Wroclaw University of Science and Technology

Electronics Guidebook

Maciej Mielcarski

15.04.2018 issue 1.0

Spis treści

1 Introduction

The main use of the present document is to provide compact and convenient help in form of quick descriptions, schematics, code and IC pinouts in order to simplify work with electronics. This guidebook is in constant development, any suggestions or comments can be sent to mielcarski.m@gmail.com

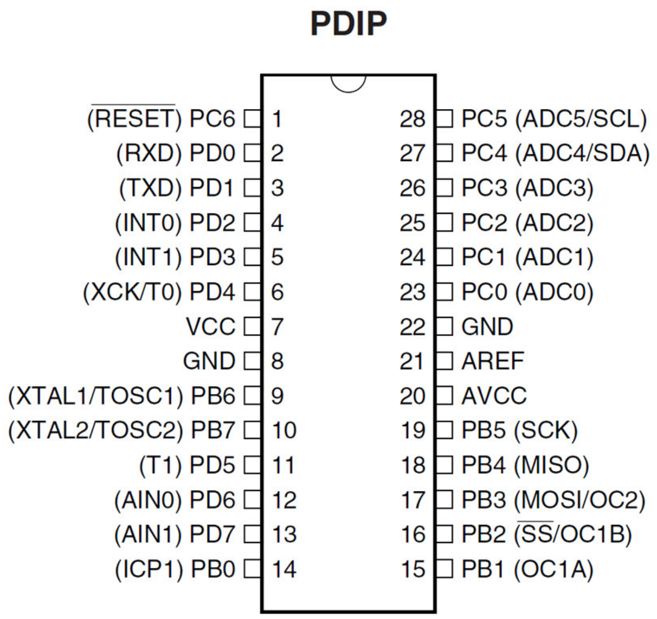
2 Atmel AVR

2.1 ATmega models

Tabela 1: AtMega MCU comparison

uC name	Clock	Flash	EEPROM	SRAM	8-bit TIM	16-bit TIM	PWM	10-bit ADC	GPIO
ATmega 8 (DIP)	16MHz	8KB	512B	1KB	2	1	3	6	23
ATmega 32 (DIP)	16MHz	32KB	1024B	2KB	2	1	4	8	32
ATmega 328 (DIP)	20MHz	32KB	1024B	2KB	2	1	4	6	23

