METACONTROLADORES: UNA ALTERNATIVA PARA EL CONTROL DE DISPOSITIVOS EN UN RTOS

META-DRIVERS: A NOVEL APPROACH TO DEVICE CONTROL IN RTOS

Daniela Dives Viera
Universidad de Guadalajara, CUCEI, México
daniela.dives5481@alumnos.udg.mx

Edwin Christian Becerra Alvarez
Universidad de Guadalajara, CUCEI, México
edwin.becerra@academicos.udg.mx

Juan José Raygoza Panduro
Universidad de Guadalajara, CUCEI, México
juan.rpanduro@academicos.udg.mx

María Regina Rivas Becerra
Universidad de Guadalajara, CUCEI, México
maria.rivas2804@alumnos.udg.mx

Recepción: 2/diciembre/2025 **Aceptación:** 25/diciembre/2025

Resumen

En este trabajo se presenta una alternativa para el control de dispositivos en un sistema embebido, donde cada uno requiere un controlador independiente, sin embargo, el mejor aprovechamiento de los recursos limitados puede impactar considerablemente su desempeño. Mediante el uso de metacontroladores es posible reducir los requerimientos de memoria y tiempo de desarrollo, así como la posibilidad de soportar nuevos dispositivos. Para tal fin, se presenta una metodología que se basa en analizar las señales de control, permitiendo identificar patrones y crear un diccionario para soportar los periféricos. Las simulaciones comportamentales realizadas en EMOS, aceleraron el proceso de depuración para las palabras diseñadas, siendo éstas utilizadas 144 veces en el SSDS. Además, las mediciones experimentales con una tarjeta Raspberry Pi Pico validan la metodología propuesta. Finalmente, se determina que existe una correspondencia

del 100% en las palabras utilizadas para los casos analizados, reduciendo la ocupación de memoria RAM del microcontrolador.

Palabras Clave: Metacontrolador, metalenguaje, RTOS, tiempo real.

Abstract

This paper presents an alternative approach for device control in embedded systems, where each device generally requires an independent driver. However, efficient utilization of limited system resources can significantly enhance overall performance. The use of metadrivers enables a reduction in memory requirements and development time, while also facilitating the integration of new devices. To achieve this, a methodology is proposed based on the analysis of control signals, allowing for the identification of patterns and the construction of a dictionary to support peripheral devices. Behavioral simulations performed using EMOS accelerated the debugging process of the designed control words, which were utilized 144 *times in the SSDS. In addition, experimental measurements conducted on a Raspberry Pi Pico board validate the proposed methodology. Results show a* 100% *match of the control words for the analyzed cases, along with a reduction in* RAM *usage on the microcontroller.*

Keywords: *Metadriver, metalanguage, real-time, RTOS.*

1. Introducción

La humanidad recurre al uso de diferentes dispositivos electrónicos todos los días, ya sea para realizar tareas de suma importancia, como puede ser el uso de aparatos médicos para soporte vital [Kaptain, 2024], hasta actividades menos críticas, por ejemplo usar un teléfono inteligente para revisar las redes sociales [Nawaz, 2025], e incluso utilizar lavadoras o microondas para realizar las actividades cotidianas, en cada uno de estos dispositivos se encuentra uno o más sistemas embebidos [Vahid, 1999], [Pont, 2022].

Los sistemas embebidos son dispositivos electrónicos con características similares a las de una computadora de propósito general, cuentan con memoria y procesador, con la gran diferencia de que sus recursos son limitados, siendo diseñados y

programados para cumplir funciones específicas, como pudiera ser el control de apertura y cierre de puertas, la regulación de temperatura de un aire acondicionado, entre un sinfín de aplicaciones, para ello un desarrollador coloca una serie de instrucciones, llamadas programa, que ejecuta el microcontrolador [Mano, 2015], [Galeano, 2009], [Lutenberg, 2022]. Es común que el desarrollador se encargue de diseñar e implementar sus propios controladores (*drivers*), sin embargo, existen diferentes alternativas para realizar esta tarea, una de ellas es el uso de un Sistema Operativo Embebido (EOS, del inglés *Embedded Operating System*), esta capa de software facilita dichas tareas al diseñador.

Por otro lado, a diferencia de los Sistemas Operativos (OS, del inglés *Operating System*) de uso general, los EOS deben realizar tareas de administración de recursos del sistema, con algunas características de tiempo real, aun así, estos deben ejecutar los procesos con restricciones en cuanto a memoria disponible y tiempo de ejecución [Pérez, 2009], [Holt, 2018]. Algunos EOS derivan de OS de uso general, entre los cuales se pueden encontrar: Embedded Linux, Windows CE y UNIX [Hallinan, 2010], [Phung, 2011], [Srirengan, 2006]. Por otro lado, los Sistemas Operativos de Tiempo-Real (RTOS, del inglés *Real-Time Operating System*), ofrecen soluciones más especializadas, para aplicaciones que requieren una respuesta en un tiempo determinado [Amos, 2020].

