# Podstawy Systemów Wbudowanych

*Wykład 3: Zasady tworzenia oprogramowania wbudowanego*

Angelika Tefelska  Dariusz Tefelski

Zakład Fizyki Jądrowej, Wydział Fizyki PW

16 marca 2017

# O czym będzie wykład...

1. Optymalizacja kodu
2. Pamięć
3. Przetwarzanie C/A
4. Przetwarzanie A/C

# Cechy systemów wbudowanych

## Cechy systemów wbudowanych

- Małe rozmiary, mała ilość pamięci operacyjnej (RAM - Random Access Memory)
- Ograniczona ilość pamięci programu
- W przypadku prostych mikrokontrolerów, głównie architektura Harwarda w przeciwieństwie do Von Neumanna.
  - Architektura von Neumanna - dane przechowywane są wspólnie z instrukcjami.
  - Architektura Harwarda - pamięć danych programu jest oddzielona od pamięci rozkazów.
- Mały pobór energii
- Względnie mała prędkość
- Rozbudowane możliwości kontroli niskopoziomowej nad peryferiami

## Sposoby programowania

- Proste mikrokontrolery zwykle wymagają dedykowanego programatora
- Zwykle możliwe jest wykorzystanie „bootloadera" - kosztem niewielkiego wykorzystania pamięci programu, możliwe jest zaprogramowanie mikrokontrolera z wykorzystaniem wbudowanych peryferiów komunikacyjnych takich jak np. port szeregowy, karta pamięci SD, itp.
- Różne zestawy Arduino/Genuino wykorzystują powyższe techniki aby uprościć programowanie - do którego potrzebne jest podłączenie do komputera poprzez interfejs USB oraz środowisko IDE, np. Arduino Leonardo posiadając obsługę USB wykorzystuje bezpośrednio programowanie poprzez bootloader. Inne płytki oparte o starsze mikrokontrolery AVR posiadają dodatkowy układ do komunikacji poprzez USB, np. konwerter USB-RS232 firmy FTDI np. FT232R albo dodatkowy mikrokontroler np. Atmega16U2, który zapewnia transmisję.
- Zaawansowane systemy wykorzystują często systemy operacyjne, które także mają dostęp do różnego rodzaju mediów, z których może być uruchomiony program sterujący, ale zawsze istnieje możliwość programowania tzw. „bare-metal".

# Cechy programów wbudowanych

## Cechy programów wbudowanych

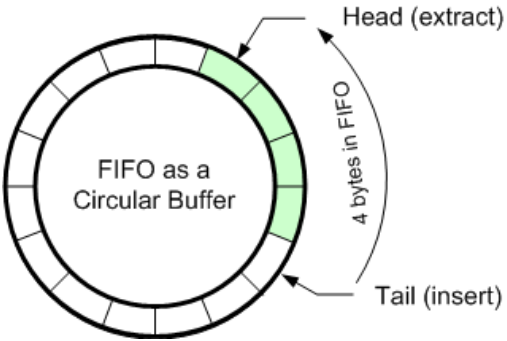
- Zwięzłość - programy są krótkie
- Specjalizacja - wykonują tylko niezbędne, ściśle określone czynności
- Obecność „nieskończonej pętli" - w której wykonuje się główny nurt programu
- Optymalizacja kodu - pod względem rozmiaru zajętości pamięci, szybkości wykonywania
- Wykorzystanie przerwań do realizacji krytycznych czasowo procedur. Takie procedury powinny wykonywać się szybko, tak by niezwłocznie powrócić do głównego nurtu programu. Synchronizacja danych pomiędzy przerwaniami, a głównym programem - wymaga zastosowania specyfikatora **volatile**.
- Unikanie wykorzystania opóźnień - pomiar czasu realizowany za pomocą timer-ów.
- Unikanie wykorzystania dynamicznego przydziału pamięci, unikanie rekurencji - silnego wykorzystania stosu
- Ze względu na ograniczenia pamięci operacyjnej - unikanie przechowywania w niej dłuższych fragmentów niezmiennicznych np. ciągi znaków do wyświetlania na wyświetlaczu - możliwe jest posługiwanie się w takim przypadku wyłącznie pamięcią programu - np. pochodzące z avr-libc biblioteka <avr/pgmspace.h> albo łatwiejsza w użyciu, przeznaczona dla Arduino, biblioteka Flash.h

# Cechy programów wykonujących procedury pomiarowe

## Cechy programów wykonujących procedury pomiarowe

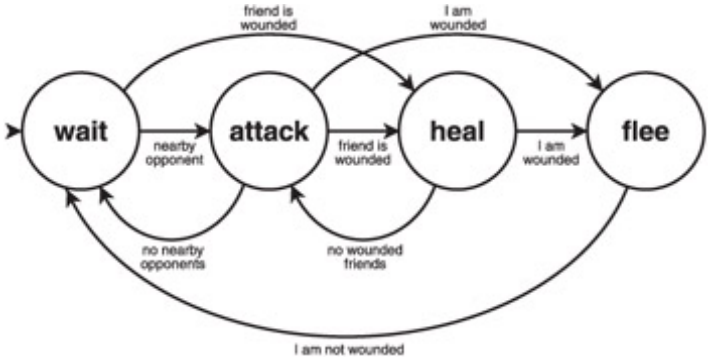
- Obecność buforów cyklicznych (circular buffer / ring buffer) - 2 indeksy - dane zapisują się pod jednym indeksem, natomiast odczyt następuje z drugiego indeksu. Taki bufor rozwiązuje problemy z synchronizacją zbierania, przetwarzania i wysyłania danych. Indeksy zawijają się na końcu bufora - tzn. wracają od początku tablicy, nadpisując dane.
- Wykorzystanie tzw. maszyny stanów

# Przykład bufora cyklicznego




Rysunek: Przykład bufora cyklicznego - np. proces „producenta" zapisuje dane wykorzystując indeks „Tail", a proces „konsumenta" odczytuje je wykorzystując indeks „Head". Bufor to zwykły liniowy obszar pamięci, w którym jeśli indeks osiągnię maksymalną wartość, to przeskakuje na pozycję 0.

