OMNIS 2

Otwartość. Modernizacja. Nowoczesność. Integracja. Społeczność.

Podstawy Elektroniki
Ćwiczenia komputerowe

Przetwarzanie analogowo-cyfrowe i cyfrowo-analogowe

Autor: Dariusz Tefelski

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską
Politechnika Warszawska

Plac Politechniki 1
00-661 Warszawa

www.pw.edu.pl

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską
Politechnika Warszawska

Plac Politechniki 1
00-661 Warszawa
www.pw.edu.pl

1 Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z zasadami działania, parametrami oraz metodami badania przetworników analogowo-cyfrowych (A/C) akiei cyfrowo-analogowych (C/A).

2 Wstęp teoretyczny

2.1 Przetworniki cyfrowo-analogowe (C/A)

Definicja

Przetwornik C/A (DAC – *Digital-to-Analog Converter*) zamienia sygnał cyfrowy (liczbę binarną) na proporcjonalny sygnał analogowy (napięcie lub prąd).

Przetworniki cyfrowo-analogowe (DAC) można podzielić na kilka głównych typów w zależności od ich architektury wewnętrznej, szybkości działania i precyzji. Oto najważniejsze z nich:

1. **Przetwornik z ważonymi rezystorami (Weighted Resistor DAC)**

   To najprostsza koncepcja teoretyczna. Wykorzystuje wzmacniacz operacyjny pracujący jako sumator, do którego podłączone są rezystory o wartościach będących potęgami dwójki (R, 2R, 4R, 8R, ...).

   - `Zasada:` Każdy bit steruje kluczem, który dołącza odpowiedni rezystor do napięcia referencyjnego.
   - `Wada:` Bardzo trudny do wykonania przy wysokich rozdzielczościach (np. 16 bitów), ponieważ wymaga rezystorów o skrajnie różnych wartościach i bardzo wysokiej precyzji.

2. **Przetwornik drabinkowy R-2R (R-2R Ladder DAC)**

   Najpowszechniejsza architektura w układach scalonych średniej prędkości. Wykorzystuje tylko dwie wartości rezystancji (R oraz $2R$).

   - `Zasada:` Drabinka dzieli napięcie referencyjne tak, że każdy kolejny węzeł wnosi połowę mniejszy prąd/napięcie do sumy wyjściowej.
   - `Zaleta:` Wymaga tylko dwóch wartości rezystorów, co jest łatwe do uzyskania w procesie produkcji półprzewodników (poprzez laserowe trymowanie) i zapewnia dobrą liniowość.
   - `Wady:` Duża liczba komponentów przy wysokich rozdzielczościach.

3. **Przetwornik z przełączanymi źródłami prądowymi (Current Steering DAC)**

   Stosowany w aplikacjach bardzo wysokiej prędkości (np. w komunikacji radiowej, generatorach sygnałów).

- `Zasada:` Zamiast rezystorów stosuje się precyzyjne źródła prądowe (tranzystory), które są przełączane bezpośrednio do wyjścia.
- `Zaleta:` Ekstremalnie szybki, może pracować z częstotliwościami rzędu gigaherców (GHz).

4. **Przetwornik Sigma-Delta (ΣΔ DAC)**

   Najpopularniejszy typ w zastosowaniach audio (karty dźwiękowe, smartfony).

   - `Zasada:` Wykorzystuje nadpróbkowanie (oversampling) i modelowanie szumu (noise shaping). Zamiast wielu bitów, generuje ciąg o bardzo wysokiej częstotliwości (często 1-bitowy), którego średnia wartość odpowiada sygnałowi analogowemu.
   - `Zaleta:` Bardzo wysoka liniowość i rozdzielczość (nawet 24-32 bity) przy niskim koszcie produkcji.
   - `Wada:` Wymaga skomplikowanych filtrów cyfrowych i analogowych na wyjściu.

5. **Przetwornik segmentowy (Segmented DAC)**

   Hybryda łącząca kilka technik.

   - `Zasada:` Kilka najbardziej znaczących bitów (MSB) jest przetwarzanych przez architekturę termometryczną (identyczne źródła prądowe, co poprawia liniowość), a pozostałe bity (LSB) przez drabinkę R-2R.
   - `Zaleta:` Łączy precyzję z szybkością, minimalizując błędy przełączania (tzw. glitches).

6. **Przetwornik PWM (Pulse Width Modulation)**

   Często stosowany w mikrokontrolerach (np. Arduino) jako „tani” DAC.

   - `Zasada:` Zmiana wypełnienia sygnału prostokątnego o stałej częstotliwości. Sygnał analogowy uzyskuje się po przejściu przez prosty filtr dolnoprzepustowy RC.
   - `Zastosowanie:` Sterowanie silnikami, jasnością LED, proste systemy audio.

