
UNIVERSIDAD AUTÓNOMA DE GUADALAJARA

CON RECONOCIMIENTO DE VALIDEZ OFICIAL DE ESTUDIOS DE LA SECRETARÍA DE
EDUCACIÓN PÚBLICA SEGÚN ACUERDO No.158 DE FECHA 17 DE JULIO DE 1991

POSGRADO E INVESTIGACIÓN



Rastreo de eventos en sistemas embebidos con interfaz al usuario utilizando código
Morse y código de Huffman

TESIS

QUE PRESENTA

DANIEL RUBIO MARTÍN DEL CAMPO

PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS COMPUTACIONALES

DIRECTOR DE TESIS:

Dra. Lina Maria Aguilar Lobo

GUADALAJARA, JALISCO. 10 de Diciembre de 2023

TTS-03-185-0734-4060

Índice general

1. Introducción	15
1.1. Descripción del problema	15
1.2. Definición del problema	16
1.3. Preguntas de Investigación	16
1.4. Objetivo General	17
1.5. Objetivos Específicos	17
1.6. Hipótesis	17
1.7. Delimitaciones	18
1.8. Justificación	18
2. Estado del Arte y Bases Teóricas	19
2.1. Marco Teórico	19
2.1.1. Código Morse	19
2.1.2. Rastreo de eventos en sistemas embebidos	21
2.1.3. Intrusión en software	22
2.1.4. Representación de datos “In Situ”	23
2.1.5. Teoría de la información	23
2.1.6. Compresión de datos	23
2.1.7. Código de Huffman	24
2.2. Marco Referencial	25
2.2.1. Metodología de rastreo eficiente mediante un autómata programable.	26
2.2.2. Rastreo de programa sin intrusión en sistemas multi tarea usando consumo de corriente eléctrica.	27

2.2.3.	Diagnóstico de fallas de motor en sitio mediante el uso de una red neuronal convolucional.	28
2.2.4.	Monitoreo de requerimientos de sistemas embebidos sin intrusión y en sitio.	29
2.2.5.	Sistema de entrada para código Morse para personas con discapacidad.	30
2.2.6.	Utilización de código Huffman para sistema de boyas de clima en el mar.	31
3.	Diseño e Implementación	33
3.1.	Descripción de la propuesta	33
3.2.	Selección de alfabeto de entrada	34
3.3.	Diseño de Hardware	35
3.4.	Diseño de Software	37
3.4.1.	Máquina de estado finita	39
3.5.	Tabla de representación Morse	43
3.6.	Generación de árbol binario de Huffman	45
3.7.	Mensajes en código de Huffman	48
4.	Resultados y Discusión	53
4.1.	Resultados	53
4.1.1.	Entorno de validación	53
4.1.2.	Casos de prueba	54
4.1.3.	Relación variables dependientes e independientes	55
4.1.4.	Limitaciones	55
4.1.5.	Población y muestra	55
4.1.6.	Funcionalidades o requisitos	56
4.1.7.	Validación	56
4.1.8.	Pruebas unitarias	58
4.1.9.	Configuración de dispositivos	60
4.1.10.	Tabla de resultados	62
4.2.	Discusión	63

5. Conclusiones y Perspectivas	66
5.1. Conclusiones	66
5.2. Perspectivas	67
Anexos	71
A. Códigos	71
A.1. morsehuffman_lib.h	71
A.2. morsehuffman_lib.c	71
A.3. morsehuffman_cfg.h	81
A.4. morsehuffman_hal.h	81
A.5. morsehuffman_hal.c	82
A.6. Algoritmos y Pseudocódigos	82
A.6.1. Funciones privadas	82
A.6.2. Funciones publicas	82

Índice de figuras

2.1. Árbol binario de código de Huffman resultante.	25
3.1. Esta figura muestra el ejemplo de un generador de energía eólico visto en dos escenarios distintos.	33
3.2. Capa de abstracción de hardware conocida como HAL.	36
3.3. Se muestra la configuración a nivel hardware para opción 1 y opción 2.	36
3.4. Ejemplo de una posible implementación a nivel hardware.	36
3.5. En esta imagen se puede ver la estructura de la librería propuesta respecto a los archivos que la componen.	38
3.6. A nivel software la implementación requerida es la misma tanto para la opción 1: Método Morse únicamente, como para la opción 2: Método Morse + Huffman.	38
3.7. En este diagrama se muestra el diseño de la máquina de estados principal que debe ser llamada a través de su API en una tarea cíclica. . .	40
3.8. Se utilizan colores para facilitar la identificación de cada tipo de símbolo, amarillo para “puntos”, verde para “líneas” y el naranja para provocar dos 0 consecutivos que marcan el final del caracter. . .	44
3.9. Árbol binario de Huffman resultante dado el alfabeto de entrada definido.	47
3.10. Diagrama de flujo de la función estática “huffmanprint_func” la cual se llama cada que hay un cambio de flanco en la salida que transmite los mensajes en Morse.	49
4.1. Entorno de validación una computadora, un analizador lógico, una “protoboard”, un LED y un microcontrolador, en este caso se muestra el ATmega32U4 desde una placa Arduino UNO.	54

4.2.	Toma de pantalla del analizador lógico de la transmisión del caracter “Q” usando el código Huffman “11100001”	57
4.3.	Toma de pantalla del contenido de un archivo “digital.csv”	57
4.4.	Toma de pantalla de un ejemplo del contenido de un archivo “morsehuffman_translation.txt” generado con el código python “CSVtranslate.py”	58
4.5.	Diagrama para pruebas de comunicación Morse y Huffman usando un Arduino UNO, una placa de pruebas y un analizador lógico.	61
4.6.	Conexión de Arduino UNO con carcasa protectora, placa de pruebas, LED de color azul y un analizador lógico.	61
4.7.	LED de color azul encendido para representar un pulso Morse.	62

Índice de tablas

2.1. Tabla de composición del código Morse	19
2.2. Tabla de representación Morse de caracteres	20
2.3. Tabla de resultados de códigos Huffman del ejemplo	25
3.1. Tabla de caracteres elegidos como parte del alfabeto de entrada. . . .	35
3.2. Tabla de valores hexadecimales de cada caracter basado en su repre- sentación Morse.	45
3.3. Tabla alfabeto de entrada del árbol Huffman con el peso de cada caracter.	46
3.4. Tabla de códigos de Huffman resultantes en representación binaria y hexadecimal.	48
3.5. Tiempos para representar cada caracter tomando como unidad de tiempo $1 \mu\text{S}$	50
3.6. Tabla con los códigos binarios reordenados por número de bits y por duración de tiempo luego del análisis de tiempos de la tabla 3.5, usando la misma referencia de $1 \mu\text{S}$ como unidad de tiempo.	52
4.1. Parámetros de configuración usados en cada hardware.	60
4.2. Tabla comparativa de resultados de las diez pruebas descritas en la sección 4.1.8 en los diferentes hardware donde se probó esta biblioteca. . . .	63

Abstract

Diseño e implementación de una librería de código en lenguaje C para ser implementada en sistemas embebidos con mínima configuración independiente del hardware y utilizando únicamente 1 GPIO configurado como salida. Una vez implementada la librería permite al sistema embebido emitir mensajes a través de 2 métodos; El primer método es utilizando código Morse con lo cual la interfaz es directamente desde el sistema embebido al ojo humano sin necesidad de hardware adicional para recibir el mensaje, aunque este método tiene las ya mencionadas ventajas de evitar la dependencia de dispositivos de lectura, por la naturaleza de la transmisión del código Morse la velocidad de emisión del mensaje es lenta, para permitir al humano poder interpretar el mensaje, por tanto este método no sería efectivo cuando la transmisión de datos debe ser de manera prácticamente inmediata, como por ejemplo para transmitir la causa de un reinicio inesperado precisamente antes de que el sistema se reinicie, pensando en cubrir estos casos se implemento el segundo método que consiste en transmitir sobre el mismo GPIO códigos de Huffman en cada cambio de flanco de la salida del microcontrolador cuando transmite mensajes en Morse e incluso sin no hay caracteres transmitiéndose el primer método, la transmisión de código Huffman sucede tan rápido (menos de 1 microsegundo) que no sería perceptible por el ojo humano en caso de que la salida configurada, por tanto este segundo método si requiere forzosamente ser recibido por un analizador lógico que más tarde podrá ser descifrado con un script de Python, dicho script también está disponible en el repositorio de esta librería junto con toda la documentación.

AUTORIZACIÓN DE PROYECTO Y DIRECTOR DE TESIS

Zapopan, Jalisco a 07 de septiembre del 2023

Daniel Rubio Martin Del Campo
Estudiante de Maestría en Ciencias Computacionales

Dra. Karina Aguilar Moreno
Directora de Posgrados

Mtro. Gerardo Adrián Martínez Fernández
Director de Departamento de Computación e Industrial

PRESENTE

Por medio del presente le notifico que la Dirección de Investigación y Desarrollo Tecnológico ha recibido el protocolo que respalda el Tema de Tesis **Rastreo de eventos en sistemas embebidos con interfaz al usuario utilizando código Morse y código de Huffman** y le asigna el ID: **TTS-03-185-0734-4060**.

Además, se reconoce como Director de Tesis a **Dra. Lina María Aguilar Lobo** adscrito al Departamento de **Computación e Industrial** de la **Universidad Autónoma de Guadalajara**.

Atentamente

“Ciencia y Libertad”



Dr. Efrén Aguilar Garnica
Director de Investigación y Desarrollo Tecnológico

Glosario

μ s Un microsegundo es la millonésima parte de un segundo.. 7, 50, 51, 52, 56, 62, 63, 65

API application programming interface, traducido al español como interfaz de programación de aplicaciones.. 5, 39, 40

ARM Advanced RISC Machine.. 26

ASCII American Standard Code for Information Interchange, traducido al español como código estándar estadounidense para el intercambio de información.. 39, 40, 49

C Lenguaje de programación de paradigma estructurado, genera archivos de extensión .c y .h. 64

Caché L2 La caché L2 comparada con la L1 tiene mayor capacidad de almacenamiento, aunque será un poco más lenta, de unos 470 GB/s y 2,8 ns de latencia. El tamaño de almacenamiento suele variar entre los 256 KB y los 18 MB.. 26

Caché L1 La caché L1 es la configuración más rápida, la que se encuentra más cerca de los núcleos. Ésta almacena los datos que inmediatamente van a ser usados por la CPU, y es por ello que las velocidades están en torno a los 1150 GB/s y la latencia en tan solo 0,9 ns.. 26

FPGA Field-programmable gate array.. 26, 29, 30

GB Un gigabyte es una unidad de almacenamiento de información estandarizada y se suele utilizar en el ámbito computacional. Esta unidad equivale a mil millones de bytes.. 26

Gem5 El simulador gem5 es un simulador de procesador y nivel de sistema de código abierto.. 26

GHz El gigahercio es un múltiplo de la unidad de medida de frecuencia hercio y equivale a mil millones de Hz.. 26

GPIO General Purpose Input Output, traducido al español como entradas y salidas de propósito general.. 8, 16, 37, 39, 60

HAL Siglas en inglés de Hardware Abstraction Layer, en español Capa de Abstracción de Hardware.. 5, 36, 64

kB Un kilobyte es una unidad de almacenamiento de información cuyo símbolo es el kB (con la 'k' en minúsculas) y equivale a mil bytes.. 26

LED light-emitting diode, traducido al español como diodo emisor de luz.. 5, 6, 34, 37, 41, 42, 43, 53, 54, 60, 61, 62, 66

MB El megabyte es una unidad de información. Es múltiplo del byte y equivale a un millón de bytes.. 26

MHz El megahercio es un múltiplo de la unidad de medida de frecuencia hercio y equivale a 1 millón de Hz.. 27

NIRM non-intrusive in-situ requirements monitoring.. 29

ohm Resistencia eléctrica que existe entre dos puntos de un conductor cuando una diferencia de potencial constante de 1 voltio, aplicada entre estos dos puntos, produce, en este conductor, una corriente de 1 amperio.. 37, 53

RMG Runtime Monitoring Graph, traducido al español como grafo de monitoreo en tiempo de ejecucion.. 29

SoC system on a chip, traducido al español como sistema en chip.. 29, 30

UML Unified Modeling Language, traducido al español como lenguaje unificado de modelado.. 29, 30

CARTA IMPRIMATUR

Zapopan, Jalisco a 1 de Noviembre del 2023

Daniel Rubio Martín del Campo
Maestría en Ciencias Computacionales

PRESENTE

Por medio del presente le notifico que después de que su Tesis titulada: **Rastreo de eventos en sistemas embebidos con interfaz al usuario utilizando código Morse y código de Huffman.**

Ha sido revisada por su Director de Tesis Dra. Lina María Aguilar Lobo y por los LECTORES Dr. Ulises Davalos Guzmán y Dr. Gerardo Adrián Martínez Fernández, se APRUEBA su contenido y su impresión.

Además, se recomienda iniciar con los trámites para llevar a cabo el EXAMEN RECEPCIONAL para obtener el título de Maestría en Ciencias Computacionales dado que la tesis aprobada forma parte de los requisitos para alcanzar dicho título.

Les solicito de la manera más atenta que incluya una copia de esta Carta Imprimatur en los preliminares de todo ejemplar de su tesis electrónica o impresa.



Atentamente
"Ciencia y Libertad"

Mtro Alejandro Martín Solís Tenorio
Director de Posgrados de Electrónica e Informática

Agradecimientos

Agradezco a mi familia, por la paciencia de permitirme estudiar en tiempos tan complejos como lo ha sido una pandemia.

A mis padres por mostrarme el camino de que siempre se puede llegar a más.

Siempre estaré agradecido con Alan Blanco por motivarme a estudiar esta maestría.

Agradezco también a la Universidad Autónoma de Guadalajara por abrirme sus puertas y a todos sus profesores que siempre fueron ejemplares, en particular a mis directores de tesis, Dr. Ulises Davalos y la Dra. Lina Aguilar.

Finalmente agradezco al CONACYT por la confianza depositada.

Dedicatoria

Dedico este trabajo a mi hijo Leonardo Daniel que ahora puede que se sienta obligado a estudiar una maestría porque dirá en tono despectivo: "¡Hasta mi papá tiene una maestría!". Perdón hijo, todo fue gracias a tu mamá.

Capítulo 1

Introducción

1.1. Descripción del problema

Existen sistemas embebidos que no cuentan con un medio para comunicar eventos durante el tiempo de ejecución, principalmente proyectos académicos que trabajan con microcontroladores (un brazo robótico, un prototipo de caja fuerte inteligente, etc) o sistemas de la industria que en su diseño no se contempla esta característica (como pudiera ser un módulo de control de carrocería de un automóvil, un reproductor de audio, etc).

Esta limitante dificulta conocer información que puede ser muy útil al momento de que una falla en el funcionamiento del sistema ocurre, pudiendo ser una ejecución normal del sistema o la verificación de una prueba de tipo caja negra si es que aplica para el sistema embebido en cuestión. Al ocurrir una falla es útil conocer información de eventos del sistema tales como el tipo de error detectado o el ultimo estado de ejecución del sistema antes de que la falla se presentara. La alternativa usual entonces es utilizar una herramienta de hardware que permita la conexión del microcontrolador a la computadora para poder correr la prueba que ha fallado a través de una sesión de depuración y así conocer los detalles de la causa del error, esto implica la intervención de hardware adicional como es el caso de la herramienta de conexión al microcontrolador y la computadora misma, así como la configuración y en caso de aplicar la instrumentación necesarias.

El no contar con un medio de comunicación de eventos en tiempo de ejecución implica un mayor esfuerzo y más tiempo para detectar errores durante la fase de

desarrollo y la dependencia de adquirir y tener disponible hardware adicional para mitigar esta limitante a través de pruebas de caja blanca.

Desde el punto de vista económico, en ambientes de la industria, la adquisición de hardware adicional puede implicar el aumento de costos de un proyecto y en ambientes académicos un gasto para el estudiante que puede no ser redituable.

Desde el punto de vista de recursos de hardware, los microcontroladores tienen un numero limitado de GPIO por lo que dedicar mas de 1 salida para tener un medio de monitorio de eventos del sistema embebido puede comprometer la viabilidad del propio medio de monitoreo o incluso del sistema embebido.

1.2. Definición del problema

Para propósitos de desarrollo de sistemas embebidos o proyectos similares (académicos o industriales) no esta disponible una biblioteca de código con las siguientes características:

- Gratuita.
- De baja complejidad de implementación
- Bajo nivel de intrusión
- Permita conocer los eventos de la ejecución del propio sistema que pudiera ser interpretado directamente In situ.
- Permita que el hardware adicional dedicado sea opcional.
- Ofrezca una solución escalable para cubrir diferentes escenarios de necesidades.

1.3. Preguntas de Investigación

¿Se puede lograr una comunicación de eventos de un sistema embebido directamente al humano sin la necesidad de un hardware adicional?

¿Se puede lograr una comunicación de eventos de un sistema embebido rápida y confiable utilizando solo 1 GPIO?

¿Se puede cubrir los escenarios donde sea necesario la transmisión de datos de manera prácticamente instantánea utilizando la misma configuración de hardware?
¿Esta herramienta puede funcionar en diferentes plataformas de hardware?

1.4. Objetivo General

Desarrollar una biblioteca de código gratuita para monitoreo en sitio de eventos durante la ejecución de sistemas embebidos, que pueda ser interpretado directamente por un ser humano, de baja complejidad, bajo costo y bajo nivel de intrusión en el sistema, que cubra todos los escenarios para las necesidades de comunicación de eventos de los sistemas embebidos.