# Przykład maszyny stanów




Rysunek: Przykład maszyny stanów opisujący zachowanie postaci w grze RPG. Graf zawiera stany - kółka i przejścia między stanami następują gdy spełniony jest określony warunek. Stan może opisywać pojedyncza zmienna np. „state"

## Proste zasady optymalizacji kodu

- **Ogranicz użycie zmiennych typu float** - jeśli jest to możliwe rób operacje na zmiennych typu int lub byte. (Zastosowanie tej zasady przyśpiesza wykonanie programu nawet 200 razy)
- **Ogranicz stosowanie funkcji matematycznych** - np. zamiast generować sinus przy użyciu funkcji sin, zastosuj tablicę z wartościami sinusa. (Taka zamiana przyśpiesza wykonanie programu nawet 10 razy)
- **Zamień wpisane wartości numeru pinu w funkcjach typu pinMode czy digitalWrite na stałe typu const byte** - taka zmiana zwiększa częstotliwość generowania sygnału nawet o ok 1kHz.
- **Jeśli nie potrzebujesz zmiennej 16bitowej (maksymalna wartość to 32 767 dla zmiennej ze znakiem) to zastąp ją zmienną typu byte** - taka zmiana zwiększa częstotliwość generowania sygnału nawet o ok 5kHz

## Zmniejszenie poboru prądu

- Wykonanie projektu korzystając z Arduino, który ma działać dzięki podłączonej baterii, wymaga rozpatrzenia kwestii poboru prądu. Bezcelowe byłoby urządzenie, które pracowałoby jedynie przez 4h...
- Istnieje szereg możliwości zmniejszenia poboru prądu. Jednym z kluczowych zagadnień jest częstotliwość taktowania zegara. W większości procesorów użytych w Arduino wynosi ona 16 MHz.
- Częstotliwość taktowania zegar może zostać zmieniona z poziomu szkicu korzystając z biblioteki *Prescaler*, którą należy pobrać z: playground.arduino.cc
- Konsekwencją zmiany częstotliwości taktowania zegara oprócz spowolnienia czasu wykonywania się szkicu jest niemożliwość skorzystania z funkcji: millis i delay. Te funkcje posiadają odpowiedniki w bibliotece Prescaler: trueMillis() i trueDelay().
- Dodatkową konsekwencją zmiany częstotliwości taktowania jest fakt, że PWM czy sterowanie serwomotorami nie będzie działać tak jak się spodziewamy (czynności zależne od czasu).

## Przykład skorzystania z biblioteki Prescaler

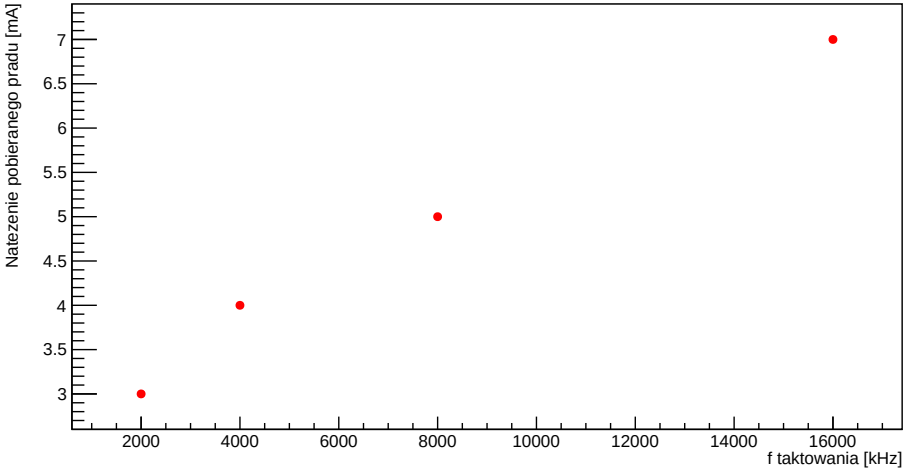
```
#include <Prescaler.h>

void setup()
{
setClockPrescaler(CLOCK_PRESCALER_256); //
    ↪ Zmniejszenie czestotliwosci 256razy
    ↪ czyli z 16MHz na 62,5kHz
}
```

## Dostępne wartości prescalera

- CLOCK_PRESCALER_1
- CLOCK_PRESCALER_2
- CLOCK_PRESCALER_4
- CLOCK_PRESCALER_8
- CLOCK_PRESCALER_16
- CLOCK_PRESCALER_32
- CLOCK_PRESCALER_64
- CLOCK_PRESCALER_128
- CLOCK_PRESCALER_256

# Natężenie pobieranego prądu vs. częstotliwość taktowania zegara



## Żywotność baterii vs. natężenie pobieranego prądu

- Żywotność podłączonej baterii (t) do arduino można wyznaczyć korzystając z:

$$t = \frac{Pojemnosc\ \ baterii(mAh)}{Natezenie\ \ pradu(mA)}$$

- Zakładając użycie typowej baterii litowo-jonowej o pojemności 2400mAh i napięciu 3.7V, należało by podłączyć dwie takie baterie szeregowo by uzyskać wymagane napięcia do zasilania arduino(7V-12V). Żywotność dwóch baterii przy wyborze danego prescalera wynosi:

| Wartość prescalera | Częstotliwość taktowania | Żywotność baterii [h] |
|---|---|---|
| 1 | 16MHz | 307 ok 12 dni |
| 2 | 8 MHz | 444 |
| 4 | 4 MHz | 600 |
| 8 | 2 MHz | 750 |
| 16 | 1 MHz | 923 |
| 32 | 500 kHz | 1043 |
| 64 | 250 kHz | 1090 |
| 128 | 125 kHz | 1142 |
| 256 | 62,5 kHz | 1142 ok 47 dni |

## Zmniejszenie poboru prądu

- Kolejnym działaniem umożliwiającym ograniczenie poboru prądu jest wylutowanie diody sygnalizującej podłączenie płytki Arduino.
- Niewielkie zmniejszenie natężenia pobieranego można uzyskać poprzez wyłączanie funkcji, których aktualnie nie potrzebujemy z poziomu szkicu. Do tego służy biblioteka: <avr/power.h>.
- Funkcje, które mogą okazać się pomocne:
  1. Włączanie/wyłączenia przetwornika A/C: power_adc_enable(), power_adc_disable().
  2. Włączanie/wyłączanie SPI: power_spi_enable(), power_spi_disable().
  3. Włączanie/wyłączanie I2C: power_twi_enable(), power_twi_disable().
  4. Włączenie/wyłączenie portu szeregowego: power_usart_enable(), power_usart_disable().
  5. Włączenie/wyłączenie wszystkich funkcji: power_all_enable(), power_all_disable().
- Ostatecznie można uśpić mikrokontroler na czas kiedy nie ma wykonywać jakiś operacji. Wybudzenie mikrokontrolera może się odbyć na dwa sposoby: poprzez przerwanie lub po upływie określonego czasu. W przypadku wybrania drugiej opcji przydatna okazuje się biblioteka: ***Narcoleptic***.