Warto wiedzieć

Jaki przetwornik C/A (DAC) wybrać?

- Audio (muzyka): Sigma-Delta (wysoka precyzja, tani).
- Pomiary laboratoryjne: R-2R (dobra liniowość DC, stabilność).
- Wideo i Radiokomunikacja: Current Steering (bardzo wysoka prędkość).
- Prosta automatyka: PWM (brak dedykowanego układu DAC).

2.2 Przetworniki analogowo-cyfrowe (A/C)

Definicja

Przetwornik A/C (ADC – *Analog-to-Digital Converter*) dokonuje kwantyzacji sygnału ciągłego w czasie i amplitudzie na postać dyskretną (cyfrową).

Warto wiedzieć

Wybór architektury przetwornika analogowo-cyfrowego (ADC) zależy od tego, co jest priorytetem: szybkość, rozdzielczość (dokładność), czy pobór mocy.

1. **Przetwornik Flash (Równoległy)**

   To najszybszy możliwy typ przetwornika.

   - `Zasada działania:` Wykorzystuje dużą liczbę komparatorów, które jednocześnie porównują napięcie wejściowe z zestawem napięć referencyjnych (dzielnik rezystorowy). Dla n bitów potrzeba $2^n - 1$ komparatorów.
   - `Zalety:` Ekstremalnie szybki (częstotliwości rzędu GHz).
   - `Wady:` Bardzo drogi, pobiera dużo prądu, ma małą rozdzielczość (zwykle do 8-10 bitów), bo każdy kolejny bit podwaja liczbę komparatorów.
   - `Zastosowanie:` Oscyloskopy cyfrowe, systemy radarowe, komunikacja światłowodowa.

2. **Przetwornik SAR (Sukcesywna Aproksymacja)**

   Najbardziej uniwersalny i najczęściej spotykany (np. w mikrokontrolerach).

   - `Zasada działania:` Wykorzystuje algorytm przeszukiwania binarnego. Sprawdza bit po bicie (od najbardziej znaczącego MSB do najmniej znaczącego LSB), porównując napięcie wejściowe z wartością generowaną przez wewnętrzny przetwornik DAC.
   - `Zalety:` Dobry kompromis między szybkością a rozdzielczością (zwykle 12-16 bitów), niski pobór mocy.
   - `Wady:` Wymaga układu Sample-and-Hold (S/H) na wejściu, by napięcie nie zmieniło się podczas cyklu konwersji.
   - `Zastosowanie:` Systemy akwizycji danych, sterowanie silnikami, medycyna.

3. **Przetwornik Sigma-Delta** ($\Sigma\Delta$)

   Król rozdzielczości i precyzji.

   - `Zasada działania:` Opiera się na nadpróbkowaniu (oversampling) i modelowaniu szumu (noise shaping). Przetwarza sygnał z bardzo wysoką częstotliwością, ale z niską rozdzielczością (często 1-bitową), a następnie stosuje zaawansowaną filtrację cyfrową.

- `Zalety:` Bardzo wysoka rozdzielczość (nawet 24-32 bity), wbudowana odporność na zakłócenia, wbudowana filtracja cyfrowa, brak potrzeby stosowania skomplikowanych filtrów analogowych.
- `Wady:` Wolny (częstotliwości wyjściowe są małe w porównaniu do taktowania), duże opóźnienia (latency).
- `Zastosowanie:` Audio hi-fi, precyzyjne wagi, czujniki temperatury, sejsmologia.

4. **Przetwornik Potokowy (Pipelined)**

   Architektura pośrednia między Flash a SAR.

   - `Zasada działania:` Proces konwersji jest podzielony na etapy (potoki). Każdy etap przetwarza kilka bitów i przekazuje resztę sygnału (błąd) do następnego etapu. Gdy pierwszy sygnał jest w drugim etapie, przetwornik może już zacząć pobierać kolejną próbkę.
   - `Zalety:` Wysoka przepustowość (szybszy niż SAR), dobra rozdzielczość (12-14 bitów).
   - `Wady:` Opóźnienie wyniku (wynik pojawia się po kilku cyklach zegara), trudniejsza kalibracja etapów.
   - `Zastosowanie:` Obrazowanie medyczne, wideo cyfrowe, nowoczesna radiokomunikacja (Software Defined Radio).

5. **Przetwornik Całkujący (Dual-Slope)**

   Stosowany tam, gdzie liczy się stabilność, a nie szybkość.

   - `Zasada działania:` Mierzy czas ładowania i rozładowania kondensatora przez napięcie wejściowe i referencyjne. Czas ten jest proporcjonalny do napięcia.
   - `Zalety:` Doskonale eliminuje zakłócenia (np. przydźwięk sieci 50Hz/60Hz), bardzo liniowy, tani.
   - `Wady:` Bardzo wolny (kilka-kilkanaście pomiarów na sekundę).
   - `Zastosowanie:` Multimetry cyfrowe (DMM), panele pomiarowe.

