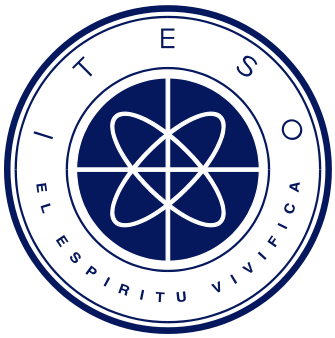
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS



CONTROLADOR AUTÓMATA DESDE PROGRAMACIÓN VISUAL POR BLOQUES

Tesina para obtener el grado de:

ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presentan: Ing. César Luis García Velasco
Ing. Ernesto Gabriel Ventura Roldán

Director: Mtro. Luis Enrique Garabito Siordia

San Pedro Tlaquepaque, Jalisco. Julio de 2018.

Agradecemos a esta gran institución educativa: ITESO, por brindar este programa de estudios con excelentes maestros y asesores; y al CONACYT, por el apoyo brindado a través de esta especialización.

Queremos extender el agradecimiento a nuestra familia por todo el apoyo que nos otorgaron; a compañeros de clase y maestros con los que convivimos a lo largo de la especialidad y que, gracias a ellos, adquirimos las habilidades y conocimientos; a las empresas donde laboramos, por interesarse en que logremos nuestras metas y nuestro crecimiento profesional y personal.

César Luis García Velasco, Número de Becario: 861838
Ernesto Gabriel Ventura Roldán, Número de Becario: 859782

Abstract

The way of interacting among various devices, communications, and processes has been changing in recent years by accelerating and diversifying the phenomenon known as Industry 4.0, which is the leading technology in smart factories. For this reason, this project seeks to create and consolidate an embedded system that improves and facilitates the development experience of these new technologies, considering the multiplatform of operating systems, communication robustness, and the simplicity of the development process, and at the same time enriching the current ecosystem of embedded systems. In the first stage, an interpreter was designed with the Blockly platform. The modified Blockly distribution generates a script that is read and executed by the development platform. In the second stage, the interpretation of this script is performed in MQX 4.1 for the FRDM K64F development card of NXP. The interpreter allows the reading of all blocks that were made in the modified Blockly distribution and transforms them into a functional code for the microcontroller of the development board, thus, achieving an integrated functionality. Considering this development, the foundations for a platform that will allow the integration of Industry 4.0 and the compliance with current market demands were created. The benefits of our development include providing an easier way of programming with visual codes (blocks), integrating this program into the FDRM K64F development board, and in future scenarios, Big data analytics.

Resumen

La forma de interactuar entre distintos dispositivos, comunicaciones y procesos ha ido cambiando en los últimos años, acelerando y diversificando el fenómeno conocido como Industria 4.0, que es la tecnología líder en las fábricas inteligentes. Por esta razón, este proyecto busca crear y consolidar un sistema embebido que mejore y facilite la experiencia de desarrollo de estas nuevas tecnologías, teniendo en cuenta la multiplataforma en sistemas operativos, la robustez de las comunicaciones y la simplicidad del proceso de desarrollo, a la vez que enriquece el ecosistema actual de sistemas embebidos. En la primera etapa, se diseñó un intérprete con la plataforma Blockly. La distribución modificada de Blockly genera un script que es leído y ejecutado por la tarjeta de desarrollo. En la segunda etapa, la interpretación de este script se realiza en MQX 4.1 para la tarjeta de desarrollo FRDM K64F de NXP. El intérprete permite la lectura de todos los bloques que se realizaron en la distribución Blockly modificada y los transforma en un código funcional para el microcontrolador de la tarjeta de desarrollo, logrando así una funcionalidad integrada. Considerando este desarrollo, se crearon las bases para una plataforma que permitirá la integración de Industria 4.0 y el cumplimiento de las demandas actuales del mercado. Los beneficios de nuestro desarrollo incluyen proporcionar una forma más fácil de programación con códigos visuales (bloques), integrando este programa en la tarjeta de desarrollo FDRM K64F, y en escenarios futuros, análisis de datos.

Lista de figuras

Abreviaturas y acrónimos

API Interfaz de Programación de Aplicaciones (del inglés, Application Program Interface)

E/S Entradas y salidas

GUI Interfaz Gráfica de Usuario (del inglés, Graphical User Interface)

IDE Entorno de Desarrollo Integrado (del inglés, Integrated Development Environmet)

LED Diodo emisor de luz (del inglés, light-emitting diode)

MCU Microcontrolador (del inglés, Microcontroller)

MQX Cola de mensajes ejecutables (del inglés, Message Queue eXecutive)

MSD Dispositivo de almacenamiento masivo (del inglés, mass storage device)

NXP Nueva Experiencia (del inglés, Next eXPerience)

RAM Memoria de Acceso Aleatorio (del inglés, Random Access Memory)

RGB Rojo, Verde y Azul (del inglés, Red, Green and Blue)

RTOS Sistema Operativo de Tiempo Real (del inglés, Real Time Operating System)

SO Sistema Operativo

USB Bus Universal en Serie (del inglés, Universal Serial Bus)

VPL Lenguaje de Programación Visual (del inglés, visual programming leguaje)

Contenido

Introducción

La industria de la manufactura está en constante crecimiento incluyendo nuevas tecnologías y/o nuevas formas de darle solución a una misma problemática. Esta industria, al ser tan especializada, tiene un alto costo en el desarrollo de nuevos productos; además, para cambiar algo, por más simple que sea, se necesita de la ayuda de un experto, que conlleva a tener un gasto y, por ende, el costo general del desarrollo incrementa.

Por ello, nos dimos a la tarea de desarrollar un proyecto que cumpla con las demandas de la industria de la manufactura, con un software fácil de manipular y modificar, un hardware sencillo de actualizar, y así evitar la figura del experto para realizar estas tareas.

Este proceso engloba una serie de herramientas, y entre ellas una tarjeta de desarrollo, que es una tarjeta de circuito impreso que contiene un microcontrolador y la lógica de soporte mínima necesaria para que un ingeniero se familiarice con el microcontrolador de la tarjeta y aprenda a programarlo. También sirve a los usuarios como método para crear prototipos de aplicaciones en productos. A diferencia de un sistema de propósito general, como una computadora en casa, usualmente una tarjeta de desarrollo contiene poco o ningún hardware dedicado a una interfaz de usuario.

La tarjeta de desarrollo será programada utilizando un sistema operativo de tiempo real, mejor conocido como RTOS (Real Time Operating System, por sus siglas en inglés), en donde la precisión total de una operación depende no sólo de su precisión lógica, sino también del tiempo en que se realiza. Los sistemas en tiempo real, así como sus plazos, se clasifican según la consecuencia del incumplimiento de un plazo. El principal deber de un sistema en tiempo real es producir resultados correctos cumpliendo con los plazos predefinidos y generándolos. En la forma más simple, los sistemas en tiempo real pueden definirse como aquellos sistemas que responden a eventos externos de manera oportuna [1].

También necesitamos una Interfaz de Programación de Aplicaciones, por sus siglas en inglés API (Application Programming Interface), que es un conjunto de procedimientos que cumplen una o muchas funciones con el fin de ser utilizadas por otro software. En términos de programación, es una capa de abstracción [2].

Nuestra propuesta es crear un sistema basado en la primicia de la sencillez de uso. Esto refiere que una persona, que no necesariamente sea un experto en el tema, sea capaz de crear y/o modificar partes fundamentales del sistema creado; por ejemplo, modificar la velocidad de una banda de transporte, la temperatura de una cámara de enfriado, la activación de motores de diversos tipos, etcétera.

Esta solución consta de dos partes fundamentales: la manipulación de una tarjeta de desarrollo que satisfaga nuestros fines y la creación de un sistema visual de programación.

La tarjeta de desarrollo a utilizar será una tarjeta de desarrollo llamada FRDM K64F de NXP, que nos brinda la posibilidad de utilizar un RTOS, específicamente MQX 4.1 que nos ayuda a crear y controlar distintas tareas previamente definidas; también incluye ciertas herramientas que controlan estas tareas en ciertos casos de aplicación.

La parte de la programación visual se realizará por bloques. Utilizaremos la API Blockly de Google que nos provee las herramientas necesarias para crear los bloques necesarios para nuestro desarrollo. Esta API permite al usuario mover y editar bloques previamente definidos y no tendrá que escribir ni una sola instrucción de programación; al terminar de hacer los cambios, la API generará un archivo que se cargará en la tarjeta de desarrollo y se interpretará para que dichos cambios tomen su debida acción.

1. Antecedentes

El proyecto de tesis está basado en herramientas de software que han sido optimizadas para poder combinarse con hardware de aplicación específica. El software base que se usó para crear estas nuevas herramientas es un antecedente importante, y es primordial para entender el contexto del desarrollo tecnológico.

1.1. La programación visual

Algunos softwares que soportan entornos de programación visuales son Actum Realizer, AgilArt, FlowCode, etcétera. El Actum Realizer utiliza un diagrama de flujo lógico y lo convierte en un código C optimizado basado en la tarjeta de destino seleccionada. AgilArt utiliza un diseñador visual basado en web que soporta la programación de controladores ARM. Maneja un firmware específico llamado AgilArt Run-Time en el hardware de destino para subir el código al hardware de destino. FlowCode es otro software en el mercado que viene como un paquete completo para diseños electromecánicos; también utiliza un diseño basado en diagramas de flujo para generar el código fuente [3].

Un trabajo relacionado con el uso de la programación visual y su ejecución en una tarjeta de desarrollo es el de M. Mahesh y P. Sivraj con el proyecto "DrawCode: Visual tool for programming microcontrollers" en el que crean un entorno de desarrollo integrado impulsado por modelos (MD-IDE) para el microcontrolador MPS430g2231 de Texas Instruments, que puede facilitar el trabajo de los desarrolladores que utilizan este microcontrolador. El trabajo tiene un entorno de desarrollo integrado, IDE (Integrated Development Environmet por sus siglas en inglés), donde el usuario puede simplemente utilizar varios bloques predefinidos, cada uno de los cuales representa un periférico o una función del microcontrolador [3].

Otro trabajo relacionado con la programación visual, e incluso específicamente con la API de Google Blockly, es el de Adin Baskoro Pratomo y Riza Satria Perdana con el proyecto llamado "Arduviz, a visual programming IDE for arduino" [4], quienes se dieron a la tarea de crear una programación visual por bloques creada en Google Blockly e implementada en una tarjeta de desarrollo Arduino. Patromo y Perdana se basan en dos herramientas de programación visual,

Ardublock y miniBloq, que están pensadas específicamente para la plataforma Arduino, argumentando que cada una tiene sus fortalezas, pero al ser desarrollos diferentes no se tiene manera de crear un programa entre ambas herramientas. Ahí es donde nace Arduviz, que tiene la mayoría de las ventajas de Ardublock y miniBloq como la generación instantánea de código y el entorno de desarrollo autónomo [4].

1.2. Interpretes para microcontroladores

El trabajo de tesis realizado por el Mtro. Luis Garabito Siordia, llamado "Simplified Development Tool for Embedded Systems", describe una herramienta creada sobre MQX que permite la adición dinámica de dispositivos de entrada/salida para hacer una herramienta flexible y da la opción de añadir nuevos dispositivos en cada arranque del sistema. La herramienta incluye un intérprete de scripts (secuencias de comandos) en tiempo real. El script a interpretar se lee al iniciar el sistema. Después de la preparación, el intérprete ejecuta el script que es la aplicación del usuario [5].

Cada una de las herramientas antes descritas dieron la pauta para poder realizar sistemas complejos, mismos que han logrado crecer hasta el punto de poder implementarlos en una tarjeta de desarrollo.

