Implementación de dispositivo de almacenamiento masivo USB en MicroPython como solución para almacenar en memoria flash el Bring-Up software en ECU'S

Ing. Francisco Javier Quirarte Pelayo[1] [2], Dr. José Cantoral Ceballos[3]

***Resumen*— Este trabajo describe la implementación para hacer de una unidad de control electrónico automotriz un dispositivo de almacenamiento masivo USB, para poder almacenar scripts de prueba de hardware, así como para poder iniciar la ECU con estos scripts de manera independiente, algo que es muy útil al someter el hardware a pruebas de resistencia, como en una cámara de temperatura. La implementación descrita en este documento se basa en el uso de un sistema de archivos FAT32 para dar formato a la memoria flash integrado en MicroPython que proporciona el entorno necesario para controlar el ECU por línea de comandos serial y crear un ambiente de pruebas.**

***Palabras clave*— Board bring-up, Electronic control unit (ECU), Mass storage device (MSD), Hardware validation, Universal Serial Bus (USB).**

Introducción

La industria automotriz se encuentra en una era de digitalización en la que los vehículos necesitan manejar más tecnología, más información y más seguridad que nunca, y las unidades electrónicas de control (ECU, por sus siglas en inglés) son sistemas cada vez más complejos, que manejan interfaces electrónicas de alta velocidad y de mayor demanda computacional. Por lo que siempre existirá la necesidad de mejorar y hacer más eficiente los procesos de validación de dichos productos [1].

En las etapas iniciales del desarrollo de los proyectos, el departamento de ingeniería eléctrica y mecánica realizan los primeros prototipos de hardware los cuales por lo general necesitan esperar al departamento de software para que libere un entregable y poder realizar las pruebas funcionales. Esto representa un alto riesgo de que, en caso de existir un problema de diseño electrónico y que este no se detecte sino en etapas tardías de desarrollo, la repercusión económica puede ser muy considerable. Por tanto, se necesita hacer más sencillo y flexible el proceso en que los ingenieros de hardware pueden correr las primeras pruebas de funcionamiento en los prototipos y poder así detectar errores y puntos de mejora lo más temprano posible en las fases de desarrollo, algo que con la metodología tradicional [2] no es posible.

Para atender esta problemática se creó una iniciativa llamada *"Bring-Up software"* (BUSW), este se puede definir como cualquier pieza de software que ayuda a probar y verifica las principales funcionalidades de un PCBA[4] en fases tempranas de desarrollo, es de un enfoque flexible, práctico y ágil y por ende no le aplican los mismos lineamientos ni especificaciones que el software de aplicación. Debe estar disponible para cuando se construya la primera muestra de hardware, y debe ayudar en la tarea de depurar las principales interfaces electrónicas. Idealmente, debe ser sencillo de usar, editar y *reflashear* por los ingenieros de hardware y debe poder correr de manera independiente para pruebas de laboratorio. Algunas de las acciones que debe cumplir el bring-up son las de controlar GPIO'S de los microcontroladores, controlar y configurar displays, interfaces de sonido, leer valores del ADC, controlar PWM, ejecutar pruebas de memoria tanto RAM como flash, controlar protocolos de comunicación inter IC's como I2C, SPI y UART, soportar comunicación por UART o CAN como medio de comunicación para enviar comandos para controlar y monitorear el dispositivo bajo prueba (DUT, por sus siglas en inglés) con una PC. Está solución a demostrado ser efectiva ayudando a lograr la validación eléctrica inicial en casi el 80% de las interfaces eléctricas en las primeras fases del proyecto [3].

El presente trabajo presenta una prueba de concepto que corre en el microcontrolador de gráficos (GC, por sus siglas en inglés) de la plataforma R-CarGen3 [4]. Esta implementación constituye una característica que complementa y amplía el alcance del uso del BUSW. El usuario podrá crear scripts de Python que ejecutan rutinas de prueba llamando a los drivers de hardware y cargar dichos scripts en la ECU o cualquier DUT equipado con

[1] Estudiante de Maestría en Sistemas Inteligentes Multimedia del Posgrado CIATEQ, A.C., unidad Jalisco, México
[2] Ingeniero de software en Continental Automotive Occidente, S.A. de C.V., francisco.quirarte@continental.com
[3] Profesor investigador en Tecnológico de Monterrey campus Querétaro, Querétaro, México, joseantonio.cantoral@tec.mx
[4] Primer PCB construida para un proyecto

Elibro con ISBN online
978-1-939982-56-8

dicho microcontrolador, como si este fuera un pendrive USB cualquiera, y opcionalmente, los scripts flasheados se puedan ejecutar de forma independiente al iniciar la plataforma.