Podsumowanie porównawcze:

Typ ADC	Szybkość	Rozdzielczość	Pobór mocy
Flash	Ekstremalna	Niska (8bit)	Bardzo wysoki
SAR	Średnia/Wysoka	Średnia (12-16 bit)	Bardzo niski
Sigma-Delta	Niska	Bardzo wysoka (24-32 bit)	Średni
Pipelined	Wysoka	Średnia (12-14 bit)	Średni/Wysoki
Całkujący	Bardzo niska	Wysoka	Niski

2.3 Podstawowe parametry i błędy

- **Rozdzielczość:** Liczba dyskretnych poziomów, na które dzielony jest zakres napięcia (wyrażona w bitach).
- **Prędkość (Sampling Rate):** Liczba konwersji na sekundę (SPS – *Samples Per Second*).
- **LSB (Least Significant Bit):** Najmniejsza zmiana napięcia, którą przetwornik może rozróżnić. Wartość LSB obliczamy jako:

$$V_{LSB} = \frac{V_{REF}}{2^n} \tag{1}$$

  gdzie V_{REF} to napięcie referencyjne, a n to liczba bitów. Rzeczywista rozdzielczość jest ograniczona przez szum i nieliniowość, co opisuje parametr ENOB.
- **Błąd kwantyzacji:** Nieunikniony błąd wynikający z dyskretyzacji sygnału, wynoszący maksymalnie ± 0.5 LSB dla idealnego przetwornika.
- **Błąd nieliniowości całkowej (INL):** Odchylenie rzeczywistej charakterystyki od linii idealnej.
- **Błąd nieliniowości różniczkowej (DNL):** Różnica między rzeczywistym krokiem napięcia a idealnym 1 LSB. Jeśli $DNL < -1$, przetwornik może być niemonotoniczny.
- **Napięcie referencyjne (V_{REF}):** Kluczowy parametr określający skalę przetwarzania. Stabilność i dokładność V_{REF} bezpośrednio wpływa na wynik pomiaru.

2.4 Konwersja wartości cyfrowej na napięcie

Wzór na obliczenie napięcia pomiarowego V_{in} na podstawie wartości cyfrowej D (przy założeniu idealnego przetwornika):

$$V_{in} = D \times \frac{V_{REF}}{2^n} \tag{2}$$

2.5 Współpraca części analogowej z przetwornikiem

Prawidłowe doprowadzenie sygnału do przetwornika wymaga uwzględnienia kilku czynników:

- **Impedancja wejściowa:** Przetworniki (szczególnie SAR) mogą pobierać prąd w momentach próbkowania. Konieczne jest stosowanie wzmacniaczy operacyjnych w układzie bufora (wtórnika napięcia).
- **Filtrowanie (Antialiasing):** Zgodnie z twierdzeniem Nyquista-Shannona, częstotliwość próbkowania musi być co najmniej dwa razy większa niż najwyższa składowa częstotliwość sygnału. Filtr dolnoprzepustowy wycina składowe powyżej częstotliwości Nyquista.

- **Dopasowanie poziomów napięć:** Sygnał musi mieścić się w zakresie $0 \dots V_{REF}$ (lub $-V_{REF} \cdots + V_{REF}$ dla przetworników bipolarnych).

2.6 Sposoby badania przetworników

- **Metoda statyczna (Histogram):** Polega na podaniu na wejście wolnozmiennego sygnału (np. trójkątnego) i zliczaniu wystąpień poszczególnych kodów cyfrowych. Pozwala na wyznaczenie DNL i INL.
- **Metoda dynamiczna (FFT):** Analiza widmowa sygnału wyjściowego po podaniu czystego sinusa. Pozwala wyznaczyć SNR, THD (Total Harmonic Distortion) oraz ENOB.

Warto wiedzieć

Wyznaczanie INL (Integral Nonlinearity – nieliniowość całkowa) przetwornika ADC polega na zmierzeniu, jak bardzo rzeczywista charakterystyka przejściowa odbiega od linii idealnej. O ile DNL (Differential Nonlinearity - nieliniowość różniczkowa) mówi nam o błędzie pojedynczego "schodka", o tyle INL mówi o skumulowanym błędzie całej charakterystyki.

