METODOLOGÍA DE CONSTRUCCIÓN DE DISTRIBUCIONES PARA PLATAFORMAS DE CÓMPUTO SOBRE HARDWARE ABIERTO EMBEBIDO

PABLO JOSUE ROJAS YEPES

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2018

METODOLOGÍA DE CONSTRUCCIÓN DE DISTRIBUCIONES PARA PLATAFORMAS DE CÓMPUTO SOBRE HARDWARE ABIERTO EMBEBIDO

PABLO JOSUE ROJAS YEPES

Trabajo de grado presentado para optar el título de ingeniero de SISTEMAS E INFORMÁTICA

Director
CARLOS JAIME BARRIOS HERNÁNDEZ
Doctor en Informática y Ciencias Computacionales

Codirector
GILBERTO JAVIER DÍAZ TORO
MSc. en ciencias de la computación

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2018

DEDICATORIA

Quiero dedicar este logro:

A mis padres Consuelo del Socorro Yepes y Pablo Rojas Herrera quienes, con su apoyo incondicional, esfuerzo y confianza hicieron posible este logro.

A mi familia, que siempre me ha apoyado en todo lo que han podido y me han aconsejado de la mejor forma, que en el tiempo que he compartido con ellos me ha enseñado una gran parte de las cosas que se y la persona que soy es gracias a ellos; son un ejemplo a seguir.

A mi hermana, quien es un apoyo fundamental para este estudiante perezoso.

También a Luz Mary Quiceno y Yidna Buitrago Caballero por sus cuidados y ayuda durante el tiempo que estuve fuera de casa, mis segundas madres.

A la fiel hermandad de la Iglesia Adventista del Séptimo Día, su ayuda fue imprescindible.

AGRADECIMIENTOS

A la Universidad Industrial de Santander por permitirme desarrollar mis competencias como profesional y persona, por su alta calidad de educación, brindando espacios para conocer personas notables y grandes profesores a lo largo de la carrera. También por la disposición de sus instalaciones y recursos a la comunidad estudiantil.

A la escuela de Ingeniería de Sistemas e Informática, en especial a la secretaria Lady por su paciencia y gestión a lo largo del proyecto, al SC3UIS por brindarme el espacio y recursos necesarios que fueron la base para el desarrollo de este proyecto de grado.

Al profesor Carlos Jaime Barrios por haber creído en el proyecto y aceptado la dirección del mismo, quien con su apoyo, correcciones y dedicación hizo posible el éxito de este proyecto.

Al profesor Gilberto Javier Diaz Toro, por su dirección, gracias por cada una de las enseñanzas, correcciones, paciencia, apoyo y orientación incondicional, compartiendo su conocimiento y experiencia para el desarrollo de este proyecto.

CONTENIDO

LISTA DE FIGURAS

Pág.

LISTA DE TABLAS

Pág.

LISTA DE GRAFICAS

Pág.

LISTA DE ANEXOS

Pág.

RESUMEN

TÍTULO: METODOLOGÍA DE CONSTRUCCIÓN DE DISTRIBUCIONES PARA PLATAFORMAS DE CÓMPUTO SOBRE HARDWARE ABIERTO EMBEBIDO.*

AUTOR: Pablo Josue Rojas Yepes.**

PALABRAS CLAVE: Arquitectura, Infraestructura, Plataforma de Sistemas, Kernel, Sistema Embebido, SBC (Single Board Computer o Computador en una Tarjeta), Linux, Buildroot, Yocto, Sistema Operativo (SO).

DESCRIPCIÓN:

Con el aumento de los dispositivos móviles, el hardware embebido ha entrado en auge debido a su gran versatilidad y multitud de usos, estos pequeños dispositivos o SBC dan la posibilidad de tener la potencia computacional que hace unos diez años proveía un equipo de escritorio. Existe una gran cantidad de SBC en el mercado y continuamente aparecen nuevos modelos con mejores especificaciones, se encuentran en todos los tamaños y colores, la relación precio/benéfico es excelente pero debido a la gran diversidad de hardware que se usa para construir las SBC se crearon iniciativas de SO para administrarlas.

Estos SO administran de buena manera las SBC pero lo hacen de forma genérica, aunque se ha invertido tiempo y esfuerzo en desarrollar estas distribuciones, la gran mayoría logra sacar solo un porcentaje de todo el potencial que ofrecen estas tarjetas, otra desventaja es que estos SO están orientados a propósitos genéricos por lo que se tiene una amplia gama de herramientas que pueden ser innecesarias a la hora de destinar las SBC a una tarea específica.

Estas desventajas manifiestan la necesidad de la construcción de un SO que no solo explote el potencial de la tarjeta, sino que se construya a la medida de la tarea que se desee desarrollar con la misma.

* Trabajo de grado

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director Phd. Carlos Jaime Barrios. Codirector Msc. Gilberto Javier Diaz Toro.

ABSTRACT

TITLE: METHODOLOGY OF BUILDING DISTRIBUTIONS FOR COMPUTER PLATFORMS ON EMBEDDED OPEN HARDWARE.*

AUTHOR: Pablo Josue Rojas Yepes.**

KEY WORDS: Architecture, Infrastructure, Systems Platform, Kernel, Embedded System, SBC (Single Board Computer or Computer on a card), Linux, Buildroot, Yocto.

DESCRIPTION:

With the increase of mobile devices, embedded hardware has entered the boom due to its great versatility and multitude of uses, these small devices or SBC give the possibility of having the computational power that about ten years ago provided a desktop computer. There is a large amount of SBC in the market and new models with better specifications are continuously appearing, they are available in all sizes and colors, the price / benefit ratio is excellent but due to the great diversity of hardware used to build SBCs They created SO initiatives to manage them.

These OSs administer the SBCs in a good way but they do it in a generic way, although time and effort have been invested in developing these distributions, the great majority manages to take out only a percentage of the full potential offered by these cards, another disadvantage is that these SOs they are oriented to generic purposes so there is a wide range of tools that may be unnecessary when allocating SBCs to a specific task.

These disadvantages manifest the need for the construction of an OS that not only exploits the potential of the card but is built to fit the task you want to develop with it.

* Degree work

** Faculty of Physical-Mechanical Engineering. School of Systems and Information Engineering. Director Phd. Carlos Jaime Barrios. Codirector Msc. Gilberto Javier Diaz Toro.

INTRODUCCIÓN

Para administrar la gran cantidad de hardware abierto embebido se han desarrollado sistemas operativos (SO) que los administran, el problema es que no todo el hardware soporta la implementación de una distribución completa, lo que limita el hardware usado y obliga al uso de versiones lite, personalizadas, desactualizadas o genéricas, aumentando en gran manera el tiempo de desarrollo de un proyecto, esto hace necesario el uso de una metodología de construcción de distribuciones que no solo se enfoquen en administrar el hardware sino en la personalización de la tarea que se quiera realizar.

Se han generado una gran variedad de proyectos que han dado forma al hardware embebido y sus implementaciones hoy en día, sumado a que poco a poco se están convirtiendo en parte esencial de la vida cotidiana, esto obliga al desarrollo de sistemas operativos (SO) que administren estos dispositivos y puedan dar el mejor rendimiento para los mismos.

Iniciaremos con el porqué es necesario una metodología de construcción de distribuciones para hardware embebido, seguido de los antecedentes y objetivo, se dará un marco teórico y metodológico de cómo se desarrolla la metodología. Se compararán diferentes tarjetas (SBC) ofrecidas por el mercado a la fecha de iniciar el proyecto para conocer sus especificaciones y desempeño, una vez elegida la tarjeta, se probarán varios SO populares midiendo su desempeño para luego ser contrastados por SO generados por la metodología. Por último, se elegirá una herramienta para la construcción de SO embebido, seguido de la metodología de cómo construir dicho SO, sumado a las pruebas de rendimiento de los diferentes SO.

1. GENERALIDADES

1.1 PLANTEAMIENTO Y JUSTIFICACIÓN DEL PROBLEMA

Con el aumento de los dispositivos móviles, se ha creado una oleada de hardware embebido, esto trajo un poder de cómputo enorme en una presentación minimalista, con un consumo energético y costo de fabricación bajos. La versatilidad de estas tarjetas permite una amplia gama de usos, pero al estar limitados por la poca cantidad de SO, se hace necesario desarrollar SO personalizados tanto para la tarjeta como para la tarea que se va a desempeñar. De la cantidad de SO, buena parte se ha dejado de atender debido a su poca popularidad, problemas de administración del hardware o incompatibilidades con programas de uso necesario en las SBC (Single Board Computer, Computadora en una Tarjeta).[1]

De los Sistemas Operativos (SO) existentes, la mayoría los administran de manera genérica, lo que permite el uso de una buena parte del hardware disponible, pero limita su potencial, haciendo necesario el desarrollo de una metodología de construcción de distribuciones que no solo se enfoque en sacar todo el potencial que ofrecen sino apuntar al uso que se les quiera dar. Se espera que las pruebas físicas de la tarjeta elegida corroboren los datos obtenidos de las pruebas publicadas por fabricantes y comunidad usando los SO más populares. El rendimiento de la RAM, Almacenamiento y Red, dan como resultado un manejo similar, cuando se compara su manejo de CPU se ven los cambios, hay que tener en cuenta que para algunas SBC (Single Board Computer, Computadoras en una Tarjeta) se cuenta con un procesador a 64 bit, la mayoría de SO están enfocados a los 32 bit.

[1] DUBEY A., KARSAI G., GOKHALE A., EMFINGER W. & KUMAR P. Drems-Os: An Operating System For Managed Distributed Real-Time Embedded Systems. Nashville: Vanderbilt University. 2017

Aunque el rendimiento de un SO de 64 bit aumenta en ciertas tareas, solo lo hace en gran manera en la CPU, manteniendo los valores de RAM, almacenamiento y red muy parecidos con los SO de 32 bits, esto deja claro la necesidad de construir SO que puedan administrar la SBC con el mayor rendimiento sin comprometer la capacidad de configurarla para un propósito general o específico, pero para construir dicho SO se deben tener en cuanta algunos antecedentes que serán tratados en la siguiente sección[2].

1.2 ANTECEDENTES

Para que hoy en día las SBC (Single Board Computer, Computadora en una Tarjeta) sean lo que son, han tenido un largo camino, OpenSPARC[1] es un proyecto de hardware iniciado por la empresa Sun Microsystems que cuenta con una licencia GPLv2. OpenRISC[3] es un diseño de CPU RISC libre realizado por OpenCores con una licencia LGLP, otro microprocesador es LEON[4], está basado en una arquitectura RISC de 32-bits con un conjunto de instrucciones SPARC-V8 diseñado por ESTEC(European Space Research and Technology Centre), el Freedom CPU Project[5] es un trabajo de colaboración abierto, su objetivo es crear y distribuir código fuente de un microprocesador de alto rendimiento bajo licencia copyleft.

Por otra parte, Arduino[6] es una plataforma libre con su propio entorno de desarrollo y programable en múltiples lenguajes, el BUG[7] al igual que arduino es

[2] BRINKSCHULTE, U. Technical Report: Artificial Dna - A Concept For Self-Building Embedded Systems. Frankfurt: Johann Wolfgang Goethe Universit. 2018.
[1] ORACLE. OpenSPARC Overview [En línea]. s.f. Disponible en: http://www.oracle.com/technetwork/systems/opensparc/index.html
[3] OPENCORES. Projects: OpenRISC [En línea]. s.f. Disponible en: http://opencores.org/projects
[4] ESA. Leon: the making of a microprocessor for space [En línea]. 2013. Disponible en: http://www.esa.int/Our_Activities/Space_Engineering_Technology/LEON_the_making_of_a_microprocessor_for_space
[5] F-CPU [En línea]. 2017. Disponible en: http://f-cpu.seul.org/
[6] ARDUINO. Home web site [En línea]. s.f. Disponible en: https://www.arduino.cc/
[7] BUG LABS. Acerca de Bug Labs [En línea]. s.f. Disponible en: http://buglabs.net/about

una plataforma abierta enfocada a la filosofía DIY (“Hágalo usted mismo” por sus siglas inglés, “Do It Yourself”) y producida por BUG LABS.

En proyectos más complejos nos encontramos con la Beelink[8], una compañía basada en la generación de productos usando hardware embebido, entre sus productos se encuentra Beelink S1, M1, AP34, BT3. Los productos generados por Banana Pi[9] (BPI) son open source, su línea de productos tiene diferentes computadoras en una placa (por sus siglas en inglés, SBC) con diferentes capacidades y funciones, las SBC pueden ejecutar diferentes distribuciones de linux haciéndolas perfectas para proyectos DIY. Otras SBC que destacan son las producidas por Orange Pi[10] (OPI), al igual que BPI es open source, pueden ejecutar sistemas como android 4.4, Ubuntu, debían o raspbian, están enfocadas a la creación de tecnología de manera simple y utilitaria. De esta misma forma podemos encontrarnos con ODROID[11], al igual que BPI y OPI tiene su línea de SBC open source, destacan la ODROID XU4, C2, C1+. Muchos han contribuido para llegar a lo que hoy conocemos como SBC, en el camino muchos desaparecieron, pero quienes les siguieron no dejaron que sus esfuerzos fueran en vano, como en todo, debe haber un líder y ese puesto por ahora pertenece a la Fundación Raspberry Pi.

