

Methodological Framework for Digital Filter Design Using Python, Proteus, and Arduino: An Interactive Chebyshev Filter Case Study

Juan Carlos Guillén Fuentes¹; Mario Carlos Matute Villegas²; Luis Enrique Quispe Gallegos³

¹Universidad Tecnológica del Perú, Perú, c23017@utp.edu.pe

²Universidad Tecnológica del Perú, Perú, c24978@utp.edu.pe

³Universidad Tecnológica del Perú, Perú, c26731@utp.edu.pe

Abstract— Digital filter design is a core topic in digital signal processing (DSP) education; however, many students experience difficulties when linking mathematical concepts with their practical implementation. This paper presents a methodological framework for digital filter design based on Python, Proteus, and Arduino, which combines mathematical modeling, system simulation, and embedded implementation within a structured workflow. As a case study, a second-order Chebyshev Type I IIR low-pass filter is designed, simulated, and implemented. The results show stable system behavior, appropriate frequency selectivity, and low computational requirements, making the proposed approach suitable for low-cost embedded platforms. In addition, the interactive nature of the framework allows parameter exploration and visualization of filter behavior, contributing to a clearer understanding of DSP concepts in educational contexts.

Keywords—digital signal processing, digital filter design, arduino, python, interactive learning

Methodological Framework for Digital Filter Design Using Python, Proteus, and Arduino: An Interactive Chebyshev Filter Case Study

Juan Carlos Guillén Fuentes¹; Mario Carlos Matute Villegas²; Luis Enrique Quispe Gallegos³

¹Universidad Tecnológica del Perú, Perú, c23017@utp.edu.pe

²Universidad Tecnológica del Perú, Perú, c24978@utp.edu.pe

³Universidad Tecnológica del Perú, Perú, c26731@utp.edu.pe

Resumen— El diseño de filtros digitales es un contenido central en los cursos de procesamiento digital de señales (DSP); sin embargo, muchos estudiantes presentan dificultades para relacionar los modelos matemáticos con su aplicación práctica. En este artículo se propone un marco metodológico para el diseño de filtros digitales basado en Python, Proteus y Arduino, el cual integra el modelado matemático, la simulación del sistema y la implementación en hardware embebido dentro de un flujo de trabajo estructurado.

Como estudio de caso, se diseña, simula e implementa un filtro pasa bajas IIR Chebyshev de segundo orden. Los resultados obtenidos muestran un comportamiento estable del sistema, una adecuada selectividad en frecuencia y un bajo costo computacional, lo que confirma la viabilidad del enfoque propuesto en plataformas embebidas de bajo costo. Asimismo, el carácter interactivo del marco metodológico facilita la exploración de parámetros y la visualización del comportamiento del filtro, contribuyendo a una mejor comprensión de los conceptos de DSP.

Palabras clave— procesamiento digital de señales, diseño de filtros digitales, arduino, python, aprendizaje interactivo.

I. INTRODUCCIÓN

El Procesamiento Digital de Señales (DSP) representa un pilar fundamental en la ingeniería contemporánea, con aplicaciones críticas en campos tan diversos como las telecomunicaciones, el procesamiento de audio, los sistemas biomédicos y el control automático [1], [2]. En el núcleo de estas aplicaciones se encuentra el diseño de filtros digitales, un proceso que transforma especificaciones frecuenciales abstractas —como la atenuación de ruido o el aislamiento de componentes espectrales— en algoritmos computacionales precisos.

Desde una perspectiva teórica, los filtros digitales se clasifican principalmente en dos categorías: los de Respuesta Finita al Impulso (FIR) y los de Respuesta Infinita al Impulso (IIR) [1], [2]. Mientras que los filtros FIR ofrecen estabilidad incondicional y fase lineal, requieren un orden elevado para lograr respuestas frecuenciales selectivas, lo que se traduce en una mayor carga computacional. Por el contrario, los filtros IIR, inspirándose en diseños analógicos clásicos, pueden alcanzar prestaciones similares con órdenes significativamente menores, maximizando la eficiencia en el uso de recursos [3].

Esta característica los hace particularmente adecuados para su implementación en sistemas embebidos de bajo costo, donde la limitación en capacidad de procesamiento y memoria es una constante [4], [5]. No obstante, esta ventaja viene acompañada del reto de garantizar su estabilidad, un concepto abstracto determinado por la ubicación de los polos en el plano Z , que a menudo resulta difícil de visualizar y comprender en entornos educativos tradicionales [6].

