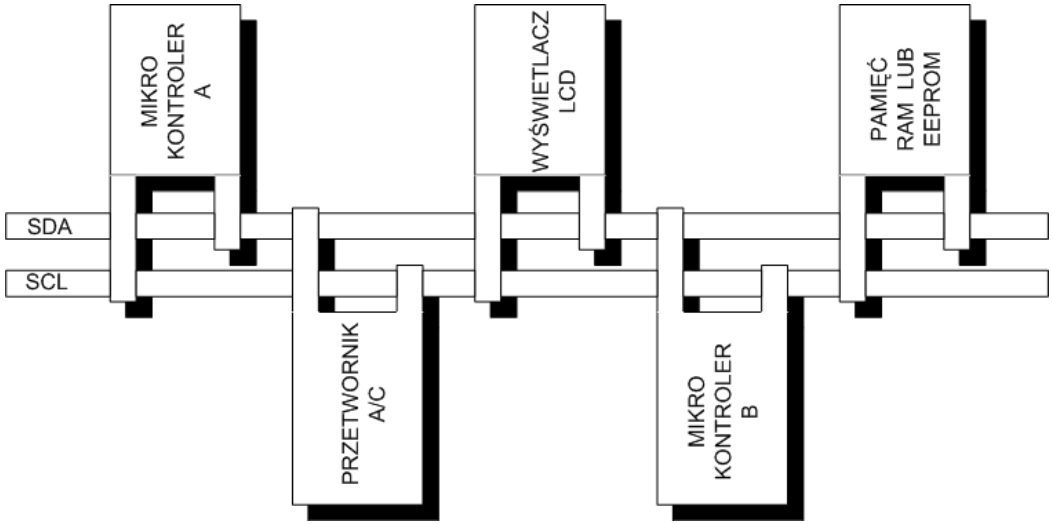
Magistrala I2C (TWI – Two Wire Interface)



Interfejs I2C ma dwie linie:

— SCL - przebieg zegarowy synchronizujący transmisję,

— SDA - do przesyłania danych w dwóch kierunkach.

Połączenie na magistrali TWI jest typy Master –Slave. Przebieg zegarowy jest generowany przez procesor – Master magistrali.

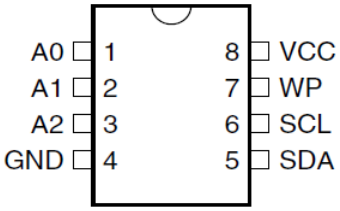
http://fabacademy.org/2018/labs/barcelona/students/oscar-gonzalezfernandez//2018/05/02/Week-13-Networking-and-communication.html

Linie magistrali TWI i linie adresowe układów Slave

Stan wysoki jest stanem „idle” linii SDA i SCL - linie magistrali TWI muszą być podciągnięte zewnętrznie pod zasilanie (mocne podciągnięcie) przez rezystor o wartości 4.7 kOHm.
Jeżeli na magistrali jest więcej niż jeden układ Slave tego samego typu to można je rozróżnić po adresie sprzętowym utworzonym przez linie adresowe A2, A1, A0 połączone do masy lub zasilania

