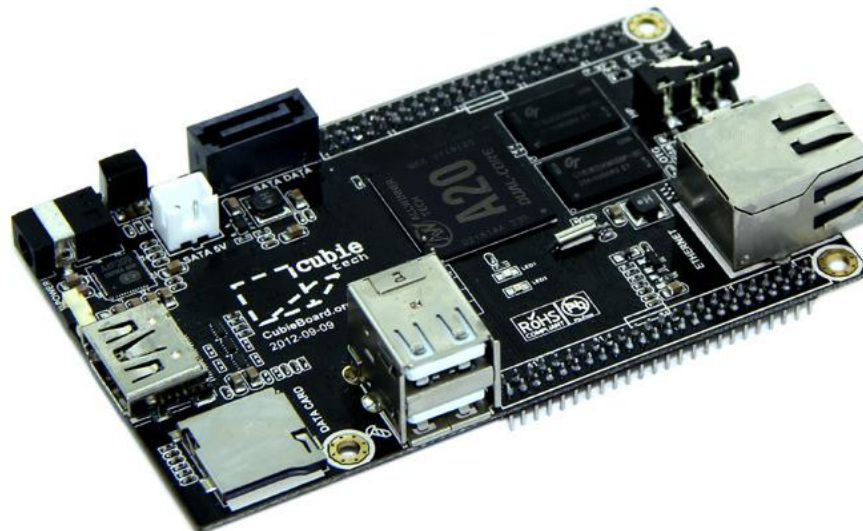


Figura 1-6 Cubieboard A20 [11].

En las siguientes tablas podemos observar la comparación de algunas de las características técnicas de los SBC mencionados anteriormente.

Nombre	Tipo de chip	CPU
--------	--------------	-----

		Arquitectura	núcleos	Frecuencia
Beaglebone Black	TI Sitara AM335x	ARM Cortex-A8	1	1 GHz
Raspberry Pi mod B	Broadcom BCM2835	ARM11	1	700 MHz
Intel Galileo gen 2.	Intel Quark SoC X1000	x86 Quark	1	400 MHz
Cubieboard	Allwinner A20	ARM Cortex-A7	2	1 GHz

Tabla 1-1 Tabla comparativa sobre características de CPU.

Nombre	GPU	RAM		
		Tamaño	Velocidad de Datos	Tipo
Beaglebone Black	PowerVR SGX530	512 MB	16	DDR3L
Raspberry Pi mod B	Broadcom VideoCore IV	512 MB	?	?
Intel Galileo gen 2.	N/A	256 MB	800	DDR3
Cubieboard	Mali-400MP2	1 GB	960	DDR3

Tabla 1-2 Tabla comparativa de GPU y RAM.

1.4 Ventajas de utilizar la tarjeta de desarrollo Beaglebone.

Beaglebone black es una de las tarjetas con mayor velocidad de procesamiento, también es el ordenador de placa reducida con mayor número de puertos de E/S que existe hasta la fecha. Esta tarjeta de desarrollo permite a los usuarios implementar todo un sistema de cómputo a cualquier proyecto de electrónica, lo cual significa que podemos incorporar cualquier tarea que desempeña una computadora. Podemos implementar una red, ya sea local o externa para poder tener comunicación con cualquier dispositivo conectado dicha red y así poder controlar las diversas funciones que nos brinda esta tarjeta de desarrollo. El Beaglebone black cuenta con una distribución de Debian Linux para sistemas embebidos.

Otra característica destacable de esta tarjeta de desarrollo es la comunidad de desarrolladores que posee, siendo la compañía Texas Instruments parte importante de esta comunidad.

Hoy en día las tarjetas Arduino son las tarjetas de desarrollo más populares y utilizadas dentro del mundo de los sistemas embebidos, sin embargo, este tipo de tarjetas se ven limitadas en muchos

aspectos en comparación con una tarjeta de desarrollo como la Beaglebone. Arduino utiliza un microcontrolador AVR de 8 bits de Atmel y ofrece puertos de E/S adecuados para conectarse con las aplicaciones del mundo real, además de contar con un entorno de desarrollo integrado y extremadamente fácil de usar. Las placas de desarrollo Arduino son apropiadas para desarrolladores que no tienen conocimientos avanzados sobre programación y estudiantes que se están iniciando en el mundo de los sistemas embebidos, sin embargo, es importante dar a conocer tarjetas de desarrollo que son capaces de ofrecer una mayor capacidad de desempeñar una o varias tareas.

En términos de herramientas de desarrollo, Beaglebone black también está bien equipado. Un intérprete de Python y un compilador C/C++ están pre configurados junto con una réplica local de Cloud9 IDE configurada para ejecutar Node.js. También se incluye la biblioteca Bonescript, basada en Node.js, que ofrece varias funciones similares a Arduino para interactuar con el hardware. No profundizaremos en el uso de Node .js y bonescript ya que fueron diseñados para el uso básico de Beaglebone, además en este trabajo se explicará el uso del Beaglebone desde un enfoque avanzado y emperando los lenguajes de programación C y C++.

En la siguiente figura podemos observar el diagrama de bloques que describe como está estructurado cada elemento que conforma el Beaglebone.

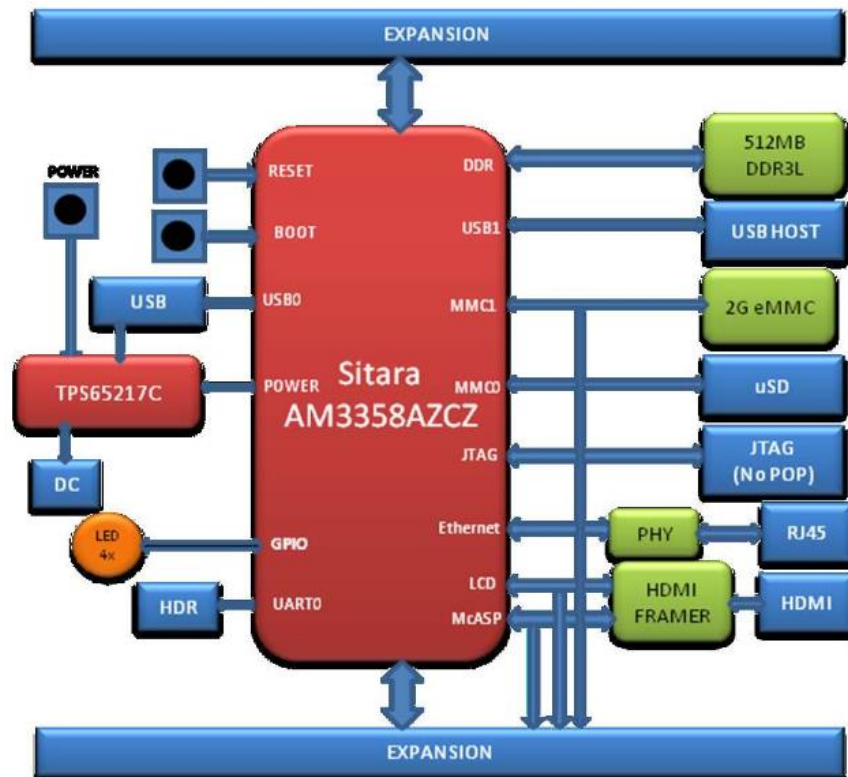


figura 1-7 Diagrama de bloques Beaglebone Black [12].

Beaglebone Black tiene un total de 92 pines accesibles a través de dos cabezales de dos filas P8 y P9. Más allá de la cantidad de GPIO (General Purpose Input/Output) disponible en Arduino o Raspberry Pi, estos pines también forman las conexiones para los módulos de expansión. Los módulos de expansión son accesorios que se montan en los pines del Beaglebone.

Otra ventaja excelente que ofrece Beaglebone es que tiene una gran cantidad de módulos de expansión a comparación de la cantidad disponibles para Raspberry Pi. El sitio de la comunidad beaglebonecapes.com, ofrecido por CircuitCo, uno de los fabricantes del Beaglebone Black, mantiene una lista de módulos de expansión compatibles que se han probado y son totalmente compatibles.

1.5 Características del Beaglebone.

Beaglebone es un SBC casi del tamaño de una tarjeta de crédito, capaz de ejecutar muchas de las tareas que nuestras computadoras ejecutan, como hojas de cálculo, procesadores de texto y juegos, también es capaz de reproducir video o capturarlo a través de una webcam.

En la siguiente tabla compararemos los datos técnicos de las tarjetas mencionadas anteriormente con respecto a su procesador.

Beaglebone fue desarrollado por la fundación Beagleboard.org con el fin de promover el uso de dispositivos de hardware enbebido y de software de código abierto. Beagleboard.org proporciona un foro para los propietarios de un Beaglebone y para los desarrolladores de hardware y software de código abierto; En este foro cada persona puede compartir ideas, conocimiento y experiencias para explotar mejor cada una de las propiedades de las tarjetas Beaglebone. Hoy en día esta tarjeta de desarrollo es fabricada por las compañías CircuitCo, element14 y Recentli.

Beagleboard.org es el resultado del esfuerzo de un grupo de personas comprometidos con el desarrollo tecnológico e interesados en la creación de dispositivos de gran capacidad, de código abierto y sistemas embebidos. La fundación beagleboard está compuesta por algunos empleados de Texas Instruments, académicos destacados de Universidades de Estados Unidos y alumnos de diversas universidades del mundo.

Beaglebone puede ser utilizado para iniciar proyectos complejos que requieran software de alto nivel y que a su vez sea necesario implementar circuitos con elementos electrónicos básicos. También representa una gran opción para proyectos en los que es necesario aprovechar las características de los sistemas operativos abiertos como Linux. Un dispositivo de software libre como beaglebone puede implementar a un proyecto tecnologico casi cualquier dispositivo que esta hecho para una PC ya que hoy en día Linux cuenta con drivers de casi cualquier dispositivo compatible con Windows como adaptadores wifi usb, webcams, tarjetas de audio usb, hubs usb, teclados, ratones, etc.

Esta tarjeta es capaz de soportar una gran cantidad de sistemas operativos de alto rendimiento. Ejemplos de los sistemas operativos que pueden ser instalados y ejecutados en la tarjeta son: Android, Debian, Angstrom, Distribution, Ubuntu, ArchLinux, Gentoo, Sabayon, Buildroot, Erlang y Fedora [13].

Capítulo 2. Software y hardware en el Beaglebone.

2.1 Núcleo de la tarjeta.

Como ya hemos mencionado anteriormente, la tarjeta que utilizamos en este trabajo es una de las más equipadas con respecto a puertos y periféricos de comunicación con otros dispositivos, además cuenta con características técnicas muy útiles para el desarrollador, como la velocidad de su procesador, su capacidad de memoria RAM, su módulo para poder tener comunicación con una tarjeta micro SD y su memoria interna de 2Gb, entre otras cosas. En la siguiente tabla podemos observar las características técnicas con las que cuenta esta tarjeta de desarrollo.

Procesador	Sitara AM3359AZCZ100, 1 GHz, 2000 MIPS
Motores de gráficos	SGX530 3d, 20M Polygons/S
Memoria SDRAM	DDR3L de 512 MB y 400 MHz
Flash integrado	MMC integrado de 2 GB y 8 bits
PMIC⁴	JTAG CTI de 20 pines, integrado y opción; cabezal serial
Fuente de alimentación.	MiniUSB o jack de CC; externa de 5 VCC a través del cabezal de expansión
Dimensiones de la placa	3.4" x 2.1"; PCB de 6 capas
Indicadores	Un LED indicador de alimentación, 2 LEDS controlados por Ethernet, 4 LEDS controlados por el usuario y el sistema operativo.
Puerto de cliente HS USB 2.0	Acceso a USB0, modo cliente vía miniUSB

⁴ Es un CI (circuito integrado) encargado de controlar el flujo de corriente eléctrica de los demás CI de un sistema, con la finalidad de optimizar el consumo de energía.

Puerto de host HS USB 2.0	Acceso a USB1, toma de tipo A, de 500 mA
Entrada de usuario	Botón de reinicio, botón de inicio, botón de encendido
Salida de video	HDMI 16b, 1280 x 1024 (máx.), 1024 x 768, 1280 x 720, 1440 x 900 con soporte EDID
Audio	Vía interfaz HDMI, estéreo
Puerto serial	Acceso UART0 vía el cabezal TTL de 6 pines y de 3.3 V (el cabezal esté relleno)
Ethernet	10/100, RJ45
Conector SM/MMC	microSD, 3.3 V
Peso	39.68 gramos
Alimentación	210-460 mA a 5 V (según la actividad y la velocidad del procesador)

Tabla 2-1. Características técnicas del Beaglebone Black rev C.

Beaglebone también posee una buena dimensión de tamaño si tomamos en cuenta sus características técnicas, además la ubicación física de cada uno de sus conectores, leds y botones presenta una ventaja a la hora de interactuar con otro tipo de dispositivos debido a que están muy bien distribuidos y hay mucho espacio entre cada uno de ellos. En la siguiente figura podemos observar e identificar cada uno de sus conectores, indicadores led y botones.

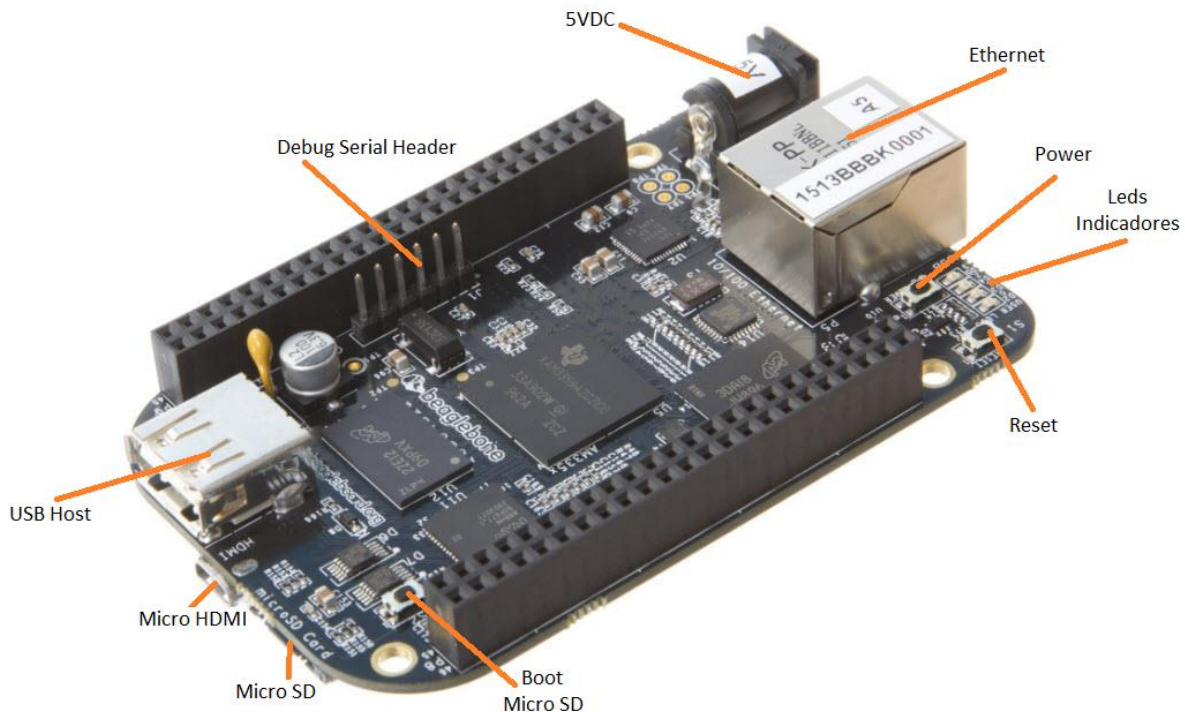


Figura 2-1. Conectores, leds y botones.

También podemos observar que cuenta con entrada para una memoria Micro SD, la cual podemos utilizar como unidad extraíble de almacenamiento o cargar un sistema operativo distinto al que trae la tarjeta. El sistema operativo puede ser cargado presionado el botón “Boot” mientras encendemos la tarjeta. El host USB con el que cuenta es de gran ayuda ya que podemos conectar un gran número de dispositivos USB compatibles con Linux, que van de impresoras, webcams, memorias e incluso un Hub USB⁵.

Otro aspecto importante de esta tarjeta es que incorpora un puerto Ethernet, el cual podemos utilizar para acceder a una red o para que la tarjeta pueda tener acceso a internet, ya que Ethernet básicamente es un estándar para redes de área local que cualquier computadora posee y que hoy en día se usa masivamente.

Al usar esta tarjeta de desarrollo podemos conectar un monitor, pantalla o display que cuente con un puerto HDMI (High Definition Media Interface por sus siglas en inglés) y visualizar el ambiente gráfico que ofrece la versión de Debían Linux que contiene nuestra tarjeta. HDMI es una norma de audio y video digital que hoy en día es usado principalmente por televisores digitales, tabletas, computadoras, proyectores, videocámaras, entre otros dispositivos.

HDMI permite el uso de vídeo computarizado, mejorado o de alta definición, así como audio digital multicanal en un único cable.

El conector estándar de HDMI tiene 19 pines, este tipo de conector está presente en la gran mayoría de dispositivos electrónicos que no son portátiles, sin embargo, la mayoría de dispositivos portátiles o de tamaño reducido cuentan con puertos HDMI denominados Mini HDMI y Micro HDMI. La salida HDMI que incluye el Beaglebone es de tipo Micro. En la imagen siguiente podemos observar los tres diferentes tipos de conectores HDMI.



Figura 2-2. Conectores HDMI tipo macho.

Con respecto a la comunicación podemos usar tres medios, los cuales son: mediante el cable USB, mediante el puerto Ethernet y mediante el Puerto Serial Debug de seis pines. Este último está incluido como un puerto de emergencia por si la tarjeta presenta algún tipo de problema o falla, ya que nos permite tener una comunicación directa con el procesador a través de uno de los múltiples módulos UART que incluye el procesador Sitara AM3358. En este trabajo no haremos mucho énfasis en este puerto de comunicaciones ya que mientras la tarjeta no presente algún tipo de complicación, es más viable comunicarnos con la tarjeta mediante otros medios.

⁵ Es un dispositivo que permite concentrar varios puertos USB a partir de un host USB.

Con respecto a la alimentación eléctrica, contamos con el puerto mini USB y un conector para un Jack de alimentación de 5.5mm de diámetro. Es recomendable utilizar el Jack de alimentación, una fuente eléctrica de 5v y 1A, sin embargo, cuando se utiliza una gran variedad de dispositivos conectados ya sea en los pines o mediante un Hub USB, se recomienda usar una fuente de alimentación que sea capaz de proveer un voltaje de 5v y una corriente de 2A.

En cuanto a los leds, cada uno está dedicado a indicar la actividad de algunos de los elementos que conforman la tarjeta. A continuación mencionaremos las funciones de cada uno de los leds indicadores.

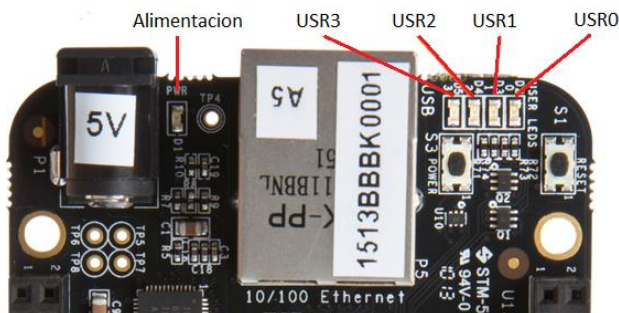


Figura 2-3. Leds Indicadores.

- USB3: Indicador de actividad de la memoria interna eMMC.
- USB2: Enciende cuando el kernel no está en un estado denominado Idle loop⁶.
- USB1: Indicador de actividad de la memoria Micro SD.
- USB0: Indicador de actividad del kernel del sistema.

2.2 Acceso a los puertos de entrada y salida (I/O).

Beaglebone cuenta con un total de 92 pines de conexión distribuidos en dos filas de 46 pines, las cuales se denominan P8 y P9. En la siguiente figura podemos observar e identificar las filas de pines P8 y P9.

⁶ Es cuando el procesador no está siendo utilizado por algún programa o aplicación, en pocas palabras, USB2 indica el estado de actividad del procesador.

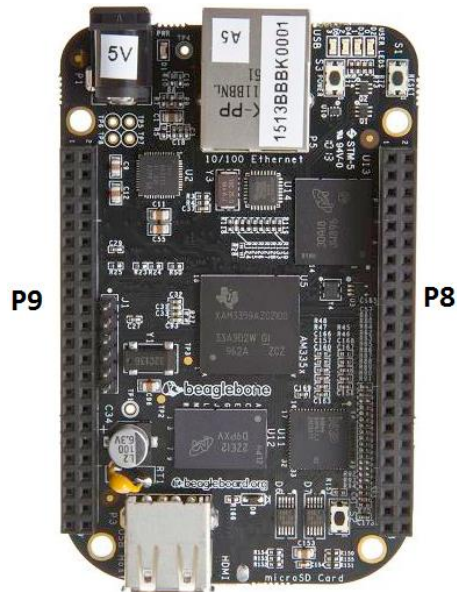


Figura 2-4. Clasificación de los pines.

Con los pines de las filas P8 y P9 podemos interactuar con un gran número de elementos sumamente útiles a la hora de desarrollar proyectos de ingeniería. La diversidad de elementos incluye pines para generar señales PWM, salidas y entradas analógicas, timers y buses sumamente útiles como I2C, UART, SPI, GMPC, MMC, entre otros. Cabe mencionar que algunos de los pines de esta tarjeta están multiplexados, por lo que algunos buses comparten algunos pines, lo que significa que no podemos usar simultáneamente algunos elementos y buses, además podemos tener hasta un máximo de 65 GPIO si se inhabilitan algunos buses.

En la siguiente tabla podemos observar una clasificación de los elementos y el número total de buses con los que podemos disponer para algún proyecto. Cabe mencionar que la distribución exacta de los elementos en cada uno de los pines está especificada en el anexo de este trabajo.

P8 y P9	Elementos	Características.
GPIO.	65 GPIOs	El número máximo de salidas y entradas digitales es de 65, todas trabajan a 3.3v, incluyendo los demás buses.
Salidas analógicas.	8 PWM	Nos permiten una señal cuadrada con un ciclo de trabajo variable, lo cual es muy útil a la hora de controlar la velocidad de motores de CD y controlar servomotores.
Entradas analógicas.	7 Entradas analógicas.	Estas entradas analógicas pueden ser usadas para leer valores de algún sensor analógico, sin embargo, el nivel de tensión no debe sobrepasar de 1.8v.

Niveles de voltaje.	5v, 3.3v y 1.8v	Contamos con dos pines que son capaces de suministrar 5v y dos que suministran 3.3v, mientras que las entradas analógicas trabajan 1.8v
Timers.	4 timers	Únicamente pueden utilizarse para generar relojes externos que interactúen con otros dispositivos.
Buses.	2 I ² C	Una interfaz para interactuar entre dos o más dispositivos de manera sincronizada.
	4 UART	Se utiliza para la conexión serial entre dos dispositivos.
	2 CAN	Es muy utilizado en procesos industriales interactuar entre varios sistemas de redes.
	2 SPI	Proporciona una conexión serial síncrona entre dos dispositivos.
	GPMC	(General Propouse Memory Controller)Podemos interactuar con dispositivos capaces de contener bloques de memoria como FPGA y ASICS ⁷ .
	2 MMC	Interfaces que son usadas para conectar la memoria eMMC y la Micro SD con el procesador.
	LCD	Este bus puede ser usado para interactuar con una pantalla LCD, (el cual se vende como un accesorio extra, pero puede ser construido por nosotros mismos sin ninguna complicación)
	2 McASP	(General-pourpose audio serial port-multichannel) es una interface de audio serial conectado al CI que controla la salida HDMI.

2.3 Accesorios externos. Tabla 2-2. Buses del Beaglebone Black.

Con respecto a dispositivos y placas de expansión, podemos decir que existe una gran cantidad de accesorios que pueden satisfacer nuestras necesidades para poder desarrollar algún proyecto, esta tarjeta de desarrollo cuenta con un número muy elevado de accesorios, superando el número de accesorios que se encuentran disponibles para sus competidores, sin embargo, en este trabajo sólo mencionaremos los accesorios básicos para poder entender el uso integral.

Memoria Micro SD.

La memoria SD es de gran ayuda ya que la tarjeta de desarrollo por sí sola cuenta con no más de 2gb de almacenamiento interno. Es más recomendable reservar la memoria interna para la instalación de programas y aplicaciones ya que con el sistema operativo que usaremos en este

⁷ Son dispositivos hechos a medida para darles un uso en particular, a diferencia de las FPGA que son capaces de implementar cualquier diseño de hardware y ser reprogramadas.

trabajo (el mismo que trae de fábrica) no es posible instalar programas en la memoria extraíble. Además, es de gran ayuda cuando deseamos utilizar la tarjeta para propósitos multimedia como reproducción de archivos de video y audio entre otras cosas.

En la página oficial de Beaglebone (<http://beagleboard.org/black>) recomiendan el uso de una memoria de tipo HC clase 10, si se desea arrancar un sistema operativo desde la memoria extraíble, esto se debe a que actualmente este tipo de memoria es la más rápida que existe en su tipo y por tal razón el sistema operativo puede correr de manera más fluida. En este trabajo utilizaremos la memoria microSD clase 4 de 8GB ya que en nuestro caso sólo se usará como dispositivo de almacenamiento de archivos.



Figura 2-5. Memoria micro SD HC clase 4.

Fuente de poder externa de 5v.

Este accesorio es sumamente necesario ya que la alimentación mediante el cable USB es deficiente cuando conectamos un gran número de elementos a la tarjeta porque la corriente eléctrica proporcionada por el puerto USB no es suficiente para la demanda de todos los dispositivos conectados. Además, es muy útil cuando se necesite tener a la tarjeta alejada de una computadora o alguna fuente de alimentación con conector USB.

Como ya habíamos mencionado, podemos emplear una fuente de poder que suministre 5v y con una corriente de 1 a 2 A. No es necesario comprar la fuente de alimentación, se puede adaptar cualquier eliminador de algún celular o aparato eléctrico que cuente con las características mencionadas anteriormente. Para este trabajo se utilizó una fuente de 5v y 2A la cual pertenecía a

un teléfono fijo inalámbrico y fue adaptada invirtiendo la polaridad⁸ para que fuera compatible con el Beaglebone.



Figura 2-6. Fuente de poder externa.

Hub USB.

Para poder conectar más de un solo dispositivo USB con nuestra tarjeta de desarrollo es necesario utilizar este tipo de dispositivos, ya que la tarjeta cuenta con solo un puerto USB. Este dispositivo es de gran ayuda porque podemos conectar varios dispositivos que pueden estar funcionando simultáneamente y sin que se afecte el rendimiento de cada uno.

Existe una gran variedad de formas y diseños de Hubs que se pueden adquirir hoy en día, sin embargo, es preferible utilizar un Hub que no tenga más de cuatro puertos ya que la velocidad de transferencia de datos se reparte entre el número de puertos. Cabe mencionar que no es recomendable comprar hubs con cables largos ya que los cables de los conectores USB están limitados a una longitud de 5m por la impedancia propia del cable.

⁸ Para invertir la polaridad únicamente cortamos los cables de alimentación para soldar en el pin central del Jack macho la alimentación positiva, mientras que la parte lateral del jack pertenece a la alimentación negativa



Figura 2-7. Hub USB marca Alteck.

Cable HDMI/mico HDMI.

Este cable es de gran ayuda ya que podemos visualizar el entorno gráfico que nos proporciona la tarjeta en una gran cantidad de monitores y televisores que cuenten con un puerto HDMI. Al visualizar el entorno gráfico del sistema operativo podemos conectar un teclado y un ratón para poder usar la tarjeta como un ordenador y tener acceso a todas las aplicaciones instaladas, incluyendo el explorador de internet con el que cuenta la tarjeta.



Figura 2-8. Cable HDMI.

Cable Ethernet.

Con este cable podemos tener acceso a nuestra tarjeta a través una red, lo cual significa que gracias al puerto Ethernet podemos incorporarnos a cualquier tipo de red con algún puerto Ethernet y tener comunicación a través de múltiples puntos de acceso conectados a la red a la que nos incorporemos.



Figura 2-9. Cable Ethernet.

2.4 Sistema operativo y comunicación con la tarjeta.

El sistema operativo con el que cuenta la tarjeta es una versión de Debían Linux diseñada especialmente para procesadores ARM y personalizada para Beaglebone. En este trabajo utilizaremos este sistema operativo ya que es el que presenta más estabilidad con la tarjeta, además es el único sistema que permite el uso de todos los elementos y buses que componen la tarjeta. En la siguiente figura podemos ver el escritorio del sistema operativo.

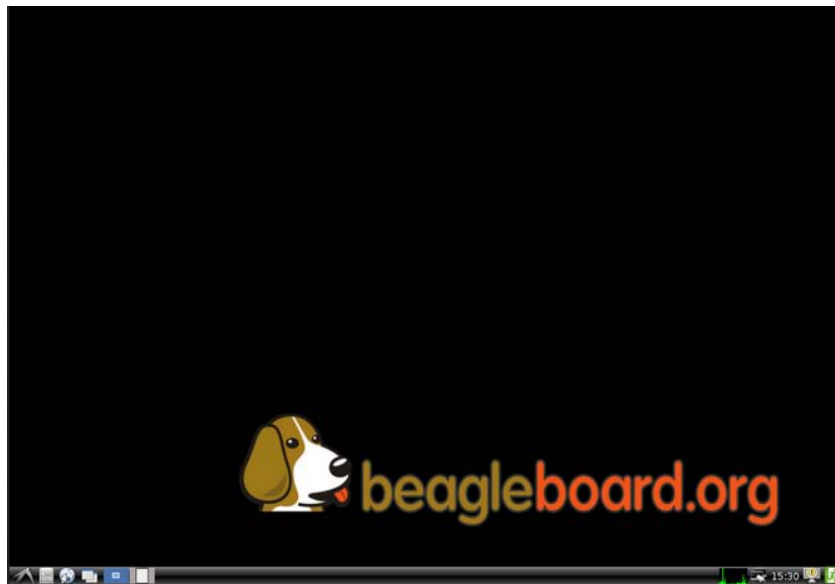


Figura 2-10. Escritorio Beaglebone.

El manejo del sistema operativo a nivel usuario (a través del entorno gráfico) es extremadamente fácil, aun cuando no se cuente con ningún conocimiento sobre sistemas operativos Basados en Linux, ya que éste fue simplificado y cuenta con muy pocas opciones. Sin embargo, para poder aprovechar al máximo nuestra tarjeta de desarrollo, debemos interactuar con el sistema operativo desde la terminal de comandos. La terminal de comandos de un sistema Linux es una ventana que

suele utilizarse para ingresar comandos escritos que nos permiten tener el control total de la computadora sin ningún tipo de restricción.

Desde la terminal podemos hacer todo tipo de acciones como: copiar, pegar, eliminar archivos, abrir aplicaciones instaladas, instalar aplicaciones, tener acceso a la mayoría del hardware y en algunos casos manipularlo directamente sin modificar el código fuente del funcionamiento del hardware.

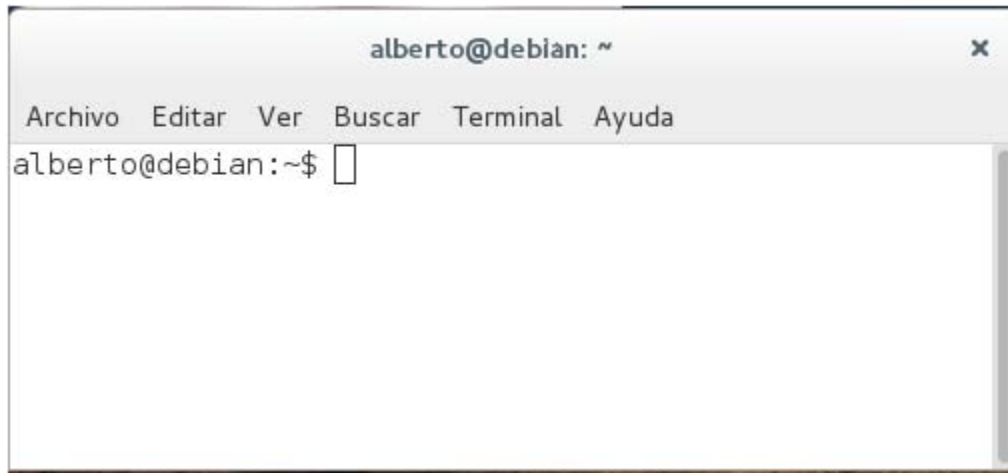


Figura 2-11. Consola de comandos Debian 8.

El aspecto de la terminal de un sistema basado en Linux suele ser muy parecido o idéntico al de la imagen 2-11, puede variar dependiendo de cada sistema operativo.

En cuanto a la accesibilidad de los sistemas operativos basado en Linux, podemos tener múltiples usuarios al igual que Windows, sin embargo, en este tipo de sistemas sólo un usuario es el que puede tener control total del sistema operativo sin ningún tipo de restricción, los demás usuarios deben tener permisos para ejecutar cierto número de tareas. A este usuario se le llama "root", el cual hace muchas cosas que un usuario común no puede, tales como cambiar el dueño o permisos de archivos y tener acceso, cualquier tipo de archivo y ubicación en cualquier unidad de almacenamiento del sistema, también existen aplicaciones que sólo el root puede instalar. El acceso al root sólo se puede efectuar mediante la terminal a diferencia de los demás usuarios que pueden acceder mediante el entorno gráfico.

El acceso a la terminal de un sistema Linux también puede ser de manera remota mediante una red, lo que quiere decir que podemos abrir y controlar la terminal de una computadora desde cualquier otro equipo, cabe mencionar que este método es el que se utiliza en todas las SBC como principal método de acceso y control. Este método de comunicación se maneja a través de un protocolo llamado SSH (Secure Shell) el cual sirve principalmente para acceder a máquinas remotas a través de una red permitiendo manejar por completo la computadora mediante un intérprete de comandos (en este caso la terminal). Este protocolo se compone principalmente por dos elementos básicos llamados cliente y servidor, el cliente es el dispositivo que accede a una máquina remota la cual se define como servidor.

Al conectar el Beaglebone a una computadora mediante el cable USB e instalar los controladores, podemos observar que nuestra computadora detecta al Beaglebone como un adaptador de red

que a su vez simula una red entre el Beaglebone y nuestra computadora, de esta manera nos permite tener comunicación vía USB mediante la implementación de protocolo SSH. Cabe mencionar que hoy en día hay un gran número de dispositivos y protocolos que son capaces de adaptar un puerto Ethernet a una interfaz USB como es el caso del Beaglebone.

Conexión SSH usando PUTTY para Windows.

Putty⁹ es una aplicación de software libre para Windows con varias opciones de comunicación con otros dispositivos como cliente SSH, telnet y comunicación serial. Putty es compatible con todas las versiones de Windows.

Con esta aplicación podemos tener acceso a la terminal de cualquier computadora con un sistema basado en Linux desde Windows vía SSH a través de una red. Para poder tener acceso a la terminal del Beaglebone mediante Windows lo primero que tenemos que hacer es conectar el Beaglebone a través del cable USB a nuestra computadora y esperar a que Windows la reconozca como un dispositivo de almacenamiento extraíble el cual contiene los controladores para Windows, Linux y Mac. Posteriormente procedemos a instalar los controladores abriendo los archivos instaladores, este paso es realmente sencillo ya que en Windows la instalación de software es muy fácil. Una vez instalados los controladores, abrimos el administrador de dispositivos en Windows y veremos un adaptador de red nuevo en nuestro sistema, tal como se muestra en la figura siguiente.

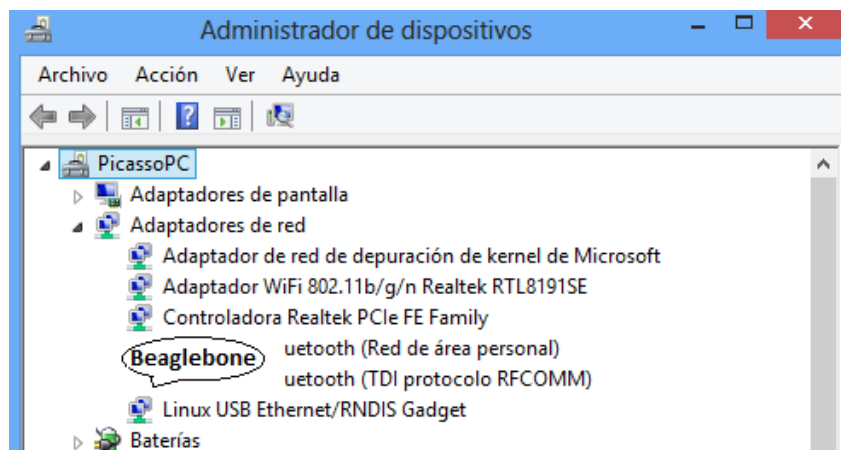


Figura 2-12. Administrador de dispositivos.

El siguiente paso es descargar Putty de su página oficial, cabe mencionar que podemos descargar el archivo instalador o el archivo ejecutable que no necesita instalación. Una vez que hayamos obtenido la aplicación procedemos abrirla para que aparezca una ventana como la que se muestra en la imagen 2-13.

⁹ Podemos descargar este software de la siguiente dirección: <http://www.putty.org/>

En toda red de computadores debe de haber una etiqueta o algún tipo de distintivo que identifique a cada una de las computadoras pertenecientes a una red, a esta etiqueta se le conoce como dirección IP¹⁰ (Internet Protocol). Como ya habíamos mencionado anteriormente, al conectar el Beaglebone con nuestra computadora mediante el cable USB se monta una red virtual entre nuestra computadora y el Beaglebone; por lo que es necesario saber la IP para poder tener acceso.

La IP correspondiente al Beaglebone es 192.168.7.2, para poder conectarnos debemos seleccionar la configuración mostrada en la figura 2-13 y dar clic en el botón “open”. En la sección “Host Name (or IP address)” escribiremos la IP y seleccionar la opción SSH en apartado que dice “Conection type” mientras que en la sección “Saved Sessions” podemos guardar la configuración para poder reutilizarla posteriormente.

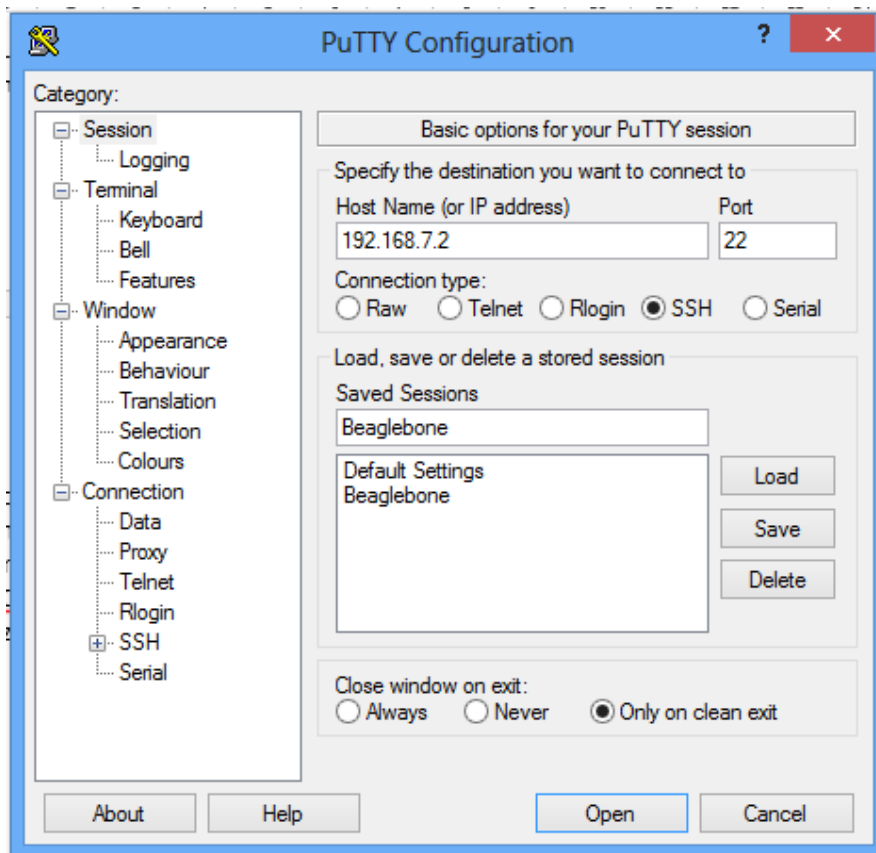


Figura 2-13. Configuración de Putty.

Cuando emerge la ventana de la terminal nos pide que escribamos el usuario con el cual queremos iniciar, nosotros iniciaremos como root ya que es el único usuario existente cuando la tarjeta está nueva, en cuanto a la contraseña, simplemente daremos INTRO si no se ha modificado o si se está usando el Beaglebone por primera vez.

¹⁰ Una **dirección IP** es una etiqueta numérica que identifica, de manera lógica y jerárquica, a una interfaz de un dispositivo dentro de una red que utilice el protocolo IP[14].

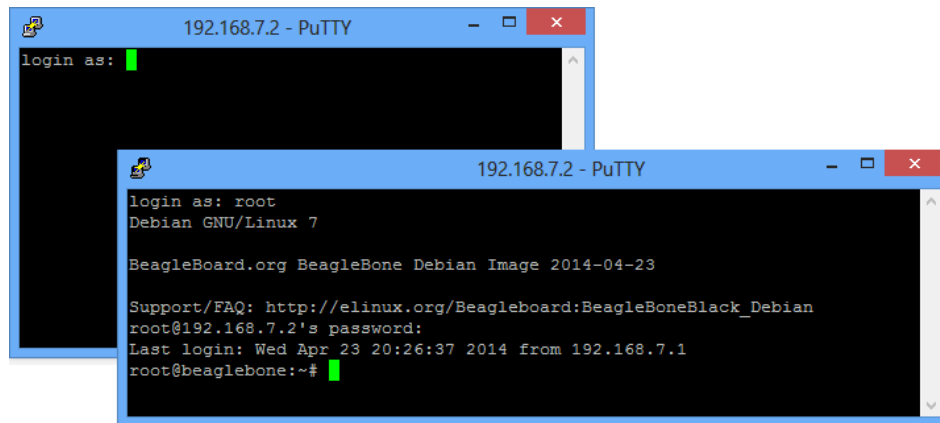


Figura 2-14. Terminal del Beaglebone en windows.

Conexión SSH usando Debían 8.

Para este trabajo se utilizó Debían 8 como sistema operativo para interactuar con el Beaglebone desde una computadora ya que este sistema operativo es empleado por una gran cantidad de desarrolladores gracias a su compatibilidad con un gran número de herramientas de desarrollo. Debían es uno de los sistemas operativos basados en Linux de mayor popularidad, desarrollado por miles de voluntarios alrededor del mundo, que colaboran a través de Internet.

El acceso a la terminal en Debían 8 es muy simple ya que lo podemos encontrar fácilmente en el menú de aplicaciones, sin embargo al abrir la terminal estaríamos accediendo a la terminal de nuestra computadora y no al del Beaglebone. Para acceder a la terminal de nuestra tarjeta lo único que necesitamos es conectarla mediante el cable USB, abrir la terminal e ingresar el comando número 24 del anexo 1, con el título de “Conexión SSH”. Cabe mencionar que en las versiones más recientes de sistemas operativos basados en Linux como Ubuntu 14.4 y Debían 8, no se necesita ningún controlador adicional ya que estos sistemas reconocen automáticamente nuestra tarjeta de desarrollo.

2.5 Repositorios del sistema operativo.

Los repositorios en Linux son grandes bancos de datos que alojan las aplicaciones y actualizaciones que el sistema necesita, entre ellos, paquetes nuevos y actualizaciones que se instalan mediante la terminal de comandos o de forma automática. Existen dos tipos de repositorios, los oficiales y no oficiales.

Los repositorios oficiales contienen las aplicaciones que cada distribución de Linux soporta y que dependiendo de sus políticas, muchas veces cuentan con un protocolo de revisión muy riguroso para asegurarse de que todos los paquetes que contienen se encuentran en estado óptimo y no representan riesgos de seguridad o estabilidad para el sistema.

Los repositorios no oficiales contienen paquetes de aplicaciones no soportadas directamente por el sistema operativo usado, por lo tanto carecen de los protocolos de revisión que mencionamos anteriormente y aunque contienen aplicaciones muy útiles para el sistema, deben ser manejados con cuidado. Estos repositorios son mantenidos por comunidades organizadas de usuarios y al ser no oficiales no están incluidos por defecto en las listas de repositorios oficiales, pero pueden agregarse muy fácilmente.

La mayoría de repositorios que se utilizan en este trabajo son oficiales, por lo que basta con un simple comando para poder instalarlos o actualizarlos. Para instalar repositorios no oficiales se tiene que modificar la lista de repositorios para poder anexarlos y posteriormente instalarlos mediante un comando.

Como ya habíamos mencionado, trabajaremos con Debían 8, por lo que necesitamos instalar un repositorio llamado "Sudo" en nuestra computadora. Para instalar dicho repositorio se debe consultar el comando necesario en el anexo tres llamado "Comandos para instalar aplicaciones".

Sudo (Substitute User DO) es una utilidad de los sistemas operativos que permite a los usuarios ejecutar programas con los privilegios de seguridad de otro usuario (normalmente el usuario root) de manera segura, convirtiéndose así temporalmente en super usuario [15].

Con los comando derivados de repositorio sudo se pueden instalar todo tipo de programas, de hecho hoy en día la gran mayoría de programas y aplicaciones comunes como Java, VLC o google chrome; se instalan mediante este tipo de comandos.

2.6 GITHUB.

GIT.

Es un tipo de software de control de versiones pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. En otras palabras, Git es un software dedicado a almacenar las diferentes versiones de código fuente de aplicaciones o programas para que los desarrolladores puedan tener una base de datos de fácil acceso, esto es de gran ayuda cuando se trabaja con proyectos muy grandes o cuando un grupo de desarrolladores trabaja conjuntamente en un solo código.

Con el paso del tiempo, Git evolucionó de ser solo un software de gestión de versiones a ser una plataforma de desarrollo colaborativo de acceso a través de internet, donde los desarrolladores pueden almacenar y compartir algunos programas y sus diferentes versiones. Hoy en día existe una gran cantidad de comunidades de este tipo, algunas de las más famosas son: GForge, GNU Savannah, Google Code, Launchpad, Github, entre otros.

GITHUB.

GitHub es la plataforma de desarrollo colaborativo que usaremos en este trabajo ya que esta plataforma es la más utilizada por desarrolladores que utilizan sistemas embebidos. Cabe destacar que GitHub es utilizado por todo tipo de programadores y desarrolladores, no sólo por desarrolladores que emplean sistemas embebidos, además de ser una de las plataformas más utilizadas en su tipo ya que podemos encontrar todo tipo de códigos y aplicaciones.

Todos los códigos desarrollados y empleados en este trabajo se almacenarán en Github ya que esta plataforma nos ofrece la posibilidad de descargar y ejecutar los programas almacenados en una cuenta de usuario a través de la consola de comandos, por lo que es sumamente práctico a la hora de querer utilizar algún código de manera rápida sin tener que copiar o volver a correr algún programa.

Para descargar e instalar GitHub y los códigos empleados en este trabajo se debe consultar el anexo 3 en la sección “Instalación de GitHub”. Este paso se debe realizar antes de continuar con el siguiente capítulo.

Capítulo 3. Comunicación entre el Beaglebone y diversos dispositivos.

3.1 Conexión a través de una red LAN.

Una red es un conjunto de equipos de cómputo y diferentes dispositivos conectados entre sí para tener comunicación e intercambiar todo tipo de datos. El principal objetivo de una red es compartir recursos e información a diferentes ubicaciones, además, también cumplen el objetivo de tener control sobre otros dispositivos de manera remota o desde un punto de acceso remoto.

Una red está formada por una gran cantidad de protocolos, cada uno ha sido desarrollado para cumplir una tarea específica dentro de una red. La comunicación a través de una red se lleva a cabo en distintas categorías básicas llamadas: la capa física y la capa lógica.

La capa física hace uso de todos los elementos que utiliza un equipo para poder sostener una comunicación con otros equipos dentro de una red, como por ejemplo: tarjetas de red, los cables y las antenas. La capa física es descrita en el modelo de referencia OSI¹¹ como modelo físico.

La primera capa del modelo OSI incluye todas las demás tareas que se necesitan realizar para lograr un enlace de red exitoso las cuales se describen en el modelo OSI con el nombre de capas. Cada capa describe una fase o tarea que se realiza en una comunicación de red. Tales capas incluyen un gran número de tareas que van desde la detección de errores, direccionamiento físico hasta el enrutamiento o la representación de la información transportada. La capa lógica incluye seis niveles descritos en el modelo de referencia OSI, los cuales son: Nivel de enlace de datos, Nivel de red, Nivel de transporte, Nivel de sesión, Nivel de presentación y Nivel de aplicación[16].

Es importante comprender qué es una red y algunas de sus características ya que gracias a ello podremos darnos cuenta de las grandes posibilidades de comunicación que podemos implementar con nuestra tarjeta de desarrollo. En este trabajo mostraremos cómo comunicarnos a través de una red LAN y como tener acceso a través de algunos dispositivos como computadoras y teléfonos inteligentes con sistemas Android e IOS.

Una red LAN (Local Area Network) es una red de equipos que abarca una área geográfica pequeña como una casa, un edificio o algún lugar que cuente con varios equipos que necesiten estar comunicados entre sí. Hoy en día es muy común tener una red LAN en casa a través de un router, el cual casi siempre es proporcionado por nuestra compañía proveedora de Internet.

Un router es un dispositivo que se encarga de enviar y recibir datos ya sea de una red a otra o de un equipo a otro a través de un cable Ethernet o WIFI. Existe una variedad muy grande de routers, sin embargo, solo nos enfocaremos en routers para redes LAN que encontramos comúnmente en servicios de internet.

¹¹ Es un modelo de red descriptivo creado en los 80 para solucionar problemas de incompatibilidad de redes y estandarizar la comunicación de computadoras.

Conexión a una red LAN.

Para poder comunicarnos con nuestra tarjeta de desarrollo a través de una red LAN es necesario disponer de un cable Ethernet y posteriormente interconectarnos a un router, para este paso es más práctico utilizar la fuente de alimentación de 5v y no el cable USB (se puede usar el cable USB si nos contamos con la fuente de alimentación) . Al realizar este sencillo paso ya formamos parte de la red conformada por el router al cual nos hayamos conectado; esto se logra gracias a un protocolo de red llamado DHCP (Dynamic Host Configuration Protocol). DHCP es un protocolo que nos permite obtener una configuración automática para poder utilizar de manera instantánea los recursos que puede proveernos una red, lo que significa que al conectarnos a un modem disponemos de una dirección IP de manera automática sin tener que modificar alguna configuración. Cabe mencionar que la mayoría de los routers de compañías proveedoras de internet tienen activado este protocolo para que cualquier equipo pueda integrarse a la red de manera automática, lo que es de gran ayuda para los usuarios que solo se integran a la red para obtener conexión a internet y no poseen conocimientos sobre redes.

Una vez conectados a una red, lo primero que requerimos para poder interactuar con nuestra tarjeta de desarrollo es la dirección IP que nos fue asignada¹², por lo que requerimos entrar al menú de configuración de nuestro router. Un router también cuenta con una dirección IP propia con la que podemos tener acceso al menú de configuración. En la mayoría de routers modernos se accede al menú de configuración escribiendo el IP del router en la barra de direcciones de un explorador de internet.

Para este trabajo es empleado un router marca TP-Link modelo TL-WR340G, sin embargo, el proceso de acceso al menú es idéntico para la mayoría de routers modernos, siendo el menú de opciones lo que varía según la marca del router.

Una vez que hayamos escrito el IP del modem en la barra de direcciones, al presionar la tecla enter nos aparecerá una ventana similar a la de la figura siguiente.

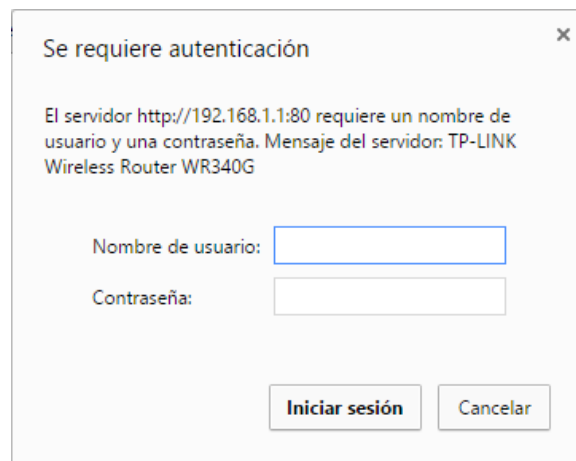


Figura 3-1. Ventana de acceso.

¹² DHCP es un protocolo que posee una lista de direcciones IP dinámicas y las va asignando a los clientes conforme éstas van quedando libres, sabiendo en todo momento quién ha estado en posesión de esa IP, cuánto tiempo la ha tenido y a quién se la ha asignado después [17].

EL nombre de usuario y contraseña varía según la marca y modelo de nuestro router o la configuración que este tenga de parte del usuario o de la compañía proveedora de internet (para saber cuál es nuestro usuario y contraseña se debe consultar el manual de usuario del router). Para el router usado en este trabajo el usuario y contraseña de fábrica es “admin”, una vez que accedamos podremos observar el entorno de configuración de nuestro router. En la siguiente figura se observa el entorno de configuración del modem usado en este trabajo.

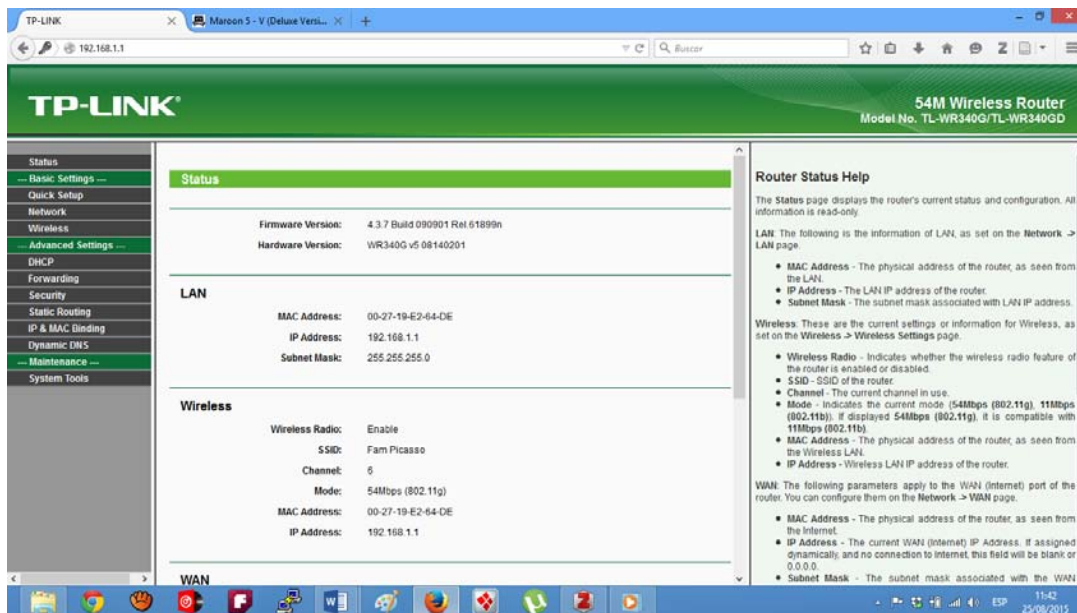


Figura 3-2. Entorno de configuración TP-Link.

Cuando accedemos al menú de configuración, lo primero que se debe hacer es buscar las opciones de configuración DHCP. Para este caso, al dar clic en la pestaña DHCP nos despliega un menú de tres opciones, el cual se muestra en la siguiente figura.

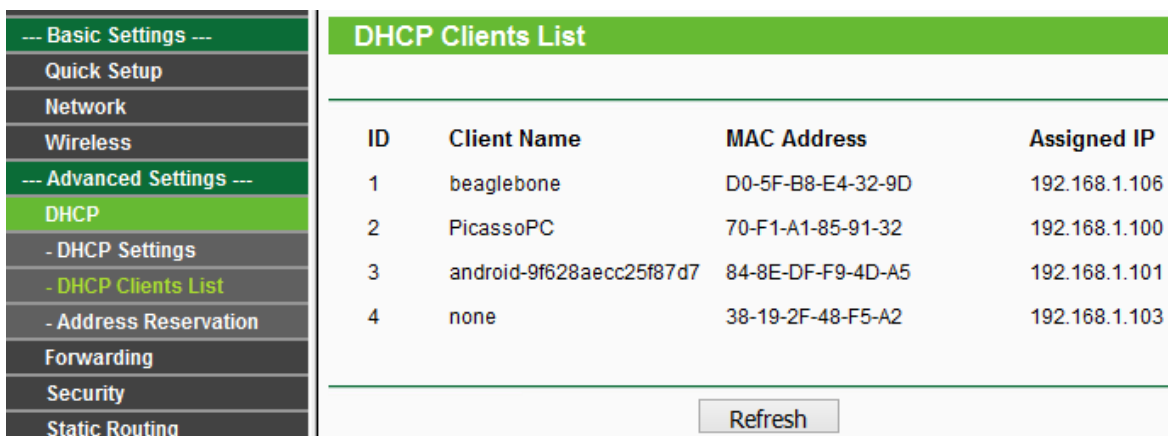


Figura 3-3. Opciones y visualización de los equipos interconectados al router TP-Link.

Para este caso nos interesa la opción que dice “DHCP client list” ya que esta opción despliega la lista de equipos interconectados en red a través del router. En routers de otras marcas debe haber una opción similar a ésta, en la cual podamos visualizar todos los equipos conectados en ese momento.

Como se puede ver en la figura 3-3, la IP asignada al Beaglebone fue 192.168.1.106, esta dirección IP será nuestra nueva dirección IP de acceso con el cual podremos establecer una conexión SSH mediante PUTTY, Debían 8 o algún otro sistema operativo basado en Linux (este paso fue explicado en el capítulo anterior). Debemos recordar que esta dirección IP puede cambiar cada que nos conectemos a la red (IP dinámico), por lo que más adelante mostraremos cómo mantener una IP fija cada vez que nos conectemos a una red.

