INFORME DE LABORATORIO

Autores: José Moreno, Valentina Restrepo
Laboratorio de Electrónica Digital III
Departamento de Ingeniería Electrónica y Telecomunicaciones
Universidad de Antioquia

Abstract

This report compares diverse Digital Signal Generator (DSG) implementations on Raspberry Pi Pico. Five programming flows are explored, including Arduino and Micro Python, assessing their ability to generate stable signals. Official libraries are exclusively used. The aim is to identify the most efficient approach for signal generation, offering insights for electronic and communication systems design.

Introducción

Este informe presenta el proceso de desarrollo y comparación de distintas implementaciones de un Generador Digital de Señales (GDS) en un entorno Raspberry Pi Pico. La investigación se centra en cinco flujos de programación y ambientes de desarrollo: Arduino con polling, Micro Python con polling, C con estrategias de polling y solo interrupciones, así como la combinación de polling e interrupciones en C. Cada implementación se evalúa en función de su capacidad para generar de manera estable señales de diversas formas de onda (seno, triangular, diente de sierra y cuadrada), considerando factores como la frecuencia máxima alcanzada y la estabilidad de las señales generadas. Para llevar a cabo este análisis, se establece un protocolo experimental que incluye la restricción del uso de librerías a aquellas oficialmente proporcionadas, evitando el empleo de librerías de terceros para garantizar la objetividad de los resultados.

Marco Teórico

Microcontrolador

La Raspberry Pi Pico W es un microcontrolador de bajo costo basado en el chip RP2040. Cuenta con un procesador ARM Cortex-M0+ con una frecuencia de reloj de hasta 133 MHz y una memoria flash integrada de hasta 16 MB. La Pico W ofrece una amplia gama de capacidades de E/S, incluyendo 30 pines GPIO, que admiten varios protocolos de comunicación como UART, SPI, I2C y PWM. Además, cuenta con un sistema de temporizador de 4 canales y un conversor analógico-digital (ADC) de 3 canales. La programación de la Pico W se puede realizar utilizando el SDK de Raspberry Pi Pico en lenguaje C/C++ o mediante MicroPython. [1]

Polling

En programación, el polling es una técnica utilizada para verificar de manera periódica el estado de un dispositivo o una condición en un sistema. Consiste en realizar ciclos de consulta o "encuestas" a intervalos regulares para verificar si se ha producido algún evento o cambio de estado que requiera atención.

En el contexto de sistemas embebidos y aplicaciones de tiempo real, el polling es comúnmente utilizado para monitorear dispositivos de entrada/salida (E/S), como sensores o botones, y para controlar la ejecución de tareas en función de eventos específicos.

La implementación del polling implica la creación de bucles de control que incluyen instrucciones de consulta o verificación del estado del dispositivo o condición de interés. Estos bucles se ejecutan de manera continua, con una frecuencia determinada por el tiempo de espera entre cada consulta. Es importante tener en cuenta que el polling puede consumir recursos de manera significativa, ya que el procesador está activamente involucrado en la verificación del estado, lo que puede afectar el rendimiento general del sistema.

Una ventaja del polling es su simplicidad y predictibilidad en la programación, ya que el flujo de control del programa está explícitamente definido por el desarrollador. Sin embargo, el polling puede ser menos adecuado para manejar múltiples eventos simultáneos o condiciones de alta frecuencia.