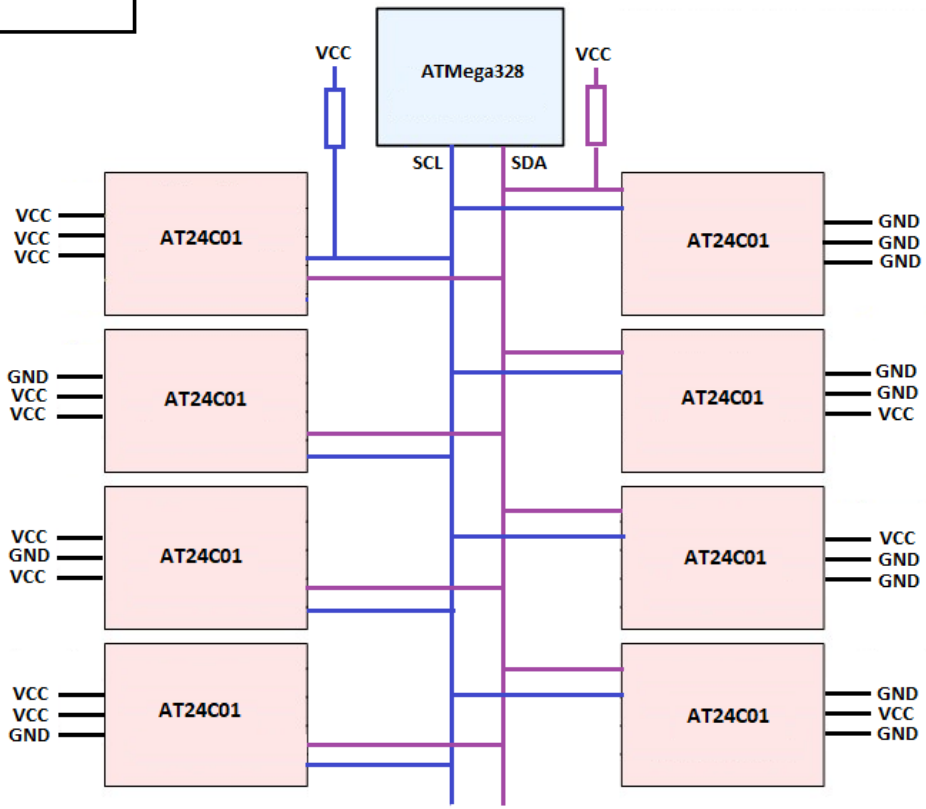
Wyprowadzenia pamięci EEPROM AT24C01-16

Linia wyprowadzenia	Funkcja Linii
A0-A2	Wejścia adresowe
SCL	Wejście zegarowe
SDA	Linia danych
WP	Zabezpieczenie przed nieuprawnionym zapisem



Liczba linii adresowych w pamięciach EEPROM AT24CXX

Typ pamięci	Linie adresowe	Liczba pamięci jednego typu w systemie
AT24C01	A2, A1, A0	8
AT24C02	A2, A1, A0	8
AT24C04	A2, A1	4
AT24C08	A2	2
AT24C16	Brak	1



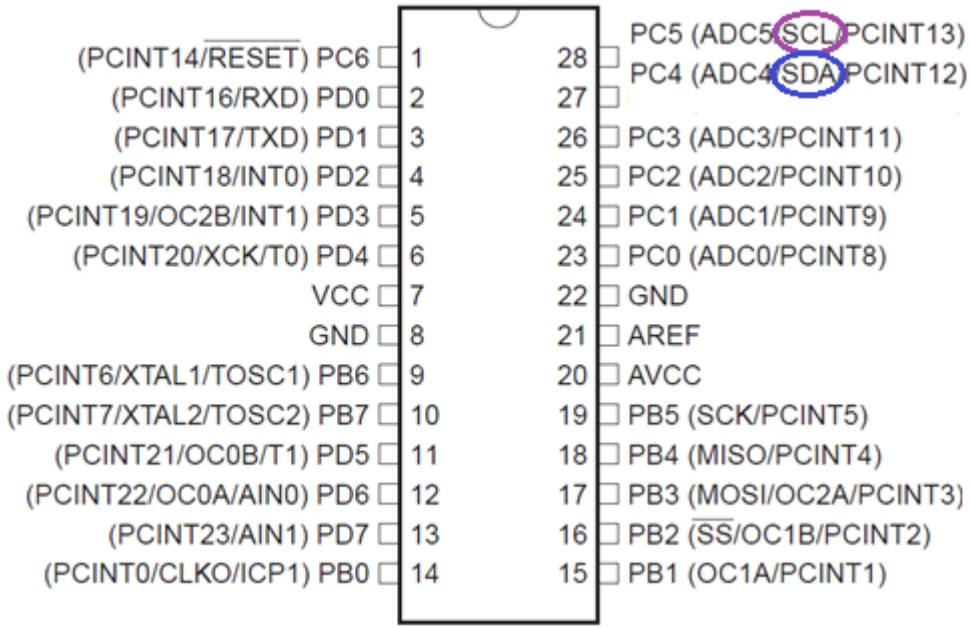
Linie adresowe A2, A1, A0 w układzie AT24C01