Otro dato importante es que varios equipos pertenecientes a la red pueden sostener una conexión SSH simultánea con nuestra tarjeta de desarrollo, lo que significa que podemos enviar órdenes de varios equipos simultáneamente. Cabe mencionar que la IP que se utilizó en el capítulo anterior para establecer la conexión SSH sólo se utiliza cuando queremos establecer una conexión SSH mediante el cable USB.

3.1 Escritorio remoto.

La tecnología de escritorio remoto nos permite realizar tareas desde una ubicación remota, utilizando un dispositivo que controla a otro a través de un ordenador cliente, es decir, podemos controlar las acciones de un dispositivo (servidor) desde una computadora ubicada en otro lugar a través de una red. Hoy en día existen infinidad de protocolos y aplicaciones que son capaces de proporcionarnos un escritorio remoto entre dispositivos que cuenten con el mismo sistema operativo o entre sistemas operativos totalmente diferentes.

Algunas de las aplicaciones de escritorio remoto más utilizadas son: Terminal Services de Microsoft, Sun Ray de Oracle, Apple Remote Desktop de Mac OS X, PcAnyWhere de Symantec y VNC que es de código abierto. La mayoría de este tipo de software cuenta con su propio protocolo de comunicación, son pocas las aplicaciones de este tipo, que comparten el mismo protocolo, lo que significa que no son compatibles entre sí y que necesitamos el mismo software instalado tanto en la computadora cliente como en la computadora servidor.

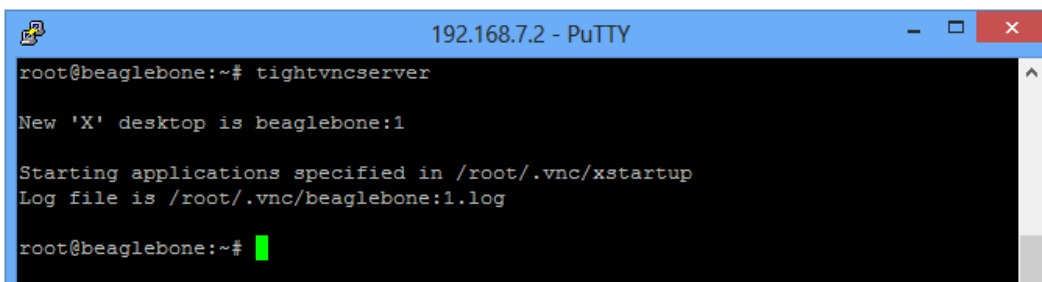
Para este trabajo se ha empleado el software VNC-Viewer desarrollado por la compañía Real VNC el cual es de código abierto, se emplea este software ya que además de ser de código abierto tiene la ventaja de poder acceder a un escritorio remoto de una computadora Linux desde un equipo con un sistema operativo Windows, ya que existe una versión para Windows.

Otro dato importante es que VNC cuenta con su propio protocolo de comunicación a través de una red, el cual también lleva el nombre de VNC, además este protocolo presenta algunas ventajas para su uso básico en redes elementales como una LAN.

El protocolo VNC permite que el cliente y el servidor negocien la codificación que se utilizará para la transmisión de datos, esto es de gran ayuda cuando el ancho de banda de los medios de transmisión es baja. Es lógico saber que cuando se realizan tareas como reproducir videos o datos guardados en nuestro servidor se requiere un ancho de banda mayor que si sólo editamos una hoja en un procesador de textos.

VNC en Windows.

Para poder acceder a nuestra tarjeta de desarrollo usando un escritorio remoto, lo primero que tenemos que hacer es abrir una aplicación llamada “tightvncserver” la cual ya viene instalada en nuestra tarjeta. Esta es una de las tantas aplicaciones que utilizan el protocolo VNC y que nos permitirá tener el control de nuestra tarjeta empleando un escritorio remoto. Para abrir esta aplicación debemos acceder a la terminal del Beaglebone y escribir el nombre de esta aplicación como un comando y posteriormente presionar la tecla enter, nos aparecerá un aviso tal como se muestra en la figura 3-4. La primera vez que accedemos a tightvncserver nos pide que definamos una contraseña para el acceso remoto.



```
192.168.7.2 - PuTTY
root@beaglebone:~# tightvncserver
New 'X' desktop is beaglebone:1
Starting applications specified in /root/.vnc/xstartup
Log file is /root/.vnc/beaglebone:1.log
root@beaglebone:~# █
```

3-4. Abriendo “tightvncserver”.

Una vez hecho el paso anterior, nuestro Beaglebone ha sido configurado como servidor para proporcionar acceso mediante escritorio remoto. El siguiente paso es descargar la aplicación VNC-Viewer¹³ e instalarlo en nuestra computadora. Al abrir la aplicación observaremos una ventana, en la cual escribiremos el IP de acceso de nuestra tarjeta(puede ser el IP empleado cuando está conectado el cable USB o un IP asignado por un router si es que nos conectamos mediante una LAN) seguido de dos puntos y el canal de transmisión TCP¹⁴, se recomienda usar el puerto 5901¹⁵.

Al dar clic en el botón que dice “Conectarse” emergerá otra ventana que nos pedirá la contraseña de acceso remoto que definimos en la terminal del Beaglebone, al abrir la aplicación “tightvncserver”, una vez escrita la contraseña damos clic en aceptar y obtendremos una ventana que nos muestra el escritorio de Debian, el cual simula el escritorio original del Beaglebone. El escritorio remoto que muestra VNC-Viewer se muestra en la figura 3-6.

¹³ Podemos descargar de manera gratuita VNC-Viewer de la siguiente dirección: <https://www.realvnc.com/>. Para este trabajo se utilizó VNC-Viewer versión 5.2.3.

¹⁴ TCP (Transmission Control Protocol) es un protocolo de transporte de datos encargado de enviar y recibir paquetes de datos de manera confiable mediante puertos, con el fin de identificar las aplicaciones emisoras y receptoras.

¹⁵ Se usa este puerto ya que la mayoría de los puertos disponibles son usados por el protocolo SSH.

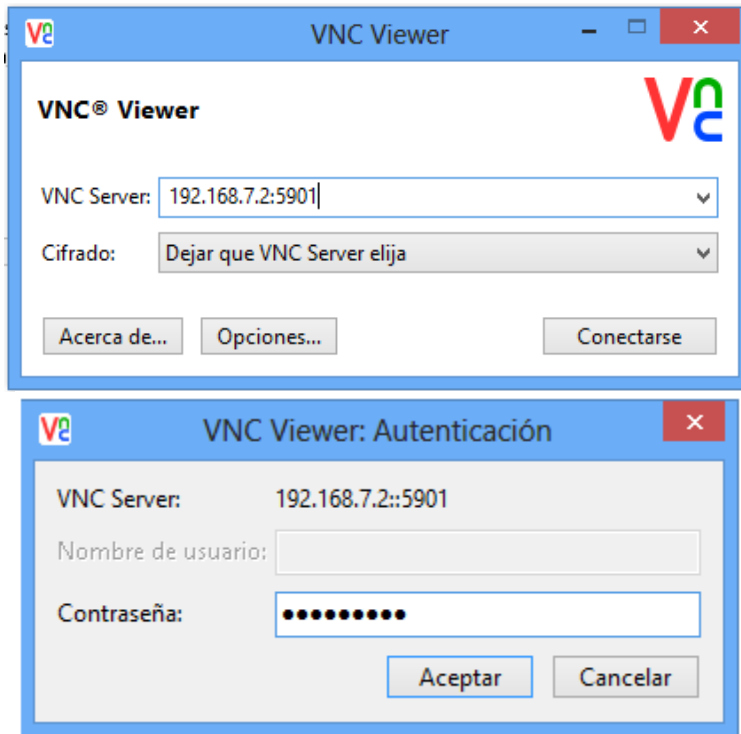


Figura 3-5. Ventanas de acceso a escritorio remoto.



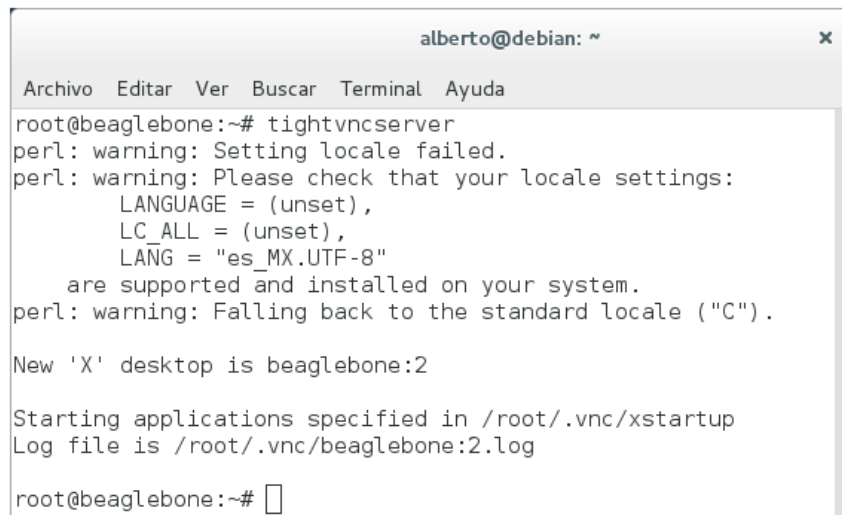
Figura 3-6 Escritorio remoto Beaglebone.

VNC en Debían 8.

Para poder abrir un escritorio remoto en debían 8, lo primero que tenemos que hacer es abrir la terminal del Beaglebone y posteriormente abrir la aplicación “tightvncserver”, como se había descrito anteriormente y como se puede observar en la figura 3-7. Cabe mencionar que este paso se tiene que hacer cada vez que queramos acceder empleando escritorio remoto ya sea en Debian o en Windows.

Una vez concluido el paso anterior, procederemos a instalar una aplicación llamada “xvnc4viewer”, la cual nos permitirá abrir una ventana de escritorio remoto en Debian. Para poder instalar esta aplicación se debe consultar el anexo número 3 en la sección “Instalación de VNCViewer”.

Posteriormente abrimos la terminal de Debian y escribimos el comando de ejecución de la aplicación, el cual está en el anexo 1 de este trabajo y es idéntico al nombre de la aplicación que instalamos anteriormente, tal como se observa en la figura 3-8. Posteriormente nos pedirá el IP del Beaglebone y la contraseña de acceso remoto que definimos en la terminal del Beaglebone al abrir la aplicación “tightvncserver” por primera vez. Una vez que hayamos accedido se mostrara una ventana con un escritorio idéntico al de la figura 3-5.



```
alberto@debian: ~
Archivo Editar Ver Buscar Terminal Ayuda
root@beaglebone:~# tightvncserver
perl: warning: Setting locale failed.
perl: warning: Please check that your locale settings:
    LANGUAGE = (unset),
    LC_ALL = (unset),
    LANG = "es_MX.UTF-8"
    are supported and installed on your system.
perl: warning: Falling back to the standard locale ("C").

New 'X' desktop is beaglebone:2

Starting applications specified in /root/.vnc/xstartup
Log file is /root/.vnc/beaglebone:2.log

root@beaglebone:~#
```

Figura 3-7. Terminal Beaglebone desde Debian 8 ejecutando “tightvncserver”.

```
alberto@debian: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
alberto@debian:~$ xvnc4viewer  
  
VNC Viewer Free Edition 4.1.1 for X - built Apr 2 2015 21:49:33  
Copyright (C) 2002-2005 RealVNC Ltd.  
See http://www.realvnc.com for information on VNC.  
Server: 192.168.7.2:5901  
  
Thu Aug 27 20:01:30 2015  
CConn:      connected to host 192.168.7.2 port 5901  
CConnection: Server supports RFB protocol version 3.8  
CConnection: Using RFB protocol version 3.8  
Password: 
```

Figura 3-8. Terminal Debian ejecutando “xvnc4viewer”.

Configuración de una IP estática.

Este proceso puede ser opcional y sólo es de ayuda cuando nos conectamos a una red LAN a través de un router mediante un cable Ethernet o un adaptador WiFi ya que, como mencionamos anteriormente, el protocolo DHCP proporciona direcciones IP distintas (dinámicas) cada que nos integramos a una red. La ventaja de crear una IP estática es que siempre que nos conectemos a una red tendremos la misma dirección IP y no tenemos que acceder al menú de configuración del router para saber qué dirección IP se nos ha asignado.

Para llevar a cabo esta configuración es necesario conectarnos con Beaglebone mediante el cable USB desde nuestra computadora y estar conectados a un router mediante un cable Ethernet o un adaptador WIFI. Posteriormente accederemos a nuestra tarjeta de desarrollo vía escritorio remoto a través del cable USB (con el IP 192.168.7.2). Una vez que podamos visualizar el escritorio remoto, procederemos a buscar el icono de configuración de parámetros de red que aparece en la barra de inicio, justo a un lado de donde se puede visualizar la hora.



Figura 3-9. Ícono de configuración de red.

Al dar clic en el ícono de configuración de red nos aparecerá una ventana como la que se muestra a continuación.



Figura 3-10. Configuraciones de red.

Después daremos clic en el ícono de Propiedades (Properties), el cual se encuentra a un lado del icono de Desconectar (Disconnect). Al dar clic al ícono de propiedades nos aparecerá una ventana como la de la figura 3-10.

Posteriormente activaremos la casilla que dice “Use Static IPs” y pondremos la configuración mostrada en la figura 3-11. Al hacer este paso también debemos colocar el IP que deseamos tener, sin embargo, no se ha puesto en la figura 3-10 ya que debemos saber qué IP queremos tener, lo cual explicaremos a continuación. Otra cosa sumamente importante es poner el IP de nuestro router en la parte que dice Gateway (Puente de acceso), en este caso el IP del router es el que se muestra en la figura 3-10 pero como ya se había mencionado anteriormente, esto puede variar para cada router.

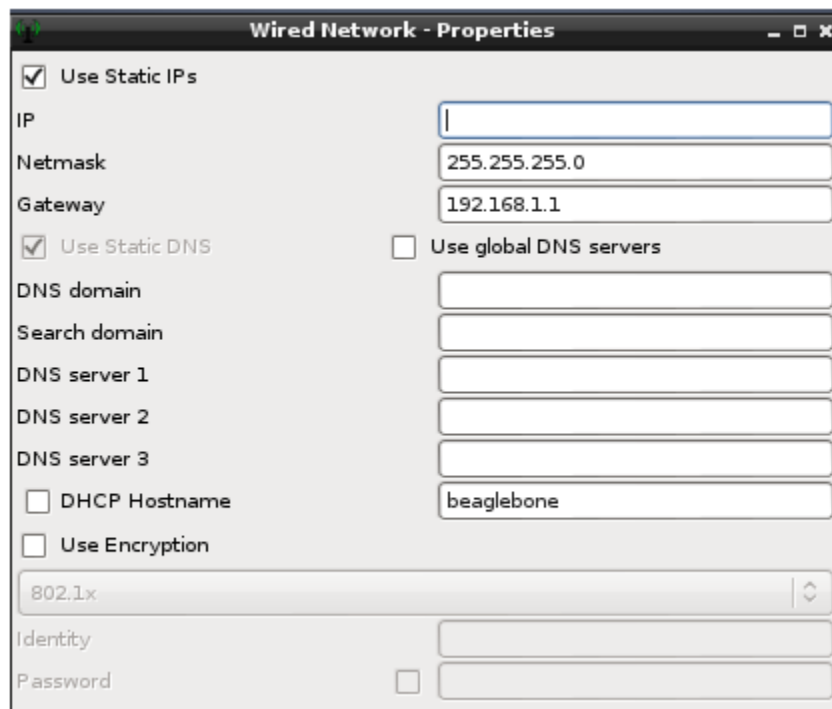


Figura 3-11. Configuración de IP estático.

Para elegir el IP debemos entender de qué manera son asignadas por nuestro router. Ya habíamos mencionado que el router asigna un IP de manera automática el cual es casi idéntico excepto por los últimos tres dígitos que va asignando de manera ascendente conforme se vayan integrando más elementos, tal como se puede ver en la figura 3-3.

En la figura 3-3 podemos observar que la asignación de cada IP según los tres últimos dígitos (sólo en este caso), no es de manera continua, esto se debe a que hay dispositivos que se conectan y desconectan de la red constantemente afectando la secuencia numérica de asignación de IP ya que el router conserva durante cierto tiempo un IP usado por un equipo que se desconectó antes de volverlo a asignar nuevamente a otro equipo, esto con la finalidad de volver a asignar el mismo IP a equipos que se desconectan y conectan a la red en periodos de tiempo muy cortos como los celulares, entre otras cosas.

Cuando nosotros asignamos un IP estático, éste ya no aparece en la lista de IPs asignados por el protocolo DHCP, ya que el protocolo DHCP sólo asigna un IP a adaptadores de red que aceptan que el protocolo DHCP les asigne un IP dinámico. Al asignar a un adaptador de red un IP estático ya no formamos parte del protocolo DHCP pero seguimos formando parte de la red, si alguna aplicación quiere tener interacción con otro equipo de la red, solo tiene que conocer su IP ya sea dinámico o estático.

Durante la realización de este trabajo se detectó el problema de que el router asignaba un IP que ya se había definido como estático, lo que ocasionó problemas de conexión con los dispositivos al tener adaptadores de red con el mismo IP. Para evitar este problema se pueden hacer dos cosas, las cuales se mencionan a continuación.

Los módems modernos cuentan con opciones para reservar IPs que el usuario no desea que sean asignados, lo cual quiere decir que a la hora de configurar un dispositivo para usar un IP estático, también debemos configurar nuestro router para que el IP que se haya definido como estático no sea asignado a ningún otro equipo que se conecte a la red. En la figura 3-3 podemos observar en el menú de opciones de DHCP una opción que dice "Address Reservation" en la cual podemos reservar uno o varios IPs que se hayan definido como estáticos. Por ejemplo, si elegimos el IP 192.168.1.109 para ser nuestro IP estático en el Beaglebone, ese mismo IP lo debemos de reservar en el menú de DHCP de nuestro router.

La otra opción que podemos hacer es escoger un IP con el cual tengamos la certeza de que el router nunca lo asignará a ningún equipo de la red. Esto se puede lograr de una manera muy sencilla, simplemente debemos conocer el número de equipos que se conectan regularmente a nuestro router, también debemos conocer el número de IPs que es capaz de asignar el router. En la siguiente imagen podemos ver las opciones para alterar el número de IPs que puede asignar el router empleado en este trabajo. Cabe mencionar que no se ha alterado ninguna opción, sólo se ingresó a este menú para ver el rango de IP que se pueden asignar de manera dinámica.

DHCP Server:	<input type="radio"/> Disable <input checked="" type="radio"/> Enable
Start IP Address:	<input type="text" value="192.168.1.100"/>
End IP Address:	<input type="text" value="192.168.1.199"/>
Address Lease Time:	<input type="text" value="120"/> minutes (1~2880 minutes, the default value is 120)
Default Gateway:	<input type="text" value="0.0.0.0"/> (optional)
Default Domain:	<input type="text"/> (optional)
Primary DNS:	<input type="text" value="0.0.0.0"/> (optional)
Secondary DNS:	<input type="text" value="0.0.0.0"/> (optional)

Figura 3-12. Opciones de IP del router TP link.

Como podemos ver en la imagen 3-11, el número de IP asignados va desde el 100 hasta el 199 según los tres últimos dígitos que son los únicos que cambian en este caso. Por lo tanto, si en nuestra red existen 5 ó 7 equipos conectados a la red, lo más seguro es que si elegimos el IP estático 192.168.1.199 o el 192.168.1.190 o el 192.168.1.160, éste no afecte o se duplique durante una asignación de IP dinámica ya que un router de una red empleada en una casa por lo regular no cuenta con más de 10 equipos. Se necesita otro tipo de dispositivos y entornos para requerir una red de más de 50 o 70 computadoras.

3.3 Conexión SSH mediante un dispositivo Android.

Android es un sistema operativo basado en Linux diseñado principalmente para dispositivos móviles con pantalla táctil como teléfonos inteligentes, tabletas, relojes inteligentes y en múltiples ocasiones lo llegamos a encontrar en televisores. Hoy en día Android es respaldado por Google y cuenta con miles de aplicaciones que han sido desarrolladas y se han ido actualizando en los últimos años. Entre las miles de aplicaciones que existen para Android, la mayoría de las cuales se pueden descargar en Google Play Store (la tienda oficial de Android) están VNC Viewer, el cual podemos utilizar como escritorio remoto, además de un gran número de aplicaciones que pueden ser utilizadas como terminal de comandos Linux.

Android ha ido evolucionando con el paso de los años y es por eso que hoy en día existen teléfonos celulares con diferentes versiones de este sistema operativo. Las diferentes versiones de Android contienen actualizaciones que se encargan de corregir fallos de programa y agregan nuevas funcionalidades. Otro dato importante es que, por existir varias versiones de Android, pueden presentarse algunos problemas de incompatibilidad con algunas aplicaciones, es obvio

que si contamos con un sistema operativo reciente y un celular de gama alta podemos ejecutar casi cualquier aplicación. Por otro lado si tenemos un celular antiguo o que no cumpla con las características que necesita nuestra aplicación, no será posible ejecutar nuestra aplicación.

Para este trabajo se utilizan aplicaciones que no requieren de un teléfono de gama alta ni de la versión más moderna de Android, sin embargo, no se garantiza que las aplicaciones implementadas en este trabajo funcionen con todas las versiones de Android o que funcionen con cualquier celular o algún otro dispositivo con sistema Android. Para este trabajo se ha empleado un celular Sony Xperia T2 ultra, el cual contiene tiene Android versión 4.4.3. Este teléfono no es de gama alta ni tiene la versión de Android más moderna pero soporta las aplicaciones que se utilizaron para este trabajo sin ningún problema.

Metodología de conexión SSH.

Antes de indicar algunos pasos para lograr la conexión SSH, lo primero que debemos hacer es asegurar que nuestro Beaglebone y nuestro teléfono celular están conectados a la misma red mediante un router (Beaglebone puede estar conectado mediante el cable Ethernet o vía Wifi). También es necesario conocer el IP de nuestro Beaglebone, ya sea dinámico o estático.

Para este trabajo se examinaron varias aplicaciones que fueron descargadas de Google Play, sin embargo la mayoría presenta problemas a la hora de manejar el teclado o simplemente les falta mejorar la manera de interacción entre el usuario y la terminal. Durante la búsqueda de aplicaciones que nos ofrecieran la posibilidad de conectarnos vía SSH y con un teclado que resultara realmente cómodo, únicamente se encontró una aplicación llamada Serverauditor¹⁶. Esta aplicación realmente puede ser capaz de simular todas las características que se pueden tener cuando se abre una terminal en una computadora, además se actualiza de una manera constante y está disponible para ser instalada en el explorador de internet Google Chrome y en el sistema operativo IOS.

Para poder abrir la terminal en nuestro teléfono celular se debe buscar la aplicación en Google Play e instalarla. Una vez instalada, abrimos la aplicación y buscamos en el menú de opciones la pestaña que dice "terminals". Una vez que hayamos abierto la sección terminals y hayamos agregado un nuevo tipo de conexión, nos aparecerá una pantalla como la que se muestra en la figura 3-13, en la cual solo debemos de poner el usuario (root), el hostname (IP del Beaglebone, en este caso se le ha asignado como una IP estática la que se muestra en la figura 3-13), y el password (si no tenemos contraseña de acceso al root lo dejamos en blanco). Para tener acceso a la terminal, sólo damos un toque sobre la paloma de color blanco que se ubica en la parte superior derecha de la pantalla de la aplicación, nos aparecerá una pantalla como la de la figura 3-13.

La terminal de comandos se puede observar en la figura 3-14, como podemos ver, al abrir una terminal remota también nos indica cual es el tipo de sistema operativo al que estamos

¹⁶ Para más información visite la página oficial de Serverauditor en la siguiente dirección: <https://serverauditor.com/>. Para este trabajo se ha utilizado Serverauditor 2.0.

accediendo y el nombre del usuario, tal como se puede observar en una terminal abierta en una computadora.

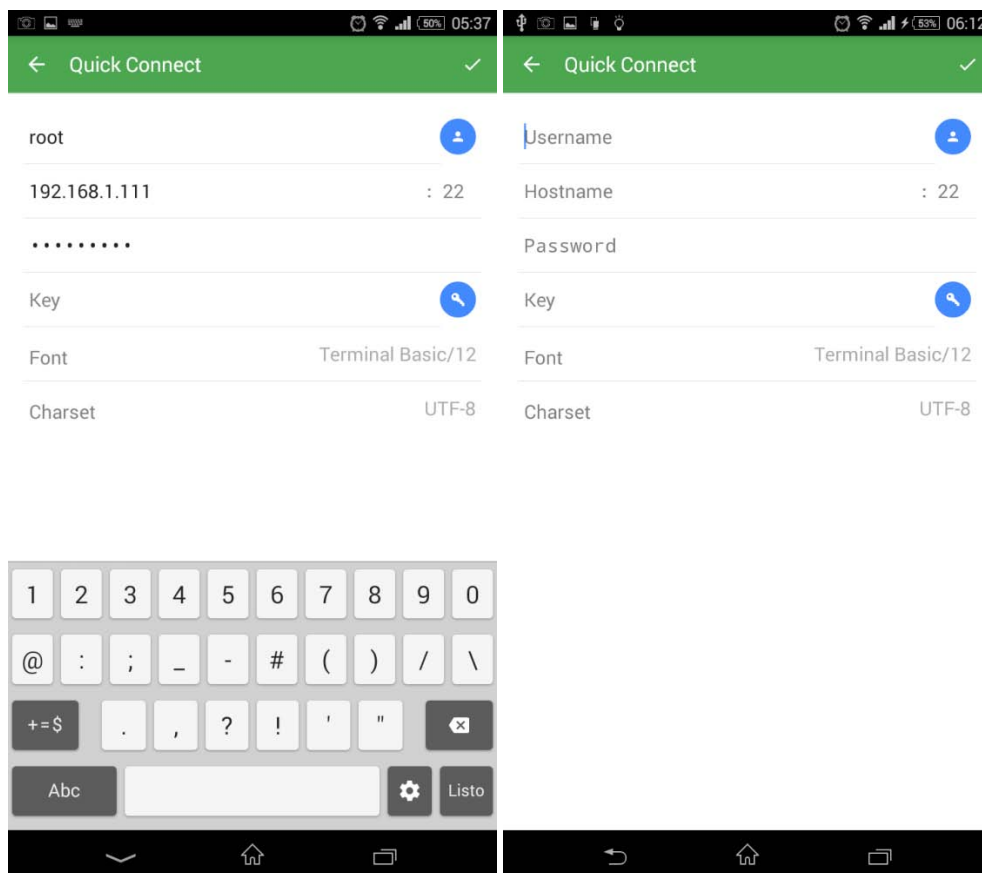


Figura 3-13. Serverauditor

```
Last login: Wed Apr 23 21:52:23 2014 from 192.168.1.103
root@beaglebone:~#
```

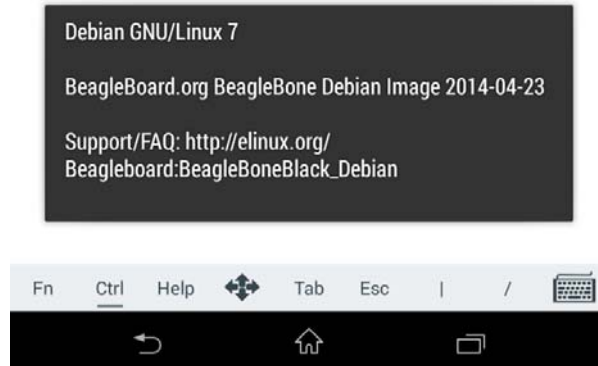


Figura 3-14. Vista de la terminal desde Android.

Escritorio remoto mediante un dispositivo Android.

Ya habíamos mencionado que VNC Viewer también está disponible para dispositivos Android, así que lo único que nos resta es descargarlo de Google Play e instalarlo en nuestro teléfono celular. Posteriormente abriremos la terminal del Beaglebone y ejecutaremos la aplicación “tightvncserver”, como ya se ha hecho anteriormente.

Una vez que accedamos a la aplicación y demos clic en ícono para crear una nueva conexión nos aparecerá una ventana, en la cual ingresaremos el IP (en mi caso utilizo el IP estático mostrado en la figura 3-15) y el canal de transmisión TCP del Beaglebone en la sección “Address”, mientras que en la sección que dice “Name” le pondremos un nombre a la conexión, puede ser el que nosotros queramos.

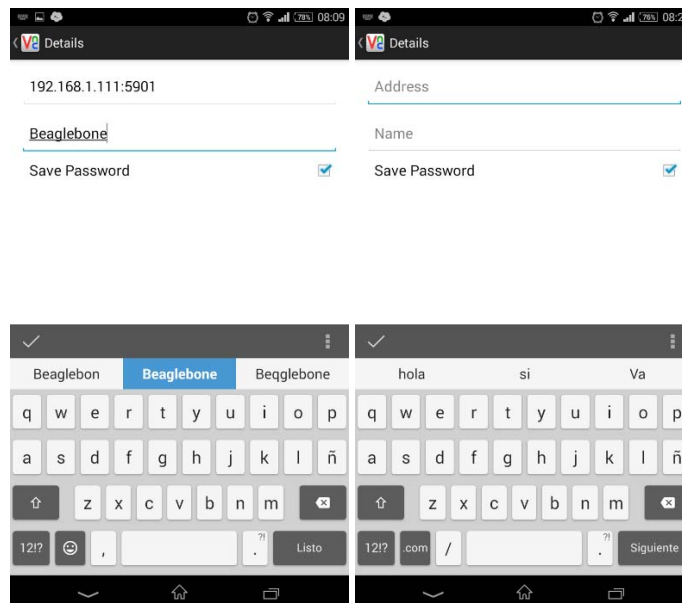


Figura 3-15. VNC viewer en Android.

Una vez concluido lo mencionado anteriormente, daremos clic en el icono de la paloma blanca que se encuentra en la parte superior izquierda del teclado. Como podemos ver en la figura 3-16, antes de establecer la conexión nos aparece una pantalla que muestra los datos que hemos ingresado con el fin de verificar que estos sean correctos. Como se puede observar en la figura 3-16, cuando demos clic en el icono que dice “Connect” nos aparecerá una pantalla que pide la contraseña de acceso remoto que definimos en la terminal del Beaglebone al abrir la aplicación tightvncserver por primera vez.

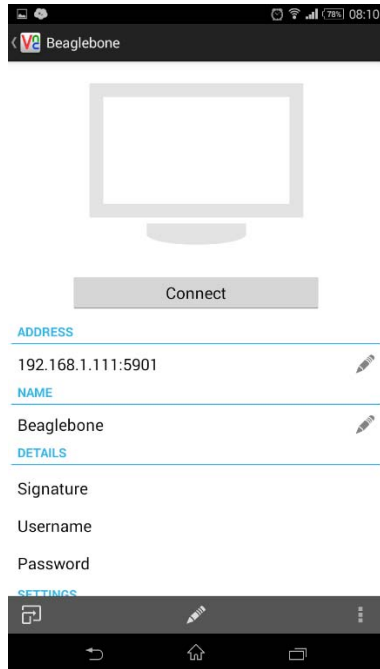


Figura 3-16. Conexión a escritorio remoto.

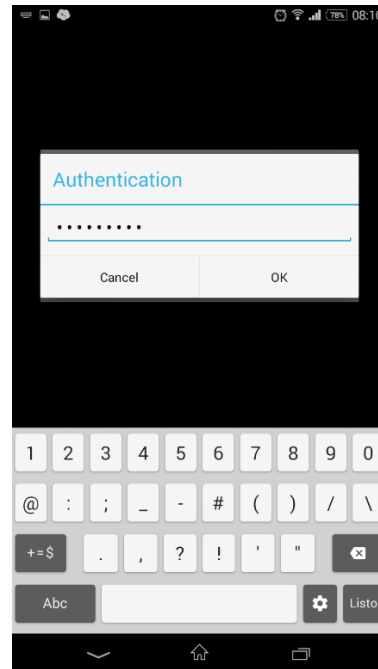


Figura 3-17. Pantalla de autenticación.

El escritorio remoto visto desde un dispositivo Android se muestra en la figura 3-18. Cabe mencionar que con los iconos que se ven en la parte superior central de la pantalla podemos manejar el escritorio simulando de manera cómoda todas las características que ofrece una computadora como el uso de un teclado y un mouse.

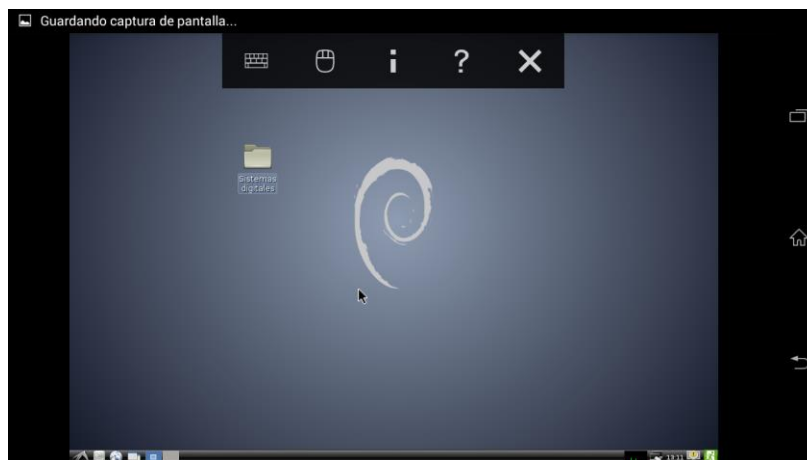


Figura 3-18. Escritorio remoto Android.

3.4 Conexión SSH mediante un dispositivo IOS.

IOS es un sistema operativo desarrollado por la compañía Apple para IPHONE, IPAD y el IPOD Touch. Este sistema operativo es única y exclusivamente para dispositivos de Apple, a diferencia de Android, el cual es usado en una gran variedad de dispositivos. A pesar de que sólo se ocupa en dispositivos de Apple, es muy popular ya que hoy en día Apple es una de las marcas más reconocidas.

Al igual que Android, IOS cuenta con varias versiones de su sistema operativo, las cuales han ido mejorando por el paso del tiempo añadiendo mejoras y corrigiendo errores. Al momento de escribir este trabajo, la versión más reciente de IOS es la 8.4.1.

Para ejecutar las aplicaciones que utilizaremos en este trabajo no se necesita el iPhone más reciente ni la versión más reciente de IOS ya que las aplicaciones que utilizaremos son las mismas que se usaron para Android. En este trabajo se ha empleado un iPhone 4s, el cual tiene instalada la versión de IOS 7.1.2.

Para esta sección no entraremos en detalle sobre las características de las aplicaciones ya que son casi idénticas a las aplicaciones que fueron hechas para Android.

Conexión SSH con IOS.

Después de ingresar a AppStore, descargar e instalar “Serverauditor”, lo único que nos queda es configurarlo de la misma manera que como se configuró en Android, en las siguientes figuras podemos observar las configuraciones correspondientes.

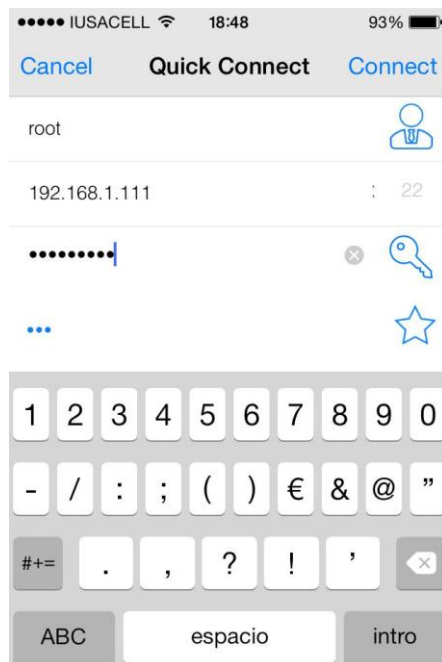


Figura 3-19. Serverauditor

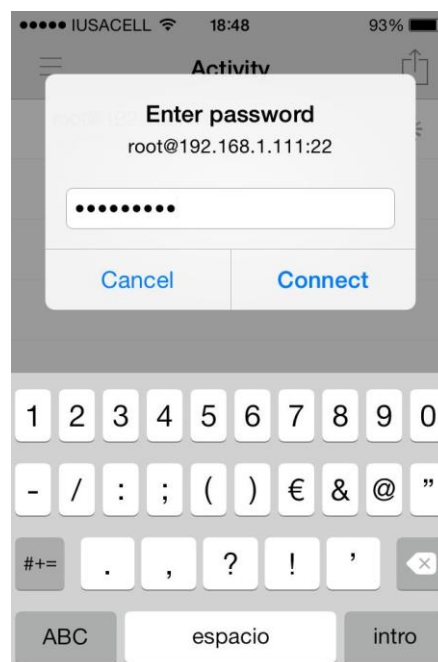


Figura 3-20. Contraseña de acceso

Escritorio remoto con IOS.

Después de descargar e instalar VNC Viewer de AppStore, procederemos a configurarlo de la misma manera que como se configuró en Android, las configuraciones se muestran a continuación.



Figura 3-21. VNC Viewer IOS.



Figura 3-22. Contraseña de acceso.

Capítulo 4. Compilación de programas con Eclipse.

4.1 Instalación de ECLIPSE.

ECLIPSE es un software de desarrollo integrado (IDE¹⁷) compuesto por un editor de código fuente y un gran número de herramientas de código abierto que nos ayudan a facilitar muchas tareas a la hora de desarrollar un proyecto tecnológico en donde intervienen un gran número de elementos.

ECLIPSE también es una comunidad de desarrolladores que juntos forman la fundación Eclipse, una organización sin fines de lucro dedicada a extender las áreas de aplicación para eclipse. Hoy en día eclipse es uno de los IDE más completos ya que ofrece soporte para un gran número de lenguajes de programación, abriendo las puertas a desarrolladores que empleen lenguajes de programación como C/C++, JAVA, PHP, entre otros.

El IDE de eclipse emplea módulos (plug-ins) para proporcionar todas sus funcionalidades, estos módulos pueden ser descargados una vez que tengamos instalado el IDE de eclipse, a diferencia de otros entornos donde las funcionalidades están todas incluidas con el software de instalación, las necesite el usuario o no. Otra ventaja sumamente importante que presenta eclipse es la capacidad de interactuar con herramientas para realizar compilación cruzada (Cross Compiler). Un compilador cruzado es capaz de crear código ejecutable para otra plataforma o arquitectura distinta a aquella en la que el compilador se está ejecutando (normalmente se le suele llamar arquitectura foránea). En este caso compilaremos programas para la arquitectura ARM desde un procesador con arquitectura x86. Este tipo de compiladores son sumamente útiles cuando se requiere compilar código para algún tipo de plataforma a la cual resulte difícil tener acceso o sea incómodo desarrollar código en dicha plataforma, por ejemplo un celular.

En cuanto módulos de cliente servidor, Eclipse provee al programador herramientas para poder interactuar con otros dispositivos mediante una red a través de protocolos como SSH y un protocolo de escritorio remoto, ya que Eclipse posee la capacidad de navegar entre los archivos y carpetas de una computadora remota.

Todo lo mencionado anteriormente convierte a Eclipse en un IDE sumamente adecuado para poder emplearlo con nuestra tarjeta de desarrollo, además es totalmente compatible con Debian 8 y Ubuntu 14.4, aunque también existe una versión para Windows. Los sistemas operativos como Windows 7 y Windows 8 no poseen herramientas que puedan interactuar conjuntamente con el IDE de eclipse para compilar y ejecutar programas hechos para arquitectura ARM. Hoy en día los sistemas operativos basados en Linux son los que tienen un mayor número de herramientas para poder desarrollar software para arquitectura ARM a través de los lenguajes de programación más utilizados como JAVA, C/C++, Python, entre otros.

Instalación de elementos para el uso de ECLIPSE.

¹⁷ Un IDE (Integrated Development Environment) es una aplicación que proporciona elementos para que el programador desarrolle software de una manera más cómoda cuando se necesita interactuar con hardware externo o de acceso remoto [18].

El primer paso antes de iniciar la instalación de eclipse es verificar con qué sistema operativo cuenta nuestra computadora¹⁸, como ya hemos mencionado, el IDE de Eclipse es compatible con Ubuntu 14.4 y Debian 8. Todos los pasos descritos en esta sección se pueden llevar a cabo en los sistemas operativos mencionados anteriormente, sin embargo, no se garantiza que funcionen en otros sistemas operativos basados en Linux.

Antes de instalar Eclipse debemos descargar e instalar una serie de repositorios y aplicaciones en nuestra computadora para que el IDE de Eclipse sea capaz de compilar y ejecutar programas hechos para la arquitectura ARM. Los comandos para instalar las herramientas necesarias se encuentran en el anexo número 4 con el título “Instalación de elementos para el uso de Eclipse”. Cabe mencionar también podemos ingresar todos los comandos del anexo 4 en la terminal y posteriormente leer en esta sección para qué se utiliza cada uno.

Lo primero que debemos hacer es crear una lista de repositorios que no son oficiales, pero son necesarios para poder ejecutar compilación cruzada en nuestra computadora, por lo que es necesario ingresar siguiente comando:

```
sudo nano /etc/apt/sources.list.d/crosstools.list
```

Al ingresar el comando mencionado nos abrirá una ventana de texto en blanco en la cual pegaremos el siguiente texto: deb http://emdebian.org/tools/debian jessie main. Tal como se observa en la figura 4-1.

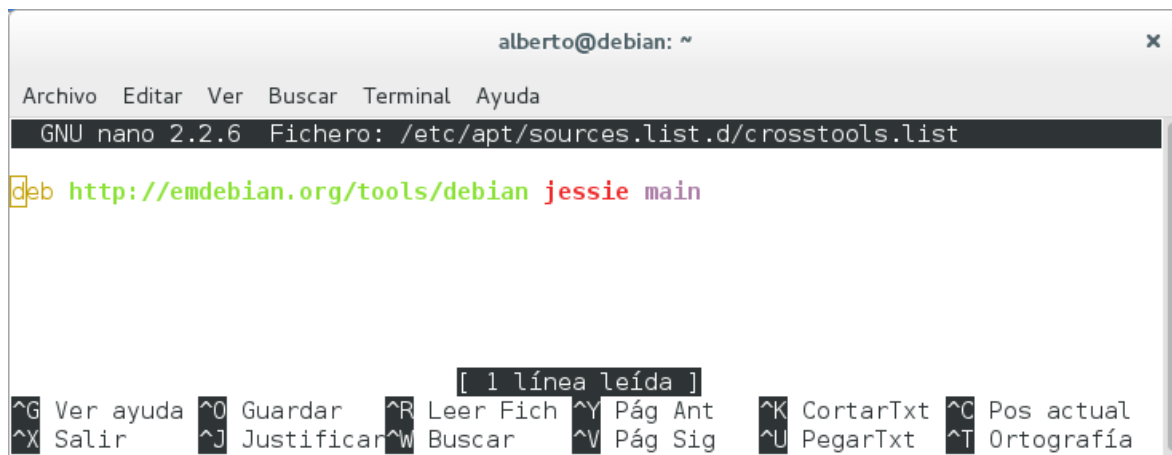


Figura 4-1. Editando Crosstools.list

Una vez hecho el paso anterior guardaremos los cambios hechos en el archivo de texto presionando las teclas ctrl+o y después daremos Enter, posteriormente saldremos de la ventana de texto presionando ctrl+x.

Un paquete sumamente importante que debemos instalar son los repositorios llamados “build-essential”, los cuales contienen un gran número de herramientas necesarias para compilar

¹⁸ Para verificar que sistema operativo tenemos debemos consultar el anexo 3 en la sección “Instalación de ScreenFetch”.

programas como por ejemplo Gcc y G++. Para instalar las aplicaciones de los repositorios “build-essentials” debemos ingresar en la consola el siguiente comando:

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

Gcc se utiliza para compilar código de C, mientras que G++ sirve para compilar código de C++. Estos compiladores son utilizados por el IDE de Eclipse para compilar nuestros códigos, pero también pueden ser utilizados directamente por el usuario desde la terminal de comandos. Podemos crear un código “.c” o “.cpp” (extensiones de los archivos de código de C y C++ respectivamente) desde la terminal de comandos consultado los códigos 5,16, 17 y 18 del anexo número 2.

Ahora procederemos a instalar una herramienta llamada CURL ingresando en la terminal el siguiente comando:

```
sudo apt-get install curl
```

CURL es una herramienta software para transferencia de archivos con sintaxis URL mediante la terminal de comandos. Esta herramienta será de gran ayuda para descargar librerías y repositorios más adelante.

Para poder trabajar con la arquitectura ARM como una arquitectura foránea y realizar compilación cruzada, debemos ingresar en la terminal, el siguiente comando:

```
sudo dpkg --add-architecture armhf
```

El repositorio instalado con el comando anterior contiene una aplicación llamada ARMhf (Hard Float). En Debian y derivados de Debian (Por ejemplo Ubuntu), “armhf” hace referencia a la arquitectura ARMv7 que utiliza toda la familia ARM Cortex, como se mencionó en el capítulo 1. Con los comandos que se muestran en la primer y cuarta línea de la figura 4-2 podemos ver con qué tipo de arquitectura trabaja nuestra computadora y qué tipo de arquitectura se puede emplear como arquitectura foránea para compilar código. En la figura 4-2 podemos observar que para este trabajo se utilizó una computadora con arquitectura x86 y se utiliza la arquitectura ARM como arquitectura foránea.

A terminal window titled "alberto@debian: ~" with a menu bar containing "Archivo", "Editar", "Ver", "Buscar", "Terminal", and "Ayuda". The terminal shows the following commands and output:

```
alberto@debian:~$ sudo dpkg --add-architecture armhf
alberto@debian:~$ dpkg --print-architecture
i386
alberto@debian:~$ dpkg --print-foreign-architectures
armhf
alberto@debian:~$ █
```

Figura 4-2. Arquitecturas de trabajo en Debian 8.

Antes de continuar, debemos actualizar la lista de repositorios que hemos agregado en los últimos pasos. Para actualizar nuestra lista de repositorios únicamente debemos ingresar el siguiente comando:

```
sudo apt-get update
```

Para poder realizar una compilación cruzada, ya sea desde la terminal o desde el IDE de eclipse, debemos instalar un repositorio llamado "crossbuild-essential-armhf". Este repositorio nos permitirá compilar códigos en C/ C++ para la arquitectura ARM con la que estamos trabajando. Para instalar "crossbuild-essential-armhf" ingresaremos en la terminal el siguiente comando:

```
sudo apt-get install crossbuild-essential-armhf
```

Hasta este punto ya podríamos desarrollar código para arquitectura ARM y compilarlo gracias a las herramientas y repositorios que hemos instalado anteriormente, sin embargo, todavía no podemos correr un archivo ejecutable hecho para arquitectura ARM desde nuestra computadora. Para poder abrir archivos ejecutables hechos para ARM en nuestra computadora instalaremos una aplicación llamada QEMU con el siguiente comando:

```
sudo apt-get install qemu-user-static
```

QEMU ¹⁹ es una aplicación para Linux con la capacidad de emular procesadores con arquitectura ARM en una máquina x86 con sistema operativo de 32 o 64 bits. Una vez que tengamos instalado QEMU podremos abrir archivos ejecutables diseñados para ARM de la misma forma que abrimos ejecutables compilados para la arquitectura de nuestra computadora.

Instalación de Eclipse.

¹⁹ QEMU tiene más funcionalidades además de las mencionadas en este trabajo. Para más información visite: http://wiki.qemu.org/Main_Page.

Para poder instalar Eclipse debemos tener acceso a su página oficial para posteriormente poder descargar el archivo instalador²⁰. En la sección de descargas podemos encontrar archivos instaladores para Windows, Linux y Mac. Una vez que seleccionemos la opción para descargar los archivos instaladores para sistemas Linux, podremos observar varios IDE que pueden ser descargados según el lenguaje de programación a emplear, en nuestro caso utilizaremos C/C++, por lo tanto debemos descargar “Eclipse IDE for C/C++ Developers” considerando el tipo de sistema operativo que tenga nuestra computadora ya sea de 32 o 64 bits.

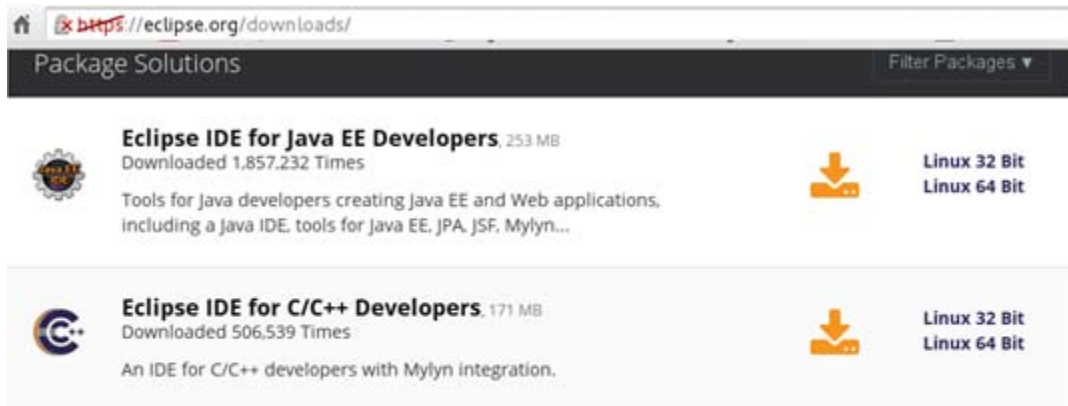


Figura 4-3. Página oficial de Eclipse.

Una vez que haya finalizado la descarga de Eclipse, procederemos a abrir la terminal y ubicarnos en la carpeta donde fue descargado nuestro archivo instalador, para este caso las descargas se concentran en la ubicación “/home/Alberto/descargas” .Para que la terminal pueda estar apuntando en nuestra ubicación de descargas, escriba el siguiente comando en la terminal:

```
cd home/Alberto/descargas/
```

Una vez que nos ubiquemos en nuestra carpeta de descargas procederemos a descomprimir el archivo que contiene el IDE de Eclipse. Cabe mencionar que el IDE de Eclipse no es instalable, para poder ejecutar Eclipse, basta con descomprimir el archivo que descargamos y dar doble clic al archivo ejecutable que dice “Eclipse”. Para poder descomprimir el archivo debemos ejecutar el siguiente comando²¹:

```
tar -xvf eclipse-cpp-luna-SR2-linux-gtk.tar.gz
```

²⁰ Par este trabajo se ha empleado Eclipse IDE for C/C++ Developers Version Luna Service Release 2(4.4.4).

²¹ Si el nombre de nuestro archivo “tar.gz” es diferente al que se muestra en este comando, debemos consultar el comando número 23 del anexo número 2 para poder descomprimir el archivo que hemos descargado.

Cuando abrimos el IDE de Eclipse por primera vez observaremos una ventana como la de la figura 4-4, la cual nos pedirá que indiquemos una carpeta de trabajo para guardar todos los códigos que desarrollaremos posteriormente. Si no modificamos la opción de carpeta de trabajo, el IDE de Eclipse creará una carpeta automáticamente llamada “workspace”.

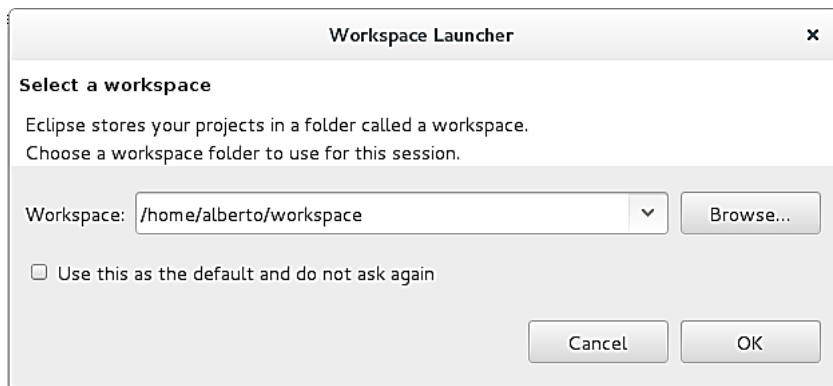


Figura 4-4. Inicio de Eclipse.

Al dar clic en el botón OK observaremos la pantalla de inicio del IDE y unos segundos después la interfaz gráfica del usuario, la cual mostrará un proyecto vacío o una hoja de código en blanco.

Para poder crear un código en C++ debemos dar clic en el símbolo que dice “New Project” el cual se encuentra en la esquina superior derecha y podemos observar en la figura 4-5 indicada con una flecha de color rojo.



Figura 4-5. Barra de menús de Eclipse.

Al dar clic en el botón “New Project” nos abrirá una ventana como la que se muestra a continuación, en la cual debemos dar clic en la opción “C++ Project” y posteriormente dar clic en el botón next.

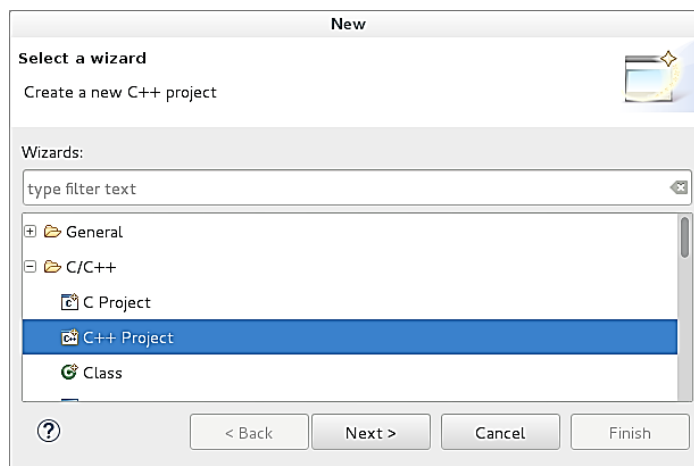


Figura 4-6. Nuevo Proyecto.

Después aparecerá una ventana como la de la figura 4-7, en la cual debemos especificar el nombre de nuestro proyecto en la sección “Project Name” mientras que en la sección “Project Type” seleccionaremos la opción “Hello world C++ Project” (cada que abramos un proyecto nuevo seleccionaremos la opción “Hello world C++ Project” ya que cuando seleccionamos “Empty Project” no es posible compilar el código que hayamos desarrollado). Por último debemos seleccionar la opción “Cross GCC” en la sección “Toolchains”, de lo contrario no podríamos realizar una compilación cruzada.

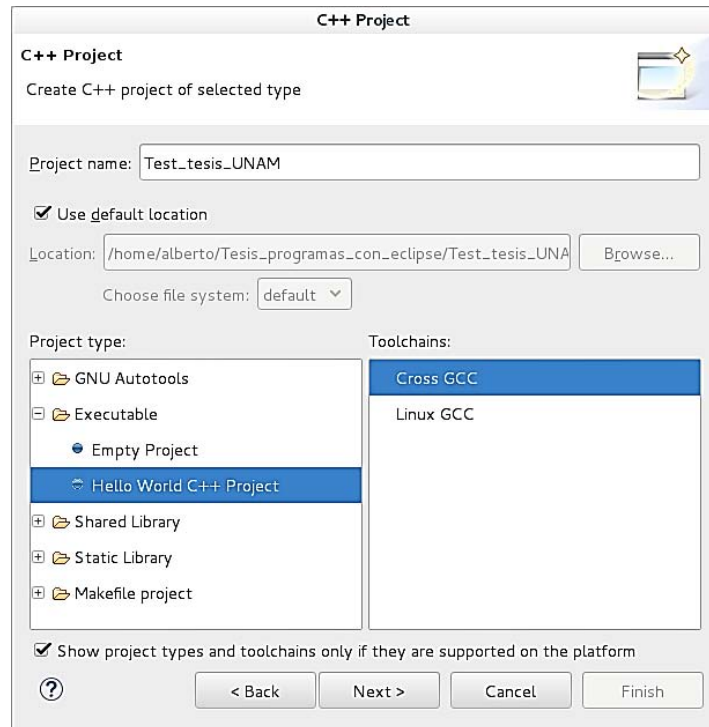


Figura 4-7. Proyecto C++.

Al dar clic en “Next” nos aparecerá una ventana como la de la figura 4-8. Esta ventana es sólo para verificar los datos del proyecto que vamos a crear, sólo se realizarán modificaciones si queremos cambiar algún dato sobre nuestro proyecto, de lo contrario debemos dar clic en “Next”.

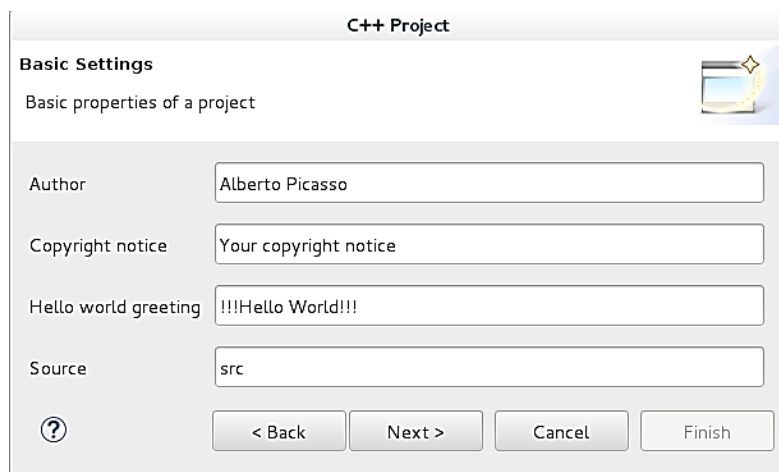


Figura 4-8. Configuración básica.

Después aparecerá una ventana llamada “Select Configuration”, en la cual debemos activar la opción “Debug” y la opción “Release” en el recuadro llamado “Configuration”, tal como se muestra en la figura 4-9.

