Adaptación de Driver Serial para la placa STM32F103C8T6 y su utilización en controladores de GSM – Bluetooth

Waldo Valiente, Esteban Carnuccio, Mariano Volker, Graciela De Luca, Gerardo García, Raúl Villca, Germán Lorenz, Matias Adagio
Departamento de Ingeniería e Investigaciones Tecnológicas
Universidad Nacional de La Matanza
Dirección: Florencio Varela 1703 – CP 1754 –
{wvaliente, ecarnuccio, mvolker, gdeluca, ggarcia} @unlam.edu.ar,
{raul.villcasd, germangelv, mati.adagio} @gmail.com

RESUMEN

En la actualidad la gran mayoría de los proyectos orientados a Internet de las Cosas, utilizan sistemas embebidos en donde resulta necesario emplear comunicación serial para comunicar la plataforma seleccionada con los distintos componentes de hardware que conforman el dispositivo IoT. Existen diversas cantidades de sensores, actuadores y mecanismos de comunicación que emplean el protocolo serial para su correcto funcionamiento. Las bases del sistema embebido que se está realizando en esta investigación se sustentan en este principio. En consecuencia, el presente trabajo expone de qué manera se desarrolló un módulo serial que permitiese utilizar la plataforma STM32F103C8T6 con dispositivos de este tipo. A partir de la utilización de este módulo, se explica de qué forma se generó un Controlador GSM, que permite manejar el SIM800L a través de comandos AT, de forma no bloqueante en un único hilo de ejecución. De la misma manera se detalla cómo se desarrolló un Controlador Bluetooth, que otorga la posibilidad de monitorear y utilizar en forma remota el dispositivo IoT durante la etapa de desarrollo. Por ese motivo en el presente trabajo se explican los métodos empleados en dicha construcción, detallando el algoritmo utilizado.

Palabras clave: *STM32, bibliotecas HAL, Driver Serial, SIM800L, HM-10.*

CONTEXTO

Nuestra Línea de Investigación es parte del proyecto *Dispositivo de asistencia de personas mediante monitoreo y análisis de datos en la nube*, dependiente de la Unidad Académica del *Departamento de Ingeniería e Investigaciones Tecnológicas*, perteneciente al programa de Investigaciones CYTMA2 de la Universidad Nacional de La Matanza, el cual está formado por docentes, investigadores y alumnos de las carreras de ingeniería en informática e ingeniería en electrónica. Este proyecto es continuación de los trabajos que viene realizando el grupo de investigación, en sistemas operativos, computación de alto rendimiento y en Internet de las cosas.

1. INTRODUCCIÓN

Esta investigación pretende desarrollar un dispositivo que ayude a realizar esta tarea de asistencia remota empleando el concepto de Internet de las Cosas, de acuerdo a lo expuesto en [1]. Implementando un dispositivo IoT que permita la asistencia de personas mediante monitoreo remoto. Debido a que el sistema embebido IoT, debe ser fácilmente portable para

que el usuario lo pueda utilizar en su vida cotidiana, el tamaño del dispositivo y su consumo energético son factores preponderantes. En consecuencia a partir de un extenso estudio para poder cumplimentar estos requisitos de hardware adecuadamente, se ha seleccionado la arquitectura Cortex-M3, a través de la utilización de la placa STM32F10C8T6, también conocida como Blue Pill. [2]. El dispositivo IoT que se está desarrollando utilizará distintos sensores, actuadores y mecanismos de comunicación. Entre los principales sensores y actuadores que se están utilizando se encuentran: el acelerómetro MPU6050 y el GPS C3-470. No obstante una de las partes elementales del dispositivo IoT es la parte comunicacional con componentes externos; tales como un servidor en la nube, computadoras y otros dispositivos Bluetooth. Para ello se está utilizando el SIM800L V2, para la comunicación GSM/GPRS con un servidor, además del Bluetooth 4.0 HM-10. La característica destacable de estos componentes es que son dispositivos seriales, por lo tanto las comunicaciones deben ser realizadas a través del protocolo serial. En consecuencia el software, que se está desarrollando para funcionar en el embebido, deberá emplear un driver que entienda dicha comunicación.