La Fundación Raspberry Pi[12] trabaja para poner el poder digital en las manos del mundo, facilitando el entendimiento y forma de nuestro mundo cada vez más digital, resolviendo problemas que les importan y preparándolos para el futuro. Proporcionan computadoras de bajo costo y alto rendimiento que pueden ser usadas para aprender, solucionar problemas y divertirse. Desarrollan recursos gratuitos para ayudar a personas a aprender sobre computación y como hacer

[8] BEELINK. Portal web [En línea]. s.f. Disponible en: http://www.bee-link.com/portal.php
[9] BANANA PI. Página principal [En línea]. s.f. Disponible en: http://www.banana-pi.org/#sbpc
[10] ORANGE PI. Página principal [En línea]. s.f. Disponible en: http://www.orangepi.org/
[11] HARDKERNEL. Productos ODROID [En línea]. s.f. Disponible en: http://www.hardkernel.com/main/main.php
[12] Fundación Raspberry Pi [En línea]. s.f. Disponible en: https://www.raspberrypi.org/

cosas con estas SBC, capacitándolos para guiar a otras personas a aprender. Con esto en mente podemos plantear a continuación el objetivo general y los objetivos específicos que darán forma al proyecto en el siguiente titular.

1.3 OBJETIVO

1.3.1 Objetivo general. Proponer una metodología de construcción de distribuciones computacionalmente eficientes para plataformas de cómputo de alto desempeño sobre hardware abierto embebido.

1.3.2 Objetivos específicos.

- Caracterizar las distribuciones más utilizadas en la Raspberry Pi 3 para conocer las prestaciones de CPU, RAM, almacenamiento y red.

- Revisar y seleccionar herramientas para la construcción de distribuciones.

- Desarrollar la metodología de construcción de distribuciones para plataformas de cómputo de alto rendimiento sobre hardware abierto embebido.

- Comparar la distribución desarrollada por la metodología con las distribuciones disponibles en internet para conocer su rendimiento en CPU, RAM, almacenamiento y red definiendo una metodología de evaluación.

Para cumplir los objetivos planteados debemos usar un marco teórico y metodológico para fundar las bases del proyecto, esto será tratado en el siguiente capítulo.

2. MARCO TEÓRICO Y METODOLÓGICO

2.1 MARCO TEÓRICO

Debemos conocer ante todo que hardware se va a emplear, seguido de, ¿qué es un sistema operativo? y ¿cómo está compuesto? para así proponer una metodología de construcción del mismo. Debemos de saber que son la mayoría de elementos mencionados y como afectaran al proyecto.

2.1.1 SBC o Computadora en una tarjeta (hardware). Las SBC son computadoras de una sola tarjeta. CPU de uno o varios núcleos, RAM, I/O, GPU y demás características de un computador en una placa base de tamaño reducido. Son más livianas, pequeñas, con mejor manejo, confiabilidad y mayor aprovechamiento de la potencia eléctrica que un computador de múltiples tarjetas. Al igual que toda computadora se hace necesario un So para administrarla.

2.1.2 Sistema Operativo (SO). Es el software que gestiona los recursos del hardware y provee servicios, algunos pueden solo ejecutar un proceso en un momento dado, otros asignan los recursos disponibles de manera alternada a los procesos que los solicitan. Los programas pueden ser ejecutados por uno o varios usuarios al mismo tiempo y los recursos de una o varias máquinas pueden ser asignados por el SO.

En general los sistemas operativos poseen los siguientes componentes:

- **KERNEL:** es un software que constituye una parte fundamental del sistema operativo, principal responsable de facilitar a los distintos programas acceso seguro al hardware, encargado de gestionar recursos, a través de llamada al

sistema. Se encarga de decidir qué programa podrá usar un dispositivo de hardware y durante cuánto tiempo.

- **GESTIÓN DE PROCESOS:** es el responsable de crear y destruir procesos, parar y reanudar procesos, ofrecer mecanismos para que los procesos puedan comunicarse y se sincronicen.

- **GESTIÓN DE LA MEMORIA PRINCIPAL:** es el responsable de conocer qué partes de la memoria están siendo utilizadas y por quién, decidir qué procesos se cargarán en memoria cuando haya espacio disponible, asignar y reclamar espacio de memoria cuando sea necesario.

- **GESTIÓN DEL ALMACENAMIENTO SECUNDARIO:** ya que la memoria principal es volátil y en ciertos casos pequeña para almacenar todos los programas y datos. El SO se encarga de planificar los discos, gestionar el espacio libre, asignar el almacenamiento, verificar que los datos se guarden en orden.

- **GESTIÓN DE ENTRADA Y SALIDA (I/O):** es un almacenamiento temporal (caché), una interfaz de controladores de dispositivos y otra para dispositivos concretos. El SO debe gestionar la caché de E/S y servir las interrupciones de los dispositivos de E/S.

- **SISTEMA DE ARCHIVOS:** es responsable de construir, eliminar archivos y directorios, ofrecer funciones para manipular archivos y directorios, establecer la correspondencia entre archivos y unidades de almacenamiento, realizar copias de seguridad de archivos.

- **SISTEMAS DE PROTECCIÓN:** controla el acceso de los programas o los usuarios a los recursos del sistema. Se encarga de distinguir entre uso autorizado

y no autorizado, especificar los controles de seguridad a realizar, forzar el uso de estos mecanismos de protección.

- **SISTEMA DE COMUNICACIONES:** es necesario poder controlar el envío y recepción de información a través de las interfaces de red. También hay que crear y mantener puntos de comunicación que sirvan a las aplicaciones para enviar y recibir información, crear y mantener conexiones virtuales entre aplicaciones que están ejecutándose localmente y otras que lo hacen remotamente.

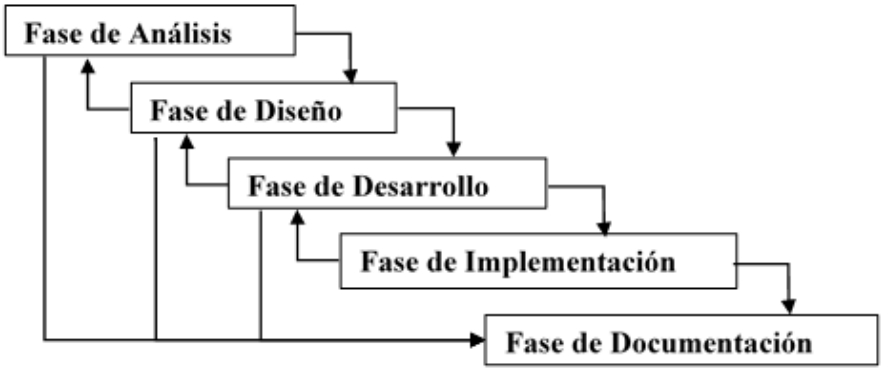
Cada uno de los temas tratados por la teoría se debe manejar y agregar al proyecto por medio de una metodología que dé como resultado el cumplimiento de los objetivos, esta metodología será tratada en el siguiente titular.

2.2 MARCO METODOLÓGICO

Durante el transcurso y desarrollo del proyecto se implementa un modelo de cascada con una variante, se puede hacer cambios en las fases anteriores para mejorar los resultados en la fase actual, solo se avanza si se está satisfecho con los resultados obtenidos. Esto también permite hacer una revisión al final de cada fase y ver el progreso del proyecto.

Se dio inicio al desarrollo de este proyecto partiendo de la pregunta, ¿Qué es necesario para la construcción de una plataforma computacionalmente eficiente basada en hardware abierto embebido?, después de un breve análisis se llegó a un punto crucial para la pregunta, un SO para la plataforma, la mayoría de sistemas existentes cumplían con normas genéricas de administración y no sacan todo el potencial que las SBC, de esta manera la pregunta cambio a el objetivo y título de este proyecto.

Figura 1. Modelo de Desarrollo en Cascada Modificada



- **Fase de Análisis**

Una vez se definió el problema, se hizo un análisis de cada uno de los puntos principales para la construcción de un SO para una SBC. Sumado a esto se analizaron diferentes SBC en el mercado para definir que SBC era apropiada para el proyecto y cómo afectaría al diseño, desarrollo e implementación de la metodología.

- **Fase de Diseño**

Usando los análisis hechos a las SBC y SO elegidos, se determinaron similitudes y diferencias entre sus características, esto facilita el diseño, mostrando los puntos comunes que deben ser cubiertos por la fase de desarrollo, dando bases para la elección de herramientas de desarrollo para el cumplimiento del objetivo del proyecto. Se debe tener en cuenta que con una nueva propuesta de diseño pueden surgir nuevos tópicos que deben ser analizados.

- **Fase de Desarrollo**

Basado en los análisis y diseños anteriores se eligen las herramientas necesarias para desarrollar los objetivos del proyecto, pueden surgir cambios en la forma en que se desarrollan los objetivos, esto podría cambiar los diseños y análisis hechos antes.

- **Fase de Implementación**

La implementación mantendrá las políticas usadas para evaluar las características de las SBC y SO usadas en la fase de análisis y diseño, puede que para cumplir con la implementación sea necesario modificar el desarrollo, diseño y análisis de la metodología.

- **Fase de Documentación**

Todos los documentos que se generaron durante el proyecto se deben revisar para ser compilados de una manera coherente en el documento final.

Una vez planteado la teoría y la metodología debemos iniciar el cumplimiento de los objetivos del proyecto, lo primero que debemos solucionar es la elección de la SBC que usaremos como base para comparar los SO populares y a la cual se van a enfocar los SO generados por la metodología, los primeros resultados los vamos a tratar en el siguiente capítulo.

3. RESULTADO DE LA COMPARACIÓN ENTRE SBC POPULARES

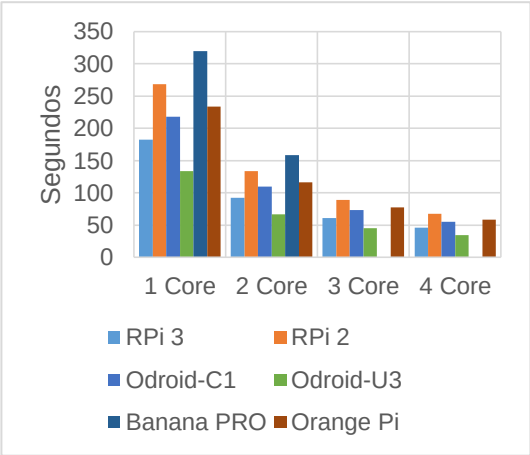
Debido a la gran cantidad de dispositivos embebidos ofrecidos en el mercado, se hace necesario el comparar sus prestaciones, al igual que sus precios para decidir que tarjeta es más representativa para el desarrollo del proyecto. Debido al objetivo del proyecto se debe buscar que, el dispositivo elegido se encuentre en la media de las especificaciones y precios de la mayoría de dispositivos embebidos ofrecidos por el mercado a la hora de iniciar el proyecto. En la Tabla 1, se listan las diferentes especificaciones de las SBC.

Tabla 1. Especificaciones de las SBC

	RPi3	**RPi2**	**Odroid-C1**	**Odroid-U3**	**Banana PRO**	**Orange Pi One H3**
CPU	Quad Core**,** ARM Cortex-A53 a 1.2GHz	Quad Core, ARM Cortex-A7 a 900MHz	Quad Core, ARM Cortex-A5 a 1.5GHz	Quad Core, ARM Cortex-A9 a 1.7GHz	Dual Core, ARM Cortex-A7 a 1GHz	Quad Core, ARM Cortex-A7 a 1.2GHz
GPU	Broadcom video core 4 a 300MHz	Broadcom video core 4 a 250MHz	Mali-450 MP2 a 600MHz	Mali-400 MP4 quad-core a 533 MHz	Mali400 MP2	Mali400 MP2 a 600MHz
RAM	1GB LPDDR2	1GB LPDDR2	1 GB DDR3	2 GB LPDDR2	1GB DDR3	512MB DDR3
Almacenamiento	Micro SD	Micro SD	Micro SD, eMMC	Micro SD, eMMC	Micro SD, SATA 2	TF card (Max. 32GB), MMC card
Red	Wi-Fi 802.11n, Bluetooth 4.1 & BLE, 10/100 Ethernet	10/100 Ethernet	10/100/1000 Ethernet	10/100 Ethernet	Wi-Fi 802.11b/g/n, 10/100/1000 Ethernet	10/100 Ethernet
o	35 – 50	35 – 45	35 - 50	70 - 90	47 - 60	30 – 45

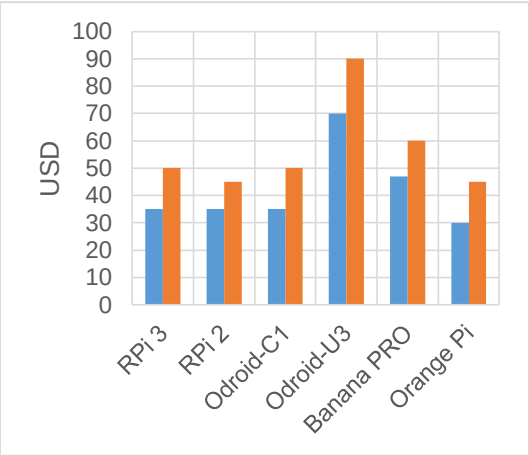
Fuente: Especificaciones de los Fabricantes

Grafica 1. Comparativa de CPU entre SBC.