Un claro ejemplo de ello es el caso de VxWorks, un RTOS para aplicaciones *hard-time*, como en el ámbito de la aeronáutica, e inclusive sirviendo en propósitos militares empleándose en vehículos aéreos no tripulados, sin embargo, tiene la desventaja de no contar con un soporte adecuado para aplicaciones de propósito general [Abuazoum, 2025], [Rico, 2022], [Abbasi, 2019].

Gran parte de los OS tienen la característica común de ser codificados utilizando el lenguaje de programación C, por lo cual las aplicaciones de usuario tienden a seguir su misma filosofía [Kernighan, 1991], [Becker, 2010]. Sin embargo, existen algunos sistemas menos conocidos que soportan programación con metalenguaje, como es Mecrisp-Stellaris basado en FORTH, y EMOS, (del inglés, *Embedded Metalanguage Operating System*), siendo este último utilizado en este trabajo [Brodie,2004], [Mecrisp,2025], [Embedded,2025].Para el caso de EMOS, soporta su

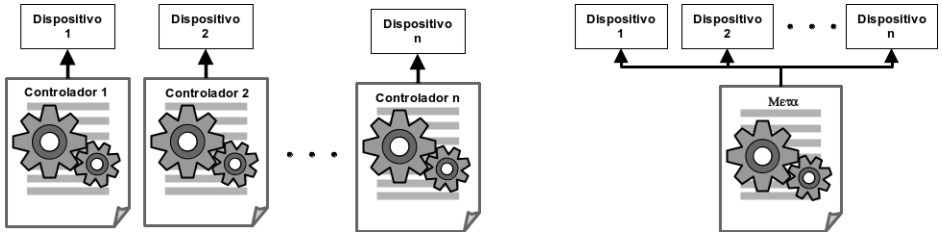
propio metalenguaje llamado Leo, que basa su función en máquinas de pilas que contienen **celdas**, las cuales son espacios para almacenar información.

Por otro lado, los programas en Leo están conformados por **palabras**, donde cualquier cadena de caracteres consecutivos, a excepción de los espacios, forman a una, creando el **diccionario de usuario**, cuya función es extender el **diccionario base**, que contiene a las primitivas del metalenguaje, siendo estas similares a las palabras reservadas de un lenguaje de programación [Becerra, 2023].

La filosofía de los metalenguajes permite acercarse a los problemas de diseño de software, desde un punto de vista diferente, ya que estos permiten describir las reglas para un lenguaje de programación, pudiendo crear nuevos [Schild, 1993].

Es común usar periféricos en las computadoras, como el hecho de conectar una memoria USB y usarla de inmediato, detrás de ello se encuentra una pieza de software muy importante, llamada controlador de dispositivo (*driver*), cuya función es proporcionar el código necesario, para establecer una comunicación entre el dispositivo y la computadora [Jadhav, 2014], [Cingel, 2017]. En los sistemas embebidos esto no suele suceder así, en este ámbito, el desarrollador debe diseñar e implementar manualmente un controlador para cada uno de los periféricos, tal como se presenta en la Figura 1a, lo cual puede resultar en una mayor ocupación de recursos del sistema, es por ello que en este trabajo se presenta la propuesta del desarrollo de **metacontroladores**, un tipo de controlador que permite gestionar uno o más dispositivos tal como se muestra en la Figura 1b.

En este trabajo se presentan dos casos de diseño con metacontroladores, los cuales se verifican experimentalmente utilizando la tarjeta de desarrollo Raspberry Pi Pico [Raspberry, 2020].



a) Uso de controladores tradicionales. *b) Propuesta de Metacontrolador.*

Fuente: elaboración propia

Figura 1 Diferencia entre el uso de controladores y metacontroladores.

2. Métodos

El primer paso a seguir para crear un metacontrolador, es analizar las señales de control requeridas. Durante este proceso de análisis, se identifican patrones que se repiten, donde estos se codifican en **palabras** para el metalenguaje (una palabra para cada patrón identificado).

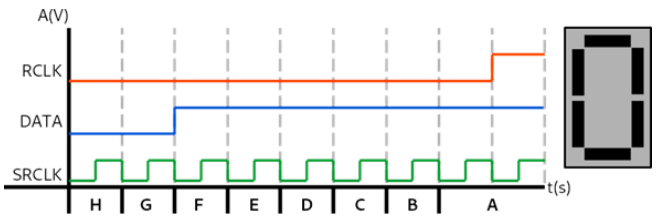
Las palabras diseñadas para replicar patrones, permiten generar el comportamiento requerido en las señales de control, y dicho programa se conoce como metacontrolador.

Para asegurar que el programa reproduzca los patrones correctamente, se realizan simulaciones comportamentales, validando que las señales de control sean las requeridas, para entonces ser implementados, donde el metacontrolador se carga a un microcontrolador, para ejecutar pruebas experimentales en tiempo real, y finalmente verificar la correcta funcionalidad de las aplicaciones.

Seven Segment Display System (SSDS)

El dispositivo SSDS, despliega los símbolos del sistema numérico hexadecimal, utilizando un *display* de 7 segmentos, para ello se emplea un registro de entrada serial – salida paralela, cuyo funcionamiento requiere de tres señales de control: SRCLK, el reloj; RCLK, el habilitador de salida; y SER/DATA, la señal de ingreso de datos hacia el registro. Para ilustrar el proceso de análisis, se toma como ejemplo el símbolo ‘0’, cuyo diagrama de señales de control se muestra en la Figura 2, donde se presentan intervalos de tiempo para los segmentos ‘A’ hasta la ‘H’.