2. LINEAS DE INVESTIGACIÓN Y DESARROLLO

La placa Blue Pill puede ser programada de dos formas distintas: la primera utilizando programas Arduino, empleando el lenguaje Wiring. Este lenguaje anexa una compleja capa de abstracción que afecta la performance del sistema. Por ese motivo se determinó que era conveniente emplear la segunda metodología, que consiste en realizar la programación de estas placas utilizando el lenguaje C y C++, y emplear las bibliotecas HAL y LL [3], que ofrecen la familia de los microcontroladores STM32, las cuales permiten un mayor control de los recursos del hardware y son autogeneradas por la herramienta STM32Cube-MX, que permite configurar la Blue Pill fácilmente. También el código fuente que autogenera el STM32Cube-MX, es creado en lenguaje C únicamente, por lo que por defecto, los proyectos deben ser construidos utilizando ese lenguaje de programación. No obstante, debido a los beneficios que ofrece la programación orientada a objetos con C++ [4] [5], se determinó conveniente programar utilizando también este último lenguaje. Por consiguiente el código del programa embebido está siendo desarrollado en forma hibrida, parte de la configuración inicial de la placa es realizada en C, a través de las bibliotecas HAL, y por otro lado el resto del programa embebido es desarrollado en C++. Adaptando el código fuente que autogenera el STM32Cube-MX y las opciones del compilador arm-none-eabi, exclusivo para la arquitectura ARM Cortex [6], para que el código fuente pueda ser desarrollado utilizando C y C++.

Diseño del Sistema

El diseño de los módulos del programa que se ejecutará en el dispositivo IoT como se muestra en la **Fig. 1** . El sistema es dirigido por un Controlador Principal, el cuál determina las acciones que deberán realizar los distintos componentes de hardware que estarán conectados a la Blue Pill. A su vez como una acción de un componente hardware conlleva distintas tareas complejas, ejecutando un conjunto de instrucciones, se determinó conveniente que estas sean realizadas por los demás Controladores para mayor simplicidad. Para el manejo de los dispositivos seriales se creó el módulo *Serial* encargado de dicha tarea.

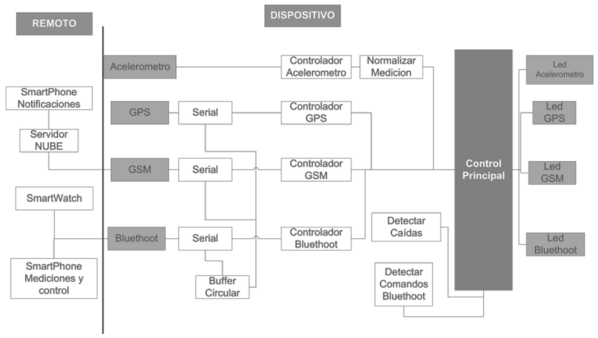

***Fig. 2** Diseño de los módulos que conforman el dispositivo IoT*

Adaptación de bibliotecas para el Driver Serial

El desarrollo del software necesario para poder comunicar de manera eficiente el Bluetooth, GPS y GSM/GPRS con la placa STM32., presenta tres mecanismos distintos para leer o escribir datos. Estas operaciones se realizan utilizando determinados registros de acuerdo al tipo de periférico con que se esté trabajando. El primer mecanismo consiste en la lectura y escritura de forma bloqueante por medio de la técnica de polling, el segundo en el uso de Interrupciones y tercer método utiliza transferencia de datos a través de DMA [7]. Haciendo uso de estas técnicas se desarrolló un driver genérico, que permite realizar la transferencia de datos de forma serial. Esto se efectuó de esta forma debido a que tanto el módulo de bluetooth, como el de GPS y el de GSM/GPRS realizan la transferencia de datos en forma serial mediante los puertos USART[1]. El dispositivo STM32F103C8T6 presenta tres puertos USART independientes, que son accesibles a través de sus pines de conexión. Para poder manipular los registros "Receive Data Register" (RDR) y "Transmit Data Register" (TDR) de la USART [7], se utilizaron parte de las bibliotecas HAL y LL configuradas con el software STM32Cube-MX, que debieron ser adaptadas en su funcionamiento, complementándolas por medio de la creación del módulo Serial (**Fig. 2**) para poder realizar la lectura y escritura continua de datos a través de la USART. Esto se debió a que las funciones de HAL y LL no están adaptadas para realizar dichas operaciones en forma ininterrumpida con la Blue Pill. Por lo que no se lograba completar la totalidad de las funcionalidades que estas otorgan. El módulo serial fue creado de manera que pueda ser configurado dinámicamente, permitiendo seleccionar el mecanismo de operación con el que se desea trabajar los puertos series: ya sea mediante polling, interrupciones o DMA. Una vez escogido el tipo de operación el driver realiza la transferencia de datos desde la USART al dispositivo a través de instrucciones que envían datos al registro TDR del puerto serie. Dado el tamaño limitado del registro, no es inmediato el envío de los bytes de las tramas al dispositivo serial destino. Por ese motivo fue necesario crear un buffer circular de transmisión que amortiguase los datos que se envían al dispositivo serial, hasta que el TDR se encuentre disponible para poder enviar esos bytes de información y otro buffer circular de recepción, con la finalidad de evitar pérdida de información en los datos que envían los dispositivos seriales a la Blue Pill. Estos datos son recibidos en su registro RDR.

