Systemy wbudowane
Bioniczna ręka

Autorzy projektu:

Igor Czaplikowski - Lider
Patryk Zielewski
Artur Kaźmierczak

Projekt wykonaliśmy na dwóch Arduino Nano. Cały użyty do projektu sprzęt był nasz.

Spis treści

1. Spis funkcjonalności

Funkcjonalność	Stan	Osoba Odpowiedzialna
SEN0064 – czujnik ugięcia	Działa	
MG-996R – serwomotor		
KY-040 – enkoder rotacyjny		
SSD1306 – wyświetlacz		
nRF24L01+ – moduł do transmisji radiowej		
Obsługa I2C		
Obsługa SPI		

2. Podział procentowy pracy

Igor Czaplikowski -

Patryk Zielewski -

Artur Kaźmierczak (230170) –

3. Opis projektu

Nasz projekt zakładał stworzenie bionicznej ręki z elementami wydrukowanymi na drukarce 3D. Skorzystaliśmy z gotowego do wydruku modelu ręki dostępnego na stronie InMove [1]. Bioniczna ręka będzie się poruszała na podstawie ruchów ręki człowieka, która będzie miała załażoną rękawice z czujnikami ugięcia SEN0064, po jednym na każdy palec. Są one podłączone do Arduino Nano, które odczytuje wartości z czujników i przy pomocy modułu transmisji radiowej nRF24L01+ wysyła dane do takiego samego modułu, który jest podłączony do drugiego Arduino Nano, działa on jako odbiornik danych i sterownik serwomotorów MG-996R.

4. Instrukcja obsługi

Na wyświetlaczu który jest przymocowany do rękawiczki z nadajnikiem, znajduje się przypięty wyświetlacz (SSD1306) na którym pokazane są 3 opcje do wyboru. Opcje wybieramy za pomocą enkodera rotacyjnego (KY-040 [2]), nie ważne w którą stronę będziemy kręcić, zaznaczenie zawsze będzie przechodziło w dół. Po pełnym przejściu listy wyświetli się czarny ekran, a następnie przy kolejnym przekręceniu enkodera pokaże się menu od początku. Aby przejść do wybranej opcji należy wcisnąć przycisk enkodera.

Pierwsza opcja „Naśladowanie" pozwala na sterowanie bioniczną ręką za pomocą rękawicy sterującej, pozwala ona na kopiowanie ruchów zginania palcy. Do każdego palca na rękawicy przymocowany jest czujnik ugięcia który zczytuje rezystancje (na podstawie znalezionego Datasheet SEN0064 [3] do tego elementu możemy zobaczyć że powinna ona być z zakresu 60 – 110 KOhms), wartości te mapujemy do kąta obrotu serwomotoru (MG-996R [4]) i wysyłamy je z nadajnika na rękawiczce do odbiornika przy bionicznej ręce. Następnie w odbiorniku wysyłamy informacje do serwomotorów które naciągają linki przymocowane do końców palców. Do serwomotorów przymocowane są 2 linki z każdego palca, jedna odpowiada za zginanie, druga za wyprostowanie.

Niestety każdy z czujników ugięcia wymaga osobnej kalibracji, bo przy takim samym ugięciu potrafią dawać różne wyniki. Planujemy zaimplementować system auto-kalibracji który przy każdym uruchomieniu nadajnika zaczytywałby maksymalne i minimalne wartości z czujników, ponieważ zdarza się, że nawet po ponownym uruchomieniu arduino, te same czujniki potrafią dawać różne wyniki.

