Sistema embebido de tiempo-real para control de procesos físicos

Diego Saldaña-Higareda [1], Verónica Quintero-Rosas[1], Arnoldo Diaz-Ramirez[1], Vidblain Amaro-Ortega[1], Francisco Ibañez-Salas[1]

Tecnológico Nacional de México/ Instituto Tecnológico de Mexicali

[1]Departamento de Sistemas y Computación

Resumen

La Industria 4.0 tiene como finalidad la implementación de fábricas inteligentes, en las que será posible automatizar los procesos de producción de bienes y servicios, creando productos altamente personalizados y a bajos costos. La Industria 4.0 será posible gracias al uso de diversas tecnologías, entre las que destacan el Internet de las Cosas y los Sistemas Ciber-Físicos. La unión de estás dos áreas del conocimiento es posible gracias a la disponibilidad y uso de sistemas embebidos y sistemas operativos de tiempo-real. En la Industria 4.0 se plantea el uso de sistemas embebidos que monitorizarán y controlarán sistemas físicos, por medio de la utilización de sistemas computacionales y redes de comunicación inalámbrica. La plataforma de hardware del sistema embebido tiene recursos limitados, como memoria o capacidad de cómputo, por lo que su elección es muy importante. Además, debido a que este tipo de aplicaciones cuentan con restricciones temporales estrictas, es necesario el uso de sistemas operativos de tiempo-real diseñados específicamente para sistemas embebidos. La elección de la plataforma de software, que incluye al sistema operativo, entorno de desarrollo de aplicaciones y comunicación, es de vital importancia. En este artículo se presenta una evaluación de diversas arquitecturas de hardware para sistemas embebidos y se selecciona una de ellas. De igual manera, se presenta una evaluación de diferentes plataformas de software (i.e., sistemas operativos de tiempo-real) para sistemas embebidos y también se elige uno de ellos. Con las plataformas de hardware y software seleccionadas, se presenta un ejemplo de la implementación de un sistema embebido para el control de sistemas físicos, como base para el desarrollo de aplicaciones para la Industria 4.0

Abstract

The main goal of Industry 4.0 is the implementation of smart factories, where the production of goods and services will be highly automated and customized, and at a low price. Industry 4.0 will be possible thanks to the use of a diversity of technologies. Among them, two technologies stand out: the Internet of Things and Cyber-Physical Systems. The convergence of these areas is possible due to the availability and use of embedded systems and real-time operating systems. The Industry 4.0 foresees the use of embedded systems to monitor and control physical systems, through the use of computer (cyber) systems and a wireless network. The hardware platform has limited resources, such as memory and processing capacity. For this reason, selecting the most appropriate is very important. Also, the selection of the software platform, which includes the operating system, the development environment and the network communication capabilities, is an important issue as well. In this document, the evaluation of several hardware architectures for embedded systems is introduced, and one of them is selected for demonstration purposes. Additionally, an evaluation of different software platforms (i.e., real-time operating systems) is presented, and also one of them is selected. The selected hardware and software platforms are used to show how to implement an embedded system to control a physical system, as an example of the development of applications for the Industry 4.0.

Palabras Clave: **Sistemas embebidos; Internet de las cosas; Sistemas de tiempo-real; Nuttx.**

Keywords: **Embedded systems; Internet of things; Real-time systems; Nuttx.**

1. INTRODUCCIÓN

La Industria 4.0 (I40) tiene como objetivo la implementación de fábricas inteligentes (*smart factories*) a través del uso de las tecnologías de la información y comunicación (TIC). El concepto de I40 fue utilizado por

primera vez en Alemania en el año 2014 por Bunse *et al.* [1]. En su artículo, los autores definieron que "*la industria inteligente o la Industria 4.0 se refiere a la evolución de los sistemas embebidos a los sistemas cíber-físicos…".* Además, establecieron que esta evolución se dará en el contexto de la Internet de las Cosas, en donde los sistemas embebidos estarán conectados por una red de comunicación entre si y a Internet [2]. Un *sistema embebido* (SE), también conocido como sistema *inmerso* o *incrustado*, es un sistema de procesamiento de información (i.e., una computadora) que está incrustado en un dispositivo mas grande [3]. Ejemplos de aplicaciones de sistemas embebidos son los automóviles, aviones, trenes, así como equipos de comunicaciones o de manufactura.

Los avances en microelectrónica han propiciado el desarrollo de sistemas embebidos con mayores capacidades de cómputo, así como el de una gran cantidad de sensores que pueden ser conectados a ellos. Esto ha motivado el surgimiento de lo que se conoce como el paradigma de la Internet de las Cosas (IoT), en el que los SE, equipados con sensores y capacidades de comunicación inalámbrica, son capaces de monitorizar sistemas físicos y enviar la información recolectada a un servidor [4]. Ejemplos de uso de sistemas embebidos para IoT pueden encontrarse en aplicaciones para la salud, agroindustria, sector automotriz y aeroespacial, por mencionar algunos [5].

Un sistema ciber-físico (CPS) es un sistema físico cuyas operaciones son monitorizadas, coordinadas y controladas por un núcleo computacional y un núcleo de comunicaciones [6]. Una característica fundamental de los CPS es que la ejecución de sus tareas de cómputo tiene restricciones temporales estrictas, que de no cumplirse puede tener consecuencias catastróficas. Por ejemplo, considérese un sistema de navegación autónoma de un vehículo. Si el sistema no detiene el vehículo a tiempo, después de detectar una posible colisión, los resultados pueden ser mortales.

La evolución de los sistemas embebidos a los sistemas ciber-físicos se lleva a cabo en el contexto del IoT, en donde los SE controlan al sistema físico con base en la información recolectada por los sensores o de otro sistema físico. La integración de estas dos áreas del conocimiento, junto con otras tecnologías, como la realidad aumentada o el aprendizaje automático (*machine learning*), ha dado como resultado lo que se conoce como la I40.

El núcleo computacional de un CPS es comúnmente un sistema embebido. Un SE puede visualizarse como una pequeña computadora con recursos limitados, que ejecuta un sistema operativo y un proceso de cómputo. El proceso puede ejecutar una o mas tareas (hilos) de manera concurrente. Con estos sistemas es posible controlar una gran cantidad de operaciones de uno o mas sistemas físicos. Una de las grandes ventajas del predominio del software en el SE es que pueden tenerse sistemas altamente reconfigurables, sin necesidad de modificar la plataforma de hardware.

El núcleo computacional de un CPS es un sistema de tiempo-real estricto. La teoría de planificación de tareas de tiempo-real estricto, particularmente para sistemas con un procesador, es un campo del conocimiento maduro con una extensa cantidad de resultados, lo que permite la implementación de sistemas con comportamiento predecible [7].

Para garantizar que las tareas de cómputo se ejecuten antes de sus plazos, es necesario que el sistema operativo utilizado en el SE sea un sistema operativo de tempo-real (SOTR). Los sistemas operativos de propósito general, como Windows o Linux, no permiten la implementación de sistemas de tiempo-real

estricto. La elección del SOTR para sistemas embebidos es de gran relevancia, ya que determinará el comportamiento y estabilidad del sistema.