1.5. Objetivos Específicos

- Investigar el estado del arte sobre las maneras más usuales de rastrear eventos de ejecución de sistemas embebidos en tiempo real.
- Definir un método de comunicación de eventos en tiempo de ejecución de sistemas embebidos que cumpla con el objetivo general 1.4.
- Diseñar una biblioteca de bajo acoplamiento de Hardware mediante el uso de una capa de abstracción.
- Implementar la biblioteca diseñada.
- Comprobar la funcionalidad de la biblioteca en sistemas embebidos reales.
- Medir y documentar los resultados de la biblioteca en términos de velocidad de y fiabilidad de la transmisión de datos.
- Publicar la biblioteca y su documentación en una plataforma pública.

1.6. Hipótesis

Una librería de código gratuita que permita el monitoreo de sistemas embebidos en sitio sin la necesidad de un hardware adicional puede ser muy útil para

el desarrollo de sistemas embebidos en la industria y para proyectos académicos, particularmente para el diagnóstico y corrección de fallos.

1.7. Delimitaciones

El presente proyecto abarca el diseño e implementación de una biblioteca de código en lenguaje C. Aunque la lógica puede ser adaptada a otros lenguajes de programación, su implementación en lenguajes diferentes de C esta fuera del alcance de este trabajo.

En la fase de pruebas se considera utilizar diferentes plataformas de Hardware como lo son Arduino, un Pic de la empresa Microchip, un ESP8266 MCU y una Raspberry PI, mas siendo una biblioteca de código, su uso no se limita a estas plataformas, pero esta fuera del alcance de este trabajo el probarlo con otras plataformas especificas.

1.8. Justificación

Una biblioteca de una herramienta de monitoreo de sistemas embebidos en sitio permitiría conocer eventos durante la ejecución de un sistema embebido sin la necesidad de un hardware específico, siendo interpretada directamente por el humano, sera útil para proyectos industriales que por cualquier motivo carezcan de esta característica, tanto en la fase de desarrollo como en la fase de despliegue del producto y particularmente en proyectos académicos al evitar altos gastos en hardware dedicado para este propósito.

Adicionalmente gracias a que se podría obtener los eventos del sistema en cualquier sitio se podría conocer dichos eventos en cualquier entorno de ejecución, incluido el entorno final de uso del sistema, permitiendo un apropiado diagnóstico de fallas por entorno, sin la necesidad de trasladar el sistema embebido que presenta las fallas a un ambiente controlado como un banco de pruebas o un laboratorio donde clásicamente se encuentran las herramientas necesarias para ejecutar este tipo de diagnósticos, beneficiando así el proceso de desarrollo así como potencialmente la atención al cliente/usuario final.

Capítulo 2

Estado del Arte y Bases Teóricas

2.1. Marco Teórico

2.1.1. Código Morse

El código Morse es un esquema de codificación unidimensional binario para lenguaje donde cada caracter es representado por una secuencia única de pulsos y espacios, habiendo 2 tipos de pulsos diferentes y 3 tipos de espacios distintos. Como se puede observar en la tabla 2.1 tanto los pulsos como los espacios se distinguen entre sí únicamente por su tiempo relativo de duración ([Guenther, 1973](#)).

El valor de la unidad de tiempo de duración es un valor fijo pero que puede tener cualquier valor que convenga, considerando que entre más largo sea este valor, mayor será el tiempo para transmitir cualquier mensaje, pero puede que ayude a ser interpretado con mayor facilidad por un ser humano.

Unidades de tiempo de duración	Pulsos	Espacios
1	Punto (●)	Símbolo
3	Línea (-)	Caracter
5-7		Palabra

Tabla 2.1: Tabla de composición del código Morse

Los caracteres básicos del idioma Inglés en código Morse ([M.1677-1, 2009](#)) se muestran en la siguiente tabla 2.2:

Caracter	Código Morse	Caracter	Código Morse
A	• -	0	- - - - -
B	- • • •	1	• - - - -
C	- • - •	2	• • - - -
D	- • •	3	• • • - -
E	•	4	• • • • -
F	• • - •	5	• • • • •
G	- - •	6	- • • • •
H	• • • •	7	- - • • •
I	• •	8	- - - • •
J	• - - -	9	- - - - •
K	- • -	.	• - • - • -
L	• - • •	,	- - • • - -
M	- -	:	- - - • • •
N	- •	?	• • - - • •
O	- - -	'	• - - - - •
P	• - - •	-	- • • • • -
Q	- - • -	/	- • • - •
R	• - •	(- • - - •
S	• • •)	- • - - • -
T	-	“ ”	• - • • • -
U	• • -	=	- • • • -
V	• • • -	+	• - • - •
W	• - -	*	- • • -
X	- • • -	@	• - - • - •
Y	- • - -		
Z	- - • •		

Tabla 2.2: Tabla de representación Morse de caracteres

A pesar de todos los métodos modernos de comunicación que existen actualmente, el uso de código Morse aún cuenta con las siguientes ventajas ([Carron, 1991](#)):

- Es el medio de signos más ampliamente reconocido en el mundo.
- Es el único código comprendido por humanos y máquinas.
- Es el único código permitido en cualquier frecuencia de telecomunicaciones.
- Es un método de comunicación que puede ser comprendido donde otros medios fallan, gracias a la diversidad de medios en el que puede ser transmitido.
- Frecuentemente es el único tipo de señal que puede ser leído a pesar de la interferencias.

La comunicación de código Morse puede ser percibida por todos los sentidos humanos, aunque algunos pueden ser más apropiados para ser usados para transmitir e interpretar estas señales, como pueden ser la vista, el oído y el tacto, esto lo convierte en un medio económicamente muy accesible, por ejemplo: para transmitirle un mensaje, bastaría con usar un dedo y presionar la piel de otra persona, usar cualquier fuente de sonido o cualquier medio que permita percibir un cambio visual como puede ser una mano tapando la luz del sol.

2.1.2. Rastreo de eventos en sistemas embebidos

Es una técnica comúnmente usada para depuración de código y análisis de desempeño de software. Concretamente, implica la detección y almacenamiento de eventos relevantes durante el tiempo de ejecución del sistema para un posterior análisis. Está enfocada a sistemas computacionales embebidos. Esta técnica se puede realizar en diferentes capas de abstracción y puede ser llevada a cabo usando soluciones de software mediante código de instrumentación, a través de soluciones de hardware como un dispositivo diseñado especialmente para esta tarea o incluso de manera híbrida hardware/software ([Kraft et al., 2010](#)).

Para el registro de trazos mediante software se usa código de instrumentación que va a requerir memoria y recursos de tiempo de ejecución del sistema. Este código de instrumentación tendrá una prueba-efecto, esto se refiere a que si el código de instrumentación se agrega o se remueven, habrá un impacto en el desempeño final

del sistema. La solución a esto es agregar esta técnica como parte del diseño del programa, considerando su tiempo de ejecución, reservándole memoria ([Thane, 2000](#)) y hacer uso del análisis de tiempo de ejecución ([Puschner and Koza, 1989](#)) y teoría de planificación de tareas ([Audsley et al., 1995](#); [Liu and Layland, 1973](#); [Xu and Parnas, 1990](#)).

Esta técnica es especialmente útil en el proceso de depuración de código en sistemas donde las entradas y salidas no deben interrumpirse deteniendo la ejecución del software en una línea determinada, por ejemplo: en sistemas que controlan un motor, pausar la ejecución del software causará que la señal que controla el motor se interrumpa y éste pierda su fuerza de torque ([Gracioli and Fischmeister, 2009](#)).

Para utilizar esta técnica en un sistema de una sola tarea se debe identificar y registrar eventos significativos en la ejecución secuencial del programa, para esto será importante acceder al reloj local del sistema para usar su información con el fin de ordenar los eventos registrados en caso de ser necesario. Para sistemas multitarea sería lo mismo que ya se ha descrito solo que se debe añadir también como evento significativo el entrelazamiento de las tareas ([Thane, 2000](#)).

El uso de código de instrumentación puede llegar a causar problemas de intrusión en el software.

2.1.3. Intrusión en software

La fuente principal de intrusión en software es la ejecución de instrucciones adicionales. La intrusión en el software se puede manifestar de diversas maneras:

- Reducción de velocidad de ejecución.
- Cambios en los patrones de referencia de la memoria.
- Diferente orden de eventos.
- Interbloqueo

Desde el punto de vista de la evaluación del desempeño, la instrumentación debe estar balanceada entre no causar intrusión y ofrecer información valiosa ([Malony](#)

[et al., 1992](#)).

2.1.4. Representación de datos “In Situ”

La frase “In Situ” viene del latín y significa “En sitio” ([Lewis and Short, 1879](#)). Como concepto de representación de datos en ciencias computacionales se tienen las siguientes definiciones:

- Acceder a datos situacionalmente apropiados en el tiempo y lugar ideales ([Ens and Irani, 2016](#)).
- La visualización de datos está embebida al mundo real, al entorno físico ([Moere and Hill, 2012](#)).
- Una visualización que está relacionada con el entorno y está basada en la relevancia de los datos al contexto físico ([White, 2009](#)).

El obtener datos de manera oportuna suele ser determinante para encontrar fallos en sistemas de tiempo real, donde su desempeño puede estar sujeto al entorno donde se encuentra trabajando el sistema. Un ejemplo podría ser el caso hipotético de un sistema embebido que presenta una falla en casa del usuario final pero la falla no se puede reproducir en el taller de servicio al cliente. En este caso sería importante recolectar la información de la falla en el contexto donde la falla se presenta.

2.1.5. Teoría de la información

Se define como el estudio de la codificación eficiente y sus consecuencias en términos de velocidad de transmisión y probabilidad de error ([Ingels, 1971](#)).

2.1.6. Compresión de datos

La compresión de datos puede verse como una rama de la teoría de la información en el cual el objetivo principal es minimizar la cantidad de datos para ser transmitidos ([Lelewer and Hirschberg, 1987](#)).

2.1.7. Código de Huffman

En 1951 David Huffman presentó su algoritmo para determinar “códigos de costo mínimo y libres de prefijos”, al que actualmente se le conoce como “código de Huffman”.

Este algoritmo trabaja en el área de la representación de datos y se enfoca en mostrar maneras eficientes de representar información comprimida.

El concepto parte de tener un alfabeto de entrada con todos los símbolos que se pretenden incluir en el código a desarrollar. Cada elemento del alfabeto debe tener un “peso” que será directamente proporcional al número de apariciones de dicho símbolo en la información que se pretende comprimir, de esta manera el símbolo que más se repita deberá tener el peso máximo y los caracteres que menos se repitan el peso mínimo, de ahí el resto de símbolos deberán tener un peso proporcional. Con estos datos de entrada se genera un alfabeto de salida basado en símbolos binarios (0, 1), de esta manera a cada elemento del alfabeto de entrada se le asignará un código de longitud variable único y fijo, la longitud estará relacionada a ser inversamente proporcional al peso que tenga cada elemento en el alfabeto de entrada, con lo cual se busca que los símbolos que más se repitan tengan una representación binaria más corta ([Moffat, 2019](#)).

Este es un ejemplo de cómo se elabora un árbol de Huffman:

1. Éste será el alfabeto de entrada (Caracter, Peso): (a,10), (b,6), (c,2), (d,1), (e,1), (f,1). Obsérvese que todos los elementos ya están ordenados de acuerdo al peso.

2. Se toman los últimos 2 elementos de la lista y se combinan para generar un nuevo elemento que se incluirá en la lista ordenada, y se repite este paso hasta obtener un solo elemento final:

(a,10), (b,6), ([e,f],2), (c,2), (d,1)

(a,10), (b,6), ([c,d],3), ([e,f],2)

(a,10), (b,6), ([[c,d],[e,f]],5)

([b,[[c,d],[e,f]]],11), (a,10)

([a,[b,[[c,d],[e,f]]]],21)

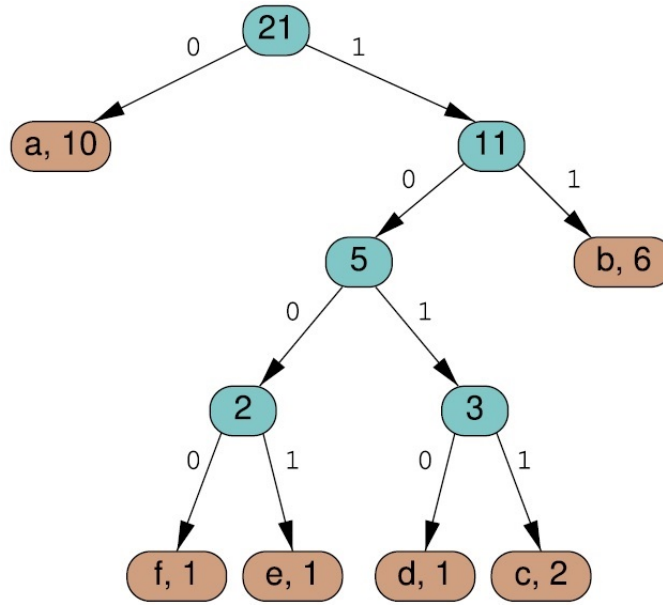


Figura 2.1: Árbol binario de código de Huffman resultante.

Basado en la figura 2.1 se obtienen los siguientes códigos únicos para cada carácter y se muestran en la siguiente tabla 2.3:

Caracter	Código Huffman
a	0
b	11
c	1011
d	1010
e	1001
f	1000

Tabla 2.3: Tabla de resultados de códigos Huffman del ejemplo

2.2. Marco Referencial

En esta sección se presentan artículos relevantes para el tema del presente trabajo desde las siguientes perspectivas: Rastreo de eventos de sistemas embebidos, diagnósticos en sitio, implementaciones modernas de código Morse y uso de código de Huffman en transmisión de datos.

2.2.1. Metodología de rastreo eficiente mediante un autómata programable.

Se estima que durante el desarrollo de software el 75 % del esfuerzo se dedica a verificación y depuración del sistema, sin embargo, a pesar de ese gran porcentaje de esfuerzo suelen haber defectos todo el tiempo dado que es difícil verificar todos los casos de prueba. Basado en eso, se puede considerar que es importante tener un método de monitoreo para continuar validando el sistema durante el desarrollo que se pueda quedar en la versión final del software para seguir monitorizando el sistema incluso después de la etapa de despliegue sin que cause un mal funcionamiento debido a sobrecarga de procesamiento.

El objetivo de este artículo ([Seo and Kurdahi, 2019](#)) es encontrar un método de monitoreo para sistemas embebidos con la menor intrusión posible y que soporte la enorme cantidad de datos que se generan para ser reportados en sistemas cada vez más complejos.

La metodología utilizada por los autores fue usar el simulador Gem5 para simular un procesador ARM de 1GHz con Caché L1 (16KB para datos y 16KB para instrucciones), Caché L2 (1MB compartido) y un 1GB de memoria y agregarle un hardware tipo procesador autómata, dedicado únicamente a reportar el monitoreo del sistema. Al mismo tiempo se montó un proyecto paralelo pero utilizando un FPGA para poder tener una referencia de comparación. Las principales ventajas que se obtienen al usar este tipo de hardware es que se puede abstraer cada máquina de estado del software, modelarlo en el autómata del hardware y cada una de estas máquinas de estado se evaluarán en paralelo, esta alta eficiencia de procesamiento puede llegar a representar una mejora de desempeño entre 45 y 3978 veces más rápido.

Se concluye entonces que un hardware tipo autómata dedicado a la tarea del monitoreo de un sistema embebido es muy eficiente, incluso mas eficiente que el uso de un FPGA que puede llegar a causar una sobrecarga de procesamiento. A través de este método se puede ser capaz de revisar el estado de todas las máquinas de estado

involucradas en el diseño del software y detectar fallas con facilidad.

2.2.2. Rastreo de programa sin intrusión en sistemas multi tarea usando consumo de corriente eléctrica.

Todo método de monitoreo tiene como principal reto que usualmente los sistemas embebidos están tan ajustados en recursos para poder ser comercialmente competitivos, por lo que esta característica del sistema debe consumir el menor número de recursos y entorpecer al mínimo posible la ejecución de la aplicación principal para lo que el sistema fue concebido.

El proposito de este artículo ([Lamichhane et al., 2018](#)) es presentar un método de rastreo de eventos para sistemas embebidos con la menor intrusión posible a través del monitoreo del nivel de consumo de corriente eléctrica del sistema embebido en los diferentes eventos del sistema, de manera que sabremos que cuando el consumo de corriente sea X , se deduce que evento esta ocurriendo, a diferencia de cuando el nivel de corriente sea Y que a su vez representara otro evento.

Los autores probaron este método utilizando un microcontrolador ATmega2560 corriendo a 1MHz y revisando el consumo de energía de la línea de alimentación con un analizador lógico Saleae. Mediante ejecuciones del sistema con diferentes tipos de código de instrumentación los autores lograron capturar qué cantidad de consumo de energía se relaciona con la ejecución de cada tarea del sistema, con estos datos se entrenó el algoritmo que identificaría qué tarea se está ejecutando en tiempo real relacionando los datos obtenidos con los parámetros ya conocidos.

Se logró un promedio de 97% de correcta identificación de qué tarea se estaba ejecutando realmente mediante la relación del consumo eléctrico con los datos previamente recabados mediante el código instrumentado, sin embargo se logró notar que la mayor parte de los errores en la identificación de tareas obedecía a que había tareas que tenían un nivel de consumo promedio muy similar, se cree que se requiere mayor granularidad en el dato del consumo de energía para poder identificar diferencias sutiles. Los autores también mencionan que el siguiente reto sería implementar

este método en sistemas embebidos multi-núcleo.