Fuente: Pruebas Publicadas por Fabricantes y Comunidad.

Grafica 2. Comparativa de Precio entre SBC.



Fuente: Amazon, eBay, Mercado Libre.

La herramienta elegida para las pruebas en cada SBC es sysbench, sysbench es una herramienta de benchmark muilti-hilo basada en LuaJIT. Se usa con más frecuencia para pruebas de base de datos, pero puede usarse también para generar cargas de trabajo arbitrariamente complejas en multiplataforma (Linux, macOS, Windows), la distribución usada para las pruebas está basada en debian a 32 bits. La prueba para la CPU declarada en la Grafica 1, consta de completar la tarea de comprobar 10000 primos, usando 1, 2, 3 o 4 hilos, el resultado se da en segundos. La segunda comparación a tener en cuenta es el precio de cada SBC manifiesta en la Grafica 2, los valores se expresan en dólares estadounidenses y no incluyen envió.

La Odroid-U3 supera a todas las SBC pero debido a su precio y el anuncio de su salida del mercado se descarta como opción para el proyecto. Concluyendo que, gracias a su popularidad, amplia comunidad, bajo precio, y mantenerse muy cerca en todas las comparativas con la U3 se elige la Raspberry Pi 3 (RPi3) como la SBC a ser usada en el proyecto. El elemento que llevo a la RPi3 a mantenerse en

la pelea, es el SO, la mayoría de SBC usaron un SO genérico, que, aunque funcionaba bien, no sacaba todo el potencial de la tarjeta. Por esto se hace necesario el comparar las distribuciones más populares para la RPi3, tema a tratarse en el siguiente capítulo.

4. CARACTERIZACIÓN DE LAS DISTRIBUCIONES ELEGIDAS

Se toman tres distribuciones, la primera es Raspbian mostrada en la Tabla 2, una distribución enfocada en la Raspberry Pi, la segunda es Ubuntu Mate visible en la Tabla 3, una distribución usada en múltiples arquitecturas y por ultimo OpenSUSE expuesta en la Tabla 4, esta distribución nos permite usar el potencial de 64 bits que aporta el procesador de la RPi3.

Tabla 2. Especificaciones y Descripción Raspbian

Raspbian	Tipo de OS	Linux
La distribución es ligera para moverse ágilmente en el hardware de la Raspberry Pi, comenzó con un entorno de escritorio LXDE y Midori como navegador web predeterminado, pero la Raspberry Pi Foundation ha creado un entorno de escritorio especial llamado **PIXEL** (Pi Improved Xwindows Environment Lightweight). Además incluye herramientas de desarrollo muy interesantes, como IDLE para Python, Scratch para programar videojuegos (muy interesante sobre todo si se combina con Arduino), la tienda de aplicaciones denominada Pi Store, etc…	Basado en	Debian (Estable)
	Origen	USA
	Arquitectura	Armhf
	Escritorio	LXDE, Pixel
	Categoría	Raspberry Pi
	Estado	Activo
	Popularidad	9.2/10
	Init Software	Systemd
	Administración de Paquetes	DEB
	Archivos de Sistema Registrados	No registra

Fuente: raspberrypi.org y distrowatch.

Tabla 3. Especificaciones y Descripción Ubuntu MATE

Ubuntu MATE	Tipo de OS	Linux
Se trata de un Ubuntu Linux con un entorno de escritorio MATE, ligero y que necesita de pocos recursos en comparación con el pesado Unity. MATE se basa en GNOME2 y es muy conocido en el mundo Linux por su gran aceptación dentro de la comunidad, tanto es así, que existen numerosas distribuciones que lo han elegido como escritorio por defecto.	Basado en	Debian, Ubuntu
	Origen	Reino Unido
	Arquitectura	armhf, i386, powerpc, x86_64
	Escritorio	MATE
	Categoría	Beginners, Desktop, Live Medium, Raspberry Pi
	Estado	Activo
	Popularidad	9.49/10
	Init Software	systemd
	Administración de Paquetes	DEB (apt), snap
	Archivos de Sistema Registrados	Btrfs, ext3, ext4, JFS, ReiserFS, XFS

Fuente: Ubuntu-mate.org y distrowatch.

Tabla 4. Especificaciones y Descripción OpenSUSE

OpenSUSE	Tipo de OS	Linux
La comunidad openSuSE no iba a ser menos y también han añadido soporte para instalar la famosa distribución Linux en cualquier plataforma ARM, además cuentan con una comunidad dedicada especialmente a la placa Raspberry Pi. Sin duda una distro para uso genérico muy atractiva y basada en paquetes RPM.	Basado en	Independent
	Origen	Alemania
	Arquitectura	arm64, armhf, ppc64, ppc64el, s390x, x86_64
	Escritorio	Cinnamon, GNOME, IceWM, KDE, LXDE, Openbox, WMaker
	Categoría	Desktop, Server, Live Medium, Raspberry Pi
	Estado	Activo
	Popularidad	8.76/10
	Init Software	Systemd
	Administración de Paquetes	RPM (zypper)
	Archivos de Sistema Registrados	Btrfs, ext3, ext4, JFS, ReiserFS, XFS

Fuente: en.opensue.org y distrowatch.

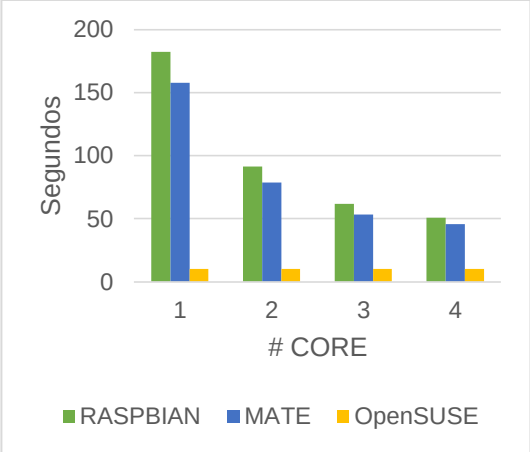
Cada una de las diferentes distribuciones tiene similitudes y diferencias además de amplios tiempos de desarrollo donde se han corregido diferentes problemas y

mejorado su administración del hardware, pero se hace necesario el comparar el rendimiento de estos sistemas para posteriormente contrastarlos con los SO generados por la metodología, lo que se hará en el siguiente capítulo.

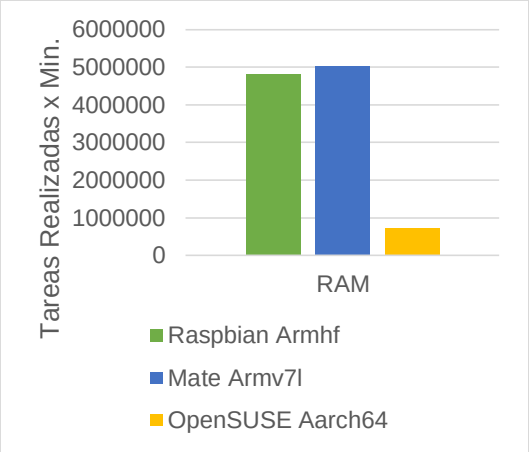
5. RESULTADO ENTRE SO ELEGIDOS

La prueba de CPU de la Grafica 3, se realizó bajo los mismos parámetros en los que se realizó para la comparación de las SBC en la Grafica 1. Los valores se mantienen similares para la prueba de CPU solo para en los SO Raspbian y Mate, debido al aprovechamiento de los 64 bits de parte de OpenSUSE la tarea realizada no significa mucho, por eso los tiempos son tan bajos y no se ve un cambio en la variación de los hilos.

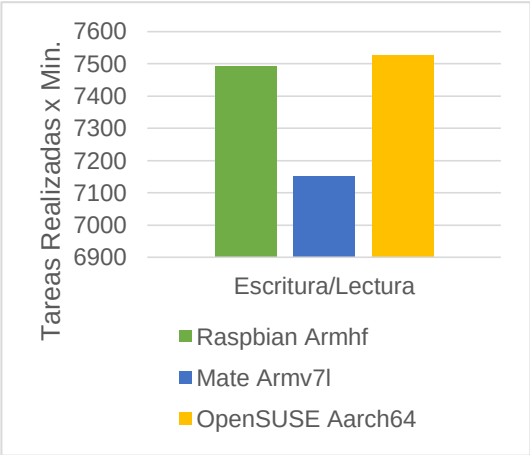
Grafica 3. Uso de CPU entre SO



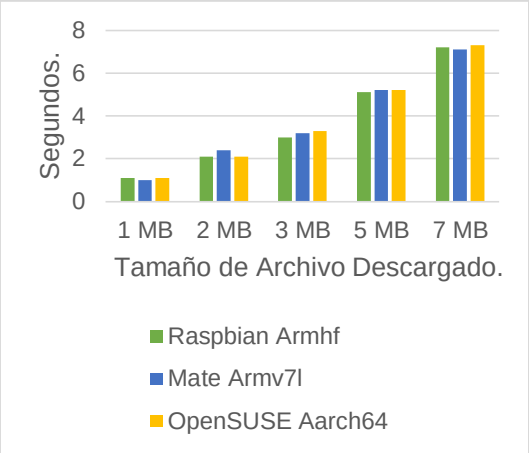
Grafica 4. Uso de RAM entre SO



Grafica 5. Uso de Almacenamiento entre SO



Grafica 6. Uso de RED entre SO



Para la prueba de RAM en la Grafica 4, se dan cuatro tareas (una por hilo) de asignar, reasignar y liberar bloques de 4 MB durante un minuto usando malloc, calloc, realloc y free, 50% del tiempo se usa en la asignación por medio de malloc, calloc, realloc y el otro 50% del tiempo se usa en la liberación por medio de free. Para la prueba de almacenamiento mostrada en la Grafica 5, se inician cuatro tareas (una por hilo) en las que se escribe, lee y elimina 128 MB de archivos temporales de manera secuencial, minimizando el efecto de la cache. Para la prueba de RED expuesta en la Grafica 6, se descargan cinco archivos de diferentes tamaños y se toma el tiempo de descarga, la velocidad de la red es de 1 MB/s.

Como resultado de esta comparación, cada SO administra el hardware a su manera, esto permite concluir que cada distribución sigue teniendo elementos genéricos que no permiten sacar todo el potencial de la tarjeta, dada esta conclusión podemos argumentar que se hace necesario el proponer una metodología para la construcción de distribuciones computacionalmente eficientes para plataformas de alto desempeño sobre hardware abierto embebido. Para

construir una distribución se hace necesario seleccionar una herramienta de construcción, este tema será tratado en el siguiente capítulo.

6. ELECCIÓN DE LA HERRAMIENTA PARA LA CONSTRUCCIÓN DE SO EMBEBIDOS

6.1 BUILDROOT, FACILITANDO EL LINUX EMBEBIDO

Buildroot es una herramienta sencilla, eficiente y fácil de usar para generar sistemas Linux embebidos a través de la compilación cruzada. Puede manejar todo: cadena de herramientas de compilación cruzada, generación de sistema de archivos raíz, compilación de imágenes de kernel y compilación de cargador de arranque. Su interfaz de configuración se parece mucho a menuconfig, gconfig y xconfig, la construcción de un sistema básico con Buildroot es fácil. Buildroot es para todos. Tiene una estructura simple que hace que sea fácil de entender y extender. Se basa sólo en el conocido lenguaje Makefile. Buildroot es un proyecto de código abierto.

6.2 YOCTO PROJECT, NO ES UNA DISTRIBUCIÓN LINUX EMBEBIDA, CREA UNA PERSONALIZADA

El proyecto Yocto es un proyecto de colaboración de código abierto que proporciona plantillas, herramientas y métodos para ayudarle a crear sistemas basados en Linux personalizados para productos embebidos, independientemente de la arquitectura del hardware. Es un entorno completo de desarrollo de Linux integrado con herramientas, metadatos y documentación, todo lo que se necesite. El proyecto Yocto promueve la adopción comunitaria de esta tecnología de código abierto, permitiendo a sus usuarios centrarse en sus características de producto y desarrollo específicos. El soporte específico de la plataforma toma la forma de capas de paquete de soporte de la tarjeta (BSP, Board Service Packages) para las que se ha desarrollado un formato estándar.

6.3 BUILDROOT VS YOCTO PROJECT

Buildroot Pros.