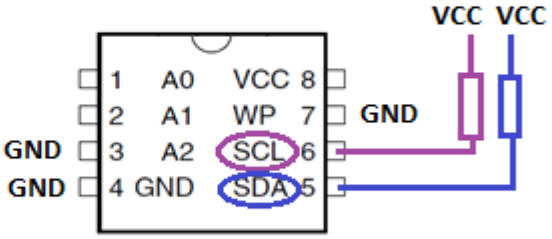
Bajt adresu urządzenia (device address)

Pierwszy bajt wysyłany przez Master procesor zaraz po sekwencji startu jest szczególny i zawiera:
— 4-ro bitowy adres rodziny urządzeń (bity 7-4) dla rodziny pamięci EEPROM: 1010
— adres sprzętowy A2, A1, A0 (bity 3-1)
— starsze bity adresu strony (lub komórki) pamięci P2, P1, P0 (bity 3-1),
— tryb (1: Read / 0: Write).

Bit	7	6	5	4	3	2	1	0
AT24C01/02	1	0	1	0	A2	A1	A0	R/W
AT24C04	1	0	1	0	A2	A1	P0	R/W
AT24C08	1	0	1	0	A2	P1	P0	R/W
AT24C16	1	0	1	0	P2	P1	P0	R/W

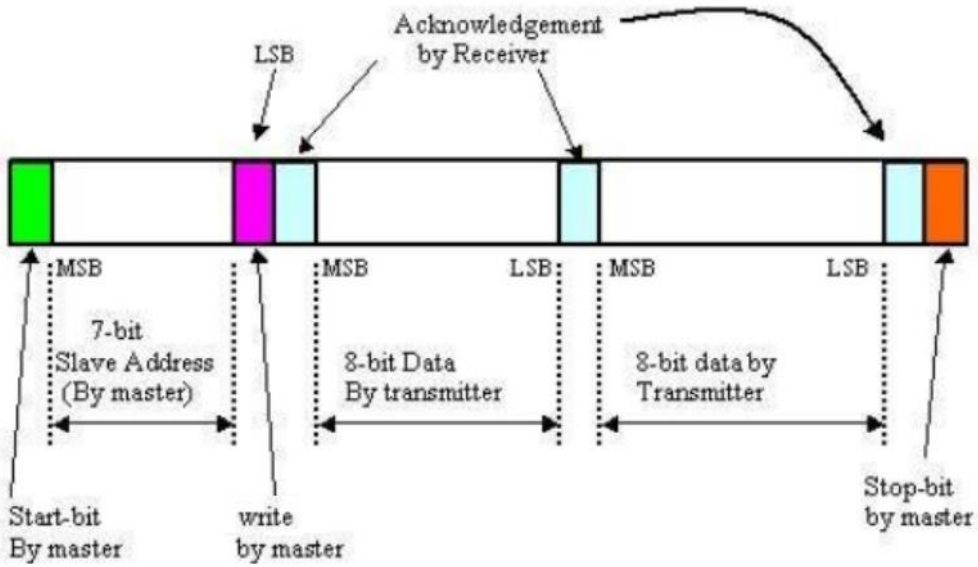
Połączenia układu laboratoryjnego (EEPROM AT24C08)





AT24C08 posiada tylko jedną linię adresową A2, która na schemacie została połączona z masą.