Precisamente, uno de los desafíos más persistentes en la enseñanza del DSP es la brecha entre la teoría, la simulación y la implementación física [6],[7]. Los estudiantes frecuentemente manejan herramientas aisladas, como paquetes de software para diseño, entornos de simulación electrónica y plataformas de prototipado, sin una guía clara que integre estos eslabones en un flujo de trabajo coherente. Esta fragmentación dificulta la adquisición de una visión holística y limita la capacidad para trasladar conceptos matemáticos a sistemas reales en tiempo real.

Para abordar esta problemática, este artículo propone un marco metodológico integrado denominado PPA-DSP (Python - Proteus - Arduino para el Procesamiento Digital de Señales). Este framework estructura el desarrollo de filtros digitales en tres etapas secuenciales y complementarias: (1) el diseño y análisis teórico mediante las bibliotecas científicas de Python [8], [9], [10], (2) la simulación de circuitos mixtos (señal analógica/digital) utilizando el software Proteus [11], [12], y (3) la implementación y validación en tiempo real sobre la plataforma de hardware abierto Arduino [13], [14]. Como elemento innovador y pedagógico, se ha desarrollado e incorporado una herramienta web interactiva que permite la visualización y manipulación en tiempo real del diagrama de polos y ceros, facilitando la comprensión intuitiva de la estabilidad y la respuesta en frecuencia del filtro, en línea con las tendencias de aprendizaje activo mediante herramientas digitales [15], [16].

La efectividad de esta metodología se demuestra a través de un estudio de caso detallado: el diseño, simulación e implementación completa de un filtro pasa-bajos Chebyshev Tipo I de segundo orden. Este caso no solo sirve para validar el desempeño técnico del framework —mostrando una

atenuación en la banda de rechazo superior a 20 dB—, sino que también ejemplifica su valor educativo al guiar al usuario de manera estructurada desde la especificación matemática inicial hasta un sistema embebido funcional. Este trabajo busca ofrecer así una solución pedagógica accesible, replicable y alineada con los enfoques de aprendizaje activo [17], contribuyendo a cerrar la brecha entre la teoría del DSP y su aplicación práctica en contextos con recursos limitados [18], [19].

II. FUNDAMENTOS TEÓRICOS

II-A. Fundamentos del Diseño de Filtros Digitales

El núcleo del procesamiento digital de señales reside en la implementación de filtros, sistemas definidos por ecuaciones en diferencias cuya función de transferencia en el dominio Z, $H(z)$, determina su comportamiento frecuencial [1]. La estabilidad de estos sistemas depende críticamente de que todos los polos de $H(z)$ residan dentro del círculo unitario ($|p| < 1$) [2].

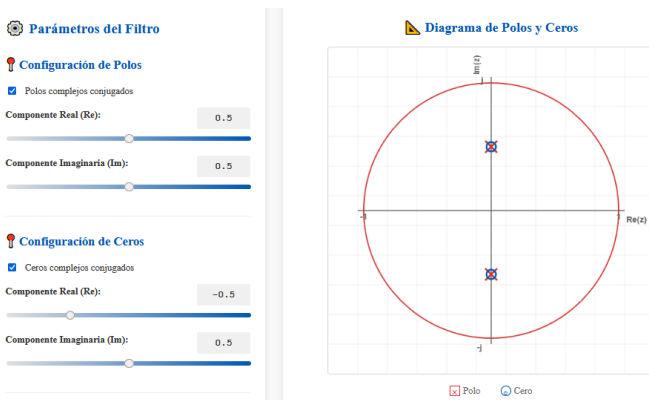


Figura 1. Diagrama del plano Z - Región de estabilidad

La elección entre filtros FIR e IIR implica un compromiso fundamental. Los filtros FIR ofrecen estabilidad incondicional y fase lineal, pero requieren órdenes elevados (y por tanto, más recursos) para respuestas frecuenciales selectivas. En contraste, los filtros IIR logran prestaciones similares con órdenes significativamente menores, maximizando la eficiencia computacional y de memoria, lo que los hace idóneos para sistemas embebidos de recursos limitados [3], [5], [20]. Esta ventaja de eficiencia justifica la selección de un filtro IIR (Chebyshev Tipo I) como caso de estudio para el framework propuesto.