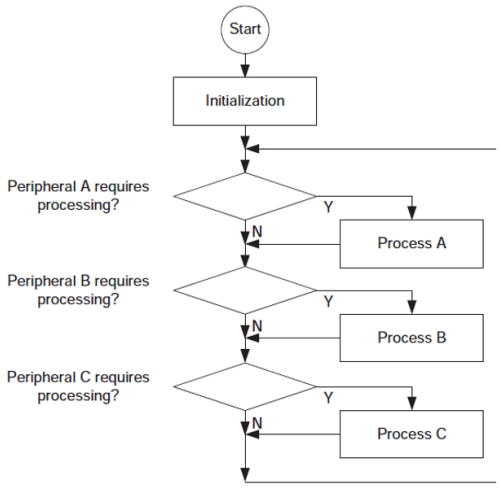

Figura 1. Estructura del Polling

Interrupciones

Las interrupciones son una técnica fundamental en programación que permite a un dispositivo o sistema responder de manera rápida a eventos externos o internos, sin necesidad de esperar activamente a que se produzcan. En lugar de depender de un ciclo de consulta continua como en el polling, las interrupciones permiten que el procesador suspenda temporalmente la ejecución de su tarea actual para manejar un evento prioritario.

Cuando se produce una interrupción, el procesador detiene temporalmente la ejecución del programa principal y pasa a ejecutar un fragmento de código conocido como "manejador de interrupción" o "rutina de servicio de interrupción" (ISR). Esta rutina está diseñada para manejar el evento que generó la interrupción de manera rápida y eficiente.

Las interrupciones pueden ser causadas por una variedad de eventos, como señales de hardware (un botón presionado, una transferencia de datos completa, etc.), temporizadores, errores de hardware, o incluso señales generadas por otros dispositivos. Cada tipo de interrupción está asociado con un número o identificador único, lo que permite al procesador determinar qué rutina de interrupción debe ejecutarse en respuesta a un evento específico.

La implementación de interrupciones implica configurar previamente el hardware y el software para que reconozcan y manejen los eventos de manera adecuada. Esto incluye asignar prioridades a las interrupciones, habilitar y deshabilitar selectivamente ciertas interrupciones según sea necesario, y escribir rutinas de interrupción eficientes y libres de errores.

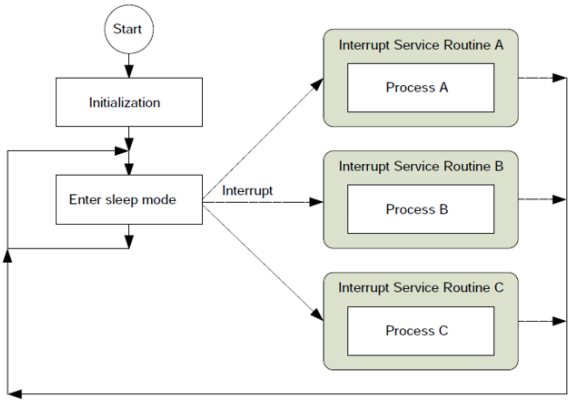

Figura 2. Estructura de las Interrupciones

Polling + Interrupciones

La combinación de polling e interrupciones en programación es una estrategia que busca aprovechar las fortalezas de ambas técnicas para mejorar la eficiencia y la capacidad de respuesta

de los sistemas embebidos y de tiempo real. Mientras que el polling implica la verificación continua del estado de los dispositivos o condiciones, las interrupciones permiten al procesador responder de inmediato a eventos críticos sin demoras. Esta estrategia es especialmente útil cuando ciertos eventos requieren una respuesta rápida y otros pueden ser verificados periódicamente. Por ejemplo, en un sistema de control, las interrupciones pueden utilizarse para manejar eventos como emergencias, mientras que el polling se emplea para supervisar el estado del sistema de manera continua. La implementación efectiva de esta estrategia requiere un equilibrio cuidadoso entre la capacidad de respuesta del sistema y el uso eficiente de los recursos del procesador. Al optimizar el diseño del sistema y el código, se puede lograr un sistema altamente eficiente y receptivo para una amplia gama de aplicaciones.