2. Marco teórico

A continuación, se describe cada una de las partes más importantes que ayudarán a entender mejor el contexto del que estamos hablando en nuestro proyecto de tesis. Cada definición y/o descripción está relacionada en su totalidad con la investigación y desarrollo de este proyecto.

2.1. Sistemas embebidos

Un sistema embebido es un sistema electrónico diseñado para realizar funciones en tiempo real. Al contrario de lo que ocurre con las computadoras, los sistemas embebidos se diseñan para cubrir necesidades específicas.

En un sistema embebido la mayoría de los componentes se encuentran incluidos en la placa base (la tarjeta de video, audio, módem). Algunos ejemplos de sistemas embebidos podrían ser dispositivos como un taxímetro, un sistema de control de acceso, la electrónica que controla una máquina expendedora o el sistema de control de una fotocopiadora. Hay cuatro elementos estructurales principales:

- Procesador: Controla la operación de la computadora y realiza sus funciones de procesamiento de datos. Cuando sólo hay un procesador, a menudo se le conoce como la unidad central de procesamiento (CPU) [8].
- Memoria principal: Almacena datos y programas. Esta memoria suele ser volátil, es decir, cuando se apaga el equipo se pierde el contenido de la memoria. Por el contrario, el contenido de la memoria del disco se conserva, incluso cuando el sistema informático está cerrado [8].
- Módulos de E/S: Mueve los datos entre el ordenador y su entorno externo. El entorno externo consiste en una variedad de dispositivos, incluyendo dispositivos de memoria secundarios (por ejemplo, discos), equipos de comunicaciones y terminales [8].
- Bus de sistema: Permite la comunicación entre procesadores, memoria principal y módulos de E/S [8].

2.2. Sistemas Operativos

Un sistema operativo (SO) explota los recursos de hardware de uno o más procesadores para proporcionar un conjunto de servicios a los usuarios del sistema. El sistema operativo también gestiona la memoria secundaria y los dispositivos de E/S (entrada/salida) en nombre de sus usuarios [6]. Las funciones clásicas de un sistema operativo se pueden agrupar en las tres categorías siguientes:

- Gestión de los recursos de la computadora.
- Ejecución de servicios para los programas.
- Ejecución de los mandatos de los usuarios.

"Con respecto a su faceta de gestor de recursos, hay que tener en cuenta que en una computadora actual suelen coexistir varios programas, del mismo o de varios usuarios, ejecutándose simultáneamente. Estos programas compiten por los recursos de la computadora, siendo el sistema operativo el encargado de arbitrar su asignación y uso. Como complemento a la gestión de recursos, el sistema operativo ha de garantizar la protección de unos programas frente a otros y ha de suministrar información sobre el uso que se hace de los recursos" [9].

El sistema operativo como máquina extendida ofrece a los programas un conjunto de servicios y llamadas al sistema que pueden solicitar cuando lo necesiten, proporcionando a los programas de visión de máquina extendida [9].

Por último, el sistema operativo es un elemento que proporciona la interfaz de usuario del sistema. El módulo del sistema operativo que permite que los usuarios dialoguen de forma interactiva con el sistema es el intérprete de mandatos o Shell [9].

2.3. Sistemas operativos en tiempo real

La computación en tiempo real es donde la exactitud del sistema no sólo depende de la exactitud del resultado lógico, sino también del tiempo de entrega del resultado. Por lo tanto, el sistema operativo debe tener características que soporten este requisito crítico para que sea llamado Sistema Operativo en Tiempo Real. El RTOS debe tener un comportamiento predecible ante eventos externos impredecibles. Un buen RTOS es aquel que tiene un comportamiento limitado (predecible) bajo cualquier escenario de carga del sistema, es decir, incluso bajo interrupciones

simultáneas y ejecución de subprocesos. Un verdadero RTOS será determinístico bajo cualquier condición. Estos sistemas operativos ocupan poco espacio, desde 10 KB a 100KB, en comparación con los sistemas operativos generales que requieren cientos de megabytes. Los sistemas en tiempo real en los que el incumplimiento de un plazo es catastrófico se llaman RTOS críticos. Si los sistemas permiten que a veces no se cumplan los plazos y todavía pueden recuperarse, son llamados RTOS no críticos [9].

2.3.1 RTOS críticos

En los RTOS críticos, si no se cumplen las restricciones de tiempo se produce un fallo del sistema. Estos sistemas se utilizan cuando es imperativo que un evento sea procesado dentro de un plazo estricto. Estas fuertes garantías son necesarias en sistemas para los que no reaccionar en un cierto intervalo de tiempo afectaría al sistema y causaría grandes pérdidas. Ejemplos de sistemas en tiempo real incluyen control de tráfico, control industrial, robótica, aviónica, etcétera [10].

2.3.2 RTOS no críticos

Un RTOS no critico es en el que los plazos se cumplen en su mayoría. Sólo se definen la precedencia y la secuencia de las operaciones de las tareas, las latencias de interrupción y las latencias de cambio de contexto son pequeñas, pero puede haber pocas desviaciones entre las latencias esperadas de las tareas y las limitaciones de tiempo observadas y se aceptan algunos errores de plazo límite [11].

2.4. Lenguaje de Programación Visual

El lenguaje de programación visual, por sus siglas en inglés VPL (visual programming leguaje), permite el desarrollo de programas de software eliminando el código textual del software con una serie de elementos gráficos visuales. VPL incorpora estos elementos gráficos como contexto primario del lenguaje ordenado de forma sistemática. Los gráficos o iconos incluidos en un programa visual sirven como entrada, actividades, conexiones y/o salida del programa.

El lenguaje visual tiene algunos tipos, tales como lenguajes basados en iconos, lenguajes de diagramación y lenguajes basados en formularios. Los lenguajes visuales no deben confundirse con el lenguaje de programación basado en interfaz gráfica de usuario, por sus siglas en inglés GUI (graphical user interface), ya que sólo proporcionan servicios de creación de programas gráficos. Sin embargo, su código/contexto es completamente textual [12].

2.4.1 Blockly

Google proporcionó un entorno de codificación visual gratuito, llamado Blockly, para el desarrollo de aplicaciones. La librería de Blockly permite a los usuarios editar sus programas usando interbloqueo, piezas gráficas o bloques. En este entorno, las construcciones de programación común como variables, expresiones aritméticas y lógicas, y bucles, se representan en forma de bloques. Una de las ventajas de Blockly es que libera a los novatos la carga de memorizar la sintaxis [13].

Pero Blockly no proporciona un vocabulario completo de bloques o un entorno de tiempo de ejecución. Los desarrolladores necesitan integrar Blockly con alguna forma de salida, construir su vocabulario y decidir cómo se ejecutará el código generado [14].

2.5. NXP

NXP Semiconductors N.V. es un fabricante global holandés de semiconductores con sede en Eindhoven, Países Bajos. La compañía fue fundada en 1953 como parte de la firma electrónica Philips, con fabricación y desarrollo en Nijmegen, Países Bajos. Conocida entonces como Philips Semiconductors, la empresa fue vendida a un consorcio de inversores de capital privado en 2006, momento en el que el nombre de la empresa cambió a NXP [15].

NXP Semiconductors proporciona una señal mixta y productos estándar basados en su experiencia en seguridad, identificación, automoción, redes, radiofrecuencia, señales analógicas y gestión de energía. Los productos de la empresa se utilizan en aplicaciones automotrices, de identificación, de infraestructura alámbrica e inalámbrica, de iluminación, industriales, de consumo, móviles e informáticas. NXP Semiconductors es el creador de la tarjeta de desarrollo FRDM K64F que se describe a continuación [15].

2.5.1 FRDM K64F tarjeta de desarrollo

La plataforma de desarrollo NXP Freedom es un conjunto de herramientas de software y hardware para evaluación y desarrollo. Es ideal para la creación rápida de prototipos de aplicaciones basadas en microcontroladores. El diseño de hardware NXP Freedom K64 incluye un microcontrolador de la serie Kinetis K, construido sobre el núcleo ARM® Cortex®-M4 [16].

El FRDM-K64F se puede utilizar para evaluar los dispositivos de las series K64, K63 y K24 de Kinetis K. Cuenta con la MCU MK64FN1M0VLLL12, que tiene una frecuencia máxima de operación de 120 MHz, 1 MB de flash, 256 KB de RAM, un controlador USB de velocidad completa, controlador Ethernet, controlador host digital seguro y periféricos analógicos y digitales [16].

Las características del hardware FRDM-K64F son las siguientes:

- MK64FN1M0VLLL12 MCU (120 MHz, 1 MB de memoria flash, 256 KB de RAM, bajo consumo, USB sin cristal y 100 LQFP).
- Interfaz USB de doble función con conector USB micro-B.
- RGB LED (diodo emisor de luz, por sus siglas en inglés).
- FXOS8700CQ - acelerómetro y magnetómetro.
- Dos botones pulsadores de usuario.
- Opción de fuente de alimentación flexible - OpenSDAv2 USB, K64 USB y fuente externa.
- Fácil acceso a la entrada/salida de MCU a través de conectores de E/S.
- Circuito de depuración OpenSDAv2 programable que soporta el software de interfaz CMSIS-DAP que proporciona:
    - Interfaz de programación del flash del dispositivo de almacenamiento masivo (MSD).
    - Interfaz de depuración CMSIS-DAP a través de una conexión USB HID sin controlador que proporciona depuración de control de ejecución y compatibilidad con herramientas IDE.
    - Interfaz de puerto serie virtual.
    - Proyecto de software CMSIS-DAP de código abierto: github.com/mbedmicro/CMSIS-DAP.

- Ethernet.
- SDHC.

- Módulo RF adicional: nRF24L01+ Nordic 2.4GHz Radio.
- Módulo Bluetooth adicional: Placa JY-MCU BT V1.05 BT [16].

La figura 2-1 muestra el diagrama de bloques del diseño del FRDM-K64F. Los componentes primarios y su colocación en el conjunto de herrajes se explican en la Figura 2-2.

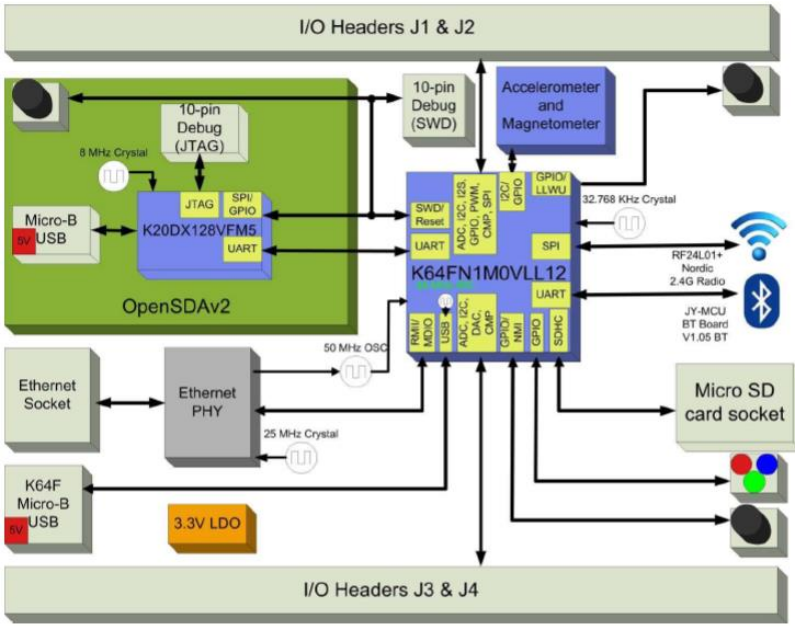


Fig. 2-1 Diagrama de bloques FRDM-K64F.