A partir del diagrama mostrado en la Figura 2, se identifican tres patrones que se repiten en varios momentos. En la Tabla 1 se muestran los comportamientos de cada uno de estos patrones, así como los segmentos en los que se presentan.



Fuente: elaboración propia

Figura 2 Diagrama de tiempo para desplegar el símbolo ‘0’ en el *display*.

Tabla 1 Patrones para el dispositivo SSDS.

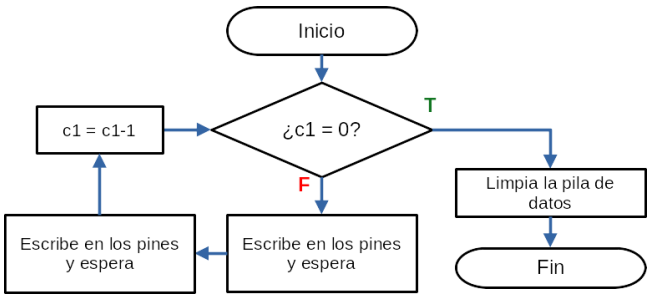
Patrón	SRCLK	DATOS	RCLK	Segmentos
Patrón 1	$0/1$	0	0	H, G
Patrón 2	$0/1$	0	1	N/A
Patrón 3	$0/1$	1	0	F hasta primera mitad de A
Patrón 4	$0/1$	1	1	Segunda mitad de A

Fuente: elaboración propia

Cabe destacar que para el caso del Patrón 2, este solo se presenta en símbolos que no requieran el segmento A del display encendido.

Es entonces que, los patrones identificados son codificados en lo que se conoce como palabras generadoras de patrones. Las palabras diseñadas utilizan un algoritmo tipo bucle, que primeramente compara el valor de la celda c1 de la pila de datos contra 0, donde el valor de c1 equivale a la cantidad de veces en que el patrón será repetido consecutivamente. Cuando el valor de c1 llegue a 0, se limpiará la pila de datos y terminará la ejecución, en caso contrario, se escribe en los pines del microcontrolador, los valores pertenecientes a la primera mitad del patrón en cuestión, y se mantienen durante $1\ ms$.

Se escriben los valores pertenecientes a la segunda mitad del patrón, manteniéndolos durante $1\ ms$, donde al finalizar esta instrucción, el valor de $c1$ disminuye en 1, y vuelve al inicio del algoritmo. El diagrama de flujo para el algoritmo de ejecución de las palabras se muestra en la Figura 3.



Fuente: elaboración propia

Figura 3 Algoritmo de ejecución para las palabras generadoras de patrones.

En la Figura 4, se presenta la codificación en el metalenguaje Leo del Patrón 1 con el algoritmo, donde este ha sido nombrado en la palabra generadora de patrón *s000100*, y las líneas 3 y 4 son las que cambian los valores de las señales de control

en los pines del microcontrolador. Con las palabras generadoras de patrones codificadas, se crea el diccionario de usuario que se muestra en la Tabla 2, junto al patrón identificado correspondiente.

```
: s000100
 loop { while dup ;
 pines 0 pins! delay
 pines 40h pins! delay
 -- } drop
end
```

Fuente: elaboración propia

Figura 4 Código de la palabra generadora de patrón *s000100*.

Tabla 2 Diccionario de usuario para el SSDS.

Patrón	Palabra	SRCLK	DATOS	RCLK
Patrón 1	*s000100*	0/1	0	0
Patrón 2	*s001101*	0/1	0	1
Patrón 3	*s010110*	0/1	1	0
Patrón 4	*s011111*	0/1	1	1

Fuente: elaboración propia

La codificación de las palabras generadoras de patrones, permite crear a las palabras generadoras de símbolos, donde cada una de estas contiene el orden requerido de los patrones identificados, para reproducir las señales de control necesarias, y así poder desplegar correctamente el símbolo en cuestión. La Figura 5 muestra el código para la palabra generadora de símbolo *d7s5*. Siguiendo estos pasos, se analiza individualmente cada grupo de señales de control, realizando la codificación para el despliegue de los símbolos, permitiendo crear el diccionario de usuario para el control del dispositivo SSDS.

```
: d7s5
1 s000100
2 s010110
1 s000100
2 s010110
1 s000100
1 s010110
1 s011111
end
```

Fuente: elaboración propia

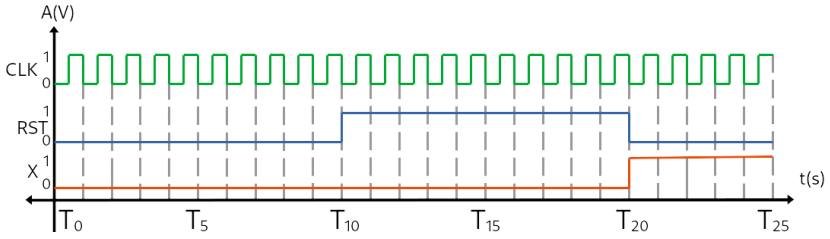
Figura 5 Codificación de la palabra generadora de símbolo *d7s5*.