## Przykład zapalenia diody przez 1s i zgaszenie przez 10s

```
#define LED 10
void setup()
{
pinMode(LED,OUTPUT);
}
void loop()
{
digitalWrite(LED,HIGH);
delay(1000);
digitalWrite(LED,LOW);
delay(10000);
}
```

## Przykład powyższy ale z użyciem biblioteki Narcoleptic

Podczas wykonania szkicu z użyciem biblioteki Narcoleptic uzyskano zmniejszenie poboru prądu o ok **80%**.

```
#include <Narcoleptic.h>
#define LED 10
void setup()
{
pinMode(LED,OUTPUT);
}
void loop()
{
digitalWrite(LED,HIGH);
Narcoleptic.delay(1000);
digitalWrite(LED,LOW);
Narcoleptic.delay(10000);
}
```

## Budzenie arduino za pomocą zewnętrznego przerwania

Do wybudzania arduino korzystając z zewnętrznego przerwania służy biblioteka: <**avr/sleep.h**>. Zasadę działania demonstruje poniższy przykład, który ma na celu zapalanie diody przez 100ms po naciśnięciu przycisku:

```
#include <avr/sleep.h>
const int LED = 13;
volatile boolean stan=false;
void setup()
{
pinMode(LED, OUTPUT);
idzSpac();
}
void loop()
{
if(stan)
{
digitalWrite(LED,HIGH);
delay(100);
digitalWrite(LED,LOW);
stan=false;
idzSpac();
}
}
```

## Budzenie arduino za pomocą zewnętrznego przerwania

```
void idzSpac()
{
set_sleep_mode(SLEEP_MODE_PWR_DOWN); //Wybranie
    ↪ trybu uspienia
sleep_enable(); //Wlaczenie mozliwosci wywolania
    ↪ uspienia
attachInterrupt(0, ustawStan, LOW); //Konfiguracja
    ↪  przerwania
sleep_mode(); //Wywolanie uspienia do nastapienia
    ↪ przerwania
sleep_disable(); //Wylaczenie mozliwosci wywolania
    ↪  uspienia
detachInterrupt(0); //Wylaczenie mozliwosci
    ↪ wykonania przerwania typu int0
}

void ustawStan()
{
stan=true;
}
```

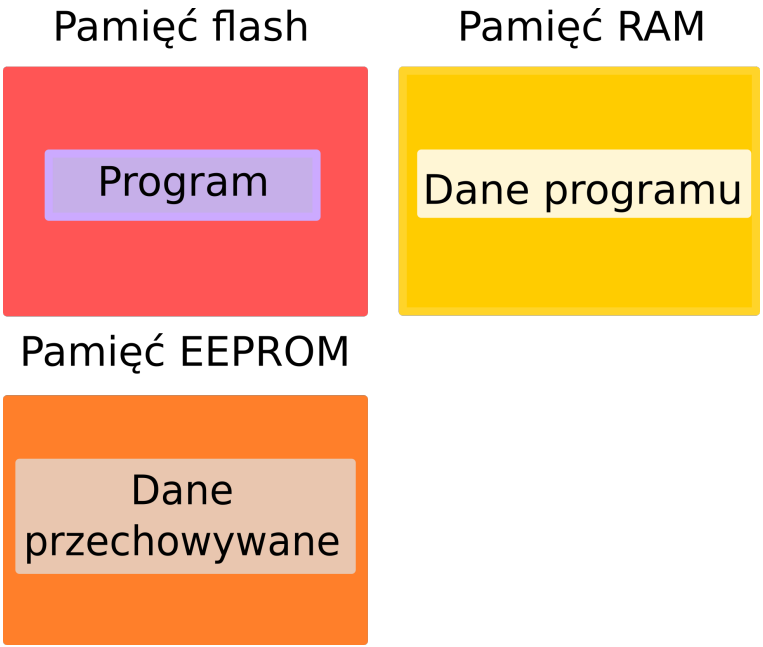
## Rodzaje trybów uśpienia

- SLEEP_MODE_IDLE - największy pobór prądu
- SLEEP_MODE_ADC
- SLEEP_MODE_PWR_SAVE
- SLEEP_MODE_STANDBY
- SLEEP_MODE_PWR_DOWN - najmniejszy pobór prądu

## Ograniczenie poboru prądu przez pin analogowy

Jeśli posiadamy czujnik, który wymaga obsługi przez pin analogowy to możemy zredukować pobór prądu poprzez podłączenie zasilania nie z 5V ale z pinu cyfrowego arduino. Tuż przed wykonaniem pomiaru należy na pin cyfrowy ustawić stan HIGH, który spowoduje podłączenie czujnik do zasilania. Następnie należy wykonać pomiar a po nim wysłać stan LOW na pin cyfrowy w ten sposób odłączając czujnik od zasilania.

# Rodzaje pamięci



## Pamięć flash

- Pamięć flash jest najbardziej pojemną pamięcią na płytce arduino (256 kB w przypadku arduino mega). Dla porównania pamięć RAM ma pojemność 8 kB w przypadku arduino mega.
- Zapisane na niej dane nie są ulotne, co kusiłoby do częstego zapisu danych właśnie do pamięci flash. W rzeczywistości trzeba zdawać sobie sprawę z kilku konsekwencji:
  1. Pamięć flash może być zapisana ok 10tyś razy, później może przestać działać prawidłowo.
  2. W pamięci flash zapisany jest kod programu, zatem należy uważać aby go nie nadpisać.
  3. W pamięci flash można jednorazowo zapisać jeden blok danych (64 bajty).
- Istnieją dwa sposoby do zapisu danych na pamięci flash. Pierwszy z nich umożliwia zapis jedynie łańcucha znaków:

```
Serial.println(F("Zapis do pamieci flash"));
```

- Drugi z nich bazuje na bibliotece <avr/pgmspace.h>.

## Pamięć flash - przykład

**Przykład:** Zapis i odczyt wartości do/z pamięci flash.

```
#include <avr/pgmspace.h>

PROGMEM int wartosci=[0,1,2,3,4]; //Przedrostek
    ↪ PROGMEM sprawi, ze tablica wartosci zostanie
    ↪  zapisana do pamieci flash

void setup()
{
Serial.begin(9600);
for(int i=0;i<5;i++)
{
int x=pgm_read_word(&value[i]); //Odczyt danych z
    ↪ pamieci flash (2 bajty), funkcja
    ↪ pgm_read_word przyjmuje jako argument adres
    ↪ a nie wartosc zmiennej stad &.
}
}
```

Inne możliwe funkcje do odczytu danych z pamięci flash:

- pgm_read_byte(adres) - umożliwia odczyt jednego bajta danych.
- pgm_read_dword(adres) - umożliwia odczyt czterech bajtów danych.