Procedura wyznaczania INL krok po kroku z wykorzystaniem najpopularniejszej w technice pomiarowej metody histogramowej:

1. **Przygotowanie stanowiska**
   - Na wejście ADC podajemy sygnał o bardzo wysokiej liniowości (zwykle falę trójkątną lub piłokształtną), której zakres nieco przekracza zakres wejściowy przetwornika (tzw. overdrive).
   - Częstotliwość sygnału musi być nieskorelowana z zegarem próbkowania, aby każda wartość napięcia miała taką samą szansę na bycie spróbkowaną.
2. **Zbieranie danych (tworzenie histogramu)**
   - Rejestrujemy bardzo dużą liczbę próbek (np. setki tysięcy lub miliony).
   - Tworzymy histogram: zliczamy, ile razy wystąpił każdy z możliwych kodów cyfrowych (0, 1, 2, ..., $2^n - 1$).
   - Dla idealnego przetwornika i idealnej fali trójkątnej, każdy kod powinien wystąpić dokładnie taką samą liczbę razy (ponieważ każdy "schodek"ma tę samą szerokość 1 LSB).
3. **Obliczenie DNL (krok pośredni)**

   Zanim wyznaczymy INL, musimy znać błędy szerokości każdego schodka (DNL):

   (a) Obliczamy średnią liczbę zliczeń na jeden kod ($H_{avg} = \frac{\sum_{i=0}^{n-1} H_i}{2^n}$).

(b) Dla każdego kodu i obliczamy DNL: $DNL(i) = \frac{H_i - H_{avg}}{H_{avg}}$ Gdzie H_i to liczba zliczeń dla kodu i.

4. **Obliczenie INL (Całkowanie)**

   INL jest sumą kumulacyjną błędów DNL. Wartość INL dla danego kodu k to suma wszystkich błędów DNL od zera do tego kodu: $INL(k) = \sum_{i=0}^{k} DNL(i)$

Metody odniesienia (Linia Idealna)

Podczas wyznaczania INL ważne jest, do jakiej „linii idealnej" się odnosimy. Stosuje się dwie metody:

1. `Metoda punktów końcowych (Best Straight Line - Endpoints)`: Linia przechodzi dokładnie przez pierwszy i ostatni zmierzony kod. Jest to najczęstsza metoda w kartach katalogowych.
2. `Metoda najmniejszych kwadratów (Best Fit)`: Linia jest prowadzona tak, aby zminimalizować średni błąd dla wszystkich kodów. Daje ona wizualnie „ładniejsze" (mniejsze) wyniki INL, ale trudniej ją odnieść do realnej kalibracji systemu.

Interpretacja wyniku:

- Jeśli $INL = 0$, przetwornik jest idealnie liniowy.
- Jeśli INL wykazuje "brzuch"(wygięcie w jedną stronę), oznacza to błąd wzmocnienia lub nieliniowość statyczną układu wejściowego.
- Jeśli gwałtownie skacze, przyczyną mogą być błędy w wagach bitów (np. złe dopasowanie rezystorów w drabinie R-2R).

W skrócie: Aby wyznaczyć INL, mierzysz jak "szerokie"są schodki (histogram), sprawdzasz o ile różnią się od ideału (DNL) i sumujesz te różnice wzdłuż całej charakterystyki (INL).

Definicja

SNR (Signal-to-Noise Ratio) – Stosunek sygnału do szumu
Jest to stosunek mocy użytecznego sygnału (zazwyczaj czystej sinusoidy o pełnej skali) do mocy szumów obecnych w pasmie częstotliwości od DC do częstotliwości Nyquista ($f_{sampling}/2$).

- `Co zawiera:` SNR uwzględnia szum kwantyzacji, szum termiczny oraz szum fazowy (jitter).
- `Czego NIE zawiera:` W klasycznej definicji SNR dla przetworników wyklucza się moc składowych harmonicznych (THD) oraz składową stałą (DC).
- `Wzór teoretyczny:` Dla idealnego n-bitowego przetwornika, gdzie jedynym źródłem szumu jest kwantyzacja: $SNR[dB] = 6.02n + 1.76$

Definicja

SINAD (Signal-to-Noise and Distortion) – Stosunek sygnału do szumu i zniekształceń
Jest to stosunek mocy sygnału użytecznego do sumarycznej mocy wszystkich pozostałych składowych widmowych (szumu oraz zniekształceń harmonicznych), z wyłączeniem składowej stałej (DC).

- `Co zawiera:` SINAD łączy w sobie `SNR` oraz `THD` (Total Harmonic Distortion). Jest to bardziej „życiowy" parametr, ponieważ uwzględnia nieliniowości przetwornika, które objawiają się powstawaniem harmonicznych.
- `Zależność:` $SINAD = \frac{P_{signal}}{P_{noise}+P_{distortion}}$

W skali logarytmicznej SINAD jest zawsze niższy (gorszy) niż SNR, ponieważ mianownik jest powiększony o moc zniekształceń.