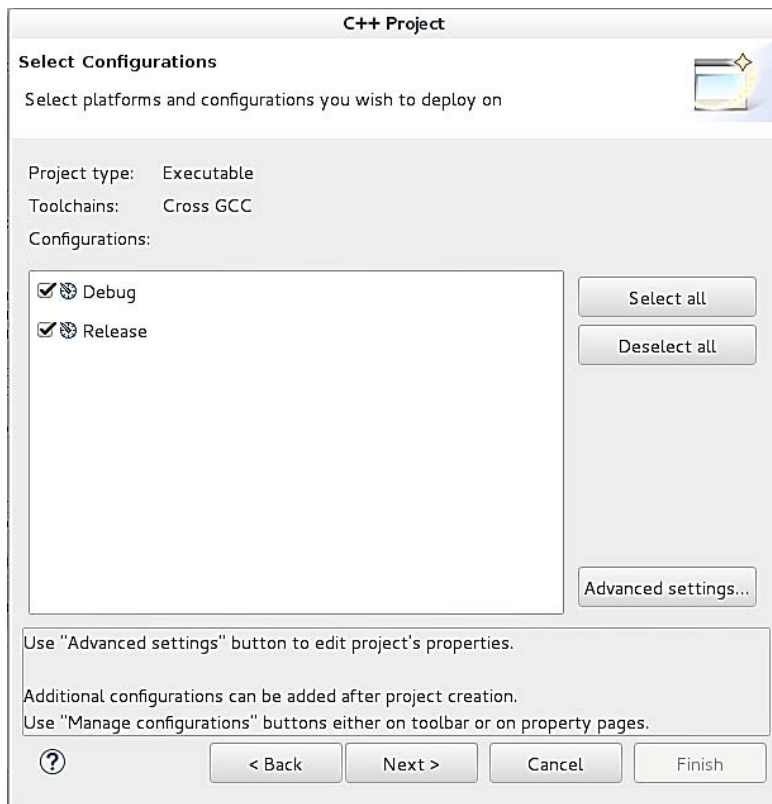


Figura 4-9. Selección de Configuración.

Por último nos aparecerá una ventana en la cual debemos ingresar los comandos Cross Gcc para poder compilar código destinado para nuestra tarjeta de desarrollo. En la sección donde dice “Cross compiler prefix” escribiremos: `arm-linux-gnueabihf-`. Mientras que en la sección donde dice “Cross compiler path” escribiremos: `/usr/bin`. Tal como se muestra en la figura 4-9.

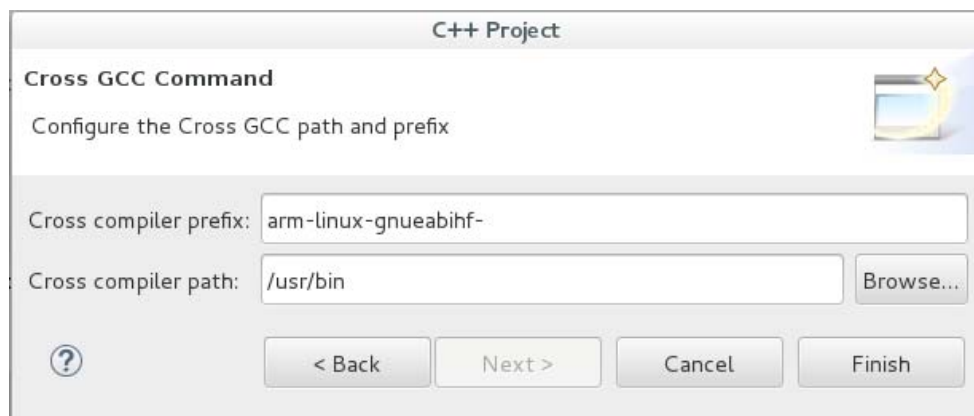
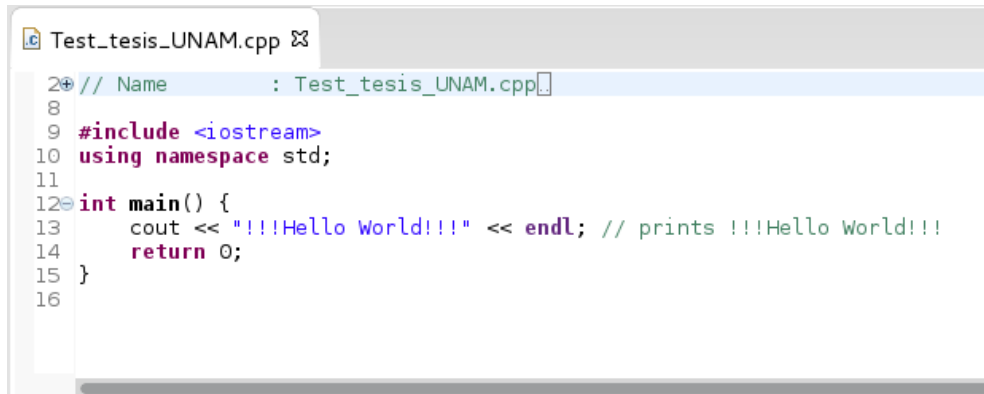


Figura 4-10. Comandos Cross Gcc.

Al dar clic en el botón finalizar (Finish), podremos observar una ventana con un código como el de la figura 4-11. Este código saldrá automáticamente cada que abramos un nuevo proyecto. Ya se había mencionado que cada vez que se abra un nuevo proyecto saldrá este programa, por lo tanto, para editar nuestro propio código debemos borrar el contenido del “int main()” y escribir nuestras líneas de código.



```
Test_tesis_UNAM.cpp
2 // Name      : Test_tesis_UNAM.cpp
8
9 #include <iostream>
10 using namespace std;
11
12 int main() {
13     cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!
14     return 0;
15 }
16
```

Figura 4-11. Ventana para editar nuestros códigos en C/C++.

Para compilar nuestro programa primero tenemos que dar clic en el botón “Build all” el cual se puede observar en la figura 4-12. Posteriormente procederemos a correr nuestro programa dando clic en el botón “Run” el cual también se indica en la figura 4-12.



Figura 4-12. Íconos “Build all y Run”.

En la parte inferior de la ventana principal de Eclipse podemos observar una pestaña que dice “Console” en la cual podemos ver nuestros programas ejecutándose, tal como se puede apreciar en la figura 4-13.

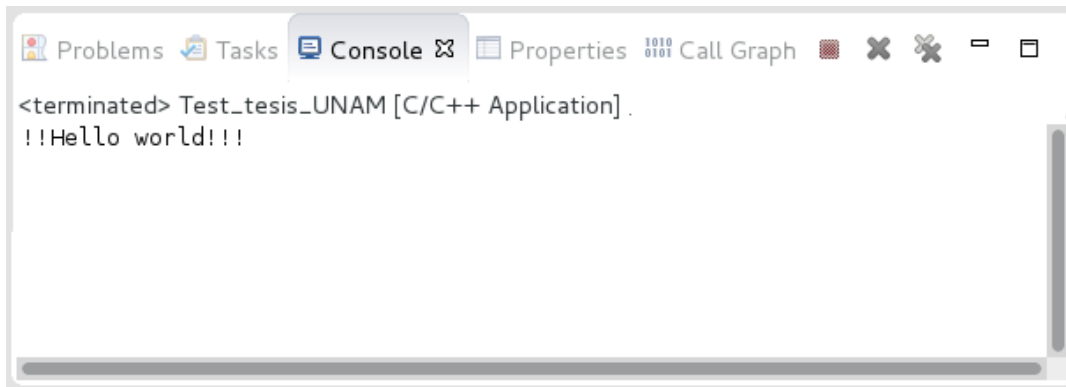


Figura 4-13. Consola de Eclipse.

En la parte lateral izquierda de la ventana principal del IDE de Eclipse podemos observar un apartado como el de la figura 4-14, en el cual se pueden visualizar los archivos que se generan al compilar nuestro código, entre estos archivos está el ejecutable, el cual podemos ver dentro del submenú que se desplaza al dar clic en “Binaries” y además podemos ejecutar en nuestra computadora o en nuestra tarjeta de desarrollo.

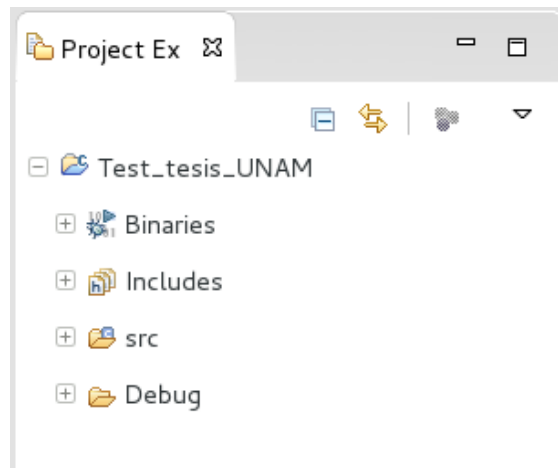


Figura 4-14. Ventana de exploración de archivos.

4.2 Sistema de Exploración Remoto.

El sistema de exploración remoto o RSE (Remote System Explorer) es un complemento de Eclipse que sirve para conectarnos a una computadora remota en una Red con la finalidad de probar los programas compilados desde el IDE de Eclipse sin tener acceso físico a la computadora sobre la que queremos trabajar. Con el RSE podemos explorar entre las diferentes carpetas y archivos, compilar código, ejecutar código e incluso tener acceso a la terminal de un sistema Linux. Este elemento es ideal para implementarlo en el uso de nuestra tarjeta de desarrollo ya que podemos hacer compilación cruzada, enviarlo al Beaglebone y ejecutarlo, todo esto sin la necesidad de salir del IDE de Eclipse.

Como ya se había mencionado anteriormente, todos los elementos de Eclipse deben descargarse e instalarse antes de poder ser utilizados por el usuario, por tal razón, en esta sección explicaremos a detalle como instalar el RSE.

Instalación del RSE.

Lo primero que tenemos que hacer es abrir el IDE de Eclipse, ubicarnos en la barra de menú y dar clic en la pestaña que dice “Help”, cuando se desplace el menú de esta opción buscaremos la opción que dice “Install New Software”. Al abrir la opción “Install new software” nos desplazará una ventana como la que se muestra en la figura 4-15.

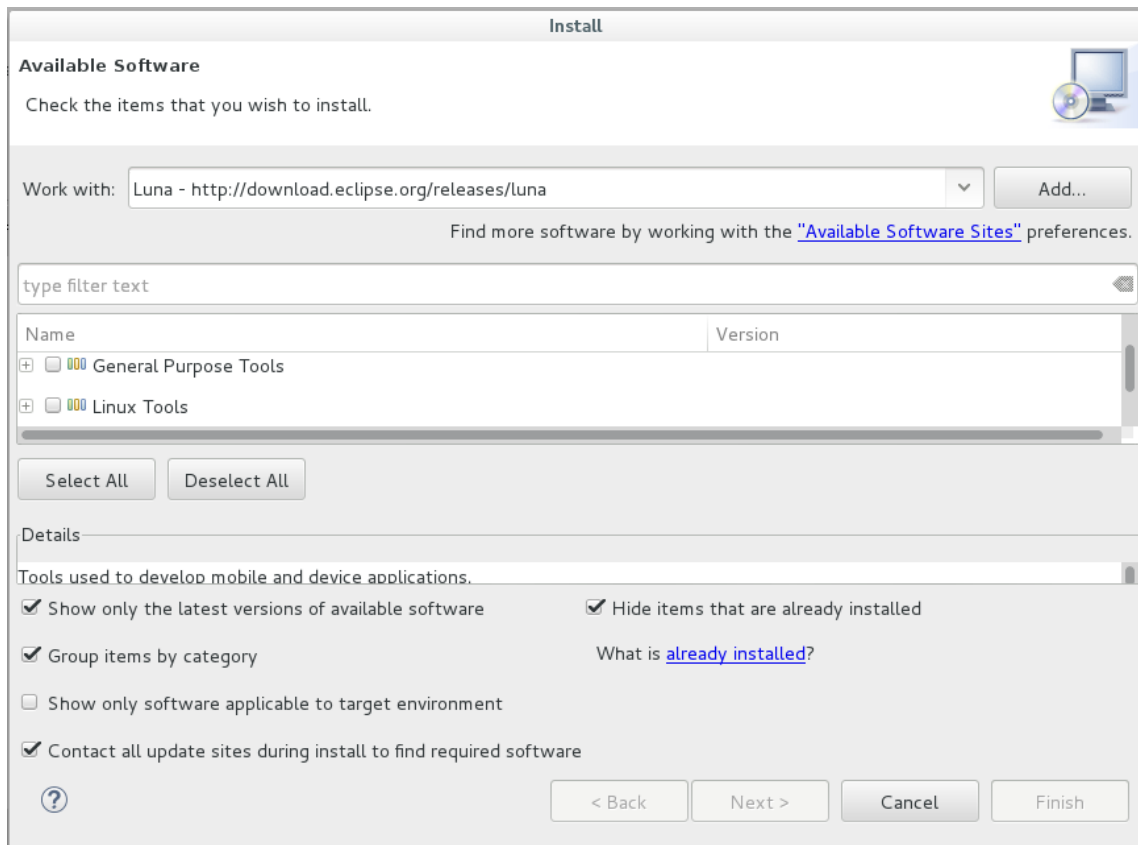


Figura 4-15. Ventana de instalación de complementos.

En la sección donde dice “Work with” debemos seleccionar la siguiente opción:

Luna - <http://download.eclipse.org/releases/luna>

En el recuadro blanco donde dice “Name” nos desplazaremos hacia abajo hasta encontrar la opción que dice “General Porpouse Tools”. Desgraciadamente el recuadro blanco de la sección mencionada anteriormente es muy pequeño y se dificulta un poco el buscar la opción que deseamos, por esta razón es recomendable bajar lentamente con las flechas de dirección del teclado en lugar de utilizar el mouse.

Una vez que hayamos encontrado la opción “General Porpouse Tools”, procederemos a abrir el submenú de esta opción para buscar la opción “Remote system explorer User Actions”, seleccionarla, tal como se observa en la figura 4-15.

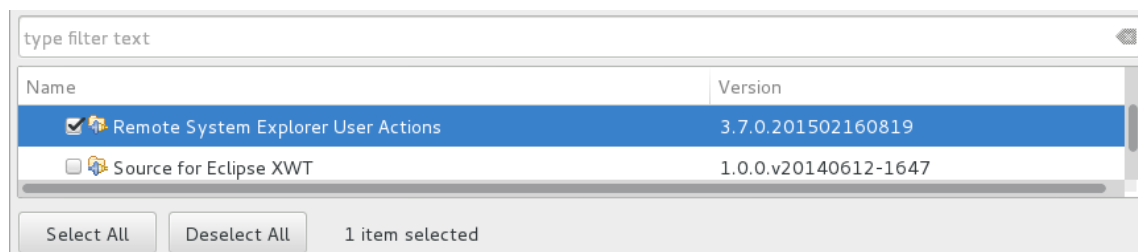


Figura 4-16. Remote System Explorer.

Al concluir el paso anterior y dar clic en “Next”, nos aparecerá un recuadro donde nos muestra todos los complementos que seleccionamos para instalar, en este caso sólo seleccionamos el paquete de sistema de exploración remoto, por lo tanto, daremos clic en el botón “Next”. Finalmente nos aparecerá otra ventana como la de la figura 4-17, donde tendremos que aceptar los términos de licencia para poder instalar nuestro complemento seleccionado, al dar clic en el botón finalizar (Finish) nos pedirá reiniciar Eclipse, al reiniciarse el programa ya tendremos instalado el complemento y estará listo para ser usado.

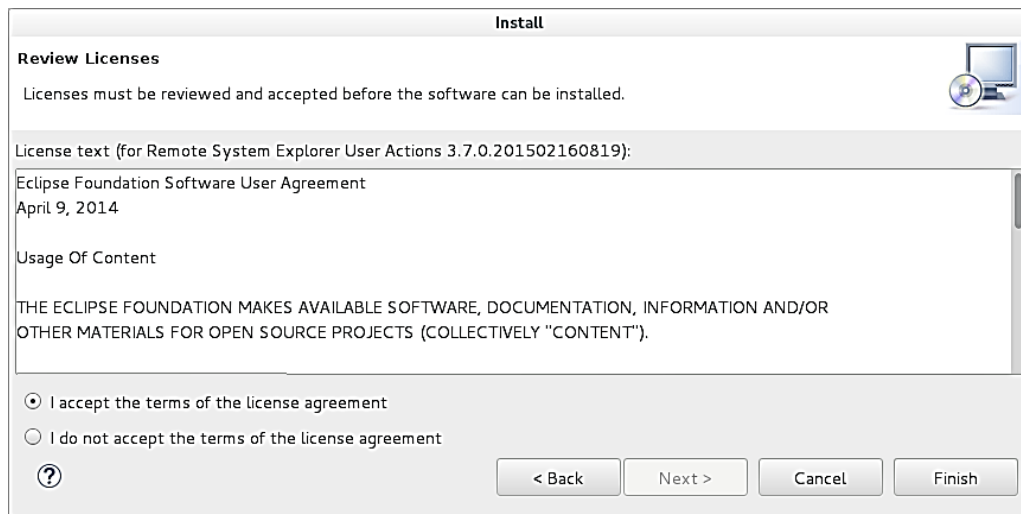


Figura 4-17. Términos de licencia.

Configuración del RSE.

Para que el panel principal del sistema de exploración remoto aparezca en una de las pestañas del explorador de archivos de la figura 4-14 debemos activar dicha pestaña. Para activar la pestaña de opciones del RSE debemos ubicarnos en la barra de menús, dar clic en el menú “Window”, buscar la opción que dice “show view” y por último dar clic en la opción que dice “other”, para que aparezca una ventana como la de la figura 4-18, en la cual sólo debemos seleccionar la carpeta “Remote systems”, posteriormente seleccionar la opción que también lleva el nombre de “Remote Systems” y dar clic en aceptar (OK). En la figura 4-19 podemos observar el panel principal del RSE, el cual se encuentra en una de las pestañas que tiene el aparatado que se encuentra a la izquierda de la ventana principal donde se editan los códigos.

Para tener comunicación con nuestra tarjeta de desarrollo debemos tener acceso ya sea vía USB o mediante una red LAN, posteriormente crear una conexión dando clic en el ícono que está marcado con un círculo rojo en la figura 4-19. Nos aparecerá una ventana como la de la figura 4.20, en la cual debemos seleccionar el tipo de dispositivo con el cual nos queremos conectar. Como se puede ver en la figura 4-20, podemos efectuar comunicación con sistemas Windows, Linux, conexiones SSH y telnet²². Para este caso seleccionaremos la opción “Linux” ya que con esta opción además de poder tener una conexión SSH nos permite navegar entre los archivos y

²² Telnet es un protocolo de red que nos permite tener control y acceso remoto a una computadora, al igual que los protocolos que hemos visto en este trabajo.

carpetas de nuestra tarjeta de desarrollo, lo que significa que también incorpora un protocolo de escritorio remoto.

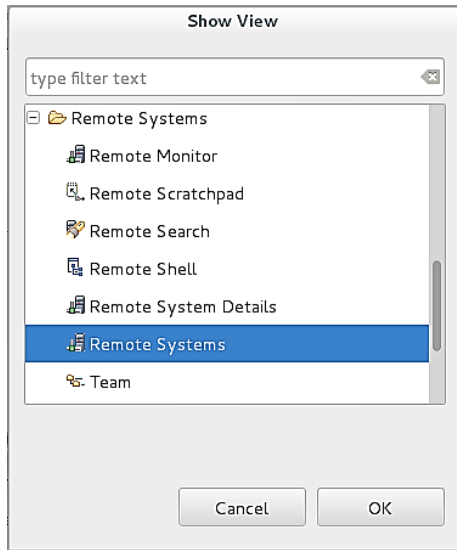


Figura 4-18. Vista de Activación del RSE.

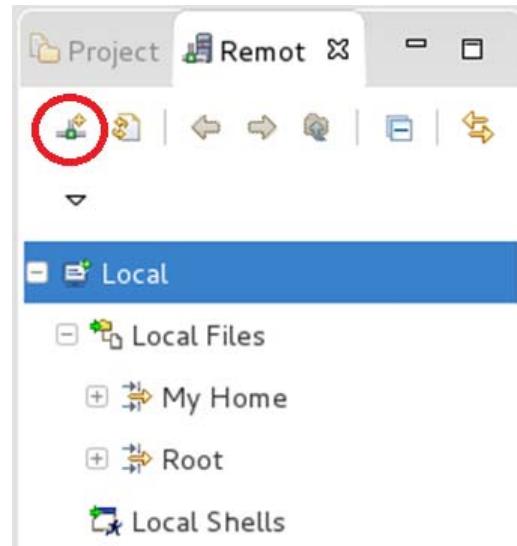


Figura 4-19. Pestaña del RSE.

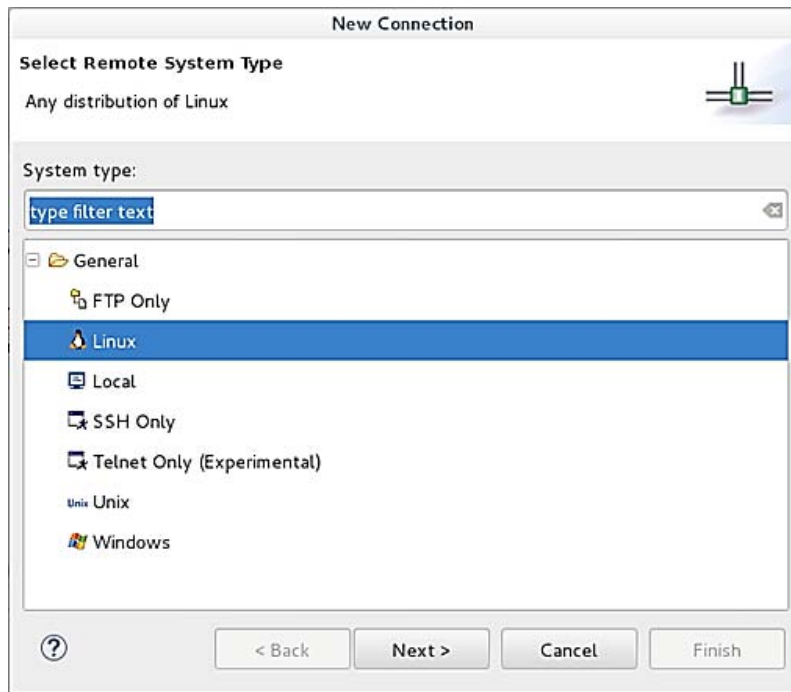


Figura 4-20. Configuración de nueva conexión.

Al dar clic en “Next” aparecerá una sección de configuración como la de la figura 4-21. En esta sección debemos escribir el IP de nuestra tarjeta de desarrollo, en el apartado llamado “Host Name”. Cabe destacar que debemos escribir el IP correspondiente al tipo de conexión que deseamos efectuar, por ejemplo, si queremos tener acceso vía USB debemos usar el IP 192.168.7.2, o si queremos tener acceso desde una red LAN podemos usar un IP dinámico o estático, para este trabajo se usó el IP estático 192.168.1.111. En el apartado “Connection Name ”

podemos darle un nombre a la conexión mientras que en el apartado “Description” podemos escribir notas referentes a la conexión.

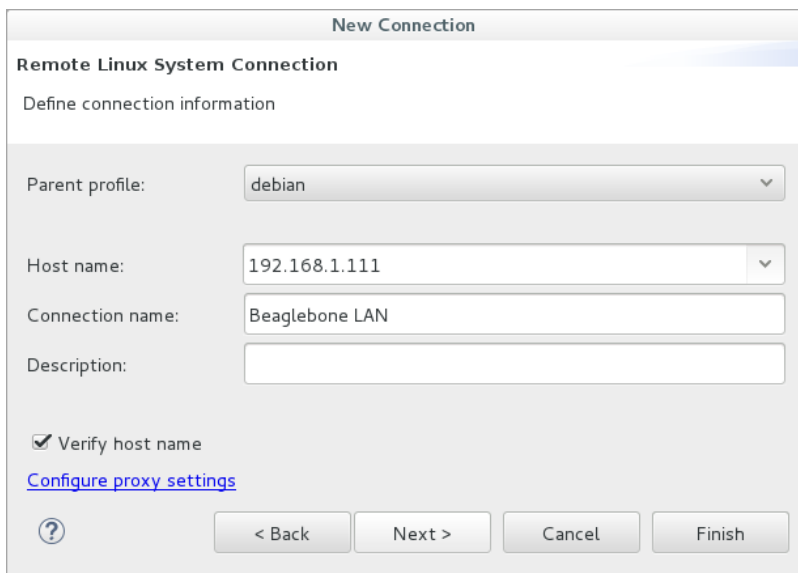


Figura 4-21. Configuración del IP.

En la figura 4-22 podemos observar las diferentes opciones de configuración que debemos seleccionar en un recuadro llamado “Configuration” cada que demos clic en el botón “next” para las cuatro secciones de configuración restantes. Existen cuatro secciones de configuración llamadas: Files, Processes, Shells y Ssh terminals, las cuales van apareciendo en el orden alfabético de la figura 4-22.

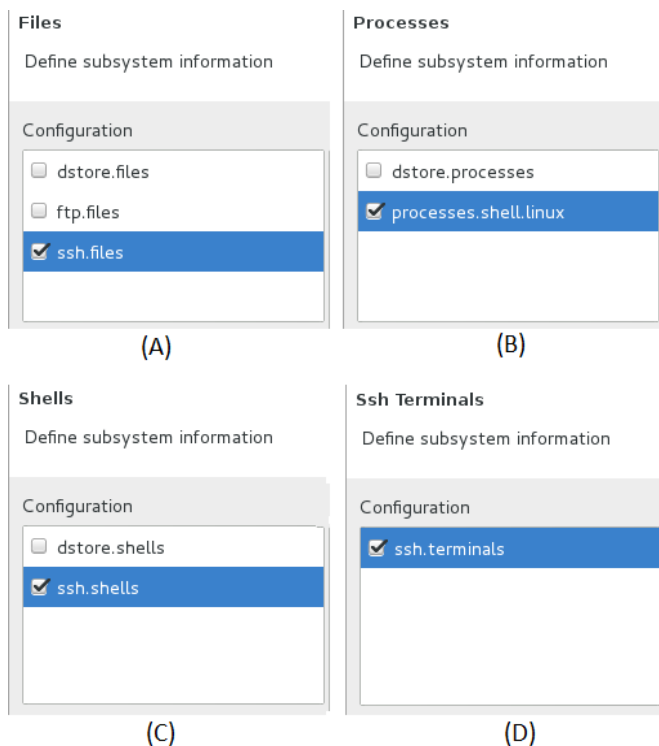


Figura 4-22. Opciones de Configuración

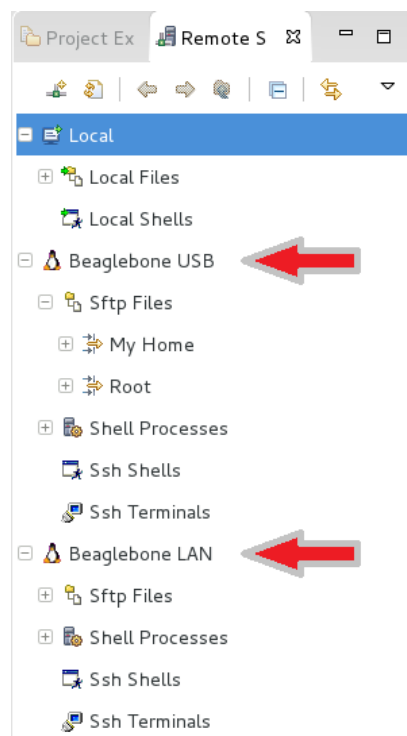


Figura 4-23. Configuraciones de RSE.

Cuando lleguemos a la última sección de opciones de configuración simplemente debemos dar clic en finalizar para que tengamos lista la configuración de conexión con nuestra tarjeta de desarrollo. Cabe mencionar que podemos tener varias configuraciones con diferentes computadoras, para este trabajo se ha configurado la conexión con nuestra tarjeta de desarrollo ya sea por USB o por una red LAN como se observa en la figura 4-23.

Copiar un programa y ejecutarlo en nuestra tarjeta de desarrollo.

Una vez que tengamos ya configurada la conexión de Eclipse con nuestra tarjeta de desarrollo podremos copiar archivos ejecutables, copiar códigos, explorar entre los archivos y carpetas de nuestra tarjeta, y además tener acceso a la terminal. En esta sección se explicará cómo copiar archivos, correr archivos ejecutables y tener acceso a la terminal del Beaglebone.

Lo primero que debemos hacer es abrir el IDE de Eclipse y conectarnos con nuestra tarjeta de desarrollo dando clic derecho sobre el icono de la conexión que llevaba el nombre que le hemos dado a nuestra conexión, para que posteriormente salga una lista de opciones como la de la figura 4-24, en la cual daremos clic en donde dice “Connect”.

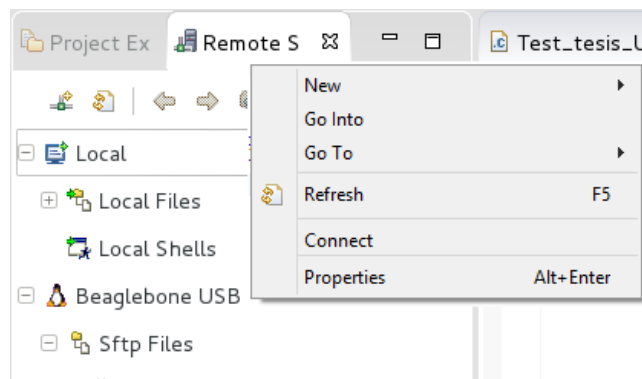


Figura 4-24. Conexión con Beaglebone.

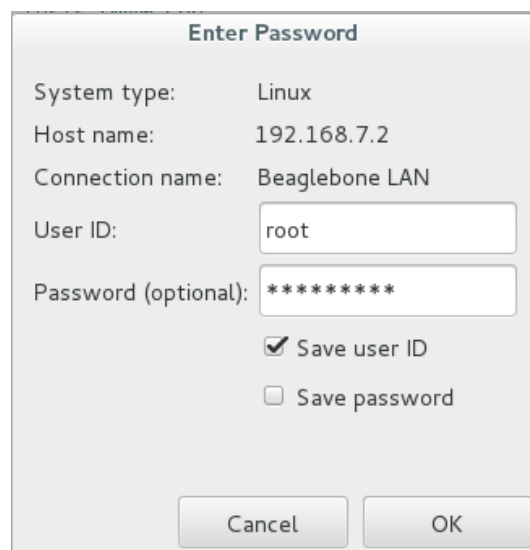


Figura 4-25. Autenticación de usuario.

Después de haber dado clic en “Connect” nos aparecerá una ventana como la de la figura 4-25 en la cual debemos ingresar el nombre de usuario y la contraseña de acceso. En la sección que dice “User ID” debemos escribir el usuario, en este caso ingresamos como root ya que es el único usuario de nuestra tarjeta, mientras que en la sección donde dice “Password” ingresaremos la contraseña de acceso del root²³ o dejaremos el espacio en blanco si no le hemos asignado ninguna contraseña al root. Al dar clic en OK nos aparecerá una ventana como la de la figura 4-26, en la cual debemos dar clic nuevamente en el botón OK para terminar de establecer la conexión.

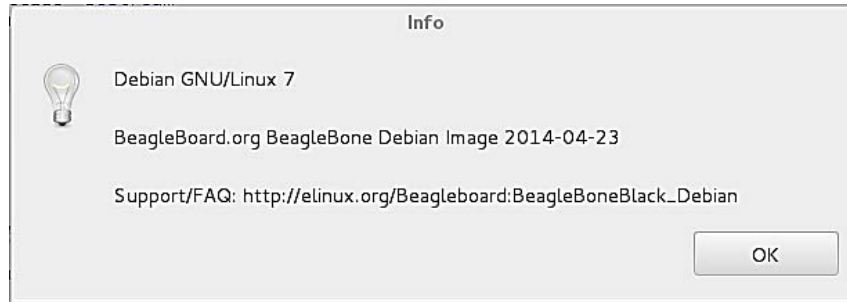


Figura 4-26. Confirmación de conexión con dispositivo remoto.

Una vez que hayamos establecido la conexión con nuestra tarjeta, podemos navegar entre los archivos y carpetas del Beaglebone para copiar o eliminar archivos, abrir la terminal y correr todo tipo de programas. Para ejemplificar cómo copiar a nuestra tarjeta un programa y correrlo, utilizaremos el programa que generamos en la sección 4.1 y cuyo código podemos observar en la figura 4-11.

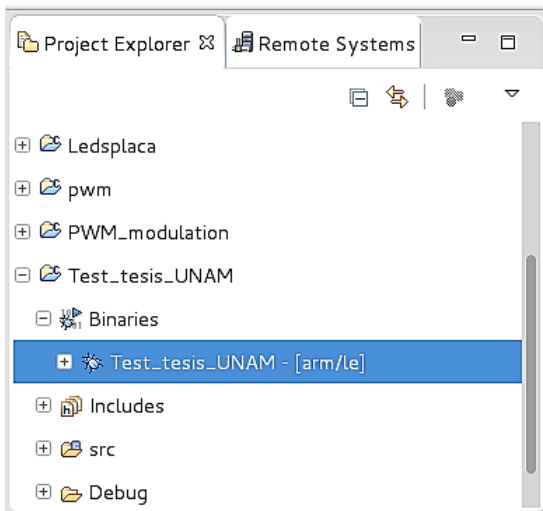


Figura 4-27. Project Explorer.

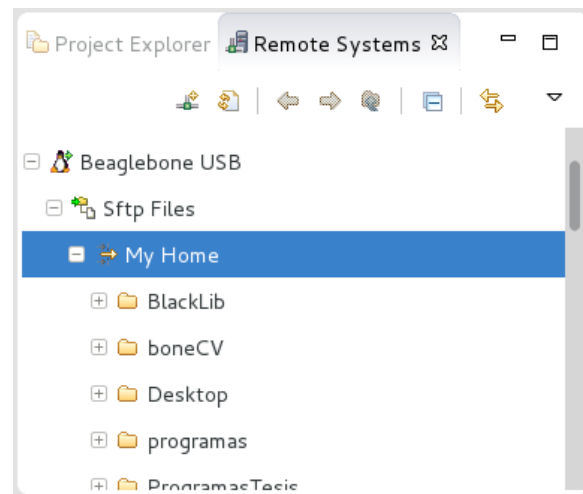


Figura 4-28. Remote Systems.

Copiar un programa a nuestra tarjeta de desarrollo resulta muy fácil con ayuda del RSE, basta con ir al panel que se encuentra a la izquierda del IDE de Eclipse, dar clic en la pestaña que dice “Project Explorer”, después abrir el submenú de la carpeta que contiene nuestro programa y después abrir otro submenú que dice “Binaries”. Por último seleccionar el archivo ejecutable que

²³ Podemos asignar o cambiar la contraseña del Root con el comando número 25 del anexo 2.

lleva el mismo nombre que le dimos al código y presionar Ctrl+C para copiar, tal como se puede observar en la figura 4-27.

Para pegar el archivo ejecutable en nuestra tarjeta de desarrollo debemos dar clic en la pestaña del RSE, abrir el submenú que dice “Sftp Files”, después abriremos el submenú llamado “My home”, ahora seleccionaremos la carpeta en la que queremos pegar el ejecutable y presionar Ctrl+V. En la figura 4-28 podemos observar algunas de las carpetas y archivos que contiene el Beaglebone utilizado para este trabajo, en los cuales podemos pegar cualquier tipo de archivo copiado de la pestaña llamada “Project Explorer”. Para este caso se copió el programa directamente en el directorio llamado “My home”, por lo tanto se seleccionó dicho directorio como se observa en la figura 4.28, posteriormente presionamos Ctrl+V para pegar el archivo ejecutable.

Una vez que tengamos el programa copiado en nuestra tarjeta de desarrollo procederemos a abrir la terminal para poder ejecutar nuestro programa. Para abrir la terminal basta con ir a la pestaña del RSE y abrir el submenú llamado “Shell Processes”, después daremos clic derecho sobre la opción que dice “Ssh Terminals” y seleccionaremos la opción “Launch terminal” que aparece en el menú que se observa en la figura 4-29.

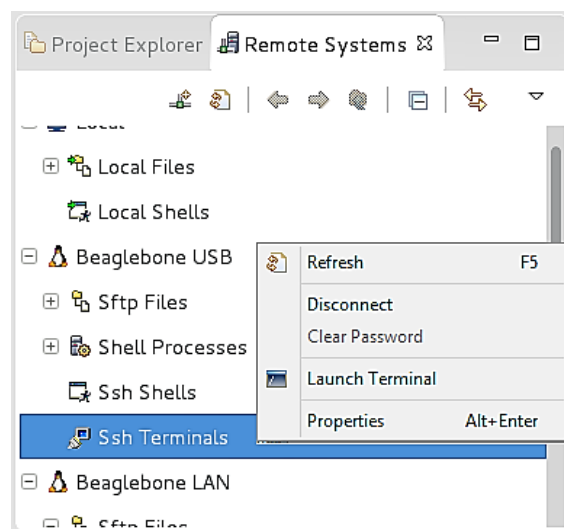


Figura 4-29. Conexión SSH Dentro de Beaglebone.

La pestaña de la terminal aparecerá en la misma ventana donde se encuentra la pestaña de la consola, como se observa en la figura 4-30.

Para poder correr nuestro programa primero tenemos que modificar los permisos de acceso de nuestro archivo ejecutable ya que al compilar un programa en Eclipse sólo le otorga permisos de ejecución a la computadora con la que fue compilado algún archivo ejecutable, por lo tanto, si copiamos un archivo a otra computadora no será posible ejecutarlo a menos que tenga permisos de ejecución. En la primera línea de la terminal de la figura 4-30 podemos observar que ingresamos el comando para ejecutar el programa que hemos copiado (para correr un programa en la terminal debemos consultar el comando número 17 del anexo 2 o ingresar la primer línea de comando de la figura 4-29), sin embargo, como no tenemos permisos de ejecución nos apareció el aviso que se puede visualizar en la segunda línea de la terminal de la figura 4-30. Para cambiar los

permisos de ejecución para que cualquier computadora pueda abrir nuestro archivo ejecutable, debemos consultar el comando número 26 del anexo 2 o escribir la tercer línea de comando de la figura 4.30. Después de haber ingresado el comando para cambiar los permisos de ejecución procederemos a correr nuestro archivo ejecutable, al correr nuestro archivo ejecutable aparecerá el letrero “!! Hello world!!” como se observa en la figura 4.30.

```

Problems Tasks Console Properties Call Graph Terminals
Beaglebone USB
root@beaglebone:~# ./Test_tesis_UNAM
-bash: ./Test_tesis_UNAM: Permission denied
root@beaglebone:~# chmod a+x Test_tesis_UNAM
root@beaglebone:~# ./Test_tesis_UNAM
!!Hello world!!!
root@beaglebone:~#
  
```

Figura 4-30. Terminal de comandos en Eclipse.

4.3 Programación de los GPIO.

Las GPIO (General Purpose Input/Output) son las entradas y salidas que el usuario puede manipular desde la terminal o desde un programa y que no tienen ningún propósito definido. Como ya se mencionó en el capítulo 2, podemos disponer de un máximo de 65 GPIO si se desactivan algunos buses, sin embargo, en este trabajo no será necesario desactivar ningún bus para el uso de las GPIO, ya que nosotros sólo utilizaremos las GPIO que no están multiplexadas y que se pueden disponer de ellas de manera rápida. En cuanto a la activación y desactivación de elementos y buses, se verá más adelante conforme vayamos usando los diferentes buses de nuestra tarjeta. En la siguiente figura se muestran en un recuadro rojo las GPIO que no están multiplexadas o a las cuales podemos tener acceso sin desactivar ningún otro bus.

DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BUTTON	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	GPIO_40	GPIO_23	13	14	GPIO_26
GPIO_48	15	16	GPIO_51	GPIO_47	15	16	GPIO_46
GPIO_4	17	18	GPIO_5	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	GPIO_22	19	20	GPIO_63
GPIO_3	21	22	GPIO_2	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	GPIO_15	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	GPIO_14	GPIO_32	25	26	GPIO_61
GPIO_125	27	28	GPIO_123	GPIO_86	27	28	GPIO_88
GPIO_121	29	30	GPIO_122	GPIO_87	29	30	GPIO_89
GPIO_120	31	32	VDD_ADC	GPIO_10	31	32	GPIO_11
AIN4	33	34	GND_ADC	GPIO_9	33	34	GPIO_81
AIN6	35	36	AIN5	GPIO_8	35	36	GPIO_80
AIN2	37	38	AIN3	GPIO_78	37	38	GPIO_79
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	GPIO_7	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	GPIO_70	45	46	GPIO_71

Figura 4-31. GPIO no multiplexadas.

Control del GPIO 49 mediante la terminal de comandos.

Todas las GPIO pueden ser manipuladas desde la terminal de comandos o desde cualquier programa hecho en C/C++ o algún otro lenguaje de programación. Para este ejemplo utilizaremos el GPIO49²⁴ ya que, por no estar multiplexado, podemos explicar el procedimiento de uso de manera rápida y concreta.

Antes de empezar con el procedimiento para controlar un GPIO desde la terminal, lo primero que debemos hacer es armar el circuito mostrado en la figura siguiente.

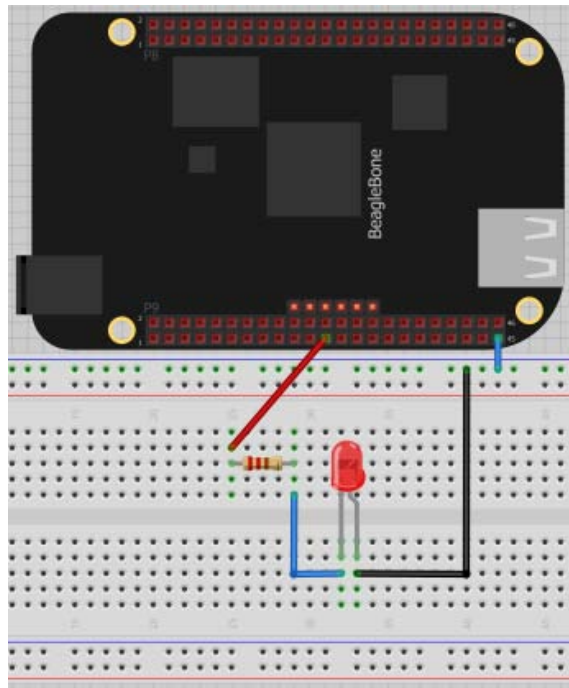


Figura 4-32. Conexión de un LED al Beaglebone²⁵.

Todos los comandos usados en esta sección se encuentran enumerados en las líneas de la figura 4-33, por tal razón, en lo que resta de esta sección solo se hará mención al número de comando.

Los archivos que se encargan de administrar el estado de las GPIO se encuentran en la ubicación `/sys/class/gpio/`. Por lo tanto, para poder utilizar algún GPIO lo primero que debemos hacer es abrir la terminal y movernos a la ubicación mencionada anteriormente con el comando 1 (ver figura 4-33).

Para poder observar el contenido de la ubicación mencionada anteriormente utilizaremos el comando número 2. Al ingresar el comando de la línea número 2 podemos ver unos archivos llamados “export”, “gpiochip0”, “gpiochip32”, “gpiochip64”, “gpiochip96” y “unexport”, los cuales son los encargados de controlar el estado de activación y desactivación de cada GPIO.

Para poder activar el GPIO 49 ingresaremos el comando número 3, posteriormente ingresaremos el comando de la línea número 4 para visualizar nuevamente los archivos que existen en la carpeta

²⁴ Podemos consultar la distribución y el número de los pines de nuestra tarjeta de desarrollo en el anexo 1

²⁵ La resistencia utilizada en el ánodo del des debe ser de 220Ω.

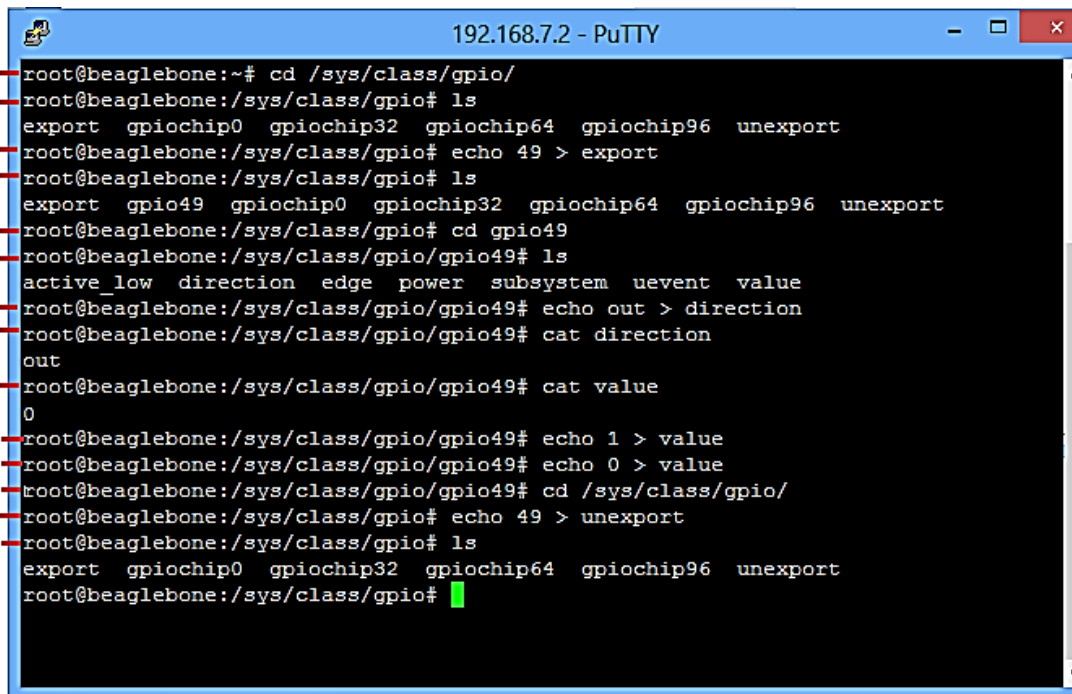
donde estamos ubicados. Al ingresar el comando de la línea número 4 observamos una vez más los archivos que estaban anteriormente y un nuevo archivo llamado “gpio49”, lo cual nos indica que el GPIO 49 está listo para ser utilizado.

El archivo llamado “gpio49” en realidad es una carpeta que se crea al activar el GPIO49. Dicha carpeta contiene los archivos necesarios para poder controlar el estado del pin que hemos activado. Para poder movernos a la carpeta del GPIO que hemos activado y visualizar los archivos que hay en dicha carpeta, debemos ingresar los comandos de las líneas 5 y 6 respectivamente. Los nombres de los archivos de la carpeta “gpio49” se pueden visualizar debajo del comando de la línea número 6 de la figura 4-32.

Dentro de la carpeta “gpio49” existe un archivo llamado “Direction”, el cual sirve para definir la dirección del GPIO que hemos activado, para este caso utilizaremos nuestro GPIO como una salida, por tal razón debemos ingresar el comando de la línea número 7. Con el comando de la línea número 8 podemos observar la dirección de nuestro GPIO mientras que con el comando de la línea 9 podemos observar si nuestro GPIO se encuentra en estado alto o bajo.

Con el comando de la línea 10 podremos cambiar nuestra GPIO a estado alto para que encienda nuestro led mientras que con el comando 11 podemos poner nuevamente nuestra GPIO en estado bajo.

Para desactivar nuestro GPIO, primero debemos regresar a la ubicación /sys/class/gpio/, para esto utilizaremos el comando número 12. Posteriormente ingresaremos el comando de la línea 13, el cual se encarga de desactivar el GPIO que activamos y eliminar la carpeta llamada “gpio49”. Para verificar que dicha carpeta fue eliminada utilizaremos el comando de la línea 14. Cabe mencionar que este procedimiento se puede hacer para cada GPIO remarcado en la figura 4-31 simplemente sustituyendo el número de GPIO en los comandos que aparecen en la figura 4-33.



```
192.168.7.2 - PuTTY
1 root@beaglebone:~# cd /sys/class/gpio/
2 root@beaglebone:/sys/class/gpio# ls
export gpiochip0 gpiochip32 gpiochip64 gpiochip96 unexport
3 root@beaglebone:/sys/class/gpio# echo 49 > export
4 root@beaglebone:/sys/class/gpio# ls
export gpio49 gpiochip0 gpiochip32 gpiochip64 gpiochip96 unexport
5 root@beaglebone:/sys/class/gpio# cd gpio49
6 root@beaglebone:/sys/class/gpio/gpio49# ls
active_low direction edge power subsystem uevent value
7 root@beaglebone:/sys/class/gpio/gpio49# echo out > direction
8 root@beaglebone:/sys/class/gpio/gpio49# cat direction
out
9 root@beaglebone:/sys/class/gpio/gpio49# cat value
0
10 root@beaglebone:/sys/class/gpio/gpio49# echo 1 > value
11 root@beaglebone:/sys/class/gpio/gpio49# echo 0 > value
12 root@beaglebone:/sys/class/gpio/gpio49# cd /sys/class/gpio/
13 root@beaglebone:/sys/class/gpio# echo 49 > unexport
14 root@beaglebone:/sys/class/gpio# ls
export gpiochip0 gpiochip32 gpiochip64 gpiochip96 unexport
root@beaglebone:/sys/class/gpio#
```

Figura 4-33. Comandos para encender un led controlando un GPIO.

Control del GPIO 49 empleando C++.

C++ es uno de los lenguajes de programación más populares ya que cuenta con una gran cantidad de elementos que permiten a los programadores desarrollar código de manera dinámica empleando los diferentes paradigmas de la programación²⁶ que existen hoy en día. En este trabajo se ha empleado C++ como lenguaje de programación ya que hay poca información y ejemplos de cómo emplear los buses de nuestra tarjeta de desarrollo con dicho lenguaje.

Hasta este punto ya debemos tener guardado en nuestro Beaglebone todos los códigos que serán empleados en este trabajo, tal como se describió en el capítulo 2 en la sección llamada GITHUB. Otra opción que podemos emplear en caso de que nuestra tarjeta de desarrollo no tenga acceso a internet es crear un proyecto en Eclipse con el código que se encuentra en el apéndice 1, copiarlo a nuestro Beaglebone y ejecutarlo tal y como se ha explicado anteriormente en este capítulo.

Para entender el funcionamiento del programa empleado en esta sección se ha implementado el diagrama de flujo de la figura 4-34.

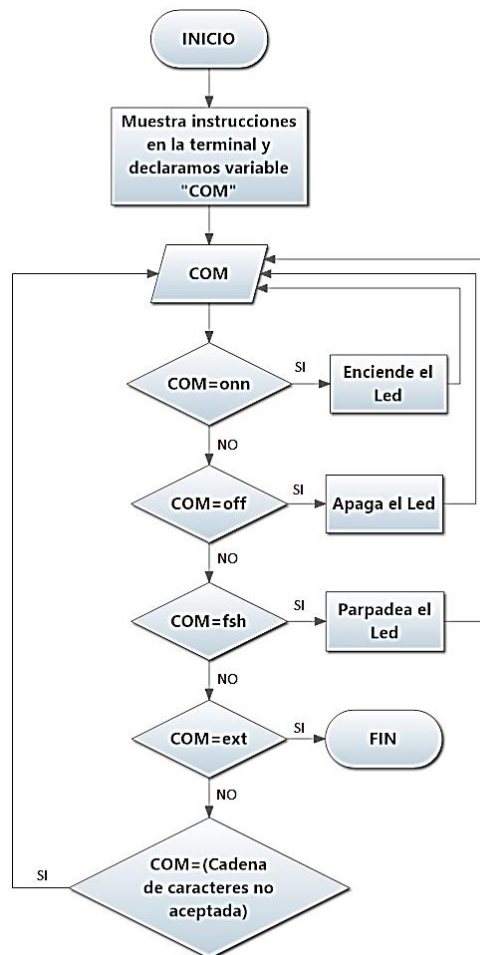


Figura 4-34. Diagrama de Flujo del programa para encender un led.


²⁶ Un paradigma de programación representa un enfoque particular o una filosofía para diseñar soluciones de programación [19].

Como se observa en la figura anterior, tenemos una variable llamada "COM", la cual será comparada con cuatro diferentes cadenas de caracteres para poder ejecutar una acción, podemos apagar el led, encenderlo, hacerlo parpadear o salir del programa. Cabe mencionar el programa es capaz de detectar y avisar al usuario cuando se ha ingresado una cadena de caracteres no válida para ejecutar una acción. Antes de iniciar con la ejecución de este programa debemos tener armado el circuito de la figura 4-32.

Para poder ejecutar nuestro programa debemos cambiarnos a la carpeta que contiene el ejecutable de nuestro programa ingresando en la terminal el comando de la línea número 1 de la figura siguiente. Cabe mencionar que la carpeta llamada "Tesis" y todas sus subcarpetas fueron descargadas del repositorio de "Github" creado para este trabajo.

Para poder observar el contenido de la carpeta a la cual nos hemos trasladado ingresaremos el comando de la línea 2 de la siguiente figura. Como podemos ver debajo de la línea número dos de la siguiente figura, hay dos archivos llamados "Blink_without_blacklib" y "Blink_without_blacklib.cpp", los cuales son el archivo ejecutable y el archivo que contiene el código, respectivamente. La razón de que lleven el nombre mencionado anteriormente es porque este programa fue codificado sin ayuda de ninguna librería, ya que más adelante utilizaremos una librería llamada BlackLib.

Como ya se ha mencionado anteriormente, antes de ejecutar un programa debemos cambiar los permisos de acceso del archivo ejecutable. Para poder cambiar los permisos de nuestro archivo ejecutable debemos ingresar el comando de la línea 3 de la siguiente figura. Por último, debemos ingresar el comando de la línea número 4 para correr nuestro programa.



```
1 root@beaglebone:~# cd Tesis/Capitulo_4/Blink_1/
2 root@beaglebone:~/Tesis/Capitulo_4/Blink_1# ls
  Blink_without_blacklib  Blink_without_blacklib.cpp
3 root@beaglebone:~/Tesis/Capitulo_4/Blink_1# chmod a+x Blink_without_blacklib
4 root@beaglebone:~/Tesis/Capitulo_4/Blink_1# ./Blink_without_blacklib

Encender un led Mediante cadenas de caracteres.

oon : enciende el led.
off  : apaga el led.
fsh  : parpadea el led seis veces.
ext  : salir del programa.

5 ingrese el comando: oon
6 ingrese el comando: off
7 ingrese el comando: fsh
8 ingrese el comando: ext
9 Fin del programa.
10 root@beaglebone:~/Tesis/Capitulo_4/Blink_1#
```

Figura 4-35. Programa para encender un LED.

Al abrir nuestro archivo ejecutable podemos ver las instrucciones de nuestro programa, tal como se observa en la figura 4-34. En las líneas 5,6, 7 y 8 podemos observar que se ha ingresado cada

una de las diferentes opciones que tiene el programa mientras que en la línea número 9 nos aparece el aviso de que nos hemos salido del programa. En la línea número 10 podemos observar que la terminal regresa a su estado normal después de finalizar el programa.

4.4 Usando BlackLib.

BlackLib es una librería de programación construida para C++ y diseñada para controlar los principales buses del Beaglebone. Con ella podemos controlar las GPIO, los convertidores ADC (Analog to Digital Converter), el puerto UART (Universal Asynchronous Receiver-Transmitter), el puerto I2C (Inter-Integrated Circuit) y las salidas PWM (pulse-width modulation). Esta librería es de gran ayuda ya que en muchos casos el tener acceso y control de algunos buses suele ser muy complicado ya que se requieren demasiadas líneas de código para poder desarrollar un programa.

En la página oficial de BlackLib²⁷ podemos descargar cualquiera de las tres versiones de esta librería que se han creado hasta este momento, sin embargo, para este trabajo se ha utilizado la versión 2.0 ya que la versión 3.0 no fue compatible con el IDE de Eclipse.

En la página oficial de BlackLib podemos encontrar ejemplos de uso de cada una de las estructuras con las que cuenta cada una de las versiones de esta librería, sin embargo, los ejemplos no vienen representados con un código listo para ser compilado ya que sólo hacen referencia a la aplicación de la sintaxis de cada estructura que contiene esta librería.

En este trabajo no se usaron todas las estructuras que tiene la librería porque lo que se pretende es mostrar cómo podemos emplear las principales estructuras para controlar de manera satisfactoria los principales buses de nuestra tarjeta de desarrollo. Si se desea tener un conocimiento más profundo sobre la librería podemos encontrar toda la información necesaria en su página oficial.

Otro aspecto importante es que para este trabajo se ha modificado la librería borrando algunos códigos de ejemplo que presentan errores a la hora de ser compilados, ya que esos ejemplos, a diferencia de toda la librería en general, no fueron compilados bajo un estándar de C++ llamado ISO C++ 11.

ISO C++ 11 es un estándar de C++ creado por ISO (Internacional Organization and Standardization) con la finalidad de eliminar algunas diferencias de código que se han presentado en compiladores, para que los códigos sean compatibles en diferentes IDEs o compiladores que cuenten con la opción de compilación ISO C++ 11. C++ 11 incluye varias mejoras al núcleo del lenguaje ya que se ha extendido la biblioteca estándar de C++, además se ha optimizado para la interacción con el hardware reduciendo los tiempos de ejecución de código. Eclipse cuenta con la opción de poder compilar código empleando el estándar C++ 11, lo que significa que los programas que sean desarrollados con Eclipse bajo este estándar serán compatibles con otros IDEs capaces de emplear este estándar para compilar código. Otros ejemplos de IDEs que son capaces de compilar código bajo este estándar son Visual Studio y QT (el cual está diseñado para facilitar el diseño de software

²⁷ En la página oficial de Blacklib (<http://blacklib.yigityuce.com/>) podemos obtener toda la información necesaria para poder explotar la librería satisfactoriamente.

con interfaz gráfica de usuario). Cabe mencionar que el hecho de que puedan compilar el mismo código no significa que lo puedan hacer para la arquitectura ARM.

Como se mencionó anteriormente, se han hecho algunas modificaciones en los archivos de la librería para tener compatibilidad con Eclipse. A continuación mencionaremos los cambios hechos en la librería y cómo poder agregarla a nuestro proyecto en Eclipse.

Al descargar la librería de la página oficial obtendremos las tres versiones de la librería en tres carpetas diferentes denominadas v1_0, v2_0 y v3_0, para nuestro caso elegiremos la carpeta llamada v2_0. Al entrar a la carpeta v2_0 observaremos un total de 24 archivos los cuales incluyen principalmente códigos con extensión “.h” y “.cpp”, carpetas y otro tipo de archivos. Posteriormente eliminamos los archivos que contenían los siguientes nombres:

- La carpeta llamada “exampleAndTiming”.
- La carpeta llamada “LICENCE”.
- La carpeta llamada “SPI_SETUP”.
- El archivo llamado exampleAndTiming.cpp
- El archivo llamado “README.md”.
- El archivo llamado “ReleasesNotes”.
- El archivo llamado “WebPage.dox”.