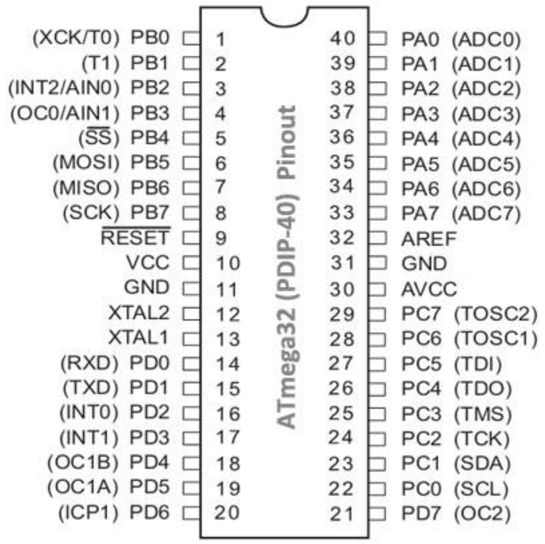
2.2 ATmega 8



Rysunek 1: Atmega 8 wyprowadzenia

2.3 Atmega 32

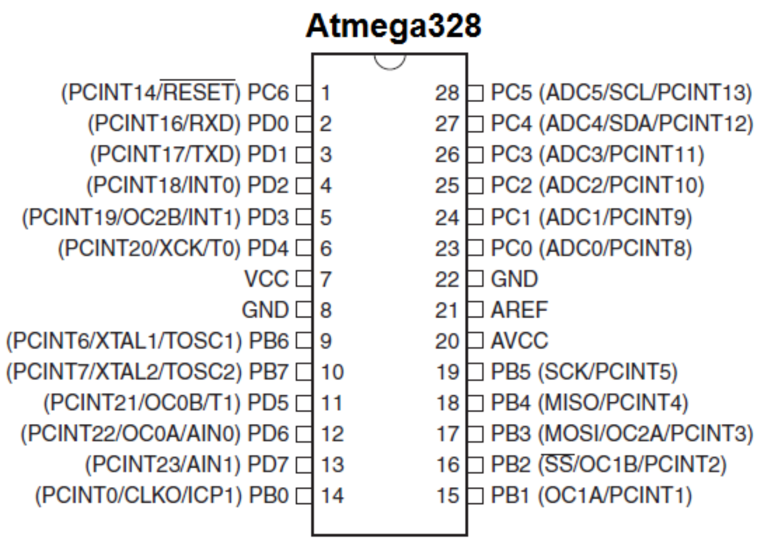
atmega 32 lorem ipsum



Rysunek 2: Atmega 32 wyprowadzenia

2.4 Atmega 328

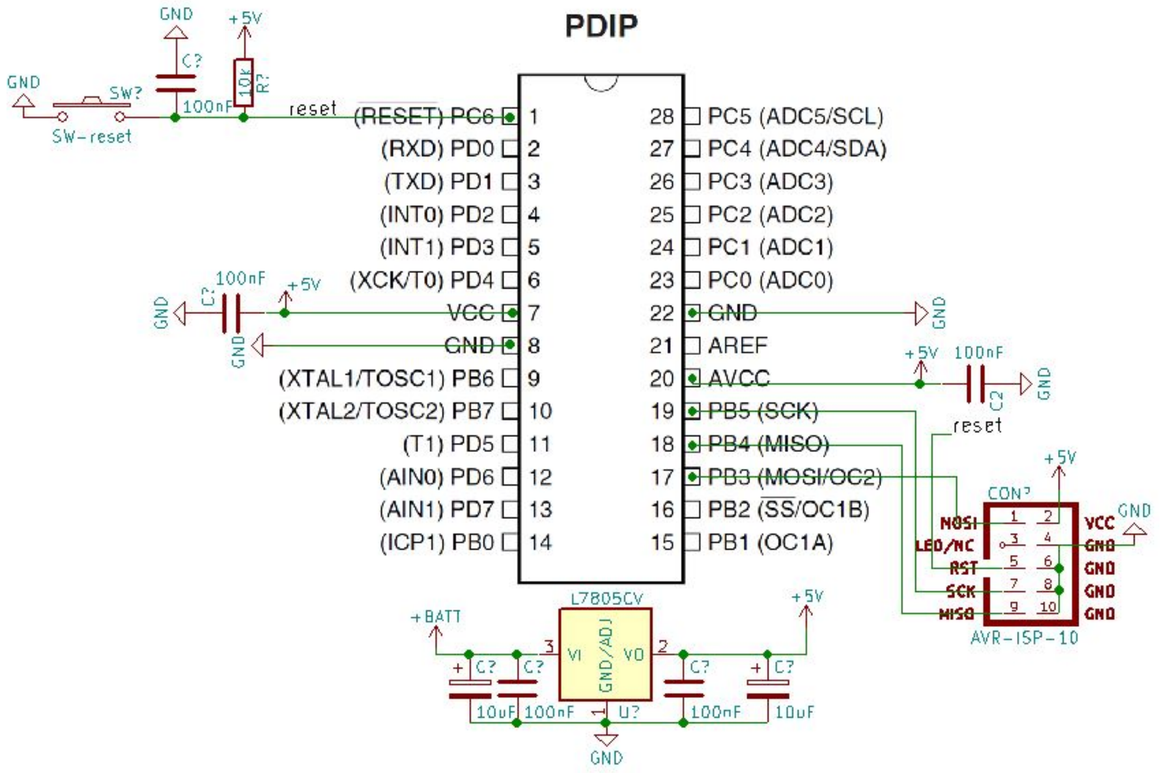
atmega 328 lorem ipsum



Rysunek 3: Atmega 328 wyprowadzenia

2.5 Zasilanie

zasilanie lorem ipsum



Rysunek 4: Atmega 8 zasilanie + programator

3 ADC

ADC converter of a n-bit resolution is capable of converting analog signal into digital values between 0 and 2^n - 1. In order to enable ADC conversion in AVR, we need to set the CPU clock prescaler and set the enable bit (according to the datasheet). For example, for atmega 328 uC, initialization of the registers is performed by this code:

```
ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // Set ADC prescalar to 128
ADMUX |= (1 << REFS0);    // Set ADC reference to AVCC
ADCSRA |= (1 << ADEN);  // Enable ADC
```