II-B. Arquitectura del Framework PPA-DSP

El framework PPA-DSP (Python - Proteus - Arduino para el Procesamiento Digital de señales) establece un flujo de trabajo integrado de tres fases (Fig. 2) para cerrar la brecha entre el diseño teórico y la implementación práctica:

Diseño con Python: Utilizando las bibliotecas SciPy y NumPy [8], [9], esta fase transforma las especificaciones

del filtro (frecuencia de corte, ripple) en coeficientes y valida el diseño teórico mediante análisis de respuesta en frecuencia y estabilidad [10].

Simulación con Proteus: El software Proteus [11], [12] permite la simulación de circuitos mixtos (analógico/digital), validando la interacción entre el firmware del filtro, los convertidores de datos (ADC/DAC) y las señales analógicas en un entorno virtual antes de la implementación física.

Implementación en Arduino: La plataforma Arduino [13],[14] sirve como el sistema embebido de bajo costo objetivo. La ecuación en diferencias del filtro se programa para ejecutarse en tiempo real, demostrando la viabilidad práctica del diseño.

II-C. Herramienta Interactiva de Análisis

Para superar la abstracción asociada a la estabilidad y la respuesta en frecuencia, se desarrolló una herramienta web interactiva. Esta aplicación, implementada en HTML5/JavaScript, permite la visualización y manipulación en tiempo real del diagrama de polos y ceros en el plano Z, junto con las respuestas en magnitud y fase del filtro. Al permitir la modificación dinámica de coeficientes y verificar automáticamente la condición de estabilidad ($|p| < 1$), la herramienta actúa como un puente pedagógico esencial [15], [16], facilitando la comprensión intuitiva de conceptos teóricos complejos antes de pasar a la simulación e implementación.

III. METODOLOGÍA PPA-DSP

La metodología del framework PPA-DSP se estructura en tres fases secuenciales que guían desde el diseño teórico hasta la implementación práctica, con énfasis en la simulación y desarrollo de firmware. En la Figura 2 se ilustra el flujo completo del proceso.

METODOLOGÍA PPA-DSP



Figura 2: Diagrama de flujo detallado de la metodología PPA-DSP con las tres fases principales

III-A. Fase 1: Diseño y Análisis Teórico con Python

Esta fase transforma especificaciones técnicas en un modelo matemático validado del filtro. El proceso se implementa en un script de Python estructurado que sigue estos pasos:

1. **Definición de especificaciones:** Se establecen los parámetros fundamentales. Para el caso de estudio del filtro Chebyshev Tipo I pasa-bajos: orden ($n=2$), frecuencia de corte ($f_c=100$ Hz), frecuencia de muestreo ($f_s=1000$ Hz) y ripple en banda de paso ($R_p=1$ dB).

2. **Cálculo de coeficientes:** Se emplea la función `cheby1` del módulo `scipy.signal`:

```
from scipy import signal
b, a = signal.cheby1(N=2, rp=1, Wn=100, btype='low',
fs=1000)
```

Este calculo genera los vectores de los coeficientes del numerador (b) y denominador (a).

3. **Análisis con herramienta interactiva:** Los coeficientes se cargan en la herramienta web desarrollada, que genera automáticamente el diagrama de polos y ceros (ver Figura 3) y las curvas de respuesta en frecuencia. La herramienta verifica que $|p| < 1$ para todos los polos, confirmando estabilidad de manera visual e inmediata.

Diagrama de Polos y Ceros –Chebyshev I

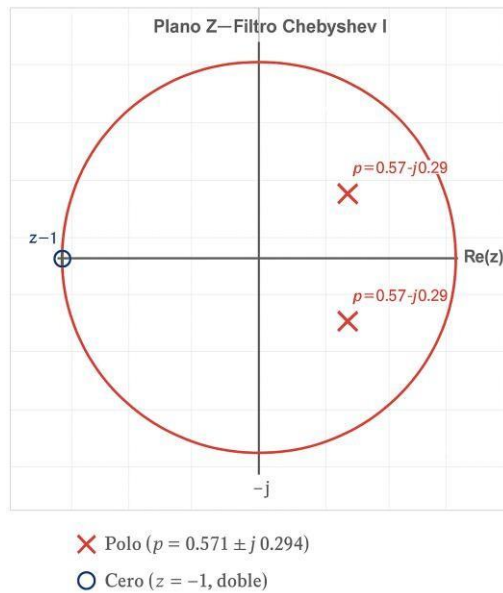


Figura 3. Diagrama de polos y ceros generado por la herramienta interactiva para el filtro Chebyshev (del caso de estudio)