Transmisja danych po magistrali TWI



Master transmit – zapis w komórce pamięci

Transmisja jest synchroniczna i odbywa się po linii SDA na zmianę w obu kierunkach.

Procesor Master rozpoczyna od sekwencji startu (zielony), po czym nadaje 8 bitów adresu do układu Slave.

Najmłodszy bit (amarantowy) jest równy zero dla określenia trybu transmisji – write.

Slave potwierdza jednym bitem (błękit) tj. impulsem ujemnym na SDA na 9-tym impulsie zegarowym na SCL.

Następnie Master nadaje kolejny bajt (8 bitów) (białe), a Slave (tu: odbiornik) potwierdza bitem 0 (błękit).

Można zapisać maksymalnie jedną stronę pamięci w jednej transmisji.

Master kończy transmisję nadając sekwencję stopu (pomarańczowy).

https://www.slideshare.net/RAMPRAKASHT1/peripherals-and-interfacing

Adresowanie komórek i stronicowanie pamięci EEPROM AT24CXX

Typ pamięci	Liczba stron pamięci	Liczba bajtów na stronę	Długość słowa adresowego w bitach
AT24C01	16	8	7
AT24C02	32	8	8
AT24C04	32	16	9
AT24C08	64	16	10
AT24C16	128	16	11

Typ pamięci	Słowo adresu	Długość adresu [bitów]
AT24C01	*XXXXXXX	7
AT24C02	XXXXXXXX	8
AT24C04	P0XXXXXXXX	9
AT24C08	P1P0XXXXXXXX	10
AT24C16	P2P1P0XXXXXXXX	11

Pamięć AT24C08 ma 64 stron po 16 bajtów na stronę a zatem 1024 komórki.
Zaadresowanie komórki wymaga słowa 10-bitowego bo 2^10 = 1024.
Adres komórki należy podzielić na dwa kolejne przesyłane bajty: bity najstarsze bity adresu P1 i P0 trafią do innego bajtu niż reszta XXXXXXXX np. 1000 = 0b1111101000 P1=1, P0=1, XXXXXXXX=11101000

Rejestry interfejsu TWI w ATMega328 – rejestr prędkości i kontrolny

TWBR – TWI Bit Rate Register

Bit	7	6	5	4	3	2	1	0	
(0xB8)	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot (PrescalerValue)}$$

Bity TWPS1 i TWPS2 znajdują się w rejestrze TWSR

TWPS1	TWPS0	Prescaler Value
0	0	1
0	1	4
1	0	16
1	1	64

TWCR – TWI Control Register

Bit	7	6	5	4	3	2	1	0	
(0xBC)	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7 – TWINT: TWI Interrupt Flag – flaga zakończenia nadawania lub odbioru bajtu; przed kolejnym bajtem należy flagę wyzerować przez zapisanie logicznej jedynki
Bit 6 – TWEA: TWI Enable Acknowledge Bit – wygeneruj potwierdzenie (po odbiorze bajtu)
Bit 5 – TWSTA: TWI START Condition Bit – wygeneruj sekwencję Startu
Bit 4 – TWSTO: TWI STOP Condition Bit – wygeneruj sekwencję Stopu
Bit 3 – TWWC: TWI Write Collision Flag – flaga kolizji
Bit 2 – TWEN: TWI Enable Bit – aktywacja interfejsu TWI
Bit 1 – Res: Reserved Bit
Bit 0 – TWIE: TWI Interrupt Enable – aktywacja przerwań od TWI

www.atmel.com

Rejestry interfejsu TWI w ATMega328 – rejestr statusowy i danych

TWSR – TWI Status Register

Bit	7	6	5	4	3	2	1	0	
(0xB9)	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

Bits 7- 3 – TWS7, TWS6, TWS5, TWS4, TWS4: bity statusów, statusy sygnalizują poprawne poprzedniej sekwencji transmisji danych (start, zapis bajtu, odczyt bajtu, stop) i wskazują błędy