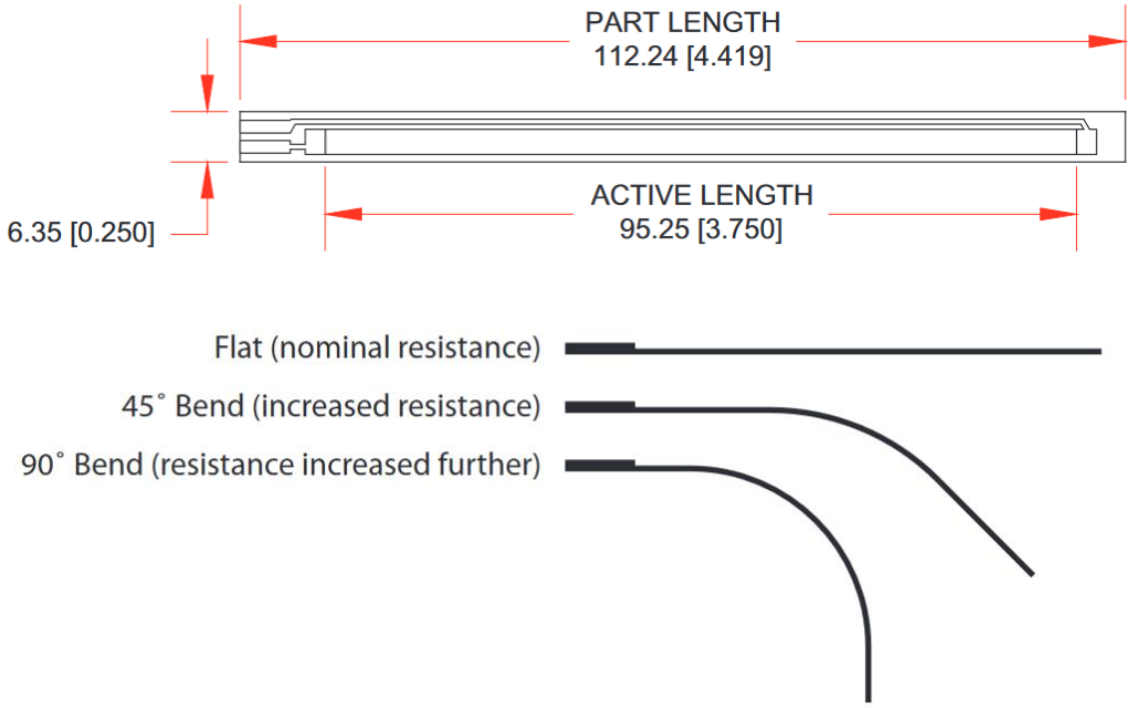
Drugą dostępną opcją jest gra „Kamien,Papier,Nozyce". Po wybraniu tej opcji rozpoczyna się odliczanie po którym zaczytywany jest wybór użytkownika. Na podstawie wartości maksymalnych i minimalnych wartości z kalibracji możemy określić kiedy dany palec jest zgięty, a kiedy wyprostowany. Więc zaimplementowaliśmy algorytm który sprawdza jakie stany mają czujnik ugięcia na nadajniku i wyświetla na ekranie czy wybrałeś papier, kamień, czy nożyce. Po czym losuje liczbę z przedziału 1-3 która reprezentuje wybór bionicznej ręki i wysyła tą informacje do odbiornika. W odbiorniku mam zdefiniowane maksymalne i minimalne kąty jakie może ustawić serwomotor. Zaimplementowaliśmy również proste metody które wysyłają informacje do silnika które palce mają być zgięte, a które wyprostowane. Dzięki temu po wylosowaniu wyboru ręki i przesłaniu tej informacji, ręka zaciska lub prostuje palce w zależności od wyboru, np. dla wyboru „kamień" wszystkie palce będą zaciśnięte. Na ekranie w nadajniku pojawi się kolejny napis z wyborem bionicznej ręki.

Trzecia opcja „Informacje" wyświetla tylko na ekranie nadajnika napis „Bioniczna Ręka".

5. Opis funkcjonalności

5.1 SEN0064 – czujnik ugięcia

Czujniki ugięcia podłączyliśmy do pinów analogowych w arduino. Czujniki działają w ten sposób, że czym bardziej są zgięte, tym jest większa oporność (powinna być w zakresie 60 – 110 kOhm). Po zmierzeniu zakupionych przez nas egzemplarzy, stwierdziliśmy że odczyty jednego z nich różnią się dość mocno od pozostałych, ale mieszczą się w zakresie. Ważne jest aby nie zginać tych czujników w nieodpowiednią stronę, mogą one się przy takiej czynności uszkodzić.