Después de haber eliminado estos archivos, cambiaremos el nombre de la carpeta llamada “v2_0” por “BlackLib”. Con esto ya está lista la librería para ser usada en cualquier programa en C++ con Eclipse. Cabe mencionar que en el repositorio llamado “Tesis”, creado para este trabajo, se incluye la librería lista para ser usada sin tener que hacer ningún tipo de modificación.

Para poder explicar cómo agregar la librería a un proyecto se ha diseñado un programa que enciende un LED a través del GPIO 49, el cual funciona de la misma manera que el programa descrito en el diagrama de flujo de la figura 4-34, sólo que esta vez se ha programado utilizando la librería BlackLib. Cabe mencionar que si comparamos el código del programa echo anteriormente con el programa actual, nos daremos cuenta que el programa actual utiliza mucho menos líneas que el código que se generó primero, esto es gracias a que estamos empleando una librería que nos simplifica la interacción con el hardware.

El código para este programa lo podemos encontrar en el apéndice 2 o en el repositorio que hemos descargado de GitHub en la carpeta llamada Blink_2 junto con su archivo ejecutable.

Lo primero que debemos hacer es crear un nuevo proyecto, tal como se ha descrito anteriormente. Al crear este proyecto aparecerán las líneas de código para el programa de “Hello world” que sale por defecto cada que creamos un proyecto nuevo y que debemos borrar para copiar y pegar (o transcribir del apéndice 2) las líneas de código del archivo “.cpp” que se encuentra en la carpeta Blink_2 con el nombre de “blinkblacklib.cpp”. Al terminar de transcribir o copiar y pegar las líneas de código debemos buscar la carpeta que crea el IDE de Eclipse para almacenar todos los códigos que vayamos creando. Como se mencionó anteriormente, la carpeta lleva el nombre de “Workspace” y la podemos encontrar en la carpeta principal del usuario en nuestro sistema operativo Debian o Ubuntu. Una vez que hayamos encontrado la carpeta Workspace la abriremos para poder observar las carpetas con los diferentes proyectos que hemos

creado, tal como se puede observar en la figura 4-36. Ahora debemos buscar la carpeta que lleva el nombre de nuestro proyecto para pegar la librería en la carpeta que contiene nuestro código, la cual lleva el nombre de “src” tal como se puede observar en la figura 4.37.

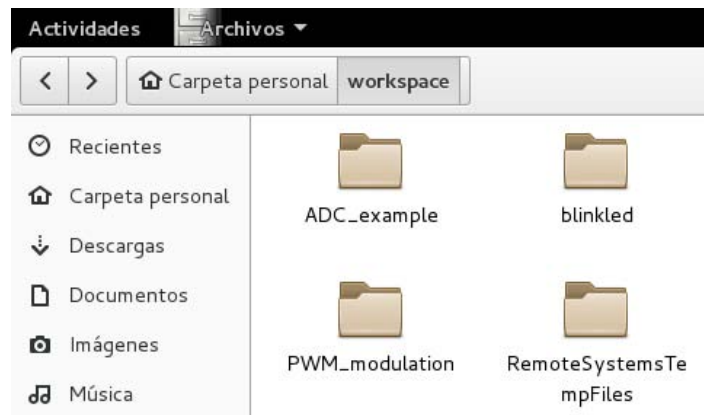


Figura 4-36. Carpeta Workspace.

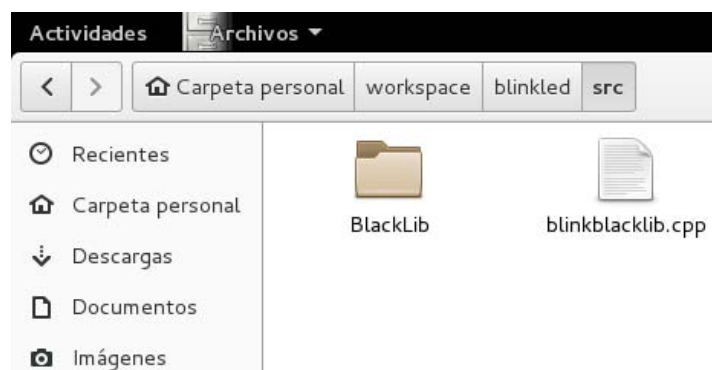


Figura 4-37. Librería BlackLib.

Ahora sólo falta ajustar las opciones de compilación del IDE de Eclipse para poder compilar bajo el estándar C++ 11. Para esto debemos ubicarnos en la barra de menús, dar clic en “Project” y buscar la opción que dice “Properties”. Al dar clic en “Properties” abrirá una ventana como la de la figura 4-38, en la cual debemos buscar la opción “C/C++ Build” y abrir el submenú para dar clic en “Settings”.

Al dar clic en “settings” abrirá un recuadro blanco en el cual aparece un menú llamado “Cross G++ compiler”, en este menú ya vienen desglosadas todas sus opciones. Después daremos clic en la opción que dice “Dialect” y seleccionaremos la opción “ISO C++ 11” en la sección que dice “Language standard” tal como se aprecia en la figura 4-37. En la sección que dice “Cross Gcc compiler” también daremos clic en la opción “Dialect” y seleccionaremos la opción “ISO C 11” en la sección que dice “Language standard”.

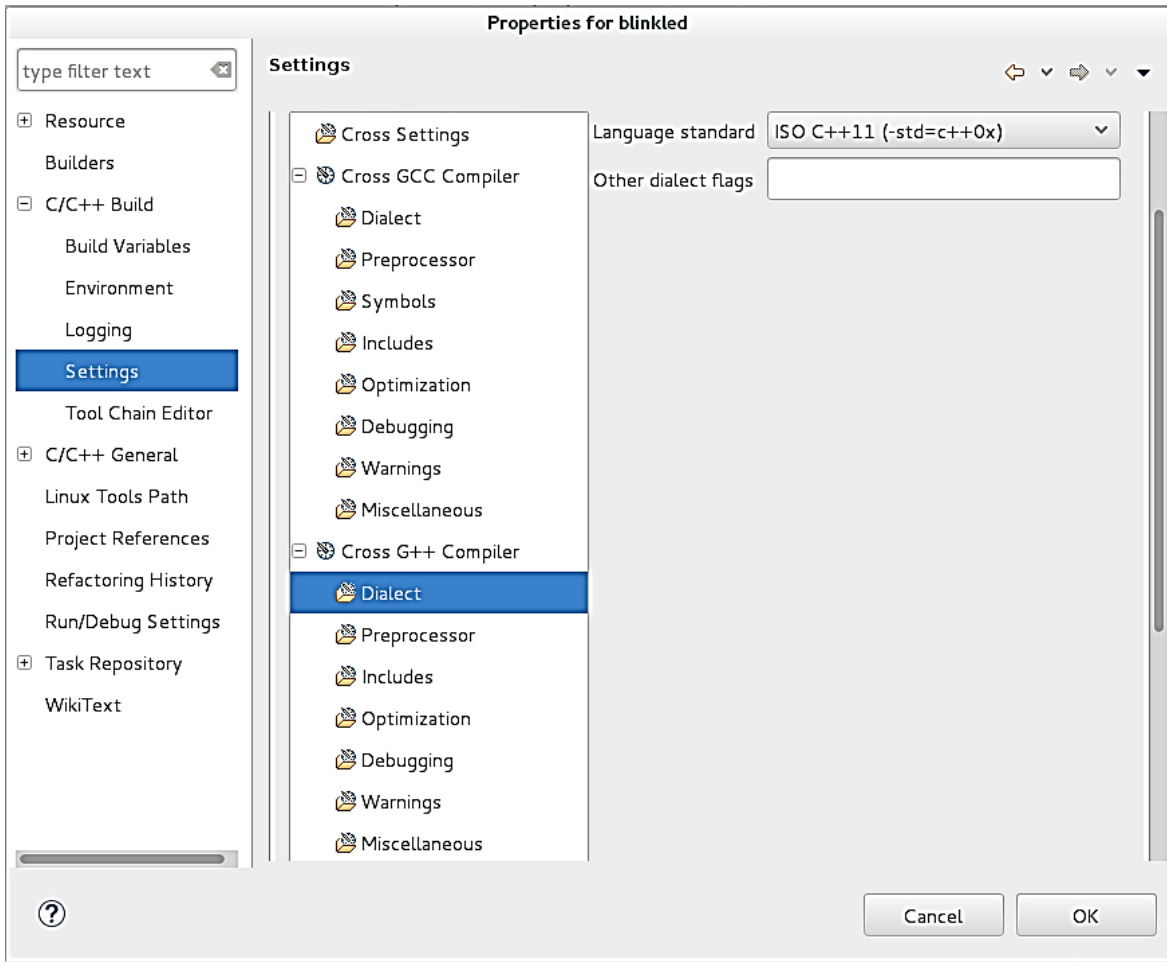


Figura 4-38. Propiedades de nuestro proyecto.

Después de haber configurado el modo de compilación y haber agregado la librería BlackLib, podremos compilar y obtener nuestro archivo ejecutable para posteriormente copiarlo a nuestro Beaglebone y ejecutar nuestro programa. En la siguiente figura se muestra en la terminal nuestro programa en ejecución.

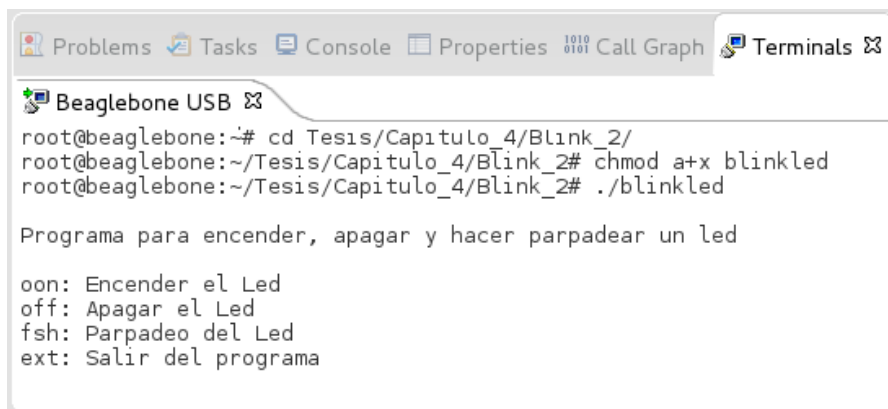


Figura 4-39. Programa en ejecución mediante la terminal de comandos.

4.5 Programación de diferentes Buses con BlackLib.

GPIO como entrada digital.

En los dos programas anteriores ya se ha visto como programar un GPIO, con y sin, emplear la librería BlackLib. Sin embargo, sólo hemos empleado un GPIO como una salida digital. Para este ejemplo utilizaremos dos GPIO como entradas digitales para hacer prender un led y para detener el programa, pero antes de empezar con la ejecución de este ejemplo, debemos hablar sobre dos configuraciones de circuitos con resistencias, denominadas “Pull-up” y “Pull-down” y cómo saber si nuestra tarjeta de desarrollo los tiene activados.

Las resistencias denominadas Pull-up y Pull-down son necesarias cuando implementamos algún tipo de interruptor, por ejemplo un botón, ya que gracias a estas resistencias podemos tener un estado alto o un estado bajo sin dejar una entrada digital en un estado indefinido. En otras palabras, si no se emplean estas resistencias, al estar abierto un interruptor estaríamos dejando sin conectar nuestra entrada digital, lo cual podría ocasionarnos problemas debido diferentes factores como por ejemplo, el ruido eléctrico.

En la figura 4-40 podemos observar que la resistencia Pull-down se conecta directamente a tierra. De esta manera, cuando el interruptor está abierto, la corriente se dirigirá hacia la resistencia dejando un valor de 0 en “Vout” y si el interruptor está cerrado la corriente se moverá hacia “Vout” dejando un valor lógico alto.

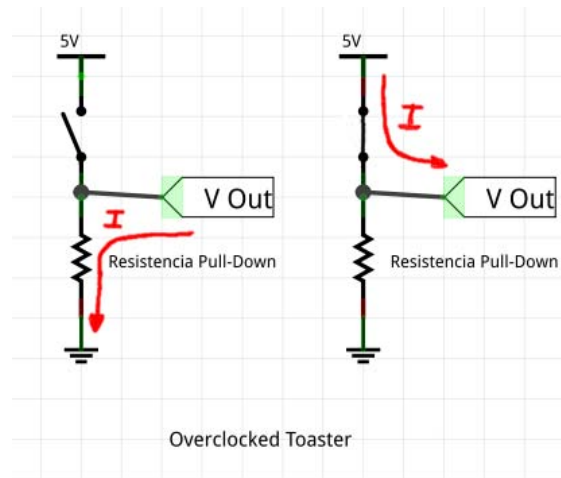


Figura 4-40. Resistencia Pull-down [19].

En la figura 4-41 podemos ver que la resistencia Pull-up se conecta directamente a la nuestra fuente de voltaje, de esta manera, cuando el interruptor está abierto la corriente va desde la fuente de alimentación a “Vout” dando un estado alto y si el interruptor está cerrado la corriente se mueve hacia tierra dejando “Vout” en estado bajo.

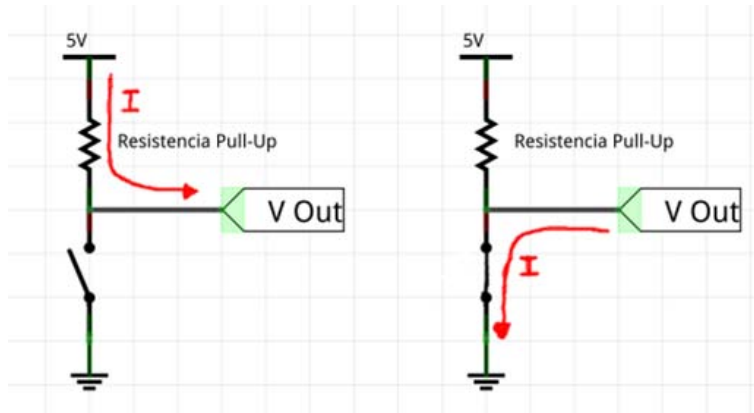


Figura 4-41. Resistencia Pull-up [20].

Beaglebone cuenta con resistencias Pull-up y Pull-down internas las cuales pueden ser configuradas mediante algunos registros internos del AM3358 o mediante la terminal de comandos. Cabe mencionar que no haremos énfasis en como activar las resistencias Pull-up y Pull-down mediante la terminal de comandos ya que lo podemos lograr fácilmente empleando BlackLib en un código.

BlackLib emplea dos modos de trabajo para cada GPIO denominados “Fastmode” y “Securemode”, de los cuales hablaremos a continuación:

“Fastmode” nos permite tomar control de cualquiera de las GPIO con las que cuenta la tarjeta sin ninguna limitación, sin importar que esté multiplexado con otro bus y que el otro bus con el que comparte el mismo pin esté activo y en uso, lo cual quiere decir que “Fastmode” interrumpe las tareas que esté ejecutando el bus con el que comparte el mismo pin para disponer de dicho pin y usarlo como GPIO. Al trabajar con una GPIO en modo “Fast” debemos ser cuidadosos ya que si elegimos trabajar con una GPIO que está multiplexada con un bus que controle una tarea importante, podría verse afectado el funcionamiento de algún dispositivo que sea de gran importancia para nuestro uso. Por ejemplo, si elegimos algún GPIO que está multiplexado con los pines que controlan el chip de la salida HDMI podríamos perder la señal de video si estuviéramos utilizando un monitor para visualizar la interfaz gráfica del sistema operativo de nuestra tarjeta.

“Securemode” a diferencia de “fastmode”, espera a que el bus con el que está multiplexado se encuentre desactivado para poder tomar control del pin que comparte con dicho bus, además cuando declaramos una GPIO como una entrada en este modo, se activa automáticamente la resistencia Pull-up. Si deseamos trabajar en “Securemode” con algún GPIO que comparte un pin con otro bus, primero debemos desactivar dicho bus. Cabe mencionar que para este trabajo hemos utilizado en “Securemode” todas las GPIO que fueron empleadas en los códigos descargados de GitHub y que se encuentran en el apéndice, además, las GPIO que utilizamos en los códigos para este trabajo no están multiplexadas con otros dispositivos. Cabe mencionar que con BlackLib sólo podemos trabajar con la resistencia Pull-up, por tal razón, si deseamos trabajar con la resistencia Pull-Down interna, BlackLib no nos será de ayuda; en este caso es mejor implementar una resistencia Pull-Down externa.

Ahora que ya sabemos que la resistencia Pull-up se puede activar mediante una instrucción de la librería BlackLib, procederemos a armar el circuito mostrado en la figura 4-42 empleando una resistencia de $220\ \Omega$ en ánodo del LED. En este circuito podemos observar que en cada uno de los botones que se han implementado, no hemos agregado ningún tipo de resistencia ya que activaremos las resistencias Pull-up mediante el programa.

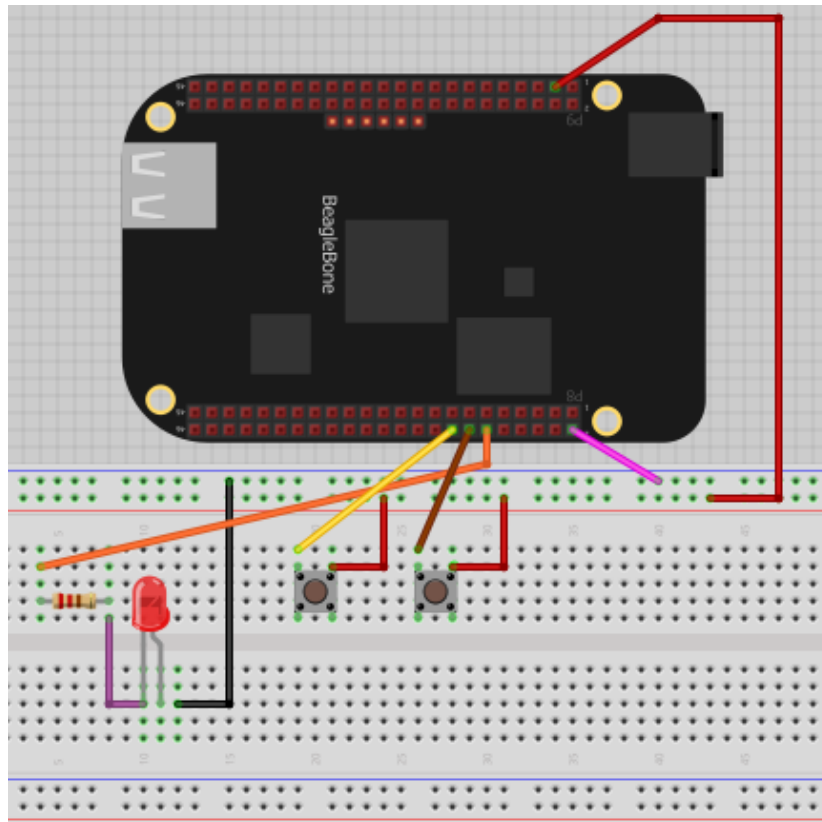
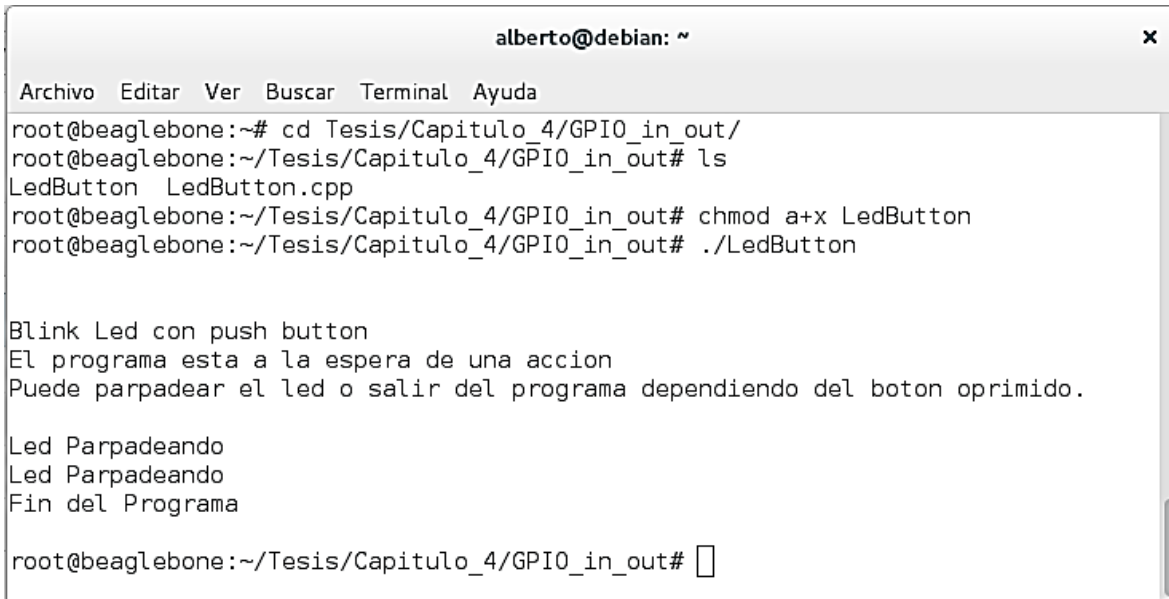


Figura 4-42. Conexiones del circuito para este ejemplo.

Una vez que ya tengamos conectado el circuito de la figura 4-42, accederemos a la terminal de nuestro Beaglebone para posteriormente ingresar los comandos necesarios para cambiar a la carpeta de nuestro repositorio donde se encuentra el archivo ejecutable, además cambiar derechos de uso de nuestro archivo ejecutable y correr nuestro programa. En la figura 4-42 podemos observar la terminal de comando del Beaglebone con los comandos necesarios para poder acceder a nuestro programa y ejecutarlo.

Cuando nuestro programa está en ejecución nos muestra las instrucciones en la terminal, además notifica en la terminal la acción que empezará a hacer inmediatamente después de que se presionó alguno de los dos botones. Como podemos ver en la imagen 4-43 se presionó dos veces el botón que hace parpadear el led, después se presionó el botón para salir del programa.



```
alberto@debian: ~
Archivo Editar Ver Buscar Terminal Ayuda
root@beaglebone:~# cd Tesis/Capitulo_4/GPIO_in_out/
root@beaglebone:~/Tesis/Capitulo_4/GPIO_in_out# ls
LedButton LedButton.cpp
root@beaglebone:~/Tesis/Capitulo_4/GPIO_in_out# chmod a+x LedButton
root@beaglebone:~/Tesis/Capitulo_4/GPIO_in_out# ./LedButton

Blink Led con push button
El programa esta a la espera de una accion
Puede parpadear el led o salir del programa dependiendo del boton oprimido.

Led Parpadeando
Led Parpadeando
Fin del Programa

root@beaglebone:~/Tesis/Capitulo_4/GPIO_in_out#
```

Figura 4-43. Programa en ejecución.

Uso de los convertidores Analógico Digital.

Nuestra tarjeta de desarrollo cuenta con 7 entradas analógicas, las cuales pueden ser usadas para conectar algún sensor o para obtener la representación digital de esta señal y poder procesarla de manera digital siempre y cuando la señal se pueda adaptar a las características que ofrecen los ADC de nuestra tarjeta. Cabe mencionar que en este trabajo no hablaremos sobre el procesamiento digital de una señal ya que este tema es demasiado complejo como para poder abordarlo en este trabajo de manera breve.

Como se mencionó en el capítulo 2, el voltaje máximo que soportan las entradas analógicas no debe exceder de 1.8V, si excedemos este voltaje podemos dañar gravemente las entradas analógicas ya que son extremadamente delicadas. El tiempo de muestreo que nos ofrecen los ADC es de 125ns, lo que significa que su frecuencia de muestreo es de 8Mhz. La resolución digital con la que disponemos es de 12 bits, lo cual significa que todas las muestras de voltaje serán representadas con una palabra de una longitud de 12 bits, es decir, que existen $2^{12} = 4096$ posibles representaciones digitales de voltaje. Si deseamos procesar una señal de manera digital debemos considerar el número de muestras que queremos tomar en un periodo, por ejemplo, si queremos tomar 1000 muestras por periodo, es bastante obvio que nuestra señal debe ser de 8khz ya que un periodo de nuestra señal de entrada duraría 125us y nuestro tiempo de muestreo es de 125ns.

Además de las 7 entradas digitales, también contamos con dos pines extras denominados VDD_ADC y GNDA_ADC, el primero se encarga de proporcionarnos un voltaje de referencia de 1.8v estable y el segundo se encarga de la conexión a tierra de nuestros dispositivos analógicos.

Para ejemplificar el uso de los ADC hemos programado un ejemplo en C++ con BlackLib en el IDE de Eclipse. Con este programa podemos controlar el encendido y apagado de cada LED de una tira de 10 LEDs variando la tensión en la entrada analógica AIN1 con un potenciómetro de 1 kΩ. Se

recomendó usar un potenciómetro de 1kΩ como divisor de voltaje si se trabaja con el voltaje de referencia de 1.8v [21].

Para poder hacer funcionar nuestro programa, primero debemos armar el circuito de la figura 4-44, con un potenciómetro de 1kΩ. Es recomendable revisar el código de este programa en el apéndice para verificar que hagamos las conexiones correctamente con nuestra tarjeta y los LEDs.

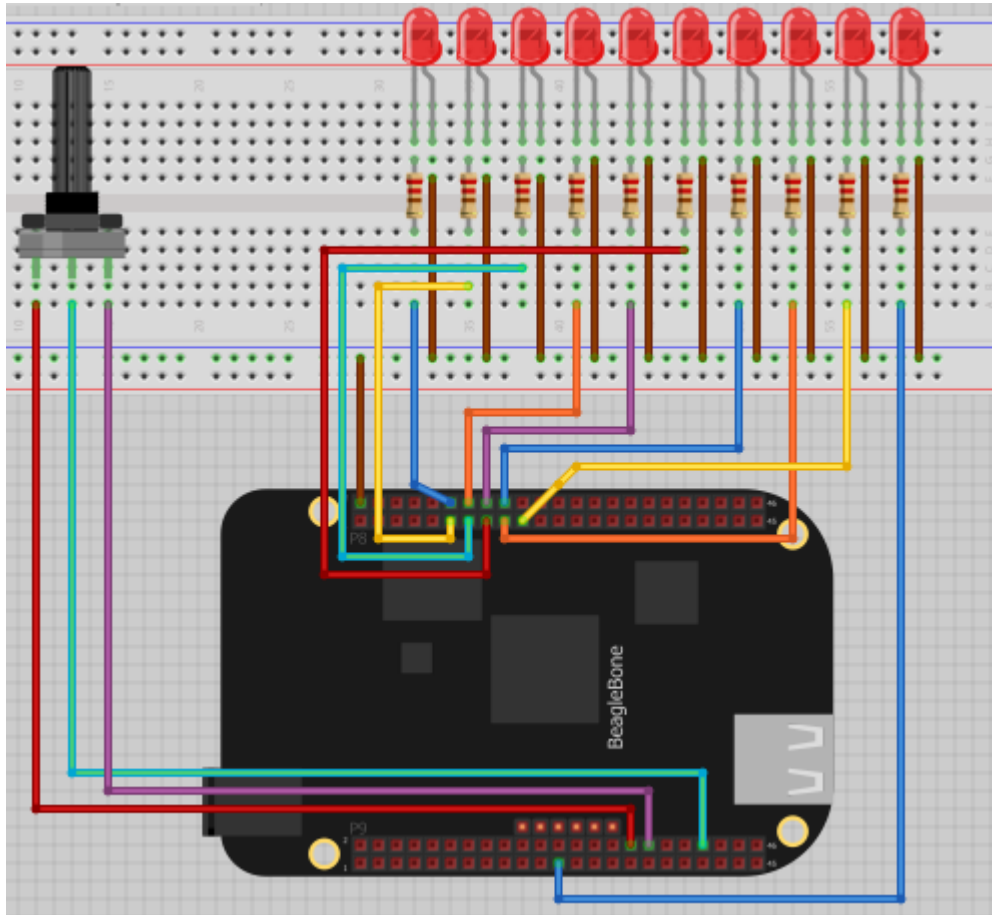


Figura 4-44. Circuito de prueba para el ADC.

```
alberto@debian: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
root@beaglebone:~# cd Tesis/Capitulo_4/Us0_adc/  
root@beaglebone:~/Tesis/Capitulo_4/Us0_adc# ls  
ADC_example ADC_example.cpp  
root@beaglebone:~/Tesis/Capitulo_4/Us0_adc# chmod a+x ADC_example  
root@beaglebone:~/Tesis/Capitulo_4/Us0_adc# ./ADC example  
  
Ejemplo de uso del ADC. Para salir del programa presione ctrl+c  
^C  
root@beaglebone:~/Tesis/Capitulo_4/Us0_adc#
```

Figura 4-45. Ejecutando, programa para el ADC en la terminal.

El procedimiento para poder ejecutar este programa es el mismo que hemos venido haciendo en los dos programas anteriores. En la figura 4-45 podemos ver la terminal con los comandos necesarios para poder cambiar los permisos de usuario y correr nuestro programa. Cuando nuestro programa esté en ejecución debemos variar la posición de la resistencia del potenciómetro para observar cómo se van prendiendo los demás LEDs, lo cual explicaremos a continuación.

Como se puede observar en el código del apéndice 4 y en la figura 4-46, BlackLib nos entrega el valor del voltaje obtenido en el ADC en mV dentro de una variable de tipo entero, por lo tanto, el valor máximo de voltaje que podríamos obtener es de 1800. Este programa provoca el encendido de un LED cuando la resistencia se encuentra en el rango de voltaje que corresponde a cada LED, además apaga al LED que estaba encendido anteriormente. En la figura 4-46 podemos ver detalladamente en un diagrama de flujo el funcionamiento de este programa.

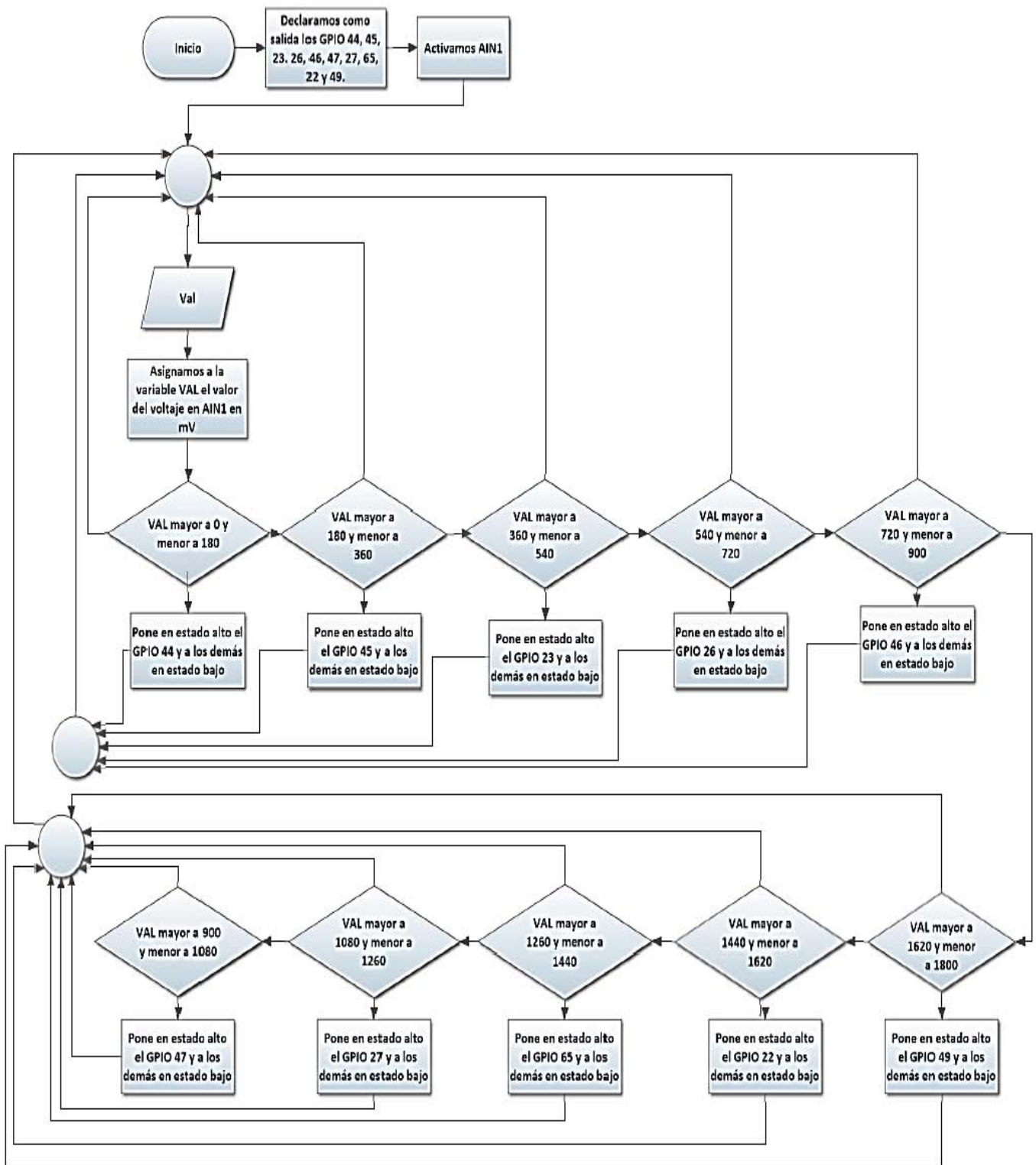



Figura 4-47. Diagrama de flujo del programa para usar el ADC.

Cuando abrimos el programa automáticamente se activan los ADC que hayamos declarado en nuestro código en C++ gracias a la librería BlackLib, sin embargo, es posible obtener valores del ADC directamente desde la terminal. A continuación explicaremos cómo obtener los valores de la entrada analógica AIN1, como una demostración de que se pueden tener acceso a los buses desde la terminal, sin embargo, esto no se hará con todos los buses ya que tener control de un bus desde la terminal no es viable en la implementación de algún proyecto o aplicación de ingeniería.

Para poder activar la entrada analógica AIN1 debemos ingresar el comando 1 de la figura 4-46. Para poder leer el valor en mV el valor que tiene nuestra entrada AIN1 debemos ingresar el comando número 2. Como podemos observar en la figura 4-46, la primera vez que captamos el valor en mV nos dio 1106, lo cual quiere decir que el potenciómetro estaba un poco arriba de la mitad de la capacidad total de su resistencia. Cuando ingresamos nuevamente el comando para leer el valor de AIN1 en la línea 3 observamos que el potenciómetro estaba aproximadamente al tope con la conexión a tierra y cuando ingresamos la cuarta línea observamos que el potenciómetro estaba aproximadamente al tope con la conexión al voltaje de referencia de 1.8v. Cabe mencionar que para este ejemplo hemos empleado la conexión SSH mediante Windows para demostrar que también se puede trabajar fácilmente empleado Putty. Esto lo hicimos para comprobar que a través de la terminal "Putty" también podemos tener acceso a los archivos que controlan los buses de la tarjeta BeagleBone.



```
192.168.7.2 - PuTTY
root@beaglebone:~# echo cape-bone-iiio > /sys/devices/bone_capemgr.*/slots
root@beaglebone:~# cat /sys/devices/ocp.3/helper.15/AIN1
1106
root@beaglebone:~# cat /sys/devices/ocp.3/helper.15/AIN1
1
root@beaglebone:~# cat /sys/devices/ocp.3/helper.15/AIN1
1798
root@beaglebone:~#
```

Figura 4-47. Uso del AIN1 desde la terminal de comandos.

Uso de las salidas PWM.

Beaglebone cuenta con 8 salidas PWM, las cuales están divididas en tres módulos denominados EHRPWM²⁸ (dos salidas PWM por cada módulo) y dos módulos denominado eCAP²⁹ (una salida PWM por módulo). Cada salida PWM puede ser manipulada mediante un programa en C++ con la

²⁸ (Enhanced High-Resolution PWM) Es una tecnología desarrollada por Texas Instruments para aumentar la velocidad de transición entre un estado bajo y un estado alto en una señal PWM. Para más información consulte: [22]

²⁹ (Enhanced Capture) Son módulos con los que cuentan diversos CI de Texas Instruments que se interconectan con los módulos que controlan buses con la finalidad de ejecutar de manera sincronizada una acción derivada de una señal PWM. Para más información consulte: [23]

librería BlackLib, sin embargo, hay salidas PWM que se encuentran multiplexadas con otros buses, además están siendo ocupados por el CI que controla el conector HDMI. BlackLib fue diseñada para desactivar el puerto HDMI en caso de que el programador necesite utilizar las PWM multiplexadas con las señales que controlan el CI del puerto HDMI, por tal razón, debemos considerar que si deseamos utilizar ciertas PWM no podremos emplear un monitor o pantalla para visualizar el entorno gráfico de usuario de Beaglebone, sólo podremos tener control vía SSH o a través de un escritorio remoto³⁰.

Otro detalle importante a observar es que existen salidas PWM que están presentes en dos pines distintos, por ejemplo, en los pines P8-19 y P8-45 podemos encontrar la salida EHRPWM2A. Esto se puede verificar en la figura 4 del anexo 1.

Para analizar el uso de las salidas PWM se han escrito dos códigos en C++ con el IDE de Eclipse. Los códigos para este programa los podemos encontrar en los apéndices 5 y 6, ó en el repositorio que descargamos de GitHub, en la carpeta llamada "capitulo_4", al igual que los programas anteriores, podemos transcribir el código en un nuevo proyecto en Eclipse, ejecutarlo y copiarlo al Beaglebone o abrir el archivo ejecutable como se ha venido explicando en los programas anteriores.

Para poder ejecutar los programas mencionados anteriormente debemos armar el circuito de la figura siguiente. Empleado una resistencia de 220Ω y un osciloscopio.

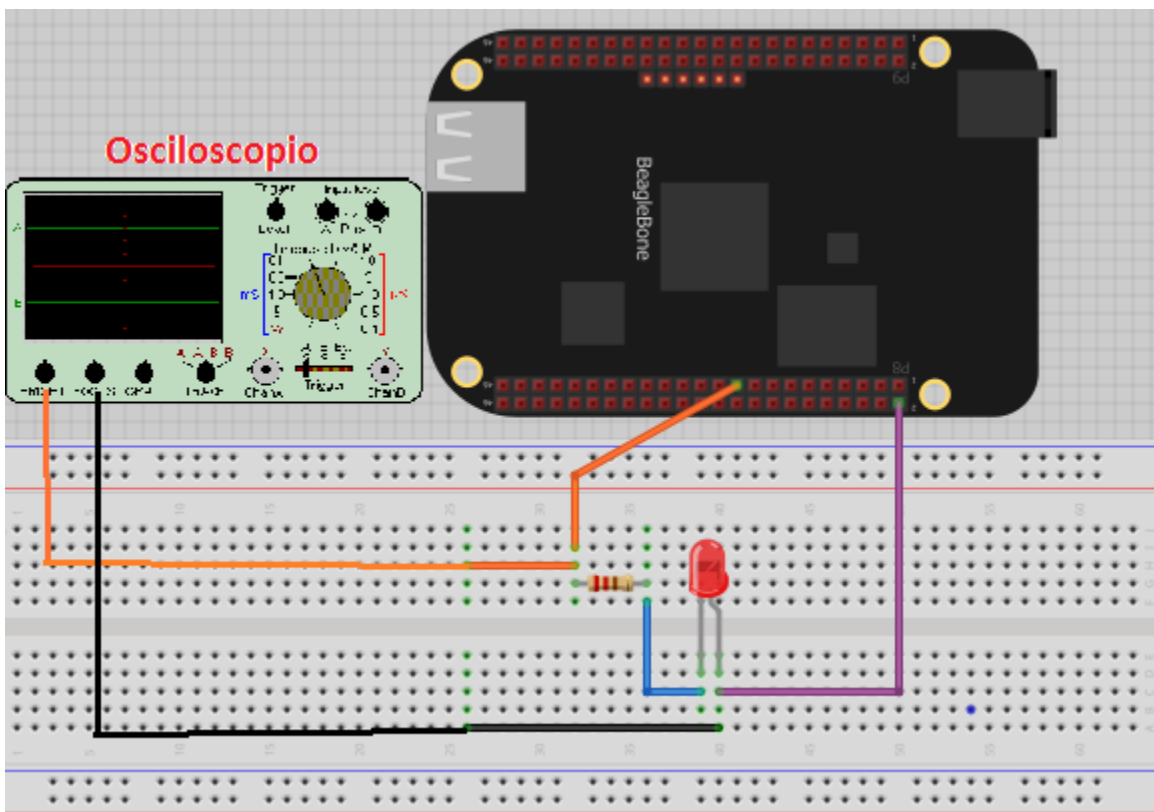


Figura 4-48. Circuito para el ejemplo de uso de una salida PWM.

³⁰ Para verificar la ubicación de las salidas PWM en los pines de la tarjeta consulte la figura 4 del anexo 1.

En la figura 4-49 observaremos el diagrama de flujo que explica en funcionamiento del primer programa. Este programa se encarga de crear una señal PWM con la frecuencia y el ciclo de trabajo que el usuario ingrese al ejecutarse el programa. Como podemos ver en el diagrama, el ciclo de trabajo (Dutty_cycle) se ingresa en porcentaje, por tal razón no puede exceder el 100%, si el número que ingresamos es mayor a 100, el programa nos pedirá que ingresemos nuevamente la variable Dutty_cycle. Una vez que se haya ingresado la frecuencia y el ciclo de trabajo de nuestra señal, el programa generará una señal PWM en el pin P9_19, la cual podremos visualizar en el osciloscopio. Cabe mencionar que si la señal es de una frecuencia muy baja y tiene un ciclo de trabajo que se aproxime al 50%, podremos ver el efecto que causa esta señal en nuestro LED. Además, este programa nos da la opción de generar otra señal o salir del programa.

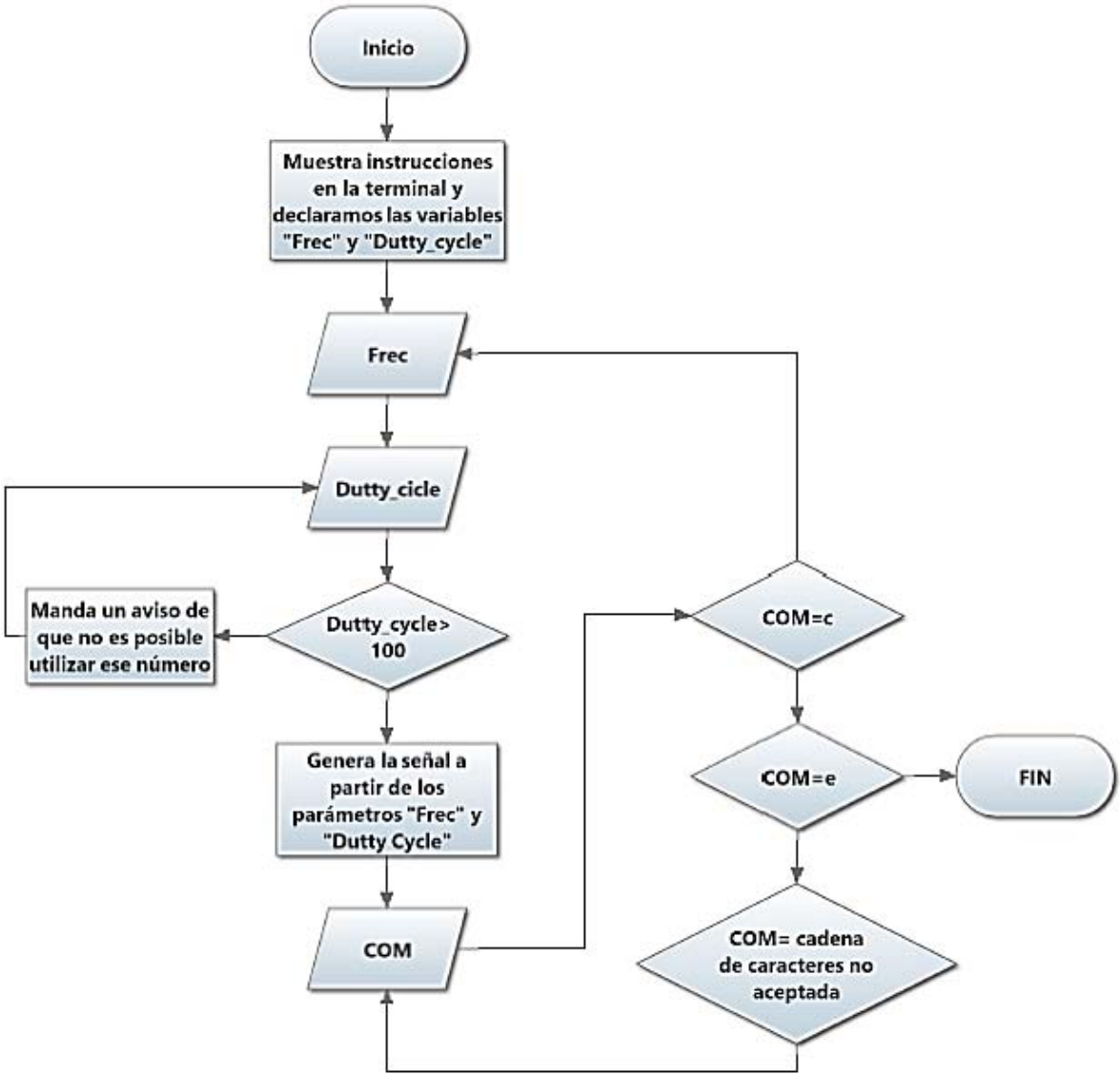
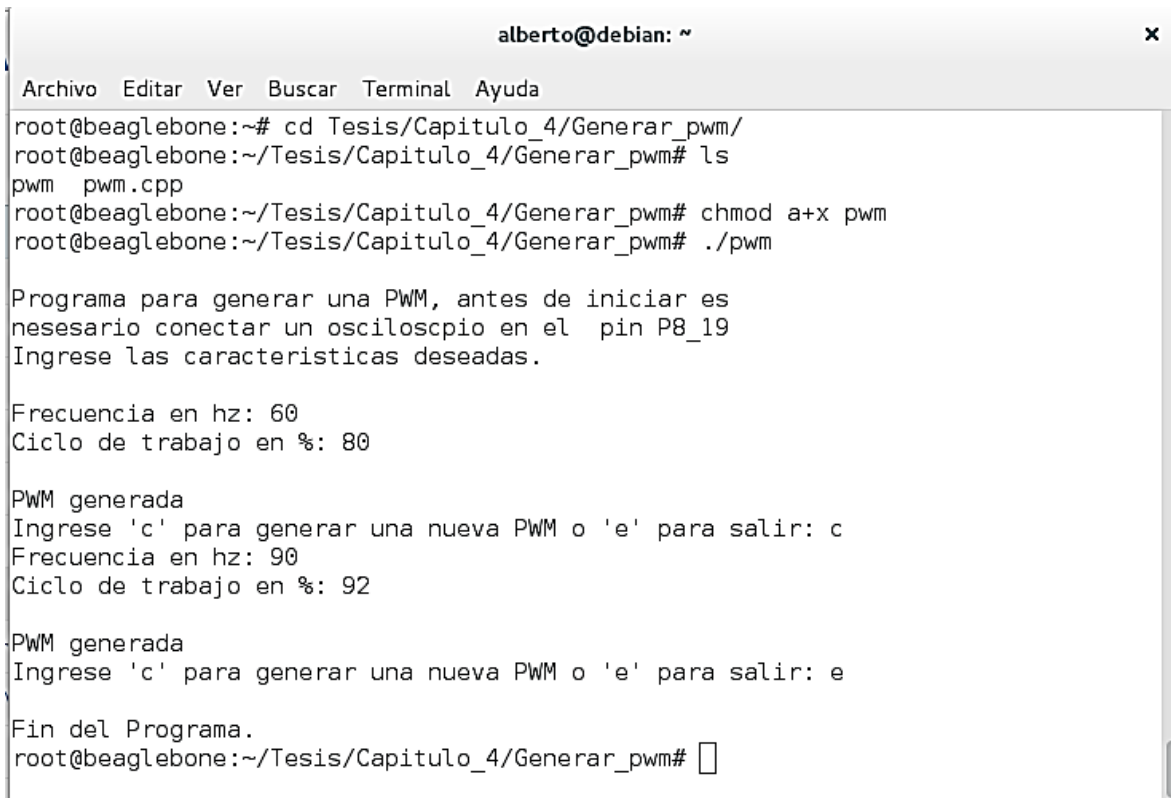


Figura 4-49. Programa para crear una señal PWM.

En la siguiente figura podemos observar la consola de comandos con todos los códigos necesarios para poder ejecutar el programa. También podemos observar que se han creado dos PWM distintas a través de las opciones que hemos definido en el código de este programa.



```
alberto@debian: ~
Archivo Editar Ver Buscar Terminal Ayuda
root@beaglebone:~# cd Tesis/Capitulo_4/Generar_pwm/
root@beaglebone:~/Tesis/Capitulo_4/Generar_pwm# ls
pwm  pwm.cpp
root@beaglebone:~/Tesis/Capitulo_4/Generar_pwm# chmod a+x pwm
root@beaglebone:~/Tesis/Capitulo_4/Generar_pwm# ./pwm

Programa para generar una PWM, antes de iniciar es
necesario conectar un osciloscopio en el pin P8_19
Ingrese las características deseadas.

Frecuencia en hz: 60
Ciclo de trabajo en %: 80

PWM generada
Ingrese 'c' para generar una nueva PWM o 'e' para salir: c
Frecuencia en hz: 90
Ciclo de trabajo en %: 92

PWM generada
Ingrese 'c' para generar una nueva PWM o 'e' para salir: e

Fin del Programa.
root@beaglebone:~/Tesis/Capitulo_4/Generar_pwm#
```

Figura 4-50. Programa en ejecución para generar una PWM.

Para el siguiente programa haremos variar de forma ascendente y descendente el ciclo de trabajo de nuestra PWM para ver cómo varía la intensidad de iluminación de nuestro LED. En la figura 4-51 se muestra la terminal de comandos con los pasos necesarios para ejecutar este programa. Cabe mencionar que si deseamos detener este programa debemos presionar Ctrl+c.

```

alberto@debian: ~
Archivo Editar Ver Buscar Terminal Ayuda
root@beaglebone:~# cd Tesis/Capitulo_4/Modulacion_pwm/
root@beaglebone:~/Tesis/Capitulo_4/Modulacion_pwm# ls
PWM_modulation PWM_modulation.cpp
root@beaglebone:~/Tesis/Capitulo_4/Modulacion_pwm# chmod a+x PWM_modulation
root@beaglebone:~/Tesis/Capitulo_4/Modulacion_pwm# ./PWM_modulation

Con este programa podremos observar la modulacion de la PWM en un osciloscopio
Ingrese la frecuencia deseada para nuestra señal pwm: 60
^C
root@beaglebone:~/Tesis/Capitulo_4/Modulacion_pwm#

```

Figura 4-51. Programa en ejecución para generar modulación PWM.

El diagrama de flujo correspondiente a este programa se muestra en la figura 4-52. En este programa sólo la frecuencia es ingresada por el usuario ya que la modulación asciende o desciende en 1% cada 200us.

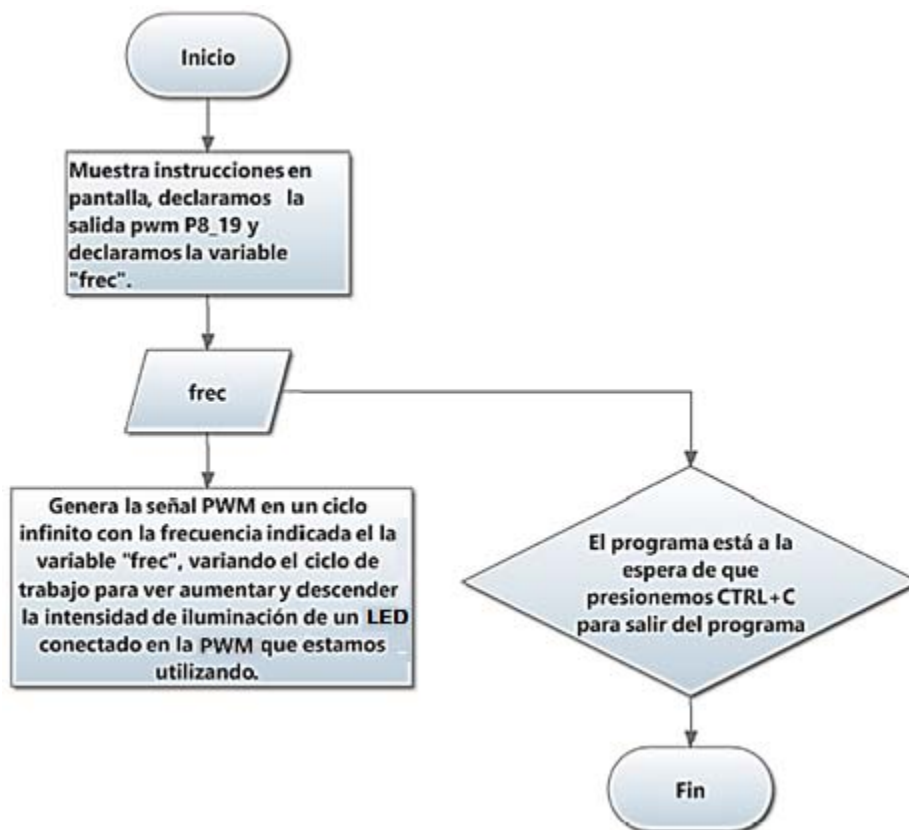


Figura 4-52 Diagrama de flujo del programa que modula una señal PWM.

Uso de un bus de comunicaciones UART.

Los dispositivos UART (Universal Asynchronous Receiver-Transmitter) son usados para tener comunicación serial con otros dispositivos de manera asíncrona. Estos dispositivos son muy

utilizados en microcontroladores para tener comunicación con otros dispositivos que también cuenten con un bus tipo UART.

Un dispositivo UART toma bytes de datos y transmite los bits individuales de forma secuencial a través de un sólo cable. En el destino, un segundo UART vuelve a ensamblar los bits en bytes de datos para poder ser utilizados por una unidad de procesamiento.

Todos los dispositivos UART cuentan con dos señales sumamente importantes llamadas TxD (Transmitted Data) y RxD (Received Data), las cuales se encargan de enviar y recibir la información entre dispositivos tipo UART. Para tener comunicación entre dos dispositivos UART debemos interconectar la señal TxD del transmisor con RxD del receptor. A continuación explicaremos detalladamente cómo funciona la comunicación entre dos dispositivos UART.

Para poder lograr una comunicación de datos asíncrona entre dos dispositivos, el transmisor y el receptor deben tener la misma velocidad, tanto de transmisión como de recepción, es decir, el tiempo de transmisión de un bit debe ser igual al tiempo de recepción (a este parámetro se le conoce como baud rate o bps en algunas ocasiones³¹). Cuando los dispositivos se encuentran en un estado de reposo (sin transmisión de datos) el transmisor envía un estado “alto” al receptor, siendo un flanco de bajada el encargado de iniciar la comunicación, por tal razón el bit de inicio siempre es un estado “bajo”. Como se puede observar en la figura 4-53, 1.5 bits después de la detección del flanco de bajada comienza el muestreo de bits con una duración de un bit entre cada muestra, iniciando la transmisión con el bit menos significativo. Después de transmitir una secuencia de 8 bits (Byte) se envía el bit de paridad el cual suele ser opcional y podemos omitir si nuestro canal de transmisión es libre de ruidos. Por último, se envía el bit de parada el cual consisten en un estado “alto” para dejar la comunicación en un estado de reposo.

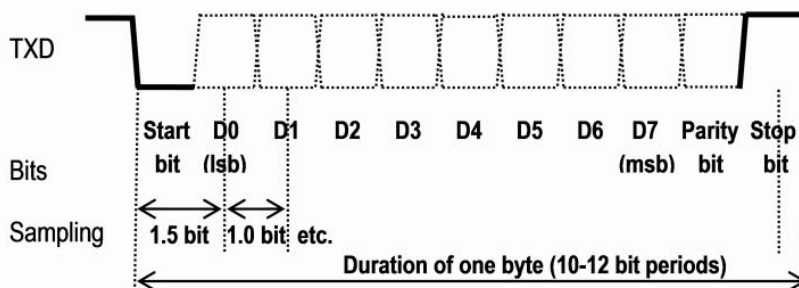


Figura 4-53. Comunicación serial de un dispositivo UART[25].

Como se mencionó anteriormente, Beaglebone cuenta con 6 UART, sin embargo, el usuario sólo puede tener acceso a 5 de estos buses ya que uno está reservado para el puerto Debug serial, del cual se habló en el capítulo 2. Otro detalle importante es que existe un bus UART que sólo tiene disponible la señal TxD, por tal razón, este bus sólo es capaz de transmitir datos. En la figura A1.5 del anexo 1 podemos observar la distribución pines de los buses UART.

Para analizar el uso de un bus UART se ha desarrollado un programa en Eclipse que es capaz de recibir y enviar datos a través de un módulo bluetooth HC05 para encender un LED. El bluetooth HC05 es un dispositivo que recibe y envía los datos a través de una UART para posteriormente

³¹ El bps es una unidad de medida que representa el número de bits por segundo en un medio de transmisión digital [24].

comunicarse inalámbricamente con otro dispositivo vía Bluetooth para enviar o recibir datos. El dispositivo HC05 es frecuentemente utilizado con las tarjetas Arduino de Atmel o con los microcontroladores PIC de Microchip. En la figura siguiente podemos observar el dispositivo HC05 que utilizaremos para este ejemplo.

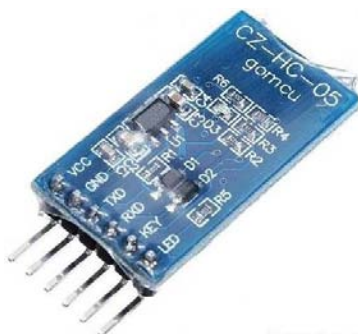


Figura 4-54. Modulo Bluetooth HC05.

El módulo HC05 está configurado para trabajar a 9600 baud, para no utilizar bit de paridad y para utilizar un solo bit de parada (Stop bit). Sin embargo, estas configuraciones pueden ser modificadas con ayuda de un submenú de comandos denominados “Comandos AT”³².