2.2.3. Diagnóstico de fallas de motor en sitio mediante el uso de una red neuronal convolucional.

Usualmente para poder hacer un diagnóstico del estado actual de un dispositivo se requiere tener el dispositivo aislado de su entorno habitual, generalmente en un banco de pruebas, un laboratorio o un lugar apto con las herramientas necesarias y ser evaluado por una persona experta capaz de determinar las necesidades de mantenimiento o reparaciones pertinentes.

El objetivo es obtener información de un motor mientras está trabajando en sus condiciones habituales con el fin de diagnosticar fallas preventivas o correctivas.

En el artículo ([Lu et al., 2019](#)) se propone utilizar una tarjeta raspberry PI 3 como un sistema portátil que cuente con dispositivos de salida para mostrar información y múltiples sensores de entrada que pueda tomar muestras de aspectos físicos de motores trabajando en su sitio habitual, tales como son su temperatura, el consumo de corriente eléctrica, nivel de vibración y el nivel de ruido. La idea es que este sistema pueda ser algo fácil de transportar y adaptar a un motor que está trabajando en su sitio habitual, los datos se obtienen a través de los sensores y se transforman a valores digitales con los cuales un experto puede llegar a interpretarlos correctamente o en este caso se plantea utilizar como experto de diagnóstico a una red neuronal convolucional que pueda ser entrenada para desarrollar esta actividad eficientemente.

Por supuesto la efectividad de la red neuronal convolucional esta sujeta al numero de muestras que tenga como registros de entrada, tanto de motores en buen estado como motores con todo tipo de fallas, en este caso se utilizaron 8 muestras, 1 motor en buen estado y 7 con diferentes fallas. El sistema demostró ser capaz de diagnosticar de manera correcta el estado de cada motor sin necesidad de mover el motor de lugar, este método se muestra como una alternativa confiable, cómoda y barata, enfocada principalmente en el mantenimiento preventivo.

2.2.4. Monitoreo de requerimientos de sistemas embebidos sin intrusión y en sitio.

Los sistemas embebidos deben ser validados adecuadamente para evitar problemas con el usuario final, esto se vuelve particularmente importante en sistemas de los cuales puede depender una vida, como pueden ser los automóviles autónomos o los dispositivos médicos. Hacer la validación de requerimientos de un sistema suele ser una tarea abrumadora debido a la complejidad cada vez mayor de los dispositivos embebidos, por lo que usualmente los sistemas solo se validan parcialmente abriendo la puerta a posibles riesgos.

El objetivo de este artículo ([Seo and Lysecky, 2018](#)) es encontrar un método de validación de un sistema embebido de acuerdo con sus requerimientos en tiempo de ejecución sin ser intrusivo.

Se propone utilizar un FPGA que lo nombran como NIRM que se agregará al SoC del sistema embebido y que estará conectado al del microcontrolador a través del “trace port”, para estar monitoreando el program counter de una manera no intrusiva. El fundamento de cómo este método verifica los requerimientos se basa en elaborar un diagrama de secuencia UML agregando siempre tiempos máximos de ejecución para cada etapa de la secuencia en el diagrama, luego este diagrama UML se reduce a un RMG donde considera todos los mensajes dentro del diagrama de secuencia como eventos del sistema a ser graficados considerando los tiempos originales del diagrama UML. Una vez que se tiene el RMG, se analiza y se optimiza para ser reducido a través de los siguientes algoritmos: “Greedy Iterative RMG Optimization Algorithm” y “Multi-phase Iterative RMG Optimization Algorithm” y “Simulated Annealing RMG Optimization Algorithm”. Una vez reducido el RMG tendremos una serie secuencial de eventos del sistema que de ser ejecutados satisfactoriamente se garantizará que el resto de secuencias se realizaron correctamente. De estos eventos de sistema se deberán obtener la dirección de memoria de inicio y de fin, de esta manera el NIRM tendrá una tabla con esas direcciones y por otro lado tendrá acceso al program counter, así el NIRM se da cuenta cuando se comienza y cuando termina la ejecución de cada uno de estos eventos considerando las restricciones de

tiempo, con esto se valida de una manera muy eficiente que el sistema cumple con sus requisitos.

Para poder aplicar este método se requiere considerarlo desde la concepción del proyecto para incluir en el SoC de la tarjeta del sistema el FPGA conectado por el “trace port” para obtener un método no intrusivo. A partir del uso de diagramas de secuencia UML que podrían considerarse como diagramas estándar en el desarrollo de sistemas embebidos, más los algoritmos propuestos por los autores se obtiene una reducción eficaz de los eventos del sistema claves a ser monitoreados, resultando en un método robusto para verificación de requerimientos en tiempo de ejecución y “en sitio”.

2.2.5. Sistema de entrada para código Morse para personas con discapacidad.

Dada la naturaleza de la escritura de código morse, se requiere un ritmo específico de escritura para poder ser interpretado correctamente, sin embargo en el caso de las personas con alguna discapacidad puede no ser posible forzarlas a aprender a escribir en morse al ritmo requerido.

El objetivo es permitir que personas con discapacidades puedan comunicarse a su propio ritmo a través de código morse utilizando un panel táctil a ser usado con sus manos y dedos.

En el presente artículo ([Ravikumar and Dathi, 2016](#)) se utiliza un kit de desarrollo de Texas Instruments MSP430F5529 que incluye un panel táctil que será utilizado como método de entrada para escribir en código morse con las manos o dedos de cualquier usuario. El panel táctil entonces detectará cuando hay presión y cuando no y transmitirá el dato al microcontrolador que interpretará el mensaje de entrada con la ayuda de métodos de aprendizaje máquina basados en técnicas de lógica difusa para poder detectar el ritmo particular de escritura de cada usuario y lograr identificar correctamente el mensaje que se quiere transmitir. Se entiende que cada usuario pueda tener un ritmo distinto e incluso los mismos usuarios pueden cambiar

de ritmo conforme pasa el tiempo, por lo que se requiere tener una personalización de la calibración de la detección de los caracteres escritos para cada usuario, esto se logra mediante la identificación de nuevos usuarios donde se les pide que introduzcan un texto predeterminado que le servirá al usuario como práctica y al sistema para recabar información que le permita entrenarse para recibir entrada de información de ese usuario.

El artículo concluye entonces que el código morse puede ser usado por personas discapacitadas que apenas pueden mover un párpado, de ahí su importancia de hacerlo accesible para cualquier persona a la cual éste pueda ser su único método de comunicación. Se piensa que puede ser útil particularmente en ambientes de hospitales, centros de rehabilitación, casas de ancianos, etc. En este particular caso de la escritura utilizando un panel táctil se hicieron pruebas utilizando y sin utilizar los métodos de lógica difusa y se demostró que usando este método el sistema mejoró sustancialmente en su porcentaje de interpretación correcta de los caracteres y palabras escritas por los usuarios con un 20 % de errores de identificación de las entradas sin la lógica difusa y solo un 5 % de errores utilizando este método.

2.2.6. Utilización de código Huffman para sistema de boyas de clima en el mar.

Las boyas de clima son un instrumento utilizado sobre el mar para sensar factores del entorno que permitan predecir el comportamiento del clima. El sistema consiste en que cada boya toma sus muestras y envía su información a un receptor que se encargará de recolectar la información de todas las boyas y usar dicha información para evaluarla. El problema es que usualmente las boyas se encuentran a una distancia considerable respecto al receptor y la información es enviada sin ningún tipo de compresión, causando demoras en la transmisión de datos.

En este artículo los autores proponen el uso de un código de Huffman que pueda ayudar a comprimir los datos a ser enviados y a su vez usar el método inverso para descomprimir los datos recibidos, haciendo más eficiente la transmisión de información.

Los resultados fueron una mejora en tiempo de transmisión pasando de 6 minutos

a 13 segundos (27.6 veces más rápido), mejorando también el promedio de bits por segundo requeridos a ser transmitidos de 2000 a 480, con una precisión de compresión del 94.4 %.

Los autores concluyen en su artículo que el código de Huffman es una gran alternativa para hacer más eficiente la transmisión de datos, dado que además de mejorar los tiempos de transmisión también aumenta la confianza en la transmisión dado que al enviar menos datos se reduce el riesgo de que algún bit se mande incorrectamente. A pesar de las ya mencionadas ventajas, los autores sostienen que no sugieren el uso de códigos de Huffman para transmisiones de tiempo real donde el tiempo de compresión/descompresión puede ser crítico ([Velasco et al., 2021](#)).

Capítulo 3

Diseño e Implementación

3.1. Descripción de la propuesta

El presente proyecto propone el desarrollo de una biblioteca de código en lenguaje C que puede ser usada en cualquier sistema embebido multitarea, independientemente de la plataforma de hardware, haciendo uso de una capa de abstracción de hardware para transmitir mensajes de los eventos que los desarrolladores del sistema consideren necesarios o provechosos (códigos de error, eventos, ejecución de una tarea, etc.) directamente al usuario mediante código Morse y en sitio.

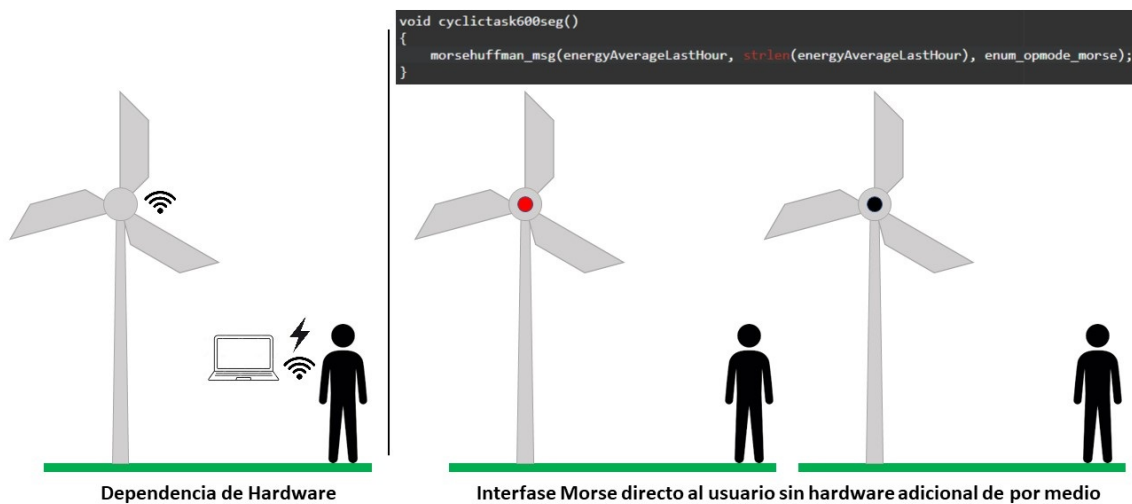


Figura 3.1: Esta figura muestra el ejemplo de un generador de energía eólico visto en dos escenarios distintos.

La figura 3.1 muestra un ejemplo donde se pueden observar dos escenarios de un mismo sistema embebido. En el primer escenario (izquierda), para conocer detalles

en tiempo de ejecución del sistema, existe una dependencia a hardware adicional como lo es una computadora que a su vez tiene las dependencias de contar con energía eléctrica para dicha computadora y señal de recepción de datos por un medio alámbrico o inalámbrico. El segundo escenario (derecha) muestra un LED rojo instalado en el mismo sistema embebido que va a transmitirle directamente al ojo de quien necesite leerlo usando código Morse el promedio de energía generada cada 10 minutos.

Esta propuesta se compone a nivel funcional de una opción principal (Opción 1) y una opción complementaria (Opción 2) de acuerdo con lo siguiente:

- Opción 1.- Transmisión de mensajes en código Morse utilizando una salida del microcontrolador del sistema embebido y un LED de manera que los mensajes pueden llegar a ser interpretados directamente por un humano sin la necesidad de un hardware adicional y en cualquier sitio donde el sistema embebido esté trabajando.
- Opción 2.- Es un complemento a la primera opción y consiste en aprovechar cada cambio de flanco de la salida que está transmitiendo mensajes en código Morse para transmitir un caracter utilizando un código de Huffman con el fin de hacer más eficiente la transmisión de los datos al enviarlos comprimidos. Este segundo método será imperceptible para un ser humano debido a la velocidad con la que se ejecuta la transmisión dado que la transmisión de cualquier caracter estará en el orden de los microsegundos, por lo cual se necesita de un analizador lógico que capture cada cambio de estado para más tarde exportar los resultados de las mediciones y utilizar mediante un código para obtener la lectura de datos transmitidos tanto en Morse como en Huffman.

3.2. Selección de alfabeto de entrada

Con el propósito de evitar tener un uso de memoria innecesariamente alto, y para no obtener códigos de Huffman demasiado largos, se busca definir en el alfabeto de entrada el mínimo de caracteres que permitan alcanzar un alto nivel de flexibilidad para transmitir la mayoría de mensajes.

Basado en ese propósito se decidió tomar como alfabeto de entrada los veintiséis caracteres del alfabeto en idioma inglés, los diez dígitos arábigos y el espacio en blanco, los cuales se pueden observar en la tabla 3.1. Es importante recalcar que no es relevante la distinción entre mayúsculas y minúsculas para la transmisión en Morse ni para la codificación de códigos Huffman.

Índice	Caracter	Índice	Caracter
1	“0”	20	“J/j”
2	“1”	21	“K/k”
3	“2”	22	“L/l”
4	“3”	23	“M/m”
5	“4”	24	“N/n”
6	“5”	25	“O/o”
7	“6”	26	“P/p”
8	“7”	27	“Q/q”
9	“8”	28	“R/r”
10	“9”	29	“S/s”
11	“A/a”	30	“T/t”
12	“B/b”	31	“U/u”
13	“C/c”	32	“V/v”
14	“D/d”	33	“W/w”
15	“E/e”	34	“X/x”
16	“F/f”	35	“Y/y”
17	“G/g”	36	“Z/z”
18	“H/h”	37	Blank space
19	“I/i”		

Tabla 3.1: Tabla de caracteres elegidos como parte del alfabeto de entrada.

3.3. Diseño de Hardware

Se busca desacoplar esta biblioteca propuesta de un hardware específico, para eso se usa una capa de abstracción de hardware intermedia entre la aplicación y

el hardware como se muestra en la figura 3.2. La idea es que al implementarse esta biblioteca en un hardware distinto solo se tenga que actualizar esta capa de abstracción con las interfaces de los controladores de hardware correspondientes.

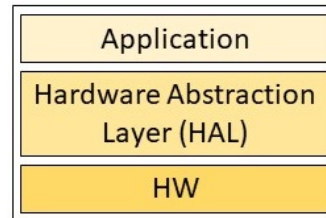


Figura 3.2: Capa de abstracción de hardware conocida como HAL.

A nivel hardware, la figura 3.3 muestra la configuración a nivel hardware de las dos opciones ya descritas sin especificar una plataforma de hardware para el microcontrolador emisor:

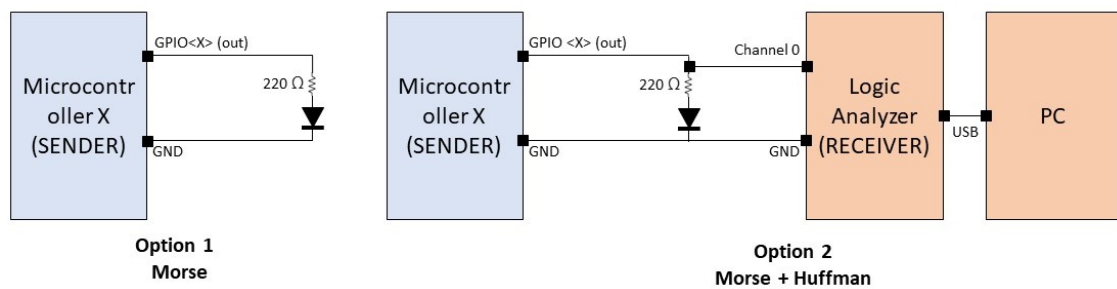


Figura 3.3: Se muestra la configuración a nivel hardware para opción 1 y opción 2.

Dado que el objetivo de este proyecto es producir una biblioteca de código en lenguaje C, la implementación no está sujeta a un hardware específico; pero con la intención de mostrar un ejemplo de referencia de una posible implementación, en la siguiente figura 3.4 se muestra un Arduino UNO con su microcontrolador ATmega328P en el lugar del microcontrolador emisor:

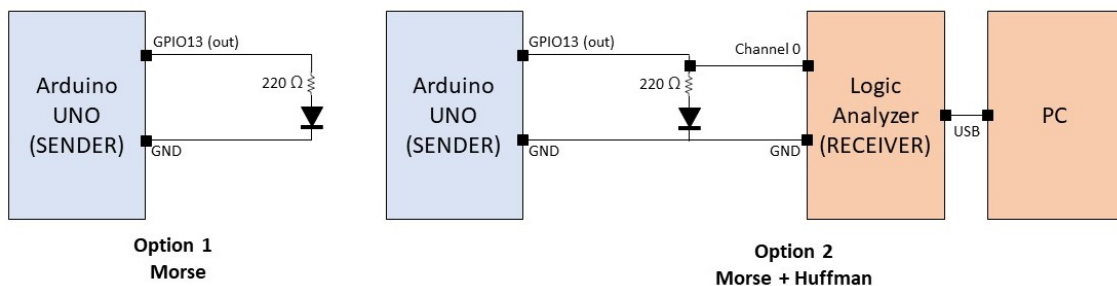


Figura 3.4: Ejemplo de una posible implementación a nivel hardware.