## Pamięć EEPROM

- Innym sposobem na przechowanie danych nawet po wyłączeniu zasilania jest skorzystanie z pamięci EEPROM. W przypadku Arduino Mega dysponujemy 4 kB pamięci EEPROM.
- Aby zapisać dane do pamięci EEPROM należy skorzystać z biblioteki EEPROM.h dołączonej do środowiska IDE.
- Przykład zapisu i odczytu jednego bajta danych do pamięci EEPROM:

```
#include <EEPROM.h>
byte wartosc=120;

void setup()
{
EEPROM.write(0,wartosc); //Pierwszy
    ↪ argument to adres komorki
    ↪ pamieci EEPROM gdzie wartosc ma
    ↪ byc zapisana
}

void loop()
{
byte odczytanaWartosc=EEPROM.read(0);
    ↪ //Jako argument funkcji podajemy
    ↪  adres komorki EEPROM, z ktorej
    ↪ chcemy odczytac dane.
}
```

# Przetworniki C/A

## Przetworniki C/A

- Przetwornik C/A - to urządzenie, które na swoim wyjściu wytwarza napięcie albo prąd proporcjonalny do sterującego przetwornikiem sygnału cyfrowego.
- Przetworniki C/A można zbudować w różnej technologii. Z czego najpopularniejsze to dzielniki rezystorowe, zbudowane w oparciu o tzw. drabinkę R-2R.
- Najważniejsze parametry przetwornika to jego rozdzielczość oraz czas ustalania się napięcia. N-bitowy DAC umożliwia wystawienie $2^N$ wartości z zakresu $0...2^N$-1. Parametr **dokładność** - określa maksymalną różnicę pomiędzy rzeczywistym napięciem wyjściowym a teoretycznym, parametr **czas ustalania** - to czas, po którym napięcie osiąga zakładaną wartość
- Parametr **liniowość** - określa maksymalną różnicę pomiędzy napięciem wyjściem a liniową interpolacją pomiędzy minimum a maksimum. Zwykle na poziomie $1/2^N$.
- Parametr **monotoniczność** - określa czy wraz z rosnącym kodem, zwiększa się także parametr wyjściowy, np. napięcie. Zwykle jest to spełnione.

# Przetworniki C/A I

Parametry występujące w notach katalogowych:

- **Resolution** - rozdzielczość, np. 12-bit, oznacza, że mamy $2^{12}$ wartości, tzn. zakres 0 – 4095 (0x000 – 0xfff).
- **LSB** - najmniej znaczący bit tzn. idealna różnica napięć między dwoma sąsiednimi kodami.

$$LSB_{ideal} = \frac{V_{ref}}{2^N}$$

gdzie: $V_{ref}$ - napięcie referencyjne - często równe napięciu zasilania przetwornika, $N$ - ilość bitów przetwornika.

- **Integral Nonlinearity (INL)** - całkowita nieliniowość - maksymalna różnica pomiędzy rzeczywistą wartością napięcia, a idealną wartością dla danego kodu.

$$INL = \frac{V_{out} - V_{ideal}}{LSB}$$

gdzie: $V_{ideal}$ = kod * LSB, $V_{out}$ - napięcie zmierzone na wyjściu przetwornika

# Przetworniki C/A II

- **Differential Nonlinearity (DNL)** - różnicowa nieliniowość - maksymalna różnica wartości pomiędzy sąsiednimi kodami.

$$DNL = \frac{\Delta V - LSB}{LSB}$$

gdzie $\Delta V$ - róznica pomiędzy napięciami zmierzonymi dla dwóch sąsiednich kodów.

- **Offset error** - błąd przesunięcia - dewiacji. Wartość napięcia pojawiająca się dla kodu 0.
- **Gain error** - błąd nachylenia - różnica pomiędzy wartością rzeczywistego napięcia a idealnym napięciem dla najwyższego kodu, po wyeliminowanie innych błędów takich jak błąd przesunięcia. Zwykle podawany jako procent FSR (Full Scale Range) - pełnego zakresu wartości, albo w wartościach LSB. Tego typu błąd można skalibrować w oprogramowaniu.
- **Full Scale Error (FSE)** - Błąd pełnego zakresu - jest sumą błędu przesunięcia i błędu nachylenia. Jest to różnica wartości rzeczywistej napięcia i idealnej dla najwyższego kodu.

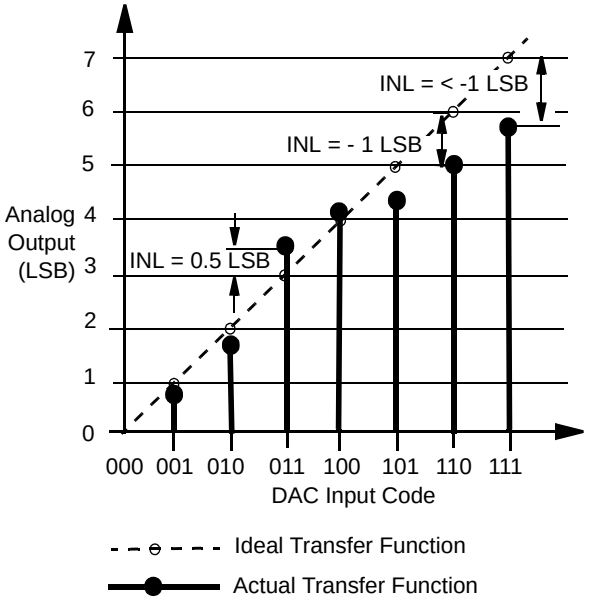
$$FSE = \frac{V_{out} - V_{ideal}}{LSB}$$