En este artículo se presentan diferentes SOTR y se comparan con base en los criterios tales como arquitectura del procesador, especificaciones técnicas (Interfaces I/O, Sensores On Board), entre otros. Como resultado de la investigación se determinó que el SOTR Nuttx tiene muchas ventajas para ser utilizado en sistemas embebidos para la I40. Por tal motivo, en el documento se presentan su entorno, instalación, componentes, configuraciones, y se explica un ejemplo de control de un sistema físico.

El resto del documento está organizado de la siguiente manera. La Sección 2 está dedicada a explicar los sistemas embebidos y presentar un estudio comparativo de algunos de ellos. En la Sección 3 se aborda el tema de los SOTR para SE y se discute un comparativo entre algunos de los mas importantes disponibles actualmente. La Sección 4 presenta NuttX, un SOTR para sistemas embebidos, así como el proceso de configuración, compilación e instalación en un SE específico. La Sección 5 muestra un ejemplo de una aplicación en Lenguaje C que se ejecuta en NuttX para el control de un sistema físico. Finalmente, en la Sección 6 se presentan las conclusiones y el trabajo futuro.

2. SISTEMAS EMBEBIDOS

A inicios de los años 2000 se experimentó un cambio del uso de computadoras personales a sistemas más pequeños, con base en el desarrollo de dos tecnologías fundamentales: los sistemas embebidos y la comunicación inalámbrica. Este cambio motivó la aparición de paradigmas tales como la computación ubicua, la inteligencia ambiental, el cómputo móvil, el uso de dispositivos *vestibles* (wereables), y en última instancia, el Internet de las Cosas y la I40.

Los sistemas embebidos son sistemas informáticos incrustados en sistemas mas grandes. Como todo sistema informático cuenta con elementos de hardware, como procesador, memoria, puertos, etc. Otras características de hardware de un SE son: velocidad del reloj u oscilador, tamaño de palabra, memoria, I/O digitales, entradas analógicas, salidas analógicas, DAC (Digital to Analog Converter), ADC (Analog to Digital Converter), buses, UART, por mencionar algunas.

Adicionalmente, los SE cuentan con software, destacando el sistema operativo y las aplicaciones. Los SE tienen como característica que su funcionalidad se centra en realizar tareas dedicadas, a diferencia de las computadoras personales que son de propósito general y cubren un amplio rango de tareas. El software embebido, esto es, el software que se ejecuta en un SE, está estrechamente relacionado con procesos físicos, tanto para monitorizarlos como para controlarlos. Por tal motivo, el software embebido habitualmente debe operar con restricciones temporales estrictas, por lo que es necesario el uso de software específico, denominado como software de tiempo-real. El diseñador de aplicaciones que utilicen un SE para el control de sistemas físicos, o CPS, debe elegir el SOTR mas conveniente, así como el API de programación de sistemas de tiempo-real. Cabe señalar que uno de los criterios de selección del SOTR tiene que ver con su soporte a algún API de programación específico, como POSIX.

Existen una buena cantidad de arquitecturas de hardware de sistemas embebidos en el mercado. La elección de la mas conveniente está asociada al contexto de la aplicación. Sin embargo, existen criterios generales que son de gran utilidad en este proceso, entre los que destacan:

1- Soporte a diversos SOTR.
2- Arquitectura del procesador (preferentemente ARM).
3- Especificaciones técnicas (Interfaces I/O, Sensores *On Board*, capacidad para integrar mas sensores)

Para determinar las opciones del SE para la implementación de aplicaciones para la I40, se llevó a cabo un estudio de tarjetas de hardware para SE existentes en el mercado. A continuación, se describen brevemente algunas de las arquitecturas de SE evaluadas.

1- Eagle 100 (Micromint)

La tarjeta Micromint Eagle 100, es una computadora "single board" desarrollada para aplicaciones de control que requieran desempeño en tiempo real, utiliza microcontroladores de 8 y 16bits con el procesador ARM Cortex-M3 [8].

Algunas características destacables de la Eagle 100 incluyen:
- Puertos para LCD
- Soporte para compiladores GNU e IAR.
- Hasta 52 -5v, dispositivos de entrada y salida.
- Puerto para teclado 4x4

2- STM3240G-EVAL (ST Microcontrollers)

La tarjeta STM3240G-Eval, es una plataforma de demostración y desarrollo de alto desempeño, con un microcontrolador de 32-Bits, con procesador ARM Cortex-M4F [9].
Las características de hardware de la tarjeta permiten la evaluación de periféricos como USB-OTG HS, Ethernet, Motores, CAN, MicroSD, USART, audio DAC, RS-232, IrDA, SRAM, ST-MEMS, EEPROM y otros para desarrollar aplicaciones. El circuito integrado ST-LINKV/V2 puede utilizarse para JTAG y SWD debugging y programación.

Algunas características destacables incluyen:
- Compatibilidad con tarjetas ISO/IEC 14443 smartcard (Radiofrecuencias)
- 3.2" 240x320 TFT color LCD con touch screen
- Módulo para cámara y conector para plugin de cámara ST-camera

3- Tiva TM4C1294 (Texas Instruments)

El kit de evaluación de la tarjeta TM4C1294, es una plataforma de bajo costo para desarrollo en la arquitectura ARM Cortex-M4F de 32Bits, módulo de hibernación y una multitud de conectividad serial simultánea [10].

Algunas características destacables de la tarjeta incluyen:
- Dual 12-bit 2SMSPS, ADCs, Motion control PWMS
- On-board, in-circuit debug interface (ICDI)
- Múltiple compatibilidad con tool chains CCS, Keil, IAR, Mentor & GCC

ISSN: 2007-4786

4- SAML21 Xplained Pro (ATMEL)

EL kit SAML21 Xplained Pro, es ideal para evaluar prototipos el microcontrolador de bajo consumo eléctrico Atmel SAM L21 ARM Cortex-M0 [11].

Algunas características destacables de la tarjeta son:

- I/O para extender su funcionalidad a otros tableros ATMEL.
- Batería de respaldo
- Debugger embebido
- Interfaces: SPI, I^2C, 4 GPIOs. Virtual COM port (CDC)

De las placas de hardware evaluadas se consideró que la mas conveniente sería utilizar la tarjeta **STM3240G-Eval** por ser una arquitectura de evaluación ideal para comenzar a experimentar con SE. Cabe señalar que una vez que se domina esta tarjeta es mas sencillo utilizar y programar otras de la misma familia. Una vez evaluadas y seleccionada la arquitectura de hardware toca turno evaluar y seleccionar la plataforma de software.