Bit 2 – bit zarezerwowany

Bits 1- 0 – TWPS1, TWPS0 – bity preskalera

TWDR – TWI Data Register

Bit	7	6	5	4	3	2	1	0	
(0xBB)	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

Procedury do transmisji danych

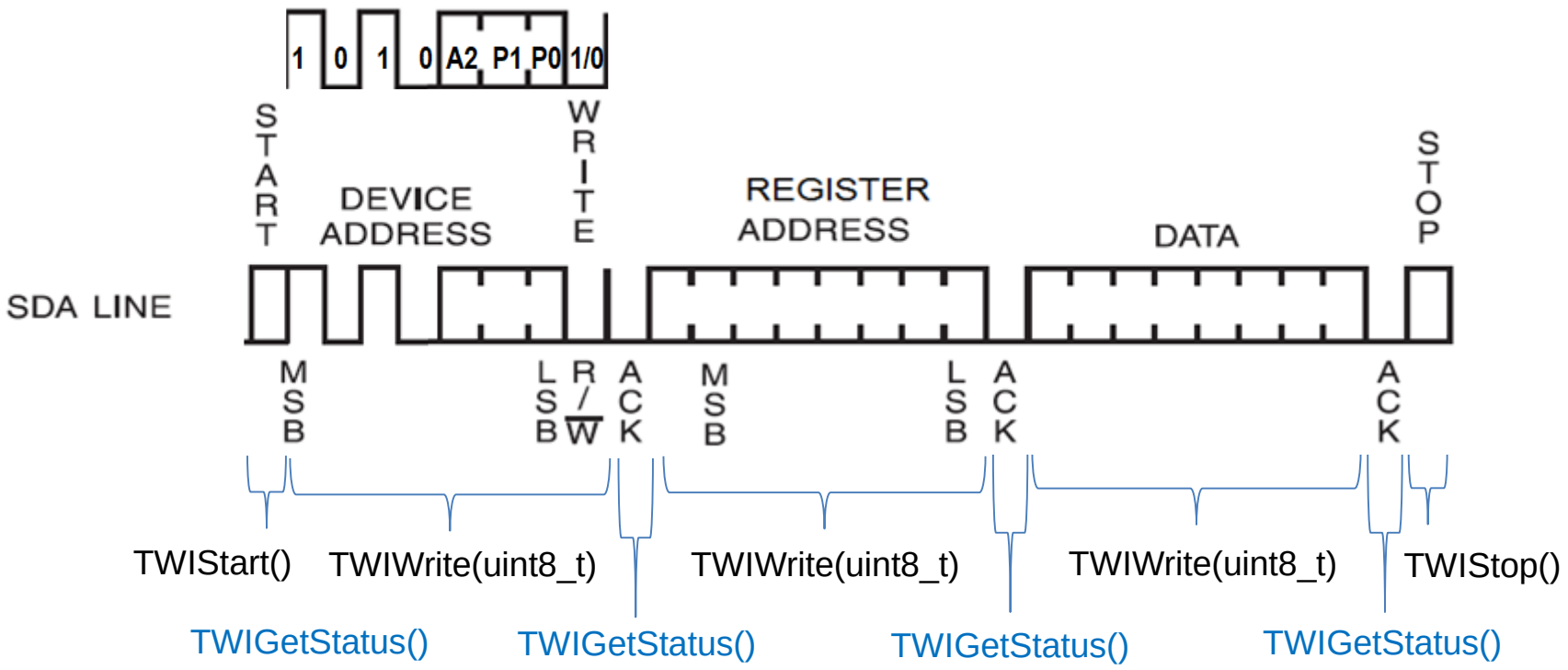
```
void TWIInit(void)
{
   //set SCL to 400kHz
   TWSR = 0x00;
   TWBR = 0x0C;
   TWCR = (1<<TWEN);
}

void TWIStart(void)
{
   TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);
   while ((TWCR & (1<<TWINT)) == 0);
}

void TWIStop(void)
{
   TWCR = (1<<TWINT)|(1<<TWSTO)|(1<<TWEN);
}

void TWIWrite(uint8_t u8data)
{
   TWDR = u8data;
   TWCR = (1<<TWINT)|(1<<TWEN);
   while ((TWCR & (1<<TWINT)) == 0);
}
```

```
uint8_t TWIReadACK(void)
{
   TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWEA);
   while ((TWCR & (1<<TWINT)) == 0);
   return TWDR;
}

uint8_t TWIReadNACK(void)
{
   TWCR = (1<<TWINT)|(1<<TWEN);
   while ((TWCR & (1<<TWINT)) == 0);
   return TWDR;
}

uint8_t TWIGetStatus(void)
{
   uint8_t status;
//zerowanie bitów prescalera przy odczytywaniu statusu
  status = TWSR & 0xF8;
   return status;
}
```