- Se centra en la simplicidad.
- Documentación, wiki, etc. Está ahí y es bastante confiable.
- La herramienta básica de Buildroot se mantiene tan pequeña y tan simple como sea posible, lo que facilita su uso y comprensión.
- Utiliza ncurses y, literalmente, tiene el mismo tipo de interfaz que la que encuentras al hacer un kernel de Linux.
- Buildroot reutiliza herramientas existentes como kconfig siempre que sea posible.
- Tiene una buena escalabilidad. Se puede construir tanto un pequeño proyecto embebido como uno grande.
- La imagen de SO que Buildroot crea también es mínima por defecto, haciendo que las compilaciones sean rápidas y entreguen un sistema "agnóstico de propósito", no adaptado a ningún caso de uso en particular.
- También puede crear un rootfs con un gestor de paquetes como opkg y luego chroot y seguir construyendo la distribución. Aunque esto requiere mucho trabajo.
- Gran selección de paquetes.
- Buildroot pone toda la información de configuración en un archivo, que se puede editar usando cualquiera de las interfaces de la herramienta kconfig del kernel. Ese archivo especifica los paquetes de arquitectura, kernel, bootloader y espacio de usuario a incluir.
- Cada cambio propuesto para el núcleo se analiza en términos de su "relación de utilidad a la complejidad". La lógica básica se implementa enteramente en Makefiles que totalizan menos de 1000 líneas de código. Esa cantidad de código puede ser leída y entendida por un individuo.

Yocto Project Pros.

- Se le ha dedicado mucho tiempo, la experiencia no se improvisa.
- Yocto intenta ser versátil y soporta una amplia gama de sistemas embebidos.
- La definición de compilaciones en las recetas, especifican qué software usar para construir y cómo construirlo mediante el apoyo de las capas, que son colecciones de recetas escritas y mantenidas por la comunidad de desarrollo.
- Se puede construir a través de línea de comandos o usar la GUI WEB de Toaster.
- Al confiar en capas definidas, la imagen de sistema predeterminada definida puede permanecer pequeña.
- Una amplia paquetería.
- Las capas permiten a la comunidad soportar nuevas placas o arquitecturas, definir nuevas pilas de aplicaciones o soportar nuevos casos de uso.
- Proporciona una forma mucho más fácil de construir un sistema de archivos raíz que buildroot.
- La producción de Yocto es "una distribución", un suministro de paquetes, aunque proporcionar un sistema de gestión de paquetes a gran escala para el dispositivo opcional y generar imágenes de disco completas es ciertamente soportado.
- Los paquetes individuales pueden reconstruirse según sea necesario, los paquetes individuales pueden ser actualizados o eliminados y la actualización de un paquete es más rápida que una reconstrucción completa.

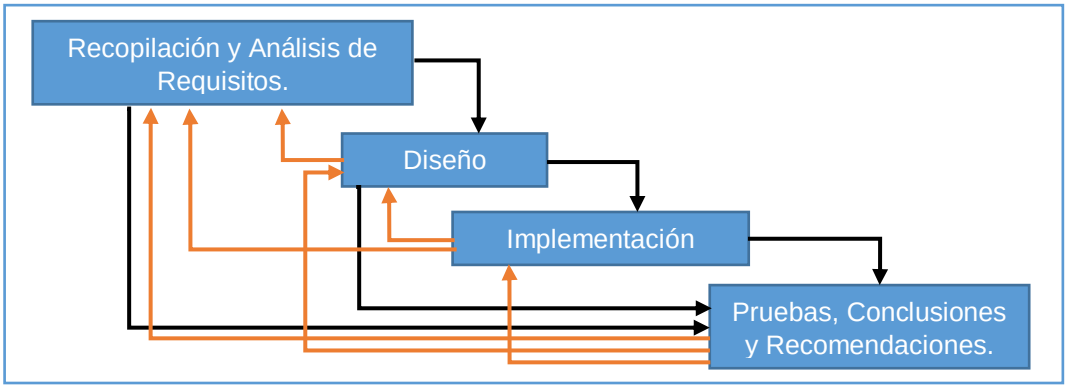
Buildroot Contras.	Yocto Project Contras.
• Se debe estar preparado para muchos problemas de compilación y depuración. • Puede instalarse en una amplia variedad de sistemas, pero en realidad sólo funciona bien con una distribución pulida o que no esté en fase de prueba. • La salida de Buildroot es una imagen del sistema de archivos raíz, nada más; esto ha llevado a algunos a llamarla un "generador de firmware". Siempre que necesite actualizar el sistema, debe regenerar la imagen completa, o de alguna manera construir su propio mecanismo de actualización (como un sistema bin binario). • Las máquinas similares no pueden compartir ninguna información de la configuración: uno debe construir cada objetivo por separado.	• Cuando los problemas surgen, se necesita tiempo y esfuerzo para averiguar qué es lo que está mal. • Capas, recetas y cualquier otra terminología puede ser confuso al principio. • Toma mucho más tiempo y esfuerzo para configurar correctamente. • Uno puede tener que reunir un montón de capas antes de comenzar una compilación, pero también facilita la creación de variaciones con poco trabajo adicional. • Yocto tiene una curva de aprendizaje más desafiante, son casi 60000 líneas de código escritas en Python, scripts de shell y un lenguaje específico de BitBake.

Tanto los alcances de buildroot como de yocto son amplios y muy variados, la curvatura de aprendizaje es más corta en buildroot aunque el solucionar algún fallo en ambos puede tomar mucho tiempo. La configuración es más lenta en yocto aunque esto puede ayudar en la creación de variaciones de la distribución más detallada, builroot puede instalarse en muchos sistemas, sumado a esto, buildroot se debe ejecutar con todos los permisos de un super usuario, esto evitara en gran manera problemas de permisos y excepciones de reglas que son necesarias para poder construir el SO. No se considera que buildroot esté por encima de yocto o viceversa. Los pros y contras han surgido al investigar cada una de las herramientas, lo que se puede sacar en claro es que tiene diferentes tipos de diseño, pero buildroot funciona mejor para los requerimientos del proyecto. Ya elegida la herramienta, se debe implementar la metodología para la construcción de la distribución, esta cuestión será tratada en el siguiente capítulo.

7. METODOLOGÍA DE CONSTRUCCIÓN DE DISTRIBUCIONES

Una distribución de Linux es un SO hecho a partir de una recopilación de software, los usuarios de Linux habitualmente obtienen su sistema operativo descargando una de las distribuciones de Linux, que están disponibles para una amplia variedad de sistemas que van desde dispositivos embebidos, computadoras personales, hasta potentes supercomputadores. Una distribución típica de Linux comprende un kernel de Linux, herramientas y bibliotecas de GNU, sistema de ventanas, administrador de ventanas y un entorno de escritorio[13].

Figura 2. Metodología de Construcción de Distribuciones.



Se recomienda el uso de la metodología en SBC que estén enfocada a una tarea específica y soporten o no una distro completa, tenga en cuenta el tiempo que puede dedicar a la construcción de la distro ya que una distro popular se podría pulir para cumplir con los objetivos del proyecto, pero si desea construir una distro dedicada a una tarea y SBC específica, generar la distro desde cero es la mejor opción. Para un desarrollo pragmático de la metodología se adopta un modelo de cascada cíclica mostrado en la Figura 2.[14]

[13] BERGER, A. S. Embedded Systems Design : An Introduction To Processes, Tools, & Techniques. (1 Ed.) New York: CRC Press. 2001
[14] HEATH, S. Embedded Systems Design. (2 Ed.) Oxford: Newnes Elsevier Science. 2003.

Como se ve en la Figura 2, existen cuatro escenarios para el desarrollo de la metodología, las líneas negras muestran el avance de un escenario a otro, las líneas naranjas son las modificaciones que se hacen a los escenarios anteriores para mejorar el escenario actual. Para diseñar e implementar un sistema embebido Linux se deben realizar muchas tareas, algunas de ellas se pueden omitir para construir una versión básica del sistema, que pueda tomarse como base y a medida que avance el proyecto se pueda ir montando los paquetes necesarios para cubrir los requerimientos del proyecto[15], otras se pueden realizar en paralelo reduciendo el tiempo de desarrollo. Independientemente de las herramientas, se deben tener conceptos básicos sobre las tareas relacionadas con la construcción del sistema.

7.1 RECOPILACIÓN Y ANÁLISIS DE REQUISITOS

Determinar los componentes del sistema es igual que hacer una lista de suministros (mercado), esto evita que el sistema tenga muchas opciones, pero no cumpla su propósito principal. No obstante, no significa que no se pueda cambiar la lista si es oportuno. Se debe analizar el hardware al cual se está orientando el sistema, esto aporta los datos necesarios e ideas de que requisitos se pueden cumplir, proporcionando una lista de atributos para el diseño del sistema. Este paso es el único que no se puede desarrollar en paralelo a otras tareas y debe completarse antes de cualquier otro. Se debe evitar usar las últimas versiones de software para el diseño, ya que muchos pueden estar en etapa de prueba, requerir de actualizaciones de otro software debido a dependencias entre paquetes o no ser compatibles con el kernel seleccionado. En su lugar, use las versiones estables y realice un seguimiento de sus progresos para que puedan ser incluidos en próximos proyectos que consigan sacar provecho a estos avances. Si tiene razones importantes para usar la última versión, analice las consecuencias para el

[15] YAGHMOUR K., MASTERS J., BEN-YOSSEF G., y GERUM P. Building Embedded Linux Systems. (2 Ed.) Sebastopol: O'Reilly Media, Inc. 2008.

resto del sistema antes de incluirlo o inclúyalo en un sistema de prueba antes de incluirlo en el principal.

Se deben tener en cuenta conceptos como: los diferentes tipos de host usados para la compilación cruzada, las configuraciones de desarrollo de compilación cruzada, la arquitectura del hardware embebido, el inicio del sistema, las configuraciones de arranque, los tipos de almacenamiento, la arquitectura del procesador, los buses e interfaces, entradas, salidas (teclado, impresora, sonido, etc.), redes (ethernet, Wi-Fi, etc.), monitoreo del sistema, etc.

7.2 DISEÑO

Una vez determinadas las características pertinentes para el diseño, se debe elegir la versión del kernel y su configuración. A diferencia del software, es bueno usar la última versión estable del kernel debido a que si usa versiones antiguas o anteriores puede que intente corregir errores que ya estén solucionados o no conseguir soporte de parte de la comunidad. Independiente de si decide actualizar el kernel, se sugiere mantener la configuración del kernel constante durante el proyecto. Esto evita problemas en el desarrollo. No obstante, se debe estudiar las opciones de configuración para que sean enfocadas al cumplimiento de los requisitos. Es importante que los involucrados en el proyecto sean conscientes de las opciones de configuración y estén de acuerdo con las opciones seleccionadas.[16,17]

El sistema de archivos raíz en un sistema embebido es similar al que se encuentra en una estación de trabajo o servidor, excepto que contiene un conjunto mínimo de aplicaciones, bibliotecas y archivos necesarios para ejecutar el sistema. No se

[16] BLACKMORE C., RAY O., y EDER K. Automatically Tuning The Gcc Compiler To Optimize The Performance Of Applications Running On Embedded Systems. Bristol: University of Bristol. 2017.
[17] CESATI, M. y BOVET, D. Understanding The Linux Kernel. (3 Ed.) Sebastopol: O'reilly Media, Inc. 2005.

debería tener que eliminar ningún componente elegido para obtener un sistema de archivos de tamaño adecuado, si es necesario es probable que no haya determinado adecuadamente los componentes del sistema. Tenga en cuenta que debe tener una estimación lo más precisa posible del tamaño de cada componente que selecciono durante la recopilación y análisis de requisitos.[18]

El arranque depende en gran medida de la arquitectura, esto involucra diferentes gestores de arranque. En una arquitectura única hay variaciones en la depuración y monitoreo proporcionados por los gestores de arranque. Para configurar el arranque se debe investigar a profundidad en el tema de los bootloader (GRUB, U-Boot, etc.).

7.3 IMPLEMENTACIÓN

Construir el kernel concibe más que solo una imagen, aunque no todos los componentes generados son necesarios para el desarrollo del proyecto, otros componentes del proyecto dependen en gran medida de los componentes del kernel. Por tanto, es preferible tener los componentes del kernel configurados, construidos y actualizados a lo largo del proyecto. Hay que tener ciertas consideraciones con el kernel, tales como: su selección, configuración (opciones y métodos), construcción del kernel y sus módulos, compilación, por ultimo su instalación.[19]

Paralelo a la construcción del kernel, puede comenzar a construir el sistema de archivos raíz (root filesystem), si no puede determinar la lista completa de componentes necesarios para su sistema de archivos puede construir uno de manera iterativa, agregando herramientas y bibliotecas necesarias a medida que avanza, no considere el sistema de archivos generado como la versión final. En su

[18] MASSA A., y BARR M. Programming Embedded Systems. (2 Ed.) Sebastopol: O'reilly Media, Inc. 2009.
[19] LABROSSE, J. J. Embedded Systems Building Blocks : Complete And Ready-To-Use Modules In C. (2 Ed.) CMP. 1999.

lugar, use el método iterativo para explorar la construcción del sistema de archivos, esto aumentara el tiempo de finalización. En el contenido de un sistema de archivos podemos encontrar: las librerías usadas (glibc, uClibc), modulo o imagen del kernel, dispositivo de archivos, aplicaciones principales del sistema (BusyBox, embutils) e inicialización del sistema. Para la configuración del sistema de archivos se debe abarcar conceptos como los tipos de sistema de archivos (Ext4, Cramfs, Tmpfs, etc.), rootfs e initramfs, el tipo y diseño de un sistema de archivos, su manejo de actualizaciones. El empaquetar y arrancar un sistema es bastante similar entre diferentes arquitecturas, pero varía dependiendo del dispositivo de almacenamiento desde el que se inicia el sistema y que gestor de arranque utiliza. Arrancar un sistema desde un flash nativo es diferente a arrancar un sistema desde un HDD SATA o un dispositivo CompactFlash e incluso diferente a arrancar desde un servidor de red.