gdzie: $V_{ideal} = (V_{ref}) \cdot \left(1 - 2^{-N}\right) - V_{offset}$

# Przetworniki C/A III

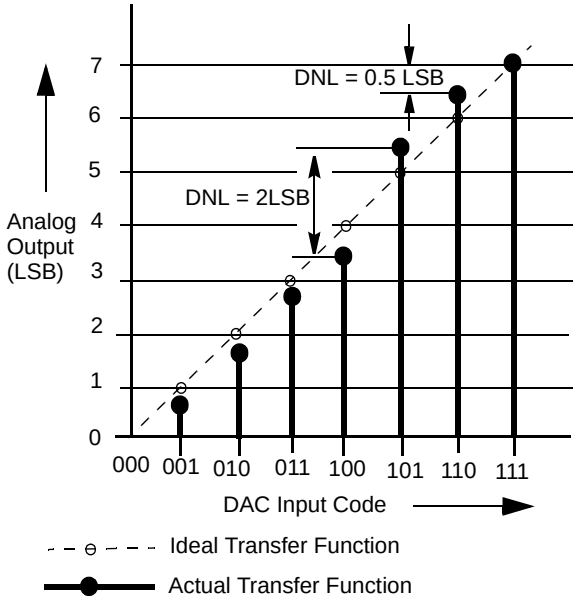
- **Gain Error Drift** - Temperaturowy dryft błędu nachylenia. Wyrażany w $ppm/^oC$
- **Offset Error Drift** - Temperaturowy dryft błędu przesunięcia. Wyrażany w $ppm/^oC$
- **Settling Time** - Czas ustalania się zadanej wartości napięcia, w zasięgu 0,5LSB.
- **Major-Code Transition Glitch** - jest to energia impulsu szpilkowego wtrąconego na wyjście przetwornika, wyrażona jako napięcie * czas trwania (nV * sec) przy przejściu o 1 LSB z kodu o pełnej zmianie bitów tzn. z np. 0b0111 11111111 na 0b1000 00000000.
- **Digital Feedthrough** - jest to impuls szpilkowy wtrącony na wyjście przetwornika, wyrażony w nV*sec, przy pełnej zmianie na liniach cyfrowych tj. z 0b0000 00000000 na 0b1111 11111111.

# Przetworniki C/A



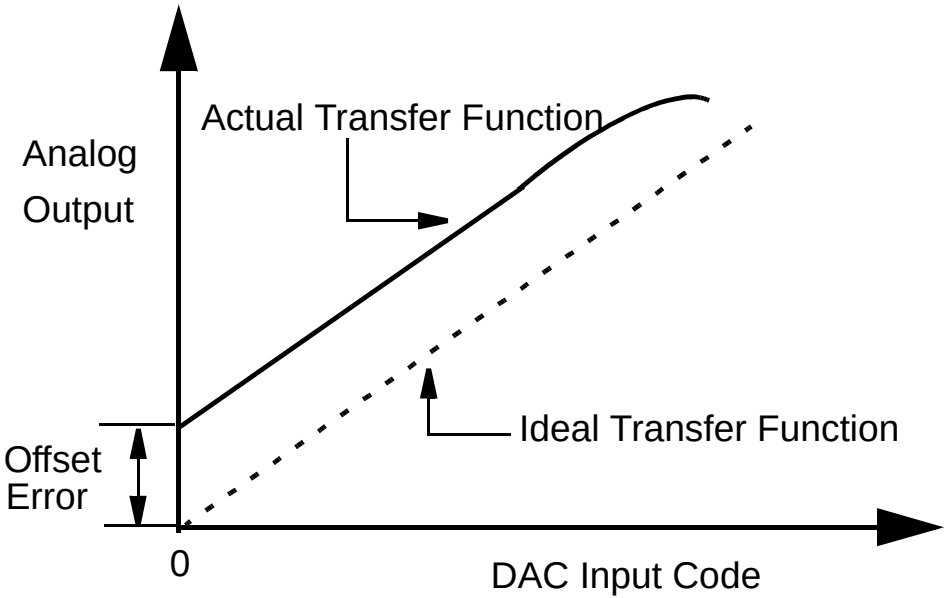

Rysunek: INL - całkowita nieliniowość

# Przetworniki C/A



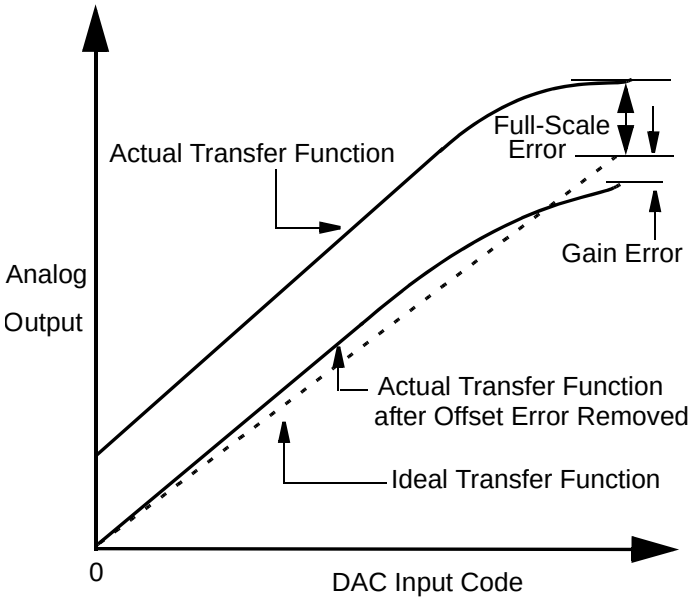

Rysunek: DNL - różnicowa nieliniowość

# Przetworniki C/A




Rysunek: Offset error

# Przetworniki C/A




Rysunek: Błąd nachylenia oraz błąd pełnego zakresu

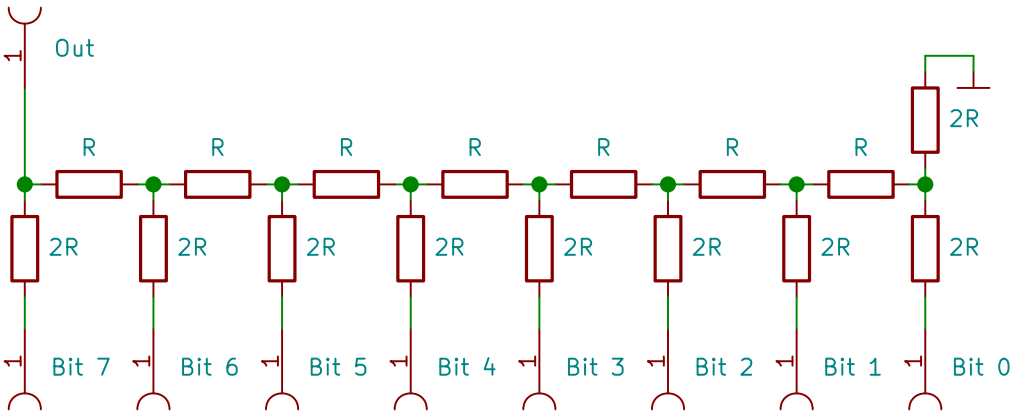
# Przetwornik C/A

## Rodzaje przetworników

- Współczesne przetworniki są wykonane jako monolityczne układy scalone (IC - integrated circuits)
- W specjalnych zastosowaniach (zwykle starsze rozwiązania), stosowane są układy hybrydowe - zawierające układy scalone i zestawy elementów dyskretnych, które mogą być swobodnie skalibrowane. Obudowa jest zbliżona rozmiarami do układów scalonych
- Stosuje się także moduły DAC zbudowane z elementów dyskretnych.