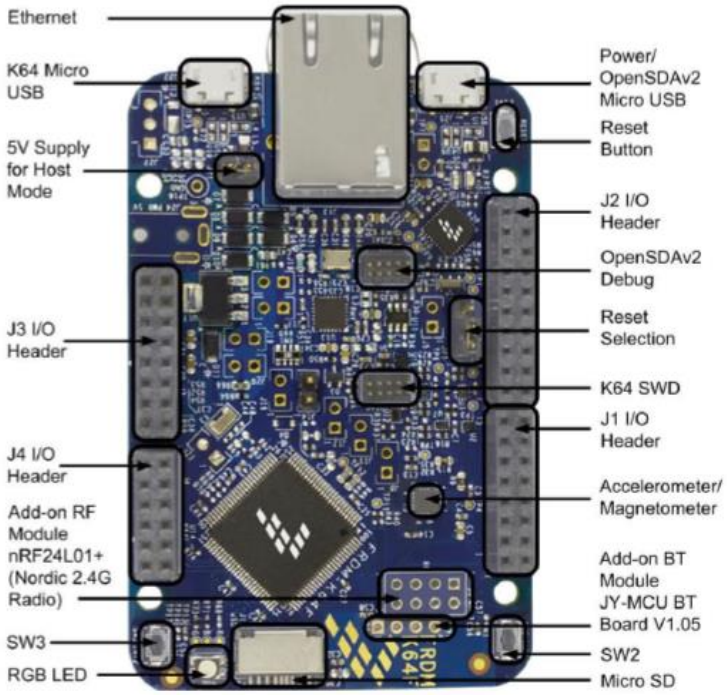


Fig. 2-2 Colocación de los principales componentes del FRDM-K64F.

3. Metodología

El correcto diseño de nuestra distribución de Blockly es la pieza más importante de nuestro proyecto, ya que, sin esta, o sin su óptimo funcionamiento, todo el desarrollo no se desempeñaría óptimamente.

Nuestra distribución tiene secciones de bloques generales que ya están incluidos en Blockly por defecto y una sección creada específicamente para nuestro proyecto. La sección llamada "Custom", como se ve en la figura 3-1 tiene dos bloques creados específicamente por nosotros, que explicamos en el orden descendente en que se encuentran y además agregamos la interpretación de los bloques IF y Repeat que están dentro de la sección LOGIC y LOOPS, respectivamente, y el soporte para la creación y asignación de variables que se encuentra en la sección VARIABLES, esto lo podemos ver en la figura 3-2. Aunado a esto también habilitamos el segundo bloque de la sección MATH, que nos permite hacer suma, resta y multiplicación y divisiones.

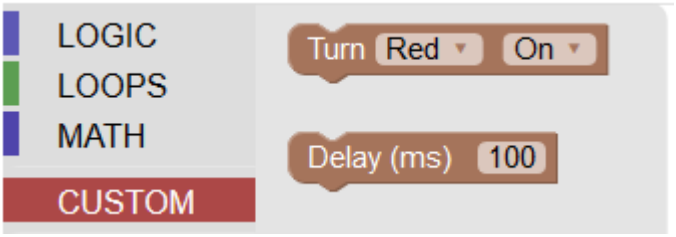


Fig. 3-1 Secciones de nuestra distribución de Blockly

El primer bloque de la sección CUSTOM está destinado a cambiar el estado de un LED RGB el cual tiene colores, rojo, verde y azul (red, green y blue, en su traducción al inglés). Este bloque tiene definidos estos tres colores y también el estado en que desean poner, On para encendido y Off para apagado. Este bloque actúa directamente sobre el LED RGB que contiene la tarjeta de desarrollo.

Fig. 3-2 Bloques IF y Repeat

Al presionar sobre la casilla del color se despliega un menú que nos brinda la opción de seleccionar el color que deseamos para el bloque (figura 3-3) y en la casilla del estado observamos que sucede algo similar. La diferencia es que nos da opción de elegir en qué estado queremos dicho color (figura 3-4).

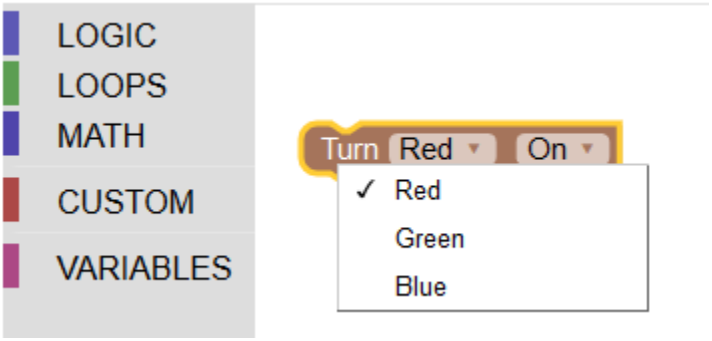


Fig. 3-3 Selección de color

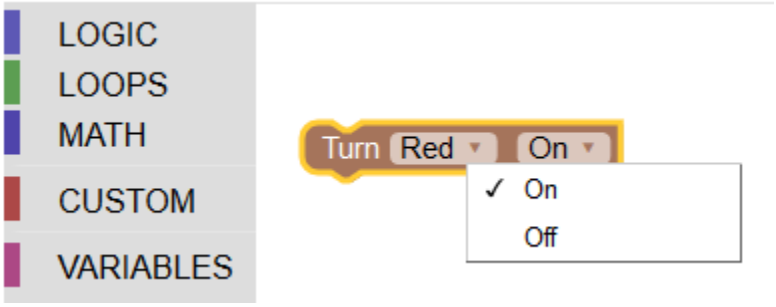


Fig. 3-4 Selección del estado del color seleccionado

El segundo bloque tiene la tarea de hacer una espera o pausa en el sistema y esta predefinida en milisegundos, es decir, si nosotros configuramos el bloque con el valor 1000, nos daría una pausa en el sistema de 1 segundo, esto haría que el microcontrolador de la tarjeta de desarrollo deje de ejecutar cualquier otra instrucción por el tiempo definido en el bloque. Como se ve en la figura 3-5 el bloque únicamente admite escribir un valor numérico en el espacio de la derecha, en esta figura tiene un valor de 100.

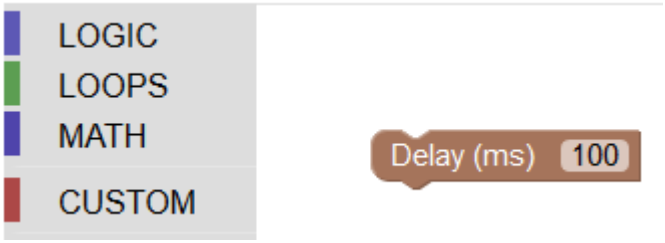


Fig. 3-5 Bloque de espera

En la sección LOGIC tenemos el bloque IF, el cual es una condicional que nos permite tomar cierta decisión al interior de nuestro algoritmo, es decir, nos facilita determinar qué acciones tomar dada o no cierta condición. Su utilización consta de dos partes, el bloque IF en sí y la condicionante a ser evaluada, estas condicionantes están predefinidas en el segundo bloque de esta sección y nos sirve para añadir las variables creadas o los valores numéricos que queramos agregar. En la figura 3-6 se muestra la adición de dicho bloque al IF y una variable creada comparada con un valor numérico. El bloque adicional nos permite cambiar la condicionante entre mayor, menor, igual y diferente; estas condiciones se evalúan entre la variable y el valor numérico que se haya colocado dentro de los espacios que tiene. La colocación de variables, además de las diferentes condicionales, las podemos observar en la figura 3-7.

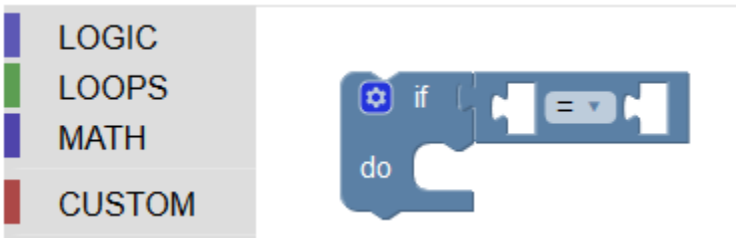


Fig. 3-6 Adición de bloque para comparación de condicional

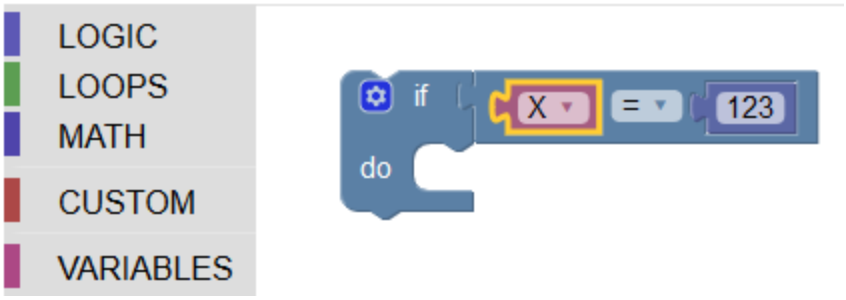


Fig. 3-7 Bloque con valores numéricos y muestra de condicionales

En la sección LOOPS podemos encontrar el bloque llamado Repeat, que nos permite realizar una iteración hasta que la condicional se cumpla. Esto se hace seleccionando la opción de While dentro del menú desplegable que contiene el bloque; hasta este momento únicamente está habilitada esta función, las demás no funcionarían. En este bloque se agrega una condicional de la misma manera que en el bloque IF, y todo el algoritmo que se encuentre dentro de dicho bloque se repetirá hasta que la condicional colocada se cumpla. En la figura 3-8 podemos ver un ejemplo de su uso.

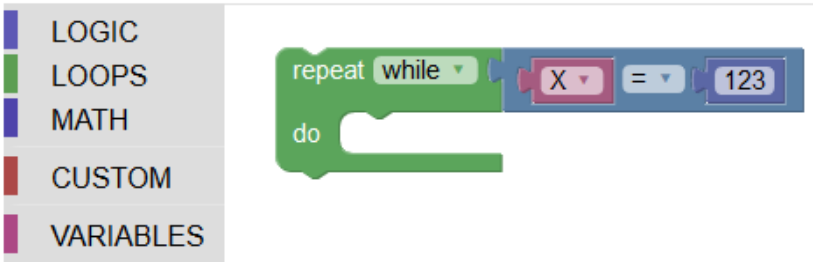


Fig. 3-8 Ejemplo de uso del bloque Repeat

En la sección VARIABLES tenemos la opción de crear una variable. Una variable es un espacio donde el programador asigna un valor, ya sea numérico o de carácter; se hace referencia a esta variable por medio del nombre que se le haya asignado. Para crear una variable basta con pulsar sobre la sección, al abrirse el menú, presionamos sobre el botón “Crear una variable” (del inglés Create a variable) y en la ventana que se despliega asignaremos el nombre que deseemos darle, esto se muestra en la figura 3-9. Una vez creada la variable aparecerá como un bloque dentro de la sección y ahí podremos asignarle un valor. Para hacer la asignación debemos utilizar el bloque llamado set y agregarle un bloque que contenga el valor numérico o carácter deseado, esto se muestra en la figura 3-10; y para hacer uso de esta variable basta con anexar el tercer bloque de esta sección en algún otro bloque de los que se han descrito anteriormente.