3. SISTEMAS OPERATIVOS DE TIEMPO-REAL PARA SISTEMAS EMBEBIDOS

Los SE monitorizan y controlan sistemas físicos, como puede ser un automóvil, un avión, un robot, un satélite, una línea de producción, entre otros. Los sistemas físicos deben ajustar sus operaciones o comportamiento en ventanas de tiempo específicas. Uno de los mayores atractivos del SE es que puede controlar una enorme cantidad de actividades del sistema físico de manera concurrente. Pero debido a que cada una de estas actividades tiene restricciones temporales y muy probablemente compartan recursos entre sí, como puertos o estructuras de datos, es necesario el uso de SOTR, como ya se ha explicado. Las aplicaciones, implementadas generalmente como hilos de ejecución (*threds*), tienen parámetros tales como periodo, tiempo de ejecución de peor caso, y plazo de ejecución. Con el uso de algoritmos de planificación de tareas de tiempo-real, técnicas de sincronización, así como APIs específicos para sistemas de tiempo-real, es posible implementar sistemas con prestaciones de tiempo-real estrictos. Y con el uso de herramientas matemáticas, conocidas como pruebas de factibilidad, es posible garantizar el correcto comportamiento del sistema [11].

Los SOTR para SE son clasificados como sistemas operativos de perfil mínimo de acuerdo con el estándar POSIX [12]. Esta clase de sistema operativo no cuenta con sistema de archivos, interfaz gráfica, manejadores de discos externos, por mencionar algunos. Generalmente pueden ejecutar tan solo un proceso, con una gran cantidad de hilos de ejecución. Tienen como objetivo el uso racional de los recursos limitados con los que cuentan.

Para la selección del sistema operativo que se implementaría en el proyecto se investigaron más de 180 SOTR. Con el objetivo de determinar cuál sería el más conveniente, se tomaron en cuenta puntos clave en su funcionamiento como, por ejemplo:

- Documentación
- Compatibilidad con POSIX
- Dirigido a sistemas embebidos
- Compatible con la arquitectura de las tarjetas STM ARM
- Licencia de código abierto

La Tabla 1 muestra los resultados de una investigación de diferentes SOTR para SE, que fueron evaluados con base en diferentes parámetros, entre los que destacan el que sean proyectos activos y que soporten el estándar POSIX de tiempo-real.

Nombre	Licencia	Plataformas	Página web oficial
RTEMS	Licencia general publica	ARM, Blackfin, ColdFire, TI C3x/C4x, H8/300, x86, 68k, Milkymist SoC, MIPS, Nios II, PowerPC, SuperH, SPARC, ER32, LEON, Mongoose-V	https://www.rtems.org/
NuttX	BSD (Berkeley software distribution)	ARM, AVR 8051, x86	http://nuttx.org/
Minix3	BSD	x86, ARM	http://www.minix3.org/
OpenBSD	BSD	alpha, amd6, arm, hppa, i386, landisk, loongson, luna88k, macppc, octeon, sgi, sparc64	https://www.openbsd.org/
RiotOS	LGPL (Lesser general public license)	AVR, ARM, MIPS32, MSP430, PIC32, x86	https://riot-os.org/

Tabla 1.- SOTR evaluados

De los SOTR evaluados destacó NuttX, mismo que se describe con mas detalle en la siguiente sección.

4. NuttX

En 2007 Gregory Nutt lanzó el SOTR Nuttx bajo la licencia BSD. Con una amplia gama de funcionalidades, Nuttx se mantiene como una de las mejores opciones para proyectos e investigaciones para el control de sistemas físicos. Además, debido a que cumple con el estándar UNIX, soporta POSIX y ANSI. Otro de sus atractivos es que existen versiones para diversas arquitecturas embebidas de hardware, como Atmel AVR, ARMFIFO, Intel, entre otras [13]. A continuación, se describe el proceso de instalación y configuración de NuttX.

4.1. Preparación del entorno

En primer lugar, es necesario elegir la arquitectura de hardware del SE. En este caso se decidió utilizar aquella que contara con procesador ARM, gracias a su alta disponibilidad, bajo costo y buen rendimiento. La placa seleccionada, como ya se mencionó, fue la tarjeta STM3240G-Eval.

Una vez seleccionada la arquitectura de hardware, para poder instalar NuttX es necesario disponer de ciertas paqueterías para un correcto funcionamiento. También, se recomienda usar una computadora personal que funcione como anfitriona o *host*. En este ejemplo, se utilizó una computadora personal Dell con 8 Gb de memoria RAM y procesador Intel i5. El sistema operativo de la computadora anfitriona fue Linux Ubuntu 16.04. Para el proceso de instalación de NuttX se requiere primero la instalación de diversos paquetes de software. A continuación, se presenta la manera de instalar los paquetes necesarios en Ubuntu, a través del comando *apt-get install*:

sudo apt-get install *automake bison build-essential flex gcc-arm-none-eabi gperf git libncurses5-dev libtool*

libusb-dev libusb-1.0.0-dev

Una vez que las paqueterías han sido instaladas correctamente, es necesario crear un espacio en el sistema de archivos (disco duro) donde se instalará NuttX. Para esto, primeramente se requiere crear una carpeta en el lugar deseado. Por ejemplo, si se desea crear la carpeta de trabajo llamada *nuttxspace* en el directorio personal, se utilizan los siguientes comandos:

mkdir *~/nuttxspace*

cd ~/nuttxspace

Dentro de esta carpeta se procede a clonar el repositorio de *OpenOCD*, el cual proporciona herramientas para compilar sobre cualquier plataforma previamente soportada. El proyecto *Open On-Chip Debbugger* (*OpenOCD*) ofrece herramientas para la programación y depuración de sistemas embebidos que actúa como interfaz JTAG. A su vez, JTAG es un estándar de la industria para le verificación de diseños y operación de circuitos impresos. Para facilitar el desarrollo e instalación de software en el sistema embebido se utiliza *OpenOCD*. Para instalarlo primero se *clona* el repositorio con el siguiente comando:

git clone *http://repo.or.cz/r/openocd.git*

En el directorio de *OpenOCD* que se descargó, se crean los archivos de configuración. El programa *Bootstrap* que se encuentra en la carpeta, ejecutará las operaciones necesarias para la instalación de las herramientas de desarrollo. Para ejecutar el programa se utilizan los siguientes comandos:

cd *openocd*

./bootstrap

Cargadas las herramientas de *OpenOCD* se deben activar aquellas con las que se pretenda programar tarjetas. Se configura *OpenOCD* para utilizar tarjetas programadoras con el siguiente comando:

./configure *-enable-internal-jimtcl -enable-maintainer-mode -disable-werror -disable-shared enable-stlink*

-enable-jlink enable-rlink -enable-vslink enable-ti-icdi -enable-remote-bitbang

ISSN: 2007-4786

Una vez que se cuentan con los paquetes necesarios y se han hecho las configuraciones preliminares, se inicia la compilación de *OpenOCD* con el siguiente comando:

Make

Una vez que la compilación se haya completado, se procede a la instalación con el siguiente comando:

sudo make *install*

4.2. Componentes y configuraciones

Teniendo *OpenOCD* instalado es posible instalar NuttX. Para este fin se requiere descargar 3 repositorios: el sistema base de NuttX, un paquete de aplicaciones probadas para este sistema y las herramientas para configuraciones del sistema. En primer lugar, es necesario estar en la carpeta de trabajo *nuttxspace:*

cd ..