El diseño de otro metacontrolador, siguiendo la metodología descrita anteriormente:

- **Neurona de impulsos con adaptación de frecuencia**. Para este caso, se diseña un metacontrolador capaz de operar correctamente, una neurona de impulsos implementada en una tarjeta FPGA, la cual produce impulsos con diferente frecuencia, como respuesta a las señales de control.

  Así como el SSDS, la neurona requiere también de tres señales de control: El reloj de sistema CLK, el estímulo X, y la señal de restablecimiento RST.

  Para estabilizar inicialmente al sistema, la señal CLK, debe trabajar durante al menos 10 ciclos de reloj, mientras RST y X se mantienen en nivel BAJO. Una vez estabilizado, la señal RST pasa a un nivel ALTO, durante otros 10 ciclos de reloj. Al finalizar, RST cambia a un nivel BAJO y X a nivel ALTO, manteniéndose así durante 500 ciclos de reloj, tal como se muestra en el diagrama de la Figura 6.



Fuente: elaboración propia

Figura 6 Diagrama de tiempo de las señales de control para la neurona de impulsos.

Al analizar las señales de control descritas, se identifican tres patrones, y de la misma forma como sucede con el dispositivo SSDS, son codificados en palabras generadoras de patrones, tal como se muestra en la Tabla 3, que también señala los periodos de tiempo en los que se presentan en el diagrama.

Tabla 3 Diccionario de usuario para la neurona de impulsos.

Patrón	Palabra	CLK	X	RST	Periodos
Patrón 1	clk	$0/1$	0	0	T_0-T_{10}
Patrón 2	rst	$0/1$	0	1	T_{10}-T_{20}
Patrón 3	x	$0/1$	1	0	T_{20}-T_{25}

Fuente: elaboración propia

Un ejemplo de la codificación de una palabra generadora de patrón para la neurona, se muestra en la Figura 7, describiendo a la palara rst, que reproduce el comportamiento del Patrón 2. Para verificar el correcto funcionamiento de los metacontroladores, se verifican comportamentalmente a través de la herramienta de simulación en EMOS.

```
: rst
 loop { while dup ;
 pines 10h pins! delay
 pines 50h pins! delay
 -- } drop
end
```

Fuente: elaboración propia

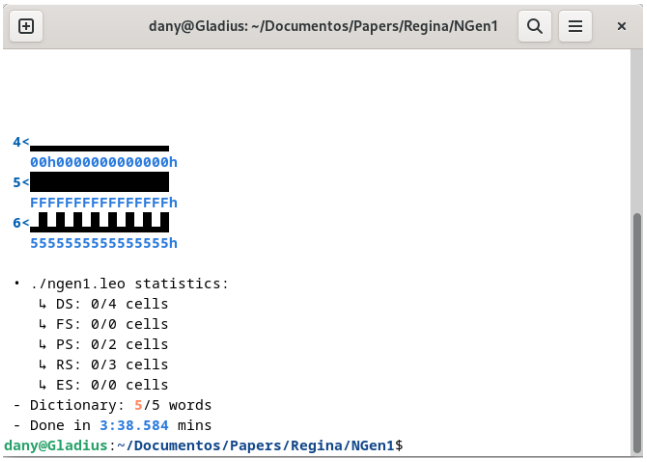
Figura 7 Codificación de la palabra *rst* para la neurona de impulsos.

- **Simulación comportamental – SSDS**. Para el SSDS, se realizaron simulaciones comportamentales de las palabras generadoras de símbolos, corroborando que las señales de control a enviar, son las requeridas para desplegar los símbolos en cuestión. En la Figura 8 se muestra la simulación realizada en EMOS mediante de la terminal de Linux, para el despliegue del símbolo '4', donde las formas de onda 4, 5 y 6 pertenecen a las señales de control RCLK, DATA y SRCLK, respectivamente. Además, la simulación indica un tiempo de ejecución aproximado de $23.228\ ms$.

Fuente: elaboración propia

Figura 8 Simulación con EMOS del símbolo 4 en la terminal de Linux.

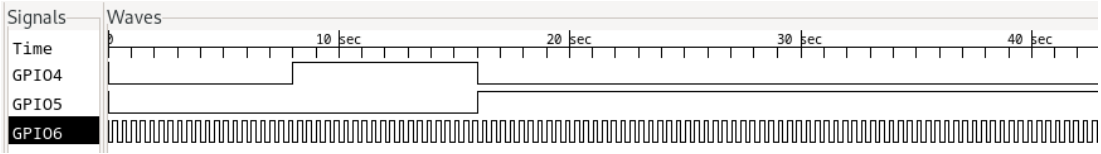
- **Neurona de impulsos con adaptación de frecuencia**. En el caso de la neurona, se verifica que el envío de las señales de control, sean fieles a lo mostrado en el diagrama de la Figura 6. Para ello, se realizan simulaciones comportamentales con el simulador incorporado en EMOS, para validar su correcto funcionamiento, tal como se muestra en la Figura 9, donde las formas de onda 4, 5 y 6, pertenecen a las señales de control RST, X y CLK, respectivamente. Además, es importante señalar que el simulador proporciona un tiempo aproximado de ejecución de $3{:}38\ minutos$.