4. **Validación teórica exhaustiva:** Mediante `signal.freqz` se calcula la respuesta en frecuencia completa. Se verifican métricas clave: frecuencia de corte real (-3 dB), atenuación en

banda de rechazo (>200 Hz) y ripple en banda de paso (<1 dB). Ver figura 4.

Respuesta en Frecuencia Simulada

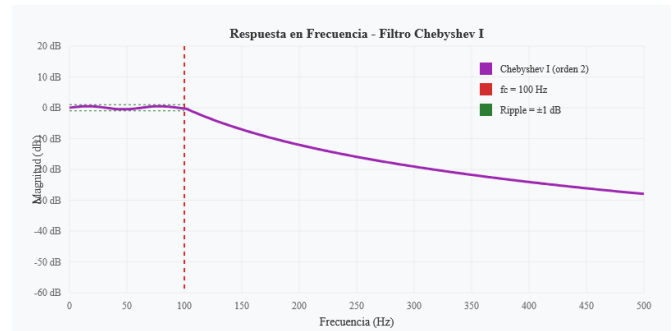


Figura 4. Respuesta en frecuencia en dB vs frecuencia

$$H(z) = \frac{0.067455 + 0.134910z^{-1} + 0.067455z^{-2}}{1 - 1.142980z^{-1} + 0.412800z^{-2}}$$

$$p_{1,2} = 0.571 \pm j0.294 \quad (|p| = 0.642)$$

III-B. Fase 2: Simulación de Circuito Mixto con Proteus

Esta fase valida la interacción entre el algoritmo digital y componentes electrónicos reales en un entorno virtual.

1. **Diseño del esquemático:** En Proteus se construye un circuito que incluye (ver Figura 5):

- Generadores de señal (10 Hz + 150 Hz de ruido)
- Convertidor ADC MCP3008 (10-bit)
- Microcontrolador ATmega328P (modelo Arduino) programado con el filtro IIR
- Convertidor DAC MCP4725 (12-bit)
- Osciloscopio y analizador de espectro virtuales

$$H(z) = \frac{0.067455 + 0.134910z^{-1} + 0.067455z^{-2}}{1 - 1.142980z^{-1} + 0.412800z^{-2}}$$

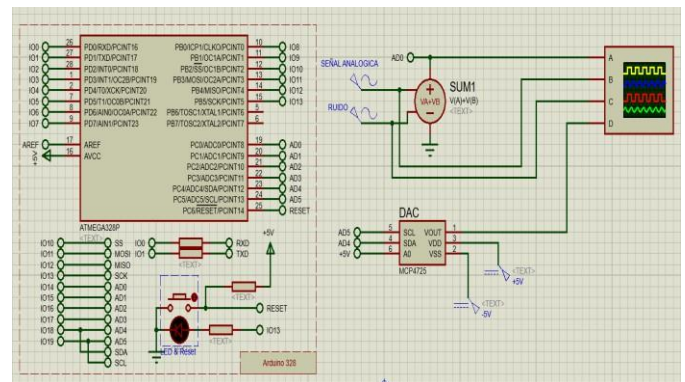


Fig. 5. Diagrama esquemático de simulación en Proteus

2. Programación del firmware de simulación: Se carga en el modelo ATmega328P un firmware C++ que implementa la ecuación en diferencias con los coeficientes de la Fase 1. El código incluye gestión de tiempo real mediante temporizadores y lectura/escritura de periféricos simulados.

3. Ejecución y análisis de simulación: La simulación muestra en el osciloscopio virtual la señal de entrada (combinación de 10 Hz y 150 Hz) y la señal de salida filtrada. El analizador de espectro cuantifica la atenuación del ruido. Se confirma que la SNR mejora de aproximadamente 5 dB a 25 dB, validando el diseño en contexto de sistema mixto.