Układy hybrydowe i moduły z elementów dyskretnych stosowane są zwykle w rozwiązaniach specjalnych np. dla uzyskania wysokiej precyzji.

# Drabinka R-2R



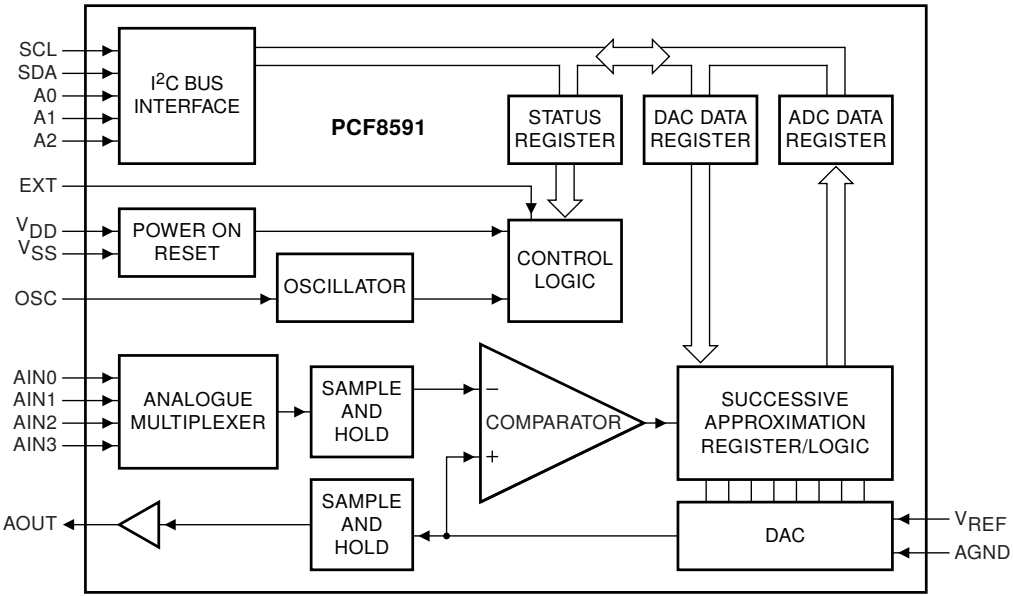

Rysunek: Zasada działania drabinki rezystorowej R-2R polega na tym, że każdy z bitów będzie miał proporcjonalne do swojej wagi wpływ na napięcie wyjściowe. Najbardziej znaczący bit (7) (MSB - Most Significant Bit) ma największy wpływ, najmniej znaczący bit (0) (LSB - Least Significant Bit) ma najmniejszy wpływ.

## Rodzaje przetworników A/C

Przetworniki A/C - to urządzenia, które zamieniają wielkość elektryczną (np. napięcie) na wartość kodową. Przetworniki A/C podobnie jak C/A charakteryzuje rozdzielczość oraz prędkość konwersji. Różne typy przetworników:

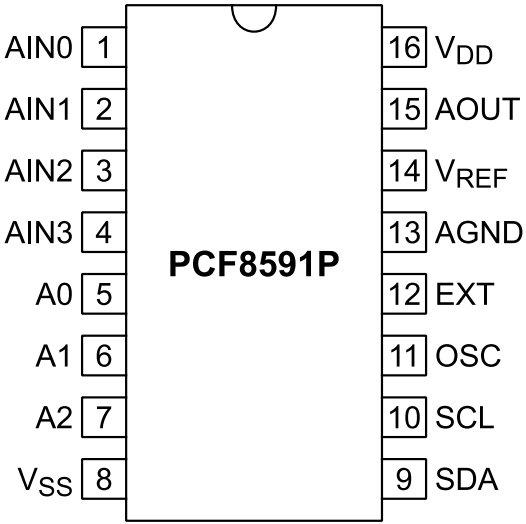
- Najprostszy przetwornik to komparator - porównuje napięcie wejściowe z napięciem referencyjnym
- Przetworniki typu FLASH - zbudowane z wielu komparatorów - odpowiadających każdej wartości kodowej - bardzo szybkie, ale kosztowne w budowie - wymagają dużo elementów.
- Przetworniki z sukcesywną aproksymacją - wykorzystują przetwornik C/A do wytwarzania napięcia porównawczego, które porównują z napięciem wejściowym. Wymaga $2^N$ porównań, albo przy zastosowaniu wyszukiwania metodą drzewa binarnego N porównań. Jest wolniejszy od typu FLASH, ale mniej kosztowny, przez co bardziej rozpowszechniony.
- Przetworniki z pojedynczym i podwójnym całkowaniem - polegające na ładowaniu kondensatora do określonego napięcia w pierwszym przypadku i pomiar czasu, drugi przypadek, to ładowanie kondensatora przez ściśle określony czas, a następnie rozładowanie do pełnego rozładowania z pomiarem czasu.
- Przetworniki Delta-Sigma, wykorzystujące modulację sygnału Delta-Sigma - otrzymywany sygnał gęstości impulsów. Obecnie przetworniki wykonane w tej technologii umożliwiają uzyskiwanie bardzo wysokich rozdzielczości przetwarzania.