- Opción 1: Para transmisión de mensajes en código Morse. En este ejemplo de implementación se usa el GPIO 13 configurado como salida de una tarjeta Arduino UNO y entre este GPIO y la tierra hay un LED con su respectiva resistencia de 220Ohm.
- Opción 2: Para transmisión de mensajes en código Morse y usando código de Huffman. La misma configuración que la opción anterior, pero se añade una línea nueva que va desde cualquier punto entre la resistencia de 220Ohm y el GPIO 13 del Arduino UNO al canal 0 de un analizador lógico así como también conectar la línea de tierra del analizador lógica a la misma tierra del Arduino UNO para prevenir posibles daños al hardware. El analizador lógico a su vez estará conectado mediante un cable USB a una computadora.

3.4. Diseño de Software

Esta biblioteca se compone de cinco archivos, dos de código fuente y tres de encabezados .h distribuidos de la siguiente manera:

- `morsehuffman_lib.h`: El archivo encabezado que se debe incluir en el archivo donde se usarán las interfaces públicas de la biblioteca. Este archivo no debe ser editado.
- `morsehuffman_lib.c`: El código fuente de las interfaces principales de la biblioteca. Este archivo no debe ser editado.
- `morsehuffman_hal.h`: El archivo encabezado que contiene las interfaces públicas de la capa de abstracción de hardware. Este archivo no debe ser editado.
- `morsehuffman_hal.c`: El código fuente de la capa de abstracción del hardware. En este archivo se deben incluir las bibliotecas de controladores del hardware que se está utilizando y usar las interfaces correspondientes para que sean llamadas dentro de las funciones ya definidas de este archivo. Este archivo debe ser editado de acuerdo a la plataforma de hardware que se utilice.
- `morsehuffman_cfg.h`: Un archivo encabezado que contiene dos valores configurables de la biblioteca que son el tamaño del buffer de datos, expresado en

número de caracteres y el otro es la cantidad de microsegundos que la biblioteca debe esperar con la salida activa cuando se transmite un dígito 0 en el método Huffman, considerando que ésta será la unidad de medida de referencia y que cuando se transmita un 1 el tiempo de espera será tres veces esta unidad.

En la imagen 3.5 se puede observar de qué manera interactúan los archivos descritos.

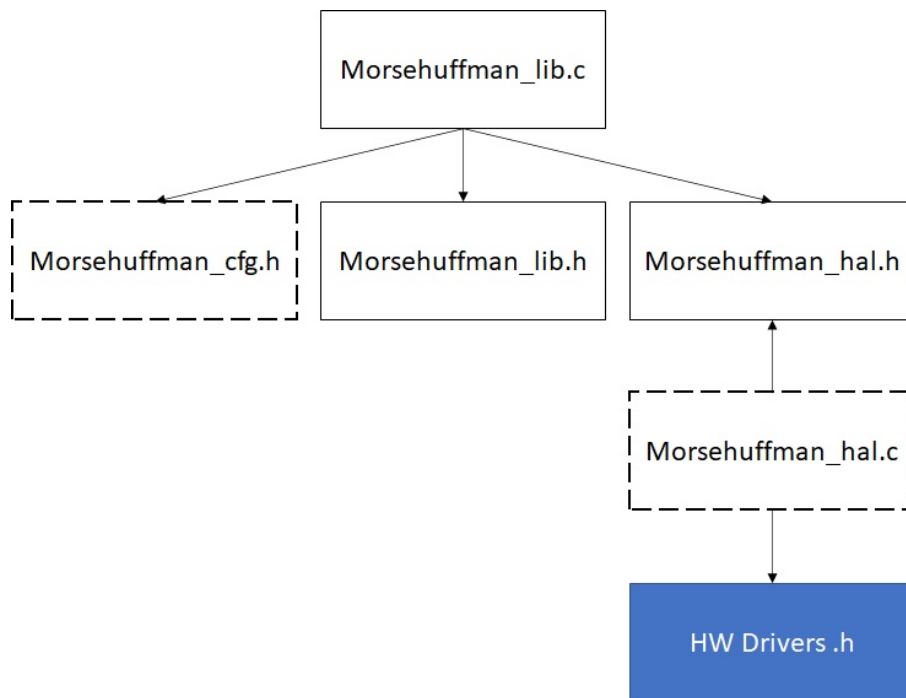


Figura 3.5: En esta imagen se puede ver la estructura de la librería propuesta respecto a los archivos que la componen.

La siguiente imagen 3.6 muestra un diagrama de la estructura de un sistema embebido típico y en que capas funcionara esta biblioteca.

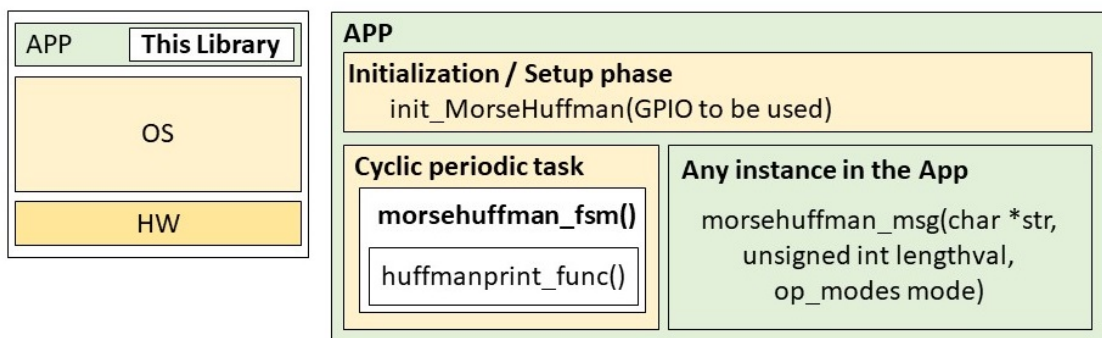


Figura 3.6: A nivel software la implementación requerida es la misma tanto para la opción 1: Método Morse únicamente, como para la opción 2: Método Morse + Huffman.

Como se puede observar en la figura 3.6, esta biblioteca solo ofrece tres interfaces públicas:

- 1. “void init_MorseHuffman(unsigned char led_id)”. Esta API debe ser llamada en la fase de inicialización. Recibe como parámetro el GPIO que se seguirá usando como salida para la comunicación Morse y Huffman si es que aplica. Se puede consultar el algoritmo 4 del apartado de anexos.
- 2. “void morsehuffman_fsm(void)”. Esta API tiene que ser añadida dentro de una tarea cíclica. El tiempo que tarde esta tarea en volver a llamarse será tomado como la unidad de tiempo base para la comunicación Morse. (Ejemplo: Si se llama cada 500ms, un “punto” Morse que usa una unidad de tiempo durará .5 segundos y una “línea” Morse que usa tres unidades de tiempo durará 1.5 segundos). En caso de que no haya nada en el buffer pendiente por ser transmitido, esta función no modificará el estado por defecto de la salida. Se puede consultar el algoritmo 5 del apartado de anexos.
- 3. “void morsehuffman_msg(char *str, unsigned int lengthval, op_modes mode)”. Esta es la API que usan los desarrolladores para enviar la cadena ASCII que requieren transmitir enviándola como parámetro de entrada junto con la longitud de dicha cadena y el método por el que desean que se transmita usando el tercer parámetro de entrada si por Código Morse o por código Huffman. Es una interfase no bloqueante dado que en realidad solo copia la cadena solicitada al buffer del método seleccionado y regresa al contexto de donde la API fue llamada a continuar con la siguiente instrucción. Se puede consultar el algoritmo 6 del apartado de anexos.

3.4.1. Máquina de estado finita

Como ya se mencionó en la sección anterior 3.4, la segunda interfase pública es “morsehuffman_fsm” y es la encargada de hacer la llamada a una máquina de estado finita que se muestra en la figura 3.7 y que se describe a detalle a continuación.

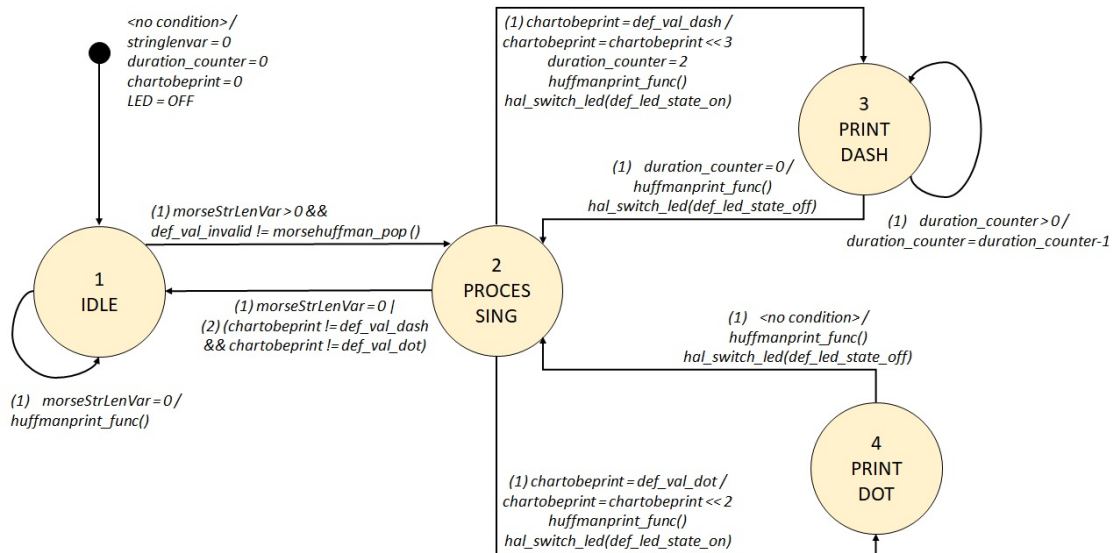


Figura 3.7: En este diagrama se muestra el diseño de la máquina de estados principal que debe ser llamada a través de su API en una tarea cíclica.

En esta máquina de estados finita existen cuatro estados:

- 1. Idle.- Estado por defecto cuando no hay datos en el buffer pendientes por ser transmitidos (esto incluye el caso de cuando la transmisión de datos ha concluido) o cuando el próximo carácter a ser transmitido es inválido. El tamaño de los datos en el buffer del método Morse a ser transmitidos siempre estará registrado en la variable “morseStrLenVar”. El siguiente carácter a ser transmitido será almacenado en la variable “chartobeprint”.

Transiciones:

1 a 1 - Idle a Idle

```

/* Conditions:*/
morseStrLenVar=0

```

1 a 2 - Idle a Processing

```

/* Conditions:*/
morseStrLenVar>0
&& chartobeprint!=def_val_invalid
/*No outputs:*/

```

- 2. Processing.- El estado Processing se encarga de obtener cuál es el siguiente carácter ASCII del buffer e identifica cuál es su representación binaria, enton-

ces comienza a procesar si se requiere transmitir una “línea” o un “punto” Morse, dependiendo de esto se puede hacer una transición al siguiente estado: “(3) Print_Dash” o “(4) Print_Dot” respectivamente. Cuando ocurre el cambio al estado “(3) Print_Dash” hay un corrimiento de tres bits a la izquierda en la variable “chartobepprint” y la variable “duration_counter” toma el valor de dos, mientras que cuando la transición es hacia el estado “(4) Print_Dot” el corrimiento a la izquierda de la variable “chartobepprint” es de dos bits y la variable “duration_counter” permanece con valor 0. En ambas transiciones se llama a la función “huffmanprint_func” y se enciende el LED.

En caso de que el símbolo a transmitir sea diferente de la “línea” o el “punto”, habrá una transición al estado inicial (1) Idle.

Para mayor detalle de cómo se lleva a cabo esta traducción de código Morse a su representación binaria se requiere consultar la sección 3.5.

Transiciones:

2 a 1 - Processing a Idle

```
/*Conditions:*/
(morseStrLenVar=0) ||
(chartobepprint=def_val_blank ||
chartobepprint=def_val_end)
/*Outputs:*/
if (morseStrLenVar > 0)
    morseStrLenVar = morseStrLenVar - 1
```

2 a 3 - Processing a Print_Dash

```
/*Conditions:*/
chartobepprint=def_val_dash
/*Outputs:*/
chartobepprint << 3
duration_counter = 2
huffmanprint_func()
switch_led(def_led_state_on)
```

2 a 4 - Processing a Print_Dot

```
/*Conditions:*/
chartobeprint=def_val_dot
/*Outputs:*/
chartobeprint << 2
huffmanprint_func()
switch_led(def_led_state_on)
```

- 3. Print_Dash.- Este estado se encarga de mantener activa la salida por las tres unidades de tiempo que requiere una “Línea” de Morse, estas 3 unidades de tiempo se representan con el contador “duration_counter” que antes de la transición a este estado toma el valor de 2. Mientras el valor del contador “duration_counter” sea mayor que 0, se permanecerá en el presente estado. Únicamente habrá un decremento de menos uno en el valor de dicho contador, una vez alcanzando el valor 0 habrá una transición al estado (2) Processing, se manda a llamar a la función “huffmanprint_func” y se apaga el LED.

Transiciones:

3 a 3 - Print_Dash a Print_Dash

```
/*Conditions:*/
duration_counter>0
/*Outputs:*/
duration_counter = duration_counter - 1
```

3 a 2 - Print_Dash a Processing

```
/*Conditions:*/
duration_counter=0
/*Outputs:*/
huffmanprint_func()
switch_led(def_led_state_off)
```

- 4. Print_Dot.- Este estado se encarga de mantener activa la salida por una unidad de tiempo que requiere un “Punto” de Morse. No hay condición a cumplir para pasar al estado (2) Processing. Una vez que se completa esta

transición se manda a llamar a la función “huffmanprint_func” y se apaga el LED.

Transiciones:

4 a 2 - Print_Dot a Processing

```
/*No condition*/  
/*Outputs:*/  
huffmanprint_func()  
switch_led(def_led_state_off)
```

3.5. Tabla de representación Morse

Analizando la representación Morse de los treinta y seis caracteres del alfabeto de entrada (descrito en la sección 3.2) que restan una vez se descarta el espacio en blanco que en Morse se representa a través de una pausa de tres unidades de tiempo, encontramos que dos caracteres se pueden representar únicamente con un símbolo. En el caso de la E que se puede representar con un solo “punto” y en el caso de la T que se representa con una sola “línea”; sin embargo, hay diez caracteres que se representan con diferentes combinaciones únicas de cinco símbolos que serían tomados como peor escenario.

Con el fin de tener valores numéricos que representen las diferentes combinaciones entre “puntos”, “líneas” y final de caracter en la menor cantidad de bits tomando como referencia el peor escenario ya descrito de cinco símbolos para un caracter, se decidió utilizar la siguiente representación a nivel bits:

- 10 = “Punto”
- 110 = “Línea”
- 00 = Fin de caracter, donde el primero de estos ceros siempre es el último, ya sea de un “punto” o de una “línea”.

Siguiendo estas definiciones encontramos que con 16 bits era suficiente para representar cualquier caracter de este alfabeto de entrada como se puede observar en la siguiente figura 3.8:

A	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
B	1	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0
C	1	1	0	1	0	1	1	0	1	0	0	0	0	0	0	0
D	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
E	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	1	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0
G	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0
H	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0
I	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
J	1	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0
K	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0
L	1	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0
M	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
N	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
O	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0
P	1	0	1	1	0	1	1	0	1	0	0	0	0	0	0	0
Q	1	1	0	1	1	0	1	0	1	1	0	0	0	0	0	0
R	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
S	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
T	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
U	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0
V	1	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0
W	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0
X	1	1	0	1	0	1	0	1	1	0	0	0	0	0	0	0
Y	1	1	0	1	0	1	1	0	1	1	0	0	0	0	0	0
Z	1	1	0	1	1	0	1	0	1	0	0	0	0	0	0	0
1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	0	0
2	1	0	1	0	1	1	0	1	1	0	1	1	0	0	0	0
3	1	0	1	0	1	0	1	1	0	1	1	0	0	0	0	0
4	1	0	1	0	1	0	1	0	1	1	0	0	0	0	0	0
5	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0
6	1	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0
7	1	1	0	1	1	0	1	0	1	0	1	0	0	0	0	0
8	1	1	0	1	1	0	1	1	0	1	0	1	0	0	0	0
9	1	1	0	1	1	0	1	1	0	1	1	0	1	0	0	0
0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	0

Figura 3.8: Se utilizan colores para facilitar la identificación de cada tipo de símbolo, amarillo para “puntos”, verde para “líneas” y el naranja para provocar dos 0 consecutivos que marcan el final del carácter.

De tal manera que con variables de 16 bits, transformando estos valores de la figura 3.8 de binario a hexadecimal, obtenemos los siguientes valores que se muestran en la tabla 3.2:

Hexadecimal	Character	Hexadecimal	Character
DB6C	“0”	A000	“I/i”
B6D8	“1”	B6C0	“J/j”
ADB0	“2”	D600	“K/k”
AB60	“3”	B500	“L/l”
AAC0	“4”	D800	“M/m”
AA80	“5”	D000	“N/n”
D540	“6”	DB00	“O/o”
DAA0	“7”	B680	“P/p”
DB50	“8”	DAC0	“Q/q”
DB68	“9”	B400	“R/r”
B000	“A/a”	A800	“S/s”
D500	“B/b”	C000	“T/t”
D680	“C/c”	AC00	“U/u”
D400	“D/d”	AB00	“V/v”
8000	“E/e”	B600	“W/w”
AD00	“F/f”	D580	“X/x”
DA00	“G/g”	D6C0	“Y/y”
AA00	“H/h”	DA80	“Z/z”

Tabla 3.2: Tabla de valores hexadecimales de cada caracter basado en su representación Morse.

3.6. Generación de árbol binario de Huffman

Se generó un árbol binario de Huffman utilizando como base el alfabeto de entrada y asignándoles un peso secuencialmente decreciente a cada caracter a partir del 37 y hasta el 1 basado en los caracteres que pudieran tener más uso utilizando los siguientes criterios:

- 1. Números del 0 al 9.
- 2. Frecuencia de uso de las letras en el idioma Inglés:
“EARIOTNSLCUDPMHGBFYWKVXZJQ” ([Dictionary](#), 2012).