Fig. 3-9 Creación de variables

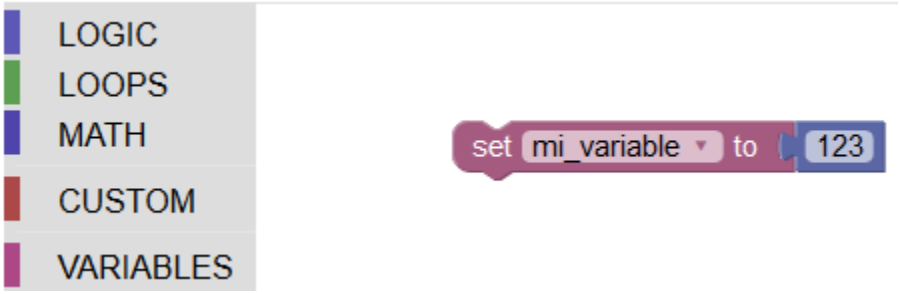


Fig. 3-10 Modificación del valor de la variable

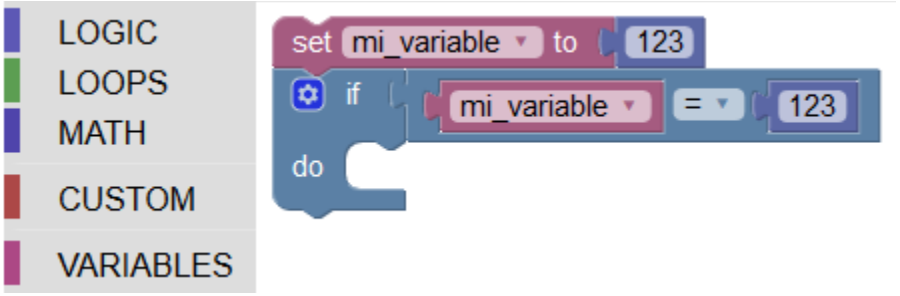


Fig. 3-11 Ejemplo de uso de una variable creada

En el segundo bloque de la sección MATH podemos realizar sumas, restas, multiplicaciones y divisiones entre una variable y un valor numérico, que a su vez será almacenada en una variable, que puede ser la misma que se está utilizando dentro del bloque o alguna otra. Su uso es sencillo, al pulsar en el menú se desplegarán cada una de las operaciones que este bloque nos brinda. En la figura 3-12 podemos ver un ejemplo de su uso.

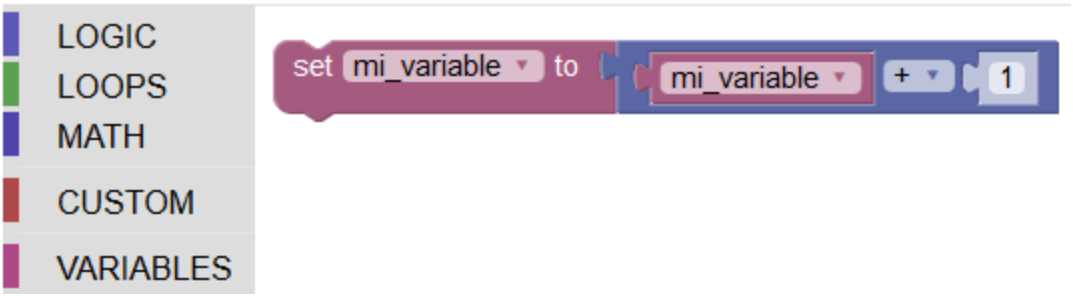


Fig. 3-12 Ejemplo de uso para el bloque de operaciones numéricas

El script que se genera con nuestra distribución de Blockly, es interpretado por un código realizado en MQX, este se encarga de leer una a una las instrucciones textuales del programa conforme estas necesitan ser ejecutadas y convertirlas en comandos entendibles por el microcontrolador de la tarjeta de desarrollo. Algunos ejemplos de lenguajes interpretados son Python, JavaScript y Ruby.

En la figura 3-13 podemos ver el diagrama de flujo que sigue el código para realizar la interpretación, y el código fuente de MQX lo podemos encontrar en el apéndice A.

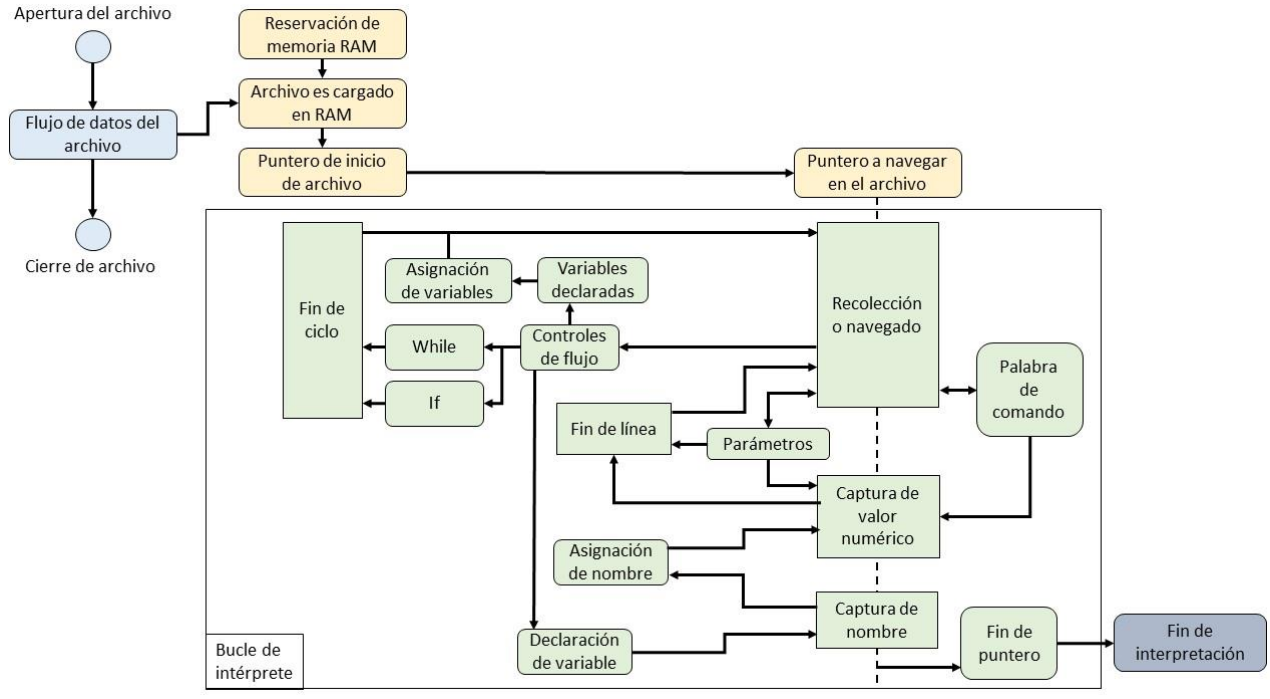


Fig. 3-13 Diagrama de flujo del código en MQX.

4. Resultados

Para demostrar los resultados de nuestra aplicación, implementamos una serie de impresiones en pantalla (consola) que corroboran que el código creado en MQX es ejecutado, además de mostrar los pasos de su ejecución.

Comenzando con la ejecución de la función “Delay”, que se muestra en nuestra distribución de Blockly en la Figura 4-1 podemos observar que la función es de 100 ms, la cual se agregará al archivo descargable a una memoria micro SD que se instala en la tarjeta de desarrollo.

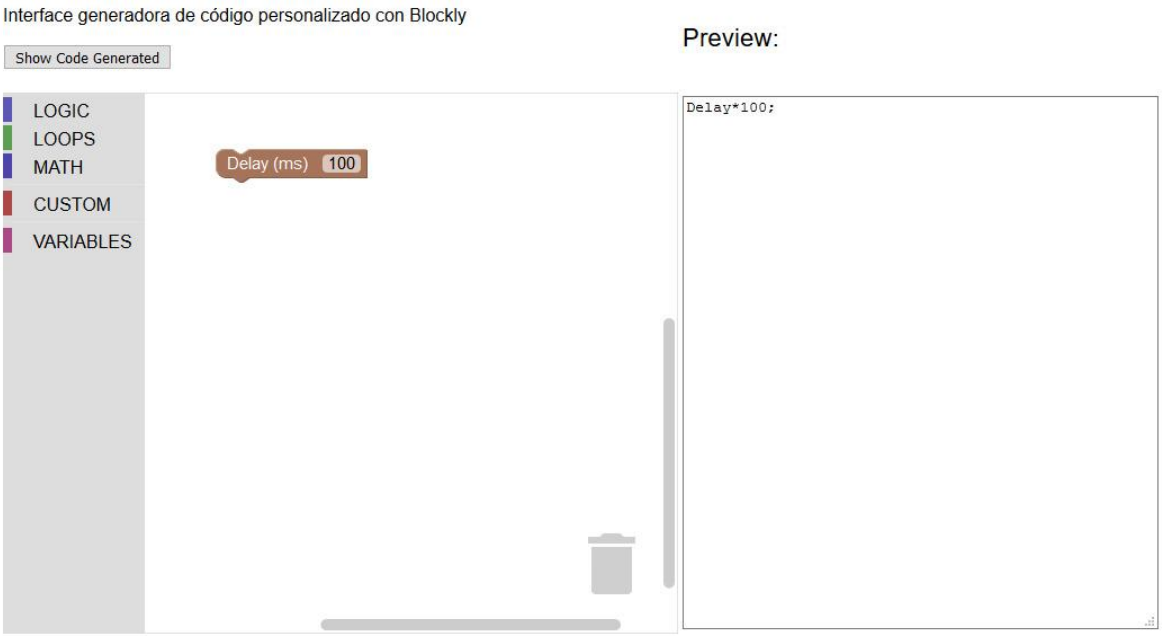


Fig. 4-1 Función generada con Blockly y su representación en texto.

En la Figura 4-2 vemos el resultado de la codificación, que de la misma manera estará en el archivo que será interpretado por la tarjeta de desarrollo.

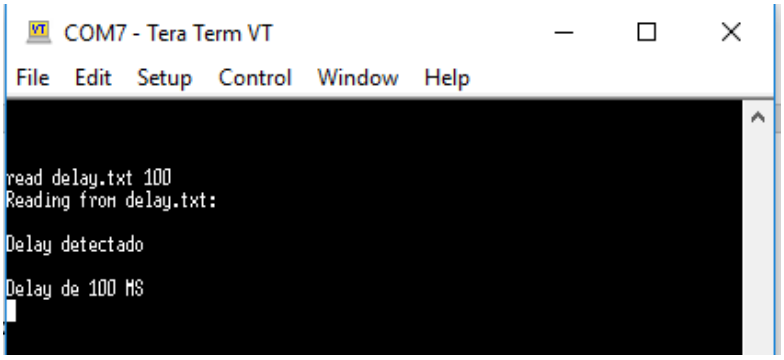


Fig. 4-2 Salida de la consola después de ejecutar la lectura del archivo

En la Figura 4-3 vemos el extracto del código creado para la interpretación del bloque "delay" que se agregó como archivo de texto por medio de una tarjeta SD.

```
int cmd = 0;
char array[count];
char delay[] = "Delay";
for(int x=0; x<=count; x++){array[x]='\0';}
if (fd && !error)  {
   bytes = 0;
   if (fseek(fd, offset, seek_mode) != IO_ERROR)  {
      printf("Reading from %s:\n", argv[1]);
      while ((c=fgetc(fd))>=0) {
         if((int)c == 42){
             if(strcmp(array,delay) == 0){
                 cmd = 1;
                 printf("\nDelay detectado\n");
                 for(int x=0; x<=count; x++){array[x]='\0';}
             }
             bytes = -1;
         }
         else if((int)c == 59){
             switch (cmd){
             case 1:
                 _time_delay(atoi(array));
                 int local = atoi(array);
                 printf("\nDelay de %d MS\n", local);
                 cmd = 0;
                 break;
             default: printf("\nNo entro\n");
             }
         }
         else{array[bytes] = c;}
         if (++bytes == count) break;
         }
      printf("\nDone.\n");
      fclose(fd);
```

Fig. 4-3 Código fuente de la interpretación del archivo de texto

Este primer código de interpretación de scripts nos deja vislumbrar la forma en la que se realizarán las demás interpretaciones. En esta aplicación se utiliza una máquina de estados simple; la ventaja de que nuestra aplicación tenga esta forma es que es fácilmente escalable y flexible.