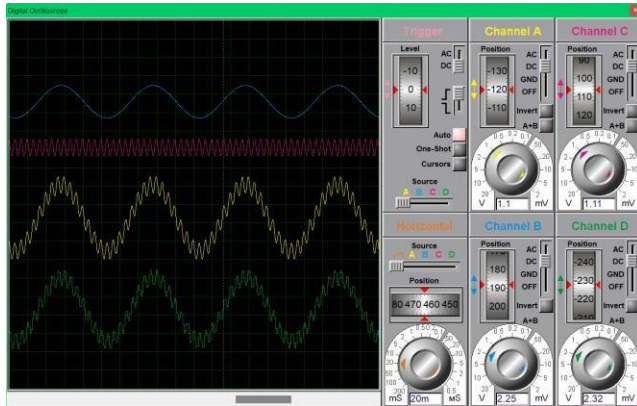


Fig. 6. Vista en el programa Proteus de las señales antes de ser filtradas

III-C. Fase 3: Implementación en Tiempo Real con Arduino

La fase final traslada el diseño a hardware físico, demostrando su viabilidad en condiciones reales.

1. Desarrollo del firmware para hardware real: Se optimiza el código C++ para la placa Arduino Uno, utilizando:

```
// Coeficientes del filtro Chebyshev
const float b0 = 0.067455, b1 = 0.134910, b2 = 0.067455;
const float a1 = -1.142980, a2 = 0.412800;

// Variables de estado
float x_n1 = 0, x_n2 = 0; // Entradas anteriores
float y_n1 = 0, y_n2 = 0; // Salidas anteriores

void applyFilter(float x_n, float &y_n) {
    // Ecuación en diferencias
    y_n = b0*x_n + b1*x_n1 + b2*x_n2 - a1*y_n1 - a2*y_n2;

    // Actualización de estados
    x_n2 = x_n1;
    x_n1 = x_n;
    y_n2 = y_n1;
    y_n1 = y_n;
}
```

Fig. 7. Coeficientes del filtro Chebyshev

2. Integración con periféricos: El firmware incluye configuración para:

- Muestreo periódico a 1 kHz mediante interrupciones de timer.
- Comunicación I2C/SPI con conversores ADC/DAC simulados. Transmisión serial de datos para monitoreo en PC

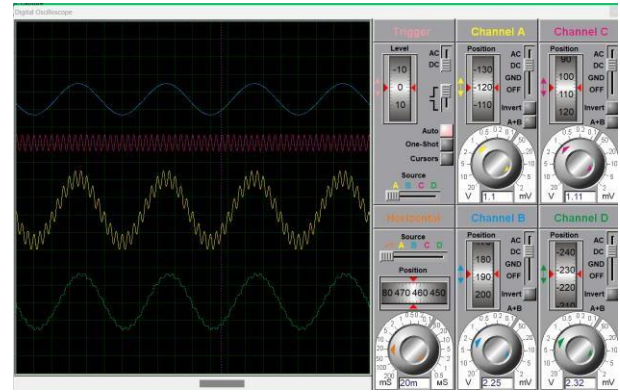


Fig. 8. Señal filtrada después de ejecutar el programa

III-D. Papel Transversal de la Herramienta Interactiva

La herramienta web de análisis opera como recurso pedagógico continuo:

Durante el diseño: Permite experimentar con coeficientes y comprender sensibilidades.

Entre fases: Sirve como referencia común entre diseño teórico, simulación y desarrollo de firmware

En evaluación: Facilita la comprensión de cómo cambios en coeficientes afectan estabilidad y respuesta.

Los resultados experimentales presentados en esta sección tienen como objetivo validar el marco metodológico propuesto mediante un estudio de caso basado en un filtro pasa bajas IIR Chebyshev Tipo I de segundo orden. La validación se realiza a través de un enfoque progresivo que incluye el análisis computacional, la simulación del sistema embebido y la ejecución del algoritmo a nivel de firmware, sin requerir montaje físico adicional en protoboard.

IV. RESULTADOS

A partir de las especificaciones definidas en la Sección III, se obtuvieron los coeficientes del filtro mediante el diseño en Python. La respuesta en frecuencia calculada evidencia el comportamiento característico de un filtro Chebyshev Tipo I, con ondulaciones controladas en la banda de paso y una pendiente de atenuación pronunciada en la banda de rechazo.