Fuente: elaboración propia

Figura 9 Simulación en EMOS con tiempo estimado de $3{:}38\ min$.

Debido a que las señales de control requieren un tiempo considerable, se utiliza el archivo VCD generado por el simulador en EMOS, para visualizar las señales mediante GTKWave, tal como se muestra en la Figura 10. Una vez terminadas las simulaciones y su depuración correspondiente, los metacontroladores se implementan, tal como se describe en la siguiente sección.



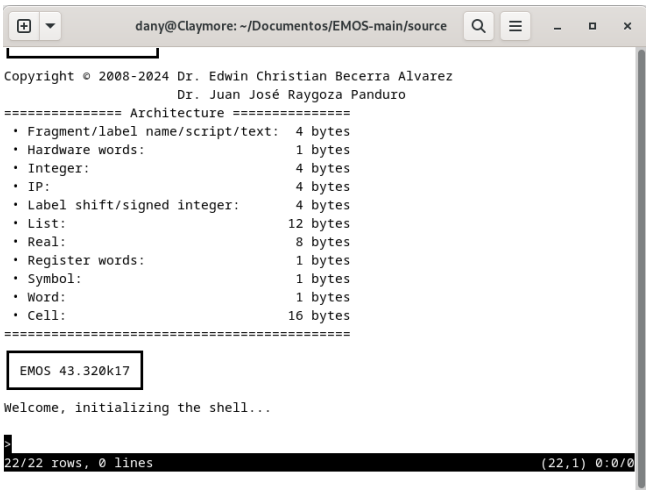
Fuente: elaboración propia

Figura 10 Simulación de la neurona de impulsos visualizada en GTKWave.

3. Resultados

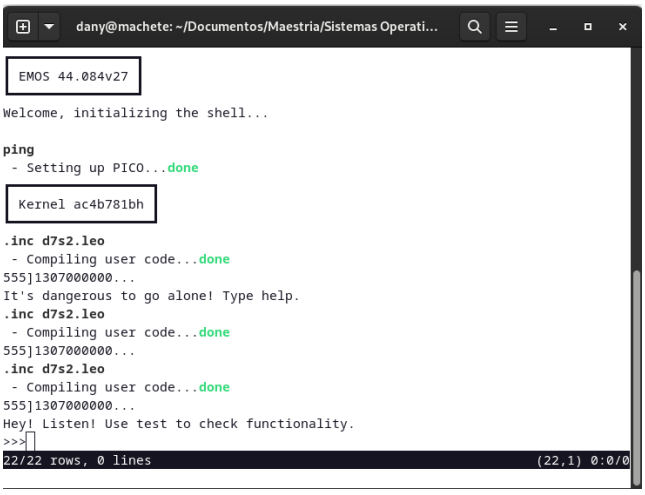
Para implementar el metacontrolador, se requiere de una comunicación entre la computadora y el microcontrolador, para ello se utiliza el comando: "*emos -serial /dev/ttyACMX"*, donde *X* es el número de terminal de conexión con él, permitiendo acceder al Shell de EMOS corriendo en este como se muestra en la Figura 11.

Una vez dentro del Shell de EMOS, los metacontroladores son cargados con el comando "*.inc nombre_del_archivo.leo"*, tal como se muestra en la Figura 12, donde se observa un ejemplo para cargar un programa llamado d7s2.leo.



Fuente: elaboración propia

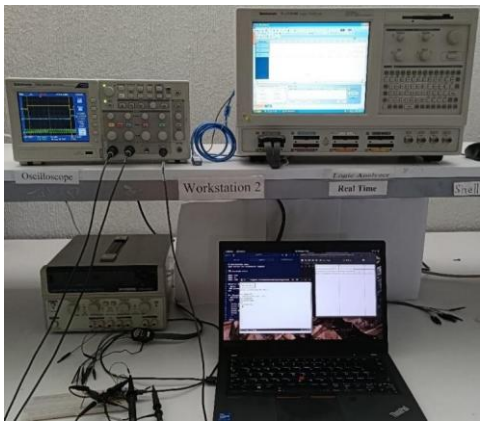
Figura 11 Shell de EMOS a través de la terminal de Linux.



Fuente: elaboración propia

Figura 12 Ejecución del código d7s2.leo en EMOS.

Para realizar mediciones en tiempo real, se utiliza el banco de trabajo mostrado en la Figura 13, el cual cuenta con una fuente de poder GW INSTEK GPS-3303, un osciloscopio Tektronix TDS 2024C, y un analizador lógico Tektronix 5204B.