`SINAD` jest podstawą do obliczenia parametru **ENOB** (Effective Number of Bits), czyli rzeczywistej rozdzielczości bitowej, którą „czujemy" w pomiarach: $ENOB = (SINAD[dB] - 1.76)/(6.02)$

SNR - mówi o poziomie „tła szumowego", **SINAD** - mówi o całkowitej degradacji sygnału (szum + zniekształcenia nieliniowe).

3 Zadania do realizacji

Do wykonania zadań wymagane są biblioteki: `numpy`, `matplotlib`, `scipy`.

3.1 Zadanie 1: Przetworniki A/C i C/A

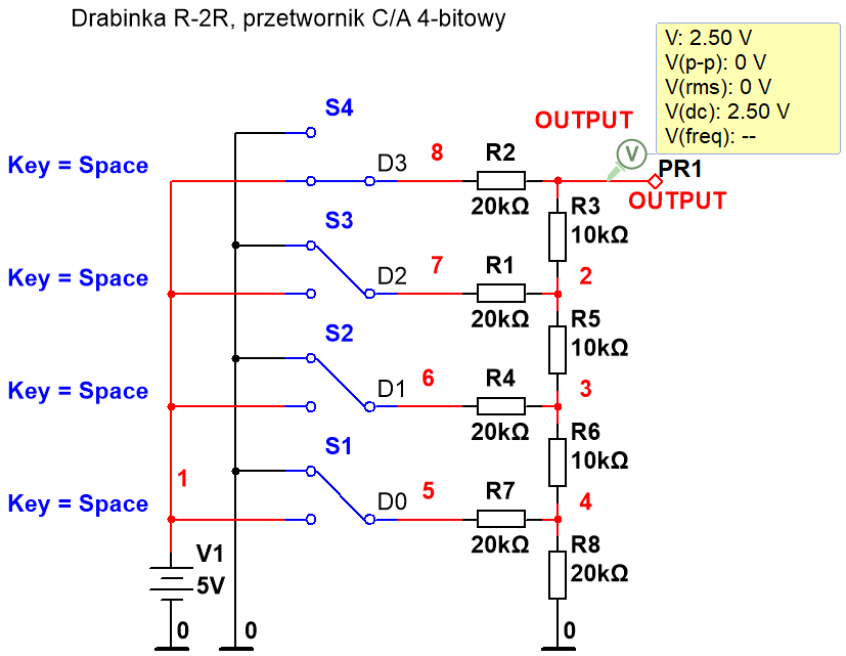
Zadanie

W programie Multisim zapoznaj się z przykładowym 8-bitowym przetwornikiem A/C, który znajduje się w: File ⟶ Open samples... ⟶ Educational Sample Circuits ⟶ Applications Circuits ⟶ **ADCexample** oraz z 4-bitowym przetwornikiem C/A, który znajduje się w : File ⟶ Open samples... ⟶ Educational Sample Circuits ⟶ Applications Circuits ⟶ **WeightedAverageDAC**

3.2 Zadanie 2: Symulacja drabinki R-2R

Zaimplementuj model matematyczny przetwornika R-2R o rozdzielczości 4 bitów. Wyznacz charakterystykę przejściową $V_{out} = f(Digital\_In)$. Wykonaj symulację przetwornika C/A 4-bitowego: drabinki R-2R w Multisim.

Symulację drabinki R-2R przedstawia listing 3.1, natomiast sposób utworzenia drabinki R-2R w Multisim przedstawia schemat na rys. 1. W tym przypadku wygodnie jest wykorzystać przełączniki `SPDT`.



Rysunek 1: Multisim: 4-bitowy przetwornik C/A, drabinka R-2R.

Listing 3.1: Skrypt przedstawiający działanie przetwornika C-A w oparciu o drabinkę rezystorów R-2R

```
import numpy as np
import matplotlib.pyplot as plt

Nbit = 4
num = 2**Nbit

# Symulacja dla Nbit bitow (od LSB do MSB)
v_ref = 5.0
results = []


def r2r_dac(bits, v_ref):
    n = len(bits)
    v_out = 0
    for i in range(n):
        v_out += bits[i] * (v_ref / (2 ** (i + 1)))
    return v_out


for i in range(num):
    binary = [int(x) for x in format(i, f"0{Nbit}b")]
    val = r2r_dac(binary, v_ref)
    print(f"{i} {binary} {val}")
    results.append(r2r_dac(binary, v_ref))
```

Listing 3.1: Skrypt przedstawiający działanie przetwornika C-A w oparciu o drabinkę rezystorów R-2R (c.d.)

```
plt.figure(figsize=(8, 6))

plt.step(range(num), results, where="mid")
plt.xlabel("Wartosc cyfrowa")
plt.ylabel("Napiecie wyjsciowe [V]")
plt.title("Charakterystyka przetwornika R-2R (4-bit)")
plt.grid(True)
plt.show()
```

3.3 Zadanie 3: Algorytm Sukcesywnej Aproksymacji (SAR)

Napisz funkcję w Pythonie symulującą działanie algorytmu SAR ADC. Funkcja powinna przyjmować napięcie wejściowe, napięcie referencyjne oraz liczbę bitów.