www.atmel.com

Protokół zapisu jednego bajtu

Celem transmisji jest zapis jednej komórki pamięci AT24CXX.
W pierwszym bajcie wysyłany jest adres urządzenia (pamięci), w drugim adres komórki, w trzecim dana dla komórki.
W jednej transmisji danych można zapisać nie więcej niż stronę danych.



Kody statusowe

Master transmitter

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware
0x08	A START condition has been transmitted
0x10	A repeated START condition has been transmitted
0x18	SLA+W has been transmitted; ACK has been received
0x20	SLA+W has been transmitted; NOT ACK has been received
0x28	Data byte has been transmitted; ACK has been received
0x30	Data byte has been transmitted; NOT ACK has been received
0x38	Arbitration lost in SLA+W or data bytes

Master receiver

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware
0x08	A START condition has been transmitted
0x10	A repeated START condition has been transmitted
0x38	Arbitration lost in SLA+R or NOT ACK bit
0x40	SLA+R has been transmitted; ACK has been received
0x48	SLA+R has been transmitted; NOT ACK has been received
0x50	Data byte has been received; ACK has been returned
0x58	Data byte has been received; NOT ACK has been returned

START – sekwencja startu
SLA+W – adres Slave'a z bitem zapisu (Write) na końcu
SLA+R – adres Slave'a z bitem odczytu (Read) na końcu
ACK – otrzymano/nadano potwierdzenie
NOT ACK – nie otrzymano lub nie nadano potwierdzenia
STOP – sekwencja stopu

Procedura zapisu jednego bajtu danych z odczytem statusów

```
#define ERROR 1
#define SUCCESS (!ERROR)


uint8_t EEWriteByte(uint8_t device_address,
uint8_t register_address,  uint8_t u8data)
{
   TWIStart();
   if (TWIGetStatus() != 0x08)
      return ERROR;

TWIWrite(device_address);
   if (TWIGetStatus() != 0x18)
      return ERROR;

TWIWrite(register_address);
   if (TWIGetStatus() != 0x28)
      return ERROR;

TWIWrite(u8data);
   if (TWIGetStatus() != 0x28)
      return ERROR;

   TWIStop();
   return SUCCESS;
}
```

Zapis jednego bajtu danych w pamięci EEPROM AT24CXX

Przykład dla AT24C08

Np. komórka 1000, linia adresowa na masie

1000 = 0b1111101000

device_address = 0b1010$A_2P_1P_0$0 = 0b10100110

register_address = 0b11101000

u8data = 0x41 //ascii A

Jeden zapis do pamięci EEPROM trwa około 10ms

Uproszczona procedura zapisu jednego bajtu danych bez odczytu statusów

```
void EEWriteByte(uint8_t device_address,
uint8_t register_address,  uint8_t u8data)
{
   TWIStart();
   TWIWrite(device_address);
   TWIWrite(register_address);
   TWIWrite(u8data);
   TWIStop();

}
```