Con esta característica, el BUSW logra el objetivo de ser más fácil de utilizar y actualizar por los ingenieros de hardware, principales usuarios de esta solución, además que se puede ampliar su caso de uso e importancia, como lo pueden ser pruebas de estrés ambiental como en una cámara de temperatura o anecoica.

A continuación, se presenta un resumen del contenido de cada sección de este trabajo. En la sección dos se describe la plataforma y las interfaces de hardware usadas en el presente trabajo. En la sección tres se describe el proceso de migración de MicroPython [5] a la plataforma R-CarGen3. En la sección cuatro se detalla el proceso de integración de FatFs [6] a MicroPython. En la sección cinco se menciona el proceso de adaptación del driver de RAMDISK[5] como driver de dispositivo de almacenamiento masivo (MSD, por sus siglas en inglés) USB. En la sección seis se presentan los resultados obtenidos y en la sección siete se redactan las conclusiones.

Componentes de hardware del sistema

La plataforma R-CarGen3, de Renesas Electronics, es la tercera generación de una familia de microcontroladores, empaquetados como system in chip (SiP), son de propósito múltiple, basados en la arquitectura ARM®, contienen capacidades graficas multimedia y de conectividad, son de grado automotriz y su rango de capacidades va desde los dispositivos de entrada (E3) hasta los más completos la serie highline (H3) [4] [7]. Cabe mencionar que el presente trabajo fue desarrollado y probado con plataformas de desarrollo para la serie midline (M3) y entry (E3), así que, de aquí en adelante, el nombre R-CarGen3, se refiere a ambas variantes M3 y E3.

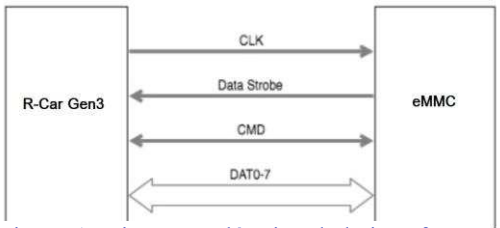

Figura 1. Diagrama eléctrico de la interfaz MMC del R-CarGen3.

Dentro del R-CarGen3 podremos encontrar un controlador de memoria flash llamado MMCIF0 [8], el cual soporta memorias de tipo SD card y MMC (multimedia card). En nuestro DUT dicha interface se conecta a una memoria eMMC (embedded multimedia card) de 32GB a como memoria flash (no volátil) del sistema [8], como se muestra en la figura 1.

El R-CarGen3 posee también, un controlador de USB 2.0 host controller que es compatible con la especificación Enhanced Host Controller Interface (EHCI) de la revisión 1.1 [7] del protocolo USB, dicha interfaz será el medio para poder conectar el PCBA con una PC como unidad de almacenamiento masivo. Dicha interfaz se ejemplifica en la figura 2.

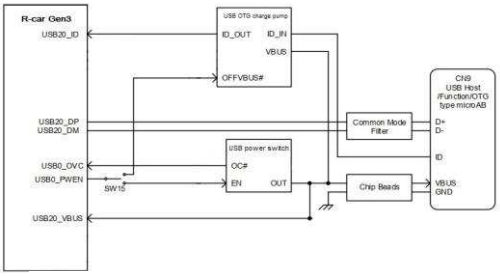

Figura 2. Diagrama eléctrico de la interfaz USB 2.0 del R-CarGen3.

Portando Micropython a la plataforma R-CarGen3

Para lograr el objetivo del BUSW se decidió usar Micropython por los muchos beneficios que este ofrece tanto como interfaz de usuario, como ambiente de desarrollo para integrar el BSP de R-CarGen3 y poder implementar un sistema de tipo línea de comandos en el que se pueden ejecutar las pruebas de hardware bare-metal.