Para la segunda revisión del código de MQX que realizamos, agregamos una máquina de estados, que nos permite interpretar cada uno de los bloques que describimos en el capítulo 3. Esta máquina de estados es de una complejidad muy superior a la mostrada anteriormente. En la figura 3-13 se muestra la definición de la máquina de estados, en donde se observará la complejidad de la cual hacemos mención.

Siguiendo con la ejecución del bloque Turn, creamos un script que lo único que hace es encender el LED azul del RGB, y a su vez nos muestra en pantalla un texto de verificación. Esto lo podemos observar en la figura 4-4.

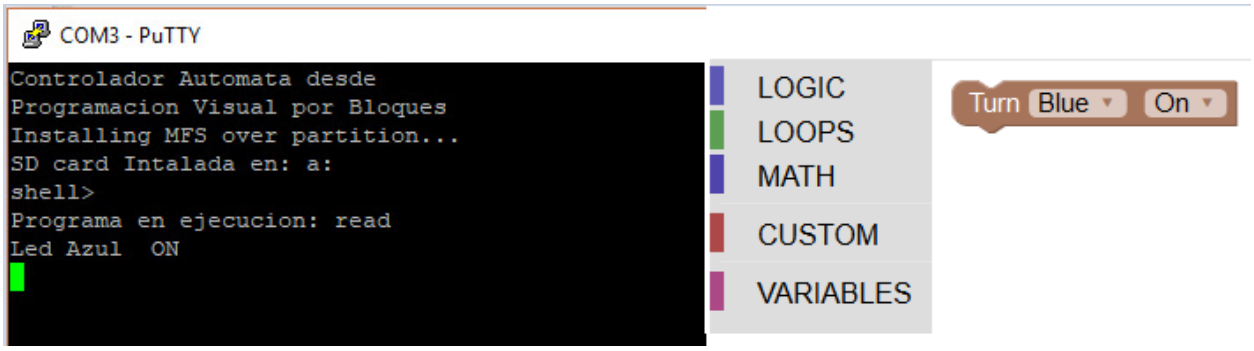


Fig. 4-4 Ejecución de bloque Turn

Para corroborar la ejecución del bloque IF y la asignación de variables podemos observar en la figura 4-5 que creamos un script que tiene una variable llamada “mi_variable”. A esta se le asigna un valor de 1, después se hace una comparación dentro de un IF, que hace la validación de la variable con el valor al que se tiene que comparar; una vez pasando la validación enciende el LED rojo del RGB y se acaba la ejecución.

La comprobación del funcionamiento del bloque Repeat estará representada mediante 3 iteraciones; por cada iteración la variable aumentará su valor en 1, el LED verde del RGB encenderá y se apagara cada 500 ms, esto se hará siempre y cuando el valor de la variable sea menor a 3, validado por un bloque IF, esto se muestra en la figura 4-6.

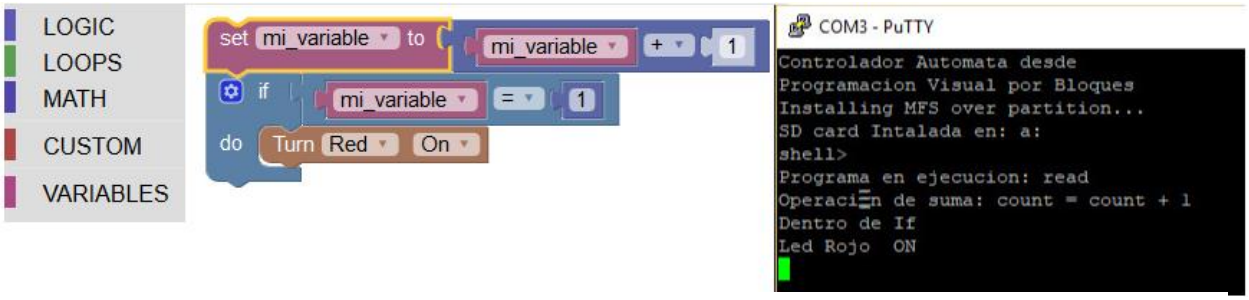


Fig. 4-5 Ejecución del Bloque IF.

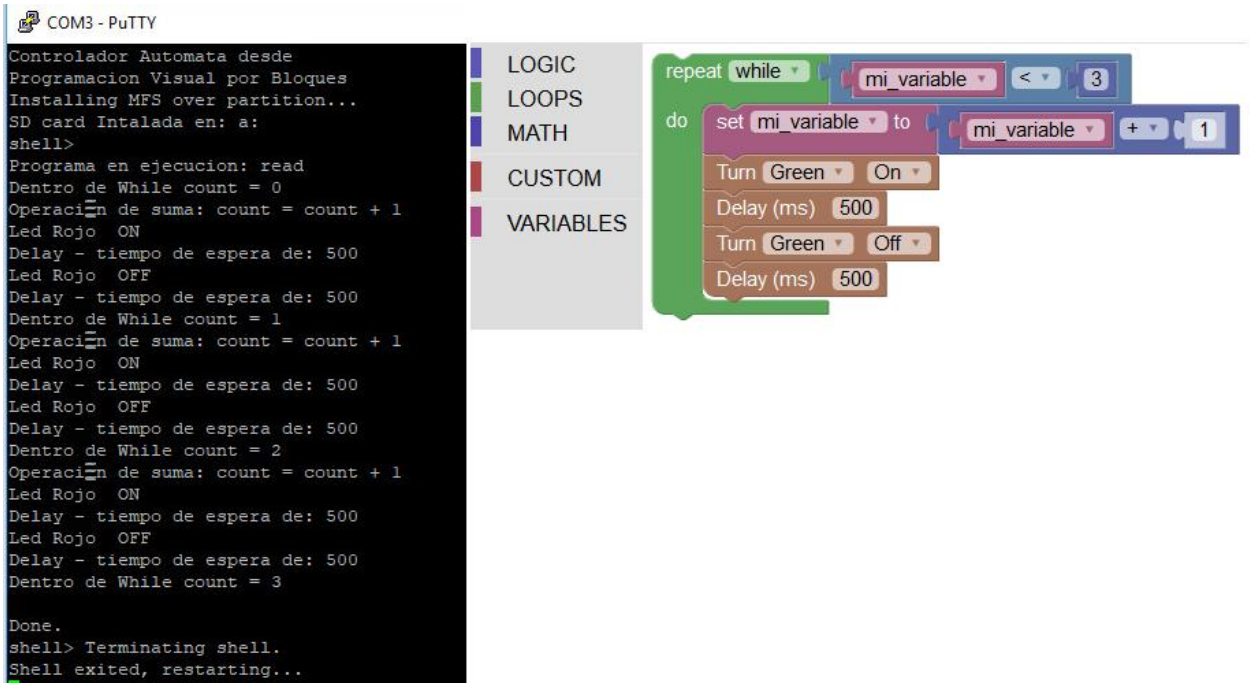


Fig. 4-6 Ejecución del bloque Repeat

Como método de visualización de resultados globales, creamos un algoritmo en nuestra distribución de Blockly que demuestra cada una de las opciones que nuestra interpretación es capaz de hacer. Este algoritmo y su transcripción a texto interpretable se muestra en la figura 4-7, y como lo hicimos anteriormente mostraremos una serie de impresiones de pantalla que avalan nuestra correcta ejecución (figura 4-8), y para comprender mejor lo que hace este script incluimos la figura 4-9 que muestra el ciclo de ejecución de dicho script.

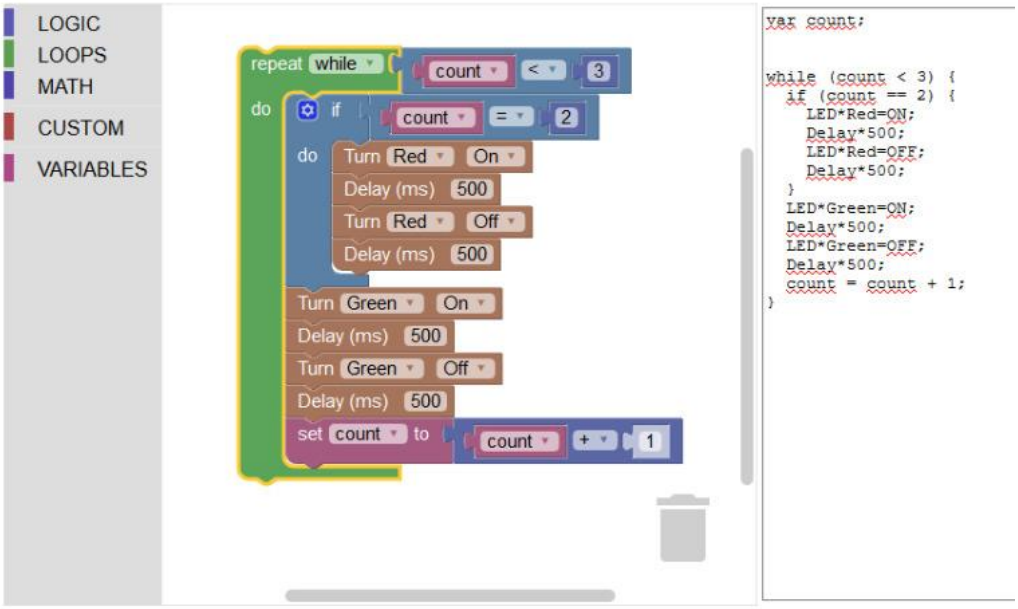


Fig. 4-7 Algoritmo de Blockly para demostración de interpretación

En la figura 4-8 se muestra en la consola cada una de las instrucciones que se ejecutan, esto es la interpretación del script que realizamos en nuestra distribución de Blockly. Se muestra cómo la variable count aumenta con cada ciclo, y a su vez habilita instrucciones, esto es debido a que en nuestro script la variable count aumenta en 1 con cada ciclo; cuando count se iguala a 2 hay un bloque IF que compara este valor y al ser cumplida la condición ejecuta una serie de instrucciones para encender y apagar el LED rojo del RGB cada 500 ms durante un ciclo. En las demás iteraciones es el LED verde del RGB el que enciende cada ciclo. Esto se cumple hasta que count es igual a 3.

.