El módulo Serial desarrollado está compuesto por una biblioteca que contiene una Clase denominada *Serial,* cuyos métodos pueden ser invocados desde cualquier parte del código fuente del programa embebido con tan solo crear un objeto de dicha clase. Por consiguiente, al iniciar el Controlador Principal se debieron instanciar tres objetos Seriales: el primero de ellos para emplear el GSM, el segundo para el Bluetooth y el tercero para poder acceder al GPS. Luego estos tres objetos son utilizados internamente por los módulos Controladores para acceder a los respectivos dispositivos. Los

[1] Universal Receptor Transmisor Sincrónico/Asincrónico

métodos más importantes de la Clase Serial son: *read y send*. De acuerdo al método de operación con el cual haya sido creado el objeto, el método *read* recibe los bytes que son enviados al puerto serial correspondiente y los retorna a su invocador. A su vez este método puede ser ejecutado de forma bloqueante o no, leyendo los bytes que fueron almacenados en el buffer circular de recepción correspondiente a dicho a objeto. Por otro lado el método *send*, permite enviar cadenas de bytes al puerto serial correspondiente, también de forma bloqueante o no bloqueante. En este último caso, cada vez que sea invocado el método *send*, se irán apilando en un buffer circular las cadenas que se desean enviar, que serán luego transmitidas cuando se libere el registro TDR por medio de un manejador de interrupciones. Para poder enviar varios parámetros al puerto serial, el método *send* utiliza las macros *va_start* y *va_end*, que son utilizadas en ANSI C para acceder a múltiples parámetros. Además para evitar problemas de sincronismo con los buffers circulares, fue necesario deshabilitar y habilitar las interrupciones en ciertas secciones del código fuente del programa. Esto se debe a que la programación del sistema embebido está siendo realizada sin multiprogramación, ejecutándose todo el programa en un único hilo de ejecución.

Utilización del Driver Serial

Los módulos de hardware GSM SIM800L y el Bluetooth HM-10 son dispositivos que deben emplear el módulo serial. Dentro del código fuente de sus Controladores se realiza los envíos de datos y comandos al componente, invocando para ello el método *send* del objeto que se encuentra asociado al mismo. Mientras que por otro lado la recepción de datos y comandos recibidos por estos componentes, es realizado a través del método *read*, para posteriormente poder analizar la información recibida y en base a ella actuar en consecuencia. Los comandos recibidos y enviados por el Controlador Bluetooth, permiten controlar y monitorear de forma remota el dispositivo IoT a través de un Smartphone, durante la etapa de desarrollo. En las etapas subsiguientes se pretende adaptar este Controlador para que pueda conectarse a otros sensores Bluetooth. Por otro lado el SIM800L funciona mediante el envío y ejecución de secuencias de comandos AT. Este componente, actúa con un patrón de ejecución-respuesta, el SIM800L ejecuta un comando AT y retorna una respuesta de confirmación de ejecución correcta. Por ese motivo la ejecución del Controlador GSM no puede ser bloqueante, aplicando las funciones HAL_Delay, debido a que se produciría una espera activa que afectaría el desempeño del sistema. Para resolver esto, muchas aplicaciones de STM32 hacen uso de paralelismo empleando un S.O. de Tiempo Real [8] [9]. No obstante estas soluciones no son óptimas en cuanto a rendimiento, debido a que ocupan demasiado espacio en memoria, entre otros factores. Por lo cual se desarrolló una solución no bloqueante, que se pueda ejecutar en un único hilo de ejecución, aplicando retardos no bloqueantes a través de contadores de tiempos de reloj, por medio de la función HAL_GetTicks. Aplicando dicho concepto, se desarrolló el funcionamiento del algoritmo que ejecutan los distintos comandos en los distintos Controladores. Para esto, se implementaron la codificación de diagramas de estados para la ejecución de los distintos comandos que conforman una operación (**Fig. 3**). Un ejemplo de ello es el caso del ControladorGSM, que para poder enviar un SMS a un número telefónico, debe ejecutar una de secuencia de tres comandos AT. En la figura **Fig. 3** se muestra el algoritmo básico que realiza dicha operación. Al ser no bloqueante, este método debe ser

ejecutado constantemente dentro del bucle del Controlador Principal.