MicroPython es una implementación eficiente y ligera del lenguaje de programación Python 3 que incluye un pequeño subconjunto de la biblioteca estándar de Python y está optimizado para ejecutarse en microcontroladores y en sistemas embebidos. MicroPython es un compilador completo de Python y un entorno de ejecución (runtime) que corre sin necesidad de sistema operativo (bare-metal). MicroPython posee una interfaz de línea de comandos interactiva (REPL) para interpretar y ejecutar comandos de Python de inmediato, con ello, es posible utilizar el hardware del microcontrolador [5]. Cuenta con la capacidad de ejecutar e importar secuencias de comandos (scripts) desde un sistema de archivos (si este está disponible). Igualmente, MicroPython ofrece un conjunto de funciones de manejo de archivos basados en la librería os de Python estándar [9].

[5] Uso de memoria RAM como unidad de almacenamiento temporal, con la ventaja de las grandes velocidades de lectura y escritura.

Para portar el proyecto de MicroPython a R-CarGen3 fue necesario una serie de procedimientos que se listan a continuación:

a) Se instaló un toolchain que soporte el BSP y el proyecto de MicroPython, nosotros elegimos: gcc-arm-noneeabi4_8-20131204 de linaro en una maquina con sistema operativo Linux (en nuestro caso una máquina virtual Ubuntu16.04 como ambiente de compilación).
b) Se creo un pequeño script para cargar la ruta del bin del toolchain al path de Ubuntu para cuando se ejecute el comando make desde la terminal, este sepa que librería va a utilizar. Dicho script se ejecuta cada vez que se inicie una nueva terminal para compilar.
c) Se descargo el proyecto de MicroPython desde el repositorio oficial en GitHub.
d) Se creo un nuevo folder con el nombre "rcar_m3" en la ruta: micropython/ports/rcar_m3.
e) Copiar y pegar los archivos de referencia que se encuentran en el folder minimal al folder "rcar_m3".
f) Se importaron los archivos del BSP dentro de ese folder.
g) Se edito el makefile añadiendo los nombres de los archivos de los drivers del BSP, se agregaron los paths de gcc y lib de linaro. Se ajusto el tamaño del heap y se integró el linkerfile del BSP.
h) Se integro el driver de comunicación serial del BSP con a los archivos de control de UART de MicroPython.
i) Se construyeron módulos de MicroPython para conectar los drivers principales del BSP, además de modificar su lógica, añadir parámetros de control o incluso rescribirlos parcialmente para poder hacer que desde Python se controle el hardware a bajo nivel.
j) Se compilo el proyecto utilizando la utilería make de GNU.
k) En la maquina host, que es de Windows, se instaló el emulador de terminal serial TeraTerm.
l) Se creo un script en TeraTerm para cargar el loader del BSP y después utilizando el loader cargar el binario de MicroPython, todo esto mediante comunicación serial UART.
m) Se verifico el funcionamiento del sistema con drivers básicos como timers y GPIOs.

Integración de FatFs con MicroPython

Para la implementación del sistema de archivos necesario para el MSD se usó el proyecto open source llamado FatFs, el cual es de libre distribución para educación, investigación y desarrollo, y que está escrito en conformidad con ANSI C (C89). FatFs es un módulo de software genérico que implementa el sistema de archivos FAT/exFAT para sistemas embebidos, para prácticamente para cualquier arquitectura y en sistemas con recursos limitados. El proyecto de FatFs soporta cualquier medio de almacenamiento físico ya que es independiente de este y su implementación, con la única condición de que sea un dispositivo orientado a acceso a bloques de tamaño fijo, cabe destacar que la capa de drivers de memoria es responsabilidad de quien porta este proyecto a una plataforma en particular [6].

Función	Requerida cuando
disk_status disk_initialize disk_read	Siempre
disk_write get_fattime disk_ioctl (CTRL_SYNC)	FF_FS_READONLY == 0[6]
disk_ioctl(GET_SECTOR_COUNT) disk_ioctl (GET_BLOCK_SIZE)	FF_USE_MKFS == 1[7]

Tabla 1. Funciones utilizadas de FatFs

Para la integración de FatFs en el ambiente de MicroPython se añadieron los archivos de FatFs al ambiente de compilación de MicroPython, se dieron de alta dichos archivos en el makefile y en los archivos de configuración como mpconfig y modhardware. Cabe destacar que solo se utilizaron las funciones más relevantes de FatFs [6] como se muestra en la tabla 1.