Antes de continuar debemos contar con una terminal con bluetooth y un sistema operativo Android o IOS, para este ejemplo usaremos una aplicación que funciona en Android. Posteriormente descargaremos de Google Play una aplicación llamada “Bluetooth Terminal”. Esta aplicación es gratuita y compatible con un gran numero teléfonos inteligentes que cuentan con un sistema Android y conexión bluetooth. En la figura 4-53 se muestra el icono de la aplicación mencionada anteriormente con la finalidad de mostrar al usuario la imagen representativa de la aplicación ya que existen otras con el mismo nombre.



Figura 4-55. Icono de Bluetooth Terminal.

Después de descargar e instalar la aplicación mencionada, procederemos a armar el circuito de la figura 4-56, empleando una resistencia de 220Ω en el ánodo del led. Para este ejemplo hemos utilizado la UART4 (ver la figura A1.5 del anexo 1).

³² Para acceder al menú de comandos AT del HC05 consulte: [26].
La lista de comandos AT está disponible en: [27]

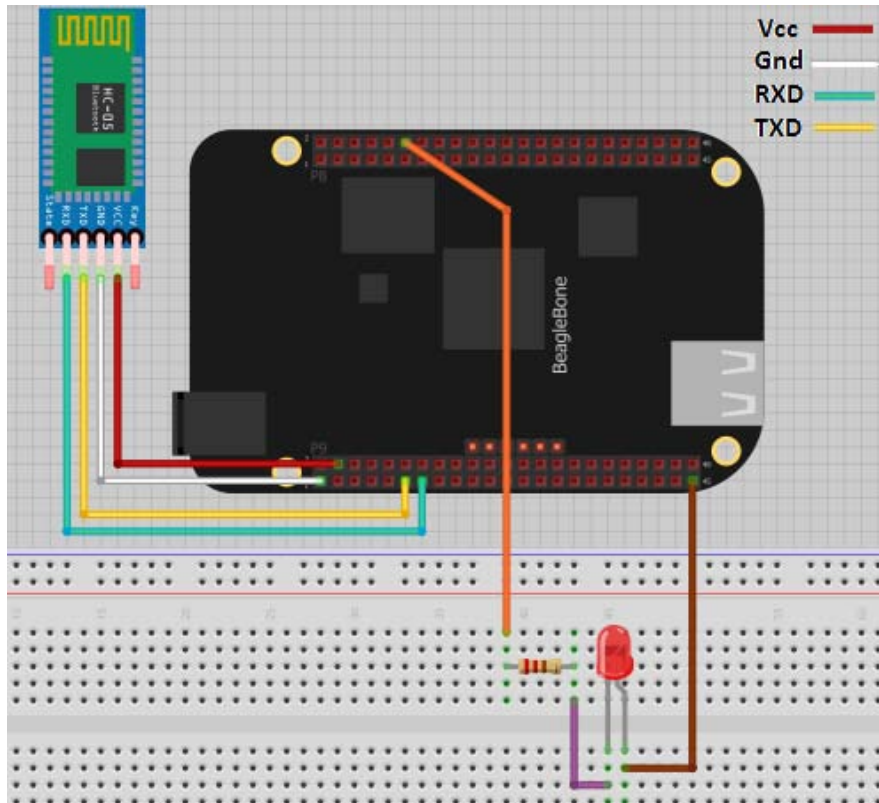


Figura 4-56. Circuito para el uso de la UART4.

Después de haber armado el circuito procederemos a conectarnos con nuestro módulo Bluetooth a través de nuestro dispositivo Android. Para ello debemos activar el bluetooth de nuestro celular y posteriormente abrir la aplicación que hemos descargado. Al abrir nuestra aplicación detectará automáticamente los dispositivos Bluetooth que estén en el rango de alcance de nuestro teléfono, en este caso sólo se detectó el módulo Bluetooth, tal como se muestra en la figura 4-57. Para establecer una conexión entre nuestro celular y nuestro dispositivo HC05 basta con dar un toque sobre el nombre de nuestro dispositivo (en este caso el HC05).

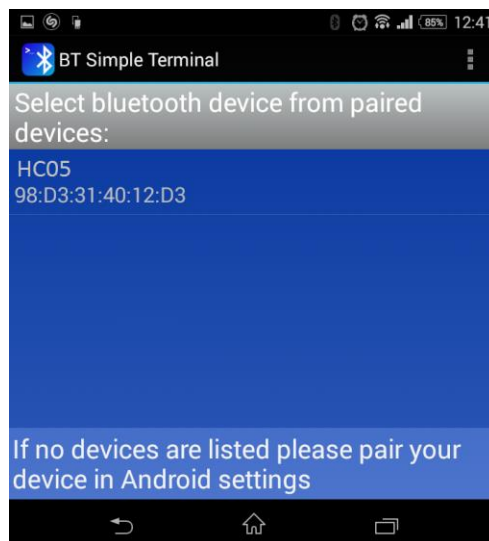
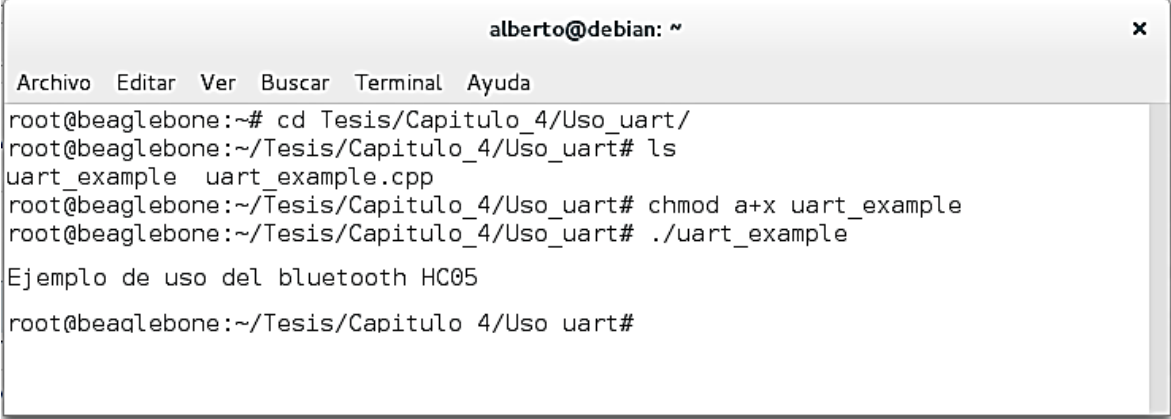


Figura 4-57. Conexión con el dispositivo HC05.

Una vez que hayamos establecido la conexión entre el dispositivo HC05 y nuestro teléfono celular debemos abrir la terminal de comandos del Beaglebone a través de una conexión SSH y ejecutar el programa para este ejemplo, el cual se encuentra en el repositorio que descargamos de GitHub. Los comandos para tener acceso al programa y ejecutarlo se muestran en la figura 4-56.



```
alberto@debian: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
root@beaglebone:~# cd Tesis/Capitulo_4/Uso_uart/  
root@beaglebone:~/Tesis/Capitulo_4/Uso_uart# ls  
uart_example  uart_example.cpp  
root@beaglebone:~/Tesis/Capitulo_4/Uso_uart# chmod a+x uart_example  
root@beaglebone:~/Tesis/Capitulo_4/Uso_uart# ./uart_example  
Ejemplo de uso del bluetooth HC05  
root@beaglebone:~/Tesis/Capitulo 4/Uso uart#
```

Figura 4-58. Comandos para ejecutar el programa para usar la UART.

Como podemos apreciar en la figura 4-57, cuando ejecutamos el programa automáticamente se envían las instrucciones a nuestro teléfono celular y las podemos visualizar mediante la aplicación Bluetooth Terminal. A través Bluetooth Terminal podemos mandar una cadena de caracteres que al ser recibida por nuestro Beaglebone, éste ejecutará una de las acciones descritas en las instrucciones enviadas previamente. Por ejemplo, si mandamos la cadena de caracteres “Encender” el led prenderá, o si mandamos la cadena de caracteres “Flash”, el led comenzará a parpadear. Para Salir del programa debemos mandar la cadena de caracteres “Salir”, tal como se aprecia en la figura 4-57.

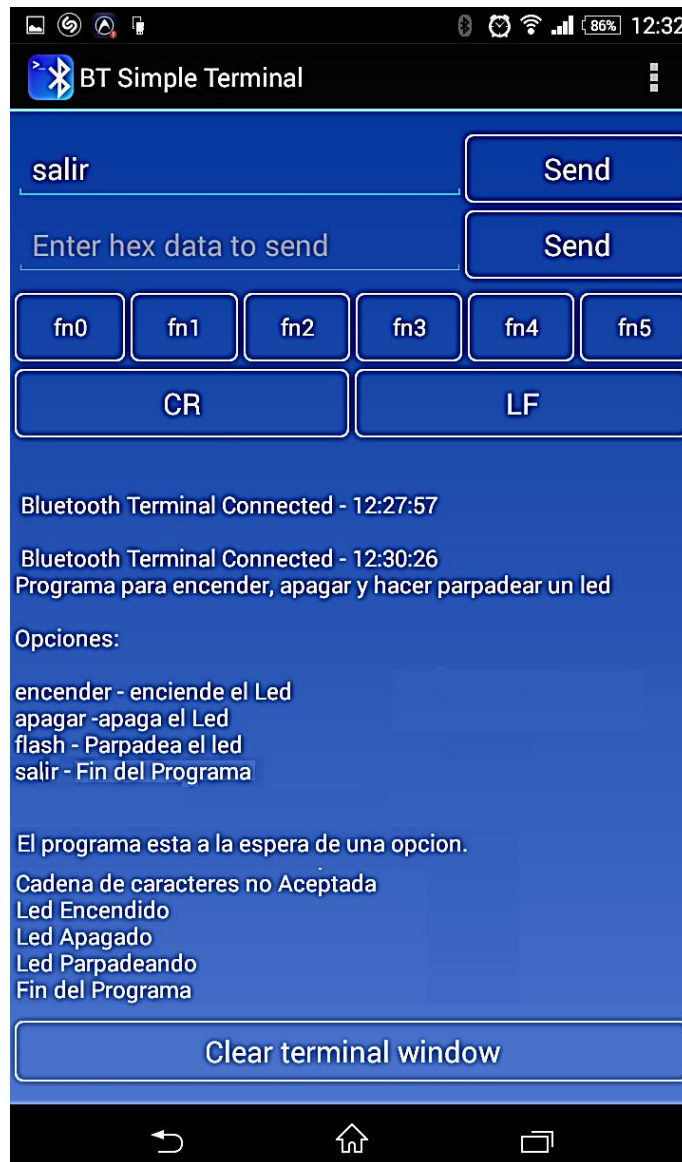


Figura 4-59. Bluetooth Terminal en ejecución.

Cabe mencionar que si deseamos compilar el programa nosotros mismos, podemos transcribir el código que se encuentra en el apéndice 7 en un nuevo proyecto de Eclipse. En la figura 4-60 podemos observar un diagrama de flujo que describe el funcionamiento de este programa. Como podemos observar, la variable “inuart” es la encargada de guardar la cadena de caracteres proveniente de nuestro dispositivo bluetooth. Después esta variable es comparada para ejecutar una acción. También podemos observar que si enviamos una cadena de caracteres que no corresponde a ninguna de las acciones a ejecutar, el programa envía un aviso.

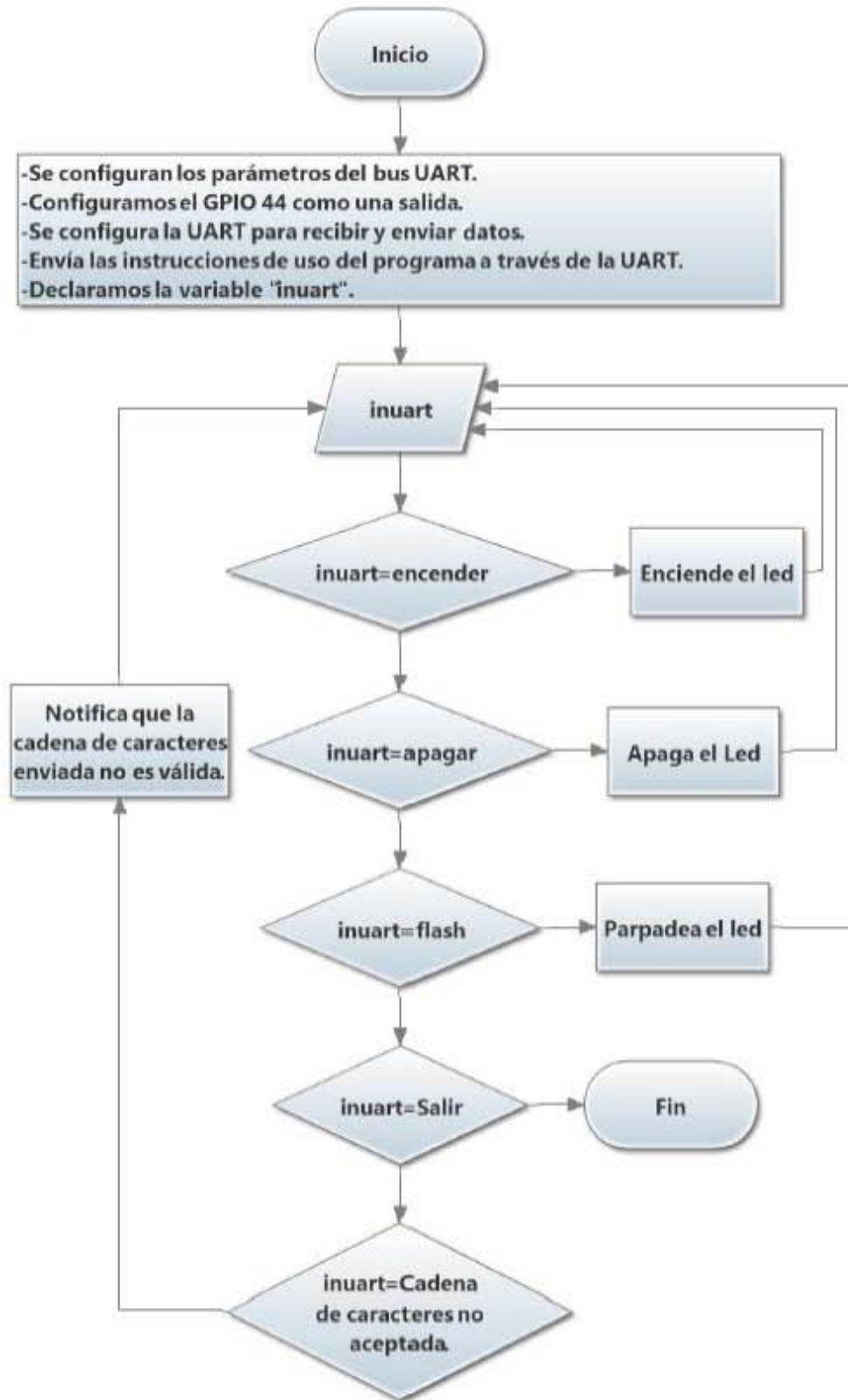


Figura 4-60. Diagrama de flujo de funcionamiento del programa que utiliza la UART4.

Uso de un bus de comunicaciones I²C.

I²C (Inter-Integrated Circuit) es un bus de comunicaciones serial ampliamente utilizado en sistemas embebidos para comunicar microcontroladores y sus periféricos. Este tipo de bus está diseñado para comunicar circuitos integrados entre sí que normalmente residen en un mismo circuito impreso, a diferencia del bus UART el cual está hecho para comunicar circuitos que no residen en la misma placa.

Una de las principales características de este bus es que utiliza dos líneas llamadas SDA y SCL para transmitir la información. La línea de transmisión SDA (Serial Data Line) es utilizada para enviar y recibir datos mientras que la línea SCL (Serial clock line) es una señal de reloj destinada para sincronizar el flujo de datos. Otra de las características principales de estas líneas es que son de drenador abierto³³ por lo que necesitan resistencias Pull-up.

Un bus I²C es capaz de interconectar múltiples dispositivos en sus líneas de comunicación formando una pequeña red de dispositivos. Cada red de dispositivos I²C está formada por un dispositivo maestro y múltiples dispositivos esclavos. El dispositivo maestro es el encargado de enviar la señal de reloj a todos los dispositivos esclavos, también se encarga de la escritura y la lectura de datos en los dispositivos esclavos. Cabe mencionar que en ocasiones se llega a pensar que los dispositivos esclavos son capaces de enviar información hacia otros dispositivos, ya sea un dispositivo maestro o un esclavo, lo cual es totalmente falso. Para enviar u obtener información de un dispositivo esclavo, el dispositivo maestro tiene que acceder al dispositivo esclavo para escribir o leer la información que tiene alojado en algún espacio de memoria como por ejemplo, un registro.

Todos los dispositivos conectados a un bus I²C cuentan con una dirección única (esta dirección suele ser expresada en hexadecimal) con la cual pueden ser identificados. Algunos dispositivos cuentan con dos direcciones diferentes para leer y escribir datos. En la figura 4-61 observamos un esquemático de cuatro dispositivos conectados en un bus I²C donde un microcontrolador es el dispositivo maestro de otros tres nodos esclavos (un ADC, un DAC, y otro microcontrolador).

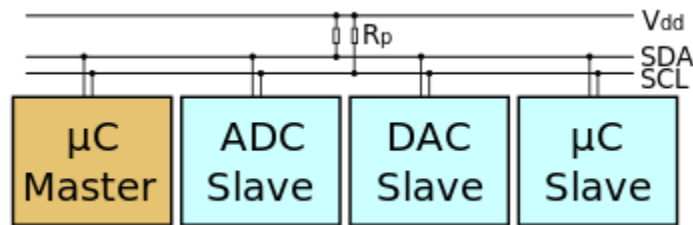


Figura 4-61. Esquemático de una red de dispositivos I²C.

En cuanto al protocolo que maneja la interfaz I²C, podemos decir que es el mismo para todos los dispositivos pero presenta algunas diferencias dependiendo del dispositivo que estemos utilizando. En este caso hablaremos del protocolo I²C enfocado al CI PCF8574 de Texas Instruments, el cual está presente en el módulo de la figura 4-60. Este módulo es una interfaz

³³ Los dispositivos con colector o drenador abierto son un tipo de dispositivos cuya salida está abierta o sin resistencia en el colector o drenador del transistor de salida. Esta configuración es empleada cuando se desean interconectar varios dispositivos en un mismo bus de datos.

serial para el display 1602A, el cual nos servirá para ejecutar un programa que implementa el uso de la interfaz I²C del Beaglebone.



Figura 4-62. Interfaz serial para el display 1602A.

Como se mencionó anteriormente, la interfaz serial de la figura 4-62 está compuesta por el CI PCF8574³⁴, es un registro de corrimiento que recibe los datos en serie de un bus I²C para posteriormente expresar el dato recibido en modo paralelo a través de 8 de sus pines. Este dispositivo también tiene la capacidad de obtener un dato paralelo para que una interfaz I²C pueda leerlos y obtener los datos en modo serial. En la figura 4-63 podemos observar el esquema de las señales con las que cuenta el PCF8574, de las cuales hablaremos a continuación.

Las señales SDA y SCL corresponden al bus I²C, que deben tener conexión con el dispositivo maestro (el PCF8574 solo puede trabajar como esclavo). Las señales A0, A1 y A2 son las encargadas de modificar la dirección de lectura y escritura del PCF8574 (en la interfaz serial que estamos utilizando ya viene predefinida la dirección 0x27, sin embargo podemos modificarla soldando las conexiones que están debajo del potenciómetro azul). Las señales P0 a P7 son las que expresan nuestro dato recibido en forma paralela, estas señales son bidireccionales ya que también podemos obtener datos expresados en paralelo para posteriormente expresarlos en modo serial. La señal "INT" está destinada para ser utilizada como interrupción cada que haya un cambio de estado en alguno de los pines P0 a P7 para que un procesador o microcontrolador tenga acceso al PCF8574 y pueda leer el dato que aparezca en los pines P0 a P7 en forma paralela.

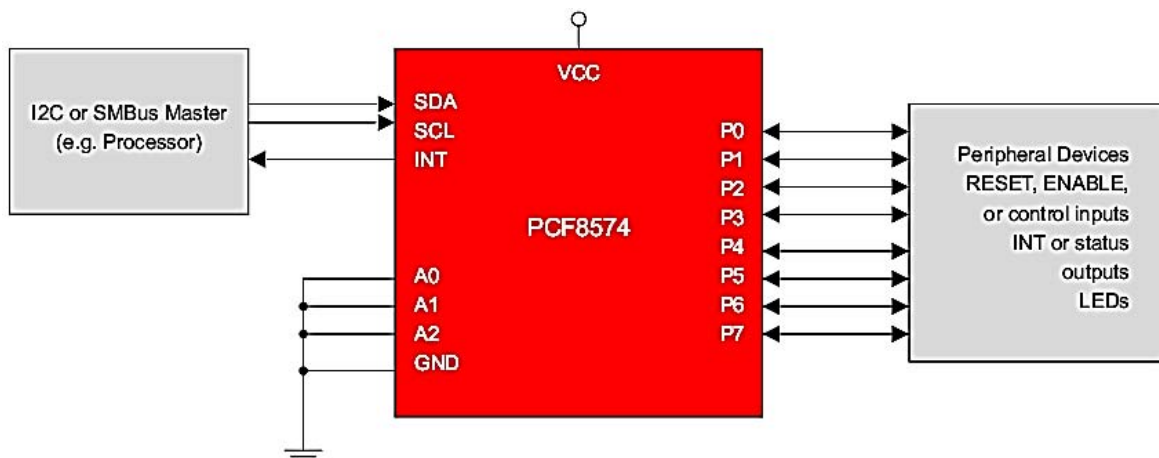


Figura 4-63. Esquema con las señales del PCF8574 [28].

³⁴ Para obtener información adicional sobre los datos técnicos de este CI podemos consultar su hoja técnica en: [28]

En la figura 4-64 podemos observar un diagrama de tiempos que describe cómo se realiza el acceso y la escritura de un dato en un bus I²C. Cuando el bus I²C está en reposo, la señal SDA se encuentra en estado alto, siendo un estado bajo emitido por SDA del dispositivo maestro, la condición de inicio de los dispositivos esclavo. Aproximadamente un ciclo de reloj después de que fue emitido el bit de inicio (todos los bits toman un ciclo de reloj para ser enviados), se manda el byte que contiene la dirección del dispositivo al que queremos tener acceso iniciando el envío con el bit más significativo. Después de mandar el byte con la dirección, se manda el bit denominado R/W, el cual define si queremos leer o escribir un dato en el dispositivo al que estamos accediendo (0 para escribir y 1 para leer). Antes de iniciar la transmisión del primer byte de datos se manda un bit llamado “ACK”, el cual se encargará de separar los bytes de datos que vaya enviando el dispositivo maestro, tal como se observa en la figura 4-62. Cabe mencionar que en algunos dispositivos más complejos como acelerómetros o giroscopios primero se manda la dirección del dispositivo, después el registro en el cual queremos escribir un dato y por último se envía el dato que queremos guardar en el registro que especificamos.

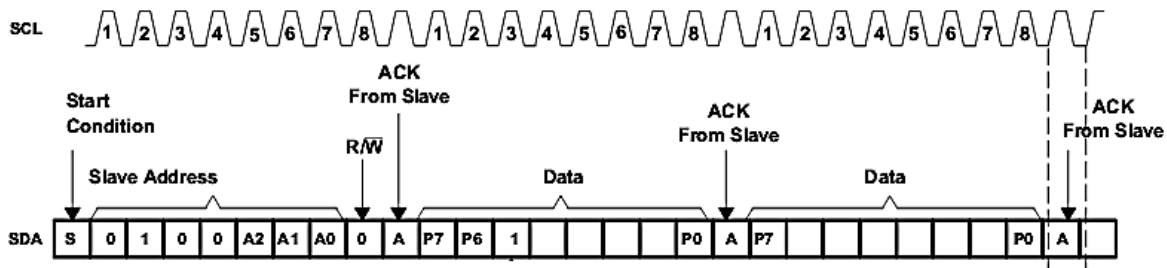


Figura 4-64. Mapa de tiempos para el envío de datos hacia un dispositivo esclavo [28].

En este trabajo solo hablaremos del modo escritura y no del modo lectura ya que para la interfaz serial que utilizamos sólo se utiliza para la escritura de los datos que queremos expresar en forma paralela³⁵.

Beaglebone cuenta con dos buses I²C, los cuales pueden ser activados fácilmente con ayuda de la librería BlackLib. Algunas señales de los buses I²C están presentes en dos pines diferentes, por ejemplo, la señal SDA del bus I2C2 está presente en los pines P9_20 y P9_22. Podemos verificar la distribución de pines de los buses I²C en la figura 8 del anexo 1.

Para mostrar el uso de una interfaz I²C con nuestra tarjeta de desarrollo se ha creado un programa en C++ con ayuda de Eclipse y BlackLib. Este programa manda una secuencia de bytes a través del bus I²C, posteriormente estos bytes llegan a nuestra interfaz serial para ser enviados por cada uno de los pines del display y así poder configurarlo y mostrar una cadena de caracteres.

Uno de los principales problemas al que nos tuvimos que enfrentar fue que no pudimos encontrar alguna fuente de información que nos proporcionara el diagrama de conexiones de la interfaz serial, por tal motivo se tuvo que revisar la continuidad entre cada uno de los pines del bus serial del PCF8574 y los pines que corresponden a las señales del display. Después de haber verificado la

³⁵ Si deseamos saber cómo opera el modo lectura consulte la página 14 del manual del PCF8574 en : [28]

continuidad en los pines, se llegó a la conclusión de que la conexión entre el PCF8574 y los pines que corresponden a las señales del display están configurados como se muestra la figura 4-65. Cabe mencionar que es muy importante conocer cómo está interconectado el PCF8574 con los pines de nuestro display ya que a partir de esta conexión se definieron los datos que son enviados a través del bus I²C del Beaglebone. Otro dato importante a tomar en cuenta es que el pin P7 es el bit más significativo, mientras que el bit P0 es el menos significativo.

El programa que se ha hecho para este ejemplo envía datos hexadecimales que son expresados en los pines P0 a P7 del PCF8574 para configurar el display empleando sólo 4 bits³⁶. Posteriormente se envían los datos necesarios para poder mostrar en pantalla la cadena de caracteres “HOLA-UNAM”. En el apéndice 9 podemos encontrar una tabla con los Bytes que son enviados por este programa para configurar el display y mostrar en pantalla la cadena de caracteres “HOLA-UNAM”.

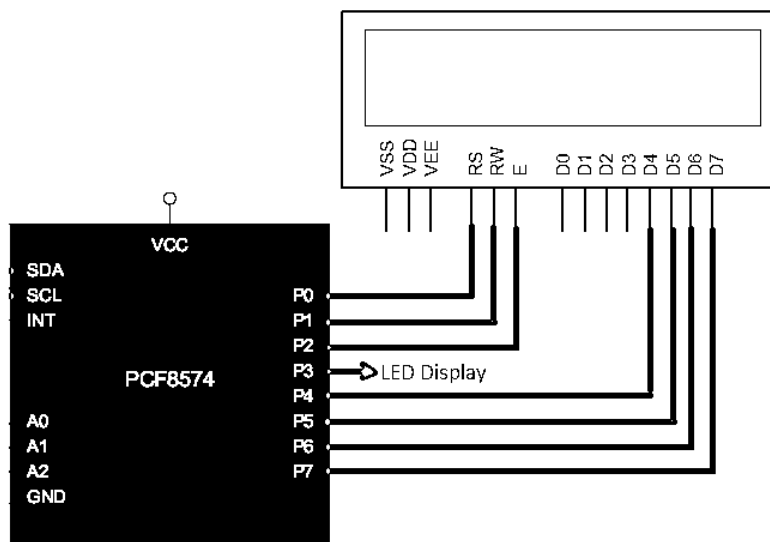


Figura 4-65. Conexión de las señales del PCF8574 con las señales del display en la interfaz serial.

Para poder ejecutar el programa que hemos creado para observar el funcionamiento del bus I²C y de la interfaz serial, debemos armar el circuito de la figura 4-66.

Una vez que hayamos terminado de armar el circuito mostrado en la figura 4-66 procederemos a conectarnos vía SSH con nuestro Beaglebone para poder ejecutar el programa con los códigos que se muestran en la figura 4-67. Al igual que en los programas anteriores se muestran los pasos para cambiar a la carpeta donde se almacena el programa y cambiar los derechos para correr el archivo ejecutable. Este programa se encuentra en el repositorio que fue descargado de GitHub. Al igual que todos los programas anteriores, también podemos crear un nuevo proyecto en Eclipse y escribir el código que se encuentra en el apéndice 8, posteriormente compilarlo y copiarlo a nuestro Beaglebone.

³⁶ Si deseamos saber cómo configurar el display 1602A empleando 4 bits consulte: [29]

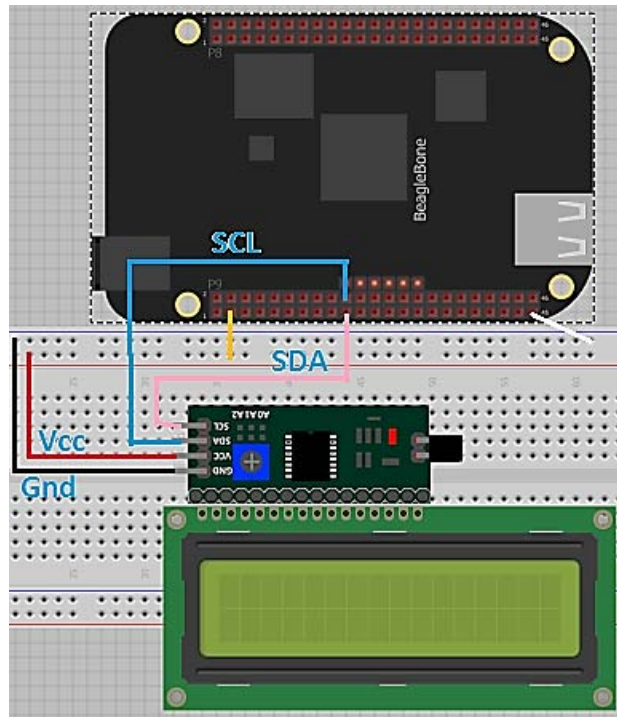


Figura 4-66. Conexión de la interfaz serial con el Display 1602A y el Beaglebone.

```

alberto@debian: ~
Archivo Editar Ver Buscar Terminal Ayuda
root@beaglebone:~# cd Tesis/Capitulo_4/Us0_i2c/
root@beaglebone:~/Tesis/Capitulo_4/Us0_i2c# ls
i2c_example  i2c_example.cpp
root@beaglebone:~/Tesis/Capitulo_4/Us0_i2c# chmod a+x i2c_example
root@beaglebone:~/Tesis/Capitulo_4/Us0_i2c# ./i2c_example

Programa de muestra de la interfaz I2C con el Display 1602A
root@beaglebone:~/Tesis/Capitulo_4/Us0_i2c#

```

Figura 4-67. Programa para el uso de una interfaz I²C.

En la figura 4-68 incluimos el diagrama de flujo que describe detalladamente el funcionamiento del programa.

Una vez que hayamos ejecutado el programa debemos poder observar la cadena de caracteres "HOLA-UNAM". Tal como se aprecia en la imagen 4-69. Cabe mencionar que en este caso si hemos incluido una imagen del resultado final del bus que estamos analizando, a diferencia de algunos casos anteriores, en los cuales no se incluyó ya que para visualizar el resultado final necesitaríamos un video que muestre el resultado (Por ejemplo, el uso de los ADC).



Figura 4-68. Diagrama de flujo que describe el funcionamiento del programa

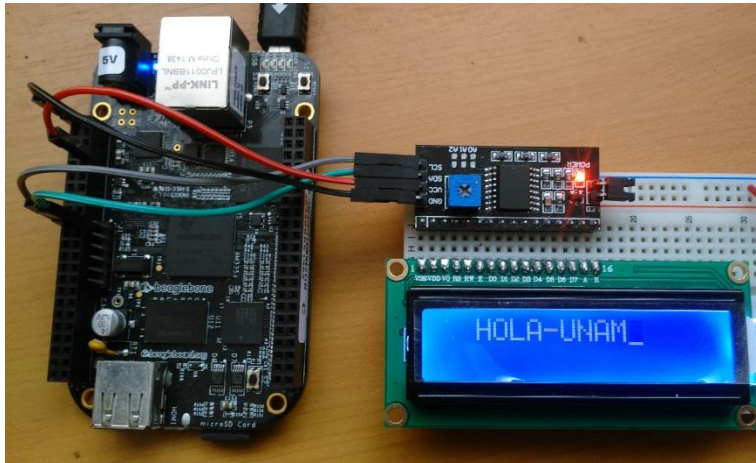


Figura 4-69. Programa en ejecución del ejemplo I²C.

Uso de la interfaz SPI a través del kernel de Linux.

En esta sección implementaremos el uso del bus SPI (Serial Peripheral Interface) a través de la terminal de comandos con el fin de mostrar un ejemplo de cómo podemos activar y desactivar los diferentes buses de nuestra tarjeta de desarrollo a través de la terminal de comandos manipulando algunos archivos del kernel³⁷ del sistema operativo del Beaglebone. Cuando hablamos de las salidas PWM se mostró de qué manera podríamos controlarlas desde la terminal de comandos, ya que los archivos del kernel que controlan las salidas PWM ya existen por defecto en el Beaglebone, lo cual no sucede para todos los dispositivos, como es el caso de los buses SPI. En esta sección observaremos cómo crear nuevos archivos de kernel para poder activar y desactivar un bus del sistema.

Para utilizar el bus SPI mediante la terminal de comandos se implementó el uso de un registro de corrimiento (CI 74HC595), el cual va conectado a unos LEDs que nos mostrarán en binario los datos que sean enviados en formato hexadecimal desde la terminal de comandos.

Antes de hablar de crear nuevos archivos de kernel y conectar nuestro circuito, debemos conocer cómo se configura y cómo funciona el protocolo de comunicación SPI.

Un bus SPI es principalmente utilizado para la transmisión de datos entre circuitos integrados en equipos electrónicos. A pesar de que el protocolo de comunicación SPI es muy viejo, hoy en día es un protocolo de comunicación que se sigue utilizando, ya que está diseñado para poder sostener una comunicación con casi cualquier dispositivo que cuente con un flujo de bits en serie regulados por una señal de reloj.

Una de las principales ventajas que presenta el bus SPI, es que cuenta con una velocidad de transmisión mayor que la de un bus I²C, además cuenta con un protocolo flexible que no está limitado a transmitir bloques de 8 bits. Cabe mencionar que aunque cuenta con muy buenas ventajas frente al bus I²C, el bus SPI es menos empleado hoy en día ya que requiere más pines de comunicación con otros dispositivos y no hay control de flujo por hardware como en el protocolo I²C.

³⁷ El kernel es un software fundamental en un sistema operativo el cual se encarga principalmente de dar acceso seguro al hardware de la computadora.

Un dispositivo SPI se puede interconectar con uno o varios dispositivos esclavo, tal como se muestra en la figura 4-70. A diferencia del bus I²C, el bus SPI no requiere de resistencias Pull up en sus líneas de comunicación, además, si queremos comunicarnos con un dispositivo esclavo, no contamos con una dirección de dispositivo como con el protocolo I²C, sino que sólo se emplea una señal que se encarga de habilitar la comunicación con los demás dispositivos. A continuación se muestran las señales con las que cuenta un bus SPI:

- SCLK (System Clock): Es el pulso que marca la sincronización.
- MOSI (Master Output Slave Input): Salida de datos del Maestro y entrada de datos al Esclavo.
- MISO (Master Input Slave Output): Salida de datos del Esclavo y entrada al Maestro.
- SS (Select): Activar la comunicación con un dispositivo esclavo.

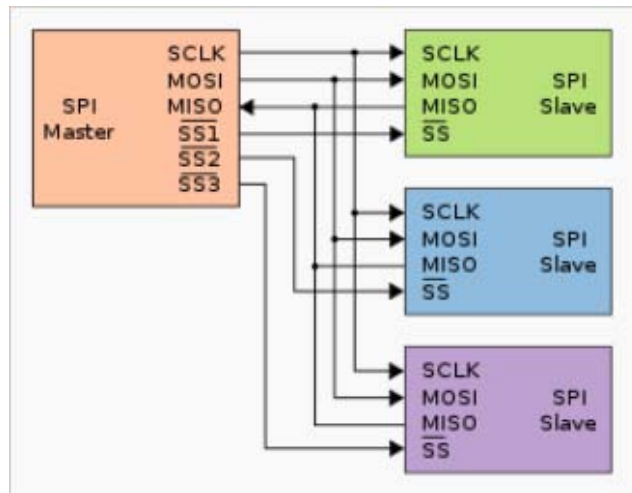


Figura 4-70. Interconexión con un dispositivo maestro y tres esclavos.

En cuanto al protocolo de funcionamiento podemos decir que es más complejo que otros protocolos como el I²C o el que utiliza un bus UART, ya que este protocolo cuenta con cuatro modos de funcionamiento, los cuales se adaptan al funcionamiento de un gran número de dispositivos como registros de corrimiento o dispositivos que necesiten de una señal de reloj para coordinar la transferencia de datos. Como el registro de corrimiento que utilizaremos más adelante, es de 8 bits, explicaremos el protocolo de funcionamiento en sus 4 modalidades con una trama de datos de 8 bits.

Dentro de los cuatro modos de funcionamiento únicamente varían dos parámetros para tener transferencia de datos. El primero de estos parámetros es la polaridad de la señal de reloj que emplea un dispositivo para poder enviar y recibir un dato. El segundo parámetro es la fase de la señal portadora de los bits de datos. En la figura 4-71 podemos observar un diagrama de tiempos que nos será de ayuda para explicar los cuatro modos de funcionamiento que mencionaremos a continuación.

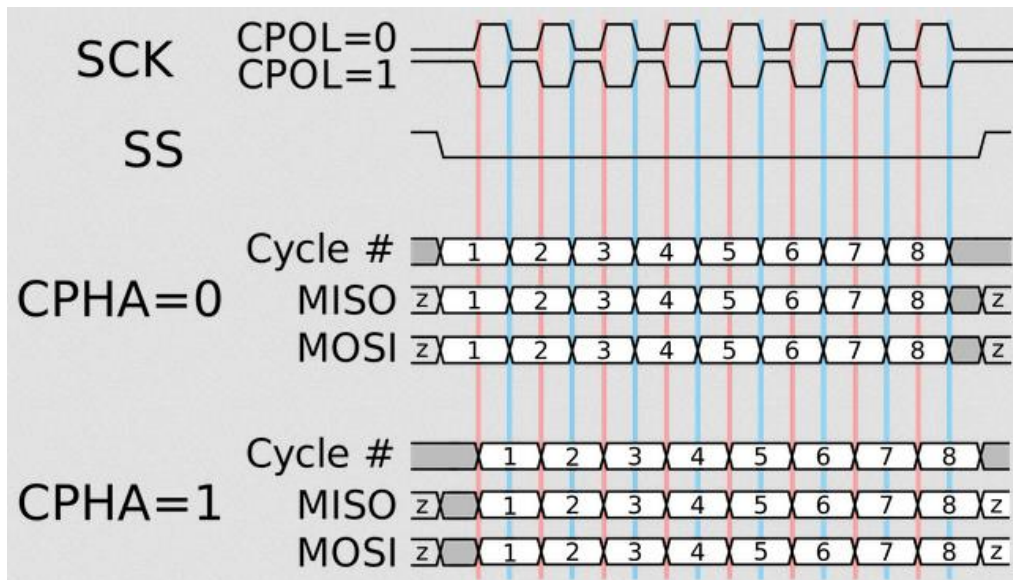


Figura 4-71. Diagrama de tiempos del Protocolo SPI [30].

En la figura 4-71 podemos observar dos señales de reloj denominadas CPOL (Clock polarity) y dos ejemplos de señales de datos denominadas CPHA (Cycle Phase), las cuales nos indicarán cada uno de los modos de funcionamiento que se describen a continuación.

- Modo 0: CPOL=0 y CPHA=0

En este modo el estado de reposo del reloj es un estado bajo. Cuando se inicia la comunicación entre dispositivos (un estado bajo en la señal SS) el dispositivo que recibe los datos procede a leer un dato recibido cada que se presenta una transición de estado bajo a estado alto, tal como lo indican las líneas rosas de la figura 4-71.

- Modo 1. CPOL=0 y CPHA=1.

En este modo el estado de reposo del reloj es un estado bajo. Cuando se inicia la comunicación entre dispositivos (un estado bajo en la señal SS) el dispositivo que recibe los datos procede a leer un dato recibido cada que se presenta una transición de estado alto a estado bajo, tal como lo indican las líneas azules de la figura 4-71.

- Modo 2. CPOL=1 y CPHA=0.

En este modo el estado de reposo del reloj es un estado alto. Cuando se inicia la comunicación entre dispositivos (un estado bajo en la señal SS) el dispositivo que recibe los datos procede a leer un dato recibido cada que se presenta una transición de estado alto a estado bajo, tal como lo indican las líneas rosas de la figura 4-71.

- Modo 3. CPOL=1 y CPHA=1.

En este modo el estado de reposo del reloj es un estado alto. Cuando se inicia la comunicación entre dispositivos (un estado bajo en la señal SS) el dispositivo que recibe los datos procede a leer un dato recibido cada que se presenta una transición de estado bajo a estado alto, tal como lo indican las líneas azules de la figura 4-71.

Beaglebone cuenta dos buses SPI que son capaces de trabajar en los cuatro modos de funcionamiento. En la figura del 7 del anexo 1 podemos observar la distribución de pines de los dos buses SPI, los cuales son denominados SPI0 y SPI1. En la nomenclatura que se utiliza en Beaglebone para referirse a las señales de los buses SPI podemos observar que la señal MOSI es denominada "D1", la señal MISO es denominada "D0", las señales SS son denominadas "CS" y la señal de reloj es denominada SCLK. Por ejemplo, el pin P9_18 tiene una nomenclatura denominada SPI0_D1, lo cual significa que este pin es la señal MOSI del bus SPI0.

Antes de iniciar la modificación y creación de nuevos archivos del kernel del sistema, se recomienda armar el circuito de la figura 4-72, ya que la creación de nuevos archivos de kernel no demora mucho tiempo, además necesitamos tener todo armado cuando iniciemos a mandar los datos en hexadecimal a través del bus SPI empleando la terminal de comandos.

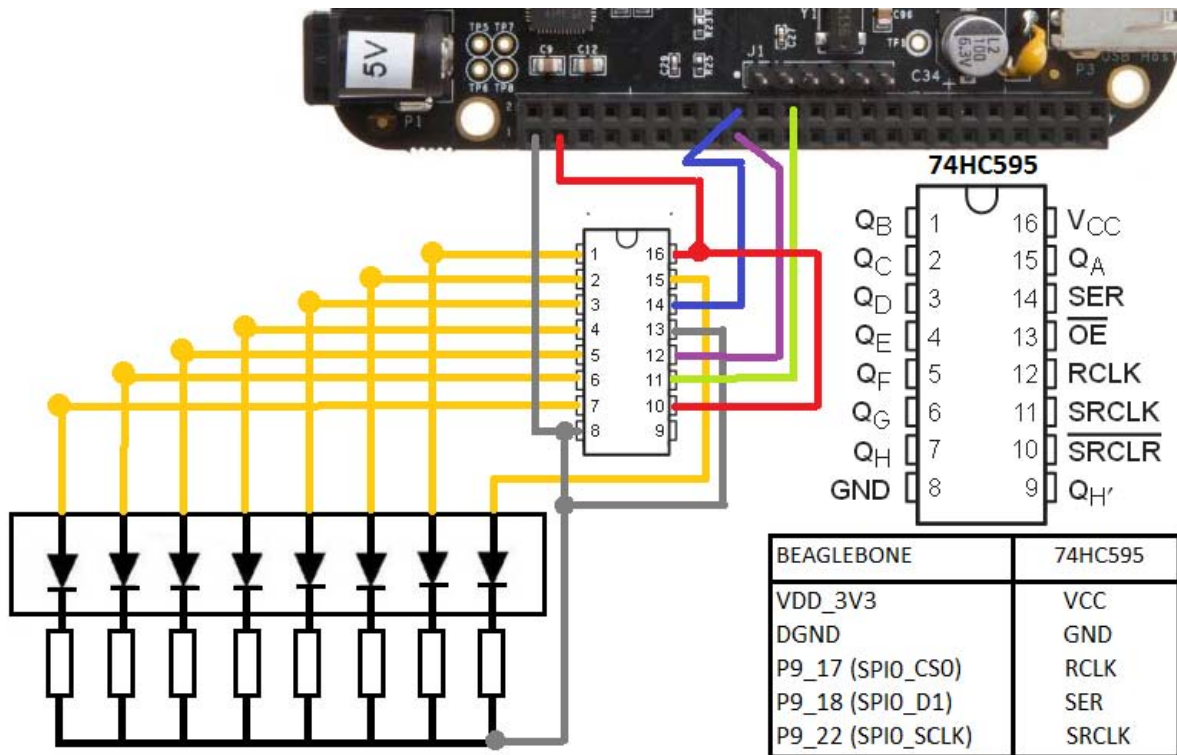


Figura 4-72. Registro de corrimiento conectado al Beaglebone.

El 74HC595³⁸ es un CI que contiene un registro de corrimiento de 8 bits, el cual está directamente conectado a un registro de almacenamiento que consta de 8 flip flops tipo D. Este CI es compatible con el protocolo SPI en modo 0, lo cual lo hace un dispositivo ideal para mostrar el funcionamiento del bus SPI del Beaglebone. A continuación explicaremos cuál es la función de cada una de las señales de control del 74HC595 (antes de continuar debemos observar y analizar el diagrama compuertas lógicas):

- SER : Recibe la señal portadora de los datos en modo serial, por tal razón se conecta con la señal MOSI del Beaglebone.
- \overline{OE} : Señal de control de compuertas de tercer estado en la salida de cada bit serial (QA...QH). Como queremos tener siempre activa la salida de datos en modo serial, se conecta a GND.
- SRCLK : Reloj de los flip flops del registro de corrimiento. Se conecta directamente al reloj maestro, ósea al reloj del Beaglebone.
- \overline{SRCLR} : Señal "Clear" de cada uno de los flip flops del registro de corrimiento. En este caso como no deseamos borrar ningún dato en los flip flops, se conecta directamente a Vcc.
- RCLK : Reloj de los flip flops tipo D del registro de almacenamiento. Como queremos obtener en la salida de cada flip flop el mismo estado que en la entrada, se conecta a CS0, el cual envía un estado bajo que es convertido en estado alto por el inversor que hay en la entrada de cada flip flop.

Cabe mencionar que la señal MISO (SPI10_D0) del Beaglebone no será utilizada ya que no recibiremos ningún dato por parte del dispositivo esclavo (74HC595), solo enviaremos datos en hexadecimal para poder visualizarlos en los LEDs. Otro dato importante a mencionar es que en el bus de datos paralelo, el bit QA es el menos significativo, mientras que el bit QH es el más significativo.

Después de armar el circuito y entender a detalle el funcionamiento del bus SPI y de nuestro registro de corrimiento, procederemos a crear un nuevo archivo en el kernel de Beaglebone, que nos permita activar el bus SPI0 y mandar datos en hexadecimal para poder visualizarlos en los LEDs conectados en el bus paralelo del 74HC595. Los pasos para crear archivos de kernel descritos a continuación también los podemos encontrar en el anexo número 5.

El primer paso que debemos hacer es tener acceso a la terminal del Beaglebone via SSH o escritorio remoto. Para crear nuestros archivos de kernel primero crearemos una carpeta llamada "SPI", posteriormente nos moveremos a la carpeta creada ingresando los siguientes comandos:

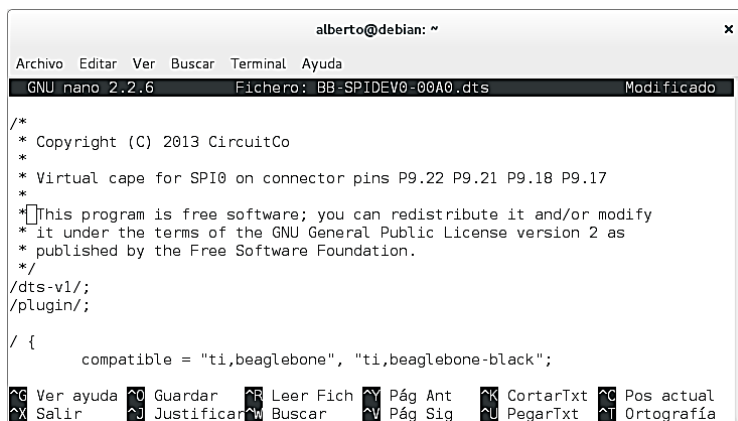
```
Mkdir SPI
cd SPI
```

El siguiente paso será crear un archivo llamado "BB-SPIDEV0-00A0.dts", el cual contendrá líneas de comandos que nos ayudarán a activar los archivos de kernel. Para crear el archivo "BB-SPIDEV0-00A0.dts", debemos ingresar el siguiente comando:

³⁸ Para poder observar el diagrama de compuertas lógicas, se recomienda consultar la página número 3 de la hoja de datos técnicos en: [31].

```
nano BB-SPIDEV0-00A0.dts
```

Al ingresa el comando mencionado anteriormente, nos aparecerá una ventana de texto como la de la siguiente figura, en la cual debemos escribir el código que se encuentra en el anexo número 10³⁹ y en el repositorio que descargamos de GitHub, en la ubicación “Tesis/Capitulo_4/SPI”. Para guardar el código y salir del editor de texto, debemos presionar CTRL+O y CTRL+X respectivamente.



```
alberto@debian: ~
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
GNU nano 2.2.6  Fichero: BB-SPIDEV0-00A0.dts  Modificado
/*
 * Copyright (C) 2013 CircuitCo
 *
 * Virtual cape for SPI0 on connector pins P9.22 P9.21 P9.18 P9.17
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */
/dts-v1/;
/plugin/;

/ {
    compatible = "ti,beaglebone", "ti,beaglebone-black";
}

Ver ayuda  Guardar  Leer Fich  Pág Ant  CortarTxt  Pos actual
Salir      Justificar  Buscar     Pág Sig  PegarTxt   Ortografía
```

Figura 4-73. Creando el archivo “BB-SPIDEV0-00A0.dts”.

Los archivos de tipo “dts” (device tree system) son archivos de texto que contiene comandos que permiten manipular el hardware de una computadora. Al compilar los archivos con extensión “dts”, se crean los archivos de kernel (archivos con extensión “dtbo”) que utiliza un sistema Linux para manipular el hardware.

Una vez que hayamos creado el archivo “BB-SPIDEV0-00A0.dts” procederemos a compilarlo con el siguiente comando:

```
dtc -O dtb -o BB-SPIDEV0-00A0.dtbo -b 0 -@ BB-SPIDEV0-00A0.dts
```

Con el comando anterior se creó un archivo llamado “BB-SPIDEV0-00A0.dtbo”, el cual debemos copiar a la ubicación “/lib/firmware/” ingresando el siguiente comando:

```
cp BB-SPIDEV0-00A0.dtbo /lib/firmware/
```

Para que el sistema operativo reconozca el archivo “BB-SPIDEV0-00A0.dts” como un nuevo archivo que forma parte del kernel del sistema debemos ingresar el siguiente comando:

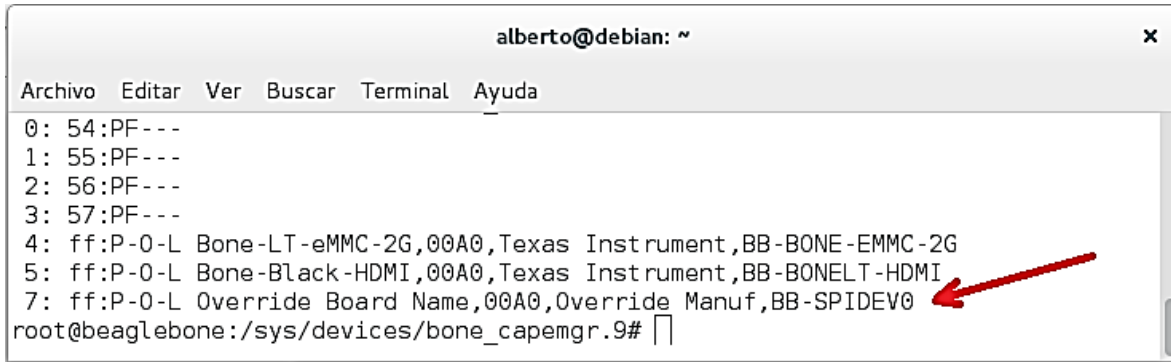
```
echo BB-SPIDEV0 > /sys/devices/bone_capemgr.9/slots
```

Para verificar que el sistema reconoce el archivo “BB-SPIDEV0-00A0.dts” como un archivo de kernel y que el SPI0 se activó correctamente ingresaremos el siguiente comando:

```
cat /sys/devices/bone_capemgr.9/slots
```

³⁹ Este código también lo podemos encontrar en: <https://github.com/jadonk/capefirmware/blob/master/arch/arm/boot/dts/BB-SPIDEV0-00A0.dts>

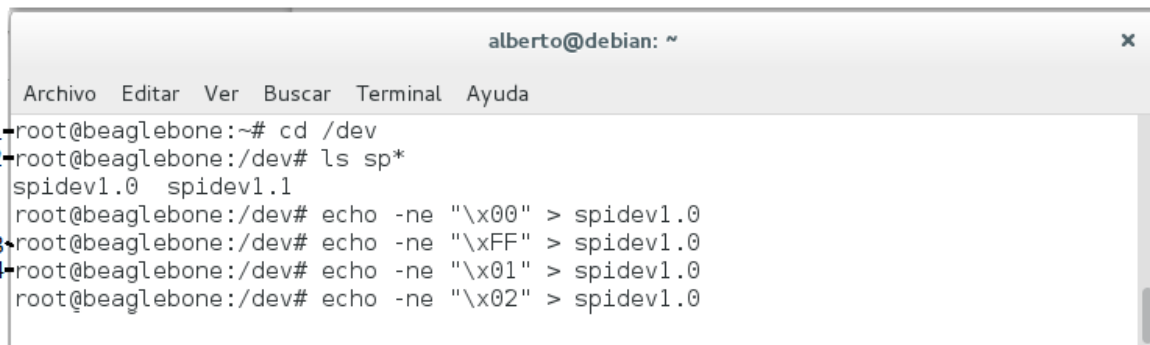
Al ingresar el comando anterior observaremos en la terminal un cuadro de texto como el que se muestra en la siguiente figura. Para saber que el SPI0 se activó correctamente, debemos poder observar un renglón como el que esta indicado con una flecha roja en la siguiente figura.



```
alberto@debian: ~
Archivo Editar Ver Buscar Terminal Ayuda
0: 54:PF---
1: 55:PF---
2: 56:PF---
3: 57:PF---
4: ff:P-0-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
5: ff:P-0-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
7: ff:P-0-L Override Board Name,00A0,Override Manuf,BB-SPIDEV0
root@beaglebone:/sys/devices/bone_capemgr.9#
```

Figura 4-74. Verificando que el bus SPI0 está activo.

Después haber activado la SPI0 con los pasos descritos anteriormente, procederemos a enviar los datos en hexadecimal. En la figura 4-75 se muestra una terminal con los comandos necesarios para enviar datos en formato hexadecimal. Con el comando 1 nos cambiamos a una carpeta llamada “dev”, la cual contiene los archivos que nos permiten enviar datos por el bus SPI. Con el comando 2 nos cercioramos de que los archivos que nos permiten enviar los datos estén presentes. Por último, con los comandos 3 y 4 enviamos los números en forma hexadecimal. Como se puede apreciar en la figura 4.75, hemos enviado los números hexadecimales 0x00, 0xFF, 0x01 y 0x02.



```
alberto@debian: ~
Archivo Editar Ver Buscar Terminal Ayuda
1-root@beaglebone:~# cd /dev
2-root@beaglebone:/dev# ls sp*
spidev1.0 spidev1.1
root@beaglebone:/dev# echo -ne "\x00" > spidev1.0
3-root@beaglebone:/dev# echo -ne "\xFF" > spidev1.0
4-root@beaglebone:/dev# echo -ne "\x01" > spidev1.0
root@beaglebone:/dev# echo -ne "\x02" > spidev1.0
```

Figura 4-75. Comandos para enviar datos en forma hexadecimal.

Capítulo 5. Procesamiento de Imágenes.

5.1 Hardware compatible para la captura de imágenes.

En el capítulo 1 mencionamos que con nuestra tarjeta de desarrollo podemos implementar a cualquier proyecto características propias de una computadora. Gracias a que Beaglebone cuenta con un sistema operativo podemos instalar aplicaciones e incluso usar algún tipo de API⁴⁰ que nos facilite el trabajo a la hora de querer implementar a un proyecto dispositivos de una complejidad superior como una cámara de video o una tarjeta de audio USB. La compatibilidad de software que tiene nuestra tarjeta es la misma que tiene un sistema operativo Linux para PC, sin embargo, la única limitante que tendríamos son los requerimientos de procesador, memoria de video y memoria gráfica que demanden las aplicaciones que queramos instalar en nuestro Beaglebone. En este capítulo implementaremos el uso de una webcam con la finalidad de probar la verdadera potencia de Beaglebone con aplicaciones de procesamiento de imágenes y transmisión de video vía Streaming (del cual se hablará más adelante).

⁴⁰ Una API (Application Programming Interface) es un conjunto de bibliotecas o aplicaciones destinadas para cumplir una o muchas funciones con el fin de ser utilizadas por otro software. Para este trabajo las API utilizadas nos servirán para tener comunicación entre componentes de software y componentes de hardware.

Para poder procesar video e imágenes con nuestra Beaglebone se ha decidido utilizar la webcam Logitech c920 ya que con esta cámara podemos obtener imágenes y video desde resoluciones muy bajas hasta llegar a tener capturas de foto y video en HD (High Definition), lo cual la hace ideal para poder poner a prueba nuestro Beaglebone. Cabe mencionar que podemos utilizar otro tipo de webcam compatible con sistemas operativos Linux, sin embargo, los procedimientos y programas empleados para este trabajo fueron destinados específicamente para la webcam C920, esto no significa que ningún otro tipo de webcam sea compatible con los programas y procedimientos que mostraremos en este capítulo⁴¹. En la figura 5-1 podemos observar la webcam Logitech c920.



Figura 5-1. Logitech C920.