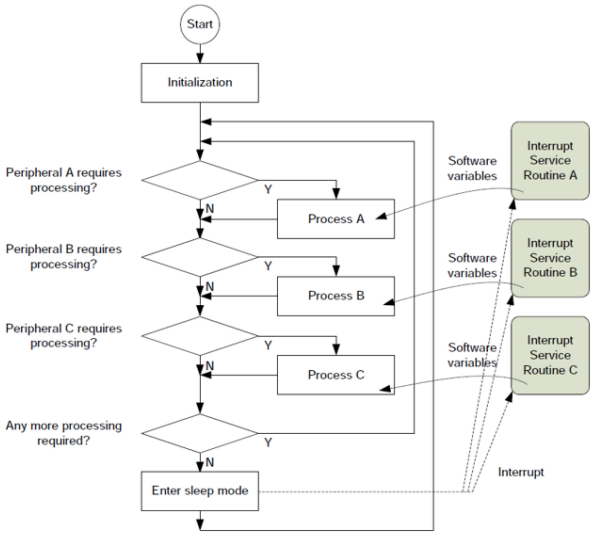

Figura 3. Estructura de Polling e Interrupciones

DAC 0808

El DAC0808 (Digital-to-Analog Converter) es un componente electrónico que convierte señales digitales en señales analógicas de salida. Es utilizado comúnmente en aplicaciones donde se requiere la generación de señales analógicas precisas y controladas a partir de datos digitales.
Consiste en una matriz de resistencias que están conectadas a un conjunto de interruptores controlados digitalmente. Estos interruptores permiten seleccionar diferentes combinaciones de resistencias para generar una salida analógica proporcional al valor digital de entrada.
La resolución del DAC0808 de 8 bits puede convertir valores digitales de 0 a 255 en un rango de voltaje analógico determinado por su configuración de alimentación y referencia.

El amplificador operacional que se implementó fue el LF353, tal como se muestra en la figura 4.

Es uno de los amplificadores operacionales más populares y ampliamente utilizados debido a su alto rendimiento, bajo ruido y amplio rango de frecuencia.

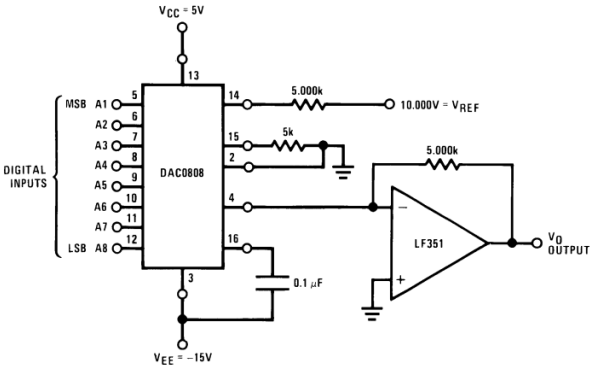

Figura 4. Montaje del DAC0808 y el OpAmp LF353

Procedimiento Experimental y Resultados

Pinout

Para el pinout (tabla 1), se comenzó con la conexión de los ocho pines del teclado matricial a la raspberry, de modo que filas y columnas tuvieran GPIO consecutivos (2 - 9) para poder recorrerlos en un arreglo en el código.

Para los demás GPIO no se tuvo en cuenta alguna condición para su conexión, por lo que los pines para la salida digital de la señal que van al DAC0808 no están consecutivos y el botón se conectó a un GPIO que facilitase su cableado.

COMPONENTE	GPIO
C1	2
C2	3
C3	4
C4	5
L1	6
L2	7
L3	8
L4	9
Botón	14
DAC 5	22
DAC 6	21
DAC 7	20
DAC 8	19
DAC 9	18
DAC 10	17
DAC 11	16
DAC 12	15

Tabla 1. Pinout del proyecto

Montaje

Para el montaje, se siguió la estructura del esquemático que se muestra en la figura 4, el único cambio fue el amplificador operacional, no se usó el LF351 sino el LF353, se conectó y polarizó adecuadamente, tal como se muestra en la figura 5.

