multiplexación, permitiendo la integración de voz, datos y video en una misma infraestructura.

Ramirez Velásquez, Tomas Alejandro.
@tomasramirez20 (GitHub)

Practica de exploración de conceptos de conversion A/D con Raspberry Pi Pico

UMNG

Est.tomas.ramirez@unimilitar.edu.co

I. INTRODUCCIÓN

En esta práctica de laboratorio se estudian los conceptos de conversión A/D y su aplicación en el contexto de la arquitectura de un sistema de comunicación digital, como se pueden modificar diferentes parametros como el numero de bits, rangos de muestreos o de cuantificacion, tiempos de muestreo, etc.

II. Fundamentos Teóricos y Metodología

A. Conversor A/D

Es un dispositivo electrónico fundamental cuya función es transformar una señal analógica del mundo real (continua en el tiempo y en amplitud), como una voz o una temperatura, en una secuencia de valores digitales (discretos en el tiempo y cuantificados en amplitud), es decir, en un lenguaje binario que los sistemas digitales puedan procesar. Esta conversión se realiza mediante dos procesos principales: el *muestreo*, que toma valores instantáneos de la señal a intervalos regulares de tiempo (definidos por la frecuencia de muestreo), y la *cuantificación*, que asigna a cada muestra un valor de un conjunto finito de niveles predefinidos. La precisión del conversor depende críticamente de su resolución (número de bits) y de su velocidad de muestreo.

En la arquitectura de un sistema de comunicación digital, el conversor A/D es el componente esencial del **transceptor** en el extremo transmisor. Su aplicación es el primer paso para digitalizar cualquier información que provenga de una fuente analógica (por ejemplo, un micrófono o una cámara de video). Una vez convertida la señal a una secuencia de bits, el sistema puede procesarla digitalmente (comprimirla, cifrarla, aplicar corrección de errores) y modularla para su transmisión eficiente y robusta a través del canal. Así, el ADC sienta las bases para todas las ventajas de las comunicaciones digitales: inmunidad al ruido, capacidad de regeneración de la señal, seguridad y

B. Relación entre read_u16() y el valor de 12 bits del ADC

En el contexto de la arquitectura de un sistema de comunicación digital, el conversor analógico-digital (ADC) es el componente fundamental que transforma una señal analógica continua (como una voz captada por un micrófono) en una secuencia de valores discretos que una unidad de procesamiento digital puede manipular. La resolución del ADC, definida por su número de bits, determina la precisión de esta representación digital. El ADC del RP2040, con su resolución de 12 bits, puede generar `4095` valores distintos (`2^12 - 1`), lo que significa que divide el voltaje de referencia en `4095` intervalos o niveles de cuantización. Este proceso de digitalización es el primer paso esencial para que cualquier información del mundo real pueda ser procesada, transmitida o almacenada por un sistema digital. [1]

La función `read_u16()` de MicroPython responde a una necesidad de estandarización y compatibilidad en la arquitectura del software. Aunque el hardware subyacente solo proporciona un valor de 12 bits, muchas librerías y sistemas esperan trabajar con valores en potencias de 2 más comunes, como 16 bits. Para lograr esto sin perder la información original, el valor de 12 bits (`code12`) se inserta en los 16 bits de la siguiente manera: se desplaza hacia la izquierda 4 posiciones (`code12 << 4`), lo que es matemáticamente equivalente a multiplicarlo por `16`. Esta operación coloca los 12 bits significativos en la parte más alta del word de 16 bits, rellenando los 4 bits menos significativos (LSB) con ceros. Así, el rango de salida de `0` a `4095` se escala linealmente a un rango de `0` a `65520` (que es `4095 * 16`), manteniendo la proporcionalidad del valor original. [2]

Esta práctica es común en el diseño de sistemas embebidos y es crucial para la interoperabilidad en una cadena de procesamiento de señal digital. El valor de 16 bits resultante puede ser utilizado directamente por otros módulos de software que esperan esa resolución, simplificando los cálculos subsiguientes como el filtrado o la compensación. Sin embargo, para interpretarlo correctamente en el contexto físico del ADC,

el diseñador del sistema debe ser consciente de que los 4 bits menos significativos no contienen información real del conversor y, por lo tanto, la resolución efectiva sigue siendo de 12 bits. Este conocimiento es clave para aplicar las técnicas de procesamiento adecuadas y garantizar la fidelidad de la señal en la arquitectura global de comunicación.