ADC single conversion function:

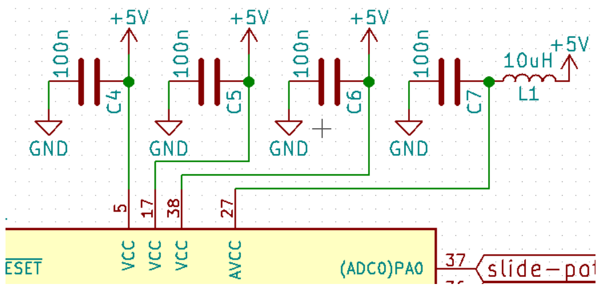
```
uint16_t ADC_read(uint8_t channel)
{
   channel &= 0x07;               // AND operation with 7 (will keep channel between 0-7)
   ADMUX = (ADMUX & 0xF8) | channel;    // clears 3 first bits before OR

   ADCSRA |= (1 << ADSC);             // start single convesrion
   while(ADCSRA & (1 << ADSC));       // wait for conversion to complete
   return ADCW;
}
```

3.1 Filtering Noise

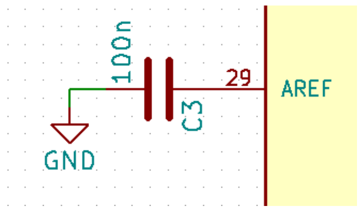
Even though filtering all of the noise from ADC reading is impossible, there are certain measures to minimize this unwanted effect.

3.1.1 Proper power supply connection

In most cases, voltage reference is preferably set to internal Vcc, which requires a **LC low-pass filter** on AVcc power supply pin and **100nF capacitor to GND on Vref** pin:



Rysunek 5: Atmega 32 AVcc LC filter



Rysunek 6: Atmega 32 Aref filtering

3.1.2 Lowering ADC resolution

If steadiness of ADC reading is more important for us than high resolution (eg. MIDI Control Change messages), lowering it will result in huge boost in suppresing oscillations in readings. It is simply done by **binary shifting** read ADC value. In order to **decrease** ADC resolution from **10 bit** to **7 bit**:

```
int adc_10bit_reading = ADC_read(channel);
int adc_7bit_reading = (adc_10bit_reading>>3);
```

3.1.3 Averaging readings

A batch of averaged readings may be more useful than single one. It is prefered for the number of averaged readings to be chosen as a power of 2 in order to minimize uC's computing power. Averaging **16** readings:

```
int adc_sum = 0;
   for(int i=0;i<16;i++)
   {
      adc_sum += ADC_read(channel);
   }
int adc_reading = adc_sum/16;
```

3.1.4 Exponential moving average (EMA)

If our ADC reading is resisting all of the conventional methods, it may be neccesery to implement additional averaging algorithm.

```
const float EMA_a = 0.6;
int EMA = 0;
int adc_reading = 0, adc_prev_reading = 0;

adc_prev_reading = adc_reading;
adc_reading = ADC_read(channel);

EMA = (EMA_a*adc_reading) + (1-EMA_a)*adc_prev_reading;
```

4 USART

Before initializing USART communication, it is required to define **CPU frequency** and desired **baud rate**, which is needed to calculate **UBRR** value, which initialises USART communication peripherals to match said frequency and baud rate:

$$MYUBRR = FOSC/16/BAUD\text{-}1$$

USART registers initialization (atmega 328):

```
UBRR0H = (unsigned char)(ubrr>>8);   // set baud rate to 9600
UBRR0L = (unsigned char)ubrr;      //
UCSR0B = (1<<RXEN0)|(1<<TXEN0);      // Enable receiver and transmitter
UCSR0C = (1<<USBS0)|(3<<UCSZ00);      // Set frame format: 8data, 2stop bit
```