7.3.1 Caso de uso de la metodología. Para este ejemplo, se usa un host Linux para el desarrollo de la distribución. Después de descargar el archivo comprimido de Buildroot de la página principal, descomprimirlo y poner la carpeta en una ubicación deseada, acceda a la carpeta de Buildroot desde la terminal, simultáneamente abra otra terminal e ingrese a la carpeta de Buildroot para acceder a la carpeta de configs, las terminales deben estar en modo super usuario, puede usar la última versión de Buildroot o la versión estable que recomiende la página web de la herramienta.

En la carpeta configs se almacenan los defconfigs que sirven como base para preconfigurar Buildroot. Raspberrypi2_defconfig puede servir como base en caso que no se encuentre una para Raspberry Pi 3, puede copiar y renombrar este archivo o simplemente dejarlo como esta, aunque estas modificaciones pueden generar problemas durante la compilación, use la versión de defconfig que más se parezca a la tarjeta que va a usar, de no encontrarse en la carpeta configs, deberá crear una configuración desde cero.

Se debe copiar el nombre del defconfig de la SBC que se va a usar en la terminal, luego usar el comando make:

make SBC_defconfig, para el ejemplo: make raspberrypi3_64_defconfig

Una vez ejecutado el comando se crea una configuración por defecto en buildroot, por ultimo escriba el comando:

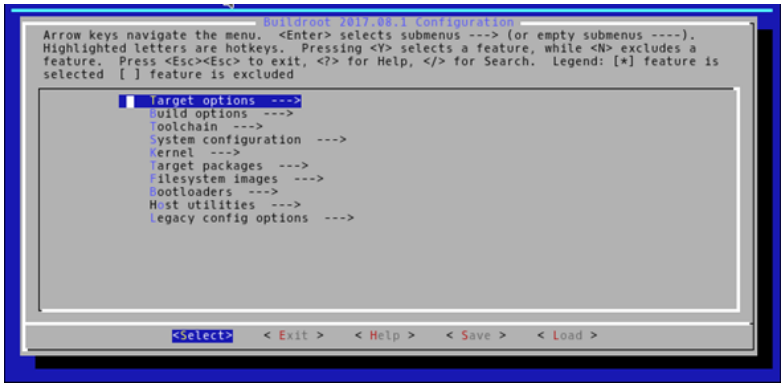
make

Esta configuración crea una distro básica, que reconoce el teclado, Ethernet, HDMI y tiene unos cuantos comandos básicos para navegar por la tarjeta y su sistema. Para pasar la distro a una SD use el Anexo D. Una vez probada la compilación básica para la tarjeta es recomendable el ampliar algunos recursos básicos para el sistema. Para mejorar las prestaciones de la distro construida, ejecute el comando:

make menuconfig

Una vez lo ejecute aparecerá la interfaz mostrada en la Figura 3.

Figura 3. Interfaz de Buildroot



Fuente: Interfaz de Buildroot.

Cada una de las opciones mostradas en la Figura 3 tiene la siguiente función:

- Target options: nos permite configurar la arquitectura, variantes del procesador de la SBC, su formato binario, etc. Por eso es necesario conocer de manera detallada la SBC para poder configurar estas opciones.

- Build options: nos permite configurar los comandos más comunes (wget, cp, etc.), la carpeta donde se descargan los archivos para la compilación, el uso de ccache, el nivel de optimización de gcc, etc.

- Toolchain: si se deja de manera predeterminada, Buildroot construirá la toolchain y la usará para la compilación cruzada de los paquetes. Si se usa una toolchain externa se elimina el tiempo de compilación de la toolchain pero se debe verificar como afecta al SO generado. También se puede habilitar soporte de librerías para c, c++, fortran, openMP entre otros.

- System configuration: se puede configurar el nombre del sistema, el sistema de inicio, el administrador /dev, el shell (se recomienda bash), entre otras.

- Kernel: Buildroot automatiza el proceso de compilación del kernel, construye el kernel usando un defconfig o un archivo de configuración especificado.

- Target Packages: Buildroot cuenta con una paquetería muy amplia, se pueden configurar paquetes de audio, video, compresión, benchmark, herramientas de desarrollo, utilidades de filesystem, intérpretes de lenguaje, aplicaciones de red (entre esas MPI), seguridad, editores de texto y muchas otras.

- Filesystem Image: Buildroot puede generar sistemas de archivos en múltiples formatos, también se puede configurar dichos filesystem.

- Bootloaders: Buildroot incluye soporte para los bootloaders más populares: U-Boot, barebox, grub, syslinux, etc. Puede también no seleccionar un bootloader para la tarjeta y aun así, builtroot configurara uno por defecto.

Si se desea pulir aún más la configuración puede usar el comando:

make linux-menuconfig

Para esta configuración se necesita un conocimiento muy amplio en la administración de sistemas operativos, hardware, librerías, etc. Como se mencionó con anterioridad, se debe seguir los consejos presentados por la metodología para la construcción de distribuciones, esta es la opción a usarse en caso de que no se encuentre un defconfig de la SBC. El paso a seguir es evaluar las distribuciones generadas, lo que se realizara en la siguiente sección.

7.4 EVALUACIÓN DE LAS DISTRIBUCIONES GENERADAS POR LA METODOLOGÍA

Usando la metodología se crean siete versiones de SO linux para la RPi3, cuatro de estas versiones están enfocadas a la arquitectura aarch64, las tres restantes están configuradas para armv7l. Para hacer contra posición se evalúan tres versiones de SO Linux más populares, orientadas a la RPi3. Las configuraciones de cada SO generado por la metodología se trataran en el siguiente titular.

7.4.1 Configuración de los SO generados. Las siguientes son las diferentes versiones de SO generadas por la metodología usando las configuraciones básicas y leves optimizaciones sin una modificación intrusiva en el kernel.

Arquitectura a 64 bits:

- OS aarch64 v1:
  - Default.
- OS aarch64 v2:
  - fp-armv8 -mcpu=cortex-a53 -march=armv8-a+crc -mtune=cortex-a53.
- OS aarch64 v3:
  - fp-armv8 -mcpu=cortex-a53 -march=armv8-a+crc -mtune=cortex-a53. Uso de toolchain de linaro.
- OS aarch64 v4:
  - vfpv4 -march=armv8-a+crc -mtune=cortex-a53 -O2 -pipe.

Arquitectura a 32 bits:

- OS armv7l v1:
  - Default.
- OS armv7l v2:
  - EABIhf -march=armv7-a -mfpu=neon-vfpv4 -mfloat-abi=hard -O2 -pipe.
- OS armv7l v3:
  - VFPv3-D16 -mfloat-abi=hard -fomit-frame-pointer -O2 -pipe.

Estas opciones “-m” se definen para la adaptación de ARM:

- -mcpu=nombre[+extenciones]. Esto especifica el nombre del procesador ARM objetivo. GCC usa este nombre para derivar el nombre de la arquitectura ARM objetivo y el tipo de procesador ARM para el cual ajustar el rendimiento.

- -march=nombre[+extenciones]. Esto especifica el nombre de la arquitectura ARM objetivo. GCC usa este nombre para determinar qué tipo de instrucciones puede emitir al generar código de ensamblaje.

- -mtune=nombre. Esta opción especifica el nombre del procesador ARM objetivo para el cual GCC debe ajustar el rendimiento del código. Para algunas implementaciones de ARM, se puede obtener un mejor rendimiento al usar esta opción.
- -mfpu=nombre. Esto especifica qué hardware de coma flotante (o emulación de hardware) está disponible. La configuración 'auto' es la predeterminada y es especial. Hace que el compilador seleccione las instrucciones de punto flotante y SIMD Avanzado basadas en la configuración de -mcpu y -march.

- -mfloat-abi=nombre. Especifica qué ABI de coma flotante usar. Los valores permitidos son: 'soft', 'softfp' y 'hard'. Al especificar 'soft', GCC genera resultados que contienen llamadas a bibliotecas para operaciones de coma flotante. 'Softfp'

permite la generación de código usando instrucciones de coma flotante de hardware, pero aún usa las convenciones de llamada de flotación suave. 'Hard' permite la generación de instrucciones de punto flotante y utiliza convenciones de llamadas específicas de FPU.

Para evaluar el comportamiento de cada sistema se usa una herramienta basada en C/C++. Las pruebas de estrés se enfocan en como administra el SO la CPU, RAM, Almacenamiento y red, y que tantas tareas puede realizar en un tiempo determinado. Este tema será tratado en la siguiente sección.

7.4.2 Herramienta de evaluación. Las pruebas se realizan con stress-ng. Fue diseñado para ejercitar varios subsistemas físicos de una computadora, así como las diversas interfaces del kernel del sistema operativo. Stress-ng se diseñó para hacer que una máquina trabaje al maximo y provocar problemas de hardware, como sobrecargas térmicas, errores en el sistema operativo que solo ocurren cuando un sistema está siendo estresado. Stress-ng consta de:

- Más de 195 pruebas de estrés.
- 70 pruebas de esfuerzo específicas de CPU que ejercen flujo flotante, entero, manipulación de bits, etc.
- Más de 20 pruebas de estrés con memoria virtual.

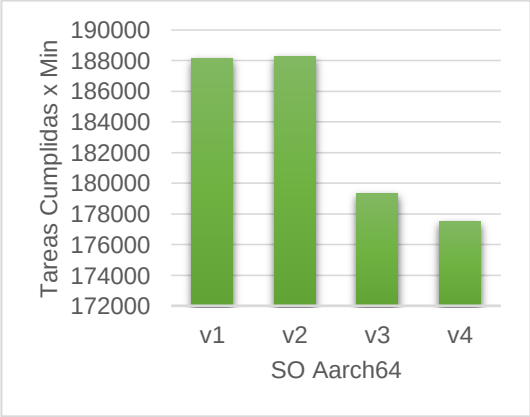
Los siguientes temas trataran los resultados obtenidos en diferentes pruebas para CPU, RAM, almacenamiento y red, en cada SO.

7.4.3 Comparativa entre versiones generadas por la metodología a 64 bits para CPU. Las cuatro pruebas para la CPU consisten en lo siguiente:

- **Cfloat:** se realizan 1000 iteraciones de una combinación de operaciones complejas de coma flotante, los resultados se muestran en la Grafica 7 (código en el Anexo A).

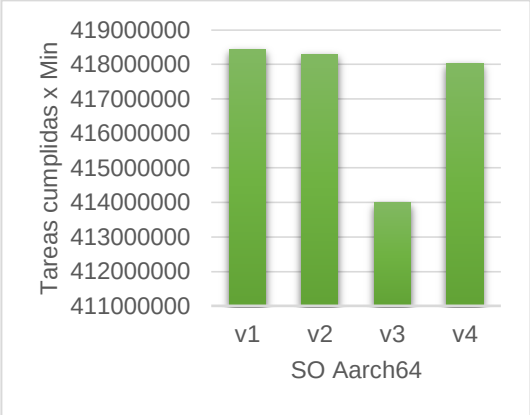
- **Clongdouble:** se realizan 1000 iteraciones de una combinación de operaciones complejas de coma flotante doble larga, los resultados se exponen en la Grafica 8 (código en el Anexo A).
- **Phi:** se calcula la Proporción Áurea φ usando series, los resultados estan en la Grafica 9 (código en el Anexo B).
- **Prime:** se encuentran todos los números primos en el rango de 0 a 1000000 usando una búsqueda ligeramente optimizada de un proceso de división de fuerza bruta, los resultados se visualizan en la Grafica 10 (código en el Anexo C).

Grafica 7. Resultado de la Prueba de Cfloat en SO Generados por la Metodología a 64 Bits.



Grafica 8. Resultado de la Prueba de Clongdouble en SO Generados por la Metodología a 64 Bits.

Grafica 9. Resultado de la Prueba de Phi en SO Generados por la Metodología a 64 Bits.

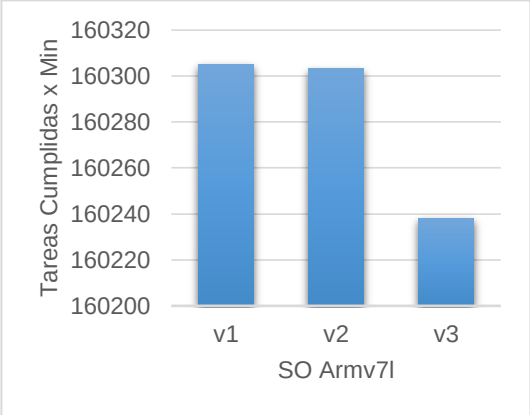


Grafica 10. Resultado de la Prueba de Prime en SO Generados por la Metodología a 64 Bits.