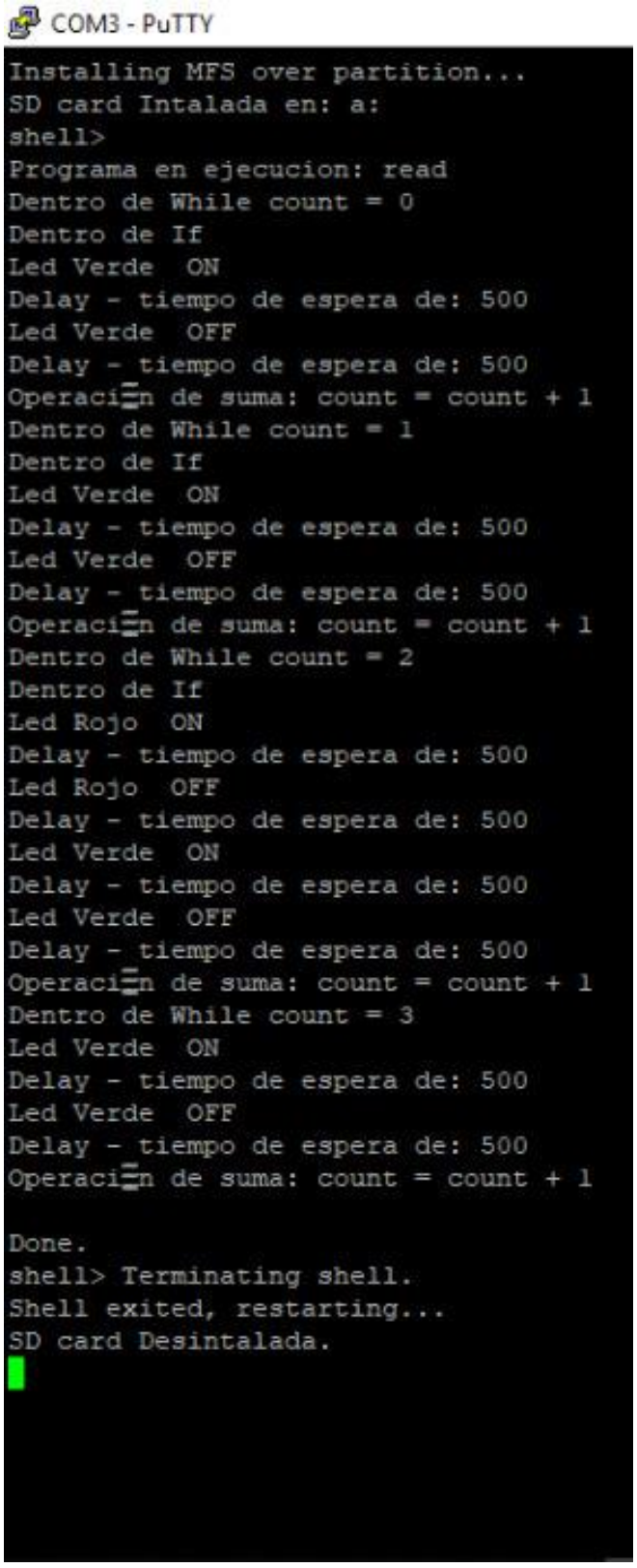


Fig. 4-8 Visualización de interpretación en consola

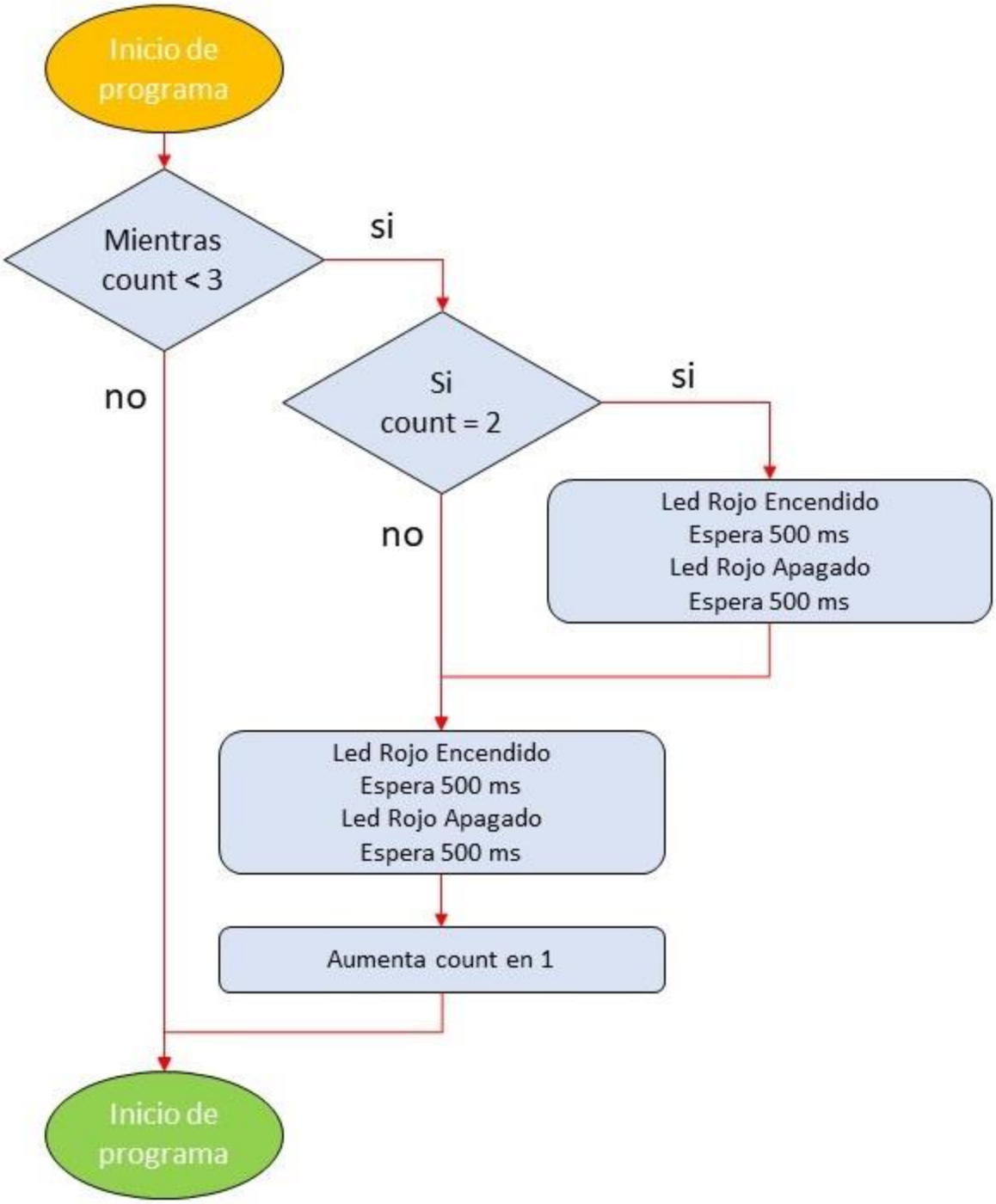


Fig. 4-9 Ciclo de ejecución de script global.

Realizando una comparación entre el algoritmo creado con Blockly y su ejecución en la consola nos podemos dar cuenta que cada una de las instrucciones, así como las validaciones se cumplen. Esta comparación se observa en la figura 4-10.

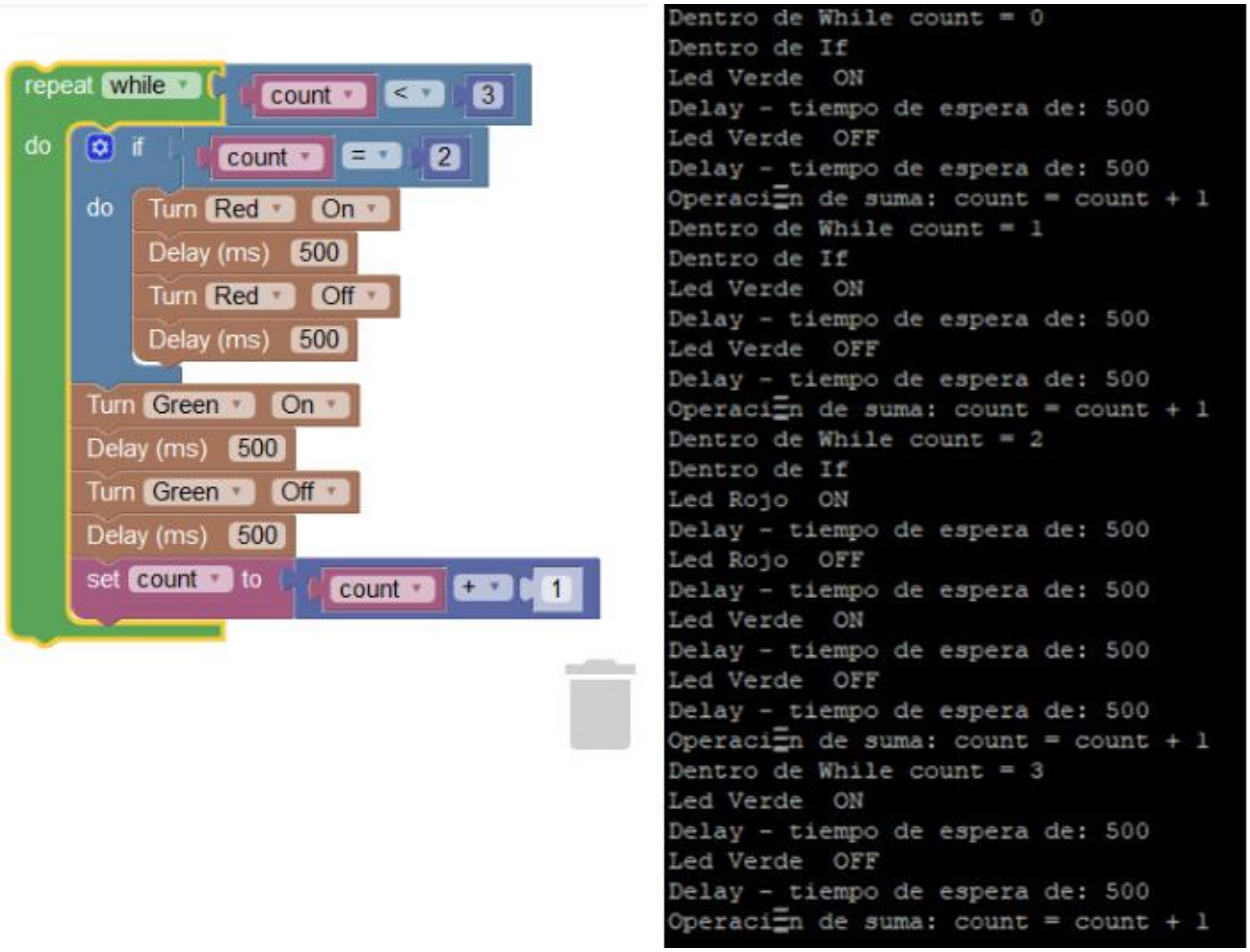


Fig. 4-10 Comparación entre algoritmo de Blockly y ejecución en la consola.

Como demostración visual brindamos un video donde se realiza esta misma ejecución, en la siguiente liga: https://goo.gl/myHTAi

Conclusiones

A lo largo del desarrollo de este proyecto nos encontramos con diversas situaciones e inconvenientes; sin embargo, se lograron sortear, pero hay que tenerlas en cuenta, ya que aún quedan casos muy específicos de ejecución que pueden no realizar lo esperado. Se usaron los casos más comunes y que se pueden dar con mayor frecuencia.

La selección del sistema operativo fue un punto crucial, pues éste le daría la robustez en la ejecución del programa o código ingresado por el usuario.

Al principio se visualizaba el objetivo de tener comandos especializados para todo tipo de señales y funciones; un inconveniente que tuvimos fue en la manipulación de señales moduladas y controlar convertidores analógicos a digitales y viceversa, pues el sistema operativo no se focalizaba en este tipo de señales. La razón por la que decidimos usar MQX fue por la robustez en la ejecución en tiempo real, tuvimos la opción de usar FreeRTOS, el cual funciona bastante bien, pero que incluso en su misma documentación y de terceros, no lo recomendaban pues tiene un uso más educativo donde el tiempo real no es tan estricto.

La programación visual fue flexible hasta cierto punto gracias al proyecto Blockly que fue la base para nuestro generador de código para el intérprete del programa.

Aún quedan muchas ideas de mejora que por razones de tiempo nos fue imposible implementar, tal es el caso del envío de datos por medios como ethernet o USB para que se auto reprogramara desde la interfaz de programación visual, implementaciones de modulación de pulsos, mejorar la inicialización automática, así como optimizaciones del intérprete para bajar la cantidad de ciclos por comando ejecutado.