Basic USART data manipulation:

```
void uart_putchar(char c)
{
   while ( !(UCSR0A & (1<<UDRE0)) )   // Wait for empty transmit buffer
   ;
    UDR0 = c;                        // Put data into buffer, sends the data
}

char uart_getchar(void) {
    loop_until_bit_is_set(UCSR0A, RXC0);   // Wait until data exists
    return UDR0;
}

void uart_putstring(char tab[])
{
   int i =0;
   while (( UCSR0A & (1<<UDRE0))  == 0){};
        while (tab[i] != 0x00)
      {
            uart_putchar(tab[i]);
         i++;
        }
}

void uart_putint(int value)
{
   char tab[16];
   itoa(value,tab,10);
   uart_putstring(tab);
}
```

In oreder to communicate with computer using USART, proper **USB-RS232 Adapter** is needed.

5 Git

5.1 Creating local and remote repository

git init - initialize local repository in curren directory
git remote add origin <link.git> - add remote repository
git config - -global user.email <email adress> - define user email
git config - -global user.name <name> - define user name

5.2 Repository status

git status - list tracked/untracked files
git branch - list all branches
git log - -diff-filter=D –summary - list all deleted files
git remote -v - list all defined remote repositories

5.3 Recovery

git checkout <commit>∧ - - <file> - recovering deleted files

5.4 Commit and push

git add . - add all of the untracked files
git commit -m "commit description" - commit all changes with label
git push -u origin master - push current master branch to origin remote repository

6 Makefile

6.1 Szablony programów

6.1.1 Kod bazowy

```
#define F_CPU 1000000UL
#include <stdio.h>
#include <avr/io.h>
#include <util/delay.h>

int main()
{
   while(1)
   {

   }

   cout<<"hello world"<<endl;
   return 0;
}
```

6.1.2 Obsługa przycisku monostabilnego

```
int isButton()              //checks if encoder button is pressed
{
   if(!(PINA & 0x08))
      return 1;
   else
      return 0;
}
```

6.1.3 migająca dioda

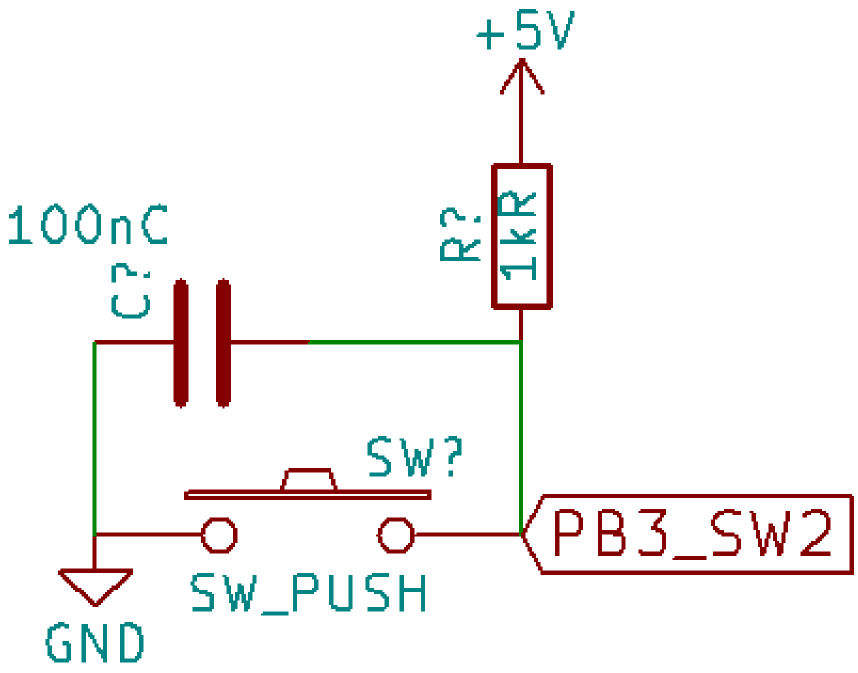
```
void blinkLed(int on, int off)    // turn on the led for
{                                //'on' and off for 'off' milisecondds
   PORTC |= (1<<LED);
   delay_ms(on);
   PORTC &= ~(1<<LED);
   delay_ms(off);
}
```