Para obtener las herramientas para configuración del sistema se utilizan los siguientes comandos:

git clone *https://bitbucket.org/nuttx/nuttx*
git clone *https://bitbucket.org/nuttx/apps*
git clone *https://bitbucket.org/nuttx/tools*

Es importante destacar que todo esto se debe hacer en el directorio que se eligió para alojar todo lo relacionado con NuttX. Dentro de la carpeta *tools* se encuentra otra carpeta llamada *kconfig-frontends*. Allí debe ejecutarse el script *configure* antes de compilar las herramientas necesarias para NuttX. Para hacerlo se ejecuta el siguiente comando:

./configure

Una vez que se ha hecho la configuración, se compila el núcleo de NuttX con el siguiente comando:

Make

Se continúa con la instalación con los siguientes comandos:

sudo make *install*
ldconfig

Será necesario ingresar a la carpeta *nuttxspace/nuttx* e ingresar a la carpeta *tools*:

cd tools

En ella se ejecuta el siguiente comando:

./configure.sh stm32f103-minimum/nsh

Con esto queda completada la instalación de NuttX en la computadora anfitriona.

ISSN: 2007-4786

4.3. Configuración del programador

Antes de iniciar con el procedimiento se deberá de conectar la tarjeta con el programador siguiendo la configuración especificada en la tarjeta, *ground* (de la tarjeta) con *ground* (del programador), etc. Se debe conectar el programador a la tarjeta en nuestra computadora. Posteriormente hay que ingresar al directorio */nuttxspace/openocd/contrib* para ejecutar el comando que permitirá acceder al programador SWD. Primeramente, se listan los archivos disponibles:

ls

El archivo que interesa encontrar es el que cuente con la siguiente estructura: *xx-openocd.rules*, donde el *xx* será un número de dos cifras que pueda variar. En nuestro caso fue el 60, aunque en otros puede ser 99, por lo que verificar antes de ejecutar el siguiente comando con tu número correspondiente sería lo más recomendado:

sudo cp *60-openocd.rules /etc/udev/rules.d/*

Se ejecuta el siguiente *trigger* para actualizar:

sudo *udevadm trigger*

Se continúa con el procedimiento ubicándonos en la carpeta de trabajo de nuevo. Antes de ejecutar el comando desconectemos y conectemos el programador para que las reglas copiadas surtan efecto.

lsusb

Con el comando anterior debemos ser capaces de observar en el listado el programador conectado marcando algo parecido a *STMicroelectronics STLINK/v2*. Después ejecutamos el comando para hacer la prueba del *openocd*:

openocd -f interface/stlink-v2.cfg -f target/stm32f1x.cfg

NOTA: En caso de tener una salida de error solo debe mantenerse presionado el botón de *reset* en la placa y al mismo tiempo ejecute de nuevo el mismo comando. Después simplemente deje de presionar y deberá notar una salida donde se especifique que se ha logrado la conexión.

4.3. Dialout y acceso a puertos

Independientemente del método que se escoja para establecer la comunicación entre la tarjeta y la computadora principal (interfaz de usuario), es necesario asegurar de que el usuario administrador sea miembro del grupo *dialout*, que es aquel que garantiza el acceso a los puertos mediante archivos. De otra forma es posible que se presenten problemas en la espera de respuesta de la aplicación o inclusive la imposibilidad de acceso a la tarjeta a través de los puertos seriales. Esto se puede resolver con el siguiente comando:

ISSN: 2007-4786

usermod -a -G dialout "username"

4.4. Programación de la tarjeta

Texane's Stlink [15] fue la herramienta seleccionada para programar la tarjeta con la aplicación generada en NuttX. *Stlink* es una herramienta de software libre para programar y depurar ST's STM32 Discovery kits. Estos kits tienen un chip que traduce los comandos enviados por una computadora anfitriona a comandos JTAG/SWD, que son utilizados por la familia de tarjetas STM32. Con la finalidad de evaluar otras alternativas, se utilizaron otras dos para este fin: *OpenOCD* y *STM32 ST-LINK Utility* (herramienta oficial para STM32). Mediante varias pruebas, *Texane's Stlink* fue la única en lograr una programación y una salida de datos exitosa. Se desconoce el motivo por el cual las últimas dos herramientas fracasaron; *STM32 ST-LINK Utility* logro una programación exitosa, pero sin salida de resultados y *OpenOCD* presentó problemas al programar la tarjeta y falló en lograr una salida de datos.

Primeramente, es necesario instalar los paquetes necesarios para la correcta operación de *Stlink*. La manera de instalarlos en Ubuntu es:

sudo apt-get install git libusb-1.0.0-dev pkg-config autotools-dev udev

Posteriormente es necesario clonar *Stlink* del repositorio a la carpeta de instalación (e.g. stlink).

git clone git://github.com/texane/stlink.git

A continuación, hay que configurar y compilar la herramienta:

cd stlink
./autogen.sh
./configure
make

También es necesario copiar el archivo *49-stlinkv2.rules* dentro de *udev*:

sudo cp . . . Downloads/stlink/49-stlinkv2.rules /etc/udev/rules.d/

Se debe también otorgar acceso a todo hardware perteneciente a STLINK V/2:

sudo udevadm control reload-rules

Para permitir el acceso a la carpeta *stlink* desde cualquier otra carpeta es conveniente agregar la ruta de la carpeta a la variable de entorno PATH:

export PATH=$PATH:$HOME/. . . /Downloads/stlink/

Finalmente, después de compilarlo, se instala el núcleo de NuttX en el sistema embebido:

st-flash write nuttx.bin 0x8000000

En la siguiente sección se explicará cómo compilar el núcleo de NuttX, contenido en el archivo binario (ejecutable) **nuttx.bin**. Una vez compilado y cargado en la tarjeta, el sistema embebido tendrá ya instalado el sistema operativo y es posible ejecutar aplicaciones que vienen como ejemplo. También es posible crear aplicaciones propias. Se revisarán estos dos casos a continuación.

5. Implementación de una aplicación en sistema embebido para controlar un sistema físico

Para el desarrollo de aplicaciones embebidas utilizando NuttX, una vez que se ha elegido la tarjeta (hardware) y se ha podido programar, debe elegirse el entorno de programación de las aplicaciones. De acuerdo con un estudio reciente, el lenguaje de programación C es uno de los mas utilizados para el desarrollo de SE debido a su eficiencia y velocidad [14]. Primeramente, es necesario instalar el compilador y entorno de programación para lenguaje C. De las alternativas disponible se decidió utilizar el conjunto de herramientas de software libre denominada *GNU ARM for ARM Emmbedded Processor* de *CodeSourcery*, que incluye compilador, ensamblador, enlazador, GCC *binutils*, depurador, bibliotecas estándares de C, entre otras. Para su instalación deben seguirse los siguientes pasos.