Fue necesario, además, elaborar una versión reducida del módulo OS del proyecto original de MicroPython para nuestra versión, usando para ello funciones de FatFs. Dicho modulo proporciona las API's para interactuar con los archivos del filesystem, como se ejemplifica en la tabla 2.

[6] Es decir, cuando dicha bandera esta deshabilitada, ya que, cuando esta se activa, el sistema se configura como solo lectura.
[7] Cuando la función f_mkfs está activa.

Función	Uso
os.chdir(path)	Cambiar el directorio actual.
os.getcwd()	Obtiene el directorio actual.
os.listdir([*dir*])	Sin argumento, muestra el contenido del directorio actual. De lo contrario, enumera el directorio dado.
os.remove(*path*)	Elimina un archivo.
os.rmdir(*path*)	Elimina un directorio.
os.rename(*old_path*, *new_path*)	Renombra un archivo.
os.stat(*path*)	Muestra el estado de un archivo o directorio. Provoca una excepción si el archivo no existe.
os.statvfs()	Muestra el estado del sistema de archivos. Devuelve una lista con los siguientes valores: ▪ tamaño de clúster / bloque del sistema de archivos ▪ tamaño de clúster / bloque del sistema de archivos. ▪ número de bloques ▪ cantidad de bloques libres. ▪ cantidad de bloques disponibles. ▪ cuatro ceros (no aplicable para el sistema de archivos FAT) ▪ longitud máxima del nombre del archivo
os.VfsFat.mount()	Monta el sistema de archivos en la eMMC
os.VfsFat.umount()	Desmonta el sistema de archivos
os.VfsFat.mkfs()	Formatea la partición en la eMMC
os.sep()	Carácter de separación usado en los *paths* del sistema de archivos
os.open(*path*, *mode*)	Alias de la función integrada open. Ejemplo: *o = open("archivo", "w")*
.read()	Devuelve todo el archivo como una cadena de caracteres si no se proporciona ningún parámetro. **Advertencia**: los archivos grandes pueden acabarse el espacio del heap de MicroPython. Si se proporciona un parámetro, devuelve el número de bytes que se proporciona como un parámetro entero. Ejemplo: *o.read()*
.write(*string*)	Escribe la cadena dada en el archivo. Ejemplo: *o.write("Hola Mundo!\n")*
.flush()	Vacía todo lo que se encuentra en RAM hacia la eMMC. Ejemplo: *o.flush()*
.close()	Cierra el archivo y vacía su búfer. Ejemplo: *o.close()*
.seek(pos)	Recibe un valor entero y coloca el cursor en la posición dada. Ejemplo: *o.seek(0)*
.tell()	Devuelve la posición del cursor como entero. Ejemplo: *o.tell()*

Tabla 2. Funciones principales de nuestra implementación del módulo OS y las API'S de interacción de archivos en el filesystem.

Usando el driver de eMMC proveniente del BSP del microcontrolador R-CarGen3 como base y tomado en cuenta que el BSP ya se había integrado con Micropython previamente, se modificaron los drivers para la escritura/escritura multibloque [10]. Posteriormente se crearon funciones de pegamento para poder conectar con las funciones disk I/O de acceso a memoria de FatFs. Dicha arquitectura se ilustra en la figura 3.

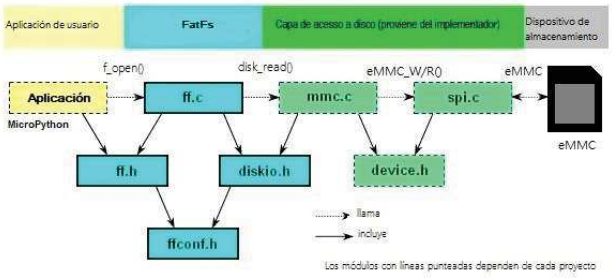

Figura 3. Capas de software utilizadas del proyecto FatFs con sus archivos más representativos [5].