III. Punto A

Se emplea el codigo mediante la interfaz de thonny a la raspberry (el codigo completo se encuentra en el repositorio de GitHub), El cual activa el A/D del pin GP26 de la ya mencionada. para explicar su funcionamiento de ilustra a continuacion.

```
while True:
    raw16  = adc.read_u16()      # 0..65535 (12 bits alineado a la izq.)
    code12 = raw16 >> 4          # 0..4095 (12 bits reales)
    volts  = (code12 * VREF) / 4095.0
    print(f"{volts:.4f}")        # <-- SOLO un número por línea
    utime.sleep(PERIOD)
```

Ilustracion 1. Relacion read_u16() y code 12

Esto le permite a la raspberry generar un codigo de cuantificacion del conversor A/D de 16 bits, el cual luego se convierte a uno de 12 con la linea *"raw16 >> 4"*. En forma decimal el numero estaria dividiendose en 16, ya que $2^4 = 16$

Para verificar el funcionamiento, agregamos tres lineas al codigo, las cuales permiten visualizar el valor medido en decimal y binario:

```
print("Voltage: {:.4f} V".format(volts))
print("Raw 16-bit: {:5d} (decimal) | {:016b} (binary)".format(raw16, raw16))
print("Code 12-bit: {:4d} (decimal) | {:012b} (binary)".format(code12, code12))
```

Ilustracion 2. Lineas para visualizar enteros medidos

El conversor mide diferentes voltajes segun la escala explicada anteriormente, para verificar su funcionamiento, se toman muestras resultantes y confirmando la conversion de 16 bits a 12. Esto mediante la idea teorica de dividir el decimal en 16, ya que se eliminan 4 bits.

Para entender mejor esta idea se presenta la siguiente muestra:

```
code12: 1606      |     011001000110
1.30
raw16: 25702        |       0110010001100110
code12: 1606      |     011001000110
1.30
raw16: 25702        |       0110010001100110
code12: 1606      |     011001000110
1.30
```

Aqui se puede ver que se esta midiendo un voltaje de 1.3V en donde el codigo de 16 bits en binario, se convierte a uno de 12, para verificar esto tambien se imprimen sus representaciones en decimal, observemos que:

$$raw16 = 25702$$

Entonces para pasar a code12, es necesario dividir en 16 por lo anteriormente explicado.

$$code12 = \frac{raw16}{16}$$

$$code12 = 1606{,}37$$

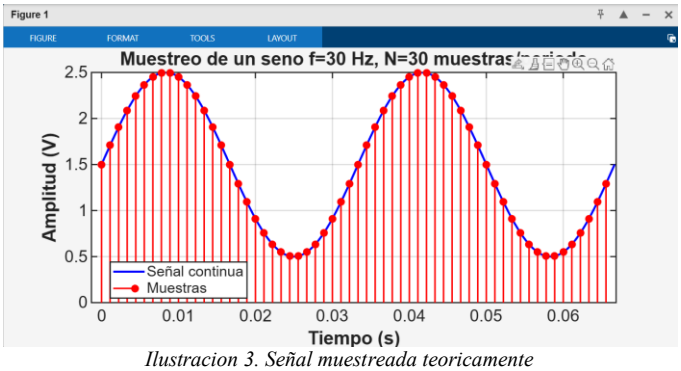
Al ser representacion binaria se aproxima al menor entero 1606, lo cual se confirma en la muestra del labpratorio, ***Si se quiere comprobar el funcionamiento de la relacion entre read16() y code12 con las muestras de las demas escalas, se encuentran en el repositorio de GitHub adjunto.***