6.1.4 Obsługa enkodera obrotowego

```
DDRA &=~ (1 << ENC_A);      // encoder pins as input
DDRA &=~ (1 << ENC_B);      //
PORTA |= (1 << ENC_B)      // with pull-up enabled
      |(1 << ENC_A);      //

uint8_t readEncoder(void)   //reads the gray code from encoder
{
 uint8_t val=0;

  if(!bit_is_clear(PINA, ENC_B))
   val |= (1<<1);

  if(!bit_is_clear(PINA, ENC_A))
   val |= (1<<0);

  return val;
}

void readEncoderCounter ()      //monitoring encoder counter
{                         //increase or decrease
   uint8_t val_tmp = 0;
   val_tmp = readEncoder();

   if(val != val_tmp)
   {
      if((val==3 && val_tmp==1))
         {
         encoderCount ++;   //clockwise turn
         }
      else if((val==2 && val_tmp==0))
         {
         encoderCount --;   //counter-clockwise turn
         }
      val = val_tmp;
      }
   delay_ms(1);
}
```

7 Podłączenie elementów wykonawczych

7.1 Eliminacja drgań styków - przycisk monostabilny

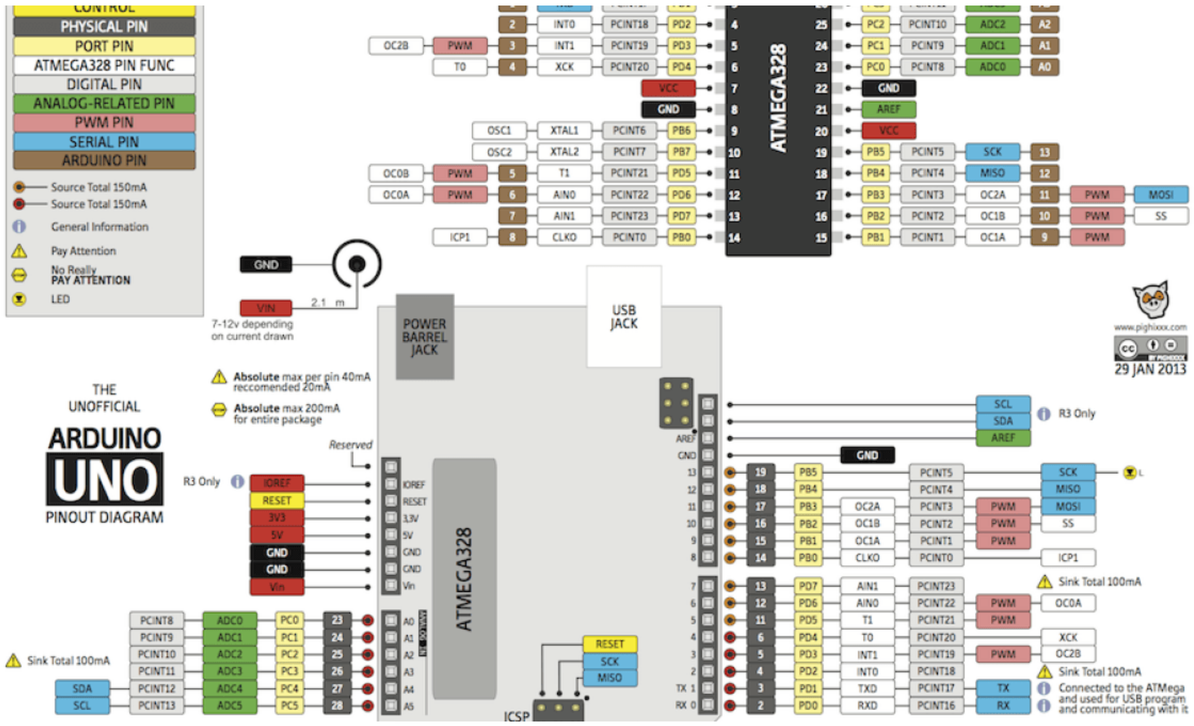


Rysunek 7: kompensacja drgań styków przycisku monostabilnego

8 Arduino

arduino lorem ipsum

8.1 Arduino UNO



Rysunek 8: arduino uno i atmega 328 - wyprowadzenia

- Arduino UNO:
- mikrokontroler: ATmega328P
- napięcie pracy: 5V
- napięcie zasilania: 6-20V
- wyjścia/wejścia cyfrowe: 14 (w tym 6 PWM)
- wejścia analogowe: 6
- wydajność prądowa pinu: 20mA
- pamięć Flash: 32 KB
- pamięć SRAM: 2KB
- pamięć EEPROM: 1KB
- zegar: 16 MHz
- wymiary: 68.6 x 53.4 mm, 25 g