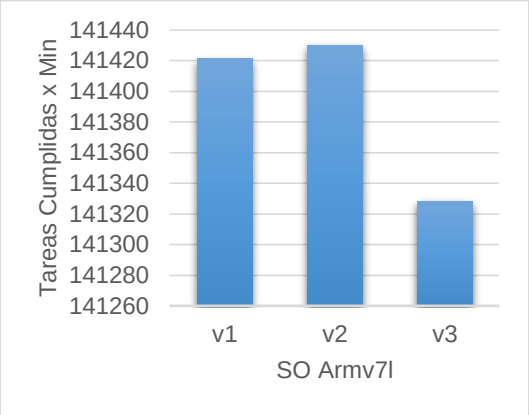
Como resultado final de las cuatro pruebas, la versión a 64 bits que mejor se desempeña es SO Aarch64 v2, en la siguiente sección evaluaremos los SO generados por la metodología a 32 bits.

7.4.4 Comparativa entre versiones generadas por la metodología a 32 bits para CPU. Al igual que en la sección anterior, se realizan las cuatro pruebas a la CPU para medir la versión de mejor rendimiento, el resultado de prueba de CPU para Cfloat se ve en la Grafica 11, en la Grafica 12, se trata el resultado de la prueba de Clongdouble, la Grafica 13 expone el resultados de la prueba de phi, por último en la Grafica 14, vemos los resultados de la prueba de primos.

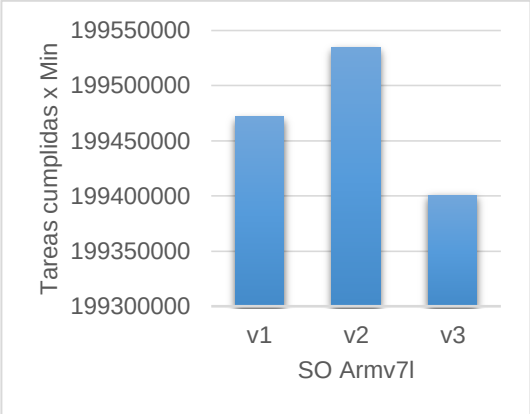
Grafica 11. Resultado de la Prueba de Cfloat en SO Generados por la Metodología a 32 Bits.



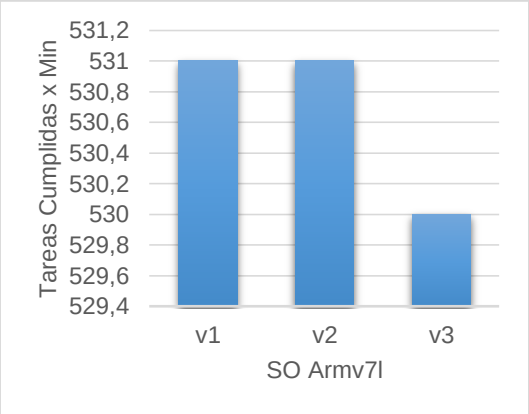
Grafica 12. Resultado de la Prueba de Clongdouble en SO Generados por la Metodología a 32 Bits.



Grafica 13. Resultado de la Prueba de Phi en SO Generados por la Metodología a 32 Bits.



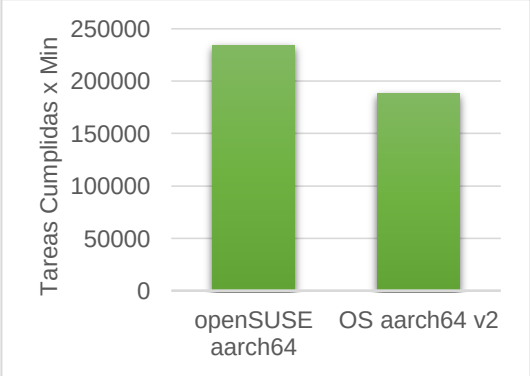
Grafica 14. Resultado de la Prueba de Prime en SO Generados por la Metodología a 32 Bits.



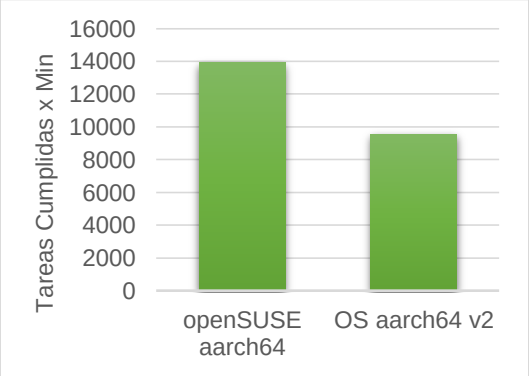
Como resultado de las cuatro pruebas, la versión a 32 bits con mejor rendimiento es SO Armv7l v2. Como ya se evaluaron las diferentes versiones generadas por la metodología y se eligieron las mejores versiones, se procederá a comparar las versiones elegidas con las versiones populares.

7.4.5 Comparativa entre versión existente y generada por la metodología a 64 bits para CPU. En las anteriores pruebas a los SO generados por la metodología se eligió la versión SO Aarch64 v2 como la de mejor rendimiento para las pruebas de CPU a 64 bits y será contrastada con OpenSUSE Aarch64. En la Grafica 15 se aprecia el resultado de la prueba de Cfloat, la Grafica 16 expone el resultado de la prueba de Clongdouble, en la Grafica 17 vemos el resultado de la prueba de phi, para finalizar, la Grafica 18 expone el resultado de la prueba de primos.

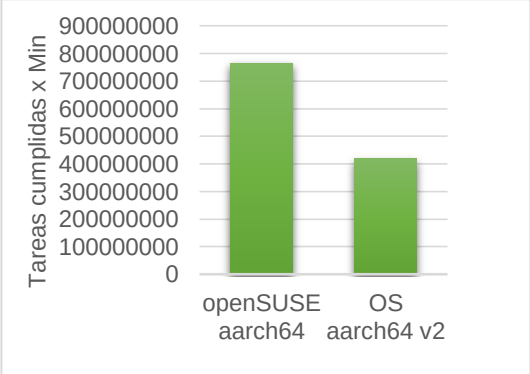
Grafica 15. Comparativa entre OpenSUSE y el SO Aarch64 v2 en la prueba de Cfloat.



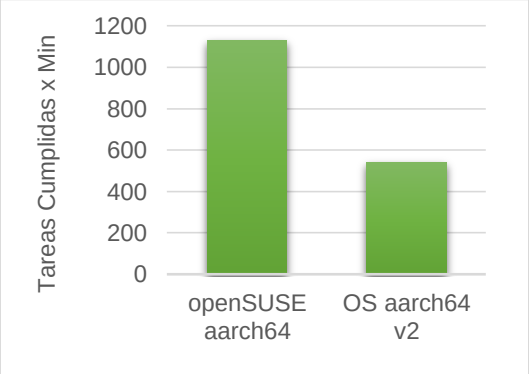
Grafica 16. Comparativa entre OpenSUSE y el SO Aarch64 v2 en la prueba de Clongdouble.



Grafica 17. Comparativa entre OpenSUSE y el SO Aarch64 v2 en la prueba de Phi.



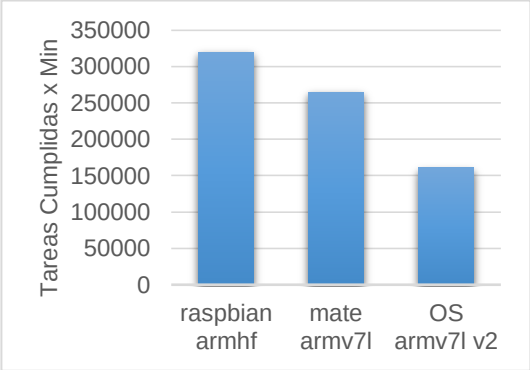
Grafica 18. Comparativa entre OpenSUSE y el SO Aarch64 v2 en la prueba de Prime.



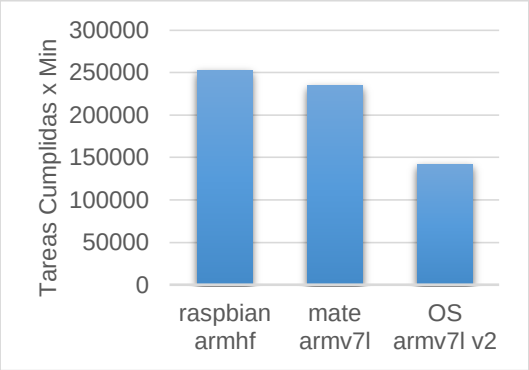
En las cuatro pruebas desarrolladas para comparar los SO a 64 bits, OpenSUSE obtiene el mejor desempeño, pero SO Aarch64 v2 tiene un nivel de optimización liviano, esto se debe a que no se profundizo en la configuración del kernel para la administración de la CPU, aun así, se obtienen valores aceptables con configuraciones mínimas y que pueden ser mejoradas en próximos proyectos, en la siguiente sección se comparan las versiones a 32 bits.

7.4.6 Comparativa entre versión generada por la metodología y existentes a 32 bits para CPU. Como se trató anteriormente, se compara la versión SO Armv7l v2 con las dos versiones más populares (al momento de iniciar el proyecto) de SO a 32 bits que son Raspbian y Ubuntu Mate. En la Grafica 19 vemos la comparación de los resultados para la prueba de Cfloat, la Grafica 20 compara el resultado en las pruebas de Clongdouble, el resultado de la prueba de phi se muestra en la Grafica 21, para finalizar, la Grafica 22 compara los resultados de la prueba de primos entre SO de 32 bits.

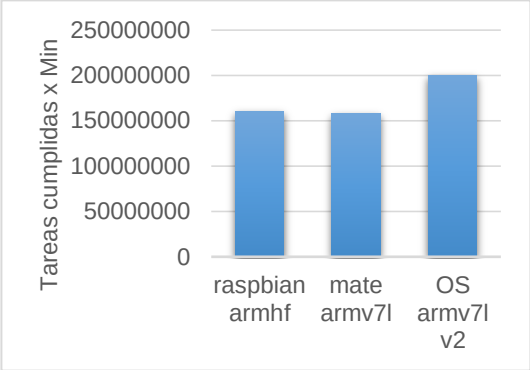
Grafica 19. Comparativa entre Raspbian, Ubuntu Mate y SO Armv7l v2 en la Prueba de Cfloat.



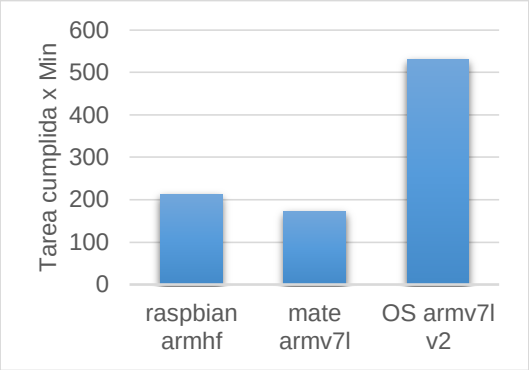
Grafica 20. Comparativa entre Raspbian, Ubuntu Mate y SO Armv7l v2 en la Prueba de Clongdouble.



Grafica 21. Comparativa entre Raspbian, Ubuntu Mate y SO Armv7l v2 en la Prueba de Phi.



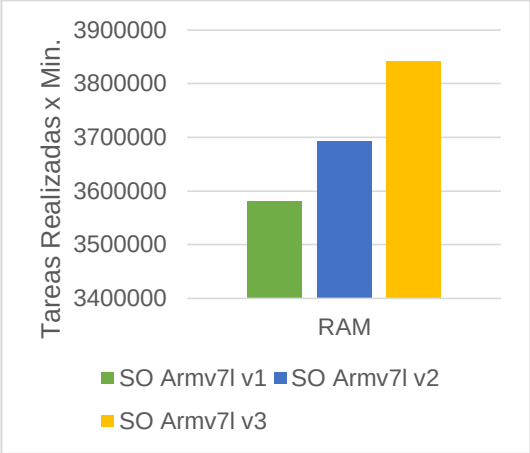
Grafica 22. Comparativa entre Raspbian, Ubuntu Mate y SO Armv7l v2 en la Prueba de Prime.



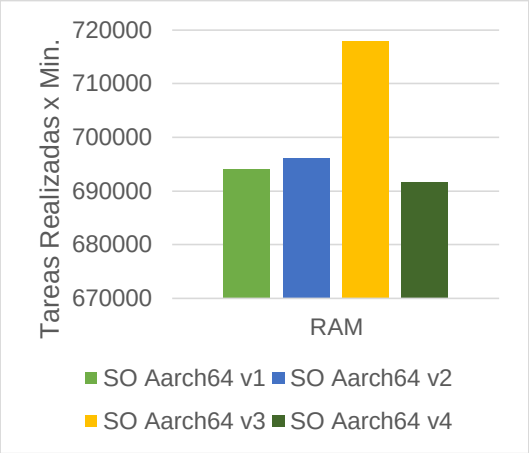
Los resultados de las pruebas arrojan un margen un poco más apretado, en las dos primeras pruebas Raspbian domina al SO Armv7l v2 debido a que Raspbian se enfoca en el manejo de coma flotante, pero en las últimas dos pruebas, las configuraciones del kernel para Raspbian y Mate están desactualizadas debido a que usan una versión de kernel anterior a la usada por SO Armv7l v2. La siguiente prueba se enfoca en el manejo de la RAM por parte de cada SO.