# Przetwornik PCF8591 - 8bit A/C i C/A




Rysunek: Schemat blokowy układu PCF8591

# Przetwornik PCF8591 - 8bit A/C i C/A



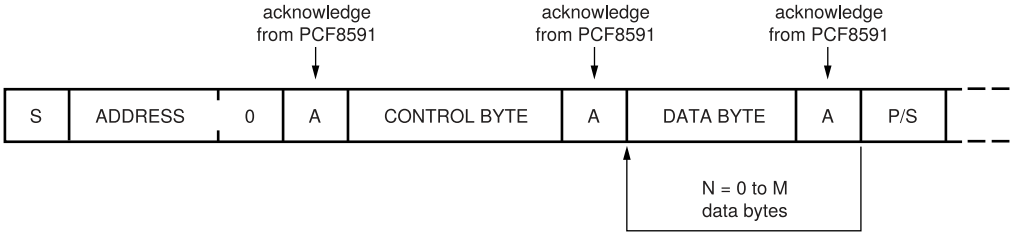

Rysunek: Widok obudowy i wyprowadzeń układu PCF8591

# Przetwornik PCF8591 - 8bit A/C i C/A

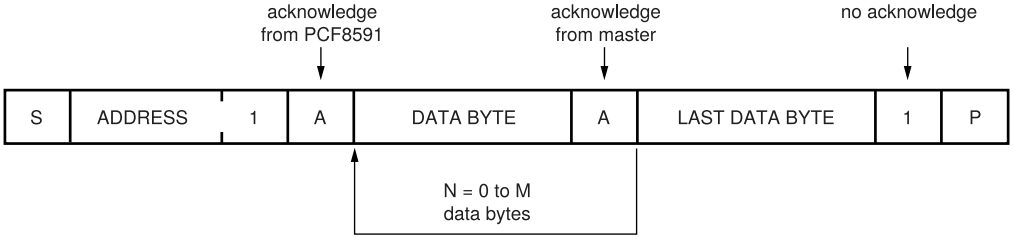
| Bit | Slave address | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **7** **MSB** | **6** | **5** | **4** | **3** | **2** | **1** | **0** **LSB** |
| slave address | 1 | 0 | 0 | 1 | A2 | A1 | A0 | R/$\overline{W}$ |

Rysunek: Adres $I^2C$ układu PCF8591

# Przetwornik PCF8591 - 8bit A/C i C/A

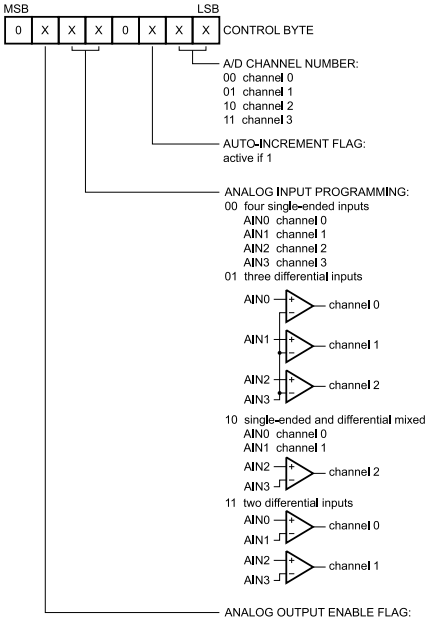



Rysunek: PCF8591 - zapis przez $I^2C$



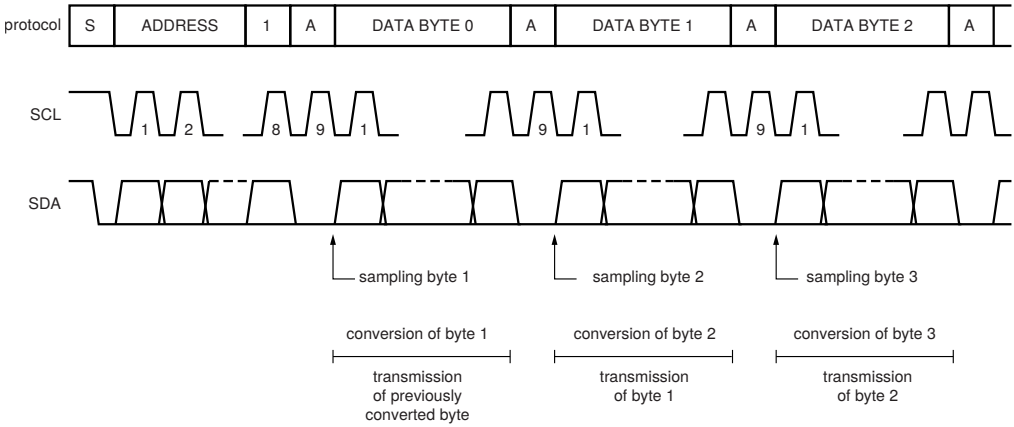

Rysunek: PCF8591 - odczyt przez $I^2C$

# Przetwornik PCF8591 - 8bit A/C i C/A



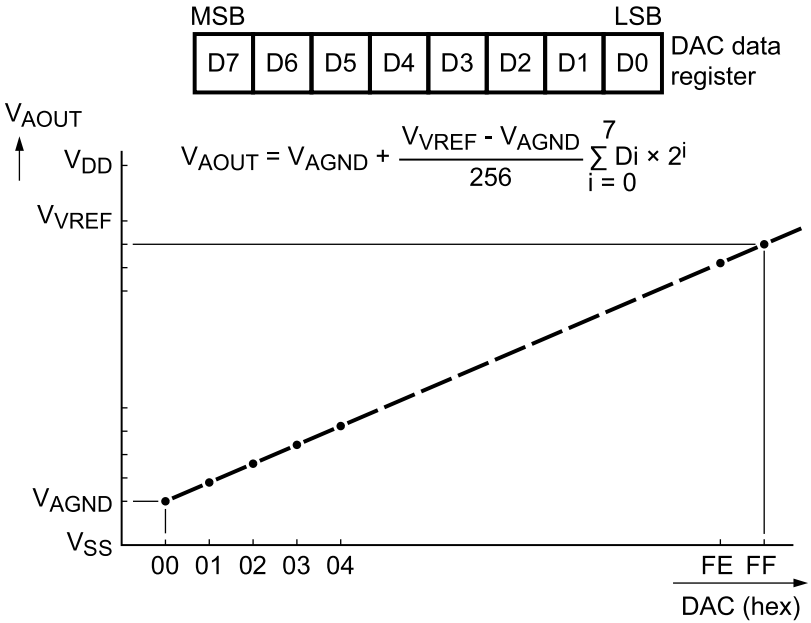

Rysunek: Bajt kontrolny układu PCF8591

# Przetwornik PCF8591 - 8bit A/C i C/A



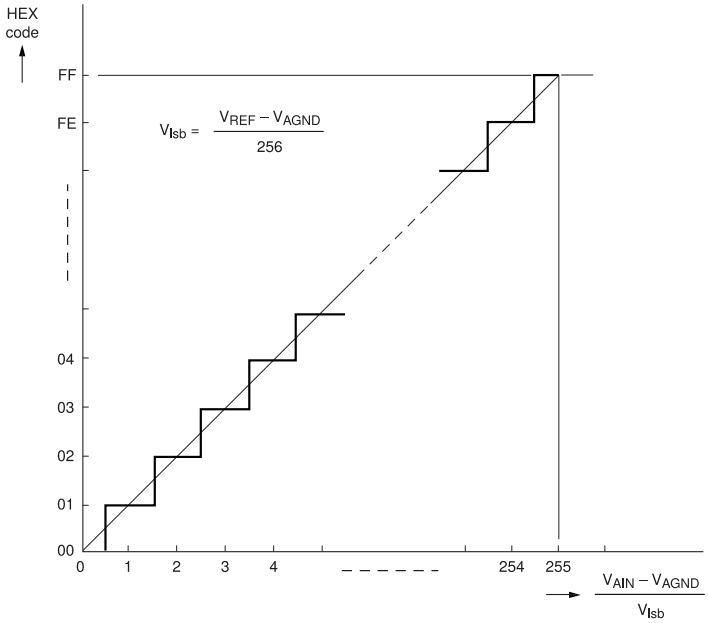

Rysunek: Transmisja danych w trakcie przetwarzania A/C z układu PCF8591 poprzez $I^2C$