- 3. Introducir el “espacio en blanco” justo después de la última vocal listada por el punto 2.

Se puede observar la tabla 3.3 con estos criterios aplicados:

Peso	Caracter	Peso	Caracter
37	“0”	18	“C/c”
36	“1”	17	“U/u”
35	“2”	16	Blank space
34	“3”	15	“D/d”
33	“4”	14	“P/p”
32	“5”	13	“M/m”
31	“6”	12	“H/h”
30	“7”	11	“G/g”
29	“8”	10	“B/b”
28	“9”	9	“F/f”
27	“E/e”	8	“Y/y”
26	“A/a”	7	“W/w”
25	“R/r”	6	“K/k”
24	“I/i”	5	“V/v”
23	“O/o”	4	“X/x”
22	“T/t”	3	“Z/z”
21	“N/n”	2	“J/j”
20	“S/s”	2	“Q/q”
19	“L/l”		

Tabla 3.3: Tabla alfabeto de entrada del árbol Huffman con el peso de cada caracter.

En la figura 3.9 podemos observar cómo se construye el árbol binario tomando como convención que siempre el camino superior tomará el 1 y el inferior el 0:

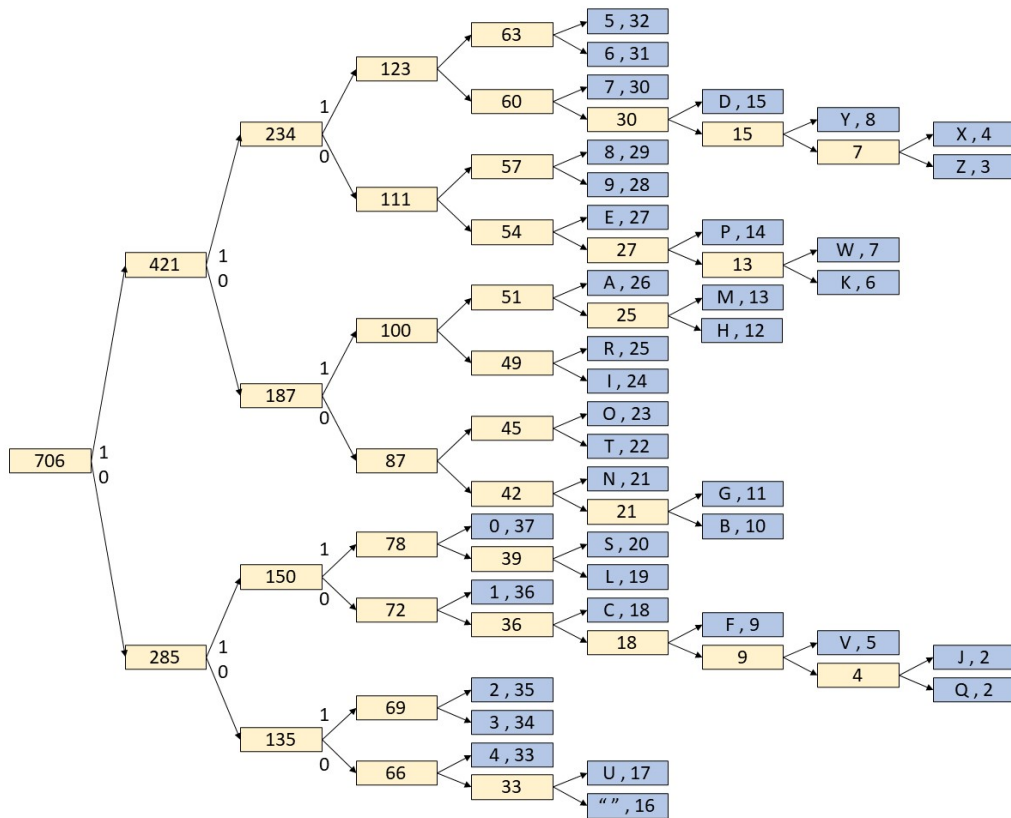


Figura 3.9: Árbol binario de Huffman resultante dado el alfabeto de entrada definido.

A manera de ejemplo tomaremos el caso de la letra “J” que resultó con uno de los peores casos en términos de longitud:

Del nodo 706 tomamos el 0 (0).

Del nodo 285 tomamos el 1 (01).

Del nodo 150 tomamos el 0 (010).

Del nodo 72 tomamos el 0 (0100).

Del nodo 36 tomamos el 0 (01000).

Del nodo 18 tomamos el 0 (010000).

Del nodo 9 tomamos el 0 (0100000).

Del nodo 4 tomamos el 1 (01000001, resultado final).

Haciendo un análisis de los resultados de este árbol binario Huffman encontramos que los códigos más cortos han quedado de 4 bits para los caracteres 0, 1, 2, 3 y 4, mientras que en el peor escenario encontramos a los caracteres X, Z, J y Q que quedaron de 8 bits. Tomando el peor escenario como base, se almacenan todos los valores en números hexadecimales dentro de variables de 8 bits como se muestra a

continuación en la tabla 3.4. Es importante mencionar que hay casos como el del número 4 y la letra U, donde al representarlos como números binarios encontramos que ambos arrojan el mismo número. Para evitar esto se colocarán ceros a la derecha del número hasta completar 8 bits. Haciendo esto el número 4 se representará como 10 hexadecimal y la U como 08 hexadecimal:

Caracter	Binario	Hexadecimal	Caracter	Binario	Hexadecimal
“0”	0111	70	“C/c”	01001	48
“1”	0101	50	“U/u”	00001	08
“2”	0011	30	Blank space	00000	00
“3”	0010	20	“D/d”	111001	E4
“4”	0001	10	“P/p”	110001	C4
“5”	11111	F8	“M/m”	101101	B4
“6”	11110	F6	“H/h”	101100	B0
“7”	11101	E8	“G/g”	100001	84
“8”	11011	D8	“B/b”	100000	80
“9”	11010	D0	“F/f”	010001	44
“E/e”	11001	C8	“Y/y”	1110001	E2
“A/a”	10111	B8	“W/w”	1100001	C2
“R/r”	10101	A8	“K/k”	1100000	C0
“I/i”	10100	A0	“V/v”	0100001	42
“O/o”	10011	98	“X/x”	11100001	E1
“T/t”	10010	90	“Z/z”	11100000	E0
“N/n”	10001	88	“J/j”	01000001	41
“S/s”	01101	68	“Q/q”	01000000	40
“L/l”	01100	60			

Tabla 3.4: Tabla de códigos de Huffman resultantes en representación binaria y hexadecimal.

3.7. Mensajes en código de Huffman

Como se puede observar en la figura 3.7, existe una función privada dentro de la biblioteca llamada “huffmanprint_func” que es llamada en cada cambio de flanco de

la salida que transmite mensajes en código Morse. El objetivo es que antes de realizar el ya mencionado cambio de flanco se aproveche ese momento para transmitir el siguiente caracter que se encuentre en el buffer del método Huffman, el cual cabe recordar que es un buffer independiente del buffer del método Morse.

Es importante señalar que en cada cambio de flanco se tiene la limitante de solo transmitir un caracter, para evitar tomar demasiado tiempo de la ejecución del programa principal y que esto se interprete como una intrusión.

La figura 3.10 muestra un diagrama de flujo que describe el comportamiento de la función “huffmanprint_func” donde primero siempre se revisa si hay datos pendientes por transmitir en el buffer; si no los hay, la función termina. En caso de que sí haya, se obtiene el siguiente caracter ASCII del buffer y se busca el código Huffman que le corresponde y se comienza a transmitir dígito por dígito.

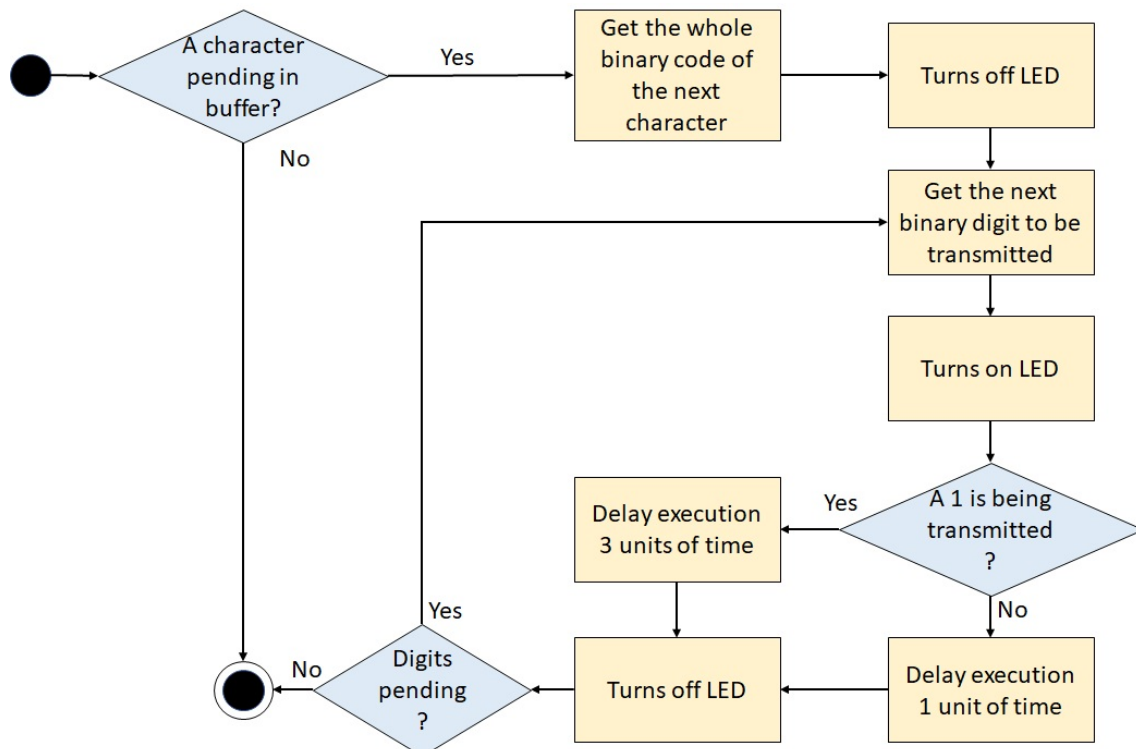


Figura 3.10: Diagrama de flujo de la función estática “huffmanprint_func” la cual se llama cada que hay un cambio de flanco en la salida que transmite los mensajes en Morse.

En este método la forma de transmitir los dígitos 0 y 1 es similar al código

Morse donde ambos dígitos son pulsos en alto en la salida del microcontrolador. La diferencia entre ambos es la duración de sus pulsos, para el 0 la duración es de una unidad de tiempo, mientras que para el dígito 1 su duración es de dos unidades de tiempo. La unidad de tiempo se expresa en μS y es un parámetro configurable llamado “def_cfg_huffman_us_0” en el archivo “morsehuffman_cfg.c”.

En la siguiente tabla 3.5 podemos observar los tiempos que tardaría cada caracter con una unidad de tiempo base de 1 μS , donde el dígito 0 binario toma una unidad de tiempo, el dígito 1 dos unidades de tiempo y el espacio entre dígitos también dura una unidad de tiempo.

Caracter	Binario	Tiempo en μS	Caracter	Binario	Tiempo en μS
“0”	0111	11	“C/c”	01001	12
“1”	0101	10	“U/u”	00001	11
“2”	0011	10	Blank space	00000	10
“3”	0010	9	“D/d”	111001	16
“4”	0001	9	“P/p”	110001	15
“5”	11111	15	“M/m”	101101	16
“6”	11110	14	“H/h”	101100	15
“7”	11101	14	“G/g”	100001	14
“8”	11011	14	“B/b”	100000	13
“9”	11010	13	“F/f”	010001	14
“E/e”	11001	13	“Y/y”	1110001	18
“A/a”	10111	14	“W/w”	1100001	17
“R/r”	10101	13	“K/k”	1100000	16
“I/i”	10100	12	“V/v”	0100001	16
“O/o”	10011	13	“X/x”	11100001	20
“T/t”	10010	12	“Z/z”	11100000	19
“N/n”	10001	12	“J/j”	01000001	18
“S/s”	01101	13	“Q/q”	01000000	17
“L/l”	01100	12			

Tabla 3.5: Tiempos para representar cada caracter tomando como unidad de tiempo 1 μS .

Como se puede advertir, a pesar de que los caracteres están ordenados de mayor

a menor por su frecuencia de uso en el idioma inglés, los tiempos no obedecen a tal orden, los tiempos están desordenados. Esto se debe a que en la practica, además de la cantidad total de bits, también influye la frecuencia de dígitos “1” que tomará más tiempo para transmitirse que los dígitos “0”, el ejemplo claro está en el código Huffman “00000” que toma $15.82 \mu\text{S}$ mientras que “11111” toma $19.15 \mu\text{S}$, ambos códigos de 5 bits de longitud.

Observando estos resultados se decidió tomar el orden del tiempo real que tarda cada codigo Huffman en transmitirse para empatar con el orden de frecuencia definido en la tabla 3.3, quedando entonces los códigos Huffman por caracter como se describe en la tabla 3.6:

Caracter	Binario	Tiempo en μS	Caracter	Binario	Tiempo en μS
"0"	0010	9	"C/c"	11011	14
"1"	0001	9	"U/u"	10111	14
"2"	0101	10	Blank space	100000	14
"3"	0011	10	"D/d"	11111	14
"4"	0111	10	"P/p"	100001	14
"5"	00000	11	"M/m"	010001	15
"6"	00001	11	"H/h"	110001	15
"7"	10100	12	"G/g"	101100	15
"8"	10010	12	"B/b"	111001	16
"9"	10001	12	"F/f"	101101	16
"E/e"	01100	12	"Y/y"	1100000	16
"A/a"	01001	12	"W/w"	0100001	16
"R/r"	11010	13	"K/k"	1100001	17
"I/i"	11001	13	"V/v"	01000000	17
"O/o"	10101	13	"X/x"	1110001	18
"T/t"	10011	13	"Z/z"	01000001	18
"N/n"	01101	13	"J/j"	11100000	19
"S/s"	11110	13	"Q/q"	11100001	20
"L/l"	11101	14			

Tabla 3.6: Tabla con los códigos binarios reordenados por número de bits y por duración de tiempo luego del análisis de tiempos de la tabla 3.5, usando la misma referencia de 1 μS como unidad de tiempo.

Capítulo 4

Resultados y Discusión

4.1. Resultados

4.1.1. Entorno de validación

Como entorno de validación de la biblioteca de código propuesta en este proyecto de tesis se tomó en cuenta el set de pruebas de la 4.1, una computadora, un analizador lógico, una placa de pruebas comúnmente llamadas “protoboard”, un LED con su respectiva resistencia de 220 ohm y un microcontrolador el cual será cambiado por diferentes dispositivos para hacer las mismas pruebas y poder realizar una comparación.

Siguiendo el propósito de esta biblioteca, el cual es ser una propuesta orientada a fines académicos y de bajo costo, se eligieron los dispositivos listados a continuación para ser parte de las pruebas de esta biblioteca, debido a que son considerados como dispositivos comunes entre la comunidad estudiantil y entusiastas de los sistemas embebidos:

1. Placa de microcontrolador Arduino UNO.
2. Microcontrolador Microchip PIC16F15313.
3. Plataforma NodeMCU ESP8266.
4. Ordenador de placa reducida Raspberry PI 3.

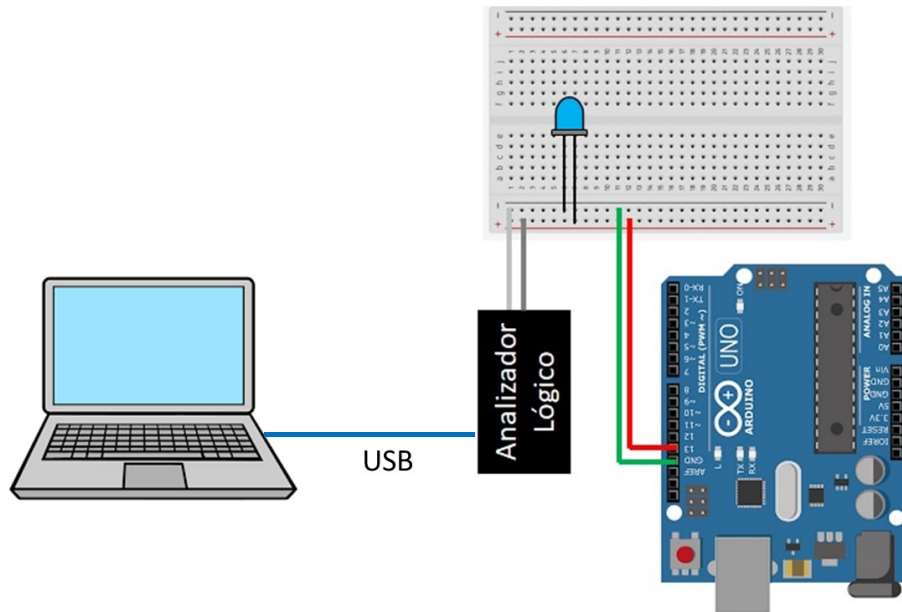


Figura 4.1: Entorno de validación una computadora, un analizador lógico, una “protoboard”, un LED y un microcontrolador, en este caso se muestra el ATmega32U4 desde una placa Arduino UNO.