Listing 3.2: Skrypt przedstawiający działanie algorytmu sukcesywnej aproksymacji w przetworniku A/C SAR

```
def sar_adc(v_in, v_ref, n_bits):
    v_upper = v_ref
    v_lower = 0
    digital_out = 0
    current_v_dac = 0

    for i in range(n_bits - 1, -1, -1):
        # Ustawienie bitu i
        trial_digital = digital_out | (1 << i)
        trial_v_dac = trial_digital * (v_ref / (2**n_bits))

        if trial_v_dac <= v_in:
            digital_out = trial_digital

    return digital_out


v_ref = 5.0
n_bits = 8
v_test = 3.14159
result = sar_adc(v_test, v_ref, n_bits)
v_recovered = result * (v_ref / (2**n_bits))

print(f"Napiecie wejsciowe: {v_test}V")
print(f"Wynik cyfrowy: {result} (0x{result:02X})")
print(f"Napiecie odtworzone: {v_recovered}V")
```

3.4 Zadanie 4: Analiza szumu kwantyzacji i ENOB

Wygeneruj sygnał sinusoidalny, poddaj go kwantyzacji i oblicz błąd kwantyzacji. Wyznacz rzeczywistą rozdzielczość (ENOB – *Effective Number of Bits*).

Listing 3.3: Skrypt wyznaczający rzeczywistą rozdzielczość przetwornika, z wykorzystaniem sygnału sinusoidalnego.

```
import numpy as np
import matplotlib.pyplot as plt

NOISY_ADC = False
NOISE_AMPLITUDE = 0.1

Nbits = 4
V_ref = 5.0


def calculate_enob(signal, quantized_signal):
    # Noise is the difference between original and quantized signal
    noise = quantized_signal - signal

    # Power of the signal (variance for AC signals)
    p_signal = np.var(signal)
    # Power of the noise (variance of the quantization error)
    p_noise = np.var(noise)

    # Signal-to-Noise Ratio in dB
    # Formula: SNR = 10 * log10(P_signal / P_noise)
    snr_val = 10 * np.log10(p_signal / p_noise)

    # Effective Number of Bits
    # Standard formula for ideal ADC: SNR = 6.02 * n + 1.76
    # So: n = (SNR - 1.76) / 6.02
    enob = (snr_val - 1.76) / 6.02
    return snr_val, enob


# Simulation parameters
t = np.linspace(0, 1, 10000)  # Increased samples for better statistics


# Signal: Sine wave spanning the full range [0, V_ref]
# Peak-to-peak amplitude = V_ref, centered at V_ref/2
amplitude = V_ref / 2
signal = amplitude * np.sin(2 * np.pi * 5 * t) + amplitude


noise = np.random.normal(0, NOISE_AMPLITUDE, len(t))
```

Listing 3.3: Skrypt wyznaczający rzeczywistą rozdzielczość przetwornika, z wykorzystaniem sygnału sinusoidalnego. (c.d.)

```
if NOISY_ADC:
    # signal_noise= np.where(signal>V_ref/2.0, signal - np.abs(noise),
    ↪  signal + np.abs(noise))
    signal_noise = signal + noise
else:
    signal_noise = signal

# Quantization logic
# levels = 2^n. For 8 bits, levels = 256.
# Discrete values are 0, 1, ..., 255.
levels = 2**Nbits
# To get ideal SNR = 6.02*n + 1.76, the signal should be quantized into
↪  exactly 2^n levels.
# Using np.floor or np.round with (levels) instead of (levels-1)
quantized_indices = np.floor((signal_noise / V_ref) * levels)
# Ensure we don't exceed the last level (levels-1)
quantized_indices = np.clip(quantized_indices, 0, levels - 1)
# Convert back to voltage - using the center of each bin for
↪  reconstruction
# to minimize mean error (bias)
quantized_signal = (quantized_indices + 0.5) * (V_ref / levels)
# quantized_signal = (quantized_indices) * (V_ref / levels)

snr_val, enob = calculate_enob(signal, quantized_signal)

print(f"Liczba bitów (n): {Nbits}")
print(f"Poziomów: {levels}")
print(f"LSB val: {(V_ref / levels):f} V")
print(f"SNR: {snr_val:.2f} dB")
print(f"ENOB: {enob:.2f} bits")
print(f"Error: ±{(V_ref / (2**enob)):.2f} V")
plt.figure(figsize=(16, 16))
plt.plot(t, signal, label="Original")
plt.plot(t, signal_noise, label="Original + Noise", alpha=0.5)
plt.step(t, quantized_signal, label="Quantized (8-bit)", color="r")
plt.legend()
plt.title("Kwantyzacja sygnalu sinusoidalnego")
plt.grid(True)
plt.show()
```

3.5 Zadanie 5: Oversampling

Napisz skrypt, który wykaże, jak technika nadpróbkowania i cyfrowa filtracja poprawiają efektywną ilość bitów (ENOB) w przetworniku A/C.