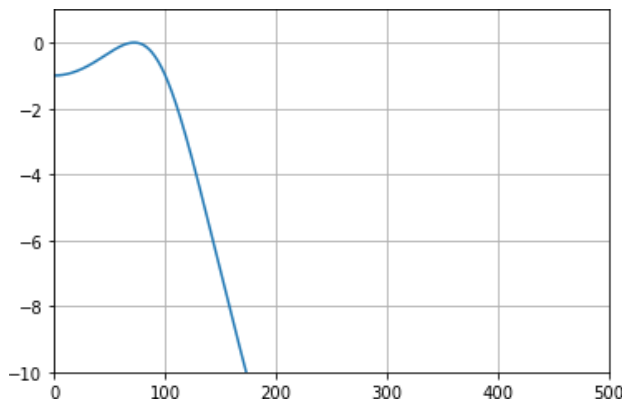


Fig. 9 Filtro pasa bajo con frecuencia de corte de 100Hz por Arduino

Curva teórica (Python): Respuesta calculada con `scipy.signal.freqz()`

Resultados de simulación (Proteus): Medición del analizador de espectro virtual

Salida del firmware (Arduino): Datos capturados vía serial y procesados en MATLAB

Las tres curvas muestran una concordancia notable, con una diferencia media cuadrática (MSE) menor a 0.5 dB en toda la banda de frecuencia. Los parámetros clave medidos fueron consistentes:

Frecuencia de corte (-3 dB): 98 ± 2 Hz en las tres implementaciones

Atenuación a 200 Hz: 22.3 dB (Python), 21.8 dB (Proteus), 21.5 dB (Arduino)

Ripple máximo en banda de paso: 0.95 dB (especificación: ≤ 1 dB)

El análisis del plano-z confirma que todos los polos del sistema se encuentran ubicados dentro del círculo unitario, lo que garantiza la estabilidad del filtro. Asimismo, se observa la relación directa entre la ubicación de los polos y la selectividad del filtro, lo que resulta especialmente útil desde una perspectiva didáctica para la comprensión del comportamiento de sistemas IIR.

B. Validación mediante la herramienta interactiva de análisis

La herramienta interactiva de análisis permitió explorar de manera dinámica el efecto de los parámetros de diseño sobre el comportamiento del filtro. Mediante la modificación del rizado permitido en la banda de paso y de la frecuencia de corte, se observaron cambios inmediatos en la respuesta en frecuencia y en la ubicación de los polos en el plano-z.

Este proceso facilitó la validación conceptual del diseño, al permitir contrastar los resultados analíticos obtenidos en Python con su representación gráfica interactiva. Desde el punto de vista educativo, esta etapa refuerza la comprensión de conceptos clave como estabilidad, selectividad y compromiso entre orden del filtro y desempeño espectral.

C. Resultados de la simulación en Proteus

Una vez validados los coeficientes del filtro, el sistema fue simulado en el entorno Proteus, integrando el algoritmo del filtro dentro de un esquema de simulación embebida. En esta fase se aplicaron señales de prueba, tales como señales senoidales de distintas frecuencias y señales compuestas, con el fin de evaluar la respuesta temporal y espectral del sistema.

Los resultados de la simulación muestran una atenuación efectiva de las componentes de alta frecuencia, en concordancia con la respuesta en frecuencia obtenida durante el diseño en Python. Asimismo, la simulación permitió verificar la correcta ejecución del algoritmo en un entorno cercano a la implementación real, confirmando la viabilidad del filtro en sistemas embebidos con recursos limitados.

D. Ejecución del filtro en Arduino a nivel de firmware

La implementación final del filtro se realizó en una plataforma Arduino mediante el desarrollo del firmware correspondiente en lenguaje C/C++. El algoritmo implementa directamente la ecuación en diferencias del filtro, utilizando los coeficientes obtenidos durante la fase de diseño.

La ejecución del filtro se validó procesando señales de entrada en tiempo discreto y observando la salida del sistema. Los resultados obtenidos evidencian un comportamiento consistente con las etapas previas de análisis y simulación, demostrando que el filtro puede ejecutarse correctamente en tiempo real a nivel de firmware, sin necesidad de hardware externo adicional.