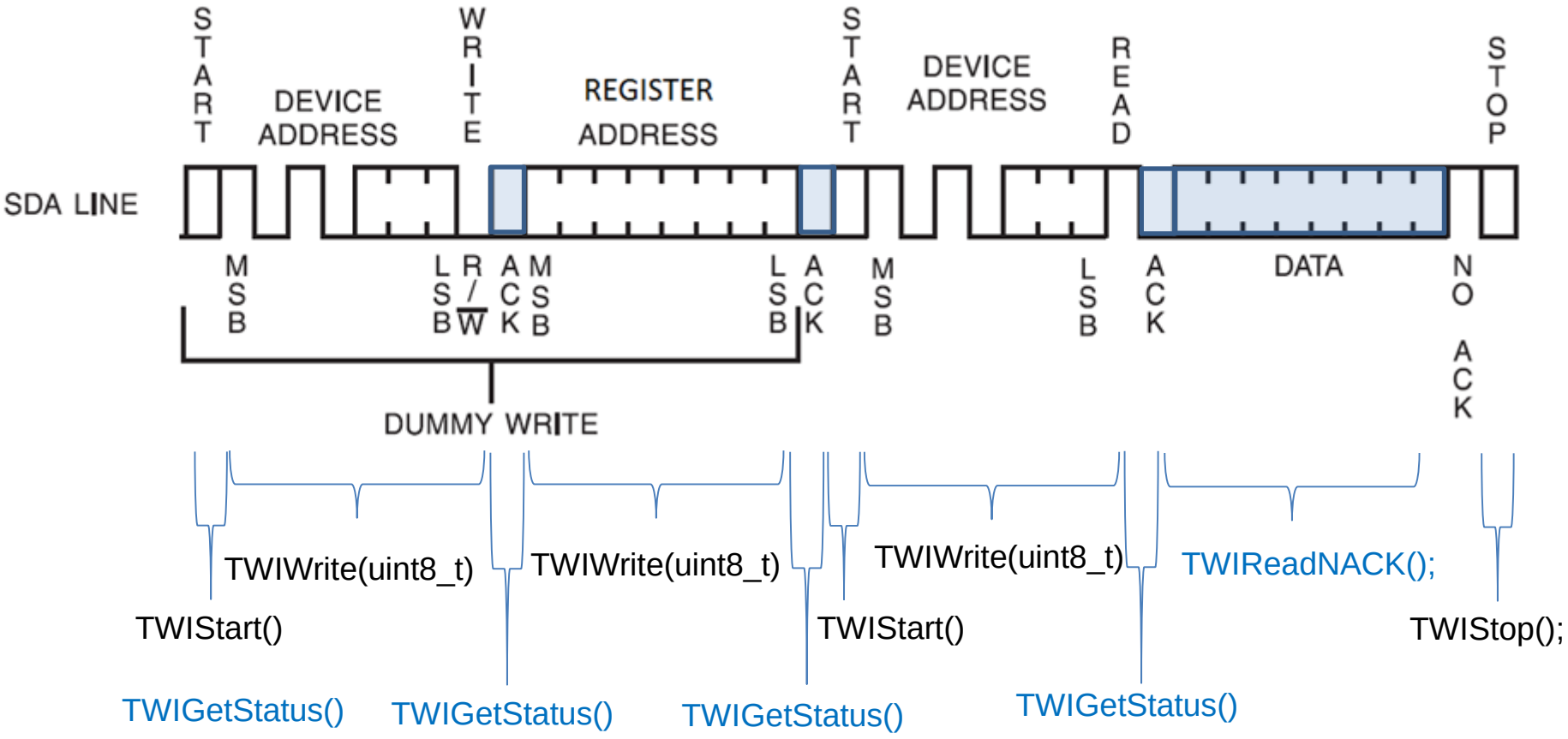
https://embedds.com/programming-avr-i2c-interface/

Protokół odczytu jednego bajtu

Protokół odczytu wymaga nadania po kolei: sekwencji startu, bajtu adresu urządzenia z najmłodszym bitem Write (0), adresu komórki pamięci z której zamierzam odczytać daną, kolejnej sekwencji startu, jeszcze raz bajtu adresu urządzenia tym razem z najmłodszym bitem Read (1). Dopiero teraz następuje odczyt bajtu (zawartości komórki).