Skrypt przedstawia listing 3.4.

Mechanizm działania:

1. Sygnał jest próbkowany z częstotliwością znacznie wyższą niż wynika to z twierdzenia Nyquista (OSR - Oversampling Ratio = 64).
2. Szum kwantyzacji, który normalnie jest rozłożony w pasmie do $f_{sampling}/2$ (częstotliwość Nyquista), zostaje „rozciągnięty" na znacznie szersze pasmo.
3. Zastosowanie cyfrowego filtra dolnoprzepustowego wycina szum znajdujący się poza pasmem użytecznym.
4. Każde czterokrotne zwiększenie częstotliwości próbkowania ($OSR = 4$) pozwala teoretycznie zyskać 1 bit rozdzielczości (6,02 dB SNR).

Uwaga!

Aby technika **Oversampling-u** była skuteczna dla mierzonych sygnałów wejściowych, zwłaszcza o stałej wartości lub wolno zmiennych konieczne jest dodanie szumu (dither/noise), który będzie powodował, że odczytywane będą sąsiednie kody na przetworniku, a efektywna wartość konwersji będzie zależała od średniej wartości z tych kodów, które będą występowały ilościowo proporcjonalnie, do którego kodu będzie bliżej mierzonej wartości rzeczywistej.

Gdybyśmy otrzymywali przy pomiarze stałej wartości zawsze ten sam kod (nie byłoby rozmycia), to technika Oversamplingu by nie działała.

Listing 3.4: Skrypt przedstawiający zysk efektywnej ilości bitów przy zastosowaniu techniki „Oversampling-u".

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal


def simulate_oversampling(n_bits=8, osr=16):
    """
    Simulates ADC oversampling and filtering.
    n_bits: internal ADC resolution
    osr: Oversampling Ratio (fs_over / fs_nyquist)
    """
    fs_nyquist = 1000  # Target Nyquist rate (after decimation)
    fs_over = fs_nyquist * osr
    duration = 0.5
    t = np.arange(0, duration, 1 / fs_over)

    # Input signal: Sine wave + small amount of dither/noise
    # (Dither is crucial for oversampling to work with DC or slow
    ↪  signals)
```

Listing 3.4: Skrypt przedstawiający zysk efektywnej ilości bitów przy zastosowaniu techniki „Oversampling-u". (c.d.)

```
f_sig = 5.23  # Frequency of interest
v_ref = 2.0
amplitude = 0.9 * (v_ref / 2)
clean_signal = amplitude * np.sin(2 * np.pi * f_sig * t) + (v_ref / 2)

# Add small dither (noise) to ensure quantization error is
↪  decorrelated
dither = np.random.normal(0, v_ref / (2**n_bits) / 2, len(t))
noisy_signal = clean_signal + dither

# 1. Quantization (at high speed)
levels = 2**n_bits
quantized_high = np.floor((noisy_signal / v_ref) * levels)
quantized_high = np.clip(quantized_high, 0, levels - 1)
quantized_high_v = (quantized_high + 0.5) * (v_ref / levels)

# 2. Digital Low-Pass Filtering
# We filter out everything above the target Nyquist frequency
nyq_target = fs_nyquist / 2
# Simple FIR filter or Butterworth
b, a = signal.butter(4, nyq_target / (fs_over / 2), btype="low")
filtered_signal = signal.filtfilt(b, a, quantized_high_v)

# 3. Decimation (downsampling)
decimated_signal = filtered_signal[::osr]
t_decimated = t[::osr]
clean_decimated = clean_signal[::osr]

# Calculate SNR and ENOB
def get_metrics(orig, quant):
    noise = quant - orig
    snr = 10 * np.log10(np.var(orig) / np.var(noise))
    enob = (snr - 1.76) / 6.02
    return snr, enob

snr_raw, enob_raw = get_metrics(clean_signal, quantized_high_v)
snr_osr, enob_osr = get_metrics(clean_decimated, decimated_signal)

# FFT Analysis for visualization
def plot_fft(sig, fs, label, color):
    win = np.hanning(len(sig))
    xf = np.fft.rfftfreq(len(sig), 1 / fs)
    yf = np.fft.rfft(sig * win)
    mag = 20 * np.log10(np.abs(yf) / len(sig))
    plt.plot(xf, mag, label=label, color=color, alpha=0.7)
```

Listing 3.4: Skrypt przedstawiający zysk efektywnej ilości bitów przy zastosowaniu techniki „Oversampling-u". (c.d.)