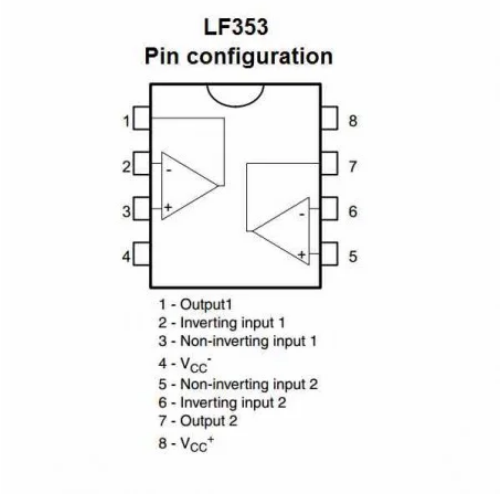

Figura 5. Pinout LF353

Para la señal de 8 bits que el DAC recibe, se dispuso esta en su bit más significativo al pin 5 del DAC, hasta su bit menos significativo al pin 12 del DAC.

Código

Para esta práctica, se propone implementar una codificación bajo diferentes flujos de codificación a través del uso de diferentes lenguajes de programación como Micropython, Arduino y C. A continuación, se muestran las diferentes implementaciones realizadas para el generador:

Arduino con Polling

En cuanto a la implementación realizada para Arduino, se tuvo que el desarrollo del programa bajo el flujo de datos de polling fue muy similar al realizado en Micropython, ya que la lógica secuencias del programa fueron muy similares a las que ya se habían implementado anteriormente. Sin embargo, fue necesario realizar una consulta en las bases de datos de Arduino para poder tomar la decisión acerca de cuáles son los tipos de datos que mejor representan las variables y los valores posibles a implementar.

Problemas encontrados:

Asimismo, se evidenció que la lectura de los datos de entrada a través del serial fue un poco más complejo debido a la configuración inicial establecida, ya que no nos permitía leer las indicaciones de impresión que habíamos agregado en el código tales como las de la lectura del botón, el tipo de señal que se estaba imprimiendo y también las configuraciones ingresadas por el teclado.

Pruebas:

Para el apartado de pruebas de polling con el uso de Arduino, se vio que no se estaban mostrando las señales de la manera que se esperaba, ya que la señal seno tenían una forma no suavizada, la triangular tenía un tiempo muerto que la hacía parecer una señal cuadrada y la señal diente de sierra no parecía conservar su forma. Por esto, se tomó la decisión de alterar la forma en la que se originan las señales originalmente y se optó por tomar un arreglo de valores que describen

correctamente la señal para posteriormente modificar estos valores dependiendo de la frecuencia, amplitud y offset ingresado en el teclado en cada interacción. Con este cambio, fue posible identificar la forma de las ondas de manera periódica en el osciloscopio, siendo la señal cuadrada la que mejor estaba representada en la salida durante el periodo de pruebas.

Micropython con Polling

El Micropython es una implementación específica de Python 3 enfocada en la ejecución de código en microcontroladores y sistemas embebidos. En cuanto a su uso para la Raspberry Pi Pico W, se tiene la posibilidad de desarrollar una codificación muy similar a la tradicionalmente conocida para Python, solo que con la consideración de las limitaciones presentes en la memoria y la potencia de procesamiento.

Problemas encontrados:

Para esta implementación específica, se tuvieron diferentes problemas en los que hubo que realizar diferentes pruebas con el hardware con el fin de identificar las etapas que no permitían tener un flujo adecuado en el código de Polling.

En primer lugar, la implementación del botón para el cambio del tipo de señal presenta varios problemas, tanto por codificación propia del pin como por el tipo de conexión que se tenía con el microcontrolador. Esto, se debía a que dentro del código no se había asignado correctamente el botón como uno tipo PULLDOWN en el pin 6. Además, en las conexiones de hardware se tenía que la entrada del microcontrolador estaba erróneamente conectada en una salida de voltaje, lo que ocasiona un corto cada vez que se presionaba el botón, resultando en que la tarjeta se desconectara del computador.