8.2 Obsługa bibliotek

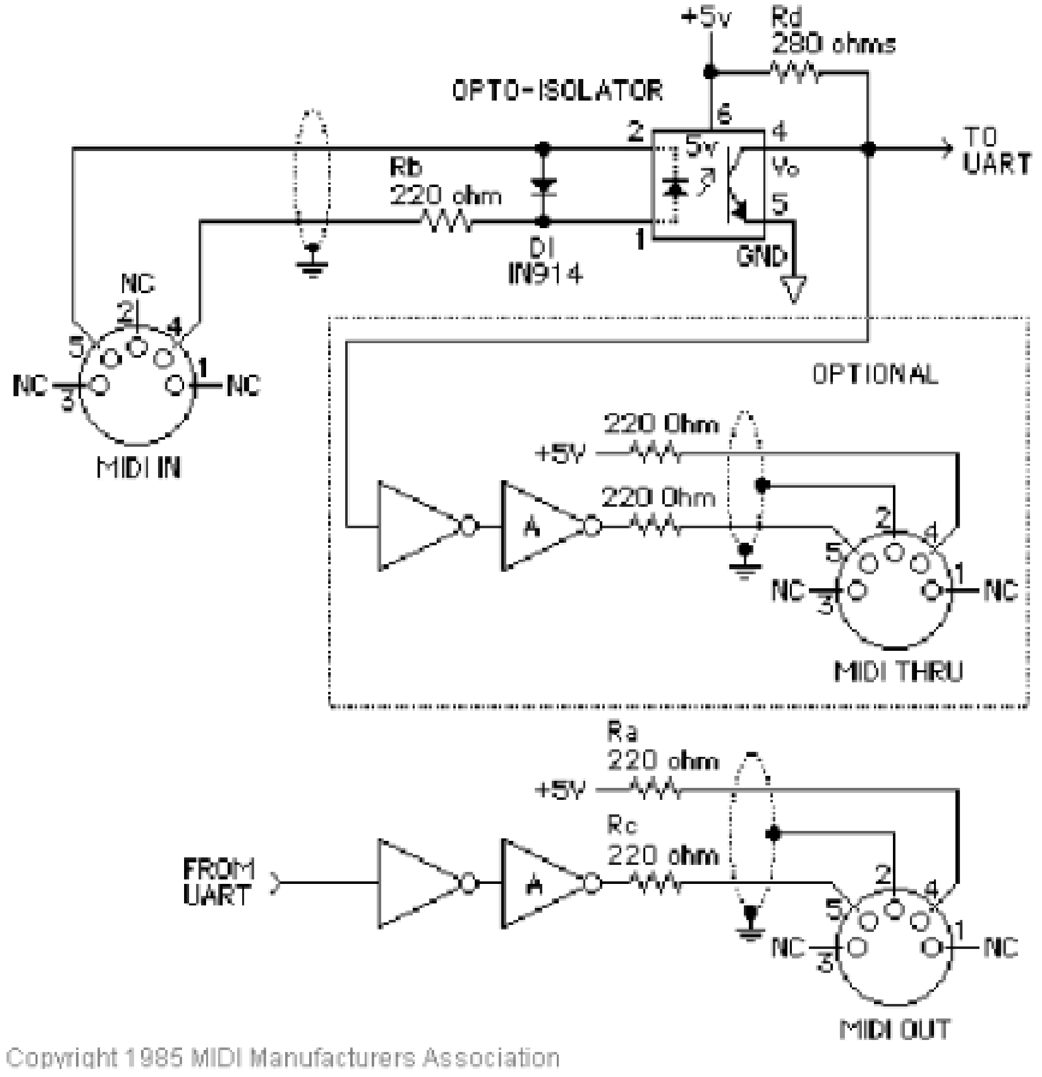
8.2.1 Podstawowe operacje na pinach

Biblioteka implementacji funkcji C++ : ¡Wire.h¿
Obsługa pinów cyfrowych na przykładzie pinu nr 5:

pinmode: pinMode(x, OUTPUT)
pullup pinMode(x, INPUT_PULLUP)
ustawianie stanu: digitalWrite(x,HIGH)
czytanie stanu: digitalRead(button)
opóźnienie: delay(50)

9 MIDI