4.1.2. Casos de prueba

Condiciones a validar

Para confirmar la funcionalidad de la biblioteca desarrollada se seguirán los siguientes pasos en cada uno de los diferentes hardware mencionados en la sección anterior:

1. Verificar que no haya errores de compilación previos a la implementación de esta biblioteca.
2. Reemplazar el contenido de las funciones que se encuentran en el archivo fuente de la capa de abstracción de hardware y verificar que no haya errores de compilación.
3. Transmitir correctamente datos únicamente utilizando código Morse.
4. Transmitir correctamente datos utilizando código Morse y códigos de Huffman de manera combinada.
5. Transmitir correctamente datos únicamente utilizando códigos de Huffman.
6. Verificar las velocidades promedio de transmisión de los dígitos “1” y “0” en los códigos Huffman.

7. Se descarten los caracteres que intenten copiarse a los buffer de Morse o Huffman cuando no haya capacidad disponible.
8. Se descarten los caracteres recibidos que sean considerados inválidos según el criterio de 3.1.

4.1.3. Relación variables dependientes e independientes

Cada mensaje que se solicite sea transmitido a través de código Morse, deberá corresponder con la codificación Morse oficial que se presenta en la tabla 2.2.

El tiempo de transmisión de los mensajes en Morse estará sujeto a cual haya sido la unidad de tiempo base que se haya configurado en cada implementación, basándose en las condiciones de la tabla 2.1.

4.1.4. Limitaciones

1. Los mensajes a transmitir a través de cualquiera de los 2 métodos, Morse o Huffman, serán definidos por los desarrolladores pudiendo ser cadenas de texto, como por ejemplo: “Error en el sensor de temperatura”, o que los mensajes obedezcan a un sistema de códigos de error o eventos previamente establecidos como por ejemplo: “E05”. Por tanto no es posible acotar el tipo de mensajes finales que se estarán transmitiendo.
2. El tamaño máximo de caracteres a transmitir de un solo mensaje o de un conjunto de mensajes siempre estará sujeto al tamaño del buffer de datos, que por defecto tendrá un tamaño de dieciséis caracteres. Los caracteres que no entren dentro de la capacidad disponible serán ignorados.

4.1.5. Población y muestra

Como ya se mencionó en la sección 4.1.1, se utilizarán cuatro diferentes plataformas de Hardware para los propósitos de pruebas donde cualquiera de ellos puede desempeñar el rol del Microcontrolador emisor descrito en la figura 3.3:

1. Microcontrolador Microchip PIC16F15313.
2. Plataforma NodeMCU ESP8266.

3. Placa de microcontrolador Arduino UNO.
4. Ordenador de placa reducida Raspberry PI 3.

4.1.6. Funcionalidades o requisitos

1. El sistema será capaz de transmitir correctamente mensajes a través de los métodos Morse y Huffman.
2. El peor caso de un carácter transmitido en código Huffman debe ser menor de $200\ \mu\text{s}$.
3. La tasa de error en la transmisión Morse debe ser menor del 1 %, mientras que la tasa de error en la transmisión de códigos Huffman debe ser menor del 5 %.

4.1.7. Validación

El método Morse de la biblioteca propuesta puede llegar a ser interpretado directamente por un humano a través del ojo, aunque necesariamente dicha persona necesitaría tener el conocimiento sobre los códigos Morse de cada carácter, mientras que el método Huffman es imposible de ser interpretado por un humano debido a la velocidad con la que se transmiten sus datos, que como ya se ha mencionado anteriormente, incluso su peor caso debe completar su transmisión en menos de $200\ \mu\text{s}$.

Es por esto que como procedimiento de validación se definieron los siguientes pasos:

1. Conectar la salida configurada en el microcontrolador a ser utilizada por la presente biblioteca al canal 0 y la línea de tierra a la tierra del microcontrolador de un analizador lógico compatible con el software “Logic 2”, propiedad de la compañía Saleae.
2. Instalar el software “Logic 2” en la computadora donde se va a conectar el analizador lógico y conectar a ella dicho dispositivo mediante el cable USB que requiera según el modelo.

3. Luego de comenzar la ejecución del programa en el microcontrolador, comenzar a capturar en el software los cambios de estado del canal 0 durante el tiempo que se requiera realizar la captura de datos.

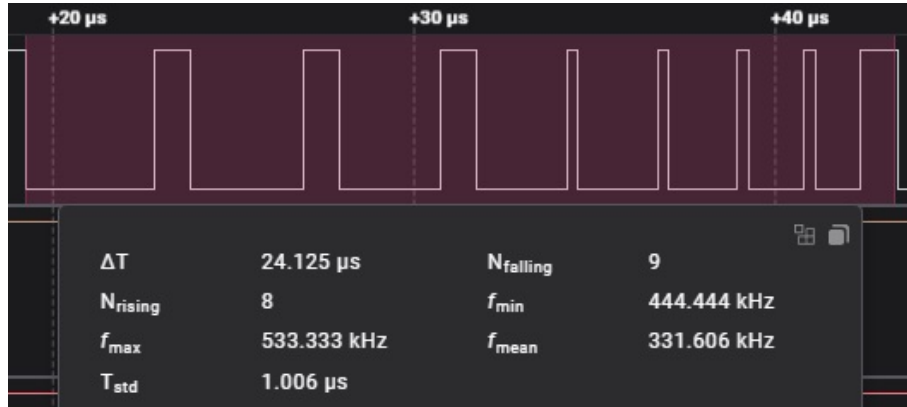


Figura 4.2: Toma de pantalla del analizador lógico de la transmisión del caracter “Q” usando el código Huffman “11100001”.

4. Una vez finalizada la sesión de captura, exportar los datos a un archivo .CSV con todas las opciones en sus valores por defecto dentro de la ventana de exportación de datos, manteniendo el nombre por defecto del archivo como “digital.csv” que contiene todos los registros de cambio de estado del canal 0, tal como se muestra en la figura 4.3.

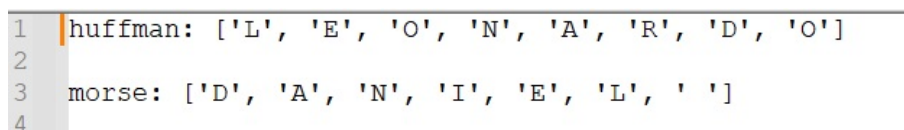
	A	B	C	D	E	F	G	H	I
1	Time [s]	Channel 0	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
2	0	0	1	1	1	1	1	1	1
3	4.31662271	1	1	1	1	1	1	1	1
4	4.31662371	0	1	1	1	1	1	1	1
5	4.31662683	1	1	1	1	1	1	1	1
6	4.31662783	0	1	1	1	1	1	1	1
7	4.31663067	1	1	1	1	1	1	1	1
8	4.31663167	0	1	1	1	1	1	1	1
9	4.31663417	1	1	1	1	1	1	1	1
10	4.3166345	0	1	1	1	1	1	1	1
11	4.31663667	1	1	1	1	1	1	1	1
12	4.31663767	0	1	1	1	1	1	1	1
13	4.31663917	1	1	1	1	1	1	1	1
14	5.81752654	0	1	1	1	1	1	1	1
15	5.81753008	1	1	1	1	1	1	1	1
16	5.81753042	0	1	1	1	1	1	1	1
17	5.81753354	1	1	1	1	1	1	1	1
18	5.81753454	0	1	1	1	1	1	1	1
19	5.81753733	1	1	1	1	1	1	1	1

Figura 4.3: Toma de pantalla del contenido de un archivo “digital.csv”.

5. En la misma carpeta donde quede almacenado el archivo .CSV del punto anterior, ejecutar el archivo “CSVtranslate.py”, que fue creado para leer todos los datos transmitidos durante la captura del analizador lógico, tanto en Morse

como en Huffman respectivamente, mostrando los resultados en un nuevo archivo generado como salida: “morsehuffman.translation.txt” del cual se puede ver un ejemplo en la figura 4.4. Este archivo de python contiene cuatro variables de configuración que pueden ser ajustadas de acuerdo a las necesidades de cada uso:

- a) `cfg_huffman_0_max_time`: Indica la duración máxima que puede tener la transmisión de un dígito “0” en Huffman, expresando en segundos. Por defecto tiene un valor de “5e-07”.
- b) `cfg_huffman_1_max_time`: Indica la duración máxima que puede tener la transmisión de un dígito “1” en Huffman, expresado en segundos. Por defecto tiene un valor de “1e-05”.
- c) `cfg_morse_unit_time_s`: Indica la duración de la unidad de tiempo Morse, recordando que un punto toma una unidad y una línea tres unidades y se expresa en segundos. Por defecto tiene un valor de “0.5”.
- d) `cfg_file_path_and`: Define el nombre del archivo .CSV que se espera leer como dato de entrada. Por defecto tiene un valor de “digital.csv”.



```

1 huffman: ['L', 'E', 'O', 'N', 'A', 'R', 'D', 'O']
2
3 morse: ['D', 'A', 'N', 'I', 'E', 'L', ' ']
4

```

Figura 4.4: Toma de pantalla de un ejemplo del contenido de un archivo “morsehuffman.translation.txt” generado con el código python “CSVtranslate.py”.

4.1.8. Pruebas unitarias

Se requieren diseñar pruebas unitarias específicas para esta biblioteca abarcando los siguientes casos:

1. Realizar una ejecución del sistema transmitiendo únicamente a través de código Morse las siguientes dos cadenas enviadas de manera independiente: “E01” y “E02” y confirmar el porcentaje de error en la transmisión de los datos mediante el uso del archivo de Python “CSVtranslate.py”.
2. Realizar una ejecución del sistema transmitiendo a través de código Morse en las siguientes dos cadenas enviadas de manera independiente: “E01” y “E02”

e inmediatamente después la cadena “VOLUNTARIAMENTE” de quince caracteres de longitud utilizando código Huffman y confirmar el porcentaje de error en la transmisión de los datos de cada método individualmente mediante el uso del archivo de Python “CSVtranslate.py”.

3. Realizar una ejecución del sistema transmitiendo a través de código Morse la cadena “SI” e inmediatamente después la cadena “QUEJAS” utilizando código Huffman de manera periódica cada ocho segundos durante treinta minutos y confirmar el porcentaje de error en la transmisión de los datos de cada método Morse y Huffman individualmente mediante el uso del archivo de Python “CSVtranslate.py”.
4. Usar el método Huffman sin usar el método Morse transmitiendo la cadena “VOLUNTARIAMENTE” y confirmar el porcentaje de error en la transmisión de los datos mediante el uso del archivo de Python “CSVtranslate.py”.
5. Saturar el tamaño de los buffer de los métodos Morse y Huffman enviando a ambos métodos dos veces la cadena “MURCIELAGO” de diez caracteres de longitud y confirmar mediante el archivo de Python “CSVtranslate.py” el porcentaje de error, verificando también que los caracteres que hayan sobrepasado la capacidad disponible del buffer sean ignorados, esperando sean los últimos seis caracteres de la segunda cadena: “IELAGO” en ambos métodos, debido a que al final de cada cadena transmitida se inserta un espacio en blanco, incluso si es necesario rechazando un carácter en el buffer para dar lugar al espacio en blanco, por lo que la salida esperada sería “MURCIELAGO MURC”, con un espacio en blanco después de la última “C”.
6. Enviar un carácter inválido, enviando a ambos métodos la cadena “PAÑUELO” y confirmar mediante el uso del archivo de Python “CSVtranslate.py” el porcentaje de error, esperando que los datos se hayan transmitido correctamente omitiéndose letra “Ñ” en Huffman y sustituyéndola por un espacio en Morse.
7. Transmitir únicamente por código de Huffman el carácter que tenga el peor caso esperado en cuanto a tiempo de transmisión, el cual en este caso sería la

letra “Q” y reportar el tiempo moda de transmisión luego de ocho transmisiones del caracter.

8. Reportar el tiempo moda de transmisión de los dígitos “0” en la transmisión de códigos Huffman.
9. Reportar el tiempo moda de transmisión de los dígitos “1” en la transmisión de códigos Huffman.
10. Reportar el tiempo moda de espacio entre pulsos de la transmisión de cualquier dígito ‘en la transmisión de códigos Huffman.

4.1.9. Configuración de dispositivos

A continuación, en la tabla 4.1 se presenta una comparativa de los valores utilizados en cada uno de los dispositivos probados, teniendo en cuenta los siguientes elementos: velocidad de reloj del microcontrolador, cantidad de bits del microcontrolador, voltaje de operación, GPIO utilizado durante las pruebas y la presencia de algún sistema operativo, en caso de que aplique.

Prueba	Arduino UNO	PIC16F15313	ESP8266	Raspberry PI 3
Velocidad de reloj	16 MHz	35 MHz	80 MHz	400 MHz
Bits de microcontrolador	8	8	32	64
Voltaje operación	5V	4.5V	5	5
GPIO usado	13	RA2	2	17
Sistema Operativo	-	-	-	Raspbian

Tabla 4.1: Parámetros de configuración usados en cada hardware.

En el diagrama 4.5 se puede observar el diseño sugerido de hardware para las pruebas utilizando el Arduino UNO. En la fotografía 4.6 se muestra el mismo diseño ya con el hardware real.

En la fotografía 4.7 se puede observar el LED encendido cuando es requerido mostrar un pulso en alto durante la comunicación Morse. Las variaciones de estado de la salida durante la transmisión de mensajes usando códigos Huffman no son perceptible en un LED para el ojo humano y es prácticamente imposible capturarlo

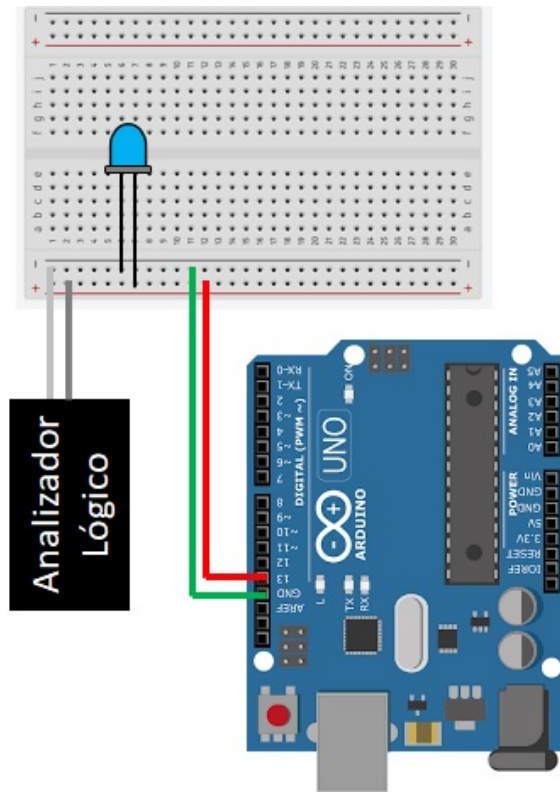


Figura 4.5: Diagrama para pruebas de comunicación Morse y Huffman usando un Arduino UNO, una placa de pruebas y un analizador lógico.

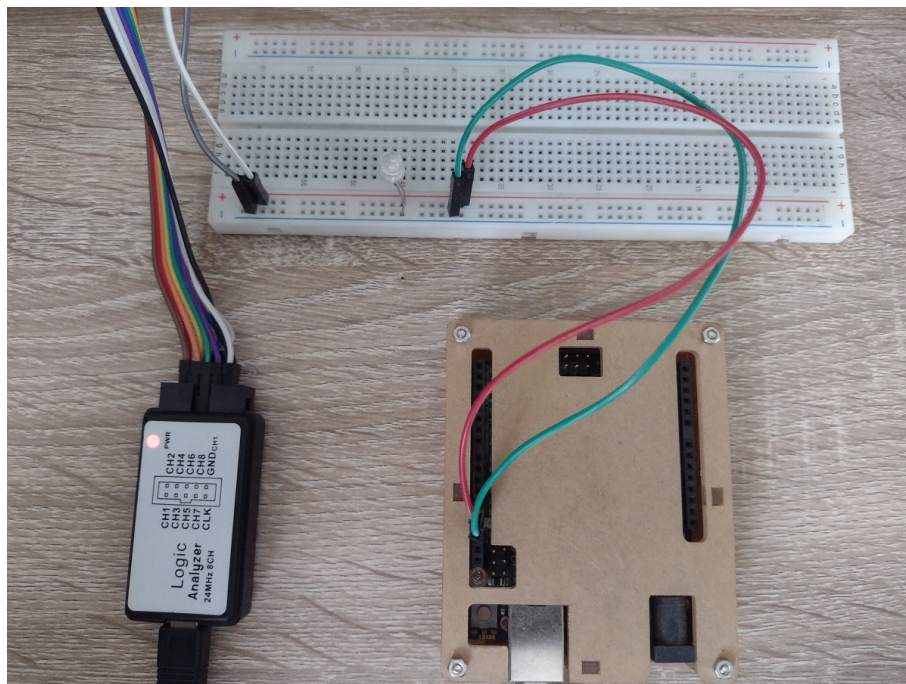


Figura 4.6: Conexión de Arduino UNO con carcasa protectora, placa de pruebas, LED de color azul y un analizador lógico.

en una fotografía dada la velocidad de variación del estado de la salida, que aún en el peor de los escenarios, el tiempo para transmitir un carácter completo es siempre

menor a $200\ \mu\text{s}$ tomando como unidad de tiempo base $1\ \mu\text{s}$.