En segundo lugar, al generar las señales para cada uno de los tipos implementados en este generador, se tuvo problemas para poder organizar de manera paramétrica cada uno de los valores de la señal desde un arreglo básico como el mostrado en el archivo de Python en la carpeta de información general de este repositorio. Sin embargo, al optar por cambiar la generación de una señal nueva en cada iteración dentro del código se resolvió el problema ya que para cada cambio de amplitud, offset o frecuencia se tenía un set de valores actualizado para transmitir en el oscilador.

Pruebas:

Luego de haber solucionado los diferentes problemas presentados durante el desarrollo de este apartado, se pudo realizar un diferente set de pruebas donde se vio que la generación básica de las señales era adecuada para cada uno de los casos de prueba. La primera generación correcta se obtuvo con la señal cuadrada, que se vio correctamente en el osciloscopio donde se analizaron las señales. Además, luego de solucionar algunos problemas con el teclado se pudo solucionar el problema de configuración de parámetros para ver un correcto funcionamiento de la generación de las señales modificadas para diferentes valores de amplitud, frecuencia y offset.

C con Polling

Para el código en C, se implementaron de manera modular las librerías necesarias para el manejo de los periféricos tales como el teclado matricial.

La lógica para realizar las funciones básicas consistió en generar un vector de aproximadamente 20 valores que representara al ser graficado mínimo un periodo de la señal, para luego modificarlo con la amplitud, la frecuencia y el offset requerido, todo a través de operaciones matemáticas e iteraciones.

Para implementar la lógica del polling, se diseñó el código de forma secuencial, es decir, no se dispuso tanto de forma modular. Tras haber declarado las funciones generadoras de señales, se inicializa todo lo necesario para el programa en el main, que se dispone como un bucle infinito al contener

dentro de sí una estructura del tipo while sin condición de salida que todo el tiempo está verificando el estado de los botones y generando las asignaciones necesarias.

Los problemas para este flujo de programación iniciaron al tener errores en la instalación de todas las herramientas necesarias, el entorno de desarrollo necesario (Pico - Visual Studio Code) no se ejecutaba, tras haber resuelto este problema surgieron otros enlazados con la imposibilidad de encontrar por parte del IDE las librerías necesarias, estos problemas fueron resueltos a tiempo.

Las pruebas arrojaron resultados satisfactorios, con frecuencias máximas de 14800 KHz aproximadamente, tal como se muestra en la tabla 2.

C con Interrupciones

El enfoque de programación utilizando interrupciones en C para este proyecto ofrece una alternativa más eficiente y precisa en comparación con el flujo de control basado en el polling. La implementación de las interrupciones permite al microcontrolador reaccionar de manera inmediata a eventos específicos, como la pulsación de un botón o la llegada de un pulso de temporización, sin necesidad de un monitoreo continuo de los periféricos.

En este contexto, se diseñó el código dividiendo las tareas en funciones modulares que se ejecutan en respuesta a eventos de interrupción. Por ejemplo, se configuraron interrupciones para el botón físico y para el teclado matricial, de modo que cuando se detecta una pulsación, se ejecuta una función de manejo de interrupciones que realiza las acciones correspondientes, como el análisis del texto ingresado por el usuario o la generación de la señal apropiada.

Además, se emplearon temporizadores y sus interrupciones asociadas para controlar la generación de la señal periódica. Por ejemplo, se utilizó un temporizador para generar interrupciones a una frecuencia determinada, lo que permite actualizar el valor de la señal de salida en intervalos regulares, garantizando así la continuidad y precisión de la forma de onda generada.

El principal beneficio de este enfoque es la eficiencia en el uso de recursos, ya que el microcontrolador solo realiza tareas cuando sea necesario, en lugar de realizar un monitoreo continuo de los periféricos. Además, al separar las tareas en funciones modulares que se ejecutan en respuesta a eventos de interrupción, se mejora la organización y la legibilidad del código.