Logitech C920 cuenta con una resolución máxima de 15 megapíxeles. Además cuenta con una función de autoenfoco similar a la de los teléfonos inteligentes actuales, lo cual le da una calidad de imágenes muy superior a las webcam promedio. En cuanto a características de video, esta cámara es capaz de grabar video en resolución HD (resolución de 1280 x 720 píxeles).

5.2 Implementación de una aplicación para capturar video e imágenes.

Hoy en día existen varias aplicaciones compatibles con sistemas Linux que nos podrían servir para capturar imágenes y video, sin embargo, como mencionamos anteriormente, el problema no radica en la compatibilidad del sistema operativo sino en la cantidad de memoria RAM y la potencia del microprocesador que requieran las aplicaciones para poder ejecutarlas. Para demostrar que al tener un sistema Linux en nuestro Beaglebone podemos instalar cualquier aplicación que está destinada para una computadora, se buscó una aplicación que fuese capaz de grabar video y capturar fotos pero que además no requiriera un microprocesador muy poderoso o demasiada memoria RAM. Después de buscar por muchos sitios de internet y foros encontramos una aplicación llamada “Guvvview”.

⁴¹ Podemos encontrar una lista de webcam compatibles con SO similares al del Beaglebone en: [32]

Gvvcview es una sencilla, pero poderosa aplicación de software libre que nos permite grabar video y fotos en varios tipos de resolución incluyendo HD (1280x720 pixeles) y Full HD (1920x1080 pixeles). Para poder instalar Gvvcview debemos abrir la terminal del beaglebone mediante una conexión ssh y escribir el comando número 6 del anexo 3.

Para poder abrir esta aplicación debemos tener acceso a nuestro Beaglebone a través del entorno gráfico de usuario del sistema operativo. Para visualizar el entorno gráfico de usuario podemos conectar el Beaglebone a un televisor empujando la salida HDMI o tener acceso desde una computadora vía escritorio remoto, como se explicó en el capítulo 3.

Una vez que hayamos tenido acceso al entorno gráfico de usuario procederemos a dar clic al menú de inicio de la esquina inferior izquierda, después iremos al menú llamando "Sound & Video" y posteriormente abriremos la aplicación Gvvcview, tal como se muestra en la figura 5-2.

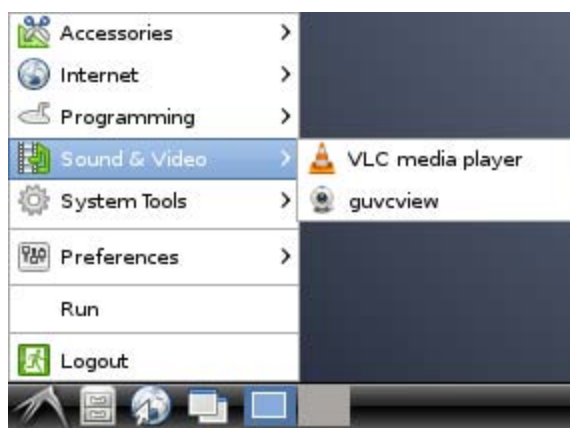


Figura 5-2. Gvvcview instalada correctamente.

Cuando abrimos Gvvcview nos aparecerá una ventana como la de la figura 5-3 en la cual podemos modificar una gran cantidad de parámetros para la captura de foto y video, entre las cuales está la resolución de la imagen o del video. En la parte inferior de la ventana principal del programa encontramos los iconos para capturar video, tomar y abrir o guardar una foto o video, entre otras cosas.

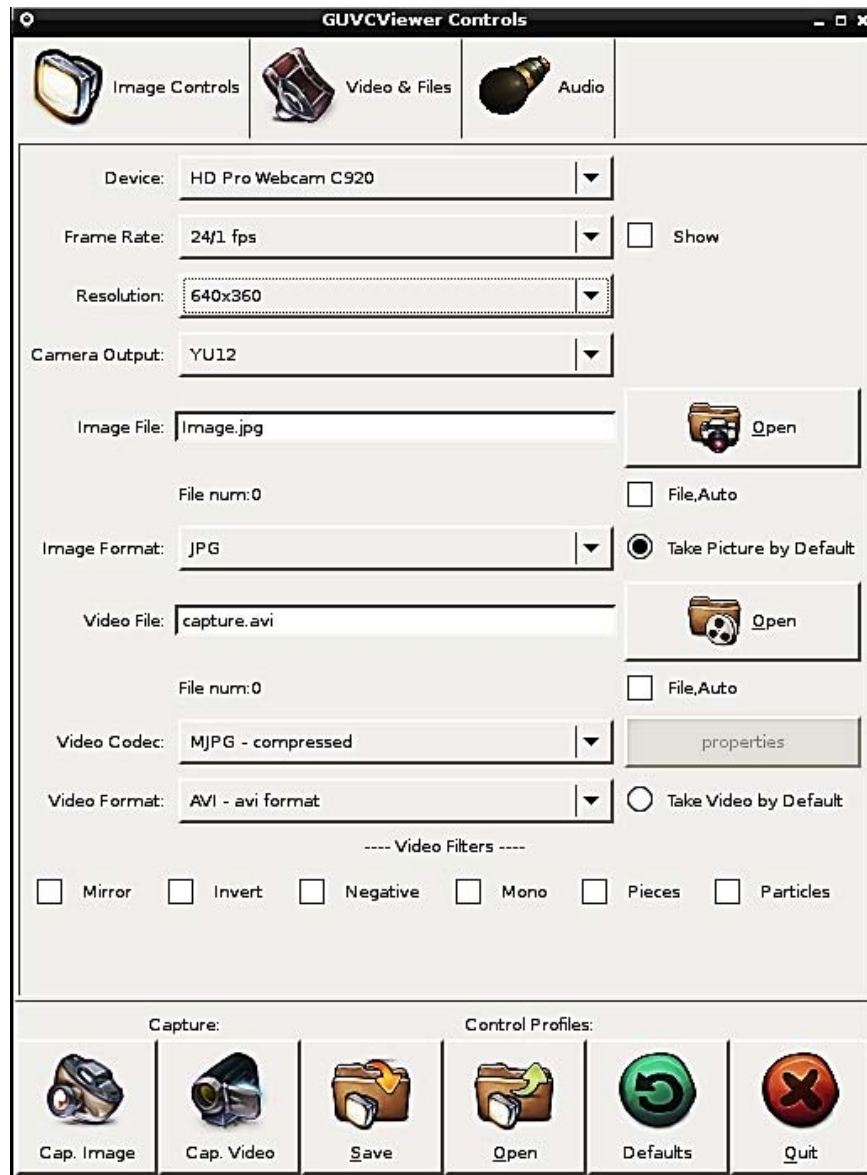


Figura 5-3. Ventana de Guvview.

Al experimentar con este programa tomando imágenes y video pudimos observar que cuando quisimos tomar una fotografía en HD hubo problemas de ralentización⁴² en nuestra tarjeta de desarrollo. También pudimos ver que la captura de video en HD fue bastante mala, sin embargo la reproducción del video es considerablemente buena ya que no tiene pausas provocadas por ralentización del procesador. Para poder reproducir un video en nuestro Beaglebone debemos instalar el reproductor VNC⁴³ escribiendo en la terminal el comando 4 del anexo número 3.

Por último decidimos bajar la resolución de captura de foto y video a una resolución de 640x360 pixeles. A esta resolución, el desempeño en la captura de foto y video es bastante bueno, además

⁴² Se refiere a la disminución o desaceleración en la ejecución de tareas y procesos del sistema operativo debido a la sobrecarga de operaciones en el procesador.

⁴³ VLC es un reproductor multimedia libre y de código abierto multiplataforma que reproduce la mayoría de archivos multimedia y diversos protocolos de transmisión de video [33].

la calidad de la imagen con respecto al color, brillo y contraste es bastante buena gracias a la cámara que estamos empleando. En la figura 5-4 se muestra una imagen de una foto tomada por nuestro Beaglebone.

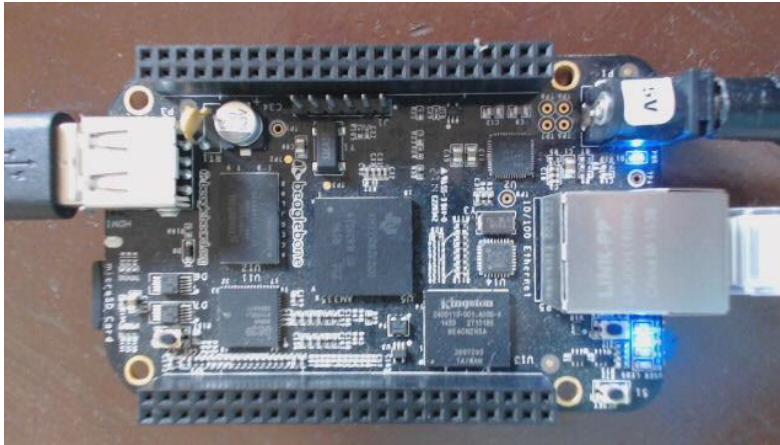


Figura 5-4. Imagen capturada por Beaglebone con la webcam Logitech C920.

5.3 Captura de imágenes empleando V4l2.

Con la aplicación que utilizamos anteriormente podemos capturar foto y video, sin embargo, para poder utilizar esa aplicación es necesario tener acceso a la interfaz gráfica de usuario ya que no es posible modificar ninguna de sus configuraciones desde la terminal, lo que significa que este programa no puede ser usado por otros programas para obtener imágenes y video. Para poder capturar imágenes y video necesitamos instalar una API llamada “V4l2”, la cual nos permite modificar sus parámetros y opciones desde la terminal. Cabe mencionar que cualquier aplicación que nos permita modificar sus parámetros y opciones desde la terminal, podemos considerarla una API ya que mediante un lenguaje de programación como C++ podemos tener acceso a la terminal de Linux y ejecutar comandos.

V4l2 es una API de captura de video e imágenes para los sistemas operativos Linux. Es muy usada actualmente ya que tiene soporte para la gran mayoría de webcams compatibles con sistemas operativos basados en Linux y además es necesaria para poder ejecutar otro tipo de software como Open CV, del cual hablaremos más adelante. Para poder instalar V4l2 y todas las herramientas necesarias para esta sección debemos abrir la terminal del Beaglebone e ingresar los comandos 7 y 8 del anexo número 3.

Para probar que V4l2 fue instalada correctamente tomaremos una foto ingresando a la terminal los comandos que se muestran en la figura 5-5. Con el comando 1 de la figura 5-5 cambiaremos a la carpeta donde se encuentra el archivo que se encarga de tomar la foto⁴⁴, el cual se encuentra en el repositorio de GitHub hecho para este trabajo. Con el comando número 2 de la figura 5-5 procederemos a tomar la imagen. Al capturar una imagen nos debe de aparecer en la consola algunos avisos como los que se muestran debajo del segundo código que ingresaremos.

⁴⁴ Este archivo es sólo un código que configura V4l2 para capturar una imagen. Podemos crear este archivo abriendo un archivo de texto empleando el comando número 5 del anexo 2 y escribiendo las líneas del código del apéndice 11. Para más información acerca de este archivo de configuración consulte: [34]

```
192.168.1.111 - PuTTY
1 root@beaglebone:~# cd Tesis/Capitulo_5/fswebcam/
root@beaglebone:~/Tesis/Capitulo_5/fswebcam# ls
fswebcam.conf
2 root@beaglebone:~/Tesis/Capitulo_5/fswebcam# fswebcam -c fswebcam.conf
--- Opening /dev/video0...
Trying source module v4l2...
/dev/video0 opened.
--- Capturing frame...
Captured frame in 0.00 seconds.
--- Processing captured image...
Putting banner at the bottom.
Setting font to /usr/share/fonts/truetype/ttf-dajavu/DejaVusans.ttf.
Setting title "Exploring Beaglebone".
Setting timestamp "%H:%M:%S %d/%m/%Y (%Z)".
Setting output format to PNG, quality 0
Writing PNG image to 'exploringBB.png'.
root@beaglebone:~/Tesis/Capitulo_5/fswebcam#
```

Figura 5-5. Captura de una imagen empleando V4I2.

Para poder visualizar la imagen que hemos capturado debemos conectarnos al Beaglebone vía escritorio remoto. Después abrir el exportador de archivos y acceder a la ubicación Tesis/Capitulo_5/fswebcam, la cual contiene una imagen llamada “exploringBB.png”.

La imagen que hemos capturado se muestra en la figura 5-6. La resolución original de la imagen tomada es de 1280x720 pixeles. Cabe mencionar que no hubo problema alguno de ralentización cuando fue capturada esta imagen.



Figura 5-6. Imagen capturada con V4I2.

5.4 Procesamiento de imágenes empleando Open CV.

OpenCV es una biblioteca de código abierto de visión artificial desarrollada por Intel con la cual podemos realizar tareas de visión computarizada como reconocimiento facial, detección de movimiento, detección de objetos y colores, detección de figuras, entre otras. Open CV es multiplataforma, existiendo versiones para Linux, Mac y Windows. OpenCV contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión artificial. Open CV

pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente, realizando su código fuente en C y C++, aunque hoy en día podemos implementar OpenCV en otros lenguajes de programación como java y Python.

OpenCV es capaz de trabajar con cualquier webcam que sea compatible con el sistema operativo que estemos empleando. En este caso OpenCV es compatible con Beaglebone ya que cuenta con un sistema operativo basado en Linux. Otro punto importante es que OpenCV utiliza V4L2 para la captura de audio y video que posteriormente son procesados para realizar tareas de visión computarizada.

Antes de continuar con la demostración del uso de un programa que implemente la librería OpenCV, debemos instalar dicha librería en nuestro Beaglebone. Para poder instalar OpenCV debemos acceder a la terminal de comandos del Beaglebone usando una conexión SSH y posteriormente escribir los comandos del inciso 9 del anexo número 3.

Para mostrar cómo podemos emplear OpenCV en un código de C++ para tomar una foto y editarla de manera automática, se ha utilizado el código del programa⁴⁵ que se encuentra en el apéndice número 12 y en el repositorio que hemos descargado de GitHub. Este programa es capaz de tomar una foto y procesarla para obtener una imagen en escala de grises y una imagen donde sólo se observan los bordes de la imagen con líneas de color blanco. Con este programa podemos obtener tres archivos de imagen (la original, la de escala de grises y la de bordes) con una sola captura de imagen.

En la figura 5-7 podemos observar una serie de comandos que hemos ingresado a la terminal para poder ejecutar nuestro programa. Con el comando 1 de la figura 5-7 cambiamos a la carpeta que contiene el código de este programa, mientras que con el comando número 2 compilamos el programa para poder generar un archivo ejecutable⁴⁶. Por último ingresamos el comando 3 de la figura 5-7 para poder correr el programa. Cabe mencionar que este programa, a diferencia de todos los programas que hemos visto, fue desarrollado en la misma consola de comandos con ayuda del comando “nano” (ver comando 5 del anexo 2) y no con ayuda de un IDE como Eclipse.

Inmediatamente después de ejecutar el programa comienza el proceso de captura de imagen, el cual dura aproximadamente 2 segundos, después se inicia el procesamiento de la imagen capturada para obtener la imagen en escala de grises y la imagen de bordes blancos. En la figura 5-9 podemos observar un diagrama de flujo que describe detalladamente en funcionamiento del programa.

En la figura 5-8 podemos observar la imagen original y el resultado del procesamiento de la imagen original en escala de grises y en bordes. Las imágenes capturadas por este programa se guardan en la misma carpeta que contiene el código y el archivo ejecutable de este programa (Tesis/Capitulo_5/openCV), por lo tanto, debemos tener una conexión de escritorio remoto para poder visualizar las imágenes.

⁴⁵ Este código fue tomado de <http://derekmolloy.ie/beaglebone-images-video-and-opencv/>. Se modificaron algunas configuraciones para tomar fotos a una resolución de 640 píxeles.

⁴⁶ Para compilar un programa en donde se incluye alguna librería de OpenCV consulte el comando número 28 del anexo 2.

```
192.168.1.111 - PuTTY
1 root@beaglebone:~# cd Tesis/Capitulo_5/openCV/
root@beaglebone:~/Tesis/Capitulo_5/openCV# ls
README.md boneCV.cpp build face face.cpp
2 root@beaglebone:~/Tesis/Capitulo_5/openCV# g++ -O2 `pkg-config --cflags --libs opencv` boneCV.cpp -o boneCV
root@beaglebone:~/Tesis/Capitulo_5/openCV# ls
README.md boneCV boneCV.cpp build face face.cpp
3 root@beaglebone:~/Tesis/Capitulo_5/openCV# ./boneCV
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
Processing - Performing Image Processing
Finished Processing - Saving images
root@beaglebone:~/Tesis/Capitulo_5/openCV#
```

Figura 5-7. Comandos para ejecutar el programa.

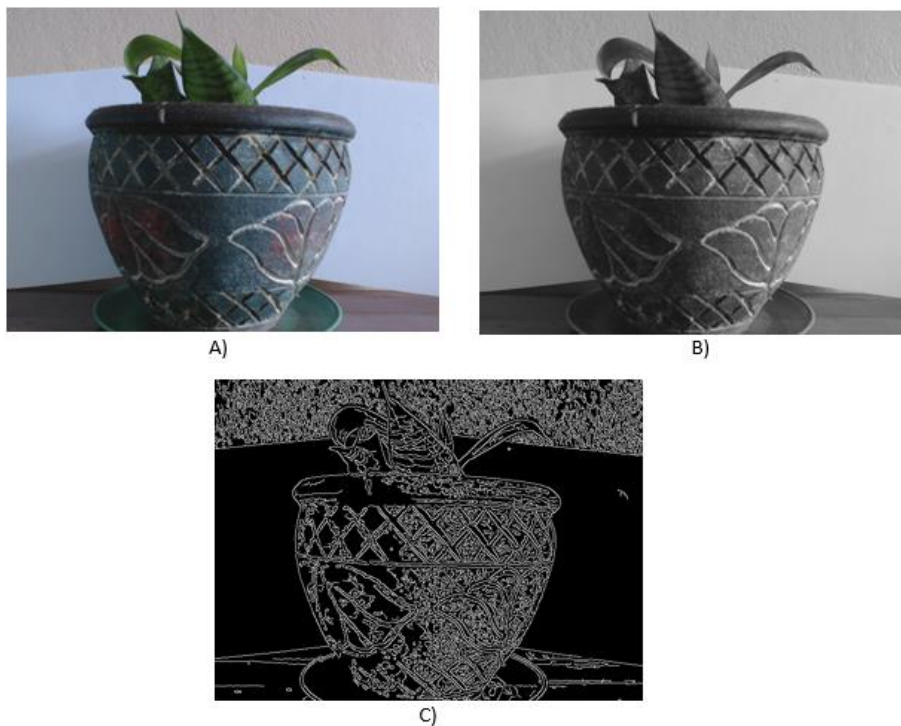


Figura 5-8. Resultado del procesado de una imagen.

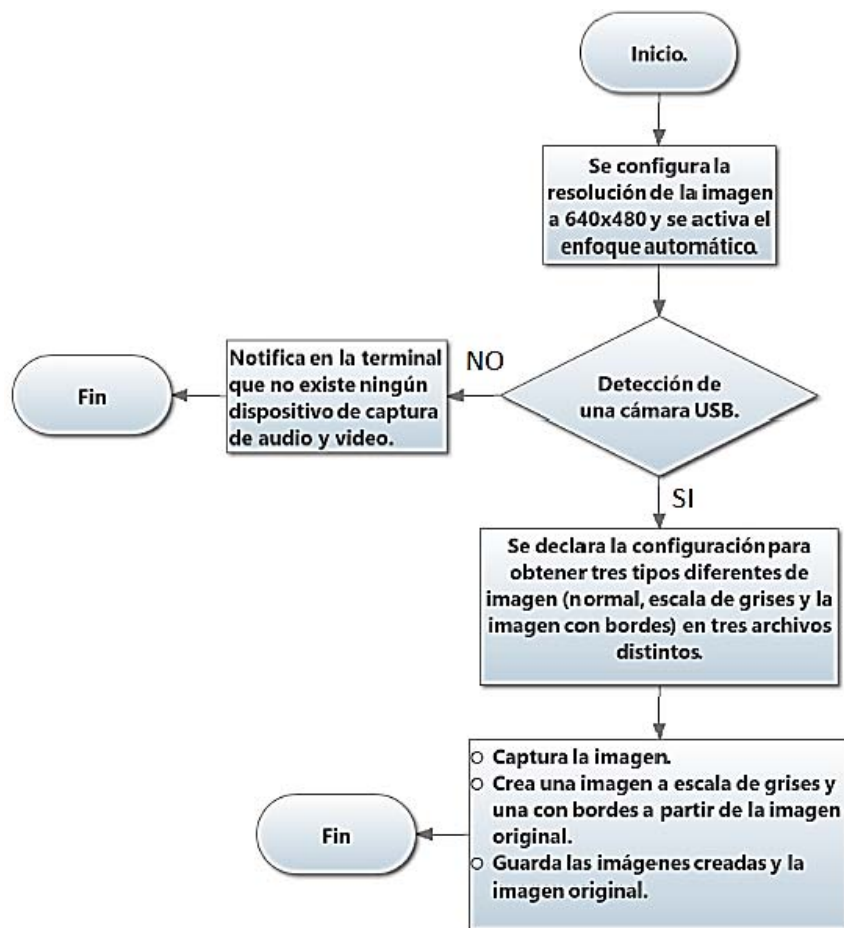


Figura 5-9. Diagrama de flujo que describe el funcionamiento del programa.

5.5 Detección de rostros empleando OpenCV.

Una de las principales ventajas que ofrece OpenCV es la capacidad de detectar objetos, formas y colores, rostros humanos e incluso patrones de movimiento. En esta sección se implementó un programa que es capaz de detectar rostros humanos, con la finalidad de mostrar el verdadero potencial con el que cuenta el Beaglebone para la implementación de proyectos de ingeniería en los cuales intervienen herramientas de software de procesamiento avanzado de imágenes.

En esta ocasión hablaremos primero del diagrama de flujo y del funcionamiento del programa ya que en este programa es más importante entender el funcionamiento del programa, a diferencia de algunos programas anteriores donde se consideró que era mejor correr el programa y después hablar de su funcionamiento.

Este programa, a diferencia de los programas anteriores, es capaz de recibir parámetros de configuración cuando se escribe el comando para ser ejecutado en la terminal⁴⁷. En el diagrama de flujo de la figura 5-10 podemos observar que nosotros podemos definir si queremos detectar

⁴⁷ Para entender más a detalle cómo funciona la programación de estos parámetros consulte: <http://c.conclase.net/curso/?cap=020c>

rostros en una imagen que está guardada en la misma carpeta que el archivo ejecutable o si deseamos detectar rostros de una imagen tomada por nuestra webcam. Si decidimos analizar una imagen que está guardada en la carpeta donde se encuentra el archivo ejecutable, el programa analiza la imagen y si detecta uno o varios rostros crea una imagen nueva remarcando los rostros con un círculo de color verde. Si a la hora de correr el programa no indicamos el nombre de algún archivo imagen para ser analizado, este procede a capturar una imagen de la webcam para posteriormente poder analizarla y remarcar los rostros detectados. Cuando se procesa una imagen (ya sea tomada de la webcam o almacenada en nuestro Beaglebone) y no se detectan rostros humanos, el programa notifica en la terminal que no se detectaron rostros humanos y finaliza el programa.

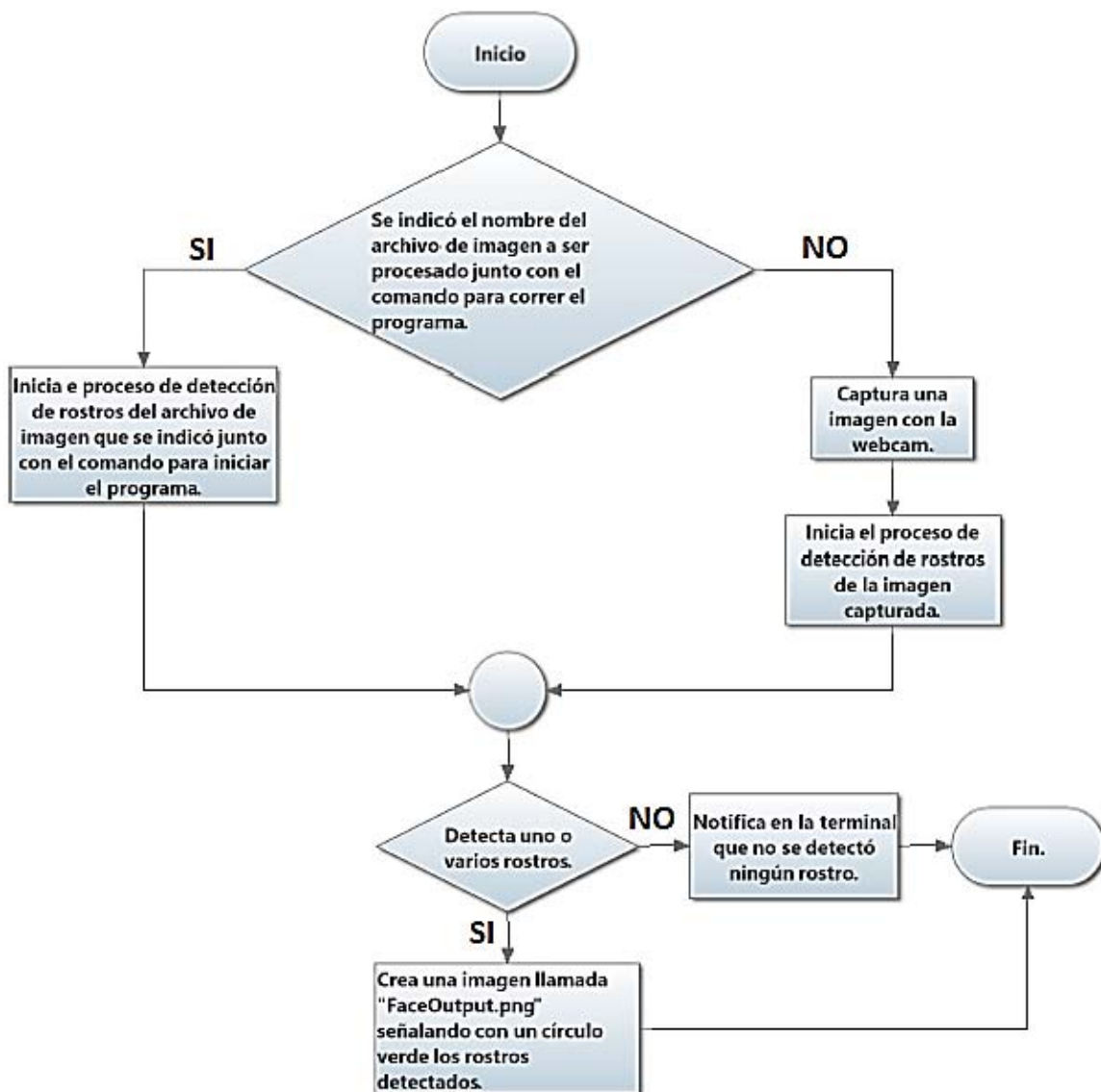
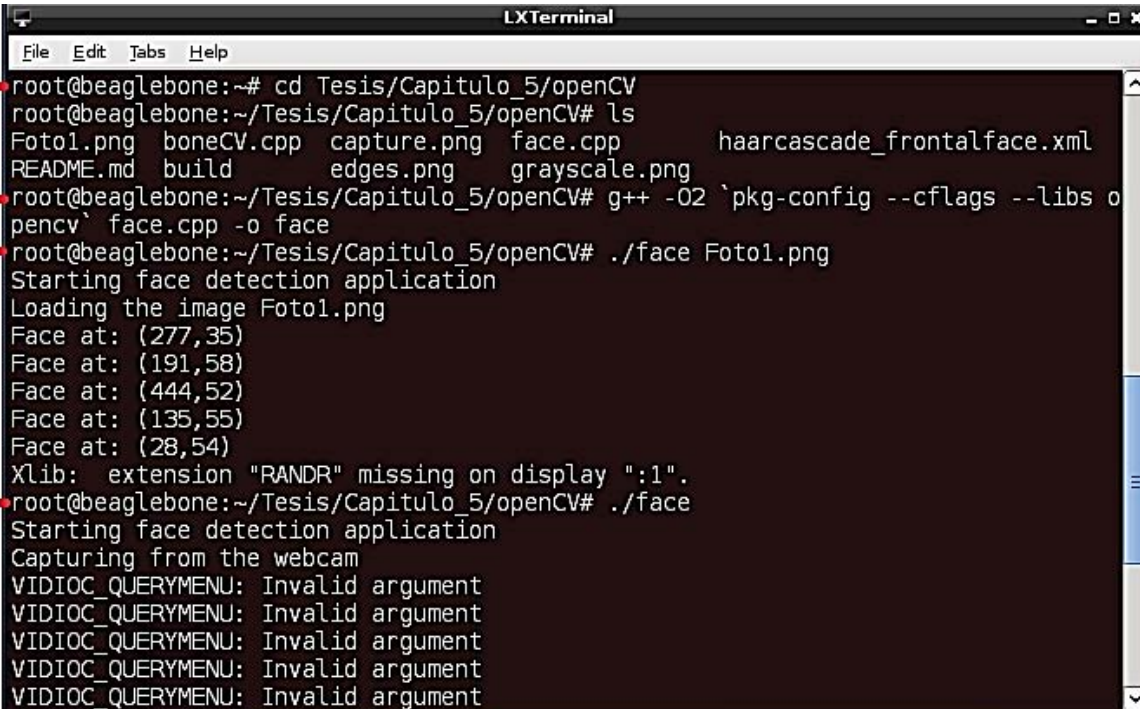


Figura 5-10. Diagrama de funcionamiento del programa.

Antes de correr el programa y mostrar un ejemplo con la terminal de comandos, se recomienda conectarnos al Beaglebone vía escritorio remoto y abrir la terminal de comandos dentro de

nuestro escritorio remoto ya que cuando se detectan rostros, el programa automáticamente abre el visor de imágenes mostrando la imagen analizada remarcando los rostros detectados. El código de este programa⁴⁸ lo podemos encontrar en el apéndice número 13 y en el repositorio que descargamos de GitHub, en la carpeta que llamada “Capítulo_5”.

En la figura 5-11 se muestra la terminal con los comandos que necesitamos para poder correr nuestro programa analizando una imagen que tenemos guardada y una imagen que fue tomada con nuestra webcam. Con el comando 1 de la figura 5-11 cambiamos a la carpeta donde se encuentra el programa, mientras que con el comando número 2 podemos compilar el programa ya que el código de éste fue escrito directamente en la terminal, tal como se explicó con el programa anterior. Con el comando 3 de la figura 5-11 corremos el programa indicándole que analice la imagen llamada “Foto1.cpp” mientras que con el comando número 4 corremos el programa para que tome una foto con la webcam y la analice para detectar rostros.



```
LXTerminal
File Edit Tabs Help
1 root@beaglebone:~# cd Tesis/Capitulo_5/openCV
root@beaglebone:~/Tesis/Capitulo_5/openCV# ls
Foto1.png boneCV.cpp capture.png face.cpp haarcascade_frontalface.xml
README.md build edges.png grayscale.png
2 root@beaglebone:~/Tesis/Capitulo_5/openCV# g++ -O2 `pkg-config --cflags --libs op
pencv` face.cpp -o face
3 root@beaglebone:~/Tesis/Capitulo_5/openCV# ./face Foto1.png
Starting face detection application
Loading the image Foto1.png
Face at: (277,35)
Face at: (191,58)
Face at: (444,52)
Face at: (135,55)
Face at: (28,54)
Xlib: extension "RANDR" missing on display ":1".
4 root@beaglebone:~/Tesis/Capitulo_5/openCV# ./face
Starting face detection application
Capturing from the webcam
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
```

Figura 5-11. Terminal con los comandos para correr el programa.

En la figura 5-12 se muestra la imagen llamada “Foto1.png”, la cual fue procesada para poder detectar rostros humanos. Como podemos ver, en algunas ocasiones se pueden llegar a generar errores en la detección de rostros ya que en la imagen podemos observar a cuatro personas y en la detección de rostros se crearon cinco círculos verdes.

En la figura 5-13 podemos observar una imagen tomada con la webcam y procesado para la detección de un rostro humano. En esta imagen no hubo errores ya que el rostro detectado ocupa

⁴⁸ Este código fue tomado de <http://derekmolloy.ie/beaglebone-images-video-and-opencv/>. Se modificaron algunas configuraciones para tomar fotos a una resolución de 800x600 pixeles.

un área muy grande de la foto. Otro punto importante a observar es que el círculo verde se adapta al tamaño aproximado del rostro detectado.



Figura 5-12. Detección de rostros en “Foto1”.

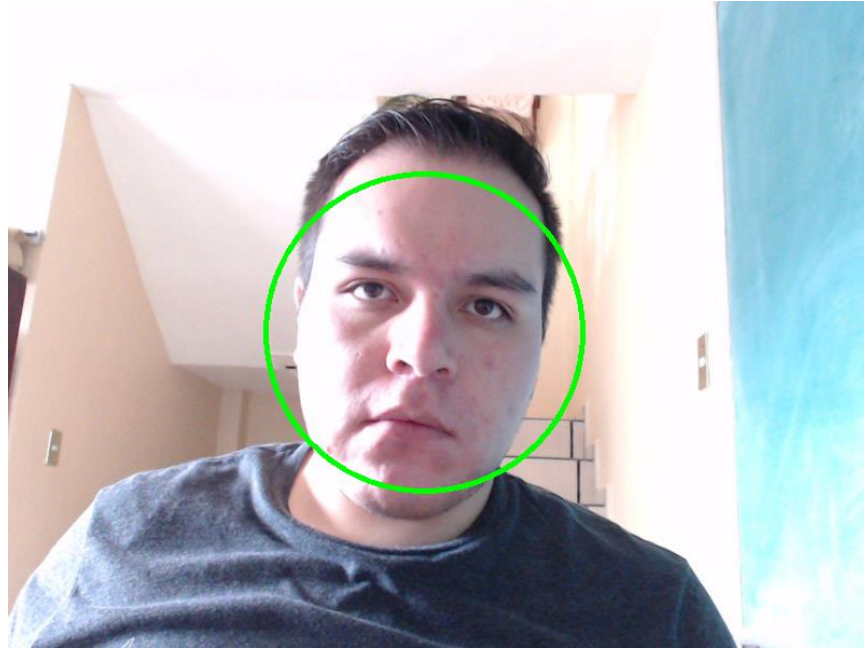


Figura 5-13. Detección de rostros de una imagen tomada por la webcam.

Capítulo 6. Interface para aplicaciones básicas de tiempo real.

6.1 PRU-ICSS.

La unidad PRU-ICSS (programmable Real-Time Unit and Industrial Communication Subsystem) consta de dos núcleos RISC destinados para ejecutar aplicaciones de tiempo real con los que cuenta el procesador AM3558. Estos núcleos son subsistemas independientes de la arquitectura del procesador ya que la línea de procesadores AM335x es una modificación de la arquitectura ARM para poder ejecutar aplicaciones de tiempo real, añadiendo núcleos que son capaces de ejecutar tareas propias de sistemas embebidos.

En la siguiente figura podemos observar un diagrama de bloques que nos muestra toda la arquitectura interna del AM3358. Las unidades de procesamiento de cada PRU están denominadas como PRU0 y PRU1, cada una cuenta con un bus de datos de propósito general (I/O), diversos tipos de memorias RAM y un bus que interconecta todos sus elementos.

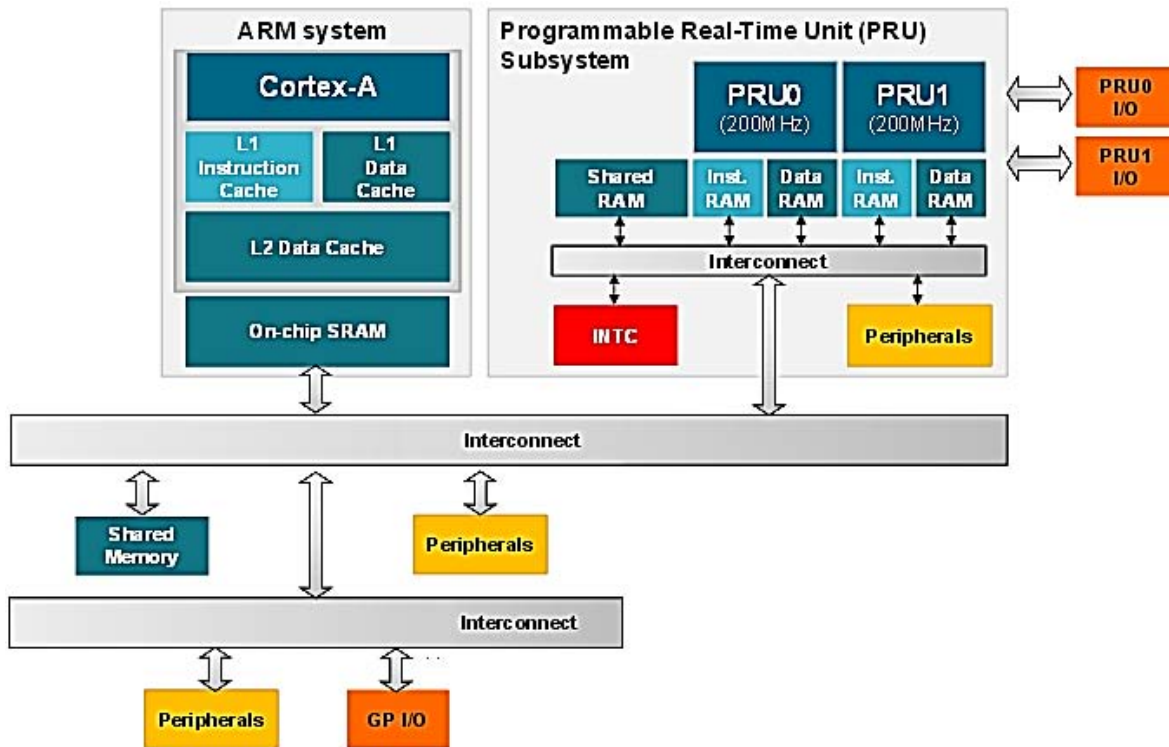


Figura 6-1-Diagrama de bloques de la integración de la arquitectura PRU a la arquitectura ARM [35].

Cada PRU puede ser considerado como un microcontrolador de 32 bits, los cuales poseen una frecuencia de reloj de 200MHz. Las PRU cuentan con una memoria RAM compartida (Shared RAM) para almacenar datos que se deseen compartir con la otra PRU, la cual tiene una capacidad de almacenamiento de 12KB. Además cuentan con una RAM propia (Data RAM) de 8KB para cada módulo. La memoria que se encarga de almacenar las instrucciones de ejecución de un programa (Inst. RAM) también es de tipo RAM, lo que significa que cada que apaguemos y encendamos nuestro Beaglebone debemos cargar los programas a esta memoria ya que no tiene la capacidad de retener datos cuando no hay suministro de energía eléctrica.

Los PRU son utilizados por el kernel de Linux para ejecutar los protocolos de comunicación de los buses UART, I²C y SPI. Como se vio en el capítulo 4, cada protocolo está sincronizado en el tiempo mediante su señal de reloj o mediante el paso del tiempo (caso del protocolo que utiliza el bus UART), por tal razón necesitan de una interface que sea capaz de enviar y recibir datos en tiempo real, ya que de lo contrario se perdería la sincronía de cada bit de datos y la comunicación sería imposible.

Algunas entradas y salidas de los GPIO del AM3358 están multiplexadas con las entradas y salidas de los PRU. Podemos tener acceso a las entradas y salidas de las PRU a través de los Pines P8 y P9 de nuestra tarjeta de desarrollo, siempre y cuando las PRU estén activas de modo que el usuario pueda disponer de ellas. Cabe mencionar que cuando utilizamos algún bus de comunicaciones como el bus UART o el Bus SPI, las PRU también se encuentran activas pero el usuario no puede disponer de ellas más que para lograr una comunicación con otro dispositivo.

6.2 Introducción al uso de los PRU-ICSS.

En esta sección se explicarán los pasos que se deben seguir para poder implementar el uso de los núcleos PRU y las fuentes de información necesarias para desarrollar alguna aplicación que involucre el uso de un PRU en un programa en C, ya que en las secciones posteriores sólo se explicará cómo ejecutar los programas de ejemplo. Cabe mencionar que antes de querer desarrollar nuestra propia aplicación que involucre el uso de un PRU, se recomienda leer todo este capítulo y poner en marcha los ejemplos que se describen más adelante. Para este capítulo sólo utilizaremos códigos desarrollados en C programados directamente en la terminal de comandos del Beaglebone, tal como se hizo en el capítulo 5.

Para poder disponer de los núcleos PRU se necesita seguir una serie de pasos que pueden resultar confusos ya que además de activar los PRU creando archivos de kernel del sistema, también debemos desarrollar código en C y código en el lenguaje ensamblador que fue creado para programar las PRU. En la figura 6-2 podemos observar una serie de pasos que se deben seguir para poder disponer de un PRU, los cuales se explicarán a continuación.

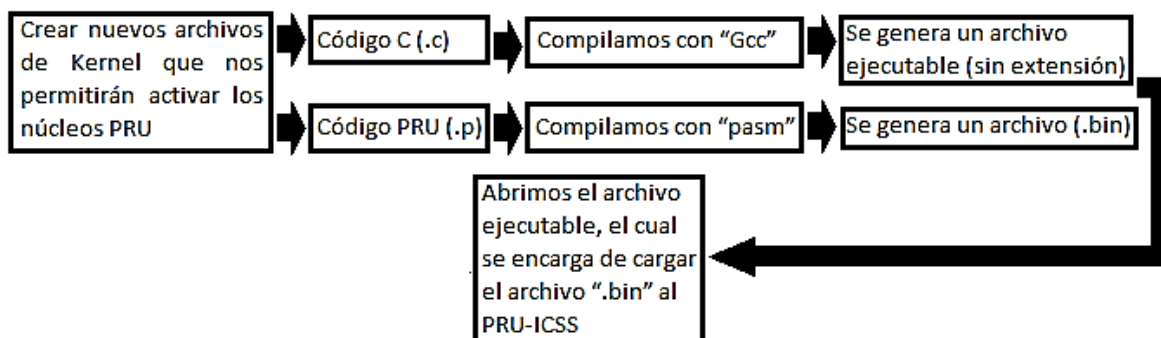


Figura 6-2. Pasos para el uso de la unidad PRU-ICSS.

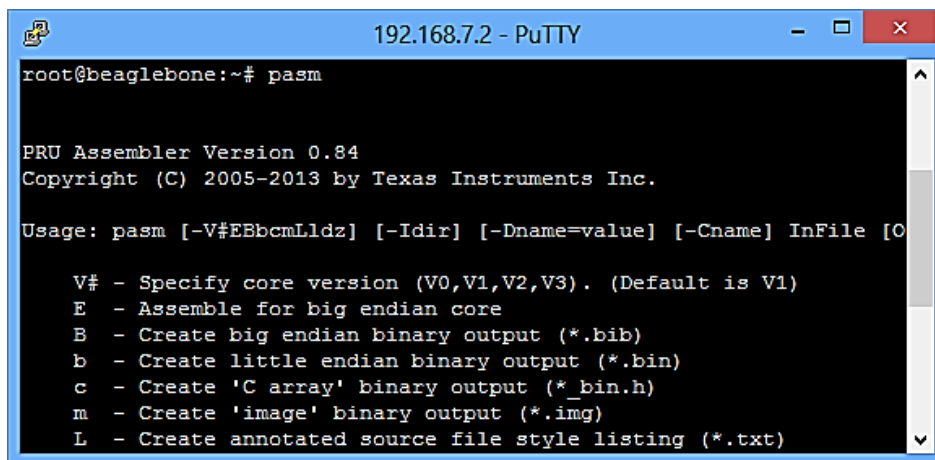
El procedimiento para crear archivos de kernel que nos permitan disponer de las PRU es el mismo que se usó en el ejemplo de uso de la interfaz SPI en el capítulo 4. Los archivos con extensión “.dts” vistos en el capítulo 4, son archivos que nos permiten configurar el modo de trabajo de todos los pines del Beaglebone. Los pines del Beaglebone tienen varios modos de funcionamiento⁴⁹, los cuales son configurados a través de los archivos de kernel “.dts”. Otro punto importante a tomar en cuenta es que antes de activar algún PRU debemos desactivar el bus HDMI, ya que más adelante utilizaremos algunos pines que están multiplexados con los pines del CI que controla el bus HDMI. La desactivación del puerto HDMI se explicará en la siguiente sección.

Como se mencionó en el capítulo anterior, podemos desarrollar código en C (.C), C++ (.cpp) o cualquier otro lenguaje de programación empleando el comando llamado “nano”, el cual se describe en la sección número 5 del anexo 2 y del cual ya se ha hablado anteriormente. Para compilar nuestros códigos en lenguaje C emplearemos el compilador Gcc que fue instalado con

⁴⁹ Las tablas de modos de funcionamiento de los pines las podemos encontrar en: <http://www.embedded-things.com/bbb/beaglebone-black-pin-mux-spreadsheet/>

todas la herramientas necesarias para el uso de Eclipse en el capítulo 4 empleando los códigos del anexo número 4. Para lograr que un programa desarrollado en lenguaje C sea capaz de “montar” un programa “.bin” en la memoria de algún PRU (tal como se observa en la figura 6-2), contamos con una API llamada “PRU Linux Application Loader”, la cual contiene dos librerías⁵⁰ llamadas “prussdrv.h” y “pruss_intc_mapping.h”. Las librerías “prussdrv.h” y “pruss_intc_mapping.h” vienen instaladas por defecto con todas la demás librerías de lenguaje C con las que cuenta la versión de Debian del Beaglebone. Cabe mencionar que dichas librerías sólo funcionan en lenguaje C y que cada librería depende de la otra, por lo tanto, en todos los programas en los que implementemos el uso de un PRU, debemos incluir las dos librerías.

El código en lenguaje ensamblador (.p) que contiene el programa de los PRU también es desarrollado empleando el comando “nano”. Para poder compilar nuestro código en ensamblador se utiliza una aplicación llamada “pasm” la cual viene instalada en Beaglebone por defecto. Para poder verificar la versión de la aplicación “pasm” con la que cuenta nuestro Beaglebone, únicamente debemos escribir el nombre de la aplicación en la terminal, tal como se muestra en la figura 6-3.



```
192.168.7.2 - PuTTY
root@beaglebone:~# pasm

PRU Assembler Version 0.84
Copyright (C) 2005-2013 by Texas Instruments Inc.

Usage: pasm [-V#EBbcmLldz] [-Idir] [-Dname=value] [-Cname] InFile [O

V# - Specify core version (V0,V1,V2,V3). (Default is V1)
E - Assemble for big endian core
B - Create big endian binary output (*.bib)
b - Create little endian binary output (*.bin)
c - Create 'C array' binary output (*.bin.h)
m - Create 'image' binary output (*.img)
L - Create annotated source file style listing (*.txt)
```

Figura 6-3. Verificando la versión de “pasm”.

Para poder desarrollar código en ensamblador debemos conocer a detalle la estructura interna de los PRU, por tal razón se recomienda leer la guía de referencia⁵¹ de uso de los PRU. Los usuarios que deseen desarrollar un programa en ensamblador para incorporarlo a alguna aplicación o proyecto deben comprender de qué manera opera cada registro y conocer todo el set de instrucciones con los que el programador puede disponer.

Un dato muy importante a tomar en cuenta es que el programa que define el funcionamiento y las tareas a realizar de un núcleo PRU es el archivo “.bin” generado con el código con extensión “.p” mientras que el archivo ejecutable generado de nuestro código “.c” al cual llamaremos *Programa principal*, sólo se encarga de cargar el programa “.bin” a un PRU y si el programador lo desea, obtener datos del PRU y hacer uso de ellos. Se hace mención a esto ya que entre la comunidad de

⁵⁰ El set de instrucciones de las librerías “prussdrv.h” y “pruss_intc_mapping.h” lo podemos encontrar en: http://processors.wiki.ti.com/index.php/PRU_Linux_Application_Loader_API_Guide.

⁵¹ El set de instrucciones y la guía de registros para desarrollar programas en ensamblador la podemos encontrar en: <http://mythopoeic.org/BBB-PRU/am335xPruReferenceGuide.pdf>

desarrolladores en internet, hay muchas confusiones sobre el papel que juega cada código, creyendo que el funcionamiento del PRU se define desde el código hecho en lenguaje C.

6.3 GPIO de tiempo real.

En esta sección utilizaremos dos GPIO del PRU0 para poder generar una salida y una entrada digital con la finalidad de detener el parpadeo de un LED cuando presionamos un botón. Antes de iniciar a explicar todos los pasos que se necesitan para ejecutar este ejemplo, debemos armar el circuito de la figura siguiente, empleando una resistencia de 220Ω.

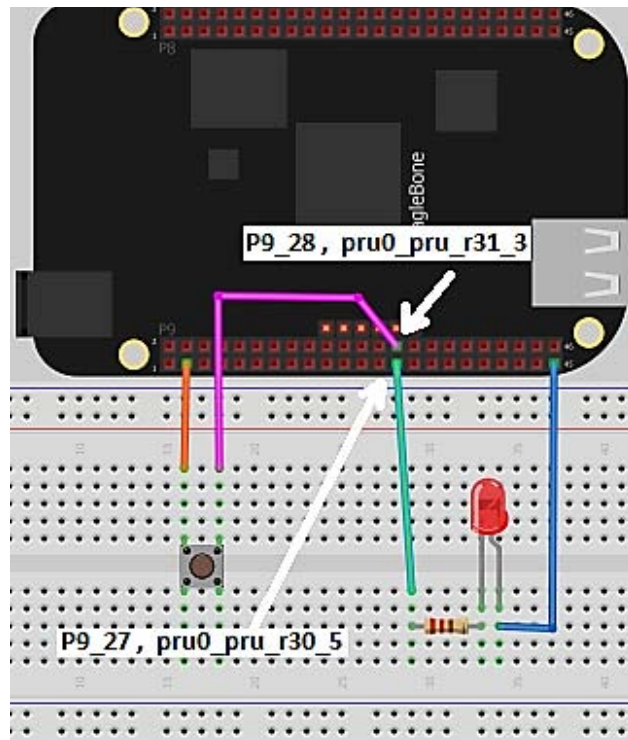


Figura 6-4. Circuito necesario para ver el funcionamiento del PRU0.

Antes de continuar es importante mencionar que cada PRU cuenta con 16 GPIO la cuales puede ser identificadas en las tablas de modos de funcionamiento⁴⁹ y en la figura 6-4 con la siguiente nomenclatura: `pruX_pru_r3Y_Z`. En la nomenclatura anterior **X** se refiere al PRU que deseamos utilizar (0 para usar el PRU0 y 1 para usar el PRU1), **Y** se refiere a la dirección del pin (1 si en una entrada y 0 si es una salida), mientras que **Z** hace referencia al número de GPIO del PRU que estamos utilizando (16 GPIO por cada PRU). Como se puede observar en la figura 6-4, las señales de entrada y salida que emplearemos están ubicadas en los pines P9_27 y P9_28.

A continuación se explicarán cada uno de los pasos necesarios para poder poner el marcha el ejemplo mostrado en esta sección.

Desactivar el bus HDMI.

Como se mencionó anteriormente, antes de poder disponer de cualquier PRU debemos desactivar el bus HDMI ya que los pines que emplearemos de aquí en adelante están multiplexados con los pines del CI que controla la señal HDMI. Para poder desactivar el bus HDMI lo primero que tenemos que hacer es tener acceso a la terminal del Beaglebone ya sea vía SSH o mediante escritorio remoto, después cambiaremos a la ubicación `/boot/uboot/` con el siguiente comando:

```
cd /boot/uboot/
```

Una vez que nos encontremos en la ubicación mencionada anteriormente, abriremos el archivo de texto que se encarga de activar y desactivar algunos componentes de hardware. Para abrir el archivo de texto ingresaremos el siguiente comando:

```
nano uEnv.txt
```

Inmediatamente después de ingresar el comando anterior nos aparecerá una ventana de texto como la de la figura 6-5, en la cual debemos buscar un renglón que está debajo del texto que dice "Disable HDMI" y borrar el carácter "#" que está señalado en la figura 6-5. Una vez que hayamos borrado el carácter procederemos a guardar las modificaciones en el archivo de texto presionando CTRL+O y CTRL+X para salir de archivo de texto. Para reactivar el bus HDMI debemos acceder nuevamente al archivo "uEnv.txt" y agregar el carácter "#" que hemos borrando anteriormente y reiniciar el Beaglebone.

Después de haber completado el paso anterior debemos reiniciar el Beaglebone para que el bus HDMI sea desactivado. Podemos reiniciar el Beaglebone presionado el botón "Power " ó ingresando el siguiente comando:

```
sudo reboot
```

```

GNU nano 2.2.6 File: uEnv.txt Modified
#Video: Uncomment to override:
##see: https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Do
#kms_force_mode=video=HDMI-A-1:1024x768@60e

##uart.
#optargs=quiet drm.debug=7 capemgr.disable_partno=BB-UART4

##Enable systemd
systemd=quiet init=/lib/systemd/systemd

##BeagleBone Cape Overrides

##BeagleBone Black:
##Disable HDMI/eMMC
#optargs=capemgr.disable_partno=BB-BONELT-HDMI, BB-BONELT-HDMIN, BB-BONE-EMMC-2G
##Disable HDMI
#optargs=capemgr.disable_partno=BB-BONELT-HDMI, BB-BONELT-HDMIN

```

Figura 6-5. Archivo de texto que se encarga de activar y desactivar hardware del sistema.

Activar el PRU0.

Como se vio en la sección anterior, el primer paso que debemos hacer para poder hacer funcionar nuestro programa de ejemplo, es activar el PRU que necesitamos mediante archivos de kernel del sistema. Cabe mencionar que los archivos con extensión “.dts” que necesitamos para este ejemplo fueron descargados del repositorio de GITHUB que fue creado para este trabajo, sin embargo, podemos crear nuestro propio archivo con extensión “.dts” con el comando nano (el cual se describe en la sección número 5 del anexo 2) transcribiendo el código que se encuentra en el apéndice número 14 y nombrando a nuestro archivo “EBB-PRU-Example.dts”. Para poder cambiar a la carpeta que contiene el archivo con extensión “.dts” que activará el PRU0, debemos tener acceso a la terminal del Beaglebone mediante una conexión SSH e ingresar el siguiente comando:

```
cd Tesis/Capitulo_6/kernel_files/
```

Para compilar el archivo llamado “EBB-PRU-Example.dts” creando un archivo llamado “EBB-PRU-Example-00A0.dtbo”, debemos ingresar el siguiente comando:

```
dtc -O dtb -o EBB-PRU-Example-00A0.dtbo -b 0 -@ EBB-PRU-Example.dts
```

El siguiente paso es copiar el archivo con extensión “.dtbo” a la ubicación /lib/firmware/ empleando el siguiente comando:

```
cp EBB-PRU-Example-00A0.dtbo /lib/firmware/
```

Ahora debemos cambiar a la carpeta /sys/devices/bone_capemgr.9/ con el siguiente comando:

```
cd /sys/devices/bone_capemgr.9/
```

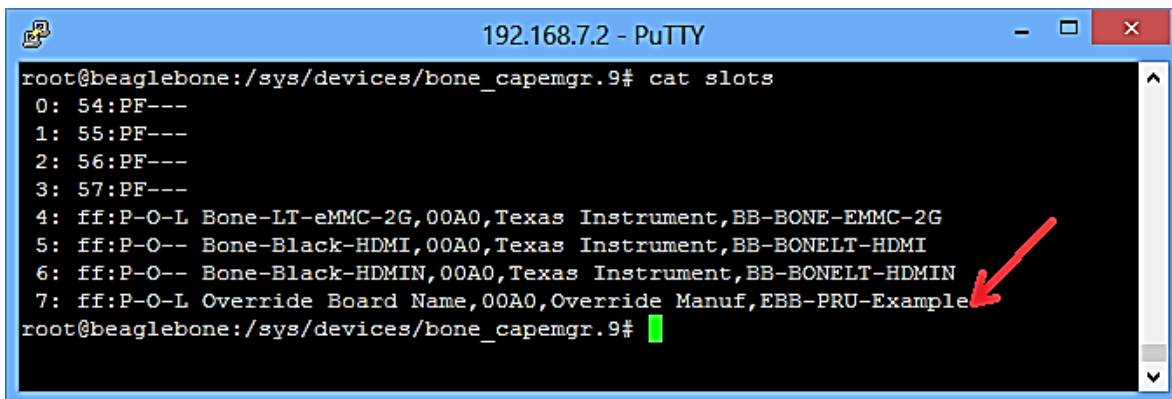
Por último, debemos activar los archivos de kernel que hemos creado ejecutando el siguiente comando:

```
echo EBB-PRU-Example > slots
```

Para verificar que el PRU0 ha sido activado correctamente ingresaremos el siguiente comando:

```
cat slots
```

Al ingresar el comando anterior nos desplegará un texto en la terminal como el de la figura siguiente, en el cual está señalado con una flecha de color rojo el renglón de texto que debemos visualizar si el PRU0 ha sido activado correctamente.



```
192.168.7.2 - PuTTY
root@beaglebone:/sys/devices/bone_capemgr.9# 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,EBB-PRU-Example
root@beaglebone:/sys/devices/bone_capemgr.9#
```

Figura 6-6. Verificando que el PRU0 está activado.

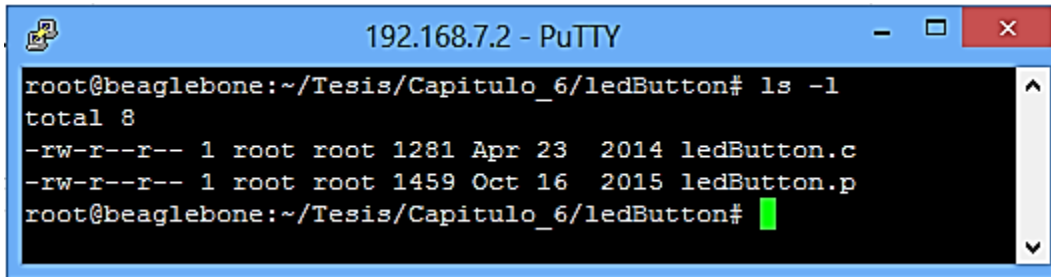
Compilar el programa del PRU0 hecho en lenguaje ensamblador.