A pesar de las varias dificultades presentadas hemos cumplido con las especificaciones principales y fundamentales, mismas que fueron las que incentivaron a crear este proyecto.

Apéndices

A. CODIGO FUENTE DE INTERPRETACION EN MQX

```
/*HEADER**********************************************************************
*
* Copyright 2008 Freescale Semiconductor, Inc.
* Copyright 2004-2008 Embedded Access Inc.
* Copyright 1989-2008 ARC International
*
* This software is owned or controlled by Freescale Semiconductor.
* Use of this software is governed by the Freescale MQX RTOS License
* distributed with this Material.
* See the MQX_RTOS_LICENSE file distributed for more details.
*
* Brief License Summary:
* This software is provided in source form for you to use free of charge,
* but it is not open source software. You are allowed to use this software
* but you cannot redistribute it or derivative works of it in source form.
* The software may be used only in connection with a product containing
* a Freescale microprocessor, microcontroller, or digital signal processor.
* See license agreement file for full license terms including other
* restrictions.
**********************************************************************************
*
* Comments:
*
*   This file contains the source for an MFS shell function.
*
*
*END**********************************************************************************/

#include <string.h>
#include <mqx.h>
#include <fio.h>
#include <mfs.h>
#include <shell.h>
#include <sh_prv.h>
#include <stdlib.h>

#if SHELLCFG_USES_MFS

/*FUNCTION*-------------------------------------------------------------
*
* Function Name    :  Shell_read
* Returned Value   : int32_t error code
* Comments  :  Reads from a file .
*
*END*-------------------------------------------------------------------*/
```

```
struct CustomVar{
char name[32];
union cont {
bool x;
char c;
int i;
float f;
}Value;
};

enum State {sStart = 0, sRecolection, sCommand, sParam, sVariable, sEndCycle,
sNumber, sName, sSetName, sEnd, sControl, sIf, sWhile, sSetValue};
enum Commands {cEmpty = 0, cDelay, cLed, cButton, cInput, cOutput};
enum Variables {vNull = 0, vInteger, vFloat, vBoolean, vChar};
enum Control {nEmpty = 0, nIf, nWhile, nVar, nSet};
enum Operation {pNull = 0, pAssignation, pEqual, pMayor, pMayorOrEqual, pMinor, pMinorOrEqual,
pDifferent, pAdd, pLess, pMult, pDiv};

int32_t  Shell_read(int32_t argc, char *argv[] )
{ /* Body */
bool          print_usage, shorthelp = FALSE;
int32_t           return_code = SHELL_EXIT_SUCCESS;
uint32_t          count, bytes;
int32_t           offset;
int32_t        seek_mode;
MQX_FILE_PTR      fd = NULL;
char          *abs_path;
_mqx_int          c;
SHELL_CONTEXT_PTR    shell_ptr = Shell_get_context( argv );
int32_t             error = 0;

print_usage = Shell_check_help_request(argc, argv, &shorthelp );

LWGPIO_STRUCT led1, led2, led3, CurrentOutput, btn1;
lwgpio_init(&led1, BSP_LED1, LWGPIO_DIR_OUTPUT, LWGPIO_VALUE_NOCHANGE);
lwgpio_init(&led2, BSP_LED2, LWGPIO_DIR_OUTPUT, LWGPIO_VALUE_NOCHANGE);
lwgpio_init(&led3, BSP_LED3, LWGPIO_DIR_OUTPUT, LWGPIO_VALUE_NOCHANGE);
lwgpio_init(&btn1, BSP_BUTTON1, LWGPIO_DIR_INPUT, LWGPIO_VALUE_NOCHANGE);

lwgpio_set_functionality(&led1, BSP_LED1_MUX_GPIO);
lwgpio_set_functionality(&led2, BSP_LED2_MUX_GPIO);
lwgpio_set_functionality(&led3, BSP_LED3_MUX_GPIO);

lwgpio_set_value(&led1, LWGPIO_VALUE_HIGH); /* set pin to 1 */
lwgpio_set_value(&led2, LWGPIO_VALUE_HIGH); /* set pin to 1 */
lwgpio_set_value(&led3, LWGPIO_VALUE_HIGH); /* set pin to 1 */

#define TERMINAL_CURSOR_POSITION_MAX    (80)
```

```
if (!print_usage)  {
if ((argc < 2) || (argc > 5)) {
printf("Error, invalid number of parameters\n");
return_code = SHELL_EXIT_ERROR;
print_usage=TRUE;
}
/* check if filesystem is mounted */
else if (NULL == Shell_get_current_filesystem(argv))
{
printf("Error, file system not mounted\n");
return_code = SHELL_EXIT_ERROR;
} else {
count = 0;
offset = 0;
seek_mode = IO_SEEK_CUR;
if (argc >= 3)  {
if ( !Shell_parse_uint_32(argv[2], &count ))  {
printf("Error, invalid length\n");
return_code = SHELL_EXIT_ERROR;
print_usage=TRUE;
}
else  {
if (argc >= 5)  {
if (strcmp(argv[3], "begin") == 0) {
seek_mode = IO_SEEK_SET;
} else if (strcmp(argv[3], "end") == 0) {
seek_mode = IO_SEEK_END;
} else if (strcmp(argv[3], "current") == 0) {
seek_mode = IO_SEEK_CUR;
} else {
printf("Error, invalid seek type\n");
return_code = SHELL_EXIT_ERROR;
print_usage=TRUE;
}

if (return_code == SHELL_EXIT_SUCCESS)  {
if (! Shell_parse_int_32(argv[4], &offset ))  {
printf("Error, invalid seek value\n");
return_code = SHELL_EXIT_ERROR;
print_usage=TRUE;
}
}
}
}
}
if (return_code == SHELL_EXIT_SUCCESS)  {
if (MFS_alloc_path(&abs_path) != MFS_NO_ERROR) {
printf("Error, unable to allocate memory for paths\n" );
```

```
return_code = SHELL_EXIT_ERROR;
} else {
error = _io_rel2abs(abs_path,shell_ptr->CURRENT_DIR,(char *) argv[1],PATHNAME_SIZE,shell_ptr-
>CURRENT_DEVICE_NAME);
if(!error)
{
fd = fopen(abs_path, "r");
}
MFS_free_path(abs_path);
bool xExit = false;
char * MemLoad = (char*)malloc(fd->SIZE+1);
memset(MemLoad,'\0',fd->SIZE+1);
char * MemInit;
enum State Machine = sRecolection;
enum Commands CurrentCommand = cEmpty;
enum Variables CurrentVariable = vNull;
enum Control ControlStack[3];
enum Operation CurrentOperation = pNull;
char * * MemoryStack = malloc(sizeof(MemLoad));
int StackControl = 0;
int PointerStack = 0;
char array[count];
struct CustomVar Data[3];
int countVariable = 0;
bytes = 0;
MemInit = MemLoad;
memset(array,'\0', count);
if (fd && !error)  {
if (fseek(fd, offset, seek_mode) != IO_ERROR)  {
printf("Programa en ejecucion: %s\n", argv[1]);
while((c=fgetc(fd)) >= 0){
*MemLoad = c;
MemLoad++;
}
MemLoad = MemInit;
do {
switch(Machine){
case sRecolection:
xExit = false;
while(!xExit && *MemLoad >= 0){
switch(*MemLoad){
case '*':
Machine = sCommand;
xExit = true;
break;
case '=':
Machine = sParam;
xExit = true;
break;
```

```
case ';':
Machine = sEnd;
xExit = true;
break;
case ' ':
if((int)array[0] > 32){
Machine = sControl;
xExit = true;
}
break;
case '}':
Machine = sEndCycle;
xExit = true;
break;
default:
if(isalpha(*MemLoad))
array[bytes++] = *MemLoad;
break;
}
if(Machine != sEndCycle){
if(*MemLoad != ';')
MemLoad++;
}
}
break;
case sControl:
if(strcmp(array,"if")==0){
while(*(MemLoad++) != '(');
Machine = sIf;
}
else if(strcmp(array,"while")==0){
*MemoryStack = &MemLoad[-6];
while(*(MemLoad++) != '(');
Machine = sWhile;
}
else if(strcmp(array,"var")==0)
Machine = sName;
else{
bytes = 0;
while(bytes < 3){
if(strcmp(Data[bytes].name,array)==0){
CurrentVariable = bytes;
Machine = sSetValue;
break;
}
else
bytes++;
}
}
```

```
bytes = 0;
memset(array,'\0', count);
break;
case sEndCycle:
if(ControlStack[--PointerStack] == nWhile){
if(StackControl>0){
StackControl--;
MemLoad = *MemoryStack;
}
}
else
while((*(MemLoad++)) != '}');
Machine = sRecolection;
break;
case sIf:
while(*MemLoad != '{'){
switch(*MemLoad){
case ' ':
bytes = 0;
if(array[0] != '\0'){
while(bytes < 4){
if(strcmp(Data[bytes].name,array)==0) {
countVariable = bytes;
bytes = 0;
memset(array,'\0', count);
break;
}
bytes++;
}
}
break;
case ')':
if(isdigit(array[0])){
printf("Dentro de If\r\n");
switch(CurrentOperation){
case pEqual:
if(Data[countVariable].Value.i == atoi(array)){
ControlStack[PointerStack++] = nIf;
Machine = sRecolection;
}
else{
ControlStack[PointerStack++] = nEmpty;
Machine = sEndCycle;
}
break;
case pMayor:
if(Data[countVariable].Value.i > atoi(array)){
ControlStack[PointerStack++] = nIf;
Machine = sRecolection;
```

```
}
else{
ControlStack[PointerStack++] = nEmpty;
Machine = sEndCycle;
}
break;
case pMinor:
if(Data[countVariable].Value.i < atoi(array)){
ControlStack[PointerStack++] = nIf;
Machine = sRecolection;
}
else{
ControlStack[PointerStack++] = nEmpty;
Machine = sEndCycle;
}
break;
case pDifferent:
if(Data[countVariable].Value.i != atoi(array)){
ControlStack[PointerStack++] = nIf;
Machine = sRecolection;
}
else{
ControlStack[PointerStack++] = nEmpty;
Machine = sEndCycle;
}
break;
}
}
else if(isalpha(array[0])){
if(array[1] != '\0'){
if(Data[countVariable].Value.x == (bool)array[0]){
Machine = sRecolection;
ControlStack[PointerStack++] = nIf;
}
else{
Machine = sEndCycle;
ControlStack[PointerStack++] = nEmpty;
}
}
else{
if(Data[countVariable].Value.c == array[0]){
Machine = sRecolection;
ControlStack[PointerStack++] = nIf;
}
else{
Machine = sEndCycle;
ControlStack[PointerStack++] = nEmpty;
}
}
```

```
}
bytes = 0;
memset(array,'\0', count);
break;
case '=':
if(*(MemLoad++) == '=')
CurrentOperation = pEqual;
break;
case '>':
if(*(MemLoad++) == '=')
CurrentOperation = pMayorOrEqual;
else
CurrentOperation = pMayor;
break;
case '<':
if(*(MemLoad++) == '=')
CurrentOperation = pMinorOrEqual;
else
CurrentOperation = pMinor;
break;
case '!':
if(*(MemLoad++) == '=')
CurrentOperation = pDifferent;
break;
default:
if(*MemLoad>32)
array[bytes++] = *MemLoad;
break;
}
MemLoad++;
}
break;
case sWhile:
StackControl++;
while(*MemLoad != '{'){
switch(*MemLoad){
case ' ':
bytes = 0;
if(array[0] != '\0'){
while(bytes < 4){
if(strcmp(Data[bytes].name,array)==0) {
countVariable = bytes;
bytes = 0;
memset(array,'\0', count);
break;
}
bytes++;
}
}
```

```
break;
case ')':
if(isdigit(array[0])){
printf("Dentro de While %s = %d\r\n", Data[countVariable].name,Data[countVariable].Value.i);
switch(CurrentOperation){
case pEqual:
if(Data[countVariable].Value.i == atoi(array)){
ControlStack[PointerStack++] = nWhile;
Machine = sRecolection;
}
else{
ControlStack[PointerStack++] = nEmpty;
Machine = sEndCycle;
}
break;
case pMayor:
if(Data[countVariable].Value.i > atoi(array)){
ControlStack[PointerStack++] = nWhile;
Machine = sRecolection;
}
else{
ControlStack[PointerStack++] = nEmpty;
Machine = sEndCycle;
}
break;
case pMinor:
if(Data[countVariable].Value.i < atoi(array)){
ControlStack[PointerStack++] = nWhile;
Machine = sRecolection;
}
else{
ControlStack[PointerStack++] = nEmpty;
Machine = sEndCycle;
}
break;
case pDifferent:
if(Data[countVariable].Value.i != atoi(array)){
ControlStack[PointerStack++] = nWhile;
Machine = sRecolection;
}
else{
ControlStack[PointerStack++] = nEmpty;
Machine = sEndCycle;
}
break;
}
}
else if(isalpha(array[0])){
if(array[1] != '\0'){
```

```
if(Data[countVariable].Value.x == (bool)array[0]){
ControlStack[PointerStack++] = nWhile;
Machine = sRecolection;
}
else{
ControlStack[PointerStack++] = nEmpty;
Machine = sEndCycle;
}
}
else{
if(Data[countVariable].Value.c == array[0]){
ControlStack[PointerStack++] = nWhile;
Machine = sRecolection;
}
else{
ControlStack[PointerStack++] = nEmpty;
Machine = sEndCycle;
}
}
}
bytes = 0;
memset(array,'\0', count);
break;
case '=':
if(*(MemLoad) == '=')
CurrentOperation = pEqual;
break;
case '>':
if(*(MemLoad) == '=')
CurrentOperation = pMayorOrEqual;
else
CurrentOperation = pMayor;
break;
case '<':
if(*(MemLoad) == '=')
CurrentOperation = pMinorOrEqual;
else
CurrentOperation = pMinor;
break;
case '!':
if(*(MemLoad) == '=')
CurrentOperation = pDifferent;
break;
default:
if(*MemLoad>32)
array[bytes++] = *MemLoad;
break;
}
MemLoad++;
```

```
}
bytes = 0;
memset(array,'\0', count);
break;
case sCommand:
if(strcmp(array, "Delay")==0)
CurrentCommand = cDelay;
else if(strcmp(array, "LED")==0)
CurrentCommand = cLed;
if(isdigit(*MemLoad))
Machine = sNumber;
else if(isalpha(*MemLoad))
Machine = sRecolection;
bytes = 0;
memset(array,'\0', count);
break;
case sName:
xExit = false;
while(!xExit && *MemLoad>=0){
if(isalpha(*MemLoad) || isdigit(*MemLoad)){
array[bytes++] = *MemLoad;
}
else {
switch(*MemLoad){
case '=':
case ';':
Machine = sSetName;
xExit = true;
break;
}
}
MemLoad++;
}
CurrentVariable = vInteger;
break;
case sParam:
if(strcmp(array, "Red")==0){
CurrentOutput = led1;
printf("Led Rojo ");
}
else if(strcmp(array, "Green")==0){
CurrentOutput = led2;
printf("Led Verde ");
}
else if(strcmp(array, "Blue")==0){
CurrentOutput = led3;
printf("Led Azul ");
}
if(isdigit(*MemLoad))
```

```
Machine = sNumber;
else if(isalpha(*MemLoad))
Machine = sRecolection;
else if(*MemLoad == ';'){
Machine = sEnd;
}
bytes = 0;
memset(array,'\0', count);
break;
case sSetValue:
xExit = false;
struct CustomVar LastValue;
CurrentOperation = pNull;
while(!xExit && *MemLoad>=0){
switch(*MemLoad){
case '+':
CurrentOperation = pAdd;
break;
case '-':
CurrentOperation = pLess;
break;
case '*':
CurrentOperation = pMult;
break;
case '/':
CurrentOperation = pDiv;
break;
case '.':
if(CurrentVariable == vFloat)
array[bytes++] = *MemLoad;
break;
case ';':
bytes--;
switch(CurrentOperation){
case pAdd:
printf("Operación de suma: %s = %s + %d\r\n", Data[bytes].name, Data[bytes].name, atoi(array));
Data[bytes].Value.i = LastValue.Value.i + atoi(array);
break;
case pLess:
Data[bytes].Value.i = LastValue.Value.i - atoi(array);
break;
case pMult:
Data[bytes].Value.i = LastValue.Value.i * atoi(array);
break;
case pDiv:
Data[bytes].Value.i = LastValue.Value.i / atoi(array);
break;
}
Machine = sRecolection;
```

```
xExit = true;
break;
case ' ':
if(array[0] != '\0'){
bytes = 0;
while(bytes < 3){
if(strcmp(Data[bytes].name,array)==0){
LastValue = Data[bytes];
break;
}
else
bytes++;
}
bytes = 0;
memset(array,'\0', count);
}
break;
default:
if(isdigit(*MemLoad) || isalpha(*MemLoad)){
array[bytes++] = *MemLoad;
}
}
MemLoad++;
}
CurrentOperation = pAssignation;
bytes = 0;
memset(array,'\0', count);
break;
case sNumber:
xExit = false;
while(!xExit && *MemLoad>=0){
switch(*MemLoad){
case '.':
if(CurrentVariable == vFloat)
array[bytes++] = *MemLoad;
break;
case ';':
Machine = sEnd;
xExit = true;
break;
default:
if(isdigit(*MemLoad)){
array[bytes++] = *MemLoad;
}
}
MemLoad++;
}
break;
case sSetName:
```

```
memcpy(Data[countVariable].name, array,32);
bytes = 0;
memset(array,'\0', count);
if(MemLoad[-1] == ';'){
Machine = sEnd;
array[bytes++] = '0';
}
else{
while(*(MemLoad++) < 32);
if(isdigit(*MemLoad)){
Machine = sNumber;
array[bytes++] = *MemLoad;
}
else if(isalpha(*MemLoad)){
Machine = sRecolection;
array[bytes++] = *MemLoad;
}
}
break;
case sEnd:
switch(CurrentCommand){
case cEmpty:
switch(CurrentVariable){
case vInteger:
Data[countVariable++].Value.i = atoi(array);
break;
case vChar:
Data[countVariable++].Value.c = array[0];
break;
case vBoolean:
Data[countVariable++].Value.x = array[0];
break;
case vFloat:
Data[countVariable++].Value.i = atof(array);
break;
}
break;
case cDelay:
printf("Delay - tiempo de espera de: %d\r\n",atoi(array));
_time_delay(atoi(array));
break;
case cLed:
if(strcmp(array,"ON") == 0){
lwgpio_set_value(&CurrentOutput, LWGPIO_VALUE_LOW);
printf(" ON\r\n");
}
else if(strcmp(array,"OFF") == 0){
lwgpio_set_value(&CurrentOutput, LWGPIO_VALUE_HIGH);
printf(" OFF\r\n");
```

```
}
break;
}
bytes = 0;
memset(array,'\0', count);
CurrentCommand = cEmpty;
CurrentVariable = vNull;
Machine = sRecolection;
MemLoad++;
break;
}
}while (*MemLoad != '\0');
printf("\nDone.\n");
fclose(fd);
} else  {
printf("Error, unable to seek file %s.\n", argv[1] );
return_code = SHELL_EXIT_ERROR;
}
} else  {
printf("Error, unable to open file %s.\n", argv[1] );
return_code = SHELL_EXIT_ERROR;
}
}
}
}
}
if (print_usage)  {
if (shorthelp)  {
printf("%s <filename> <bytes> [<seek_mode>] [<offset>]\n", argv[0]);
} else  {
printf("Usage: %s <filename> <bytes> [<seek_mode>] [<offset>]\n", argv[0]);
printf("   <filename>   = filename to display\n");
printf("   <bytes>      = number of bytes to read\n");
printf("   <seek_mode>  = one of: begin, end or current\n");
printf("   <offset>     = seek offset\n");
}
}
return return_code;
} /* Endbody */

#endif //SHELLCFG_USES_MFS

/* EOF*/
```