IV. Punto B

Para este punto se implementa el codigo adc.m (que se encuentra en el repositorio de GitHub), el cual muestrea la señal

$$x(t) = 1{,}5 + 1\,sen(2\pi(30Hz)t)$$

Los parametros de esta señal fueron diseñados teniendo en cuenta la ventana de cuantificacion que nos brinda la raspberry, ya que va de 0V a 3,316 (medidos directamente cn un multimetro en el pin de voltaje de referecia de la raspberry). Dicho esto se grafica la señal continua y muestreada teoricamente:



Ilustracion 3. Señal muestreada teoricamente

Ahora para hacerlo experimentalmente, mediante el generador de señales configuramos los parametros de DC, amplitud pico a pico y frecuencia. Y los conectamos al conversor A/D y tierra de la raspberry, para que se lea correctamente hay que confirgurar estos parametros tambien en el codigo **ADC_Sampling.py .** implementarlo en la rasberry mediante la interfaz de Thonny,

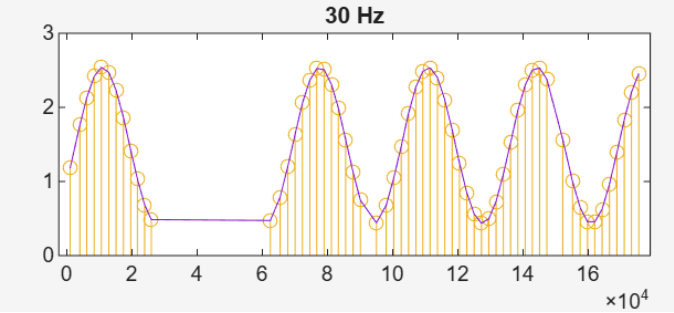
Este codigo genera un archivo .csv el cual recibe por nombre adc_capture_#.csv, donde se registran valores de voltaje con respecto al tiempo, estos archivos son necesarios procesarlos

mediante el siguiente codigo (tener en cuenta que se muestran fragmentos de codigo, completo se encuentra en el repositorio adjunto):

```
t2=adc_capture_3.t_us;
y2=adc_capture_3.volts;
    subplot(4,1,2)
    stem(t2,y2)
    hold on
    plot(t2,y2)
    title('30 Hz')
```

Ilustracion 4. Codigo para muestreo experimental

El comando stem permite graficar la linea vertical sobre la muestra, y el plot nos permite ver la señal continua, obteniendo como resultado:



Ilustracion 5. Muestreo experimental procesado en MATlab

Aqui se evidencia los datos tomados por la raspberry, frente a una señal de 30Hz, se puede observar que hay una perdida de muestras en la señal, esto se debe a que el conversor A/D no logra medir durante ese instante de tiempo, bien sea porque la señal varia muy rapido (frecuencia muy alta), o porque el tiempo de muestra no esta bien configurado segun la señal en cuention.

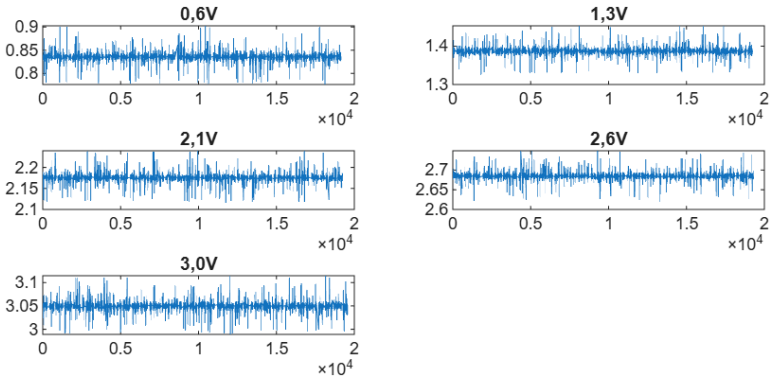
Para solucionar esto se implemnetan relojes de control para indicar a la raspberry cuando tomar una muestra del conversor, haciendo un proceso mas exacto en la toma de datos sin perder muestras.

Se muestrearon 4 señales, 3 senos de diferentes frecuencias y una triangular, se adjunta la que mejor obtuvo muestreo y si se quiere visualizar y analizar el mismo proceso, en el repositorio de GitHub se encuentran las demas señales tanto teoricas como experimentales.