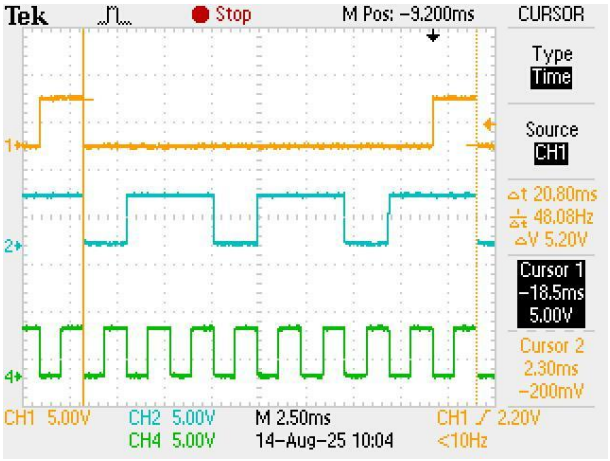


Fuente: elaboración propia

Figura 13 Banco de trabajo para realizar mediciones experimentales en tiempo real.

Seven Segment Display System

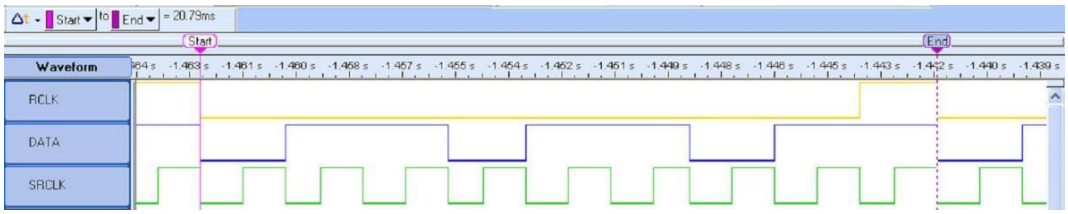
En la Figura 14 se muestra una medición experimental obtenida con el osciloscopio para el despliegue del símbolo 5, donde se observa que el tiempo de transmisión de las señales de control hacia el SSDS, es de $20.8\,ms$ de duración, siendo los canales 1, 2 y 4, las señales de control RCLK, DATA y SRCLK, respectivamente.



Fuente: elaboración propia

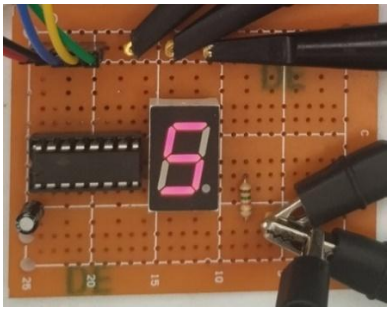
Figura 14 Medición de las señales de control para el símbolo 5 en el osciloscopio.

Para verificar el funcionamiento del metacontrolador, también se realizan mediciones con el analizador lógico, permitiendo validar el comportamiento simulado, tal como se muestra en la Figura 15. Al validar las señales de control, se puede verificar el correcto funcionamiento del metacontrolador (Figura 16), donde el símbolo 5 se despliega en el *display* de 7 segmentos del SSDS.



Fuente: elaboración propia

Figura 15 Señales de control para el símbolo 5 medidas con el analizador lógico.



Fuente: elaboración propia

Figura 16 Metacontrolador desplegando el símbolo 5 en el SSDS.

Las palabras generadoras de patrones se presentan en diversas ocasiones, siendo utilizadas 144 veces. En la Tabla 4, se presenta la frecuencia de uso para cada palabra generadora de patrón en el diccionario de usuario. Una vez verificado experimentalmente en tiempo real el funcionamiento del metacontrolador ahora es necesario hacer lo mismo para la neurona de impulsos.

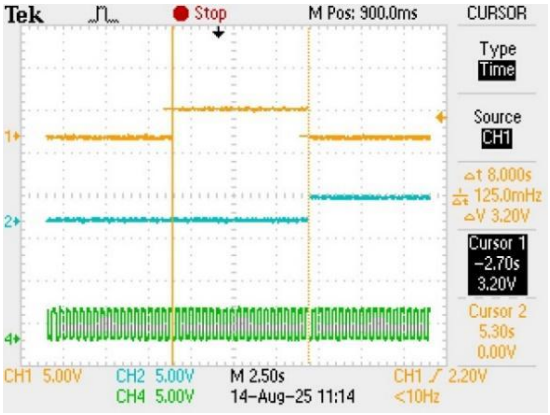
Tabla 4 Frecuencia de uso para las palabras generadoras de patrones – SSDS.

Palabra	Apariciones	Porcentaje
s010110	**76**	**52.77%**
s000100	**52**	**36.11%**
s011111	11	7.64%
s001101	5	3.47%