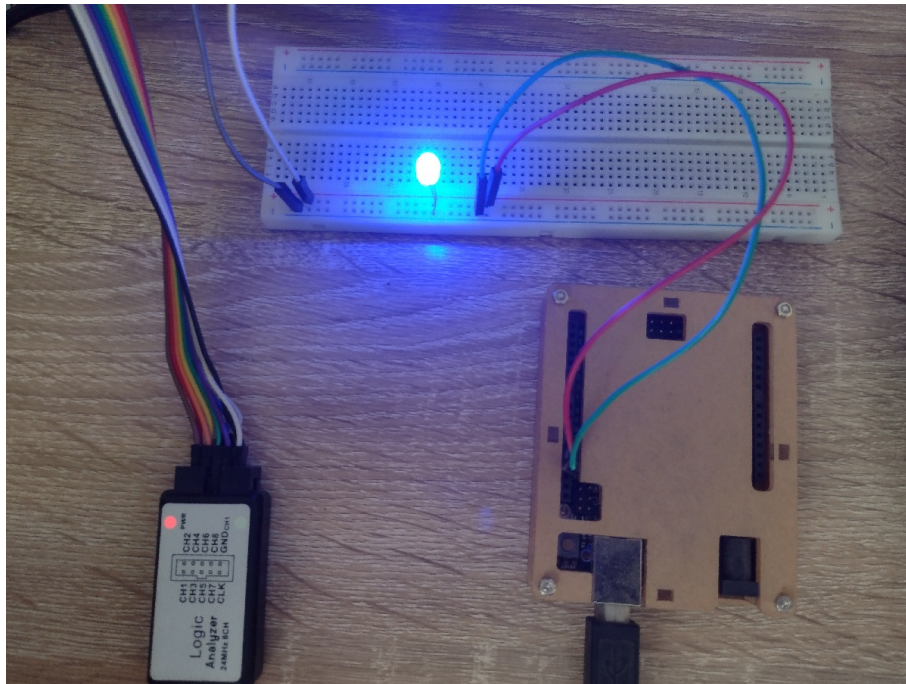


Figura 4.7: LED de color azul encendido para representar un pulso Morse.

4.1.10. Tabla de resultados

En la siguiente tabla 4.2, se presentan los resultados de cada uno de los dispositivos probados ejecutando las pruebas unitarias descritas en la sección 4.1.8. En resumen, las primeras seis pruebas arrojan como resultado un porcentaje de error en la transmisión de datos. Por lo tanto, cuanto más cercano al cero sea el resultado, mejor será. En las últimas cuatro pruebas, la evaluación se realiza en función del tiempo, donde también se prefieren resultados más cercanos al cero. Para tener una evaluación integral, es necesario obtener resultados positivos en ambos aspectos. No es deseable tener transmisiones muy veloces con poca fiabilidad.

Prueba	Arduino UNO	PIC16F15313	ESP8266	Raspberry PI 3
Prueba 1	0 %	0 %	0 %	0 %
Prueba 2	0 %	0 %	0 %	0 %
Prueba 3	0 %,0 %	0 %,0 %	0 %,0 %	0.27 %,2.73 %
Prueba 4	0 %	0 %	0 %	0 %
Prueba 5	0 %	0 %	0 %	0 %
Prueba 6	0 %	0 %	0 %	0 %
Prueba 7	20.583 μ s	102.917 μ s	23.833 μ s	12.208 μ s
Prueba 8	0.333 μ s	6 μ s	2.167 μ s	1 μ s
Prueba 9	1 μ s	7.125 μ s	3.042 μ s	1.833 μ s
Prueba 10	3.125 μ s	9.083 μ s	0.417 μ s	0.125 μ s

Tabla 4.2: Tabla comparativa de resultados de las diez pruebas descritas en la sección 4.1.8 en los diferentes hardware donde se probó esta biblioteca.

4.2. Discusión

Como se puede observar en los resultados de la tabla 4.2, la biblioteca funcionó sin problemas en las cuatro plataformas. Solo fue necesario modificar la capa de abstracción de hardware para utilizar las interfaces propias de cada dispositivo.

Las mediciones de transmisión Morse no se consideraron, ya que en todas las plataformas esta transmisión está sujeta a la unidad de tiempo base configurada. Por lo tanto, en todas ellas, el comportamiento de tiempos de transmisión fue el mismo.

En cuanto a la transmisión de códigos Huffman, esta ocurrió sin errores en todos los casos de prueba de todos los dispositivos, excepto en la prueba de transmisión de datos durante treinta minutos continuos utilizando la Raspberry Pi 3. A pesar de esto, los márgenes de error estuvieron dentro de los límites definidos en la sección 4.1.6. Cabe recordar que este fue el único hardware que se utilizó sobre un sistema operativo completo, Raspbian, como ya se señaló en la sección 4.1.

La Raspberry Pi 3 fue el dispositivo que transmitió los códigos Huffman más rápido, aunque también fue el único que presentó un porcentaje de error diferente

a 0 %. El microcontrolador más lento fue el PIC16F15313, que es un dispositivo de capacidades limitadas.

Arduino UNO y ESP8266 tuvieron resultados muy similares en cuanto al tiempo de transmisión del peor caso de carácter en código Huffman. Esto se debe a que Arduino UNO tuvo un mejor desempeño en el tiempo de transmisión de los dígitos que los tiempos de ESP8266, pero Arduino UNO tuvo tiempos de separación entre pulsos más largos. Estas condiciones equilibraron los tiempos en ambos dispositivos.

Haciendo una evaluación integral de los resultados, se puede afirmar que de los cuatro dispositivos utilizados durante las pruebas, la biblioteca se comportó mejor en Arduino UNO, ya que mantuvo un porcentaje de error del 0 % en todas las pruebas relacionadas con este aspecto. Entre los dispositivos con el mismo porcentaje de error, tuvo los mejores tiempos, con excepción de la prueba diez, donde presentó un mejor desempeño el ESP8266.

En términos de fiabilidad, se puede concluir que en los microcontroladores que no contaban con un sistema operativo, el porcentaje de error se mantuvo en 0 %. Los servicios básicos de los sistemas operativos pueden provocar un desfase en el tiempo de ejecución de la máquina de estados principal de la biblioteca, provocando errores en la transmisión. Se sugiere el uso de esta biblioteca en un ambiente sin sistema operativo o, en su defecto, ajustar las prioridades de las tareas en el mismo sistema operativo para evitar este tipo de problemas.

Se concluye también que la biblioteca fue compatible con los cuatro microcontroladores usados en estas pruebas, pero se puede inferir que la compatibilidad alcanzará a cualquier dispositivo que pueda ejecutar código C, dado que no se identificaron limitantes más allá de adaptar apropiadamente la capa de abstracción de hardware HAL para cada microcontrolador.

Aunque los tiempos de transmisión puedan parecer cortos, su nivel de intrusión dependerá del propósito final del sistema donde se ejecute la biblioteca y de su con-

figuración particular. Por ejemplo, en un sistema donde el tiempo de respuesta es crítico, $102.917\mu s$ para transmitir un carácter, que es lo que tarda el PIC16F15313, pudiera ser inaceptable. Aunque hipotéticamente, desde el inicio, difícilmente se elegiría un microcontrolador tan limitado para un sistema donde el tiempo de respuesta sea tan importante.

Capítulo 5

Conclusiones y Perspectivas

5.1. Conclusiones

La biblioteca funciona como se concibió, haciendo uso de una sola salida del microcontrolador para transmitir directamente al humano código Morse mediante el uso de un LED, y para transmitir información a gran velocidad mediante el uso de códigos Huffman. Incluso se pueden combinar ambos métodos en la misma ejecución, lo que demuestra un desarrollo satisfactorio desde el punto de vista funcional.

Además, se cumple con el propósito descrito en la sección 1.8 de justificación, al ser una alternativa de bajo costo orientada a fines académicos. Al día de hoy, un LED tiene un costo de 0.25 dólares, y un analizador lógico genérico, como el utilizado en el desarrollo de este proyecto, tiene un costo promedio inferior a 13 dólares.

El proceso de adaptar la biblioteca a diferentes plataformas de hardware se simplifica mediante la actualización de la capa de abstracción de hardware. Esto implica adaptar las tres interfaces incluidas en el archivo “morsehuffman_hal.c”, mencionado en la sección 3.4. El proceso de actualización está documentado en el archivo “README.md” incluido en el repositorio de la biblioteca.

No se pretende proponer esta biblioteca como un reemplazo absoluto de todos los métodos existentes para conocer los eventos que suceden durante la ejecución de un sistema embebido. El objetivo es ofrecer una alternativa más que puede ser útil en

ciertos contextos donde los recursos son escasos, como en muchos casos dentro del entorno académico y en algunos casos de la industria, donde el tiempo de respuesta puede ser clave.

La biblioteca está publicada en la plataforma "GitHub" con su respectiva documentación en inglés, disponible para la mayor cantidad de personas posible en la dirección: "https://github.com/danubio23/morsehuffman_lib".

5.2. Perspectivas

Como perspectivas del trabajo futuro, se podría considerar migrar el código de la presente biblioteca a otros lenguajes de programación que permitan alcanzar otras plataformas de hardware. También se podría contemplar la implementación de un método en tiempo real para descifrar la comunicación de código Morse y código Huffman del dispositivo emisor. Esto se lograría utilizando un segundo microcontrolador con una entrada configurada con interrupciones, conectada a la salida del dispositivo emisor y conociendo los alfabetos de entrada de ambos métodos, Morse y Huffman.

Durante el desarrollo de esta biblioteca, no se optó desde el inicio por este enfoque, ya que implicaría la inclusión de un segundo microcontrolador, contradiciendo parte del objetivo general 1.4 definido, que busca obtener una biblioteca de baja complejidad y bajo costo de implementación.

Siempre es posible personalizar el alfabeto de entrada de los códigos Huffman para priorizar códigos más cortos según el criterio del desarrollador que implemente la biblioteca en su sistema. Incluso se puede reducir el alfabeto para obtener códigos más cortos y lograr ahorros de tiempo en las transmisiones de datos de este método.

Bibliografía

- Audsley, N. C., Burns, A., Davis, R. I., Tindell, K. W., and Wellings, A. J. (1995). Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems*, 8(2):173–198.
- Carron, L. (1991). *Morse Code: The Essential Language*. Gas Engineering and Operating Practices. American Radio Relay League.
- Dictionary, O. (2012). *What is the frequency of the letters of the alphabet in English?* Oxford University Press.
- Ens, B. and Irani, P. (2016). Spatial analytic interfaces: Spatial user interfaces for in situ visual analytics. *IEEE computer graphics and applications*, 37(2):66–79.
- Gracioli, G. and Fischmeister, S. (2009). Tracing interrupts in embedded software. *ACM Sigplan Notices*, 44(7):137–146.
- Guenther, J. A. (1973). *Machine recognition of hand-sent Morse code using the PDP-12 computer*. National Technical Information Service, US Department of commerce, Springfield, Virginia.
- Ingels, F. M. (1971). *Information and coding theory*. Intext Educational Publishers.
- Kraft, J., Wall, A., and Kienle, H. (2010). Trace recording for embedded systems: Lessons learned from five industrial projects. In *International Conference on Runtime Verification*, pages 315–329. Springer.
- Lamichhane, K., Moreno, C., and Fischmeister, S. (2018). Non-intrusive program tracing of non-preemptive multitasking systems using power consumption. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1147–1150. IEEE.
- Lelewer, D. A. and Hirschberg, D. S. (1987). Data compression. *ACM Computing Surveys (CSUR)*, 19(3):261–296.
- Lewis, C. and Short, C. (1879). *Latin dictionary*. Oxford, Clarendon Press.

- Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61.
- Lu, S., Qian, G., He, Q., Liu, F., Liu, Y., and Wang, Q. (2019). In situ motor fault diagnosis using enhanced convolutional neural network in an embedded system. *IEEE Sensors Journal*, 20(15):8287–8296.
- M.1677-1, R. (2009). *International Morse Code*. International Telecommunication Union.
- Malony, A. D., Reed, D. A., and Wijshoff, H. A. G. (1992). Performance measurement intrusion and perturbation analysis. *IEEE Transactions on Parallel & Distributed Systems*, 3(04):433–450.
- Moere, A. V. and Hill, D. (2012). Designing for the situated and public visualization of urban data. *Journal of Urban Technology*, 19(2):25–46.
- Moffat, A. (2019). Huffman coding. *ACM Computing Surveys (CSUR)*, 52(4):1–35.
- Puschner, P. and Koza, C. (1989). Calculating the maximum execution time of real-time programs. *Real-time systems*, 1(2):159–176.
- Ravikumar, C. and Dathi, M. (2016). A fuzzy-logic based morse code entry system with a touch-pad interface for physically disabled persons. In *2016 IEEE Annual India Conference (INDICON)*, pages 1–5. IEEE.
- Seo, M. and Kurdahi, F. (2019). Efficient tracing methodology using automata processor. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–18.
- Seo, M. and Lysecky, R. (2018). Non-intrusive in-situ requirements monitoring of embedded system. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 23(5):1–27.
- Thane, H. (2000). *Monitoring, testing and debugging of distributed real-time systems*. PhD thesis, Maskinkonstruktion.
- Velasco, J. S., Credo, M., Salazar, A. M. P., De Ocampo, S. R. S., Enriquez, V. J. C., Padilla, M. V. C., Galido, E. A., Arago, N. M., Virrey, G. C., and Tolentino, L. K. S. (2021). Utilization of huffman coding for weather buoy system using long-range communication. *International Journal of Advanced Technology and Engineering Exploration*, 8(79).
- White, S. M. (2009). *Interaction and presentation techniques for situated visualiza-*

tion. Columbia University.

Xu, J. and Parnas, D. L. (1990). Scheduling processes with release times, deadlines, precedence and exclusion relations. *IEEE Transactions on software engineering*, 16(3):360–369.

Anexos A

Códigos

A.1. morsehuffman_lib.h

```
#ifndef morsehuffman_lib_h_
#define morsehuffman_lib_h_

typedef enum
{
    enum_opmode_morse,
    enum_opmode_huffman
}enum_op_modes;

typedef enum_op_modes op_modes;

void init_MorseHuffman(unsigned char led_id);
void morsehuffman_fsm(void);
void morsehuffman_msg(const char *str, op_modes mode);

#endif
```

A.2. morsehuffman_lib.c

```
#include "morsehuffman_lib.h"
#include "morsehuffman_hal.h"
#include "morsehuffman_cfg.h"

/*END configurable parameters*/
```

```

/*Definitions*/
#define def_huffman_us_1 (def_cfg_huffman_us_0*2u)
#define def_huffman_us_idle def_cfg_huffman_us_0
#define def_val_dot 0x8000u
#define def_val_dash 0xC000u
#define def_val_end 0x0000u
#define def_val_blank 0x0001u
#define def_val_invalid 0xFFFFu
#define def_offset_space_char 1u
#define def_space_ascii 32u
#define def_led_state_off 0u
#define def_led_state_on 1u
#define def_offset_digit 48u
#define def_offset_capitalletter 55u
#define def_offset_lowercaseletter 87u
#define def_symbol_blank 36u
#define def_symbol_invalid 37u
#define def_symbol_max 38u
#define def_fsm_state_idle 0u
#define def_fsm_state_processing 1u
#define def_fsm_state_print_dash 2u
#define def_fsm_state_print_dot 3u
#define def_ascii_val_0 48u
#define def_ascii_val_9 57u
#define def_ascii_val_cap_A 65u
#define def_ascii_val_cap_Z 90u
#define def_ascii_val_low_a 97u
#define def_ascii_val_low_z 122u

/*Prototypes*/
static void huffmanprint_func(void);
static void morsehuffman_pop(unsigned short * value, unsigned char * bits,
↪ op_modes mode);
static unsigned char index_translation(unsigned char par0);

/*Global variable declaration*/
typedef struct
{

```



```

    unsigned char value;
    unsigned char bits;
}huffmantable;

static char morsebuffer[def_cfg_buffer_size] = {0};
static char huffmanbuffer[def_cfg_buffer_size] = {0};
static unsigned int morseStrLenVar = 0;
static unsigned int huffmanStrLenVar = 0;

/*Local functions*/
static unsigned char index_translation(unsigned char par0)
{
    unsigned char index;

    if((par0 >= def_ascii_val_0) && (par0 <= def_ascii_val_9)) //0 - 9 ascii
        ↪ range
    {
        index = par0 - def_offset_digit;
    }
    else if((par0 >= def_ascii_val_cap_A) && (par0 <= def_ascii_val_cap_Z)) //A -
        ↪ Z ascii range
    {
        index = par0 - def_offset_capitalletter;
    }
    else if((par0 >= def_ascii_val_low_a) && (par0 <= def_ascii_val_low_z)) //a -
        ↪ z ascii range
    {
        index = par0 - def_offset_lowercaseletter;
    }
    else if(def_space_ascii == par0) //space
    {
        index = def_symbol_blank;
    }
    else
    {
        index = def_symbol_invalid;
    }

    return index;
}

```

```

}

static void morsehuffman_pop(unsigned short * value, unsigned char * bits,
↪ op_modes mode)
{
    char *ptrBuffer;
    unsigned char index;
    const huffmantable st_huffmantable[def_symbol_max] =
    {
        {0x20u,4}, //0
        {0x10u,4}, //1
        {0x50u,4}, //2
        {0x30u,4}, //3
        {0x70u,4}, //4
        {0x00u,5}, //5
        {0x08u,5}, //6
        {0xA0u,5}, //7
        {0x90u,5}, //8
        {0x88u,5}, //9
        {0x48u,5}, //A
        {0xE4u,6}, //B
        {0xD8u,5}, //C
        {0xF8u,5}, //D
        {0x60u,5}, //E
        {0xB4u,6}, //F
        {0xB0u,6}, //G
        {0xC4u,6}, //H
        {0xC8u,5}, //I
        {0xE0u,8}, //J
        {0xC2u,7}, //K
        {0xE8u,5}, //L
        {0x44u,6}, //M
        {0x68u,5}, //N
        {0xA8u,5}, //O
        {0x84u,6}, //P
        {0xE1u,8}, //Q
        {0xD0u,5}, //R
        {0xF0u,5}, //S
        {0x98u,5}, //T
    }