Primero es necesario descargar las fuentes del siguiente sitio:

http://www.codesourcery.com/sgpp/lite/arm/portal/package4573/public/armnone-linux-gnueabi/arm-2009q1-203-arm-none-linux-gnueabi.bin

Posteriormente es necesario crear y configurar los permisos de acceso a la carpeta de trabajo:

sudo mkdir -p /opt/codesourcery
sudo chmod ugo+wrx /opt/codesourcery

En caso de que el intérprete de comando por defecto no sea *bash*, cambiar a *bash*:

sudo rm /bin/sh
sudo ln -s /bin/bash /bin/sh

Hay que instalar los paquetes necesarios para la operación del entorno de desarrollo. A continuación se muestran la manera de instalar los paquetes requeridos:

sudo apt-get install libgtk2.0-0:i386 libxtst6:i386 gtk2-engines-murrine:i386 libstdc++6 libxt6:i386
sudo apt-get install libdbus-glib-1-2:i386 libasound2:i386

También se requiere instalar la máquina virtual de Java. En el ejemplo se instaló el *Java Runtime Environmnt 6*:

sudo apt-get install openjdk-6-jre

En caso de requerirlo y no estar en el repositorio oficial de paquetes Ubuntu una versión mas nueva del JRE, pero si existe en otro repositorio, es posible agregarlo a la lista de repositorios para su instalación. Aquí un ejemplo para el caso de la versión 9 del JRE:

sudo add-apt-repository ppa:openjdk-r/ppa

sudo apt-get update
sudo apt-get install openjdk-9-jre

Es común que el ejecutable para instalación que se descargó no tenga permisos de ejecución. En ese caso se asignan de la siguiente manera:

chmod ugo+x arm-2009q1-203-arm-none-linux-gnueabi.bin ./arm-2009q1-203-arm-none-linux-gnueabi.bin

La ejecución del programa de instalación puede hacerse en consola. Es importante hacerlo en la carpeta de trabajo.

./arm-2009q1-203-arm-none-linux-gnueabi.bin -i console

Una vez que se cuenta con el entorno de desarrollo es posible compilar el núcleo de NuttX y posteriormente comenzar a desarrollar aplicaciones embebidas. El SOTR NuttX opera con ciertas diferencias en algunas de las categorías de hardware disponibles. Dependiendo del dispositivo que se utilice, la preparación del ambiente de trabajo deberá cumplir con ciertos requerimientos exclusivos. Es por ese motivo que se recomienda experimentar con la plataforma antes de trabajar de manera completa. En esta sección se presenta un ejemplo del uso de NuttX para el control de un sistema físico. El objetivo es el de evaluar el desempeño de este SOTR para sistemas embebidos. Primeramente, se presentará como compilar y ejecutar una aplicación ejemplo que es parte de la distribución de NuttX.

5.1. Compilación de NuttX y de un programa ejemplo

Una vez que el entorno de la computadora anfitriona está completamente preparado, es posible proseguir con la compilación de NuttX y de un programa ejemplo. En este caso utilizando la tarjeta STM3240G-Eval, se muestra como compilar un programa sencillo, llamado "***Hello World example***" y que forma parte de la distribución de NuttX. A continuación, la secuencia de operaciones.

1. Se abre una terminal (intérprete de comandos) en la computadora anfitriona, preferentemente como administrador, en la carpeta raíz de NuttX. Por ejemplo: /home/username/Desktop/nuttx-dev/nuttx-7.22

2. Al estar ubicados en la carpeta raíz de NuttX es conveniente *limpiar* las configuraciones pasadas. Cada vez que se compila un programa en NuttX se debe ejecutar este comando: ***make*** *distclean*

3. Después hay que cambiarse a la carpeta *tools* con: ***cd*** *tools*

4. Allí se ejecuta el siguiente comando, especificando el nombre de la tarjeta que se utiliza y el nombre del programa a ejecutar (en este caso el programa tiene el nombre de *nsh*): **./configure.sh** stm3240g-eval/nsh

5. Regresamos a la carpeta raíz con: **cd ..**

6. Se ejecuta el menú de configuración gráfico con: **make** *menuconfig*

7. Primeramente, se configura la aplicación en NuttX para que funcione mediante la consola serial por USB. Regularmente esto no es necesario con otras tarjetas pues la salida serial usualmente está integrada con la salida USB, pero en este caso es necesario. Los pasos siguientes aportaran las configuraciones necesarias.

8. En el proceso de configuración se ingresa a la opción ***Build setup*** verificando que la opción ***Build Host Plataform*** esté orientada a Linux.

9. Se ingresa a la opción ***Binary Output Formats*** y se verifica que la opción ***raw binary format*** esté marcada.

10. Se ingresa a a opción ***System Type***. Allí se activa la opción ***STM32 Peripheral Support***.

11. En la opción ***STM32 Peripheral Support*** se verifica que las opciones "OTG HS y OTG FS" estén marcadas. Esto permitirá utilizar las entradas USB de la tarjeta.

12. Se ingresa a la opción ***Device Drivers***. Allí se activa la opción con el nombre de ***Serial Driver Support***.

13. Entramos a ***Serial Driver Support*** y se verifica que ***Serial console*** esté marcada como ***No serial console***.

14. Se activa la opción ***USB Device Driver Support***.

15. En ***USB Device Driver Support*** se marca la opción ***USB modem (CDC/ACM) support***.

16. En ***USB modem (CDC/ACM) support*** se selecciona la opción ***CDC/ACM console device***.

17. En ***Board Selection*** se activa la opción ***Enable boardclt() interface***.

18. Finalmente, una vez que la opción ***Enable boardclt() interface*** esté activa, se desplegaran varias opciones. De ellas se activa tan solo ***Enable USB device controls.***

19. Una vez terminadas las configuraciones para "***USB serial***" se prosigue con la configuración.

20. Después de salir al menú principal y se selecciona ***Networking Support*** y se verifica que la opción ***Networking support*** este desactivada si es que no se utilizarán servicios de comunicación en red.

21. En el menú principal se selecciona ***Application Configuration.*** En ella hay una opción llamada ***Examples***. Se activa la opción ***Hello, ¡World! example***, que es el ejemplo que se ejecutará.

22. En ***Application Configuration*** y solo como precaución, se verifica que en ***File System Utilities*** la opción ***mkfatfs utility*** esté desactivada.

23. Se sale del menú de configuración a la consola de comandos asegurándonos de salvar los cambios realizados.

24. Se ejecuta el comando: ***#make***. Esto compilará NuttX

25. Si la compilación fue exitosa, encontraremos en la carpeta raíz de NuttX un archivo llamado **nuttx.bin**. Como se comentó anteriormente, este archivo ejecutable contiene el núcleo de NuttX.

26. Prosiguiendo ahora con la programación de la tarjeta, se copia el archivo **nuttx.bin** a la carpeta donde se instaló Texane's stlink.

27. Se conecta la tarjeta STM3240G-Eval a la computadora anfitriona utilizado el cable USB A to B.

28. En la terminal se cambia a la carpeta de Texane's stlink y se carga el núcleo en la tarjeta con el siguiente comando: ***st-flash write*** *nuttx.bin 0x8000000.* Cabe señalar que el programa ***Hello, ¡World! example*** se incluyó en el archivo **nuttx.bin**.

29. Como se mencionó, se utiliza el cable USB A to Micro B, de la computadora a la tarjeta. Es importante que el extremo del cable a la tarjeta se conecte en la ranura OTG FS y no a la OTG HS.

30. En la terminal se ejecuta el comando ***dmesg***. Esto nos permitirá visualizar todos los dispositivos conectados a la computadora y verificar si lo último que se conectó fue el cable USB A to Micro B a la tarjeta.