7.4.7 Comparativa para la administración de la RAM entre SO generados por la metodología y distros populares. Para la prueba de RAM (Anexo E), se dan cuatro tareas (una por hilo) de asignar, reasignar y liberar bloques de 4 MB durante un minuto, 50% del tiempo se usa en la asignación o reasignación y el otro 50% del tiempo se usa en la liberación. En la Grafica 23 y 24 se muestra los resultados de cada SO generado por la metodología administrando la RAM, en la Grafica 25 y 26 se compara la mejor versión de Armv7l y Aarch64 con su parte popular.

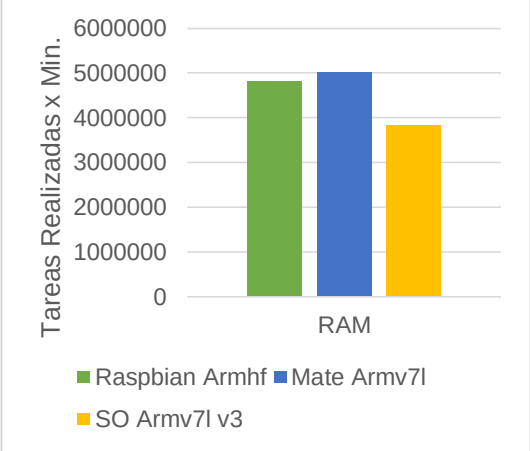
Grafica 23. Comparativa entre SO Armv7l Generados por la Metodología Administrando la RAM.



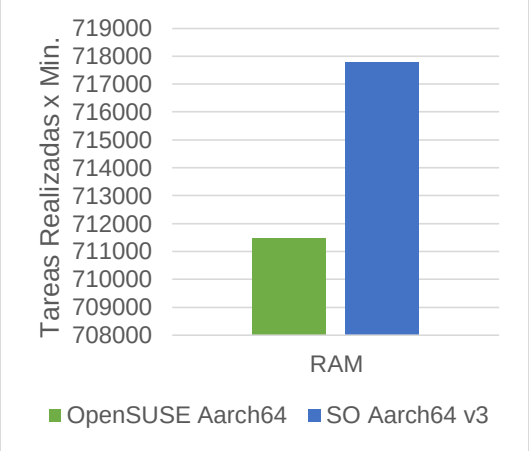
Grafica 24. Comparativa entre SO Aarch64 Generados por la Metodología Administrando la RAM.



Grafica 25. Comparativa entre Raspbian, Ubuntu Mate y SO Armv7l v3 Administrando la RAM.



Grafica 26. Comparativa entre OpenSUSE y el SO Aarch64 v3 Administrando la RAM.

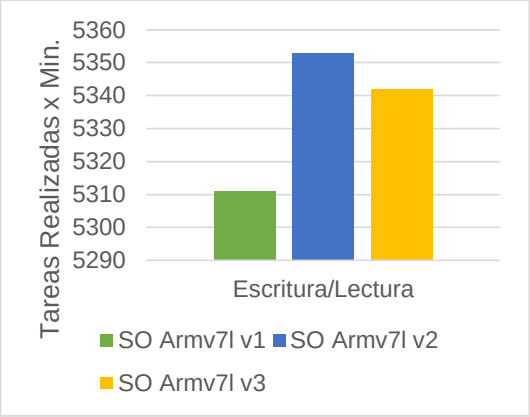


Los resultados muestran un desempeño superior para SO Aarch64 v3 en los resultados de la administración de la RAM para los SO de 64 bits, esto se da gracias a una versión más reciente del kernel. En los SO de 32 bit hay una

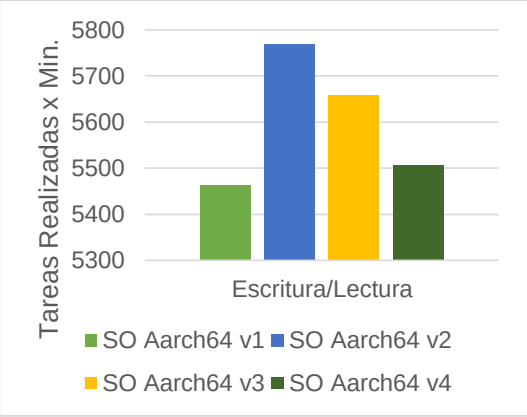
deferencia con los SO populares que puede ser saldada optimizando la administración de la RAM por parte del kernel, esto puede ser cubierto en próximos proyectos. El siguiente paso es probar la capacidad de administración de almacenamiento en cada SO.

7.4.8 Comparativa para la administración de almacenamiento entre SO generados por la metodología y distros populares. Para la prueba de almacenamiento (Anexo F), se inician cuatro tareas (una por hilo) en las que se escribe, lee y elimina 128 MB de archivos temporales de manera secuencial, minimizando el efecto de la cache. En la Grafica 27 y 28 se muestra los resultados de cada SO generado administrando el almacenamiento, en la Grafica 29 y 30 se compara la mejor versión de Armv7l y Aarch64 con su parte popular.

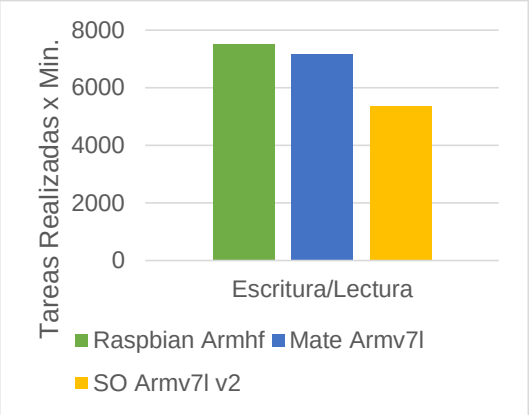
Grafica 27. Comparativa entre SO Armv7l Generados por la Metodología Administrando el Almacenamiento.



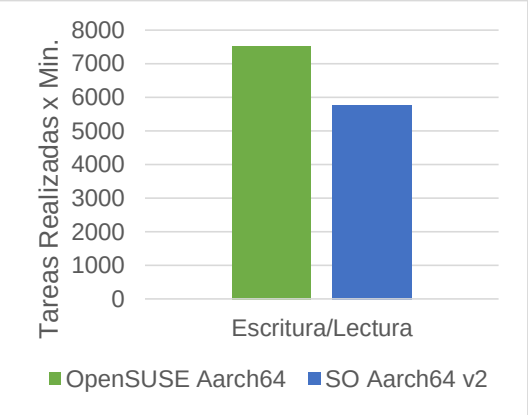
Grafica 28. Comparativa entre SO Aarch64 Generados por la Metodología Administrando el Almacenamiento.



Grafica 29. Comparativa entre Raspbian, Ubuntu Mate y el SO Armv7l v2 Administrando el Almacenamiento.



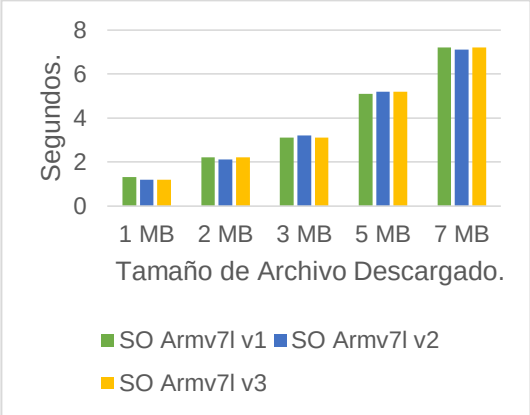
Grafica 30. Comparativa entre OpenSUSE y el SO Aarch64 v2 Administrando el Almacenamiento.



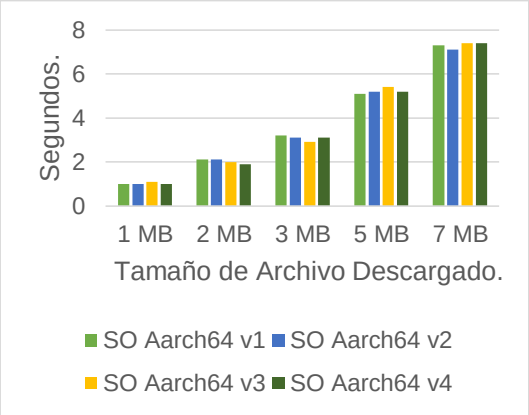
Como conclusión de la prueba de almacenamiento, las versiones populares tienen un comportamiento similar y superior al de las versiones generadas, esta diferencia puede ser cubierta mejorando las configuraciones de almacenamiento, esto puede realizarse en proyectos posteriores. Como último paso se hace una prueba de la administración de red.

7.4.9 Comparativa para la administración de red entre SO generados por la metodología y distros populares. Para la prueba de RED, se descargan cinco archivos de diferentes tamaños y se toma el tiempo de descarga, la velocidad de la red es de 1 MB/s. En la Grafica 31 y 32 se muestran los resultados de cada SO generado para la prueba de red, en la Grafica 33 y 34 se compara la mejor versión de Armv7l y Aarch64 con su parte popular.

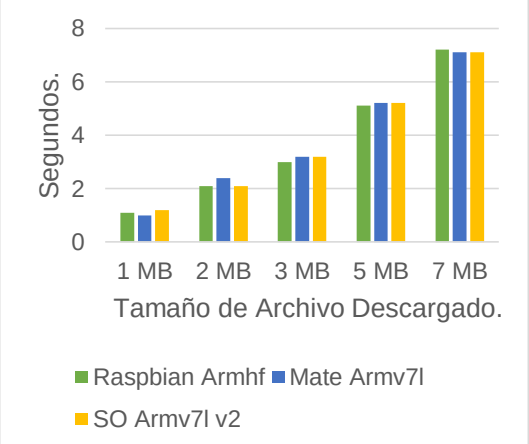
Grafica 31. Comparativa entre SO Armv7l Generados por la Metodología Administrando la Red.



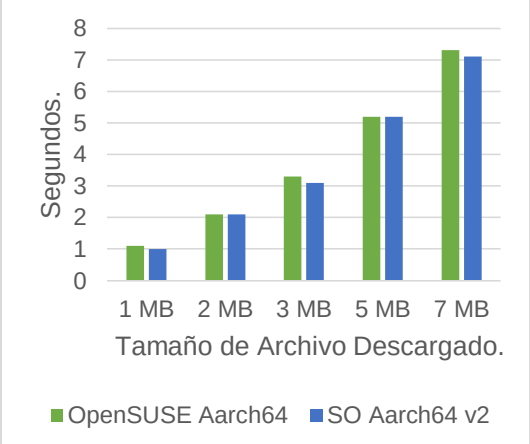
Grafica 32. Comparativa entre SO Aarch64 Generados por la Metodología Administrando la Red.



Grafica 33. Comparativa entre Raspbian, Ubuntu Mate y el SO Armv7l v2 Administrando la Red.



Grafica 34. Comparativa entre OpenSUSE y el SO Aarch64 v2 Administrando la Red.



El comportamiento de cada SO es similar en esta prueba, esto muestra que la administración de los recursos en todos los sistemas esta de manera predeterminada, debido a que no se realizó una configuración especifica en la

construcción de los SO generados por la metodología. Se concluye que los SO generados son una buena base para iniciar el desarrollo de distribuciones que extraigan el mayor potencial de un dispositivo embebido. Esto genera la necesidad de profundizar en el desarrollo de distribuciones computacionalmente eficientes para hardware embebido. Aunque la potencia computacional del hardware embebido es modesta, si se agrupa, puede lograr cuotas de desempeño aceptables, reduciendo el consumo eléctrico y costos de hardware. Si se utilizan opciones de optimización correctas se puede mejorar significativamente las prestaciones de la distribución. Se concluye entonces que se necesita más tiempo para estudiar opciones de optimización para aprovechar los distintos componentes del hardware.

8. CONCLUSIONES

La metodología de construcción de distribuciones propuesta es el logro destacable del proyecto debido al rendimiento obtenido por las versiones generadas, porqué, son una buena base para el desarrollo de proyectos que busquen acelerar el uso del recurso que más usa la aplicación objetivo del proyecto y esta es a su vez, la razón de la metodología.