Adaptación del driver de ramdisk como MSD

Por último, se tomó como punto de partida los drivers para la interface USB 2.0, en dicho set de drives se provee una funcionalidad de habilitar la interfaz de USB como un disco de RAM. Dicha función inicializa el controlador de USB y lo configura como un dispositivo de almacenamiento masivo como se especifica en los estándares [10] [11] [12], el cual obedece a cierto set de comandos SCSI [13] y cuya memoria de operación es una sección de la RAM interna del controlador. Entonces se modificó el driver para hacer trasferencias del contenido de RAM a la memoria flash eMMC, obteniendo así un dispositivo de almacenamiento masivo por USB. Esto se ilustra en la figura 4, donde las flechas rojas indican las interacciones entre los componentes de hardware.

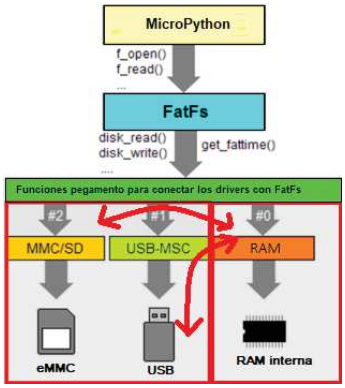

Figura 4. Vista simplificada de la arquitectura de software utilizada [5].

Resultados

El DUT se conecta con éxito a una PC con Windows (figura 5) y que al ejecutar un comando en MicroPython el R-CarGen3 se configura e inicializa como un MSD USB para que el usuario pueda crear, editar, borrar o copiar y pegar archivos en la memoria flash. En nuestras pruebas, Windows le asigno el volumen (E:) como unidad de disco externa de 1MB (figuras 6 y 7).

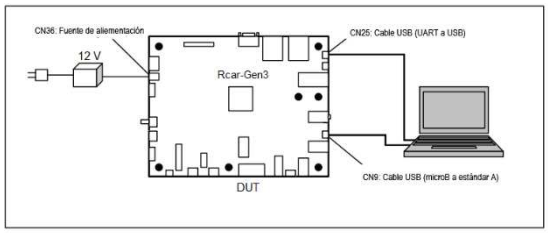

Figura 6. Conexión de hardware en el DUT [8].

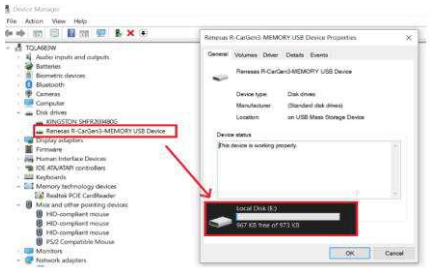

Figura 5. Windows reconoce al DUT como un MSD.

Adicionalmente se puede interactuar con las funciones de manejo de archivos desde el CLI (figura 8). Si el usuario lo decide puede guardar un script con el nombre de boot.py y después de un reset el sistema puede ejecutar dicho script justo después de arrancar (figura 9).

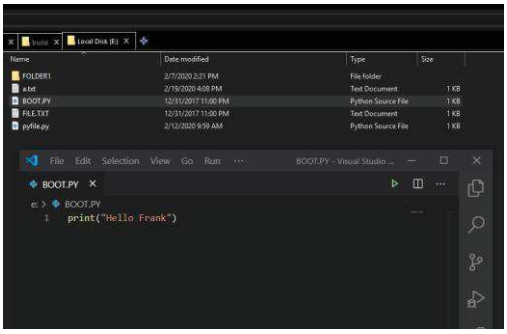

Figura 7. Vista del explorador de archivos.

```
>>> boot=open("boot.py","r")
invoking stat
>>> boot.read()
invoking stat
[DBG]:disk_read()
reading 1 sectors at pos 102 to drive 0
invoking stat
'print("Hello Frank")'
>>> boot.close()
invoking stat
invoking stat
>>>
```