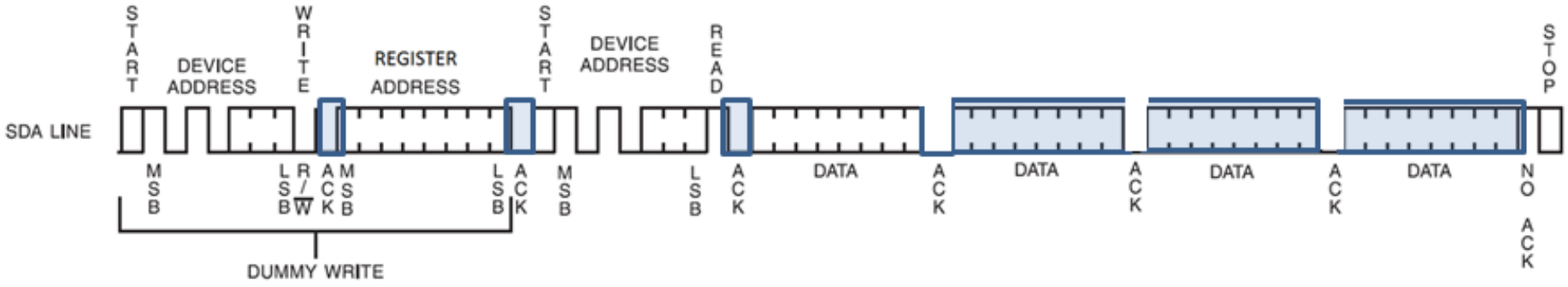
Master nadajnik – biały

Slave nadajnik - błękitny



Protokół odczytu wielu bajtów

W jednej transmisji danych można odczytać dowolną liczbę komórek – również całą pamięć.
Master potwierdza (ACK) odczytanie wszystkich komórek za wyjątkiem ostatniej (NO ACK),
po czym nadaje sekwencję stopu.
Master nadajnik – biały
Slave nadajnik - błękitny



Odczyt jednego bajtu danych z pamięci AT24CXX

Procedura odczytu jednego bajtu danych z odczytem statusów

```
#define ERROR 0
#define SUCCESS (!ERROR)
uint8_t   u8data

uint8_t EEWriteByte(uint8_t device_address,
uint8_t register_address,  uint8_t *u8data)
{
    TWIStart();
    if (TWIGetStatus() != 0x08)
        return ERROR;
TWIWrite(device_address);
    if (TWIGetStatus() != 0x18)
        return ERROR;
TWIWrite(register_address);
    if (TWIGetStatus() != 0x28)
        return ERROR;
TWIStart();
    if (TWIGetStatus() != 0x10)
        return ERROR;
TWIWrite(device_address | 1);
    if (TWIGetStatus() != 0x40)
        return ERROR;
*u8data = TWIReadNACK();
    if (TWIGetStatus() != 0x58)
        return ERROR;
    TWIStop();
    return SUCCESS;
}
```

Uproszczona procedura odczytu jednego bajtu danych bez odczytu statusów

```
void EEWriteByte(uint8_t device_address,
uint8_t register_address,  uint8_t *u8data)
{
    TWIStart();
    TWIWrite(device_address);
    TWIWrite(register_address);
    TWIStart();
    TWIWrite(device_address | 1);
    *u8data = TWIReadNACK();
    TWIStop();
}
```

https://embedds.com/programming-avr-i2c-interface/