# Przetwornik PCF8591 - 8bit A/C i C/A



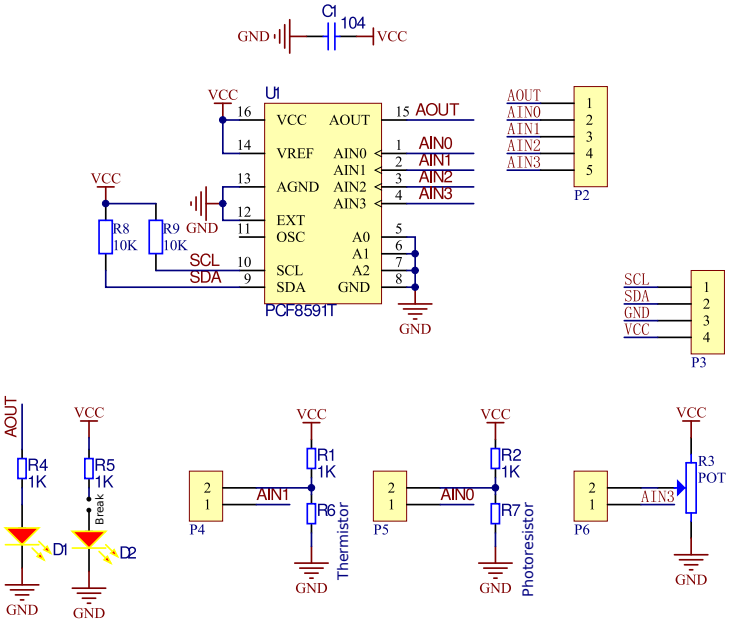

Rysunek: Konwersja C/A

# Przetwornik PCF8591 - 8bit A/C i C/A



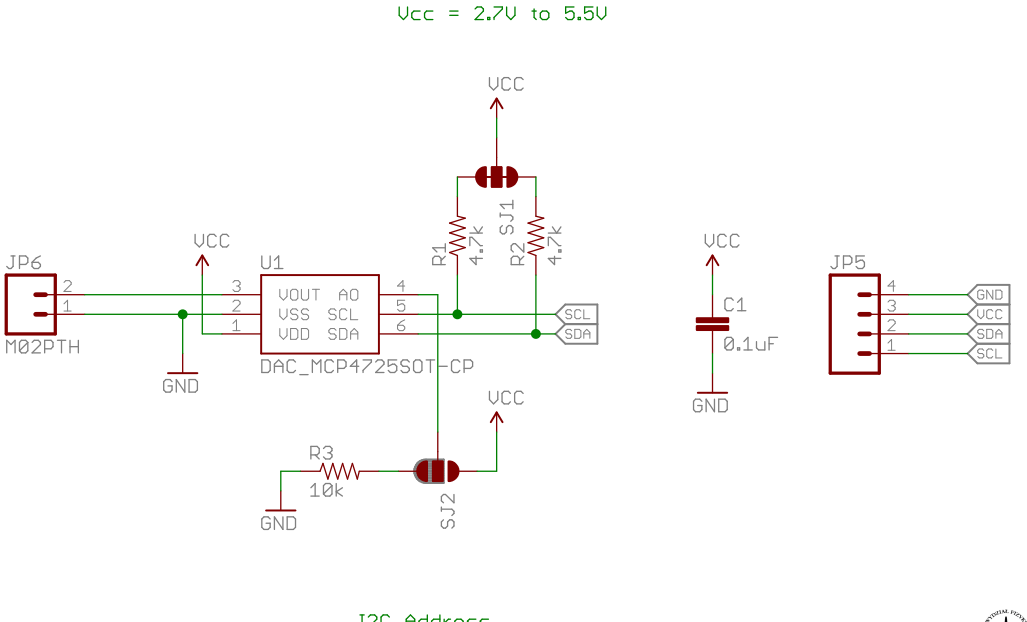

Rysunek: Konwersja A/C

# Przetwornik PCF8591 - 8bit A/C i C/A



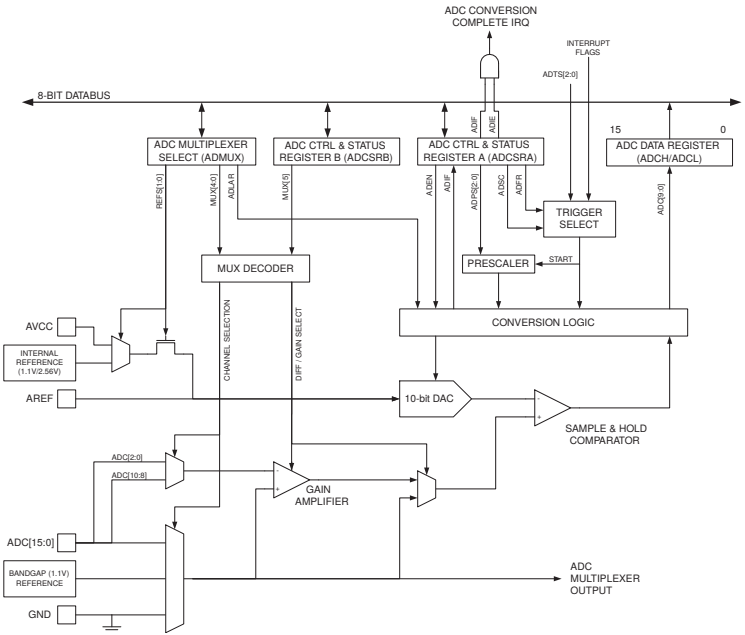

Rysunek: Schemat modułu z PCF8591 wykorzystywanego w ćwiczeniu laboratoryjnym

# Przetwornik MCP4725 12bit C/A




Rysunek: Schemat modułu z przetwornikiem C/A MCP4725.

# Przetwornik C/A wbudowany w ATmega2560 - 10bit A/C




Rysunek: Schemat blokowy przetwornika wbudowanego w mikrokontroler ATmega2560

THAT'S ALL FOR TODAY....
ANY QUESTION??