Como se mencionó anteriormente, existe un compilador para archivos “.p” llamado “pasm” el cual utilizaremos para compilar el código en ensamblador llamado “ledButon.p” que se encuentra en el repositorio que hemos descargado de GitHub (en la ubicación: Tesis/Capitulo_6/ledButton/) y en el anexo número 15 (podemos crear el archivo “ledButton.p” con el código del apéndice 15 y el comando Nano). Para poder ubicarnos en la carpeta donde se encuentra el archivo “ledButon.p” debemos ingresar los siguientes comandos:

```
cd
```

```
cd Tesis/Capitulo_6/ledButton/
```

Al ingresar el comando “ls-l” podemos observar los archivos contenidos en la ubicación Tesis/Capitulo_6/ledButton/, tal como se observa en la siguiente figura.



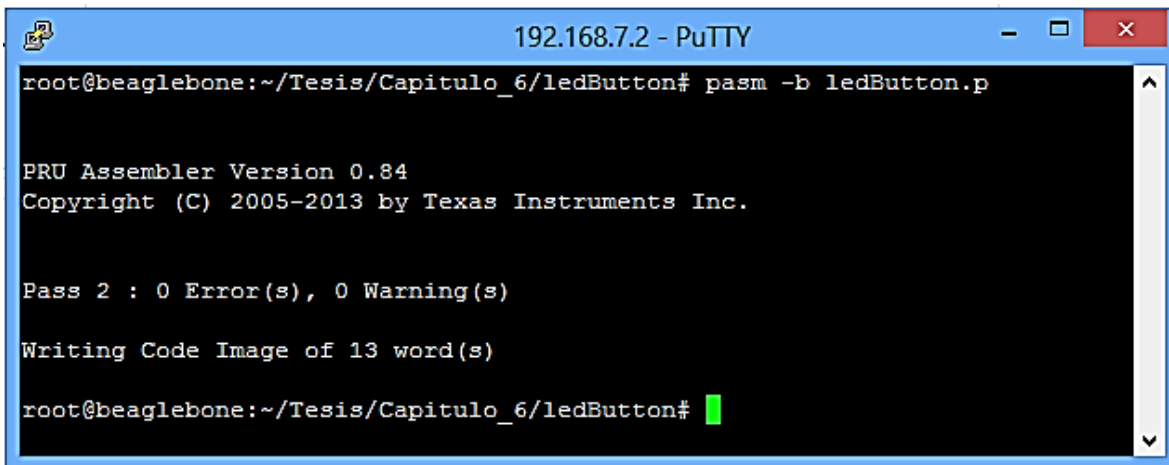
```
192.168.7.2 - PuTTY
root@beaglebone:~/Tesis/Capitulo_6/ledButton# ls -l
total 8
-rw-r--r-- 1 root root 1281 Apr 23 2014 ledButton.c
-rw-r--r-- 1 root root 1459 Oct 16 2015 ledButton.p
root@beaglebone:~/Tesis/Capitulo_6/ledButton#
```

Figura 6-7. Archivos de la carpeta “LedButton”.

En la figura 6-7 podemos observar el archivo “ledButton.c” y el archivo “ledButton.p”, el cual compilaremos ingresando el siguiente comando:

```
pasm -b ledButton.p
```

Cuando “pasm” termine de compilar el código aparecerá un aviso como el de la siguiente figura, el cual nos indica el número de errores (Error) y el número de advertencias (Warning).



```
192.168.7.2 - PuTTY
root@beaglebone:~/Tesis/Capitulo_6/ledButton# pasm -b ledButton.p

PRU Assembler Version 0.84
Copyright (C) 2005-2013 by Texas Instruments Inc.

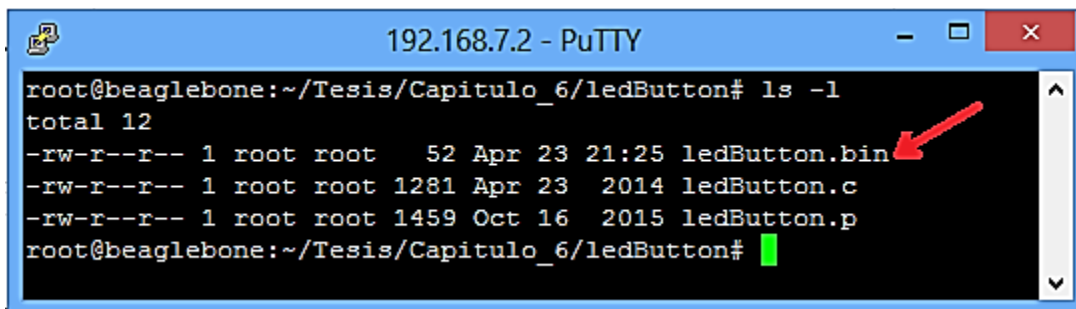
Pass 2 : 0 Error(s), 0 Warning(s)

Writing Code Image of 13 word(s)

root@beaglebone:~/Tesis/Capitulo_6/ledButton#
```

Figura 6-8. Compilando nuestro programa en ensamblador.

En la figura 6-9 podemos observar que ingresamos nuevamente el comando “ls -l” para verificar que se haya creado el archivo “ledbutton.bin” tras finalizar la compilación.



```
192.168.7.2 - PuTTY
root@beaglebone:~/Tesis/Capitulo_6/ledButton# ls -l
total 12
-rw-r--r-- 1 root root 52 Apr 23 21:25 ledButton.bin
-rw-r--r-- 1 root root 1281 Apr 23 2014 ledButton.c
-rw-r--r-- 1 root root 1459 Oct 16 2015 ledButton.p
root@beaglebone:~/Tesis/Capitulo_6/ledButton#
```

Figura 6-9. Verificando el archivo “ledButton.bin”.

Compilar el código en C y ejecutar el programa.

El código en lenguaje C desarrollado para este ejemplo lo podemos encontrar en el repositorio que descargamos de GitHub (Tesis/Capitulo_6/ledButton/ con el nombre de “ledButton.c”) y en el apéndice número 16 (podemos crear el archivo “ledButton.c” con el código del apéndice 16 y el comando Nano).

Para poder ejecutar el programa principal debemos ir a la carpeta donde se encuentra el código, posteriormente compilarlo para obtener el archivo ejecutable que nos permitirá correr el programa. Para poder ubicarnos en la carpeta donde se encuentra el archivo “ledbutton.c” debemos ingresar en la terminal los siguientes comandos:

```
cd  
cd Tesis/Capitulo_6/ledButton/
```

Una vez que estemos en la ubicación Tesis/Capitulo_6/ledButton/, debemos compilar nuestro código en C con el siguiente comando:

```
gcc ledButton.c -o ledButton -lpthread -lprussdrv
```

Una vez que hayamos compilado el programa podemos verificar (con el comando “ls -l”) que el archivo ejecutable se haya creado, tal como se muestra en la siguiente figura.

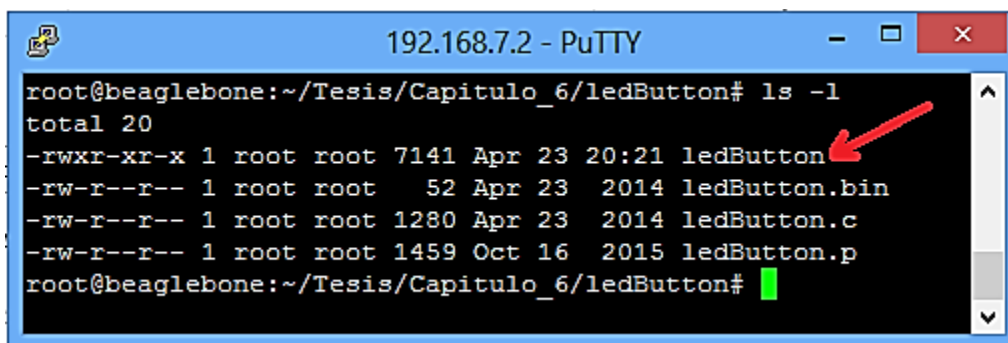


Figura 6-10. Verificando el archivo “ledButton”.

Para ejecutar el programa principal ingresaremos a la terminal el siguiente comando:

```
./ledButton
```

Al ejecutar el programa principal, automáticamente empezará a parpadear el LED, lo cual nos indica que el programa “ledButton.bin” se ha cargado correctamente en el PRU0. Al presionar el botón que hemos conectado al pin P9_28, dejará de parpadear el LED, y el programa cargado en el PRU0 finalizará. Cuando finaliza el programa en la PRU0 el programa principal muestra un aviso que nos indica que el programa ha finalizado y el número de veces que se ha ejecutado el programa, tal como se muestra en la figura 6-11. Se muestra el número de veces que se ha ejecutado el programa con la finalidad de mostrar que podemos tener acceso a algunos registros de memoria de los PRU y hacer uso de la información obtenida⁵².

⁵² Para más información sobre los registros y el mapa de memoria de los PRU-ICSS, consulte: [36]

```

192.168.7.2 - PuTTY
root@beaglebone:~/Tesis/Capitulo_6/ledButton# ./ledButton
EBB PRU programa finalizado, ejecucion del programa # 1.
root@beaglebone:~/Tesis/Capitulo_6/ledButton# ./ledButton
EBB PRU programa finalizado, ejecucion del programa # 2.
root@beaglebone:~/Tesis/Capitulo_6/ledButton# ./ledButton
EBB PRU programa finalizado, ejecucion del programa # 3.
root@beaglebone:~/Tesis/Capitulo_6/ledButton# █

```

Figura 6-11. Ejecutando el programa principal.

En la siguiente figura podemos observar un diagrama de flujo que explica detalladamente el funcionamiento en conjunto del programa principal y el programa de PRU0.

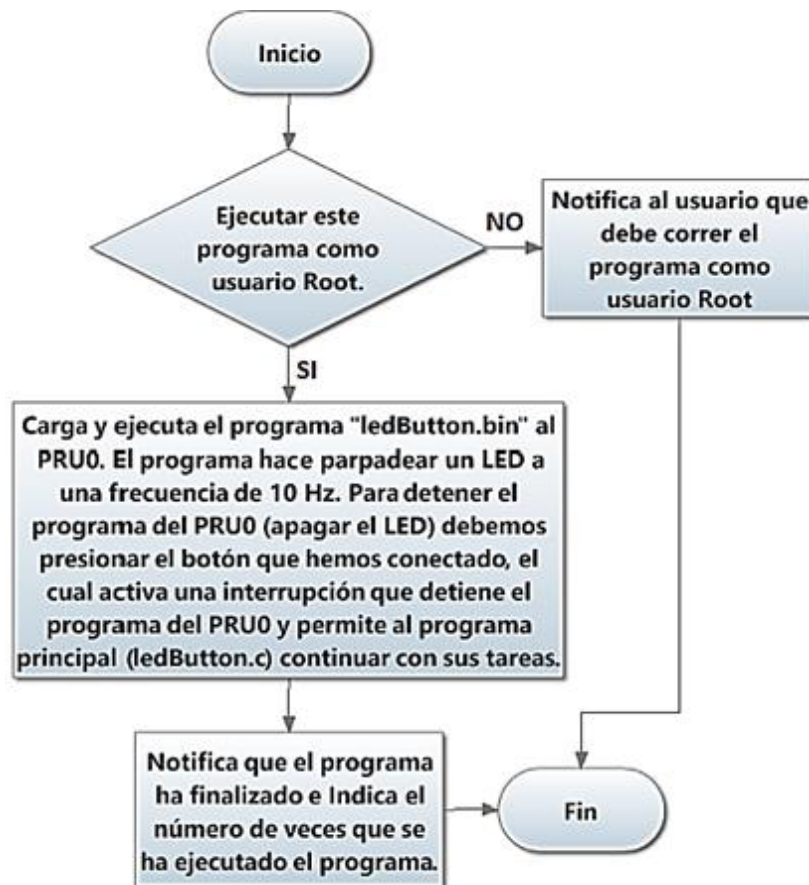


Figura 6-12. Diagrama de flujo para el ejemplo de uso del PRU0.

En el capítulo 4 pudimos observar que podemos disponer de un GPIO empleado C++ en el IDE de Eclipse, sin embargo, cuando utilizamos los GPIO de propósito general del AM3358 tenemos un retardo en el tiempo y en muchas ocasiones este retardo suele ser variable cada que se utiliza el

GPIO. Esto se debe principalmente a que los lenguajes de programación de alto nivel como C++ deben realizar un gran número de operaciones para ejecutar una tarea, a diferencia de los lenguajes de bajo nivel, los cuales tienen la ventaja de realizar tareas con pocas operaciones empleando el lenguaje ensamblador creado para el AM3358.

6.4 Implementación con un sensor Ultrasónico.

En esta sección ejecutaremos un programa que es capaz de medir la distancia a la que se encuentra un objeto utilizando el sensor de ultrasonido HC-SR04. Antes de iniciar a armar el circuito que necesitamos, hablaremos del funcionamiento del HC-SR04 y de los elementos necesarios para poder ejecutar el ejemplo de esta sección.

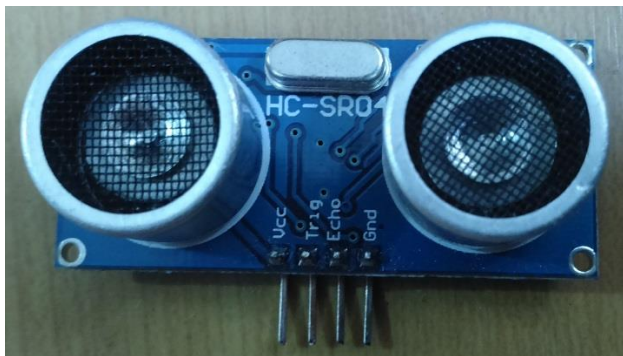


Figura 6-13. Sensor HC04.

El HC-SR04 es un sensor de proximidad que detecta objetos que van desde pocos centímetros hasta no más de 2 metros. Este sensor emite un sonido a una frecuencia que no puede escuchar el oído humano, con el fin de medir el tiempo en que la señal de sonido emitida tarda en regresar, convirtiendo el tiempo medido en la distancia a la cual se encuentra el objeto. El módulo HC-SR04 cuenta con 4 pines llamados: Vcc, Trig, Echo y Gnd. Lo cuales se pueden observar en la figura 6-13.

El funcionamiento del módulo HC-SR04 es fácil de comprender ya que únicamente utiliza dos pines (Echo y Trig) para poder interactuar con microcontroladores o dispositivos de tiempo real. En la figura 6-14 podemos observar un diagrama de tiempos que nos ayudará a comprender mejor el funcionamiento del HC-SR04. Para poder iniciar la lectura de una distancia lo primero que debemos hacer es mandar un estado alto por el pin Trig (el estado alto puede durar $10\mu\text{s}$ o más), $10\mu\text{s}$ después de haber iniciado el estado alto en el pin "Trig", se mandará un tren de 8 pulsos a 40kHz por el transmisor de la onda de sonido (Sound wave), por último, cuando termine de enviarse el tren de 8 pulsos, la salida "Echo" se pondrá en estado alto y cambiará a estado bajo cuando la onda sonora enviada regrese al receptor de ondas del HC-SR04.

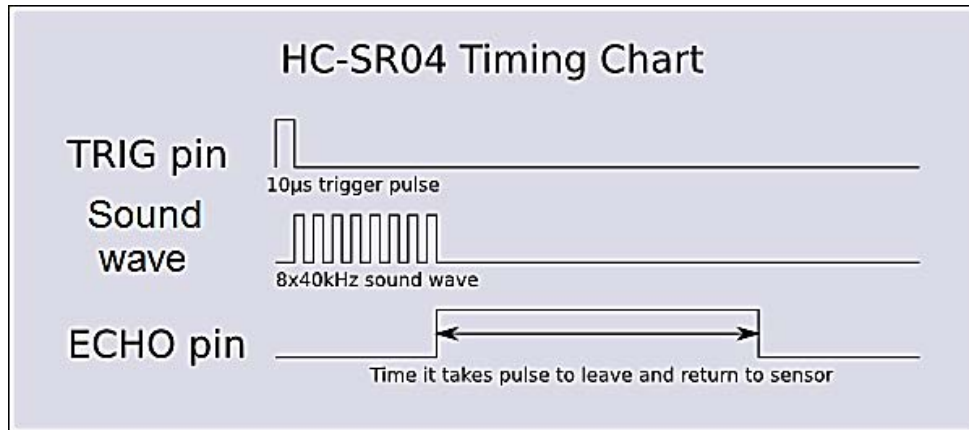


Figura 6-14. Diagrama de tiempos que describe el funcionamiento del HC-SR04.

Para poder obtener la distancia a la que se encuentra un objeto debemos contar el tiempo (en μs) que permanece en estado alto el pin "Echo". Una vez que tengamos el tiempo que duró en estado alto el pin "Echo", obtendremos la distancia en centímetros aplicando la siguiente formula [37]:

$$\text{Distancia en cm.} = \frac{\mu\text{s}}{58}$$

El HC-SR04 es un dispositivo que trabaja a 5v, por lo tanto, en estado alto envía un voltaje de 5v y detecta como estado alto los voltajes mayores a 4v, lo cual nos generó un problema a la hora de querer adaptar nuestro Beaglebone con el HC-SR04 y que para el pin "Trig" necesitamos una señal digital con un voltaje de 5v mientras que la salida de 5v del pin "echo" debemos reducirla a 3.3v para que sea compatible con nuestro Beaglebone. Para convertir los voltajes de 3.3v a 5v y viceversa se utilizó el CI CD4050⁵³ con la finalidad de dar a conocer este CI como una alternativa a un regulador, el cual cuenta con tiempos de retardo de niveles lógicos muy lentos.

EL CD4050⁵⁴ es un CI que consta de 6 buffers independientes los cuales tienen un gran número de aplicaciones en electrónica, sin embargo, en este trabajo sólo nos centraremos en utilizarlos como convertidores de niveles lógicos. En electrónica Digital, un buffer es un dispositivo que actúa como un amplificador seguidor de voltaje, lo que significa que si una señal llega con poca corriente o voltaje, el circuito seguidor compensa esa pérdida con el voltaje de referencia del amplificador, el cual suele conectarse directamente a la fuente de alimentación. O por el contrario, si una señal llega con un voltaje mayor al voltaje de referencia, un seguidor de voltaje también funciona como regulador, impidiendo enviar a la salida del seguidor un voltaje mayor al voltaje de referencia.

⁵³ Cabe mencionar que también es posible implementar un divisor de voltaje y un transistor en saturación y corte.

⁵⁴ Para más información consulte la hoja de datos técnicos en: <https://www.fairchildsemi.com/datasheets/CD/CD4049UBC.pdf>

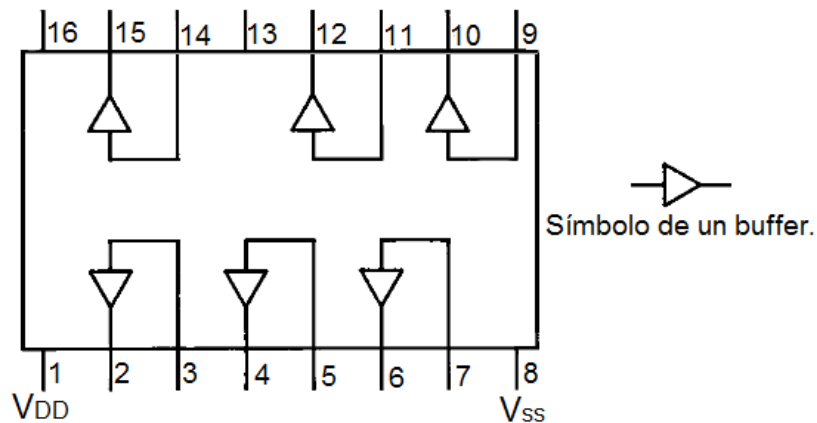


Figura 6-15. Estructura del CD4050.

Todo lo mencionado anteriormente hace al CD4050 un elemento ideal para nuestro propósito ya que además es barato y fácil de conseguir en comparación con los dispositivos convertidores de niveles lógicos más comunes. Cabe mencionar que el tiempo de retardo del CD4050 es de 15ns empleando voltajes de 5v o menores, lo que significa que no afectaría significativamente las mediciones ya que las mediciones de tiempo son tomadas en μs . Además, el tiempo de retardo del CD4050 sólo ocasionaría un desfase en el tiempo de las señales, no afectando el tiempo en que las señales permanecen en alto o bajo, ya que este es el parámetro más importante para tener una medición correcta. En la figura 6-15 podemos observar la estructura interna del CD4050 junto al símbolo que se suele usar en electrónica digital para describir un buffer.

Después de haber entendido el funcionamiento de los elementos necesarios para este ejemplo procederemos a armar el circuito de la figura siguiente.

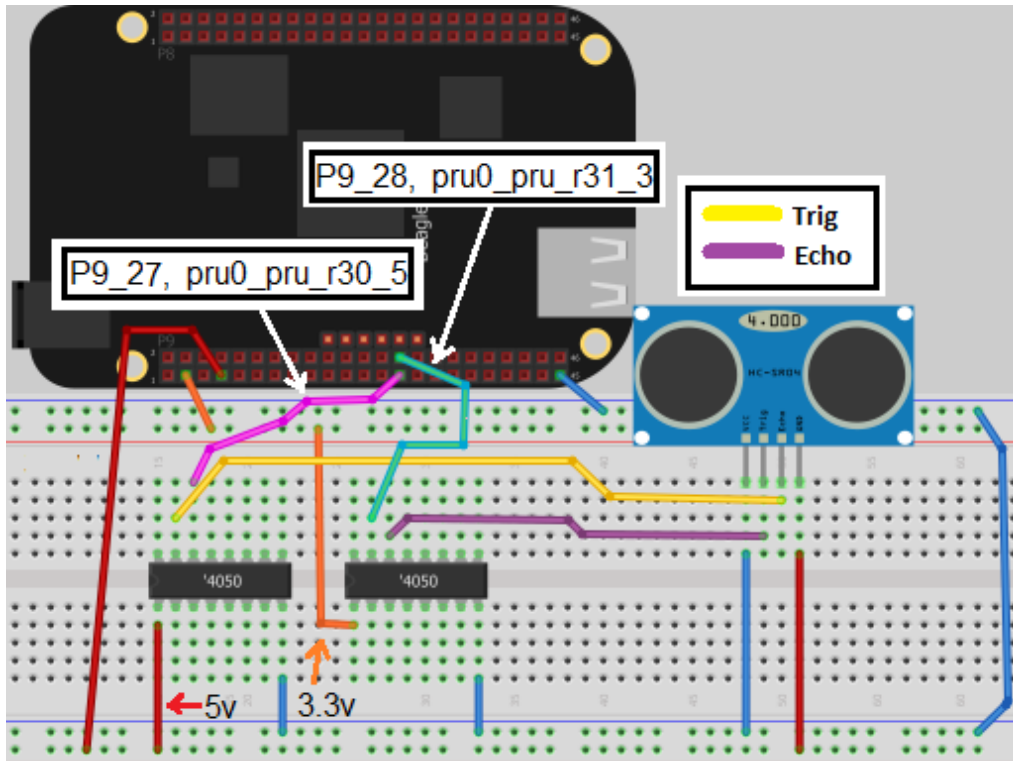


Figura 6-16. Circuito que implementa el HC-SR04 y el CD4050.

Como podemos ver en la figura 6-16, hemos utilizado dos CD4050 ya que el voltaje de referencia (Vdd) es el voltaje de alimentación de todo el circuito, lo que significa que todos los buffers de cada CD4050 tienen el mismo voltaje de referencia. En este caso es necesario convertir los niveles lógicos de 3.3v a 5v y de 5v a 3.3v, por tal razón, se utilizan dos CD4050, cada uno conectado a la fuente de voltaje de 3.3v y 5v.

Con respecto a los pines del Beaglebone, utilizaremos los mismos pines que se utilizaron en el ejemplo anterior. El pin P9_28 es el encargado de generar la señal que recibirá el pin "Trig" del HC-SR04, mientras que el pin P9_27 será el encargado de recibir el estado alto del pin "Echo".

A continuación realizaremos cada uno de los pasos para poder ejecutar el programa principal de este ejemplo. Para este ejemplo también es necesario tener desactivado el bus HDMI, por tal razón, debemos asegurarnos de que así sea. El procedimiento de ejecución de este ejemplo y algunos comandos son idénticos al ejemplo anterior, por lo tanto, no profundizaremos en las secciones que ya se hayan visto anteriormente.

Podemos encontrar los códigos necesarios para este ejemplo en el repositorio que descargamos de GitHub, en la ubicación `"/Tesis/Capitulo_6/ultrasonic"`. Si no contamos con el repositorio de GitHub podemos crear el código en ensamblador (`ultrasonic.p`) con el comando `nano` y el código que aparece en el apéndice 17. También podemos crear el código principal (`ultrasonic.c`) empleando el código del apéndice 18 y el comando `nano`.

Activar el PRU0.

Para activar el PRU0 debemos ingresar a la terminal del Beaglebone los siguientes comandos:

```
cd Tesis/Capitulo_6/kernel_files/  
  
dtc -O dtb -o EBB-PRU-Example-00A0.dtbo -b 0 -@ EBB-PRU-Example.dts  
  
cp EBB-PRU-Example-00A0.dtbo /lib/firmware/  
  
cd /sys/devices/bone_capemgr.9/  
  
echo EBB-PRU-Example > slots
```

Compilar el programa del PRU0 hecho en lenguaje ensamblador.

Para compilar el archivo llamado “ultrasonic.p”, debemos ingresar a la terminal del Beaglebone los siguientes comandos:

```
cd  
cd Tesis/Capitulo_6/ultrasonic/  
pasm -b ultrasonic.p
```

Compilar el código en C y ejecutar el programa.

Para compilar el archivo llamado “ultrasonic.c” debemos ingresar a la terminal del Beaglebone el siguiente comando:

```
gcc ultrasonic.c -o ultrasonic -lpthread -lprussdrv
```

Después de haber compilado el programa debemos correr el archivo ejecutable con el siguiente comando:

```
./ultrasonic
```

Al ejecutar este programa observaremos que el programa automáticamente inicia la detección de objetos y muestra la medición obtenida en pantalla, tal como se muestra en la siguiente figura.

```
192.168.7.2 - PuTTY
root@beaglebone:~/Tesis/Capitulo_6/ultrasonic# ./ultrasonic
Programa que mide la distancia a la que se encuentra un objeto.
Objeto encontrado a 4.718621 cm) █
```

Figura 6-17. Programa en ejecución.

El programa toma una medición cada 100ms, hasta completar un total de 500 mediciones. Cada una de las mediciones tomadas se muestra en la misma línea o renglón (se borra del renglón la medición anterior y se muestra la nueva medición), por tal razón, en la figura 6-17 sólo se puede apreciar un renglón con una sola medición. En la figura 6-18 podemos apreciar que cuando el programa termina las 500 mediciones, muestra en la terminal algunos detalles de la ejecución del programa,

```
192.168.7.2 - PuTTY
root@beaglebone:~/Tesis/Capitulo_6/ultrasonic# ./ultrasonic
Programa que mide la distancia a la que se encuentra un objeto.
Programa de deteccion de distancias finalizado, ejecucion numero: 7.
Los detalles de la ejecucion del programa se muestran a continuacion:
- El numero de muestras tomadas fue: 500.
- El tiempo de retardo es de 100 milisegundos
- La ultima lectura de distancia tomada es de 4.719655 cm.
root@beaglebone:~/Tesis/Capitulo_6/ultrasonic# █
```

Figura 6-18. Finalización del programa.

El programa es capaz de mostrar el número de veces que se ha ejecutado desde que se encendió el Beaglebone, también muestra el número de muestras que fueron tomadas, el tiempo de retardo entre cada medición y la longitud en cm de la última medición tomada.

En la figura 6-19 se muestra un diagrama de flujo que nos describe a detalle el funcionamiento del programa. Este programa, al igual que el programa del ejemplo anterior, verifica que el usuario que lo está ejecutando sea el root, ya que sólo el root puede tener acceso a registros y mapas de memoria de los PRU. Este código es capaz de ejecutar el programa cargado en la PRU0 más de una vez, a diferencia del ejemplo anterior, el cual sólo ejecuta el programa del PRU0 sólo una vez. Por cada medición se ejecuta el programa de la PRU0 ya que el programa solo puede obtener una medición por cada vez que se ejecuta, lo cual quiere decir que el programa principal ejecuta 500 veces el código cargado en la PRU0. Cuando el programa principal finaliza 500 mediciones, muestra los datos de la figura 6-18 y termina la ejecución.

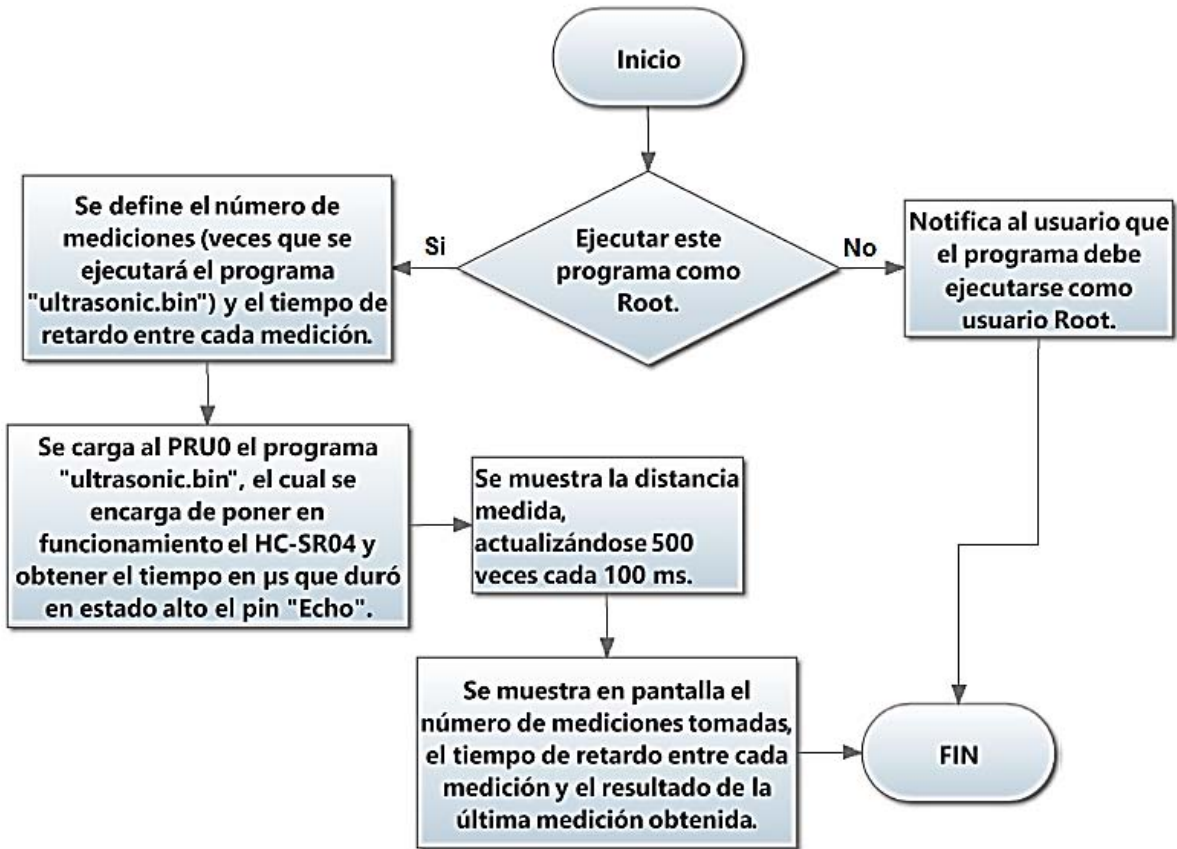


Figura 6-19. Diagrama de flujo de funcionamiento del programa principal.

Cabe mencionar que la implementación de un sensor ultrasónico y de cualquier otro sensor de tiempo real solo es posible llevarla a cabo a través de los GPIO de los PRU ya que como se mencionó anteriormente, estos GPIO tienen la ventaja de actuar más rápido al recibir instrucciones que tardan menos tiempo en ser decodificadas y ejecutadas. Además el tiempo en que se ejecuta la petición del programa suele ser siempre el mismo, a diferencia de los GPIO de uso general del Sitara AM338, los cuales tienen la desventaja de tener un tiempo variable el cual depende de la carga de trabajo del procesador.

Conclusiones

A lo largo de este trabajo hemos explorado y analizado detalladamente con ejemplos cada uno de los buses y puertos con los que cuenta el Beaglebone y gracias a esto pudimos comprender el verdadero potencial de esta tarjeta de desarrollo. No solo se observó el funcionamiento y utilización de los buses de comunicación y elementos más usados (I²c, UART, SPI, ADC etc.), sino que también pudimos analizar a detalle cómo interactuar con un sistema operativo Linux como usuarios avanzados a través de la terminal de comandos, lo cual nos dio conocimientos avanzados sobre dichos sistemas operativos.

También conocimos a detalle cómo está estructurado internamente el procesador ARM Sitara AM3358 e incluso pudimos implementar el uso de un lenguaje ensamblador que nos ayudó a utilizar elementos de tiempo real.

Otro de los hallazgos más importantes durante la realización de este trabajo fue la implementación del Beaglebone a una red LAN, lo cual nos da un punto de partida para poder usar nuestro Beaglebone como un servidor web y tener comunicación desde una ubicación remota para controlar algún proceso e incluso podríamos utilizarlo como servidor de almacenamiento y poder descargar contenido almacenado en el Beaglebone desde otras computadoras empleando un protocolo de transmisión de archivos llamado FTP (file transfer protocol).

Cabe mencionar que en este trabajo se utilizaron los lenguajes de programación C/C++ ya que estos son los más utilizados en programación de sistemas embebidos, sin embargo, podemos implementar otro tipo de lenguajes de programación, los cuales presentan más ventajas en aplicaciones que requieran de una interfaz gráfica, como es el caso de JAVA. Si se desea crear entorno gráfico utilizando los lenguajes de programación C/C++ podemos implementar un IDE llamado QT, que cuenta con soporte para la arquitectura ARM y del cual se habló brevemente en este trabajo.

Beaglebone Black, al igual que las demás SBC, son tarjetas de desarrollo que fueron diseñadas no solo para desarrollar proyectos electrónicos, sino que además tiene como finalidad difundir y enseñar de una manera muy práctica cómo manejar apropiadamente un sistema basado en Linux y conocer más a detalle cómo es que podemos disponer del hardware del sistema creando nuevos archivos de kernel y teniendo acceso a archivos que controlan el hardware de manera directa.

Con este trabajo hemos demostrado que las posibilidades de desarrollo de proyectos con Beaglebone son muy grandes, sin embargo, para poder sacar el máximo provecho de nuestra tarjeta de desarrollo necesitamos tener conocimientos profundos sobre sistemas operativos basados en linux, por tal razón, es recomendable estudiar acerca de la arquitectura Linux, basándonos muy especialmente en el kernel del sistema, ya que de éste depende el que nuestra tarjeta sea compatible con algún dispositivo que deseemos implementar en algún proyecto. También es recomendable tener conocimientos profundos sobre un lenguaje de programación ya que como se vio a lo largo de este trabajo, la programación efectuada en Beaglebone es mucho más compleja que en tarjetas de desarrollo como Arduino o en los microcontroladores PIC.

La organización de los temas se ha elaborado de tal manera que se presenten de forma sistematizada y que sirvan como una guía útil en el desarrollo de un proyecto. Para conseguir esto, se llevó a cabo un arduo trabajo de investigación y de organización de la información dispersa existente. La bibliografía presentada, da una muestra de ello.

El trabajo es el inicio y constituye una invitación para desarrollar una amplia gama de posibilidades de aplicación de las tarjetas de desarrollo con sistema operativo.

BIBLIOGRAFÍA.

[1] «Arquitectura ARM», *Wikipedia, la enciclopedia libre*. 01-jul-2015.

- [2] «Cortex-A Series - ARM». [En línea]. Disponible en: <http://www.arm.com/products/processors/cortex-a/>. [Accedido: 12-ago-2015].
- [3] «ARM Information Center». [En línea]. Disponible en: <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0344k/11002917.html>. [Accedido: 06-ago-2015].
- [4] «Ciclo de instrucción», *Wikipedia, la enciclopedia libre*. 11-may-2016.
- [5] «EtherCAT», *Wikipedia, la enciclopedia libre*. 24-sep-2014.
- [6] «Profibus», *Wikipedia, la enciclopedia libre*. 08-jul-2015.
- [7] «AM335x Sitara Processors (Rev. H) - sprs717h.pdf». [En línea]. Disponible en: <http://www.ti.com/lit/ds/sprs717h/sprs717h.pdf>. [Accedido: 12-ago-2015].
- [8] «Choose Between Raspberry Pi or BeagleBone Black | Make», *Make: DIY Projects, How-Tos, Electronics, Crafts and Ideas for Makers*. [En línea]. Disponible en: <http://makezine.com/magazine/how-to-choose-the-right-platform-raspberry-pi-or-beaglebone-black/>. [Accedido: 12-ago-2015].
- [9] «Raspberry Pi 2 on sale now at \$35», *Raspberry Pi* [En línea]. Disponible en: <https://www.raspberrypi.org/blog/raspberry-pi-2-on-sale/> [Accedido: 06-ago-2015].
- [10] «Arduino - IntelGalileo». [En línea]. Disponible en: <https://www.arduino.cc/en/ArduinoCertified/IntelGalileo>. [Accedido: 07-jul-2015].
- [11] «Cubieboard A20», *Seeed Studio Bazaar*. [En línea]. Disponible en: <http://www.seeedstudio.com/depot/Cubieboard-A20-p-1553.html>. [Accedido: 07-jul-2015].
- [12] «BBB_SRM.pdf». [En línea]. Disponible en: http://www.adafruit.com/datasheets/BBB_SRM.pdf. [Accedido: 12-ago-2015].
- [13] «BeagleBoard.org - getting-started». [En línea]. Disponible en: <http://beagleboard.org/getting-started>. [Accedido: 07-jul-2015].
- [14] «Dirección IP», *Wikipedia, la enciclopedia libre*. 05-ago-2015.
- [15] «sudo», *Wikipedia, la enciclopedia libre*. 12-may-2015.
- [16] «Modelo OSI», *Wikipedia, la enciclopedia libre*. 19-ago-2015.
- [17] «Dynamic Host Configuration Protocol», *Wikipedia, la enciclopedia libre*. 05-jul-2015.
- [18] «Ambiente de desarrollo integrado», *Wikipedia, la enciclopedia libre*. 13-jul-2015.
- [19] «Paradigma de programación», *Wikipedia, la enciclopedia libre*. 09-sep-2015.
- [20] «Resistencias Pull-Down y Pull-Up», *OvToaster*. [En línea]. Disponible en: <http://ovtoaster.com/resistencias-pulldown-y-pullup/>. [Accedido: 01-oct-2015].
- [21] D. Molloy, *Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux*, 1 edition. Indianapolis, IN: Wiley, 2014.
- [22] «TMS320C674x/OMAP-L1x Processor Enhanced High-Resolution PWM (eHRPWM) UG (Rev. C) - sprufl3c.pdf». [En línea]. Disponible en: <http://www.ti.com/lit/ug/sprufl3c/sprufl3c.pdf>
- [23] «TMS320x280x, 2801x, 2804x Enhanced Capture (ECAP - spru807b.pdf». [En línea]. Disponible en: <http://www.ti.com/lit/ug/spru807b/spru807b.pdf>
- [24] «Baudio», *Wikipedia, la enciclopedia libre*. 27-jul-2015.
- [25] «LPC812 MAX Experiment: UART | mbed». [En línea]. Disponible en: https://developer.mbed.org/users/embeddedartists/notebook/lpc812-max-experiment_uart/. [Accedido: 12-oct-2015].
- [26] E Enigmatico, *Comandos AT con Arduino uno: modulo bluetooth HC-05*. 2013 [En línea]. Disponible en: <https://www.youtube.com/watch?v=xwu33aA89qQ>.
- [27] «Microsoft Word - HC-0305 serial module AT commamd set201104修订.doc - HC-0305_serial_module_AT_command_set_201104_revised.pdf». [En línea]. Disponible en:

- http://www.linotux.ch/arduino/HC-0305_serial_module_AT_command_set_201104_revised.pdf .
- [28] «PCF8574 Remote 8-Bit I/O Expander for I2C Bus (Rev. H) - pcf8574.pdf». [En línea]. Disponible en:
<http://www.ti.com/lit/ds/sprs717h/sprs717h.pdf>. [Accedido: 12-ago-2015].
- [29] «Microsoft Word - TC1602A-01T SpecV00 2009-09-23.doc - TC1602A-01T.pdf». .
- [30] «Serial Peripheral Interface Bus», *Wikipedia, the free encyclopedia*. 26-oct-2015.
- [31] «SN54HC595, SN74HC595 (Rev. G) - SN74HC595.pdf». .
- [32] «RPI USB Webcams - eLinux.org». [En línea]. Disponible en:
http://elinux.org/RPI_USB_Webcams. [Accedido: 20-oct-2015].
- [33] «VideoLAN - Página oficial del Reproductor multimedia VLC, el “framework” de código abierto de vídeo!» [En línea]. Disponible en: <http://www.videolan.org/vlc/>. [Accedido: 21-oct-2015].
- [34] «derekmolloy/exploringBB», *GitHub*. [En línea]. Disponible en:
<https://github.com/derekmolloy/exploringBB>. [Accedido: 21-oct-2015].
- [35] «Microsoft PowerPoint - sitara_boot_camp_pru-module1-hw-overview.pptx - Sitara_boot_camp_pru-module1-hw-overview.pdf». .
- [36] «AM335x PRU-ICSS Reference Guide (Rev. A) - am335xPruReferenceGuide.pdf». .
- [37] «HC-SR04 - HCSR04.pdf». .

ANEXOS.

ANEXO 1: DISTRIBUCIÓN DE PINES.....	137
-------------------------------------	-----

ANEXO 2: COMADOS DE CONTROL ELEMENTALES.....	141
ANEXO 3: COMADOS PARA INSTALAR APLICACIONES.....	144
ANEXO 4: HERAMIENTAS PARA EL USO DE ECLIPSE.....	146
ANEXO 5: CREAR ARCHIVOS DE KERNEL PARA ACTIVAR EL BUS SPI0.....	147

ANEXO 1. DISTRIBUCIÓN DE PINES.

Cape expansion headers

P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BTN	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	GPIO_40	GPIO_23	13	14	GPIO_26
GPIO_48	15	16	GPIO_51	GPIO_47	15	16	GPIO_46
GPIO_4	17	18	GPIO_5	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	GPIO_22	19	20	GPIO_63
GPIO_3	21	22	GPIO_2	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	GPIO_15	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	GPIO_14	GPIO_32	25	26	GPIO_61
GPIO_125	27	28	GPIO_123	GPIO_86	27	28	GPIO_88
GPIO_121	29	30	GPIO_122	GPIO_87	29	30	GPIO_89
GPIO_120	31	32	VDD_ADC	GPIO_10	31	32	GPIO_11
AIN4	33	34	GND_ADC	GPIO_9	33	34	GPIO_81
AIN6	35	36	AIN5	GPIO_8	35	36	GPIO_80
AIN2	37	38	AIN3	GPIO_78	37	38	GPIO_79
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	GPIO_7	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	GPIO_70	45	46	GPIO_71

Figura A1.1.

Black eMMC and HDMI pins

P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	MMC1_DAT6	3	4	MMC1_DAT7
VDD_5V	5	6	VDD_5V	MMC1_DAT2	5	6	MMC1_DAT3
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BTN	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	GPIO_40	GPIO_23	13	14	GPIO_26
GPIO_48	15	16	GPIO_51	GPIO_47	15	16	GPIO_46
GPIO_4	17	18	GPIO_5	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	GPIO_22	19	20	MMC1_CMD
GPIO_3	21	22	GPIO_2	MMC1_CLK	21	22	MMC1_DAT5
GPIO_49	23	24	GPIO_15	MMC1_DAT4	23	24	MMC1_DAT1
GPIO_117	25	26	GPIO_14	MMC1_DAT0	25	26	GPIO_61
GPIO_125	27	28	SPI1_CS0	LCD_VSYNC	27	28	LCD_PCLK
SPI1_D0	29	30	GPIO_122	LCD_HSYNC	29	30	LCD_AC_BIAS_E
SPI1_SCLK	31	32	VDD_ADC	LCD_DATA14	31	32	LCD_DATA15
AIN4	33	34	GND_ADC	LCD_DATA13	33	34	LCD_DATA11
AIN6	35	36	AIN5	LCD_DATA12	35	36	LCD_DATA10
AIN2	37	38	AIN3	LCD_DATA8	37	38	LCD_DATA9
AIN0	39	40	AIN1	LCD_DATA6	39	40	LCD_DATA7
GPIO_20	41	42	GPIO_7	LCD_DATA4	41	42	LCD_DATA5
DGND	43	44	DGND	LCD_DATA2	43	44	LCD_DATA3
DGND	45	46	DGND	LCD_DATA0	45	46	LCD_DATA1

Figura A1.2.

65 possible digital I/Os

P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BUT	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	GPIO_40	GPIO_23	13	14	GPIO_26
GPIO_48	15	16	GPIO_51	GPIO_47	15	16	GPIO_46
GPIO_4	17	18	GPIO_5	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	GPIO_22	19	20	GPIO_63
GPIO_3	21	22	GPIO_2	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	GPIO_15	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	GPIO_14	GPIO_32	25	26	GPIO_61
GPIO_125	27	28	GPIO_123	GPIO_86	27	28	GPIO_88
GPIO_121	29	30	GPIO_122	GPIO_87	29	30	GPIO_89
GPIO_120	31	32	VDD_ADC	GPIO_10	31	32	GPIO_11
AIN4	33	34	GNDA_ADC	GPIO_9	33	34	GPIO_81
AIN6	35	36	AIN5	GPIO_8	35	36	GPIO_80
AIN2	37	38	AIN3	GPIO_78	37	38	GPIO_79
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	GPIO_7	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	GPIO_70	45	46	GPIO_71

Figura A1.3.

8 PWMs and 4 timers

P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	TIMER4	7	8	TIMER7
PWR_BUT	9	10	SYS_RESETN	TIMER5	9	10	TIMER6
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	EHRPWM1A	EHRPWM2B	13	14	GPIO_26
GPIO_48	15	16	EHRPWM1B	GPIO_47	15	16	GPIO_46
GPIO_4	17	18	GPIO_5	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	EHRPWM2A	19	20	GPIO_63
EHRPWM0B	21	22	EHRPWM0A	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	GPIO_15	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	GPIO_14	GPIO_32	25	26	GPIO_61
GPIO_125	27	28	ECAPPWM2	GPIO_86	27	28	GPIO_88
EHRPWM0B	29	30	GPIO_122	GPIO_87	29	30	GPIO_89
EHRPWM0A	31	32	VDD_ADC	GPIO_10	31	32	GPIO_11
AIN4	33	34	GNDA_ADC	GPIO_9	33	34	EHRPWM1B
AIN6	35	36	AIN5	GPIO_8	35	36	EHRPWM1A
AIN2	37	38	AIN3	GPIO_78	37	38	GPIO_79
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	ECAPPWMO	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	EHRPWM2A	45	46	EHRPWM2B

Figura A1.4.

4.5 serial UARTs

P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BUT	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
UART4_RXD	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
UART4_TXD	13	14	GPIO_40	GPIO_23	13	14	GPIO_26
GPIO_48	15	16	GPIO_51	GPIO_47	15	16	GPIO_46
GPIO_4	17	18	GPIO_5	GPIO_27	17	18	GPIO_65
UART1_RTSN	19	20	UART1_CTSN	GPIO_22	19	20	GPIO_63
UART2_TXD	21	22	UART2_RXD	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	UART1_TXD	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	UART1_RXD	GPIO_32	25	26	GPIO_61
GPIO_125	27	28	GPIO_123	GPIO_86	27	28	GPIO_88
GPIO_121	29	30	GPIO_122	GPIO_87	29	30	GPIO_89
GPIO_120	31	32	VDD_ADC	UART5_CTSN+	31	32	UART5_RTSN
AIN4	33	34	GND_ADC	UART4_RTSN	33	34	UART3_RTSN
AIN6	35	36	AIN5	UART4_CTSN	35	36	UART3_CTSN
AIN2	37	38	AIN3	UARR5_TXD+	37	38	UART5_RXD+
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	UART3_TXD	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	GPIO_70	45	46	GPIO_71

Figura A1.5.

7 analog inputs (1.8V)

P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BUT	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	GPIO_40	GPIO_23	13	14	GPIO_26
GPIO_48	15	16	GPIO_51	GPIO_47	15	16	GPIO_46
GPIO_4	17	18	GPIO_5	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	GPIO_22	19	20	GPIO_63
GPIO_3	21	22	GPIO_2	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	GPIO_15	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	GPIO_14	GPIO_32	25	26	GPIO_61
GPIO_125	27	28	GPIO_123	GPIO_86	27	28	GPIO_88
GPIO_121	29	30	GPIO_122	GPIO_87	29	30	GPIO_89
GPIO_120	31	32	VDD_ADC	GPIO_10	31	32	GPIO_11
AIN4	33	34	GND_ADC	GPIO_9	33	34	GPIO_81
AIN6	35	36	AIN5	GPIO_8	35	36	GPIO_80
AIN2	37	38	AIN3	GPIO_78	37	38	GPIO_79
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	GPIO_7	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	GPIO_70	45	46	GPIO_71

Figura A1.6.

2 SPI ports

P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BUT	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	GPIO_40	GPIO_23	13	14	GPIO_26
GPIO_48	15	16	GPIO_51	GPIO_47	15	16	GPIO_46
SPIO_CS0	17	18	SPIO_D1	GPIO_27	17	18	GPIO_65
SPI1_CS1	19	20	SPI1_CS0	GPIO_22	19	20	GPIO_63
SPIO_DO	21	22	SPIO_SCLK	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	GPIO_15	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	GPIO_14	GPIO_32	25	26	GPIO_61
GPIO_125	27	28	SPI1_CS0	GPIO_86	27	28	GPIO_88
SPI1_DO	29	30	SPI1_D1	GPIO_87	29	30	GPIO_89
SPI1_SCLK	31	32	VDD_ADC	GPIO_10	31	32	GPIO_11
AIN4	33	34	GND_ADC	GPIO_9	33	34	GPIO_81
AIN6	35	36	AIN5	GPIO_8	35	36	GPIO_80
AIN2	37	38	AIN3	GPIO_78	37	38	GPIO_79
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	SPI1_CS1	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	GPIO_70	45	46	GPIO_71

Figura A1.7.

2 I2C ports

P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BUT	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	GPIO_40	GPIO_23	13	14	GPIO_26
GPIO_48	15	16	GPIO_51	GPIO_47	15	16	GPIO_46
I2C1_SCL	17	18	I2C1_SDA	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	GPIO_22	19	20	GPIO_63
I2C2_SCL	21	22	I2C2_SDA	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	I2C1_SCL	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	I2C1_SDA	GPIO_32	25	26	GPIO_61
GPIO_125	27	28	GPIO_123	GPIO_86	27	28	GPIO_88
GPIO_121	29	30	GPIO_122	GPIO_87	29	30	GPIO_89
GPIO_120	31	32	VDD_ADC	GPIO_10	31	32	GPIO_11
AIN4	33	34	GND_ADC	GPIO_9	33	34	GPIO_81
AIN6	35	36	AIN5	GPIO_8	35	36	GPIO_80
AIN2	37	38	AIN3	GPIO_78	37	38	GPIO_79
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	GPIO_7	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	GPIO_70	45	46	GPIO_71

Figura A1.8.

ANEXO 2: COMANDOS DE CONTROL ELEMENTALES.

Acción	Comando, sintaxis y ejemplo de uso.
1.- Hora y fecha del sistema.	<code>Date</code>
2.- Información del sistema.	<code>uname -a</code>
3.- Ver dispositivos conectados en nuestro host USB.	<code>Lsusb</code>
4.-Leer cualquier archivo de texto de nuestra ubicación actual.	<code>cat [nombre del archivo y extensión]</code> Ejemplos: <code>cat prueba.cpp</code> <code>cat ejemplo.txt</code>
5.- Crear cualquier archivo que contenga texto.	<code>nano [nombre del archivo y extensión]</code> Ejemplos: <code>nano ejemplo.txt</code> <code>nano holamundo.cpp</code> <code>nano holamundo.xml</code>
6.-Cambiar de carpeta.	<code>cd [ruta de la carpeta]</code> Ejemplos: <code>cd /sys/class/gpio/</code> <code>cd /sys/class/pwm/</code>
7.- Regresar a la carpeta donde nos ubicamos por defecto.	<code>Cd</code>
8.- Ver archivos de la carpeta donde estamos ubicados.	<code>ls -l</code>
9.- Ver archivos de una ubicación en específico.	<code>ls [ubicación de los archivos a observar]</code> Ejemplo: <code>ls /sys/class/gpio/</code>
10.-Limpiar pantalla de la terminal.	<code>Clear</code>
11.- Borrar archivo de nuestra ubicación actual.	<code>rm [nombre del archivo y extensión]</code> Ejemplo: <code>rm ejemplo.txt</code> <code>rm holamundo.cpp</code>

12.-Ver ruta de la carpeta donde estamos ubicados.	<code>Pwd</code>
13.-Crear nueva carpeta en nuestra ubicación actual.	<code>mkdir [nombre de la carpeta a crear]</code> Ejemplo: <code>mkdir TesisUnam</code> <code>mkdir FESC</code>
14.- Copiar archivos de una ubicación a otra.	<code>cp [origen] [destino]</code> Ejemplo: <code>cp /home/unam.cpp/ /home/respaldo/unam.cpp/</code>
15.- Mover archivo.	<code>mv [origen] [destino]</code> Ejemplo: <code>mv /home/unam.cpp/ /home/respaldo/unam.cpp/</code>
16.- Compilar un código c++.	<code>g++ [nombre del archivo a compilar] -o [nombre del archivo ejecutable]</code> Ejemplo: <code>g++ holamundo.cpp -o holamundo</code>
17.-Correr un archivo ejecutable (los ejecutables de sistemas linux no tienen extensión).	<code>./[nombre del archivo ejecutable]</code> Ejemplo: <code>./holamundo</code>
18.- Compilar un código en C.	<code>gcc [nombre del archivo a compilar] -o [nombre del archivo ejecutable]</code> Ejemplo: <code>gcc holamundo.cpp -o holamundo</code>
19.- Apagar el sistema.	<code>sudo shutdown -h now</code>
20.-Reiniciar el sistema.	<code>sudo reboot</code>
21.-Activar modo servidor para escritorio remoto.	<code>Tightvncserver</code>
22.-Abrir escritorio remoto.	<code>xvnc4viewer</code>
23. Descomprimir archivos tar.gz	<code>tar -xvf [Nombre del archivo a descomprimir]</code> Ejemplo: <code>tar -xvf eclipse-cpp-luna-SR2-linux-gtk.tar.gz</code>
24. Conexión SSH.	<code>ssh [IP] -l root</code> ó <code>ssh root@[IP]</code>

	<p>Ejemplos:</p> <pre>ssh 192.168.7.2 -l root ssh root@192.168.7.2</pre>
25. Cambiar contraseña del Root.	Passwd
26. Cambiar permisos para poder abrir un archivo ejecutable.	<pre>chmod a+x [Nombre el archivo]</pre> <p>Ejemplo.</p> <pre>chmod a+x Test_Tesis_Unam</pre>
27. Ingresar como Root en la terminal de Devian 8 y Ubuntu.	<pre>su - sudo -s</pre>
28 Compilar un código en C++ que incluya la alguna librería de OpenCV.	<pre>g++ -O2 `pkg-config --cflags --libs opencv` [nombre del archivo y extensión] -o [nombre del archivo ejecutable]</pre> <p>Ejemplo.</p> <pre>g++ -O2 `pkg-config --cflags --libs opencv` face.cpp -o face</pre>

ANEXO 3: COMANDOS PARA INSTALAR APLICACIONES.

1.- Instalación de Sudo para Devian 8.

Primero debemos ingresar como usuario root con el siguiente comando:

```
su -
```

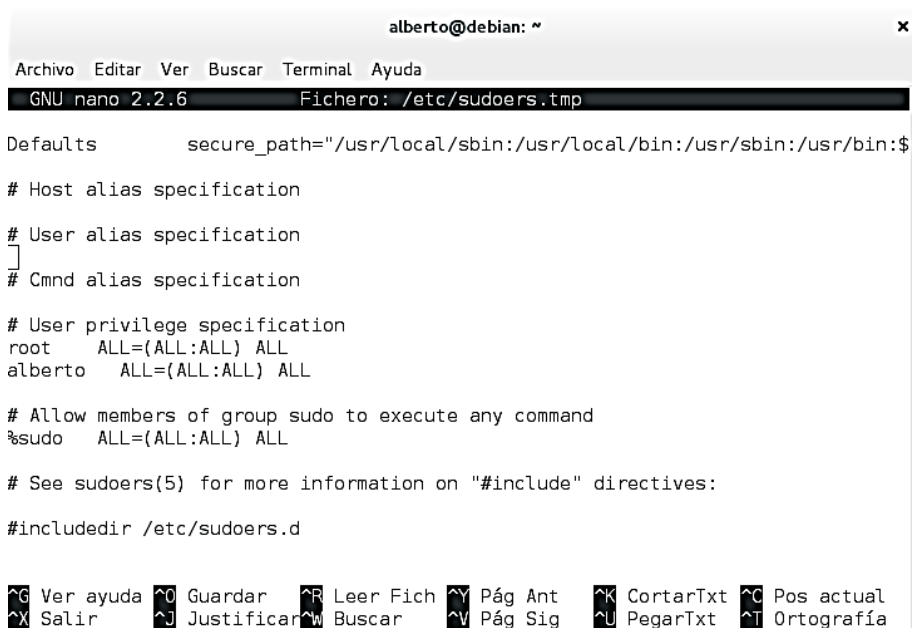
Ahora ingresaremos el siguiente comando el cual se encarga de instalar los repositorios Sudo:

```
apt-get install sudo
```

Después de instalar el repositorio sudo, abriremos un archivo llamado “viSudo”, el cual sirve para que nuestro usuario actual pueda ejecutar los comandos Sudo. Podemos abrir el comando viSudo con el siguiente comando:

```
visudo
```

Después de acceder al archivo “visudo” nos aparecerá una ventana como la siguiente:



```
alberto@debian: ~
Archivo Editar Ver Buscar Terminal Ayuda
GNU nano 2.2.6 Fichero: /etc/sudoers.tmp
Defaults      secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:$
# Host alias specification
# User alias specification
]
# Cmnd alias specification
# User privilege specification
root    ALL=(ALL:ALL) ALL
alberto ALL=(ALL:ALL) ALL
# Allow members of group sudo to execute any command
%sudo  ALL=(ALL:ALL) ALL
# See sudoers(5) for more information on "#include" directives:
#include_dir /etc/sudoers.d
^G Ver ayuda ^O Guardar ^R Leer Fich ^Y Pág Ant ^K CortarTxt ^C Pos actual
^X Salir    ^J Justificar ^W Buscar   ^V Pág Sig ^U PegarTxt ^T Ortografía
```

Figura A3.1.