Bibliografía

[1] Diseño de Sistemas Operativos en Ambientes Embebidos, nota de clases para ESE008A, Departamento de Electrónica, Sistemas e Informática, Instituto Tecnlóico de Estudios Superiores de Occidente, Primavera 2018.

[2] Hipertextual, «¿Qué es una API?,» 15 Mayo 2014. [En línea]. Available: https://hipertextual.com/archivo/2014/05/que-es-api/. [Último acceso: 28 Mayo 2018].

[3] M. M. Kumar y P. Sivraj, «DrawCode: Visual tool for programming microcontrollers,» de *2017 3rd International Conference on Advances in Computing,Communication & Automation (ICACCA) (Fall)*, Dehradun, India, 2017.

[4] A. B. Pratomo y R. S. Perdana, «Arduviz, a visual programming IDE for arduino,» de *2017 International Conference on Data and Software Engineering (ICoDSE)*, Palembang, Indonesia, 2017.

[5] L. E. Garabito Siordia, «Simplified Development Tool for Embedded Systems,» Zapopan, Jalisco, 2010.

[6] R. Kamal, Embedded Systems: Architecture, Programming and Design, McGraw-Hill Education, 2009.

[7] W. Stallings, Operating Systems: Internals and Design Principles, New Jersey: Pearson, 2011.

[8] What is operating system (OS)? - Definition from WhatIs.com", «WhatIs.com,» WhatIs.com, Septiembre 2016. [En línea]. Available: https://whatis.techtarget.com/definition/operating-system-OS. [Último acceso: 6 Junio 2018].

[9] F. Pérez y J. C. Felíx García, Problemas de Sistemas Operativos: De la base al diseño, Madrid: McGraw-Hill/Interamericana de España, S. A. U., 2003.

[10] S. P. M., «Embedded Operating Systems for Real-Time Applications,» de *Electronic Systems Group, EE Dept, IIT Bombay*, Bombay, 2002.

[11] A. Nogueira y M. Calha, «Predictability and Efficiency in Contemporary Hard RTOS for Multiprocessor Systems,» de *2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications*, Toyama, 211.

[12] Techopedia Inc, «techopedia,» 15 junio 2012. [En línea]. Available: https://www.techopedia.com/definition/22855/visual-programming-language-vpl. [Último acceso: 11 junio 2018].

[13] W. C. Shih, «Mining Learners' Behavioral Sequential Patterns in a Blockly Visual Programming Educational Game,» de *2017 International Conference on Industrial Engineering, Management Science and Application (ICIMSA)*, Seoul, 2017.

[14] E. Pasternak, R. Fenichel y A. N. Marshall, «Tips for creating a block language with blockly,» de *2017 IEEE Blocks and Beyond Workshop (B&B)*, Raleigh, NC, 2017.

[15] Wikipedia, «www.wikipedia.com,» 3 junio 2018. [En línea]. Available: https://en.wikipedia.org/wiki/NXP_Semiconductors. [Último acceso: 13 junio 2018].

[16] NXP Semiconductors B.V., «FRDM-K64F Freedom Module User's Guide,» 2016.

Índice