Sin embargo, la implementación de interrupciones también puede introducir cierta complejidad adicional en el código, especialmente en lo que respecta a la gestión de prioridades de interrupción y la sincronización de eventos. Es importante tener en cuenta estos aspectos al diseñar y depurar el sistema.

Discusión de resultados

Calificación de dificultad para cada desarrollo
Frecuencias máximas alcanzadas
Diagramas implementados

A pesar de que en la guía se establece una frecuencia máxima de 12 MHz, el limitante de la frecuencia que tendrá el generador de señales se encuentra en el DAC0808 y se establecerá en 5 MHz (ver figura 6).

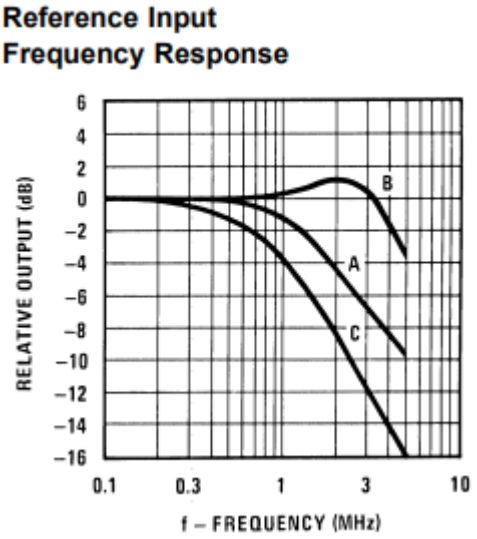

Figura 6. Respuesta en frecuencia del DAC0808

Las frecuencias máximas alcanzadas se establecen en la tabla 2.

Flujo de Programación	Frecuencia (Hz)
Arduino c	2.1K
Micropython	53.2
C Polling	14.8K
C Interrupciones	4.2K

Tabla 2. Frecuencias alcanzadas

Conclusiones

- Una ventaja clave de las interrupciones es su capacidad para reducir la latencia y mejorar la respuesta del sistema a eventos críticos en tiempo real. Al permitir que el procesador atienda rápidamente eventos importantes sin interrumpir su flujo de ejecución normal, las interrupciones son especialmente útiles en aplicaciones donde la velocidad y la precisión son críticas, como el control de dispositivos en tiempo real, la captura de datos en alta velocidad, o la comunicación con periféricos. Además, el uso indiscriminado de interrupciones puede aumentar la complejidad del código y consumir recursos del sistema, por lo que es importante equilibrar su uso con otras técnicas de programación según las necesidades específicas de la aplicación.

- Respecto a la dificultad de cada flujo de programación, los más complejos fueron los algoritmos en el lenguaje C, pues la capa de abstracción es mayor, también estos fueron los que dieron mejor rendimiento respecto a frecuencias. Los menos complejos fueron Arduino y Micropython, pero también los que presentaron menor rendimiento. Aunque Arduino sí presentó mayor rendimiento respecto a Micropython.

Referencias

[1] Monk, S. (2022). *Raspberry Pi Cookbook*. " O'Reilly Media, Inc.".

[2] Jung, C. D., & Siberrt, E. (1989, February). Polling in concurrent programming. In *Proceedings of the 17th conference on ACM Annual Computer Science Conference* (pp. 204-209).

[3] Zelkowitz, M. (1971). Interrupt driven programming. *Communications of the ACM*, *14*(6), 417-418.

[4] Texas Instruments. (2022). "DAC0808: 8-Bit Digital-to-Analog Converters." [Online]. Available: https://www.ti.com/lit/ds/symlink/dac0808.pdf.

[5] Texas Instruments. (2022). LF353: Wide Bandwidth Dual JFET Input Operational Amplifier. [Datasheet]. Recuperado de https://www.ti.com/lit/ds/symlink/lf353-n.pdf