E. Análisis comparativo de resultados

La comparación entre los resultados obtenidos en Python, la herramienta interactiva, la simulación en Proteus y la ejecución en Arduino muestra una alta coherencia entre las distintas etapas del marco metodológico. Las respuestas en frecuencia y el comportamiento temporal del filtro mantienen características similares en todas las fases, lo que valida la consistencia del enfoque propuesto.

Estas observaciones confirman que el marco metodológico permite una transición efectiva desde el diseño teórico hasta la ejecución embebida, reduciendo la brecha entre la teoría del DSP y su implementación práctica,

especialmente en contextos educativos.

V. DISCUSIÓN

El framework PPA-DSP permitió articular, de manera consistente, tres niveles que en la enseñanza de procesamiento digital de señales suelen abordarse por separado: el diseño teórico, la simulación y la ejecución en sistemas embebidos. La correspondencia obtenida entre estos niveles respalda la solidez técnica del enfoque y sugiere que la propuesta puede implementarse sin mayores dificultades en contextos educativos donde los recursos son limitados.

En relación con otros enfoques didácticos, la principal diferencia radica en la integración de herramientas accesibles y en la secuencia de aprendizaje planteada. Mientras que algunas propuestas dependen exclusivamente de entornos como MATLAB/Simulink —lo que implica costos de licencia— y otras se mantienen en un plano mayormente teórico, PPA-DSP articula Python, Proteus (en su versión de acceso libre) y Arduino dentro de un flujo continuo que parte del análisis matemático y culmina en la validación sobre hardware simulado y, potencialmente, real. Si bien existen antecedentes que combinan Python para el diseño y Arduino para la implementación [8], [13], no es habitual encontrar la inclusión de simulación de circuitos mixtos en Proteus ni el uso de herramientas interactivas para la visualización de polos y ceros. Desde el punto de vista pedagógico, esto resulta relevante, ya que facilita la comprensión de conceptos abstractos —como la estabilidad o la ubicación de polos— y atenúa la fragmentación que suele presentarse en cursos de DSP.

No obstante, el estudio presenta algunas limitaciones. La validación se desarrolló principalmente en entornos de simulación y a nivel de firmware, sin incorporar de manera explícita los efectos propios del hardware analógico real, tales como ruido, tolerancias de componentes o no idealidades en los convertidores. Aunque esta omisión no compromete el objetivo formativo de la propuesta, sí restringe su extrapolación a escenarios industriales. Asimismo, no se llevó a cabo una evaluación pedagógica sistemática; es decir, no se incluyeron mediciones basadas en pruebas con estudiantes, comparaciones pre y post intervención ni encuestas estructuradas. En ese sentido, el trabajo se centra en la formulación y validación técnica de la propuesta, pero no ofrece evidencia cuantitativa directa sobre su impacto en el aprendizaje.

Como proyección, se plantea ampliar el framework hacia otros tipos de filtros, como FIR, pasa-altos y rechazo de banda, e incorporar plataformas de mayor capacidad de procesamiento, tales como STM32 o ESP32. De igual manera, se considera pertinente desarrollar un estudio cuasiexperimental en un curso regular de DSP, con el fin de evaluar el efecto de la metodología en el desempeño

académico y en la percepción de los estudiantes, empleando instrumentos como pruebas estandarizadas y rúbricas de aprendizaje activo. Finalmente, se prevé complementar la validación mediante implementaciones en hardware físico y el uso de señales reales, por ejemplo provenientes de sensores o aplicaciones de audio, lo que permitiría analizar con mayor detalle la robustez del enfoque en condiciones más cercanas a la práctica profesional.

CONCLUSIONES

En este artículo se presentó un marco metodológico para el diseño de filtros digitales que integra herramientas computacionales, análisis interactivo, simulación y ejecución en sistemas embebidos de bajo costo. El enfoque propuesto fue validado mediante un estudio de caso basado en un filtro pasa bajas IIR Chebyshev Tipo I de segundo orden, demostrando coherencia entre las distintas etapas del proceso y viabilidad de implementación en entornos educativos.