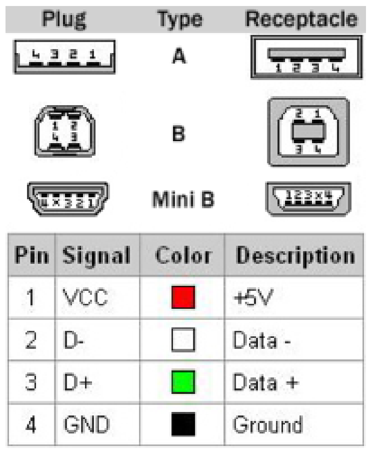
9.1 Podłączenie



Rysunek 9: Prawidłowe podłączenie wejścia, przejścia i wyjścia MIDI według MIDI.org

10 Konektory

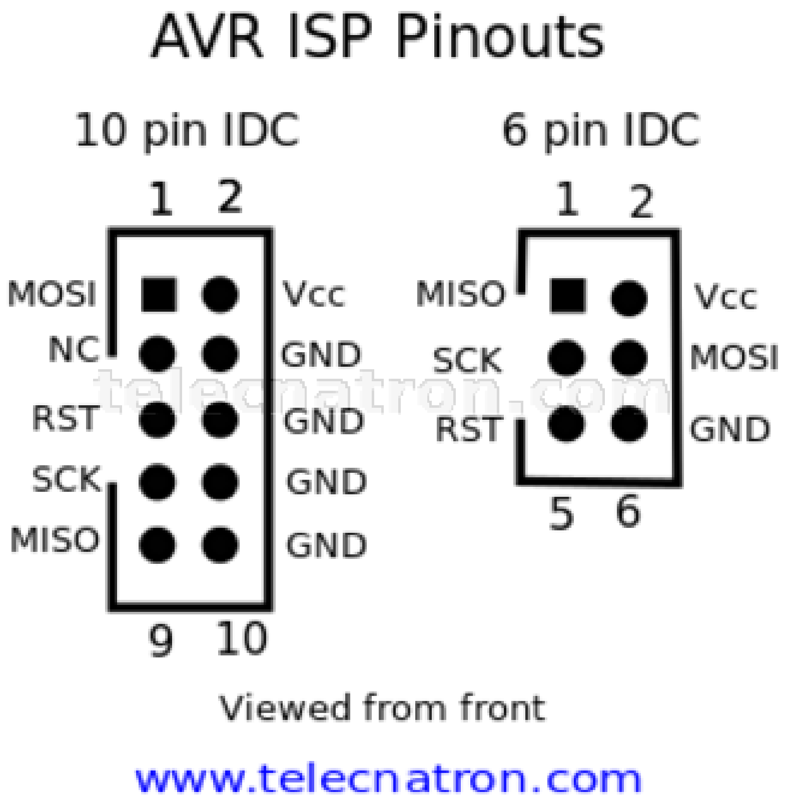
10.1 USB



Pin	Signal	Color	Description
1	VCC	🟥	+5V
2	D-	⬜	Data -
3	D+	🟩	Data +
4	GND	⬛	Ground

Rysunek 10: USB wyprowadzenia

10.2 AVR-ISP



Rysunek 11: AVR-ISP wyprowadzenia