31. Se ejecuta la aplicación *Minicom* con: ***minicon*** *-s*. Una vez en ejecución de configura la opción ***Serial port setup***.

32. Recordando el puerto que se mostró en la lista de dispositivos en la opción anterior, se cambia **Serial Device** por el que se mostró. Al mismo tiempo, ***Hardware/Software flow control*** deben estar en ***No***.***

33. Finalmente se sale de Minicom con "***Exit***".

34. Se presiona la tecla ***Enter*** tres veces para visualizar la ejecución de la aplicación.

35. Una vez dentro del ***nuttshell*** se escribe "**?**" para visualizar las aplicaciones disponibles en la tarjeta.

36. En la Fig. 1 es posible visualizar en el rectángulo rojo en el que la aplicación "***hello***", que se especificó en el menú de configuración, aparece en el listado de aplicaciones disponibles. Se ejecuta la aplicación escribiendo ***hello*** y será posible visualizar los resultados de una aplicación en NuttX ejecutándose en una tarjeta STM3240G-EVAL. La Fig. 2 muestra los resultados de su ejecución.

Figura 1.- Lista de aplicaciones disponibles en NuttX

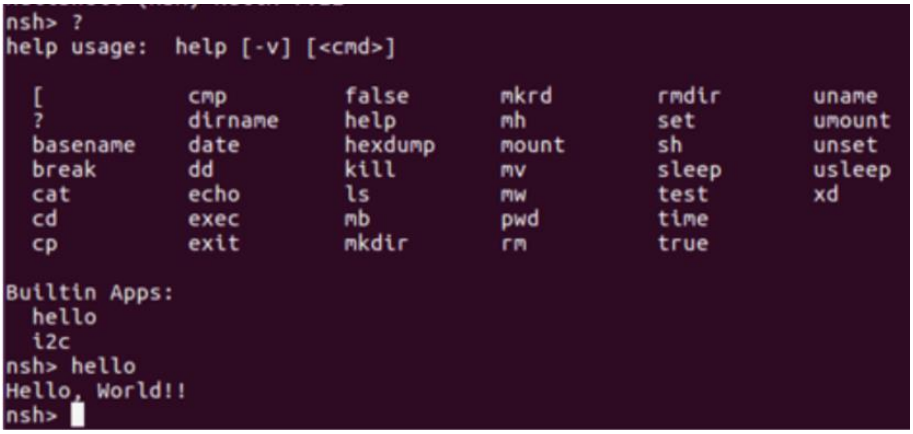

Figura 2.- Ejecución de aplicación Hello, World!

Una vez que se ha explicado cómo configurar NuttX, compilarlo, instalarlo en la tarjeta seleccionada y ejecutar una aplicación ejemplo, se mostrará como desarrollar una aplicación para el control de un sistema físico.

5.2. Aplicación embebida para el control de sistemas físicos

La aplicación desarrollada en esta sección utilizó el estándar de programación POSIX de tiempo-real [15], que está disponible en sistemas operativos UNIX, tales como Linux, MacOS y Android. Debido a que NuttX sigue el estándar UNIX, es posible aprovechar la portabilidad del estándar. Cabe señalar que se ejecuta tan solo un proceso y que cada actividad se implementa a través de hilos (*threads*) de POSIX.

El programa implementado en NuttX tiene como funcionalidad primaria la ejecución de tres hilos con distintas prioridades, que se muestran en el Algoritmo 1. Cada hilo se ejecuta de manera periódica. Esto es, cada hilo se activará periódicamente y una vez concluida su ejecución, se desactivará hasta el inicio del siguiente periodo, en el que se activará una nueva instancia del hilo.

```
param1.sched_priority = 97;
param2.sched_priority = 98;
param3.sched_priority = 99;
```

Algoritmo 1.- Asignación de prioridades a los hilos

Durante la ejecución de cada hilo se llevará a cabo la lectura de un sensor de profundidad HC-SR04. El dato recibido será solo el que fue tomado durante la ejecución del hilo. No se leen dos datos o más en una misma ejecución del hilo. Cuando la distancia tomada es menor o igual a 10 cm se encenderá un LED; si en la ejecución el dato es mayor a 10 el LED se apagará. En el caso de que siga siendo menor o igual a 10, el LED continuara encendido. La Fig. 3 muestra el diseño del sistema.

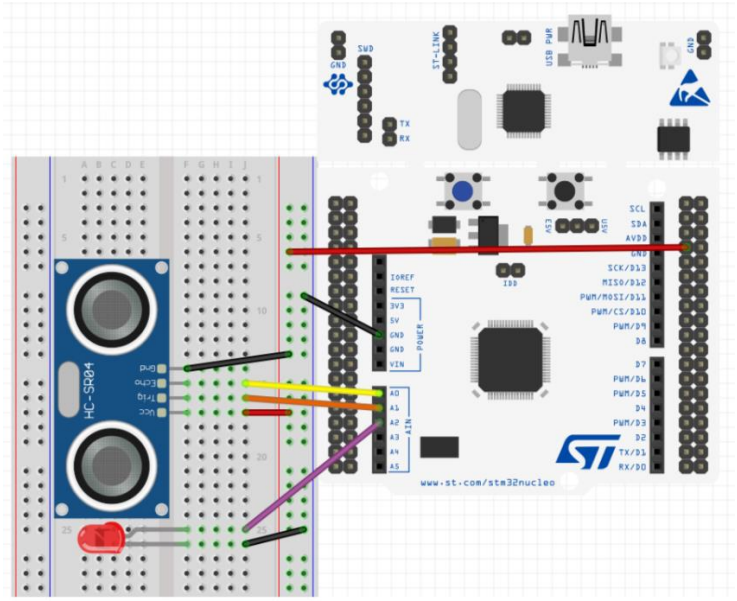

Figura 3.- Diagrama del circuito principal

El comportamiento de cada hilo está configurado en tres distintas funciones o métodos. Estos hilos primeramente deben ser creados y posteriormente ejecutados con respecto a sus configuraciones. Los hilos son creados en el procedimiento *main()* del programa. El Algoritmo 2 muestra la manera en que los hilos se crean. Dentro de los procedimientos de cada hilo se encuentra la sección de código responsable de la lectura del sensor y el comportamiento del LED.

```
/* Create threads*/
int main() {
.
  printf("THREAD CREATE BEFORE\n");
  pthread_create(&in, &inAttr, (void *) &inThread, NULL);
  printf("THREAD CREATE 01\n");
  pthread_create(&pr, &prAttr, (void *) &prThread, NULL);
  printf("THREAD CREATE 02\n");
  pthread_create(&out, &ouAttr, (void *) &ouThread, NULL);
  printf("THREAD CREATE 03\n");
  /* Join threads*/
  pthread_join(in, NULL);
  pthread_join(pr, NULL);
  pthread_join(out, NULL);
  printf("MAIN THREAD FINISHED\n");
  pthread_exit(NULL);

}
```