V. Punto C

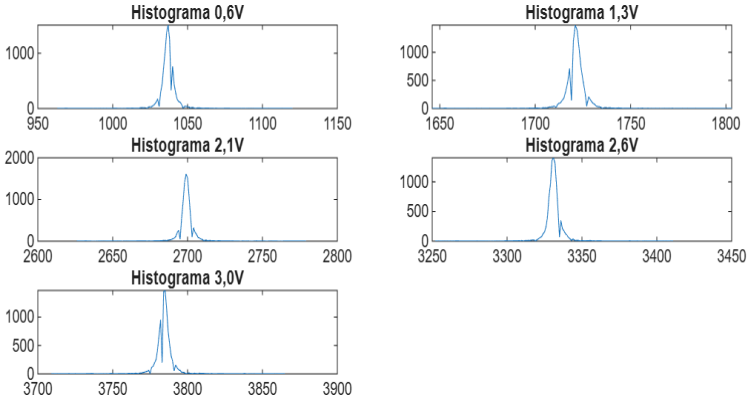
Ahora mediante el codigo sampling_2.py (que se encuentra en el repositorio), la raspberry mide diferentes voltajes DC entre el rango de cuantizacion de la raspberry (0V-3,326V), este codigo permitira obetner 2 archivos .csv (voltaje en el dominio del tiempo e histograma). Los cuales seran procesados en MATlab con el fin de analizar las graficas resultantes:

Se escogen los voltajes de 0.6V, 1.3V, 2.1V, 2.6V y 3V. Mediante la interfaz de Thonny podemos acceder a estos archivos generados y almacenados por la raspberry, descargarlos y procesarlos en MATlab, obteniendo el siguiente resultado:



Ilustracion 6. Muestras de voltaje en el dominio del tiempo

Como bien se menciono, se generan 2 archivos, el segundo es un histograma que define el valor de ADC segun la frecuencia, donde luego del procesamiento en MATlab resulta:



Ilustracion 7. Muestras histogramas

Los codigos que procesan los archivos generados por "sampling_2.py" al igual que los archivos .csv generados, se pueden encontrar en el repositorio de GitHub .

VI. Analisis y Resultados

- → El análisis revela una distinción crítica entre la resolución física del hardware (12 bits) y la representación software mediante read_u16(). Si bien el valor de 16 bits sugiere una mayor precisión, los 4 bits menos significativos son siempre cero, por lo que la resolución efectiva y la relación señal-ruido (SNR) permanecen determinadas únicamente por los 12 bits del ADC. Este enfoque de "alineación a la izquierda" es una solución de software eficiente para la compatibilidad con sistemas que operan nativamente con bytes o words de 16 bits, pero no mejora la capacidad fundamental de cuantificación del sistema.
- → El funcionamiento del ADC se identificó como la etapa inicial y fundamental en la cadena de un

sistema de comunicación digital. Su correcta configuración (rango de voltaje, frecuencia de muestreo) condiciona el desempeño de todas las etapas posteriores.

→ Los resultados experimentales demuestran la importancia crítica de seleccionar parámetros de muestreo y cuantificación apropiados para la señal de interés.

→ En el contexto de una arquitectura de comunicación digital, se concluye que el ADC es el bloque fundamental que interfacea el mundo analógico con el digital. Su correcto diseño y operación es un prerrequisito indispensable para las ventajas de las comunicaciones digitales, como la inmunidad al ruido, el procesamiento de señal y la capacidad de multiplexación.

VII. REFERENCIAS

I. [1] Smith, S. W. (1997). *The scientist and engineer's guide to digital signal processing*. California Technical Publishing. Capítulo 3: ADC and DAC. https://www.dspguide.com/ch3.htm

II. [2] Raspberry Pi Ltd. (2023). *MicroPython documentation: class ADC – analog to digital conversion*. https://docs.micropython.org/en/latest/library/machine.ADC.html

III. [3] https://github.com/tomasramirez20/Practica-4-Com.Digitales