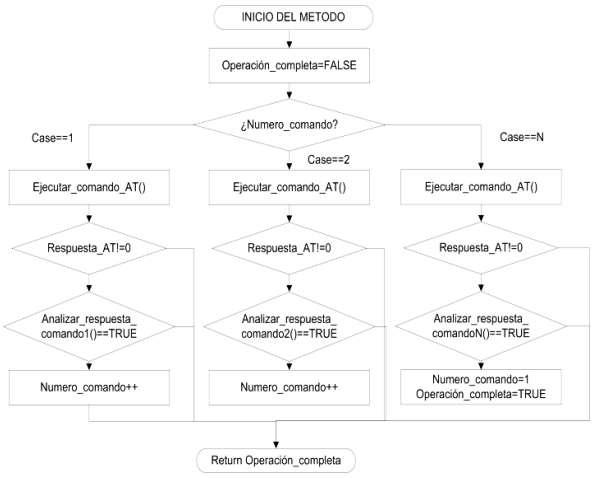

Fig. 3 Diagrama de flujo del Algoritmo básico para ejecutar un comando AT en forma no bloqueante

Un aspecto importante, es que dentro de la función Ejecutar_comando_AT se invoca al método *send,* para enviarle al SIM800L el comando AT que deberá ejecutar. Posteriormente en dicha sección de código, se realiza una espera sin detener el transcurso normal del programa, para poder recibir la respuesta del comando AT previamente ejecutado, utilizando el método *read*.

3. RESULTADOS OBTENIDOS/ESPERADOS

Se ha conseguido desarrollar un módulo que permite realizar una comunicación serial con distintos dispositivos que emplean dicho protocolo, adaptando las bibliotecas HAL y LL, que genera el STM32Cube-MX, De esta forma se han desarrollado algoritmos que permiten controlar y monitorear la ejecución del dispositivo IoT, durante la etapa de desarrollo en forma remota por medio del Bluetooth. Al mismo tiempo se han creado APIS que permiten ejecutar funcionalidades del módulo GSM como: Realizar llamadas telefónicas, enviar SMS y mensajes POST a un servidor, de forma no bloqueante en un único hilo de ejecución.

4. FORMACIÓN DE RECURSOS HUMANOS

La presente línea de investigación dentro del Departamento de Ingeniería e Investigaciones Tecnológicas forma parte del trabajo que dos investigadores se encuentran realizando para su tesis de maestrías. Completan el grupo de investigación cuatro alumnos de Ingeniería que se encuentran finalizando su formación de grado y realizan su iniciación a la investigación.

5. BIBLIOGRAFÍA

[1] G. De Luca, E. Carnuccio, G. Garcia, D. Guilianelli, M. Volker, W. Valiente, V. Raul y M. Vittorio, «Dispositivo de asistencia de personas mediante monitoreo IoT,» de *Libro de Actas Wicc 2018*, Abril 2018, pp. 854-858.

[2] STMicroelectronics, «Specifications STM32F103C8,» [En línea]. Available: https://www.st.com/en/microcontrollers/stm32f103c8.html.

[3] STMicroelectronics, Description of STM32F1 HAL and Low-layer drivers, 2017.

[4] S. Dan, «C++ for Embedded C Programers,» 2013. [En línea]. Available: http://www.dansaks.com/talks/ESC-205.pdf.

[5] upwork, 2017. [En línea]. Available: https://www.upwork.com/hiring/development/c-vs-c-plus-plus/.

[6] ARM, ARM Compiler User Guide, vol. Version 6.7, Cambridge, 2017.

[7] STMicroelectronics, de *STM32F101xx, STM32F102xx,STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs*, 2018.

[8] N. Askari, 2018. [En línea]. Available: https://github.com/nimaltd/Sim800_V2.

[9] T. Majerle, 2018. [En línea]. Available: https://github.com/MaJerle/GSM_AT_Lib.