Algoritmo 2.- Creación de hilos

La obtención de datos del sensor no es de manera directa. El sensor está constantemente recibiendo datos y escribiéndolos en un archivo llamado *disto*, el cual es creado automáticamente dentro de la aplicación NuttX, en la ruta /dev/disto. Se debe tomar en cuenta que los datos arrojados por el sensor son datos crudos, por lo cual deben ser transformados en medidas legibles por medio de una simple división entre 58 para centímetros y entre 148 para pulgadas. Básicamente, lo que se hace es leer y guardar el último dato escrito en *disto* y convertirlo a centímetros. El Algoritmo 3 muestra el código para leer los datos del sensor y hacer las conversiones a centímetros

```
int c;
char str [ 10 ];
char *ptr;
FILE *_le;
_le = fopen("/dev/disto", "r"); // Se abre el archivo _disto_
if ( _le != NULL )
{
  char line[10];
  fgets ( line, sizeof line, _le );    //Se lee la última linea escrita
  c = strtol(line, &ptr, 10);    // Se convierte el dato leído a numérico
  c = c/58;    // Se transforma a centímetros
  printf(" %ld\n",c);    //Se imprime el dato recibido en centímetros
  fclose ( _le );
}
```

Algoritmo 3 Lectura y convención de datos del sensor escritos en _disto_

El manejo del LED funciona de una manera similar. Dentro de la carpeta de NuttX */dev* se encuentra un archivo llamado *userleds*, el cual nos ayudará a controlar el comportamiento del LED. Este archivo solo almacena datos en formato hexadecimal. Dependiendo del número de LEDs con los que se trabaje, los datos que reciba el archivo pueden variar. En este caso solo necesitamos dos datos hexadecimales: "*0x00*" para encender el LED y "0x01" para apagarlo. A diferencia de "*disto*" en él se almacena los datos leídos, *userleds* se utiliza para enviar datos.

5.3. Compilación y ejecución de la aplicación

En la carpeta de trabajo existe la carpeta *apps/examples*. En ella debe crearse la carpeta con el nombre del proyecto. En este caso es *hcsr05rt*. En esa carpeta deben estar tanto el código de la aplicación, en este caso *hcsr05rt_main.c*, así como el resto de los archivos necesarios, como *semaphore.h*, *time.h* y *times.h*. Para compilar la aplicación es necesario la creación de tres archivos. El primero de ellos es el denominado *Kconfig*, cuyo contenido se muestra en el Algoritmo 4.

```
config EXAMPLES_HCSR05RT
bool "\"hcsr05rt\" example"
default n
  ---help---
Enable the \"HC_SR05RT\"
if EXAMPLES_HCSR05RT
    config EXAMPLES_HCSR05RT_PROGNAME
    string "Program name"
    default "HCRT"
    depends on BUILD_KERNEL
     --help---
     This is the name of the program that will be use when the NSH ELF
     program is installed.
    config EXAMPLES_HCSR05RT_PRIORITY
    int "hcsr05rt task priority"
    default 100
    config EXAMPLES_HCSR05RT_STACKSIZE
    int "hcsr05rt stack size" default 2048
endif
```

Algoritmo 4.- Kconfig

El segundo archivo tiene el título de *Make.defs*, y en él se establecen definiciones necesarias en el proceso de compilación. La estructura de ese archivo para el proyecto se muestra en el Algoritmo 5.

```
ifeq ($(CONFIG_EXAMPLES_HCSR05RT),y)
    CONFIGURED_APPS += examples/hcsr05rt
endif
```

Algoritmo 5.- Make.defs

El último archivo es *Make*, que contiene las instrucciones y dependencias necesarias para compilar el programa. El Algoritmo 6 muestra su contenido.

```
-include $(TOPDIR)/Make.defs
# HC_SR05RT built-in application info
 CONFIG_EXAMPLES_HCSR05RT_PRIORITY ?= SCHED_PRIORITY_DEFAULT
 CONFIG_EXAMPLES_HCSR05RT_STACKSIZE ?= 2048
APPNAME = hcsr05rt
PRIORITY = $(CONFIG_EXAMPLES_HCSR05RT_PRIORITY)
STACKSIZE = $(CONFIG_EXAMPLES_HCSR05RT_STACKSIZE)
# HC_SR05RT Example
ASRCS = CSRCS = MAINSRC = hcsr05rt_main.c
CONFIG_EXAMPLES_HCSR05RT_PROGNAME ?= hcsr05rt$(EXEEXT)
PROGNAME = $(CONFIG_EXAMPLES_HCSR05RT_PROGNAME) include
$(APPDIR)/Application.mk
```

Algoritmo 6.- Make

Una vez completada la información de estos tres archivos es posible compilar la aplicación. Para esto hay que seguir los pasos explicados en la subsección 5.1, particularmente del 1 al 20. Debido a que se desarrolló una aplicación propia, en la sección ***Application Configuration***, en la sub-opción ***Examples***, se selecciona la aplicación desarrollada (hcsr05rt), como se muestra en la Fig. 4.

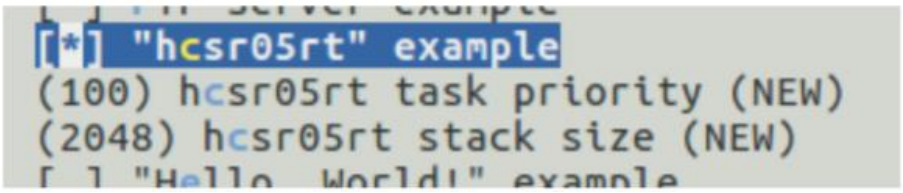

Figura 4.- Selección del ejemplo *hcsr05rt*

Así, ya que en la aplicación se utiliza un LED, es necesario seleccionar ***LED Support*** en la opción ***Device Drivers***, tal y como se muestra en la Fig. 5. En esa sección es importante seleccionar *LED driver* y *Generic Lower Half LED Driver*. Debido a que se utilizará un LED externo (ver Fig. 3) es necesario desactivar el LED interno de la tarjeta, en la Opción ***Board Selection*** y sub-opción ***Board LED Status Support***.

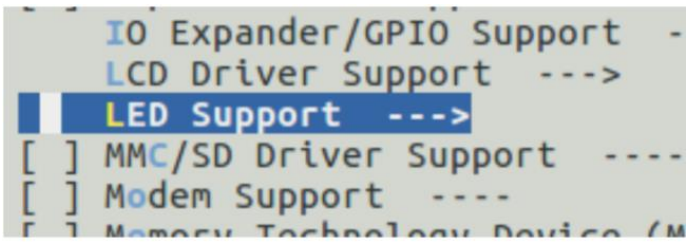

Figura 5.- Selección del ejemplo *LED Support*

Una vez concluido el proceso de configuración, para compilar se invoca al comando *make*:

make

Como resultado de la compilación exitosa del proyecto se generará el archivo ejecutable *nuttx.bin*, que es primero copiado a la carpeta de *Texane´s stlink* y posteriormente cargado en la tarjeta:

st-fashwritenuttx.binox8000000

Finalmente, se ejecuta la aplicación para verificar su comportamiento. Como se explicó anteriormente, para esto se utiliza la aplicación *Minicom*, que permite comunicar a la computadora anfitriona con la tarjeta a través del puerto serial, y así recibir, mostrar y visualizar la información procesada por el sistema embebido. Como se mencionó previamente, en la terminal *Minicom* se escribe *Exit* y posteriormente la tecla *Enter* tres veces, para acceder al proceso *NSH*. Para visualizar sus aplicaciones se ejecuta el comando "?", lo que debe mostrar la existencia de la aplicación *hcsr05rt*. La Fig. 6 muestra los resultados obtenidos al ejecutar dicha aplicación. Como puede observarse, los hilos ejecutan correctamente.