Fuente: elaboración propia

Neurona de impulsos con adaptación de frecuencia

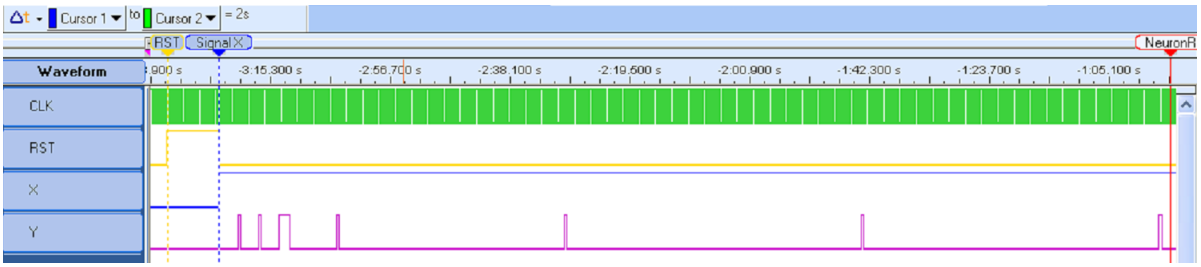
La medición experimental realizada en el osciloscopio para la neurona, se muestra en la Figura 17, donde se pueden observar las señales RST, X y CLK; en los canales 1, 2 y 4; respectivamente. Cabe destacar que los cursores indican un tiempo de $8\ s$ para la duración del Patrón 2, correspondiente a la palabra *rst*.



Fuente: elaboración propia

Figura 17 Medición de las señales de control para la neurona de impulsos.

La Figura 18 muestra la respuesta en tiempo real de la neurona de impulsos a las señales de control, en el analizador lógico, siendo esta la señal mostrada en la parte inferior de la captura.



Fuente: elaboración propia

Figura 18 Medición de la neurona de impulso con el analizador lógico.

Para la neurona, el diccionario del metacontrolador se conforma por las palabras generadoras de patrones *rst*, *clk* y *x*, cuya frecuencia de uso se muestra en la Tabla 5, donde la palabra *x* se utiliza más del 90% de las veces por el metacontrolador, mientras que las palabras *clk* y *rst*, representan menos del 10% de uso.

Tabla 5 Frecuencia de aparición de las palabras para la neurona de impulso.

Palabra	Apariciones	Porcentaje
clk	**20**	**3.70%**
Rst	**20**	**3.70%**
X	500	92.60 %

Fuente: elaboración propia

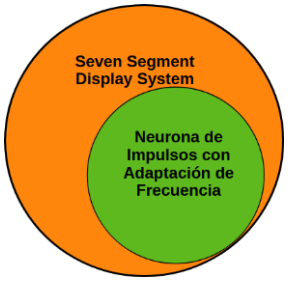
4. Discusión

Cabe destacar que, de los dos dispositivos analizados en este trabajo, el metacontrolador para la neurona de impulsos con adaptación de frecuencia, es el más demandante de ambos, ya que para la neurona se transmite una totalidad de 540 patrones, mientras que para el SSDS solo se requieren 144. Además, la palabra *x* es la que más se utiliza (92.60% de las veces), mientras que la palabra *s010110* presenta un uso de 52.77%. Por lo cual, es importante mencionar que, las palabras creadas en el diccionario para el SSDS, son 100% compatibles para el control de la neurona de impulso, pues se puede observar la equivalencia entre patrones y palabras, tal como se muestra en la Tabla 6. En la Figura 19 se muestra el uso de las palabras utilizadas para el SSDS y la neurona de impulsos, donde el metacontrolador permite el manejo de ambos dispositivos (Figura 1b), requiriendo una menor cantidad de palabras para la neurona de impulso.

Tabla 6 Correspondencia entre las palabras para el SSDS y la neurona de impulso.

Palabras Utilizadas		**Señales de Control**		
SSDS	**Neurona**	**SRCLK/CLK**	**DATA/X**	**RCLK/RST**
s000100	*clk*	**0/1**	**0**	**0**
s010110	*x*	**0/1**	**1**	**0**
s001101	*rst*	0/1	0	1

Fuente: elaboración propia



Fuente: elaboración propia

Figura 19 Distribución de palabras para el metacontrolador.

5. Conclusiones

El análisis de señales de control mediante la metodología propuesta permite la identificación de patrones, los cuales a su vez son la base para crear las palabras que forman al metacontrolador. En algunos casos, como es del SSDS, también es necesario crear otras palabras más complejas que hacen uso de las palabras generadoras de patrones (palabras generadoras de símbolos).

La simulación comportamental del metacontrolador permite reducir el tiempo de desarrollo al verificar y depurar las palabras del diccionario.

Las mediciones experimentales en tiempo real de los metacontroladores, en los dispositivos validan el correcto funcionamiento, demostrando que la metodología propuesta es viable.

Finalmente, la propuesta de diseñar un metacontrolador que controle diversos dispositivos, se verifica tanto con simulaciones comportamentales como con mediciones experimentales en tiempo real, permitiendo la reducción de ocupación de memoria en sistemas embebidos.

6. Referencias y Bibliografía

[1] Abbasi, A., Wetzels, J., Holz, T., Etalle, S., (2019). Challenges in Designing Exploit Mitigations for Deeply Embedded Systems, Proc. of the IEEE European Symposium on Security and Privacy, pp. 31 – 46. doi: 10.1109/EuroSP.2019.00013.

[2] Abuazoum, S., (2025). A Comparative Analysis of VxWorks OS and Windows CE in Embedded Real-Time Applications, Wadi Alshatti University Journal of Pure and Applied Sciences, Vol. 3, Issue 2, 2025. doi: 10.63318/waujpasv3i2_02.