Figura 8. Abriendo, leyendo y cerrando el archivo boot.py desde el CLI

```
*** Mounting filesystem ***
[DBG]:disk_read_emmc_direct()
reading 2048 sectors at pos 0 to drive 0
invoking disk_initialize
invoking stat
[DBG]:disk_read()
reading 1 sectors at pos 0 to drive 0
[DBG]:disk_read()
reading 1 sectors at pos 63 to drive 0
- Mounted!
*** Opening file ***
invoking stat
[DBG]:disk_read()
reading 1 sectors at pos 70 to drive 0
invoking stat
- File opened
=== File execution started ===
invoking stat
[DBG]:disk_read()
reading 1 sectors at pos 102 to drive 0
invoking stat
Hello Frank
invoking stat
invoking stat
=== File execution done ===
MicroPython v1.9.3 on 2020-07-29; EvalBoard with GC2019 M3
>>>
```

Figura 9. Ejemplo del sistema arrancando con el script boot.py

Comentarios finales

En la hipótesis se planteó que se podría implementar un sistema de software basado en Micropython para obtener los beneficios de un sistema de almacenamiento masivo por USB y poder almacenar en memoria flash y bootear los scripts de prueba de hardware en las unidades (ECU's) basadas en R-CarGen3, lo cual se cumplió de manera satisfactoria.

Este proyecto entonces constituye una herramienta para poder crear rutinas de prueba que son fáciles de modificar y reflashear lo que permite más control, flexibilidad e independencia de trabajo al ingeniero de hardware encargado de la validación eléctrica. Entonces se prevé por tanto una mejora en tiempo y efectividad para el proceso de validación eléctrica y por consiguiente una mejora en costos. La solución propuesta también puede ser útil para otras empresas de la industria automotriz y de diseño de productos electrónicos (incluso con microntroladores más sencillos) que pretendan resolver el mismo problema de validación tardía de hardware.

Una limitante para la reutilización de este proyecto en otros microcontroladores dependerá de la disponibilidad de las interfaces utilizadas (USB y eMMC) así como sus drivers y/o BSP. Otra limitante es que el sistema solo puede correr un hilo de ejecución a la vez, sin embargo, se está trabajando para que la siguiente versión incluya la nueva funcionalidad multithreading de MicroPython para poder crear mayor tráfico de datos y poder ejecutar pruebas de emisión eléctrica.

Referencias

[1] BusinessWire, «Renesas Advanced Automotive R-Car M3 Adopted by Continental for Its Body High-Performance Computer,» 7 July 2020. [En línea]. Available: https://www.businesswire.com/news/home/20200707005241/en/. [Último acceso: 2 August 2020].

[2] ID RD EEN, «EMK Version 1.15 © Continental AG,» 2019.

[3] Andreas Koch, «Bring up software: New approach with MicroPython,» Continental AG, 2019.

[4] Renesas Electronics Corporation, «R-Car,» 3 08 2020. [En línea]. Available: https://www.renesas.com/us/en/products/automotive/automotive-lsis/r-car.html.

[5] D. P. George, 2018. [En línea]. Available: http://micropython.org/. [Último acceso: 25 July 2020].

[6] ChaN, «FatFs - Generic FAT Filesystem Module,» 04 12 2019. [En línea]. Available: http://elm-chan.org/fsw/ff/00index_e.html. [Último acceso: 2 August 2020].

[7] Renesas Electronics, R-Car Series, 3rd Generation. User's Manual: Hardware., 2019.

[8] STMicroelectronics, «Developing applications on STM32Cube™ with FatFs,» 2019.

[9] P. S. a. c. Damien P. George, «MicroPython documentation,» 04 12 2020. [En línea]. Available: http://docs.micropython.org/en/latest/. [Último acceso: 25 July 2020].

[10] JEDEC Solid State Technology Association, Embedded Multi-Media Card (eMMC) Electrical Standard (5.1), 2015.

[11] Renesas Electronics, «R-CarM3-SiP System Evaluation Board Hardware Manual,» 2016.

[12] USB Implementers Forum, Inc., Universal Serial Bus Mass Storage Specification For Bootability, Revision 1.0 ed., 2004.

[13] USB Implementers Forum, Inc., Universal Serial Bus Mass Storage Class Bulk-Only Transport, Revision 1.0 ed., 1999.

[14] USB Implementers Forum, Inc., Universal Serial Bus Mass Storage Class, Revision 1.4 ed., 2010.

[15] Seagate Technology LLC., SCSI Commands, 100293068, Rev. J ed., 2016.