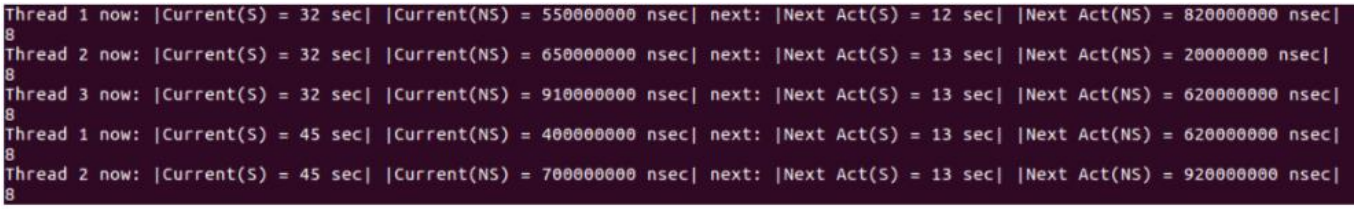

Figura 6.- Resultados de la ejecución de la aplicación *hcsr05rt*

6.- CONCLUSIONES Y TRABAJO FUTURO

Los SE son esenciales en aplicaciones del Internet de las Cosas y los Sistemas Cíber-Físicos, que además son la base de la Industria 4.0. Dos elementos son de suma importancia al implementar aplicaciones para estas áreas del conocimiento: la arquitectura de hardware y la plataforma de software. Con respecto a la arquitectura de hardware, y tomando en consideración las limitaciones de recursos inherentes a los SE, es de suma importancia su selección. Y con respecto a la plataforma de software, considerando la naturaleza de las aplicaciones para la I40, que tienen restricciones temporales estrictas, es imprescindible el uso de un sistema operativo de tiempo-real para sistemas embebidos, de tal manera que se garantice el cumplimiento de las restricciones temporales.

En este artículo se presentó un estudio de diversas arquitecturas de hardware y plataformas de software para la implementación de sistemas embebidos para la Industria 4-0. Se seleccionó una tarjeta de hardware, denominada STM3240G-Eval, ya que ofrece una buena cantidad de prestaciones, pertenece a una familia de placas con importantes capacidades, y utiliza un procesador altamente soportado: ARM. Además, se seleccionó un SOTR para sistemas embebidos, NuttX, debido a que cuenta con buen soporte, por ser un proyecto activo y por cumplir con el estándar UNIX, lo que garantiza portabilidad de sus aplicaciones a través del estándar POSIX. En el artículo se presentó como configurar y utilizar ambas plataformas, y se presentó un ejemplo del desarrollo de una aplicación que interactúa con un sistema físico.

Como trabajo futuro se tiene el investigar el uso de diferentes tecnologías de comunicación inalámbrica, como ZigBee o LoRa, con las plataformas utilizadas en este documento.

7.- REFERENCIAS BIBLIOGRÁFICAS

[1] B. Bunse, H. Kagermann, H. Kagermann y W. Walhster, «Industrie 4.0 smart manufacturing for the future,» *Centre for Science, Technology and Innovation,* pp. 1 - 38, 2014.

[2] A.-W. Scheer, «Whitepaper - Industry 4.0: From vision to implementation,» pp. 1-2, 09 2015.

[3] P. Marwedel, Embedded Systems Design, Embedded systems foundations of cyber-physical systems, Springer, 2006.

[4] J. A. Stankovic, «Research Directions for the Internet of Things,» *IEEE Internet of Things Journal,* vol. 1, nº 1, pp. 3 - 9, 2014.

[5] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari y M. Ayyash, «Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,» *IEEE Communications Surveys & Tutorials,* vol. 17, nº 4, pp. 2347-2376, 2015.

[6] R. Rajkumar, I. Lee, L. Sha y J. Stankovic, «Cyber-physical systems: The next computing revolution,» de *In proceedings of the Design Automation Conference*, 2018.

[7] L. Sha, T. Abdelzaher y et al., «Real Time Scheduling Theory: A Historical Perspective,» *Real-time systems,* vol. 28, pp. 101 - 155, 2004.

[8] M. Wells, «Micromint USA, Micromint.com,» 2017. [En línea]. Available: http://www.micromint.com/component/content/article.html?id=77. [Último acceso: Febrero 2019].

[9] S. Microeletronics, «STM3240G-EVAL - Evaluation board with STM32F407IG MCU,» [En línea]. Available: http://www.st.com/en/evaluation-tools/stm3240g-eval.html. . [Último acceso: Febrero 2019].

[10] T. Instrument, «ARM® Cortex®-M4F-Based MCU TM4C1294 Connected LaunchPad™ Evaluation Kit,» 2016. [En línea]. Available: http://www.ti.com/tool/EK-TM4C1294XL. [Último acceso: 2016 Febrero].

[11] G. Butazzo, Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, Springer, 2011.

[12] M. A. Rivas y M. G. Harbour, «Evaluation of new POSIX real-time operating systems services for small embedded platforms,» de *In proceedings of IEEE 15th Euromicro Conference on Real-Time Systems*, Porto, Portugal, 2013.

[13] C. o. nuttx.org, «NuttX Real-Time Operating System --- NuttX Real-Time Operating System,» 2018. [En línea]. Available: http://nuttx.org/doku.php?id=nuttx&rev=1536409789.

[14] S. Cass y P. Bulusu, «IEEE Spectrum Interactive: The Top Programming Languages 2018,» 31 Julio 2018. [En línea]. Available: https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018. [Último acceso: Febrero 2019].

[15] B. Gallmeister, POSIX.4 Programmers Guide: Programming for the Real World, O'Reilly Media, 1995.

[16] M. K. D. P. M. Dr. Ralf C. Schlaepfer, «Industry 4.0 Challenges and solutions for the digital transformation and use of exponential technologies,» 2015. [En línea]. Available: https://www2.deloitte.com/content/dam/Deloitte/ch/Documents/manufacturing/ch-en-manufacturing-industry-4-0-24102014.pdf.

[17] T. Noergaard, Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers, Elsevier, 2005.

[18] J. W. Valvano, Embedded Microcomputer Systems: Real Time Interfacing, Global Engineering: Christopher M. Shortt, 2012.

[19] C. Y. Qing Li, Real-Time Concepts for Embedded Systems, CMP Books, 2003.

[20] A. C. A.-D. Remzi H. Arpaci-Dusseau, Operating Systems: Three Easy Pieces, Arpaci-Dusseau Books, 2018.

[21] H. Kopetz, REAL-TIME SYSTEMS: Design Principles for Distributed Embedded Applications, KLUWER ACADEMIC PUBLISHERS, 1997.