En la sección llamada “User privilege specification”, debajo de “root” añadiremos nuestro nombre de usuario en Debian (en este caso el usuario es “alberto”), después daremos un espacio y escribiremos el siguiente texto: ALL=(ALL:ALL) ALL. Por último presionaremos CTRL+C para guardar y CTRL+X para salir del archivo de texto. Al finalizar este paso ya podremos usar los comandos Sudo sin ningún tipo de restricción.

2.- Instalación de GitHub.

```
sudo apt-get install git
```

3.- Instalación del reproductor VIC.

```
sudo apt-get install vlc
```

4.-Instalación de VNCViewer.

```
sudo apt-get install xvnc4viewer
```

5.-Instalación de ScreenFetch.

```
sudo apt-get install screenfetch
```

Para verificar que Sistema Operativo tenemos instalado en nuestra computadora.

```
screenfetch
```

6.- Instalación de guvcview.

```
sudo apt-get install guvcview
```

7.- Actualización de repositorios.

```
sudo apt-get update
```

8.- Instalación de la API V4l2 y de la aplicación Fswebcam.

```
sudo apt-get install libv4l-dev v4l-utils fswebcam gpicview libav-tools.
```

9.- Instalación de OpenCV para Beaglebone.

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

```
sudo apt-get install build-essential cmake pkg-config
```

```
sudo apt-get install libtiff4-dev libjpeg-dev libjasper-dev libpng12-dev
```

```
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
```

ANEXO 4: INSTALACIÓN DE ELEMENTOS PARA EL USO DE ECLIPSE

1. -

```
sudo nano /etc/apt/sources.list.d/crosstools.list (pegar el siguiente URL:  
deb http://emdebian.org/tools/debian jessie main).
```

2. -

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

3. -

```
sudo apt-get install curl
```

4. -

```
sudo dpkg --add-architecture armhf
```

5. -

```
dpkg --print-architecture
```

6. -

```
dpkg --print-foreign-architectures
```

7. -

```
sudo apt-get update.
```

8. -

```
sudo apt-get install crossbuild-essential-armhf
```

9. -

```
sudo apt-get install qemu-user-static.
```

ANEXO 5: CREAR ARCHIVOS DE KERNEL PARA ACTIVAR EL BUS SPI0.

1.- Creamos una carpeta llamada SPI, la cual tendrá los archivos que vayamos creando.

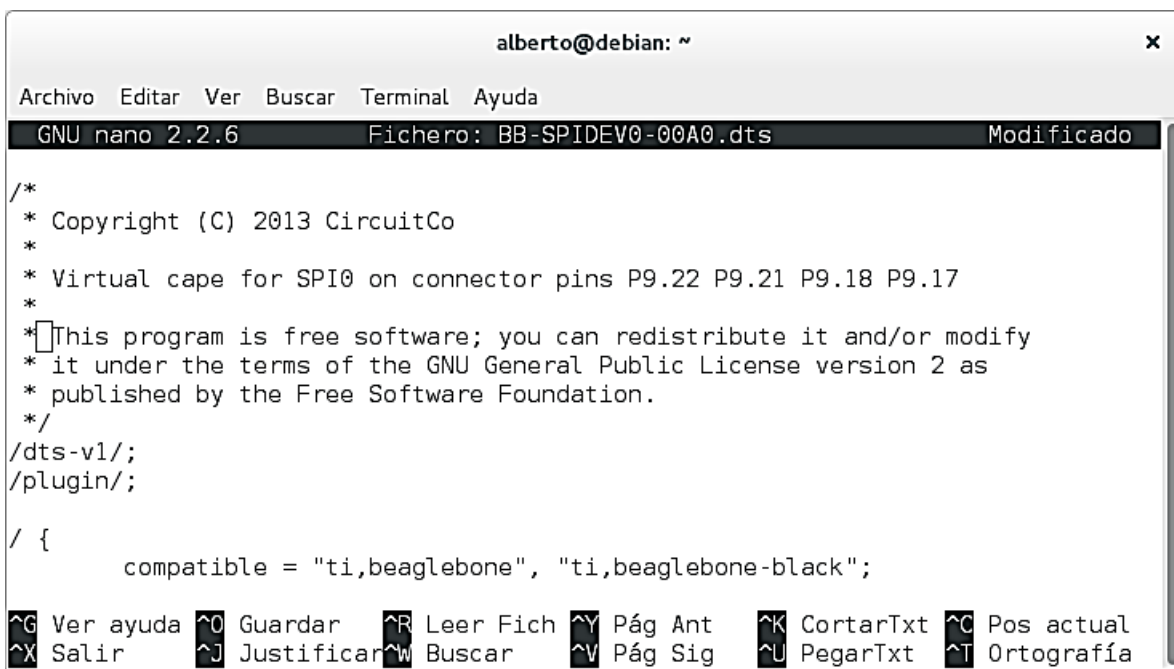
```
Mkdir SPI
```

2.- Nos movemos a la carpeta que acabamos de crear.

```
cd SPI
```

3.- Creamos el primer archivo que contiene un código de configuración con extensión “.dts”. Nos aparecerá una ventana como la que se muestra a continuación, en la cual debemos escribir el código que se encuentra en el anexo 10 (también podemos obtener el código en: <https://github.com/jadonk/cape-firmware/blob/master/arch/arm/boot/dts/BB-SPIDEV0-00A0.dts>). Para guardar los cambios en el archivo presionamos CTRL+O y para salir presionamos CTRL+X.

```
nano BB-SPIDEV0-00A0.dts
```



```
alberto@debian: ~
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
GNU nano 2.2.6  Fichero: BB-SPIDEV0-00A0.dts  Modificado
/*
 * Copyright (C) 2013 CircuitCo
 *
 * Virtual cape for SPI0 on connector pins P9.22 P9.21 P9.18 P9.17
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */
/dts-v1/;
/plugin/;

/ {
    compatible = "ti,beaglebone", "ti,beaglebone-black";
}
^G Ver ayuda  ^O Guardar  ^R Leer Fich  ^Y Pág Ant  ^K CortarTxt  ^C Pos actual
^X Salir      ^J Justificar  ^W Buscar    ^V Pág Sig  ^U PegarTxt   ^T Ortografía
```

Figura A5.1.

4.- Compilamos el archivo que acabamos de crear. Al compilarlo nos creara un archivo llamado “BB-SPIDEV0-00A0.dtbo”.

```
dtc -O dtb -o BB-SPIDEV0-00A0.dtbo -b 0 -@ BB-SPIDEV0-00A0.dts
```

5.- Copiamos el archivo llamado “BB-SPIDEV0-00A0.dtbo” a la ubicación /lib/firmware/

```
cp BB-SPIDEV0-00A0.dtbo /lib/firmware/
```

6.- Ingresamos el siguiente código para activar el SPI0.

```
echo BB-SPIDEV0 > /sys/devices/bone_capemgr.9/slots
```

7.- Con el siguiente código verificaremos que el SPI0 se ha activado correctamente. Observaremos un texto en la terminal similar al de la figura siguiente.

```
cat /sys/devices/bone_capemgr.9/slots
```

8.- Nos cambiaremos a una carpeta llamada Dev.

```
cd /dev
```

8.- Para mandar un valor en hexadecimal (por ejemplo "FF").

```
echo -ne "\xFF" > spidev1.0
```

Apéndices.

Apéndice 1. Programa que implementa el uso de un GPIO con BlackLib.....	150
Apéndice 2. Programa que implementa el uso de un GPIO sin ninguna librería.....	153
Apéndice 3. Programa que implementa el uso de dos GPIO como entradas digitales.....	155
Apéndice 4. Programa que implementa el uso de una entrada Analógica.....	157
Apéndice 5. Programa que genera una señal PWM.....	160
Apéndice 6. Programa que modula una señal PWM.....	162
Apéndice 7. Programa que implementa el uso de un bus UART.....	164
Apéndice 8. Programa que implementa el uso de un bus I ² C.....	167
Apéndice 9. Tablas de datos para la configuración serial del display 1602A.....	166
Apéndice 10. Código para activar el SPI0.....	173
Apéndice 11. Código de configuración de V4I2.....	175
Apéndice 12. Programa que captura y procesa una imagen empleando OpenCV.....	176
Apéndice 13. Programa que reconoce rostros humanos empleando OpenCV.....	177
Apéndice 14. Código para activar el PRU0.....	179
Apéndice 15. Programa en ensamblador para el GPIO de tiempo real.....	181
Apéndice 16. Programa en C que implementa el uso del PRU0 con un LED.....	182
Apéndice 17. Programa en ensamblador para el funcionamiento del HC-SR04.....	183
Apéndice 18. Programa en C que implementa el uso del sensor HC-SR04.....	185

Apéndice 1. Programa que implementa uso de un GPIO como salida digital sin utilizar BlackLib.

```
//=====
// Name      : Blink_without_blacklib.cpp
// Author    : Alberto Picasso
// Version   : 1.0
// Copyright : Este programa puede ser usado y distribuido libremente
// Description : Encender un led sin implementar ninguna librería
//=====

#include<iostream>
#include<fstream>
#include<cstring>
#include<unistd.h>
using namespace std;

//definimos la ubicación de los archivos que controlan la salida del
//pin seleccionado
#define gpio_en "/sys/class/gpio"
#define gpio49 "/sys/class/gpio/gpio49"

//función para activar el pin que utilizamos.
void gpioEnable(){
std::fstream fs;
fs.open(gpio_en"/export", std::fstream::out);
fs<<"49";
fs.close();
}

//Función para desactivar el pin que utilizamos
void gpioDisable(){
std::fstream fs;
fs.open(gpio_en"/unexport", std::fstream::out);
fs<<"49";
fs.close();
}

//Función para programar nuestro pin como una salida
void gpioOut(){
std::fstream fs;
fs.open(gpio49"/direction", std::fstream::out);
fs<<"out";
fs.close();
}

//Función para encender el led
void encender(){
std::fstream fs;
gpioOut();
fs.open(gpio49"/value", std::fstream::out);
fs<<"1";
fs.close();
}
```

```

//función para apagar el led
void apagar(){
    std::fstream fs;
    gpioOut();
    fs.open("gpio49/value", std::fstream::out);
    fs<<"0";
    fs.close();
}

//función para hacer parpadear el led
void flash(){
    std::fstream fs;
    gpioOut();
    int i;
    for(i=0;i<6;i++){//funcion para que parpadee 6 veaces
        fs.open("gpio49/value", std::fstream::out);
        fs<<"1";
        fs.close();
        usleep(500000);
        fs.open("gpio49/value", std::fstream::out);
        fs<<"0";
        fs.close();
        usleep(500000);
    }
}

//Program principal.
int main(){

//primero que nada debemos activar el pin de salida llamando a la función
//"active_gpio"
    gpioEnable();

    char com[4];
//cadena de tres caracteres (se pone el 4 por el caracter nulo)

    std::fstream fs;//declaramos fs (Filestream)

//Mandamos a la pantalla de la consola cadenas de caracteres con las
instrucciones para ejecutar el programa.

    cout<<" "<<endl;
    cout<<"Encender un led Mediante cadenas de caracteres."<<endl;
    cout<<" "<<endl;
    cout<<"oon : enciende el led."<<endl;
    cout<<"off : apaga el led."<<endl;
    cout<<"fsh : parpadea el led seis veces."<<endl;
    cout<<"ext : salir del programa."<<endl;

loop://parte donde regresa el programa cuando se ejecuta el goto.
    //en lugar de loop podemos usar cualquier otro nombre

    cout<<" "<<endl;
    cout<<"ingrese el comando: ";
    cin>> com;

```

```

if (strcmp(com,"oon")==0){//esta función prende el led
    encender();
    goto loop;
}

else if (strcmp(com,"off")==0){ //función que apaga el led.
    apagar();
    goto loop;
}

else if (strcmp(com,"ext")==0){
    cout<<"Fin del programa."<<endl;
    //este if no cuenta con goto así que cuando elegimos esta opción
    //finaliza el programa
}

else if (strcmp(com,"fsh")==0){ //Función para hacer parpadear el led.
    flash();
    goto loop;
}
else{
    cout<<"Comando no aceptado"<<endl;
    goto loop;
}

    gpioDisable();//Cuando finaliza el programa deja al pin que usamos en
su configuración inicial.
    return 0;
}

```

Apéndice 2. Programa que implementa el uso de un GPIO como salida digital.

```
//=====
// Name      : blinkled.cpp
// Author    : Alberto Picasso
// Version   : 1.0
// Copyright : Este programa puede ser usado y distribuido libremente
// Description : Bink led empleando la libreria BlackLib
//=====

#include <string>
#include <iostream>
#include <unistd.h>
#include "BlackLib/BlackLib.h"//llamamos a la librería BlackLib

using namespace std;
using namespace BlackLib;

    BlackGPIO LED(GPIO_49, output, SecureMode);//declaramos el pin 49 como
//una salida, de manera global

//función para encender el led
void encender(){
    LED.setValue(high);
}

//funcion para apagar el led
void apagar(){
    LED.setValue(low);
}

//función para hacer parpadear el led
void flash(){
    int i;
    for(i=0;i<6;i++){
        LED.setValue(high);
        sleep(1);
        LED.setValue(low);
        sleep(1);
    }
}

//Programa principal
int main() {

    cout<< " " << endl;
    cout<<"Programa para encender, apagar y hacer parpadear un
led"<<endl<<endl;;
    cout<<"oon: Encender el Led"<<endl;
    cout<<"off: Apagar el Led"<<endl;
    cout<<"fsh: Parpadeo del Led"<<endl;
    cout<<"ext: Salir del programa"<<endl<<endl;

    loop://Aqui es dond e regresa cilicamente el programa hasta que
metamos el coamndo exit

    cout<<"Ingrese el comando: ";
```

```

    char com[4];
    //cadena de tres caracteres para almacenar variables(se pone el cuatro
    //por el carácter nulo)

    cin>>com;

    //condicionales que comparan la instrucción insertada y ejecutan las
    //Funciones según la petición del usuario.

    if(strcmp(com,"oon")==0){
        encender();
        goto loop;
    }

    else if (strcmp(com,"off")==0){
        apagar();
        goto loop;
    }

    else if(strcmp(com,"fsh")==0){
        flash();
        goto loop;
    }

    else if(strcmp(com,"ext")==0){
        cout<<endl<<"Fin del Programa"<<endl;
//Aquí no hay retorno del loop, por lo tanto nos lleva a la parte final
//del programa
    }

    else{
        cout<<endl<<"Comando no aceptado. "<<endl;
        goto loop;
    }
    return 0;
}

```

Apéndice 3. Programa que implementa el uso de dos GPIO como entradas digitales

```
//=====
// Name      : LedButton.cpp
// Author    : Alberto Picasso
// Version   : 1.0
// Copyright : Este programa puede ser usado y distribuido libremente
// Description : Blink Led con push button.
//=====

#include <iostream>
#include <unistd.h>
#include <unistd.h>
#include "BlackLib/BlackLib.h" //Declaramos la librería BlackLib

using namespace std;
using namespace BlackLib;

//Función para hacer parpadear el led.
void flash(){

    BlackGPIO LED(GPIO_44, output, SecureMode);
    //declaramos el Gpio 44 como una salida

    int i;
    for(i=0;i<10;i++){
        LED.setValue(high); //Ponemos el led en estado alto
        usleep(400000);
        LED.setValue(low); //Ponemos el led en estado bajo
        usleep(400000);
    }
}

//Programa principal
int main() {

    //mostramos instrucciones en Pantalla.
    cout<<" "<<endl<<endl;
    cout<<"Blink Led con push button " << endl;
    cout<<"El programa esta a la espera de una accion"<<endl;
    cout<<"Puede parpadear el led o salir del programa dependiendo del
    boton oprimido."<<endl<<endl;

    //Declaramos los dos pines que utilizaremos como entradas, uno es para
    //prender el led y el otro es para terminar el programa.
    BlackGPIO myGpio1(GPIO_26, input, SecureMode);
    BlackGPIO myGpio2(GPIO_46, input, SecureMode);

    for(;;){
    //Ciclo infinito (hay muchas formas de crear ciclos infinitos). Este
    ciclo es similar a "void loop" de los Arduino pero este ciclo si es
    posible detenerlo.

```

```

//si hay una alto en el GPIO 26 llama a la función para hacer parpadear
//el led
    if( myGpio1.isHigh() )
    {
        std::cout << "Led Parpadeando" << std::endl;
        flash();//llamado de la funcion flash
    }

//si detecta un alto en el GPIO 46 nos saca del ciclo, justo a donde está
//la etiqueta "stop".
    if(myGpio2.isHigh()){
        goto stop;
    }
}

stop:
cout<<"Fin del Programa"<<endl<<endl;
//Envía este letrero a la terminal cuando salimos del Programa.

return 0;
}

```

Apéndice 4. Programa que implementa el uso de una entrada Analógica.

```
//=====
// Name      : ADC_example.cpp
// Author    : Alberto Picasso
// Version   : 1.0
// Copyright : Este programa puede ser usado y distribuido libremente
// Description : Uso del ADC1 con la librería BlackLib
//=====

#include <iostream>
#include "BlackLib/BlackLib.h" //inclusiones la librerian BlackLib

using namespace std;
using namespace BlackLib; //incluimos BlackLib como un namespace.

//Configuración de los pines de salida
BlackGPIO LED0(GPIO_44, output, SecureMode);
BlackGPIO LED1(GPIO_45, output, SecureMode);
BlackGPIO LED2(GPIO_23, output, SecureMode);
BlackGPIO LED3(GPIO_26, output, SecureMode);
BlackGPIO LED4(GPIO_46, output, SecureMode);
BlackGPIO LED5(GPIO_47, output, SecureMode);
BlackGPIO LED6(GPIO_27, output, SecureMode);
BlackGPIO LED7(GPIO_65, output, SecureMode);
BlackGPIO LED8(GPIO_22, output, SecureMode);
BlackGPIO LED9(GPIO_49, output, SecureMode);

//Configuración del ADC que queremos emplear. En este caso nosotros
//elegimos el ADC 1
BlackADC myAdc(AIN1);
    int val;

//Función que activa el led 0
void led_0(){
    LED0.setValue(high);
    LED0.setValue(low);
}

//Función que activa el led 1
void led_1(){
    LED1.setValue(high);
    LED1.setValue(low);
}

//Función que activa el led 2
void led_2(){
    LED2.setValue(high);
    LED2.setValue(low);
}

//Función que activa el led 3
void led_3(){
    LED3.setValue(high);
    LED3.setValue(low);
}
```



```

//Función que activa el led 4
void led_4(){
    LED4.setValue(high);
    LED4.setValue(low);
}

//Función que activa el led 5
void led_5(){
    LED5.setValue(high);
    LED5.setValue(low);
}

//Función que activa el led 6
void led_6(){
    LED6.setValue(high);
    LED6.setValue(low);
}

//Función que activa el led 7
void led_7(){
    LED7.setValue(high);
    LED7.setValue(low);
}

//Función que activa el led 8
void led_8(){
    LED8.setValue(high);
    LED8.setValue(low);
}

//Función que activa el led 9
void led_9(){
    LED9.setValue(high);
    LED9.setValue(low);
}

//Programa principal
int main()
{
    cout<<"Ejemplo de uso del ADC. Para salir del programa presione
    ctrl+c"<<endl;

    //Ciclo infinito que supervisa continuamente los niveles de voltaje en
    ADC1

    for(;;){

        myAdc >> val;//Asignamos el valor del voltaje en mV a la variable val

        if((val>0)&(val<180)) {
            led_0();
        }

        if((val>180)&(val<360)) {
            led_1();
        }

        if((val>360)&(val<540)) {
            led_2();
        }
    }
}

```

```
    if((val>540)&(val<720)) {  
        led_3();    }  
  
    if((val>720)&(val<900)) {  
        led_4();    }  
  
    if((val>900)&(val<1080)) {  
        led_5();    }  
  
    if((val>1080)&(val<1260)) {  
        led_6();    }  
  
    if((val>1260)&(val<1440)) {  
        led_7();    }  
  
    if((val>1440)&(val<1620)) {  
        led_8();    }  
  
    if((val>1620)&(val<1800)) {  
        led_9();    }  
    }  
    return 0;  
}
```

Apéndice 5. Programa que genera una señal PWM.

```
//=====
// Name      : pwm.cpp
// Author    : Alberto Picasso
// Version   : 1.0
// Copyright : Este programa puede ser usado y distribuido libremente
// Description : Programa para crear una señal PWM
//=====

#include <iostream>
#include <unistd.h>
#include <cstring>
#include "BlackLib/BlackLib.h" //incluimos la librería BlackLib

using namespace std;
using namespace BlackLib; // agregamos el namespace BlackLib

double period;
//Declaramos de la variable "Period" como una variable global

//función que se encarga de generar la señal PWM
void pwm(float frec, float dutty_cycle){

    float x;
    x=1/frec; //Calculamos el periodo de la señal

    period= x*(1E9); //La convertimos en microsegundos

    BlackPWM myPwm(P8_19);
//Declaramos el pin "P8_9" como una salida PWM

    myPwm.setDutyPercent(100.0); // Antes de generar la PWM es
//necesario declararla con un ciclo de trabajo del 100%

    myPwm.setPeriodTime(period); // Con esta instrucción le indicamos
//al programa el periodo de la señal en microsegundos

    myPwm.setDutyPercent(dutty_cycle); // Con esta instrucción
//podemos indicarle al programa el ciclo de trabajo.
}

//Programa Principal.
int main() {

    //Instrucciones para que el usuario pueda generar la PWM
    cout<<endl<<"Programa para generar una PWM, antes de iniciar es" <<endl;
    cout<<"necesario conectar un osciloscopio en el pin P8_19" << endl;
    cout << "Ingrese las características deseadas."<<endl<<endl;

    loop2:
//retorno de programa cuando el usuario decide generar una nueva pwm

    float frec,dutty_cycle;
//Declaramos variables que por su nombre es fácil deducir su función.
```

```

    cout << "Frecuencia en hz: ";
    cin >> frec;

    loop:
//Retorno de programa cuando el usuario ingresa un número no permitido
//para el ciclo de trabajo.

    cout << "Ciclo de trabajo en %: ";
    cin >> dutty_cycle;
    cout <<endl<< "PWM generada"<<endl;
//Lanza el texto anterior cuando la señal es generada.

        if(dutty_cycle>100){
// Retornamos a donde está la etiqueta 'Loop' si escribimos un numero
de ciclo de trabajo no valido
            cout<<"Ingresen un número menor de 100."<< endl;
            goto loop;
        }

        pwm(frec,dutty_cycle);
// llamamos a la función "PWM"

    loop4:
// El programa retorna hasta esta etiqueta cuando una letra no es
aceptada

        cout<<"Ingrese 'c' para generar una nueva PWM o 'e' para salir: ";

        char com[2];
//variable para ingresar una cadena de caracteres de dos caracteres.

        cin >> com;

        if(strcmp(com,"c")==0){
//Si el usuario decide generar una nueva PWM el programa regresa a la
etiqueta loop2.
            goto loop2;
        }

        else if(strcmp(com,"e")==0){
//Si decidimos salir del programa nos vamos hasta loop3
            goto loop3;
        }
        else {
            cout<<endl<<"Letra no Aceptada."<<endl;
//Si nos equivocamos de letra aparece este aviso.
            goto loop4;
        }

    loop3:
// Etiqueta que no ayuda a finalizar el programa

    cout <<endl<<"Fin del Programa."<<endl;
    return 0;
}

```

Apéndice 6. Programa que modula una señal PWM.

```
//=====
// Name      : PWM_modulation.cpp.
// Author    : Alberto Picasso.
// Version   : 1.0
// Copyright : Este programa puede ser usado y distribuido libremente.
// Description : Modulacion de una señal PWM.
//=====

#include <iostream>
#include <unistd.h>
#include "BlackLib/BlackLib.h" //agregamos la librería BlackLib

using namespace std;
using namespace BlackLib;

float period; //variables globales, declaramos "period" como variable
gloval

//función encargada de realizar la modulación PWM
void pwm(float frec){

//convertimos la frecuencia en periodo
    float x,per_time;
    x= 1/frec;
    period=x*(1E9);

//Ciclo que se encarga de ir aumentando en 1% el ciclo de trabajo.
    for(int i=1;i<101;i++){

        per_time=i*period/100;

        BlackPWM myPwm(BlackLib::P8_19);
        myPwm.setDutyPercent(100.0);
        myPwm.setPeriodTime(period);

        myPwm.setSpaceRatioTime(per_time, nanosecond);
        usleep(200);
    }

//Ciclo que se encarga de ir disminuyendo en 1% el ciclo de trabajo.

    for(int i=100;i>0;i--){
        per_time=i*period/100;
        BlackPWM myPwm(BlackLib::P8_19);
        myPwm.setDutyPercent(100.0);
        myPwm.setPeriodTime(period);

        myPwm.setSpaceRatioTime(per_time, nanosecond);
        usleep(200);
    }
}
}
```

```

//Función que manda las instrucciones a la terminal.
void instrucciones(){

cout<<endl<<"Con este programa podremos observar la modulacion de la PWM
en un osciloscopio"<<endl;

cout<<"Ingrese la frecuencia deseada para nuestra señal pwm: ";
}

//Programa principal.
int main() {

    instrucciones();//llamamos a la función que muestra las instrucciones

    float frec; //variable que almacenará la frecuencia de la señal.
    cin >> frec;

//ciclo infinito que llama a la funcion encargada de modular la señal PWM
    for(;;){
        pwm(frec);
    }

    return 0;
}

```

Apéndice 7. Programa que implementa el uso de un bus UART

```
//=====
// Name      : uart_example.cpp
// Author    : Alberto Picasso
// Version   : 1.0
// Copyright : Este programa puede ser usado y distribuido libremente
// Description : Ejemplo de uso de la interfaz UART4
//=====

#include <iostream>
#include <unistd.h>
#include "BlackLib/BlackLib.h"

using namespace BlackLib; //declaramos BlackLib como un namespace
using namespace std;

//configuración básica de la UART y del pin que utilizamos como salida
//para el led
    BlackUART myUart(UART4, Baud9600, ParityNo, StopOne, Char8 );
    BlackGPIO LED(GPIO_44, output, SecureMode);

//funciones que realizan las acciones del led.
void oon(){
    LED.setValue(high);
}
void off(){
    LED.setValue(low);
}
void flash(){
    int i;
        for(i=0;i<10;i++){
            LED.setValue(high); //Pone el led en estado alto
            usleep(400000);
            LED.setValue(low); //Pone el led en estado bajo
            usleep(400000);
        }
}

//Programa principal.
int main() {
cout<<"Ejemplo de uso del bluetooth HC05"<<endl;

//configuración de comunicación del puerto uart
    myUart.open(ReadWrite | NonBlock );
    myUart.flush(bothDirection );

//Variables que guardan cadenas de caracteres que contienen las
//instrucciones de uso del programa, las cuales serán enviadas por
//bluetooth.

    string outuart1 = "Programa para encender, apagar y hacer
parpadear un led\n\n";

    string outuart2 = "Opciones: \n\nencender - enciende el Led
\napagar -apaga el Led \nflash - Parpadea el led\n\n";
```

```

        string outuart3 = "\nEl programa esta a la espera de una opcion.
\n";

//con las siguientes instrucciones enviamos las variables anteriores a la
//UART.
        myUart << outuart1;
        myUart << outuart2;
        myUart << outuart3;

        loop:
//cada que se termina de ejecutar una tarea el programa regresa a esta
linea

        usleep(27000);
//retardo de uS que nos permite leer los paquetes de datos recibidos

        string inuart;
//Variable que nos ayudara a comparar los datos recibidos para ejecutar
//las acciones del led

        myUart >> inuart;
//pasamos la cadena de caracteres recibida a la variable inuart para
poder compararla.

        //enciende el led
        if(inuart=="encender"){
            string outuart4 = "\nLed Encendido";
            myUart << outuart4;
            on();
            goto loop;
        }

        //apaga el led
        else if(inuart=="apagar"){
            string outuart5 = "\nLed Apagado";
            myUart << outuart5;
            off();
            goto loop;
        }

        //parpadea el led
        else if(inuart=="flash"){
            string outuart6 = "\nLed Parpadeando";
            myUart << outuart6;
            flash();
            goto loop;
        }

        /*Esto se puso ya que cuando no enviamos ningún paquete de datos vía
bluetooth nos llega un paquete de datos con la cadena de caracteres
"UartReadError"*/
        else if(inuart=="UartReadError"){
            goto loop;
        }

```



```
//termina de ejecutar el programa al recibir la cadena de caracteres
//"salir"
    else if(inuart=="salir"){
        string outuart7 = "\nFin del Programa";
        myUart << outuart7;
        goto exit;
    }
//Nos manda un aviso si enviamos una cadena de caracteres no válida para
//ejecutar una acción.

    else {
        string outuart8 = "\nCadena de caracteres no Aceptada";
        myUart << outuart8;
        goto loop;
    }

    exit: // etiqueta para terminar de ejecutar el programa.
    return 0;
}
```

Apéndice 8. Programa que implementa el uso de un bus I²C.

```
//=====
// Name      : i2c_example.cpp
// Author    : Alberto Picasso
// Version   : 1.0
// Copyright : Este Programa Puede ser usado y distribuido libremente
// Description : Ejemplo del uso de la intefaz I2C con el display 1602A
//=====

#include <iostream>
#include <unistd.h>
#include "BlackLib/BlackLib.h"//Declaramos la librería BlackLib

using namespace std;
using namespace BlackLib;

BlackLib::BlackI2C myI2c(I2C_1, 0x27);
//Declaramos la configuración del Dispositivo I2C de manera Global

//función que se encarga de la configuración del display
void lcd_config(){
    myI2c.open(ReadWrite | NonBlock );

    int time=10000;

    uint8_t dato38 = 0x38;      uint8_t dato08 = 0x08;
    uint8_t dato3c = 0x3c;      uint8_t dato0c = 0x0c;
    uint8_t dato28 = 0x28;      uint8_t datof8 = 0xf8;
    uint8_t dato2c = 0x2c;      uint8_t datofc = 0xfc;
    uint8_t dato18 = 0x18;      uint8_t dato00 = 0x00;
    uint8_t datolc = 0x1c;      uint8_t datoff = 0xff;

    myI2c.writeByte(dato00, datoff);      usleep(time);
    myI2c.writeByte(dato00, dato38);      usleep(time);
    myI2c.writeByte(dato00, dato3c);      usleep(time);
    myI2c.writeByte(dato00, dato38);      usleep(time);
    myI2c.writeByte(dato00, dato3c);      usleep(time);
    myI2c.writeByte(dato00, dato38);      usleep(time);
    myI2c.writeByte(dato00, dato3c);      usleep(time);
    myI2c.writeByte(dato00, dato38);      usleep(time);
    myI2c.writeByte(dato00, dato28);      usleep(time);
    myI2c.writeByte(dato00, dato2c);      usleep(time);
    myI2c.writeByte(dato00, dato28);      usleep(time);
    myI2c.writeByte(dato00, dato2c);      usleep(time);
    myI2c.writeByte(dato00, dato28);      usleep(time);
    myI2c.writeByte(dato00, dato18);      usleep(time);
    myI2c.writeByte(dato00, datolc);      usleep(time);
    myI2c.writeByte(dato00, dato18);      usleep(time);
    myI2c.writeByte(dato00, dato08);      usleep(time);
    myI2c.writeByte(dato00, dato0c);      usleep(time);
    myI2c.writeByte(dato00, dato08);      usleep(time);
    myI2c.writeByte(dato00, datof8);      usleep(time);
    myI2c.writeByte(dato00, datofc);      usleep(time);
    myI2c.writeByte(dato00, datof8);      usleep(time);
}
}
```

```

//Función encargada de imprimir la letra "h"
void word_h(){
    myI2c.open( ReadWrite | NonBlock );

    uint8_t h1 = 0x4d;      uint8_t h3 = 0x8d;  uint8_t h0 = 0x00;
    uint8_t h2 = 0x49;      uint8_t h4 = 0x89;

    int time=25000;
    myI2c.writeByte(h0,h1);    usleep(time);
    myI2c.writeByte(h0,h2);    usleep(time);
    myI2c.writeByte(h0,h3);    usleep(time);
    myI2c.writeByte(h0,h4);    usleep(time);
}

//Función encargada de imprimir la letra "o"
void word_o(){
    myI2c.open( ReadWrite | NonBlock );

    uint8_t o1 = 0x4d;      uint8_t o3 = 0xfd;  uint8_t o0 = 0x00;
    uint8_t o2 = 0x49;      uint8_t o4 = 0xf9;

    int time=25000;
    myI2c.writeByte(o0,o1);    usleep(time);
    myI2c.writeByte(o0,o2);    usleep(time);
    myI2c.writeByte(o0,o3);    usleep(time);
    myI2c.writeByte(o0,o4);    usleep(time);
}

//Función encargada de imprimir la letra "l"
void word_l(){
    myI2c.open( ReadWrite | NonBlock );

    uint8_t l1 = 0x4d;      uint8_t l3 = 0xcd;  uint8_t l0 = 0x00;
    uint8_t l2 = 0x49;      uint8_t l4 = 0xc9;

    int time=25000;
    myI2c.writeByte(l0,l1);    usleep(time);
    myI2c.writeByte(l0,l2);    usleep(time);
    myI2c.writeByte(l0,l3);    usleep(time);
    myI2c.writeByte(l0,l4);    usleep(time);
}

//Función encargada de imprimir la letra "a"
void word_a(){
    myI2c.open( ReadWrite | NonBlock );

    uint8_t a1 = 0x4d;      uint8_t a3 = 0x1d;  uint8_t a0 = 0x00;
    uint8_t a2 = 0x49;      uint8_t a4 = 0x19;

    int time=25000;
    myI2c.writeByte(a0,a1);    usleep(time);
    myI2c.writeByte(a0,a2);    usleep(time);
    myI2c.writeByte(a0,a3);    usleep(time);
    myI2c.writeByte(a0,a4);    usleep(time);
}

```

```

//Función encargada de imprimir el carácter "-"
void word_() {
    myI2c.open(ReadWrite | NonBlock );

    uint8_t e1 = 0x2d;      uint8_t e3 = 0xdd;  uint8_t e0 = 0x00;
    uint8_t e2 = 0x29;      uint8_t e4 = 0xd9;

    int time=25000;
    myI2c.writeByte(e0,e1);    usleep(time);
    myI2c.writeByte(e0,e2);    usleep(time);
    myI2c.writeByte(e0,e3);    usleep(time);
    myI2c.writeByte(e0,e4);    usleep(time);
}

//Función encargada de imprimir la letra "u"
void word_u() {
    myI2c.open(ReadWrite | NonBlock );

    uint8_t u1 = 0x5d;      uint8_t u3 = 0x5d;  uint8_t u0 = 0x00;
    uint8_t u2 = 0x59;      uint8_t u4 = 0x59;

    int time=25000;
    myI2c.writeByte(u0,u1);    usleep(time);
    myI2c.writeByte(u0,u2);    usleep(time);
    myI2c.writeByte(u0,u3);    usleep(time);
    myI2c.writeByte(u0,u4);    usleep(time);
}

//Función encargada de imprimir la letra "n"
void word_n() {
    myI2c.open(ReadWrite | NonBlock );

    uint8_t n1 = 0x4d;      uint8_t n3 = 0xed;  uint8_t n0 = 0x00;
    uint8_t n2 = 0x49;      uint8_t n4 = 0xe9;

    int time=25000;
    myI2c.writeByte(n0,n1);    usleep(time);
    myI2c.writeByte(n0,n2);    usleep(time);
    myI2c.writeByte(n0,n3);    usleep(time);
    myI2c.writeByte(n0,n4);    usleep(time);
}

//Función encargada de imprimir la letra "m"
void word_m() {
    myI2c.open(ReadWrite | NonBlock );

    uint8_t m1 = 0x4d;      uint8_t m3 = 0xdd;  uint8_t m0 = 0x00;
    uint8_t m2 = 0x49;      uint8_t m4 = 0xd9;

    int time=25000;
    myI2c.writeByte(m0,m1);    usleep(time);
    myI2c.writeByte(m0,m2);    usleep(time);
    myI2c.writeByte(m0,m3);    usleep(time);
    myI2c.writeByte(m0,m4);    usleep(time);
}

```

```

//Programa Principal
int main() {

    int time2=250000;
//variable que usamos para generar un retraso en el tiempo de ejecucion
//de cada funcion.

    cout << "Programa de muestra de la interfaz I2C con el Display
1602A" << endl;
    lcd_config();
    usleep(time2);
    word_h();
    usleep(time2);
    word_o();
    usleep(time2);
    word_l();
    usleep(time2);
    word_a();
    usleep(time2);
    word_();
    usleep(time2);
    word_u();
    usleep(time2);
    word_n();
    usleep(time2);
    word_a();
    usleep(time2);
    word_m();
    return 0;
}

```

Apéndice 9. Tablas de datos para la configuración serial del display 1602A.

1.- Bytes para la configuración del LCD

1602A	RS	R/W	E	LED	D4	D5	D6	D7(M.S)	Dato en hexadecimal
PCF8574	P0	P1	P2	P3	P4	P5	P6	P7	
Bytes para la configuración del LCD									
	0	0	0	1	1	1	0	0	38
	0	0	1	1	1	1	0	0	3C
	0	0	0	1	1	1	0	0	38
	0	0	1	1	1	1	0	0	3C
	0	0	0	1	1	1	0	0	38
	0	0	1	1	1	1	0	0	3C
	0	0	0	1	1	1	0	0	38
	0	0	0	1	0	1	0	0	28
	0	0	1	1	0	1	0	0	2C
	0	0	0	1	0	1	0	0	28
	0	0	1	1	0	1	0	0	2C
	0	0	0	1	0	1	0	0	28
	0	0	0	1	1	0	0	0	18
	0	0	1	1	1	0	0	0	1C
	0	0	0	1	1	0	0	0	18
	0	0	0	1	0	0	0	0	08
	0	0	1	1	0	0	0	0	0C
	0	0	0	1	0	0	0	0	08
	0	0	0	1	1	1	1	1	F8
	0	0	1	1	0	0	0	0	FC
	0	0	0	1	0	0	0	0	F8
Bytes para mostrar la letra "H"									
	1	0	1	1	0	0	1	0	4D
	1	0	0	1	0	0	1	0	49
	1	0	1	1	0	0	0	1	8D
	1	0	0	1	0	0	0	1	89
Bytes para mostrar la letra "O"									
	1	0	1	1	0	0	1	0	4D
	1	0	0	1	0	0	1	0	49
	1	0	1	1	1	1	1	1	FD
	1	0	0	1	1	1	1	1	F9
Bytes para mostrar la letra "L"									
	1	0	1	1	0	0	1	0	4D
	1	0	0	1	0	0	1	0	49
	1	0	1	1	0	0	1	1	CD
	1	0	0	1	0	0	1	1	C9
Bytes para mostrar la letra "A"									
	1	0	1	1	0	0	1	0	4D
	1	0	0	1	0	0	1	0	49
	1	0	1	1	1	0	0	0	1D

	1	0	0	1	1	0	0	0	19
Bytes para mostrar el carácter “-”									
	1	0	1	1	0	1	0	0	2D
	1	0	0	1	0	1	0	0	29
	1	0	1	1	1	0	1	1	DD
	1	0	0	1	1	0	1	1	D9
Bytes para mostrar la letra “U”									
	1	0	1	1	1	0	1	0	5D
	1	0	0	1	1	0	1	0	59
	1	0	1	1	1	0	1	0	5D
	1	0	0	1	1	0	1	0	59
Bytes para mostrar la letra “N”									
	1	0	1	1	0	0	1	0	4D
	1	0	0	1	0	0	1	0	49
	1	0	1	1	0	1	1	1	ED
	1	0	0	1	0	1	1	1	E9
Bytes para mostrar la letra “M”									
	1	0	1	1	0	0	1	0	4D
	1	0	0	1	0	0	1	0	49
	1	0	1	1	1	0	1	1	DD
	1	0	0	1	1	0	1	1	D9

Apéndice 10. Código para activar el SPI0.

```
/*
 * Copyright (C) 2013 CircuitCo
 *
 * Virtual cape for SPI0 on connector pins P9.22 P9.21 P9.18 P9.17
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

/dts-v1/;
/plugin/;

/ {
    compatible = "ti,beaglebone", "ti,beaglebone-black";

    /* identification */
    part-number = "BB-SPI0";
    version = "00A0";

    /* state the resources this cape uses */
    exclusive-use =
        /* the pin header uses */
        "P9.17",      /* spi0_cs0 */
        "P9.18",      /* spi0_d1 */
        "P9.21",      /* spi0_d0 */
        "P9.22",      /* spi0_sclk */
        /* the hardware ip uses */
        "spi0";

    fragment@0 {
        target = <&am33xx_pinmux>;
        __overlay__ {
            /* default state has all gpios released and mode
set to uart1 */
            bb_spi0_pins: pinmux_bb_spi0_pins {
                pinctrl-single,pins = <
                    0x150 0x30 /*
spi0_sclk.spi0_sclk, INPUT_PULLUP | MODE0 */
                    0x154 0x30 /* spi0_d0.spi0_d0,
INPUT_PULLUP | MODE0 */
                    0x158 0x10 /* spi0_d1.spi0_d1,
OUTPUT_PULLUP | MODE0 */
                    0x15c 0x10 /*
spi0_cs0.spi0_cs0, OUTPUT_PULLUP | MODE0 */
                >;
            };
        };
    };

    fragment@1 {
        target = <&spi0>; /* spi0 is numbered correctly */
        __overlay__ {
            #address-cells = <1>;
        };
    };
};
```


Apéndice 11. Código de configuración de V4l2.

* Este código fue tomado de:

*<https://github.com/derekmolloy/exploringBB/blob/master/chp12/fswebcam/fswebcam.conf>

```
device /dev/video0
input 0
resolution 1280x720
bottom-banner
font /usr/share/fonts/truetype/ttf-dejavu/DejaVuSans.ttf
title "Exploring BeagleBone"
timestamp "%H:%M:%S %d/%m/%Y (%Z)"
png 0
save exploringBB.png
```

Apéndice 12. Programa que captura y procesa una imagen con OpenCV.

```
/* Programa que procesa imagines empleando Open_CV.
   Este programa fue tomado y modificado de:
   https://github.com/derekmolloy/exploringBB/blob/master/chp12/openCV/boneC
   V.cpp
*/

#include<iostream>
#include<opencv2/opencv.hpp> // C++ OpenCV include file
using namespace std;
using namespace cv;          // using the cv namespace too

int main()
{
    VideoCapture capture(0); // capturing from /dev/video0

    cout << "Started Processing - Capturing Image" << endl;
    // set any properties in the VideoCapture object

    capture.set(CV_CAP_PROP_FRAME_WIDTH,1280); // width pixels
    capture.set(CV_CAP_PROP_FRAME_HEIGHT,720); // height pixels
    capture.set(CV_CAP_PROP_GAIN, 0);          // Enable auto gain etc.
    if(!capture.isOpened()){ // connect to the camera
        cout << "Failed to connect to the camera." << endl;
    }

    Mat frame, gray, edges; // images for original, grayscale and edge

    capture >> frame;        // capture the image to the frame
    if(frame.empty()){ // did the capture succeed?
        cout << "Failed to capture an image" << endl;
        return -1;
    }
    cout << "Processing - Performing Image Processing" << endl;
    cvtColor(frame, gray, CV_BGR2GRAY); // convert to grayscale
    blur(gray, edges, Size(3,3)); // blur grayscale image 3x3kernel

    // use Canny edge detector that outputs to the same image
    // low threshold = 10, high threshold = 30, kernel size = 3

    Canny(edges, edges, 10, 30, 3); // Run Canny edge detector
    cout << "Finished Processing - Saving images" << endl;

    imwrite("capture.png", frame);
    imwrite("grayscale.png", gray);
    imwrite("edges.png", edges);
    return 0;
}
```

Apéndice 13. Programa que reconoce rostros humanos empleando OpenCV.

```
/*
Programa que detecta rostros humanos.

Este programa fue tomado y modificado de:
https://github.com/derekmolloy/exploringBB/blob/master/chp12/openCV/face.
cpp
*/

#include <iostream>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/imgproc/imgproc.hpp>
using namespace std;
using namespace cv;

int main(int argc, char *args[])
{
    Mat frame;
    VideoCapture *capture; // capture needs full scope of main(),
    cout << "Starting face detection application" << endl;

    if(argc==2){ // loading image from a file
        cout << "Loading the image " << args[1] << endl;
        frame = imread(args[1], CV_LOAD_IMAGE_COLOR);
    }

    else {
        cout << "Capturing from the webcam" << endl;
        capture = new VideoCapture(0);
        // set any properties in the VideoCapture object
        capture->set(CV_CAP_PROP_FRAME_WIDTH,1280); // width pixels
        capture->set(CV_CAP_PROP_FRAME_HEIGHT,720); // height pixels
        if(!capture->isOpened()){ // connect to the camera
            cout << "Failed to connect to the camera." << endl;
            return 1;
        }

        *capture >> frame; // populate the frame with captured image
        cout << "Successfully captured a frame." << endl;
    }

    if (!frame.data){
        cout << "Invalid image data... exiting!" << endl;
        return 1;
    }

    // loading the classifier from a file (standard OpenCV example classifier

    CascadeClassifier faceCascade;
    faceCascade.load("haarcascade_frontalface.xml");

    // faces is a STL vector of faces - will store the detected faces
    std::vector<Rect> faces;
```

```

// detect objects in the scene using the classifier above
// (frame, faces, scale factor, min neighbors, flags, min size, max size

    faceCascade.detectMultiScale(frame, faces, 1.1, 3,
                                0 | CV_HAAR_SCALE_IMAGE, Size(50,50));
    if(faces.size()==0){
        cout << "No faces detected!" << endl;    // display the image
    anyway
    }

    // draw oval around the detected faces in the faces vector
    for(int i=0; i<faces.size(); i++)
    {

        // Using the center point and a rectangle to create an ellipse
        Point cent(faces[i].x+faces[i].width*0.5,
faces[i].y+faces[i].height*0.5);

        RotatedRect rect(cent, Size(faces[i].width,faces[i].width),0);
        // image, rectangle, color=green, thickness=3, linetype=8
        ellipse(frame, rect, Scalar(0,255,0), 3, 8);
        cout << "Face at: (" << faces[i].x << ", " <<faces[i].y << ")" <<
endl;
    }

    imshow("EBB OpenCV face detection", frame); // display image results
    imwrite("faceOutput.png", frame);         // save image too
    waitKey(0);                                 // display image until key
press
    return 0;
}

```

Apéndice 14. Código para activar el PRU0.

```
/* Archivo de kernel que activa el PRU0.
 * Este código fue obtenido de:
https://github.com/derekmolloy/exploringBB/blob/master/chp13/overlay/EBB-PRU-Example.dts
 */

/dts-v1/;
/plugin/;

/ {
    compatible = "ti,beaglebone", "ti,beaglebone-black";

    part-number = "EBB-PRU-Example";
    version = "00A0";

    /* This overlay uses the following resources */
    exclusive-use =
        "P9.11", "P9.13", "P9.27", "P9.28", "pru0";

    fragment@0 {
        target = &am33xx_pinmux;
        __overlay__ {

            gpio_pins: pinmux_gpio_pins {          // The GPIO pins
                pinctrl-single,pins = <
                    0x070 0x07 // P9_11 MODE7 | OUTPUT | GPIO pull-down
                    0x074 0x27 // P9_13 MODE7 | INPUT | GPIO pull-down
                >;
            };

            pru_pru_pins: pinmux_pru_pru_pins {    // The PRU pin modes
                pinctrl-single,pins = <
                    0x1a4 0x05 // P9_27 pr1_pru0_pru_r30_5, MODE5 | OUTPUT |
                    0x19c 0x26 // P9_28 pr1_pru0_pru_r31_3, MODE6 | INPUT | P
                >;
            };
        };
    };

    fragment@1 {          // Enable the PRUSS
        target = &pruss;
        __overlay__ {
            status = "okay";
            pinctrl-names = "default";
            pinctrl-0 = <&pru_pru_pins>;
        };
    };
};
```

```
fragment@2 {           // Enable the GPIOs
    target = <&ocp>;
    __overlay__ {
        gpio_helper {
            compatible = "gpio-of-helper";
            status = "okay";
            pinctrl-names = "default";
            pinctrl-0 = <&gpio_pins>;
        };
    };
};
```

Apéndice 15. Programa en ensamblador para el GPIO de tiempo real.

```
// Programa que hace parpadear un LED:
// Este programa fue obtenido de:
//https://github.com/derekmolloy/exploringBB/blob/master/chp13/ledButton/
//ledButton.p

.origin 0 // start of program in PRU memory
.entrypoint START // program entry point (for a debugger)

#define INS_PER_US 200 // 5ns per instruction
#define INS_PER_DELAY_LOOP 2 // two instructions per delay loop
// set up a 50ms delay
#define DELAY 50 * 1000 * (INS_PER_US / INS_PER_DELAY_LOOP)

#define PRU0_R31_VEC_VALID 32 // allows notification of program
completion
#define PRU_EVTOUT_0 3 // the event number that is sent back

START:
    SET r30.t5 // turn on the output pin (LED on)
    MOV r0, DELAY // store the length of the delay in REG0
DELAYON:
    SUB r0, r0, 1 // Decrement REG0 by 1
    QBNE DELAYON, r0, 0 // Loop to DELAYON, unless REG0=0
LEDOFF:
    CLR r30.t5 // clear the output bin (LED off)
    MOV r0, DELAY // Reset REG0 to the length of the delay
DELAYOFF:
    SUB r0, r0, 1 // decrement REG0 by 1
    QBNE DELAYOFF, r0, 0 // Loop to DELAYOFF, unless REG0=0

    QBBC START, r31.t3 // is the button pressed? If not, loop

END: // notify the calling app that finished
    MOV R31.b0, PRU0_R31_VEC_VALID | PRU_EVTOUT_0
    HALT // halt the pru program
```


Apéndice 16. Programa en C que implementa el uso del PRU0 con un LED.

```
/* Programa que hace parpadear un led hasta que es presionado un botón.

Este programa fue tomado y modificado de:
https://github.com/derekmolloy/exploringBB/blob/master/chp13/ledButton/ledButton.c
*/

#include <stdio.h>
#include <stdlib.h>
#include <prussdrv.h>
#include <pruss_intc_mapping.h>

#define PRU_NUM0 // using PRU0 for these examples

int main (void)
{
    if(getuid()!=0){
        printf("Debes correr este programa como usuario root.\n");
        exit(EXIT_FAILURE);
    }
    // Initialize structure used by prussdrv_pruwait_intc
    // PRUSS_INTC_INITDATA is found in pruss_intc_mapping.h
    tpruss_intc_initdata pruss_intc_initdata = PRUSS_INTC_INITDATA;

    // Allocate and initialize memory
    prussdrv_init ();
    prussdrv_open (PRU_EVTOUT_0);

    // Map PRU's interrupts
    prussdrv_pruwait_intc_init(&pruss_intc_initdata);

    // Load and execute the PRU program on the PRU
    prussdrv_exec_program (PRU_NUM, "./ledButton.bin");

    // Wait for event completion from PRU, returns the PRU_EVTOUT_0 number
    int n = prussdrv_pru_wait_event (PRU_EVTOUT_0);
    printf("EBB PRU programa finalizado, ejecucion del programa # %d.\n",
n);

    // Disable PRU and close memory mappings
    prussdrv_pru_disable(PRU_NUM);
    prussdrv_exit ();
    return EXIT_SUCCESS;
}
```

Apéndice 17. Programa en ensamblador para el funcionamiento del HC-SR04.

```
// Programa en ensamblador para controlar el sensor HC-SR04.
// Este código fue obtenido de:
//https://github.com/derekmolloy/exploringBB/blob/master/chp13/ultrasonic
///ultrasonic.p

.origin 0 // offset of start of program in PRU memory
.entrypoint START // program entry point used by the debugger

#define TRIGGER_PULSE_US 10
#define INS_PER_US 200
#define INS_PER_LOOP 2
#define TRIGGER_COUNT (TRIGGER_PULSE_US * INS_PER_US) /
INS_PER_LOOP
#define SAMPLE_DELAY_1MS (1000 * INS_PER_US) / INS_PER_LOOP
#define PRU0_R31_VEC_VALID 32;
#define PRU_EVTOUT_0 3
#define PRU_EVTOUT_1 4

// Using register 0 for all temporary storage (reused multiple times)
START:
// Read number of samples to read and inter-sample delay
MOV r0, 0x00000000 //load the memory location
LBBO r1, r0, 0, 4 //load the value into memory - keep r1

// Read the sample delay

MOV r0, 0x00000004 //the sample delay is the second 32 bits
LBBO r2, r0, 0, 4 //the sample delay is stored in r2

MAINLOOP:
MOV r0, TRIGGER_COUNT //store length of the trigger pulse delay
SET r30.t5 //set the trigger high

TRIGGERING: // delay for 10us
SUB r0, r0, 1 // decrement loop counter
QBNE TRIGGERING, r0, 0 // repeat loop unless zero
CLR r30.t5 // 10us over, set the trigger low - pulse
// sent

// clear the counter and wait until the echo goes high

MOV r3, 0 // r3 will store the echo pulse width
WBS r31.t3 // wait until the echo goes high

// start counting (measuring echo pulse width) until the echo goes low

COUNTING:
ADD r3, r3, 1 // increment the counter by 1
QBBS COUNTING, r31.t3 // loop if the echo is still high
// at this point the echo is now low - write the value to shared
memory
```

```

MOV    r0, 0x00000008      // going to write the result to this
                          // address

SBBO   r3, r0, 0, 4       // store the count at this address

// one more sample iteration has taken place

SUB    r1, r1, 1          // take 1 away from the number of
//                          //iterations
MOV    r0, r2             // need a delay between samples

SAMPLEDELAY:              // do this loop r2 times (1ms delay each //
//                          //time)

SUB    r0, r0, 1          // decrement counter by 1
MOV    r4, SAMPLE_DELAY_1MS // load 1ms delay into r4

DELAY1MS:
SUB    r4, r4, 1
QBNE   DELAY1MS, r4, 0    // keep going until 1ms has elapsed
QBNE   SAMPLEDELAY, r0, 0 // repeat loop unless zero

// generate an interrupt to update the display on the host computer
MOV R31.b0, PRU0_R31_VEC_VALID | PRU_EVTOUT_1

QBNE   MAINLOOP, r1, 0    // loop if the no of iterations has not
passed

END:
MOV R31.b0, PRU0_R31_VEC_VALID | PRU_EVTOUT_0
HALT

```

Apéndice 18. Programa en C que implementa el uso del sensor HC-SR04.

```
/* Programa que controla el sensor HC-SR04 a traves de PRU0

Este codigo fue modificado de:
https://github.com/derekmolloy/exploringBB/blob/master/chp13/ultrasonic/ultrasonic.c
*/

#include <stdio.h>
#include <stdlib.h>
#include <prussdrv.h>
#include <pruss_intc_mapping.h>
#include <pthread.h>
#include <unistd.h>

#define PRU_NUM 0

static void *pru0DataMemory;
static unsigned int *pru0DataMemory_int;

void *threadFunction(void *value){
    do {
        int notimes = prussdrv_pru_wait_event (PRU_EVTOUT_1);
        unsigned int raw_distance = *(pru0DataMemory_int+2);
        float distcm = ((float)raw_distance / (100 * 58));
        printf("Objeto encontrado a %f cm)          \r", distcm);
        prussdrv_pru_clear_event (PRU_EVTOUT_1, PRU0_ARM_INTERRUPT);
    } while (1);
}

int main (void)
{
    if(getuid()!=0){
        printf("Debes correr este programa como usuario root.\n");
        exit(EXIT_FAILURE);
    }
    pthread_t thread;
    tpruss_intc_initdata pruss_intc_initdata = PRUSS_INTC_INITDATA;

    // Allocate and initialize memory
    prussdrv_init ();
    prussdrv_open (PRU_EVTOUT_0);
    prussdrv_open (PRU_EVTOUT_1);

    // Map PRU's INTC
    prussdrv_pruintc_init(&pruss_intc_initdata);

    // Copy data to PRU memory - different way
    prussdrv_map_prumem(PRUSS0_PRU0_DATARAM, &pru0DataMemory);
    pru0DataMemory_int = (unsigned int *) pru0DataMemory;

    // Use the first 4 bytes for the number of samples
    *pru0DataMemory_int = 500;
}
```

```

    // Use the second 4 bytes for the sample delay in ms
    *(pru0DataMemory_int+1) = 100;    // 2 milli seconds between samples

    // Load and execute binary on PRU
printf("  \n");
printf("Programa que mide la distancia a la que se encuentra un objeto.
\n");

printf(" \n");
prussdrv_exec_program (PRU_NUM, "./ultrasonic.bin");
if(pthread_create(&thread, NULL, &threadFunction, NULL)){
    printf("Fallo la creacion de thread!");
}

int n = prussdrv_pru_wait_event (PRU_EVTOUT_0);

printf("Programa de deteccion de distancias finalizado, ejecucion numero:
%d.\n", n);

printf("Los detalles de la ejecucion del programa se muestran a
continuacion: \n");

printf(" \n");
printf("- El numero de muestras tomadas fue: %d.\n",*pru0DataMemory_int);
printf("- El tiempo de retardo es de %d ", *(pru0DataMemory_int+1));
    printf(" milisegundos \n");

    // distance in cm = time (ms) / 58 according to datasheet
    unsigned int raw_distance = *(pru0DataMemory_int+2);
    float distcm = ((float)raw_distance / (100 * 58));

printf("- La ultima lectura de distancia tomada es de %f cm.\n", distcm);
printf("  \n");

/* Disable PRU and close memory mappings */
prussdrv_pru_disable(PRU_NUM);
prussdrv_exit ();
return EXIT_SUCCESS;
}

```