[3] Amos, B., (2020). Hands-On RTOS with Microcontrollers: Building real-time embedded systems using FreeRTOS, STM32 MCUs, and SEGGER debug tools. Packt Publishing. ISBN: 978-1-83882-673-4.

[4] Becerra Alvarez, E. C., Raygoza Panduro, J. J., Rivera Dominguez, J., Ortega Cisneros, S., González Vidal, J. L., (2023). ForEmb: A Forth-Inspired, Real Time Interpreter for Embedded Systems, Proc. of the 12th International Conference

On Software Process Improvement (CIMPS), pp. 215-224, Cuernavaca, México. doi: 10.1109/CIMPS61323.2023.10528835.

[5] Becker, M., Di Guglielmo, G., Fummi, F., Mueller, W., Pravadelli, G. and Xie, T., (2010). RTOS-aware refinement for TLM2.0-based HW/SW designs, Proc. of the 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010), Dresden, Germany, pp. 1052-1058. doi: 10.1109/DATE.2010.5456965.

[6] Brodie, L., (2004). Thinking FORTH, Creative Commons. ISBN: 0-9764587-0-5.

[7] Cingel, M., Novak, M. and Fryza, T., (2017). Characteristics of SPI drivers in RTOS environment, Proc. of the 2017 27th International Conference Radioelektronika, Brno, Czech Republic, pp. 1-6. doi: 10.1109/RADIOELEK.2017.7937586.

[8] Embedded Metalanguage Operating System, (2025). Accessed on: 12/08/2025. Available on: https://github.com/labcibernetica/EMOS.

[9] Galeano, G., (2009). Programación de Sistemas Embebidos en C, Alfaomega, 2009. ISBN: 9789586827706.

[10] Hallinan, C., (2010). Embedded Linux Primer: A Practical Real-World Approach, 2nd Ed., Pearson Education. ISBN: 978-0-13-167984-9.

[11] Holt, A., Huang, C., (2018). Embedded Operating Systems: A Practical Approach, 2nd Ed., Springer International Publishing AG. doi: 0.1007/978-3-319-72977-0.

[12] Jadhav, M. S., (2014). Easy Linux Device Driver, 2nd Ed., HighTechEasy Publishing. ISBN WWH48Y3155N.

[13] Kaptain, R. J., Helle, T., Larsen, S. M., (2024). Everyday technology and assistive technology supporting everyday life activities in adults living with COPD – a narrative literature review. Disability and Rehabilitation: Assistive Technology, Vol. 20, No.4, 742–756. doi: 10.1080/17483107.2024.2431627.

[14] Kernighan, B. W., Ritchie, D. M., (1991). El Lenguaje de Programación C, 2da Ed., Prentice-Hall Hispanoamérica S. A. ISBN: 979-8477795994.

[15] Lutenberg, A., Gomez, P., Pernia, E., (2022). A Beginner's Guide to Designing Embedded System Applications on ARM Cortex-M Microcontrollers, Arm Education Media. ISBN: 978-1911531418.

[16] Mano, M. M., Kime, C. R., Martin, T., (2015). Logic and Computer Design Fundamentals, 5th Ed., California State University, Pearson Higher Education. ISBN: 978-0134080123.

[17] Mecrisp stellaris unofficial userdoc, (2025). Accessed on: 12/08/2025. Available on: https://mecrisp-stellaris-folkdoc.sourceforge.io/.

[18] Nawaz, S., Mitchell, M., Linden, T., Bhowmik, J., (2025). Exploring smartphone usage patterns and perceived dependency: Across-sectional study in Australia. Acta Psychologica, Vol. 257, 105100. doi: 10.1016/j.actpsy.2025.105100.

[19] Pérez, D. A., (2009). Lecturas en Ciencias de la Computación – Sistemas Operativos Embebidos, Centro de Investigación en Comunicaciones y Redes (CICORE), Caracas. ISSN: 1316-6239.

[20] Phung, S., Jones D., Joubert, T., (2011). Professional Windows Embedded Compact 7, Wiley. ISBN 978-1118050460.

[21] Pont, M. J., (2022). Embedded C, Pearson Education Limited. ISBN: 020179523X.

[22] Raspberry Pi Pico Datasheet: An RP2040-based microcontroller board, Raspberry Pi Ltd, 2020.

[23] Rico, R., Rico-Azagra, J. and Gil-Martínez, M., (2022). Hardware and RTOS Design of a Flight controller for Professional Applications, in IEEE Access, vol. 10, pp. 134870-134883. doi: 10.1109/ACCESS.2022.3232749.

[24] Schild, U. J., Herzog, S., (1993). The Use of Meta-Rules in Rule Based Legal Computer Systems, Proc. of the 4th International Conference on Artificial Intelligence and Law (ICAIL), pp. 100 – 109, Association for Computing Machinery (ACM). doi: 10.1145/158976.15898

[25] Srirengan, K., (2006). UNDERSTANDING UNIX, 7th Ed., Prentice-Hall. ISBN: 9788120314894.

[26] Vahid, F., Givargis, T., (1999). Embedded System Design: A Unified Hardware/Software Approach, University of California, 1999. ISBN: 978-0-471-38678-0.