La caracterización y pruebas realizadas a las distribuciones más utilizadas en la Raspberry Pi 3 para conocer las prestaciones del hardware nos muestran que cada sistema administra la SBC a su manera, porqué, cada SO está configurado para tener un mejor rendimiento en el tópico que los desarrolladores consideran necesario pero que no siempre puede cubrir las necesidades del proyecto. Para la caracterización de las distribuciones más usadas en la Raspberry Pi 3 se concluye que además de las características físicas, el SO es decisivo a la hora de evaluar una SBC para el desarrollo de un proyecto, usar un SO genérico puede funcionar bien, pero cabe la posibilidad que no saque todo el potencial de la tarjeta en el desarrollo de una tarea específica. Para dar base a esta conclusión se toma el desempeño de la Raspberry Pi 3 en la Grafica 3, 4, 5, 6, que usan diferentes SO para cumplir la tarea de calcular 10.000 primos, luego asignar o reasignar 4 bloques de 4 MB y liberarlos en RAM durante un minuto, seguido de escribir, leer y eliminar 128 MB de archivos temporales durante un minuto, por último, se descargan cinco archivos de diferentes tamaños y se toma su tiempo. Estos resultados muestran que cada SO administra a su manera el hardware, por eso la tarea de calcular primos para OpenSUSE es casi irrelevante, su administración de la RAM es baja y tiene el mejor rendimiento escribiendo y leyendo, cosas similares suceden para los otros SO populares.

Al revisar y seleccionar la herramienta de construcción de distribuciones se concluye que tanto los alcances de buildroot como yocto son similares dejando la decisión a factores como el tiempo de desarrollo del proyecto, personalización de la distribución, tamaño final del SO, gusto del equipo de trabajo del proyecto, etc. Porqué, en la tabla de comparación de las herramientas se muestran diferentes puntos que suman o restan en la elección, también al momento del diseño y implementación de la metodología se generan conclusiones que ayudan a afianzar la elección o recomendar cambios en proyectos futuros.

En el desarrollo de la metodología de construcción de distribuciones se genera una metodología que recopila, analiza, diseña e implementa de manera recursiva y evolutiva la construcción de distribuciones, porqué, a medida que se recopila y analizan los requisitos del SO se aprende sobre el funcionamiento de los componentes del SO que se desea construir y sus implicaciones en el proyecto, en el diseño e implementación podemos realizar un aprendizaje iterativo de las herramientas de construcción y desarrollo necesarias para lograr el objetivo del proyecto, a su vez, se generan conclusiones y recomendaciones para la documentación del proyecto y bases de referencia para futuros proyectos.

La comparación de los SO desarrollados con los SO elegidos da una buena base para aumentar la investigación y desarrollo de distribuciones de objetivo específico, porqué, los comportamientos expresados en los resultados obtenidos en las diferentes pruebas generan incógnitas que desde diferentes puntos de vista de las ciencias de la computación entre otras ramas de investigación, es necesario solucionarlas. Comparar los SO desarrollados por la metodología con versiones populares permite medir la utilidad de la misma, los SO generados son una buena base para iniciar el desarrollo de distribuciones que extraiga el mayor potencial de un dispositivo embebido ya sea en un uso genérico o una aplicación específica. Si se utilizan opciones de optimización correctas se puede mejorar significativamente las prestaciones de la distribución. Se concluye entonces que se necesita más

tiempo para estudiar opciones de optimización para aprovechar los distintos componentes del hardware. Debido a la multitud de hardware embebido, una amplia cantidad de estos no tienen los recursos necesarios para soportar una distribución completa, debido a esto, se concluye que la metodología es la mayor contribución para que se puedan usar esta amplia variedad de dispositivos en futuros proyectos de investigación y desarrollo.

9. RECOMENDACIONES

Para lograr el objetivo del proyecto se usaron diferentes combinaciones de arquitecturas como host de desarrollo, entre las diferentes configuraciones cabe destacar el uso de SSD, estos dispositivos reducen a un cuarto los tiempos de compilación de los SO.

Cabe resaltar que no se necesitan maquinas potentes para compilar estos sistemas, pero sí de conexiones amplias a internet, se recomiendan 10 MB, ya que todos los paquetes necesarios para la compilación son descargados de repositorios.

Si se van a generar varios SO, se recomienda tener un amplio espacio en disco, 100 GB sería una opción conservadora, aunque dependiendo de cómo se configure buildroot el tamaño de los SO puede ir desde 60 MB a 1 GB.

Se debe tener cuidado al momento de realizar las pruebas debido a que la RPi3 es una SBC de disipación pasiva y es muy susceptible al thermal throttling, reduciendo la frecuencia del procesador para controlar la disipación de calor.

BIBLIOGRAFÍA

ARDUINO. Home web site [En línea]. s.f. Disponible en: https://www.arduino.cc/

BANANA PI. Página principal [En línea]. s.f. Disponible en: http://www.banana-pi.org/#sbpc

BARR, M. Embedded Systems Glossary. [En línea] Disponible en https://barrgroup.com/Embedded-Systems/Glossary.

BEELINK. Portal web [En línea]. s.f. Disponible en: http://www.bee-link.com/portal.php

BERGER, A. S. Embedded Systems Design: An Introduction To Processes, Tools, & Techniques. (1 Ed.) New York: CRC Press. 2001

BLACKMORE C., RAY O., y EDER K. Automatically Tuning The Gcc Compiler To Optimize The Performance Of Applications Running On Embedded Systems. Bristol: University of Bristol. 2017

BRINKSCHULTE, U. Technical Report: Artificial Dna - A Concept For Self-Building Embedded Systems. Frankfurt: Johann Wolfgang Goethe Universit. 2018.

BUG LABS. Acerca de Bug Labs [En línea]. s.f. Disponible en: http://buglabs.net/about

CATSOULIS, J. Designing Embedded Hardware. (2 Ed.) Sebastopol: O'Reilly Media, Inc., 2005.

CESATI, M. y BOVET, D. Understanding The Linux Kernel. (3 Ed.) Sebastopol: O'reilly Media, Inc. 2005.

DAVCEV D., STOJKOSKA B., KALAJDZISKI S., TRIVODALIEV K. Project Based Learning of Embedded Systems. Skopje: University “Sts. Cyril and Methodious”. 2017.

DUBEY A., KARSAI G., GOKHALE A., EMFINGER W. & KUMAR P. Drems-Os: An Operating System For Managed Distributed Real-Time Embedded Systems. Nashville: Vanderbilt University. 2017

ESA. Leon: the making of a microprocessor for space [En línea]. 2013. Disponible en: http://www.esa.int/Our_Activities/Space_Engineering_Technology/LEON_the_making_of_a_microprocessor_for_space

F-CPU [En linea]. 2017. Disponible en: http://f-cpu.seul.org/

Fundación Raspberry Pi [En línea]. s.f. Disponible en: https://www.raspberrypi.org/

GANSSLE, J. G. The art of programming embedded systems. (1 Ed.) San Diego, CA: Academic Press. 2012.

GOOGLE. Google Efficient computing. [En línea]. Consultado el 27 de mayo de 2017. Disponible en https://www.google.com/about/datacenters/efficiency/internal/.

HAMACHER, C. Computer organization and embedded systems. (6 Ed.) New York: McGraw-Hill. 2011.

HARDKERNEL. Productos ODROID [En línea]. s.f. Disponible en: http://www.hardkernel.com/main/main.php

HEATH, S. Embedded Systems Design. (2 Ed.) Oxford: Newnes Elsevier Science. 2003.

LABROSSE, J. J. Embedded Systems Building Blocks : Complete And Ready-To-Use Modules In C. (2 Ed.) CMP. 1999.

MARTÍ M., y MAKI A. A multitask deep learning model for real-time deployment in embedded systems. Barcelona: Universitat Politecnica de Catalunya. 2017.

MARWEDEL, P. Embedded system design: embedded systems foundations of cyber-physical systems. (2 Ed.) Dordrecht: Springer.

MASSA A., y BARR M. Programming Embedded Systems. (2 Ed.) Sebastopol: O'reilly Media, Inc. 2009.

NÚÑEZ J., HOSSEINABADY M., AMIRI M., RODRÍGUEZ A., ASENJO R., NAVARRO A., GRAN R., y SUÁREZ D. Parallelizing Workload Execution in Embedded and High-Performance Heterogeneous Systems. Manchester: ACM. 2018.

OPENCORES. Projects: OpenRISC [En línea]. Disponible en: http://opencores.org/projects

ORACLE. OpenSPARC Overview [En línea]. Disponible en http://www.oracle.com/technetwork/systems/opensparc/index.html

ORANGE PI. Página principal [En línea]. s.f. Disponible en: http://www.orangepi.org/

PEARCE, J. M. Maximizar la rentabilidad de la inversión para la salud pública con hardware médico de código abierto. Gaceta Sanitaria. [En línea]. 2015. Consultado el 21 de junio de 2017. Disponible en http://www.gacetasanitaria.org/es/maximizar-rentabilidad-inversion-salud-publica/articulo/S0213911115000679/.

PEARCE, J. M. Quantifying the value of open source hardware development. Michigan: Moderm Economy. 2015.

SINGH R., GEHLOT A., SINGH B., Y CHOUDHURY S. Arduino-Based Embedded Systems. CRC Press. 2017.

SVOGOR I., y CARLSON J. SCALL: Software Component Allocator for Heterogeneous Embedded Systems. Varaždin: eprint arXiv. 2016.

TRIPATHI S., DANE G., KANG B., BHASKARAN V., NGUYEN T. LCDet: Low-Complexity Fully-Convolutional Neural Networks for Object Detection in Embedded Systems. 2017.

WIKIPEDIA. Green Computing. [En línea]. Consutado el 27 de Mayo de 2017. Disponible en: https://en.wikipedia.org/wiki/Green_computing.

YAGHMOUR K., MASTERS J., BEN-YOSSEF G., y GERUM P. Building Embedded Linux Systems. (2 Ed.) Sebastopol: O'Reilly Media, Inc. 2008.

ZURAWSKI, R. Embedded Systems Handbook. CRC Press. 2005.

ANEXOS

ANEXO A. CÓDIGO PARA PRUEBAS DE COMA FLOTANTE

```
#define stress_cpu_complex(_type, _ltype, _name, _csin, _ccos)
static void HOT OPTIMIZE3 TARGET_CLONES stress_cpu_ ## _name(const char
*name)
{
        int i;
        _type cI = I;
        _type a = FP(0.18728, _ltype) + cI * FP(0.2762, _ltype),
                b = mwc32() - cI * FP(0.11121, _ltype),
                c = mwc32() + cI * mwc32(), d;
        (void)name;
        for (i = 0; i < 1000; i++) {
                float_ops(_type, a, b, c, d, _csin, _ccos);
        }
        double_put(a + b + c + d);
}
```

Para profundizar en el código de las pruebas de CPU consulte: http://kernel.ubuntu.com/git/cking/stress-ng.git/tree/stress-cpu.c

Anexo B. Código para la prueba de solución de phi

```
static void HOT OPTIMIZE3 TARGET_CLONES stress_cpu_phi(const char
*name)
{
        long double phi; /* Golden ratio */
        const long double precision = 1.0e-15L;
```

```
        const long double phi_ = (1.0L + sqrtl(5.0L)) / 2.0L;
        register uint64_t a, b;
        const uint64_t mask = 1ULL << 63;
        int i;
        /* Pick any two starting points */
        a = mwc64() % 99;
        b = mwc64() % 99;
        /* Iterate until we approach overflow */
        for (i = 0; (i < 64) && !((a | b) & mask); i++) {
                /* Find nth term */
                register uint64_t c = a + b;
                a = b;
                b = c;
        }
        /* And we have the golden ratio */
        phi = (long double)b / (long double)a;
        if ((g_opt_flags & OPT_FLAGS_VERIFY) &&
            (fabsl(phi - phi_) > precision))
                pr_fail("%s: Golden Ratio phi not accurate enough\n", name);
}
```

Para profundizar en el código de las pruebas de CPU consulte: http://kernel.ubuntu.com/git/cking/stress-ng.git/tree/stress-cpu.c

Anexo C. Código para prueba de búsqueda de primos

```
static void stress_cpu_prime(const char *name)
{
        uint32_t i, nprimes = 0;
        for (i = 0; i < 1000000; i++) {
                if (is_prime(i))
```

```
                    nprimes++;
        }
        if ((g_opt_flags & OPT_FLAGS_VERIFY) && (nprimes !=     78498))
                pr_fail("%s: prime error detected, number of primes between 0 and
                1000000 miscalculated\n", name);
}
```

Para profundizar en el código de las pruebas de CPU consulte: http://kernel.ubuntu.com/git/cking/stress-ng.git/tree/stress-cpu.c

Anexo D. Comando para copiar SO generado a una SD.

dd if=/home/usuario/Documentos/buildroot-XXXX.XX.X/output/image /sdcard.img of=/dev/sdX bs=1M

En la parte de if, se coloca la dirección en la que se encuentra la imagen generada por la compilación de buildroot, para of, coloque la dirección en la que se encuentra la SD. Para saber cuál es la dirección de la SD use uno de estos comandos, no olvide que debe estar en modo super usuario:
lsblk -fm, fdisk –l, df –h.

Anexo E. Código para pruebas de RAM

Consulte la siguiente página web para conocer el código de prueba de la ram: http://kernel.ubuntu.com/git/cking/stress-ng.git/tree/stress-malloc.c

Anexo F. Código para pruebas de escritura y lectura

Consulte la siguiente página web para conocer el código de la prueba de escritura y lectura: http://kernel.ubuntu.com/git/cking/stress-ng.git/tree/stress-hdd.c