```

```

    {0xB8u, 5}, //U
    {0x40u, 8}, //V
    {0x42u, 7}, //W
    {0xE2u, 7}, //X
    {0xC0u, 7}, //Y
    {0x41u, 8}, //Z
    {0x80u, 6}, //Blank
    {0xFFu, 0} //Invalid
};

const unsigned short au16_symbols_morse[def_symbol_max] =
{
    0xDB6Cu, //0
    0xB6D8u, //1
    0xADB0u, //2
    0xAB60u, //3
    0xAAC0u, //4
    0xAA80u, //5
    0xD540u, //6
    0xDAA0u, //7
    0xDB50u, //8
    0xDB68u, //9
    0xB000u, //Aa
    0xD500u, //Bb
    0xD680u, //Cc
    0xD400u, //Dd
    0x8000u, //Ee
    0xAD00u, //Ff
    0xDA00u, //Gg
    0xAA00u, //Hh
    0xA000u, //Ii
    0xB6C0u, //Jj
    0xD600u, //Kk
    0xB500u, //Ll
    0xD800u, //Mm
    0xD000u, //Nn
    0xDB00u, //Oo
    0xB680u, //Pp
    0xDAC0u, //Qq
    0xB400u, //Rr

```

```

    0xA800u, //Ss
    0xC000u, //Tt
    0xAC00u, //Uu
    0xAB00u, //Vv
    0xB600u, //Ww
    0xD580u, //Xx
    0xD6C0u, //Yy
    0xDA80u, //Zz
    0x0001u, //Blank
    def_val_invalid //Invalid
};

if(enum_opmode_huffman == mode)
{
    ptrBuffer = huffmanbuffer;
}
else
{
    ptrBuffer = morsebuffer;
}

index = index_translation((unsigned char)ptrBuffer[0u]);

if(enum_opmode_huffman == mode)
{
    *value = (unsigned short)st_huffmantable[index].value;
    *bits = st_huffmantable[index].bits;
    huffmanStrLenVar--;
}
else
{
    *value = au16_symbols_morse[index];
    morseStrLenVar--;
}

for(index = 0u; index < (def_cfg_buffer_size-1u); index++)
{
    ptrBuffer[index] = ptrBuffer[index+1u];
}

```

```

    ptrBuffer[def_cfg_buffer_size-1u] = (char)0u;
}

/*Public functions*/
void init_MorseHuffman(unsigned char led_id)
{
    hal_setuppin(led_id);
}

void morsehuffman_fsm(void)
{
    static unsigned char fsm_state = def_fsm_state_idle;
    static unsigned short chartobeprint = 0;
    static unsigned short duration_counter;

    switch(fsm_state)
    {
        case def_fsm_state_idle:
        {
            if(morseStrLenVar > 0u)
            {
                morsehuffman_pop(&chartobeprint, (unsigned char
↵ *)0u, enum_opmode_morse);
                if(def_val_invalid != chartobeprint)
                {
                    fsm_state = def_fsm_state_processing;
                }
            }
            else
            {
                huffmanprint_func();
            }

            }break;

        case def_fsm_state_processing:
        {

```

```

    if(morseStrLenVar == 0u)
    {
        fsm_state = def_fsm_state_idle;
    }
    else
    {
        if((chartobeprint & def_val_dash) == def_val_dash)
        {
            chartobeprint = chartobeprint << 3u;
            duration_counter = 2;
            fsm_state = def_fsm_state_print_dash;
            huffmanprint_func();
            hal_switch_led(def_led_state_on);
        }
        else if((chartobeprint & def_val_dot) == def_val_dot)
        {
            chartobeprint = chartobeprint << 2u;
            fsm_state = def_fsm_state_print_dot;
            huffmanprint_func();
            hal_switch_led(def_led_state_on);
        }
        else
        {
            fsm_state = def_fsm_state_idle;
        }
    }
}break;

case def_fsm_state_print_dash:
{
    if(duration_counter > 0u)
    {
        duration_counter--;
    }
    else
    {
        fsm_state = def_fsm_state_processing;
        huffmanprint_func();
        hal_switch_led(def_led_state_off);
    }
}

```

```

        }
    }break;
    case def_fsm_state_print_dot:
    {
        fsm_state = def_fsm_state_processing;
        huffmanprint_func();
        hal_switch_led(def_led_state_off);
    }break;
    default:
    {
        fsm_state = def_fsm_state_idle;
    }break;
}
}

static void huffmanprint_func(void)
{
    unsigned char bits;
    unsigned short huffmancode;
    unsigned char index;
    unsigned char toPrint = 0u;

    if(0u != huffmanStrLenVar)
    {
        morsehuffman_pop(&huffmancode,&bits,enum_opmode_huffman);

        hal_switch_led(def_led_state_off);

        for(index = 0u; index < bits; index++)
        {
            toPrint = (unsigned char)((huffmancode >> (8u-(1u+index))) & 0x01u);
            hal_switch_led(def_led_state_on);
            if(1u == toPrint)
            {
                hal_delayexecution_us((unsigned char)def_huffman_us_1);
            }
            else
            {
                hal_delayexecution_us((unsigned char)def_cfg_huffman_us_0);
            }
        }
    }
}

```

```

    }
    hal_switch_led(def_led_state_off);
    hal_delayexecution_us((unsigned char)def_huffman_us_idle);
}
}
}

void morsehuffman_msg(const char *str, op_modes mode)
{
    unsigned char index;
    unsigned char lengthval = 0;
    unsigned int *ptr_StrLenVar;
    char *ptr_buff_string;

    while(str[lengthval] != 0)
    {
        lengthval++;
    }

    if((str != (const char *)0u) && (lengthval > 0u))
    {
        if(enum_opmode_huffman == mode)
        {
            ptr_buff_string = huffmanbuffer;
            ptr_StrLenVar = &huffmanStrLenVar;
        }
        else
        {
            ptr_buff_string = morsebuffer;
            ptr_StrLenVar = &morseStrLenVar;
        }

        if((def_cfg_buffer_size - *ptr_StrLenVar - def_offset_space_char)
            ↪ < lengthval)
        {
            lengthval = def_cfg_buffer_size - *ptr_StrLenVar -
            ↪ def_offset_space_char;
        }
    }
}

```



```

    for(index = 0u; index < (lengthval+def_offset_space_char);
        ↪ index++)
    {
        if(index < lengthval)
        {
            ptr_buff_string[*ptr_StrLenVar+index] =
            ↪ str[index];
        }
        else
        {
            ptr_buff_string[*ptr_StrLenVar+index] =
            ↪ (char)def_space_ascii;
        }
    }
    *ptr_StrLenVar += (unsigned
    ↪ int)(lengthval+def_offset_space_char);
}
}

```

A.3. morsehuffman_cfg.h

```

#ifndef morsehuffman_cfg_h_
#define morsehuffman_cfg_h_

/*Configurable parameters*/
#define def_cfg_buffer_size 16u
#define def_cfg_huffman_us_0 1u

#endif

```

A.4. morsehuffman_hal.h

```

#ifndef morsehuffman_hal_h_
#define morsehuffman_hal_h_

#include "Arduino.h"

void hal_switch_led(unsigned char state);

```

```

void hal_delayexecution_us(unsigned char us);
void hal_setuppin(unsigned char led_id);

#endif

```

A.5. morsehuffman_hal.c

```

/*File configured for Arduino UNO platform*/

#include "morsehuffman_hal.h"

char outputled = 13;

/*Abstraction layer*/
void hal_switch_led(unsigned char state)
{
    PINB = (PINB | (state << 5));
}

void hal_delayexecution_us(unsigned char us)
{
    delayMicroseconds(us);
}

void hal_setuppin(unsigned char led_id)
{
    outputled = led_id;
    pinMode(led_id, OUTPUT);
}

```

A.6. Algoritmos y Pseudocódigos

A.6.1. Funciones privadas

A.6.2. Funciones publicas

Algorithm 1: huffmanprint_func

Descripción: En caso de que haya datos en el buffer del método Huffman, toma el siguiente carácter de dicho buffer y lo transmite.

Entradas: -

```
1: if datos en el buffer huffman  $\neq 0$  then
    2: obtiene bits a ser transmitidos
    3: inicializa estado de la salida del microcontrolador a APAGADO
    4: for bits a ser transmitidos  $\neq 0$  do
        5: cambia estado de la salida del microcontrolador a ENCENDIDO
        6: if bit transmitido  $== 1$  then
            | 7: retrasar ejecución 2 unidades de tiempo
        else
            | 8: retrasar ejecución 1 unidad de tiempo
        end
        9: cambiar estado de la salida del microcontrolador a APAGADO
        10: retrasar ejecución 1 unidad de tiempo
    end
end
```

Algorithm 2: morsehuffman_pop

Descripción: Revisa el buffer correspondiente de acuerdo a la entrada recibida, valida si hay un carácter valido a transmitir y encuentra su equivalencia en bits de acuerdo al método recibido en los parámetros de entrada.

Entradas: modo de operación

```
1: if modo de operación  $==$  huffman then
    | 2: apuntar a buffer de método Huffman
else
    | 3: apuntar a buffer de método Morse
end
4: verifica si el siguiente caracter en el buffer apuntado es valido
5: if modo de operación  $==$  huffman then
    | 6: devuelve los bits correspondientes y el numero de bits
else
    | 7: devuelve los bits correspondientes
end
8: se reduce el tamaño del buffer apuntado
9: se recorren los caracteres todos hacia el índice menor inmediato
```

Algorithm 3: index_translation

Descripción: Función que verifica el índice correcto dentro del arreglo de caracteres validos para el caracter recibido como parámetro de entrada o si en su defecto es un caracter invalido.

Entradas: caracter a evaluar

```
1: if caracter a evaluar == dígito numero entre 0 y 9 then  
  | 2: index = caracter a evaluar - 48  
else  
  | 3: if caracter a evaluar == letra entre A y Z mayúsculas then  
    | 4: index = caracter a evaluar - 36  
  else  
    | 5: if caracter a evaluar == letra entre a y z minúsculas then  
      | 6: index = caracter a evaluar - 87  
    else  
      | 7: if caracter a evaluar == espacio en blanco then  
        | 8: index = 36  
      else  
        | 9: index = invalido  
      end  
    end  
  end  
end  
end
```

Algorithm 4: Init_MorseHuffman

Descripción: Función de inicialización.

Entradas: - 1: Llama a la función HAL que inicializa la salida del microcontrolador seleccionada para operar la presente librería.

Algorithm 5: morsehuffman_fsm

Descripción: Maquina de estado principal.

Entradas: - 1: **switch** *fsm_state* **do**

2: **case** *idle* **do**

3: **if** *morseStrLenVar* $\neq 0$ **AND** *caracter a transmitir* $==$ *valido*

then

 | 6: *fsm_state* $==$ *processing*

else

 | 7: Transmite caracter por Huffman si aplica

end

end

8: **case** *processing* **do**

9: **if** *simbolo a transmitir* $==$ *dash* **AND** *morseStrLenVar* $== 0$

then

 | 10: caracter a ser transmitido recorre 2 bits a izquierda

 | 11: *duration_counter* = 2

 | 12: *fsm_state* $==$ *dash*

 | 13: Transmite caracter por Huffman si aplica

 | 14: Cambia estado de salida del a ENCENDIDO

else

 | 15: **if** *simbolo a transmitir* $==$ *dot* **AND** *morseStrLenVar* $== 0$

then

 | 16: caracter a ser transmitido recorre 2 bits a izquierda

 | 17: *fsm_state* $==$ *dot*

 | 18: Transmite caracter por Huffman si aplica

 | 19: Cambia estado de salida a ENCENDIDO

else

 | 20: *fsm_state* $==$ *idle*

end

end

end

21: **case** *print_dash* **do**

22: **if** *duration_counter* $\neq 0$ **then**

 | 23: *duration_counter* = *duration_counter* - 1

else

 | 24: *fsm_state* $==$ *processing*

 | 25: Transmite caracter por Huffman si aplica

 | 26: Cambia estado de salida a APAGADO

end

end

27: **case** *print_dot* **do**

28: *fsm_state* $==$ *processing*

29: Transmite caracter por Huffman si aplica

30: Cambia estado de salida a APAGADO

end

31: **case** *default* **do**

 | 32: *fsm_state* $==$ *idle*

end

end

Algorithm 6: morsehuffman_msg

Descripción: Función utilizada por los clientes cuando requieran transmitir algún caracter por el método seleccionado a través de los parámetros de entrada.

Entradas: Cadena de datos a transmitir, método de transmisión 1: Se obtiene la longitud de la cadena recibida como parámetro de entrada.

2: **if** *Cadena recibida* \neq *nulo* **AND** *longitud de cadena recibida* \neq 0 **then**

 3: **if** *método de transmisión* == *Huffman* **then**

 | 4. Punteros internos apuntan a buffer Huffman

else

 | 5. Punteros internos apuntan a buffer Morse

end

6: **if** *Espacio disponible en buffer* \wedge *longitud de cadena recibida* **then**

 | 7: Se ajusta la longitud de la cadena recibida de acuerdo al espacio disponible en el buffer correspondiente.

end

8. Se copia la cadena recibida al buffer correspondiente. 9. Se agrega un espacio en blanco al final de la cadena.

end

Declaración expresa

La responsabilidad por los hechos, ideas y doctrinas expuestas en este proyecto corresponden exclusivamente a los directamente involucrados en su desarrollo, y el patrimonio intelectual de la misma a la Universidad Autónoma de Guadalajara.

CARTA DE DECLARACIÓN DE ORIGINALIDAD

Zapopan, Jalisco a 05 de Septiembre del 2023

Lina María Aguilar Lobo
Gilberto Martínez
PRESENTE

Por medio del presente el que suscribe en calidad de autor exclusivo de la Tesis o Proyecto Aplicado "Rastreo de eventos en sistemas embebidos con interfaz al usuario utilizando código Morse y código de Huffman" con número ID T4060 declaro que es original y no contiene citas ni transcripciones de otras obras sin otorgar el debido crédito a los poseedores de los derechos y, en el caso del uso de imágenes, fotografías o documentos que así lo requieran, se cuenta con las debidas autorizaciones de reproducción de quienes poseen los derechos patrimoniales.

En caso de existir alguna impugnación con el contenido o autoría de la tesis, la responsabilidad será exclusivamente mía relevando de toda responsabilidad a la UNIVERSIDAD AUTÓNOMA DE GUADALAJARA de cualquier demanda o reclamación que pudiera llegar a formular alguna persona física o moral que se considere con derecho sobre el texto. Asumo todas las consecuencias legales y económicas que se deriven de esta situación.

Atentamente



Daniel Rubio Martín del Campo

4702476

CARTA DE LIBERACIÓN DE TESIS POR SINODAL

Zapopan, Jalisco a 1 de Noviembre del 2023

Daniel Rubio Martín del Campo
Maestría en Ciencias Computacionales


PRESENTE

Por medio del presente le notifico que he concluido la LECTURA de la tesis titulada Rastreo de eventos en sistemas embebidos con interfaz al usuario utilizando código Morse y código de Huffman y tengo las siguientes observaciones:

- El estudiante ha realizado todas las revisiones solicitadas.
- La tesis cumple con todos los requisitos formales de la institución.
- La tesis cuenta con una buena fundamentación teórica y una revisión completa del estado del arte.
- La tesis presenta un diseño completo y un análisis de resultados completo.
- Las conclusiones y el trabajo futuro son presentados
- Se cita correctamente la bibliografía.

Por otro lado, le informo que ACEPTO la invitación a participar como SINODAL en su Examen Recepcional.

Atentamente



Dra. Lina Maria Aguilar Lobo

Profesor Investigador Titular
Departamento de Computación e Industrial
Unidad Académica de Diseño, Ciencia y Tecnología

c.c.p. Director de Tesis, Director de Programa Académico, Dirección de Investigación y Desarrollo Tecnológico

CARTA DE LIBERACIÓN DE TESIS POR SINODAL

Zapopan, Jalisco a 1 de Noviembre del 2023

Daniel Rubio Martín del Campo
Maestría en Ciencias Computacionales

PRESENTE

Por medio del presente le notifico que he concluido la LECTURA de la tesis titulada Rastreo de eventos en sistemas embebidos con interfaz al usuario utilizando código Morse y código de Huffman y tengo las siguientes observaciones:

- Corrección ortográfica en el apartado.
- Presentación de algoritmos en conjunto con pseudocódigos implementados

Por otro lado, le informo que ACEPTO la invitación a participar como SINODAL en su Examen Recepcional.

Atentamente



Dr. Gerardo Adrián Martínez Fernández

Nombramiento
Departamento de Computación e Industrial

c.c.p. Director de Tesis, Director de Programa Académico, Dirección de Investigación y Desarrollo Tecnológico

CARTA DE LIBERACIÓN DE TESIS POR SINODAL

Zapopan, Jalisco a 10 diciembre del 2023

Daniel Rubio Martín del Campo
Maestría en Ciencias Computacionales

PRESENTE

Por medio del presente le notifico que he concluido la LECTURA de la tesis titulada Rastreo de eventos en sistemas embebidos con interfaz al usuario utilizando código Morse y código de Huffman y tengo las siguientes observaciones:

- El proyecto tiene una gran aportación para la industria automotriz
- El documento ha sido entregado tomando en cuenta las recomendaciones presentadas.

Por otro lado, le informo que ACEPTO la invitación a participar como SINODAL en su Examen Recepcional.

Atentamente



Dr. Ulises Davalos Guzman

Nombramiento
Departamento de Adscripción

c.c.p. Director de Tesis, Director de Programa Académico, Dirección de Investigación y Desarrollo Tecnológico