```
plt.figure(figsize=(12, 8))

# Frequency domain plot
plt.subplot(2, 1, 1)
plot_fft(
    quantized_high_v, fs_over, f"Raw {n_bits}-bit (High speed)",
    ↪ "gray"
)
plot_fft(decimated_signal, fs_nyquist, f"Oversampled (OSR={osr})",
↪ "blue")
plt.title(f"Widmo sygnału: Surowy vs Oversampling (OSR={osr})")
plt.ylabel("Amplituda [dB]")
plt.xlabel("Częstotliwość [Hz]")
plt.grid(True)
plt.legend()

# Time domain comparison
plt.subplot(2, 1, 2)
plt.plot(t[:400], clean_signal[:400], "r--", label="Oryginał",
↪ alpha=0.5)
plt.step(
    t[:400],
    quantized_high_v[:400],
    "gray",
    label=f"Kwantyzacja {n_bits}-bit",
)
plt.plot(
    t_decimated[: 400 // osr],
    decimated_signal[: 400 // osr],
    "b-o",
    label="Po filtracji i decymacji",
)
plt.title("Porównanie w dziedzinie czasu (fragment)")
plt.xlabel("Czas [s]")
plt.ylabel("Napięcie [V]")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

print("-" * 30)
print(f"Parametry: Bity={n_bits}, OSR={osr}")
print(f"Surowy  - SNR: {snr_raw:.2f} dB, ENOB: {enob_raw:.2f} bits")
print(f"Oversampled - SNR: {snr_osr:.2f} dB, ENOB: {enob_osr:.2f}
↪ bits")
```

Listing 3.4: Skrypt przedstawiający zysk efektywnej ilości bitów przy zastosowaniu techniki „Oversampling-u". (c.d.)

```
    print(f"Zysk rozdzielczości: {enob_osr - enob_raw:.2f} bits")
    print(
        f"Teoretyczny zysk: {np.log2(osr):.2f} bits (dla filtracji i
        ↪ decymacji)"
    )
    print("-" * 30)


# Uruchomienie symulacji
simulate_oversampling(n_bits=8, osr=64)
```

Kluczowe wnioski z symulacji:

- Na wykresie widmowym zauważysz, że tło szumowe (noise floor) dla sygnału z nadpróbkowaniem jest znacznie niżej niż dla surowego przetwornika.
- W dziedzinie czasu sygnał po decymacji jest znacznie „gładszy" i lepiej przybliża oryginał niż schodkowy przebieg 8-bitowy.
- Parametr ENOB wzrasta o ok. 0.5 bita na każde podwojenie częstotliwości (jeśli stosujemy samo filtrowanie) lub o 1 bit, jeśli stosujemy zaawansowane techniki jak **noise shaping** (typowe dla przetworników Sigma-Delta). W powyższym skrypcie zysk wynosi $log_2(\sqrt{OSR})$, co dla $OSR = 64$ daje ok. 3 bity zysku.

4 Wnioski i badanie przetworników

W ramach ćwiczenia należy:

1. Porównać teoretyczną rozdzielczość z wyznaczonym ENOB.
2. Zaobserwować wpływ napięcia referencyjnego na zakres i błąd kwantyzacji.
3. Wyjaśnić, dlaczego rzeczywiste przetworniki mają błędy nieliniowości INL/DNL (np. niedopasowanie rezystorów w drabinie R-2R).

Informacje dodatkowe

Więcej informacji dostępne jest na stronie przedmiotu EwEF, w zakładce dotyczącej ćwiczeń: https://labe.engined.eu/index.php/%C4%86wiczenia_EwEF. W szczególności ta tematyka zawarta jest w materiałach z wykładu 7: https://labe.engined.eu/data/_uploaded/media/EwEF/EwEF_w7.pdf. Wykład 7 w postaci wideo zawiera sporo dodatkowych informacji. https://labe.engined.eu/data/_uploaded/media/EwEF/EwEF_w7.mp4.

4.1 Wymagane oprogramowanie

Do wykonania ćwiczenia wymagany jest NI Multisim, Jupyter-lab i Python 3 z zainstalowanymi modułami:

- `Matplotlib` — do wizualizacji wyników,
- `Numpy` — do podstawowych struktur danych i obliczeń,
- `Scipy` — do obliczeń na sygnałach

5 Autorzy i historia opracowania

- dr inż. Dariusz Tefelski - wersja z 2026r.