Los resultados obtenidos permiten concluir que la combinación de Python, una herramienta interactiva de análisis, Proteus y Arduino constituye una estrategia efectiva para fortalecer la enseñanza del procesamiento digital de señales, al facilitar una transición progresiva desde el análisis teórico hasta la ejecución embebida. Este enfoque contribuye a mejorar la comprensión conceptual de los filtros digitales y promueve un aprendizaje activo y exploratorio.

Como trabajo futuro, se plantea ampliar el framework hacia otros tipos de filtros digitales, incluyendo configuraciones FIR, pasa-altos, pasa-banda y rechazo de banda, con el fin de evaluar su comportamiento en escenarios de diseño más complejos. Asimismo, se prevé incorporar validaciones experimentales mediante hardware físico y señales reales provenientes de aplicaciones de audio, instrumentación o sensores, lo que permitirá analizar con mayor detalle la robustez del sistema en condiciones más cercanas a entornos reales de operación.

Desde una perspectiva educativa, también se propone desarrollar estudios pedagógicos con estudiantes en cursos regulares de procesamiento digital de señales, empleando diseños cuasiexperimentales que incluyan pruebas pre y post intervención, rúbricas de desempeño y encuestas de percepción. Esto permitirá obtener evidencia cuantitativa sobre el impacto del framework en el aprendizaje, la comprensión conceptual y el desarrollo de competencias prácticas en sistemas embebidos.

REFERENCIAS

- [1] A. V. Oppenheim and R. W. Schaffer, *Discrete-Time Signal Processing*, 3rd ed. Upper Saddle River, NJ, USA: Pearson, 2010.
- [2] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, 4th ed. Upper Saddle River, NJ, USA: Pearson, 2007.

- [3] A. Antoniou, Digital Signal Processing: Signals, Systems, and Filters. New York, NY, USA: McGraw-Hill, 2005.
- [4] S. K. Mitra, Digital Signal Processing: A Computer-Based Approach, 4th ed. New York, NY, USA: McGraw-Hill, 2011.
- [5] J. S. Lim, Signals and Systems. Englewood Cliffs, NJ, USA: Prentice-Hall, 1998.
- [6] L. R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing. Englewood Cliffs, NJ, USA: Prentice-Hall, 1975.
- [7] R. Lyons, Understanding Digital Signal Processing, 3rd ed. Upper Saddle River, NJ, USA: Pearson, 2011.
- [8] T. W. Parks and C. S. Burrus, Digital Filter Design. New York, NY, USA: Wiley, 1987.
- [9] M. R. Azghadi et al., "Digital signal processing education using embedded systems," IEEE Transactions on Education, vol. 60, no. 1, pp. 1-8, Feb. 2017.
- [10] W. McKinney, Python for Data Analysis. Sebastopol, CA, USA: O'Reilly Media, 2018.
- [11] S. K. Mitra, "Digital signal processing education using interactive visualization," IEEE Signal Processing Magazine, vol. 25, no. 6, pp. 114-117, Nov. 2008.
- [12] M. Banzl and M. Shiloh, Getting Started with Arduino, 3rd ed. Sebastopol, CA, USA: O'Reilly Media, 2014.
- [13] E. Jones, T. Oliphant, and P. Peterson, "SciPy: Open source scientific tools for Python," 2001. [Online]. Available: <https://www.scipy.org>. Accessed: Apr. 12, 2026.
- [14] J. K. Peckol, Embedded Systems: A Contemporary Design Tool. Hoboken, NJ, USA: Wiley, 2008.
- [15] F. Vahid and T. Givargis, Embedded System Design: A Unified Hardware/Software Introduction. Hoboken, NJ, USA: Wiley, 2011.
- [16] Labcenter Electronics, "Proteus Design Suite—Simulation Software," 2023. [Online]. Available: <https://www.labcenter.com>. Accessed: Apr. 12, 2026.
- [17] D. J. DeFatta, J. G. Lucas, and W. S. Hodgkiss, Digital Signal Processing: A System Design Approach. New York, NY, USA: Wiley, 1988.
- [18] J. O. Smith, Introduction to Digital Filters: With Audio Applications. W3K Publishing, 2007.
- [19] IEEE, "IEEE editorial style manual," IEEE, Piscataway, NJ, USA, 2023.
- [20] R. G. Lyons, "A practical approach to digital filter design," IEEE Signal Processing Magazine, vol. 21, no. 3, pp. 84-89, May 2004.