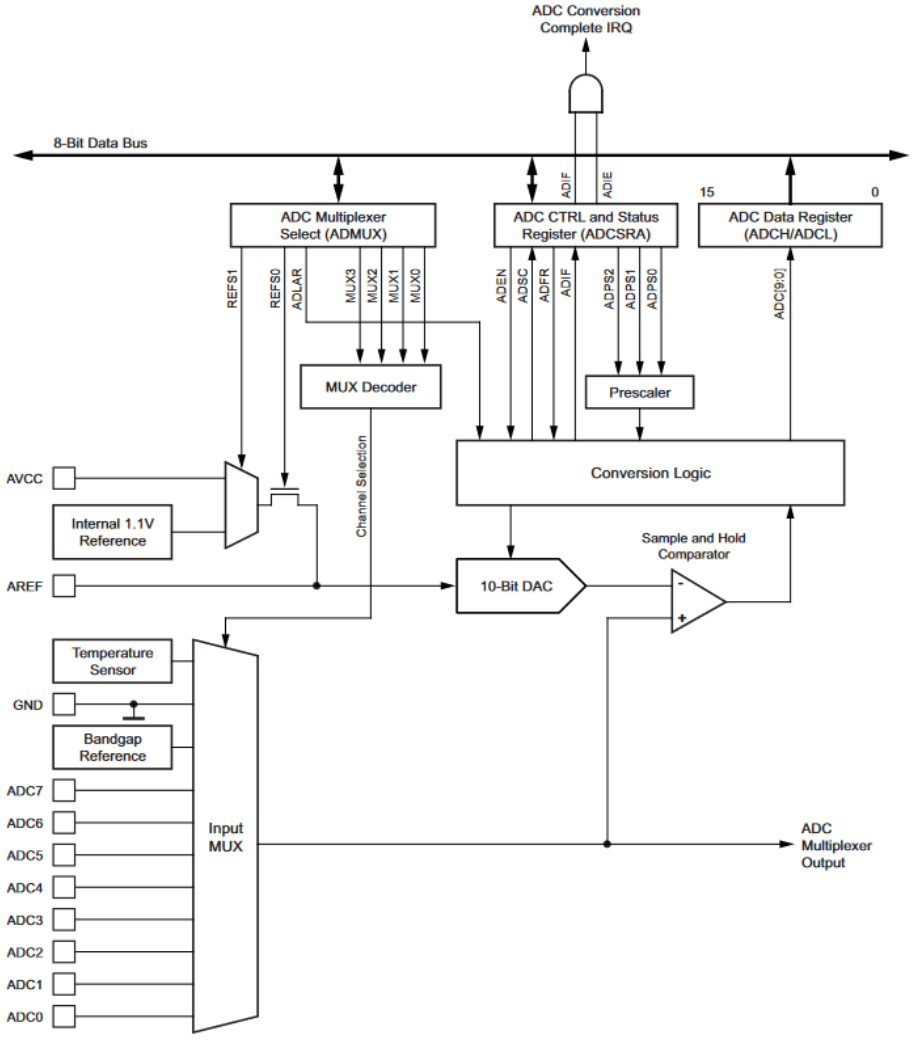

```
pinMode(A0, INPUT);
pinMode(A1, INPUT);
pinMode(A2, INPUT);                 //ustawienie pinów czujników zgięcia jako
pinMode(A3, INPUT);                 //wejście
pinMode(A6, INPUT);
```

Ustawiamy piny czujników ugięcia w tryb INPUT (Digital Pins [5]), który powoduje że pin jest w stanie wysokiej impedancji, czyli aby zmienić stan pinu z jednego na inny potrzebny jest mały prąd.

```
low[0] = analogRead(A6);
low[1] = analogRead(A3);
low[2] = analogRead(A0);
low[3] = analogRead(A1);
low[4] = analogRead(A2);
```

Za pomocą metod analogRead() z biblioteki arduino zaczytujemy wartości z pinów gdzie są podłączone czujniki ugięcia.

ATmega328P posiada 10 bitowy ADC, który jest podłączony do 8-kanałowego analogowego multipleksera, który umożliwia 8 wejść napięciowych. ADC przekształca analogowe napięcie wejściowe na 10-bitową wartość cyfrową poprzez kolejne przybliżanie. Minimalną wartość prądu wejściowego reprezentuje GND, a maksymalną napięcie na pinie AREF pomniejszone o 1 LSB. Na wyjściu otrzymujemy wynik z zakresu 1-1024.



Kanał wejścia analogowego jest wybierany poprzez zapisanie bitów MUX w ADMUX.

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
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	

Bity 7:6 – REFS1:0 – Reference Selection Bits, są to bity używane do wyboru napięcia odniesienia. Używane są kombinacje:

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

ADC potrzebuje napięcia odniesienia do pracy, mamy do tego celu 3 piny AREF, AVCC i GND. Możemy dostarczyć własne napięcie odniesienia po pinach AREF i GND, w tym celu należy wybrać opcję pierwszą. Możesz również podłączyć kondensator przez pin AREF i go uziemić, aby zapobiec szumom, lub możesz go nie podłączać. Jeżeli chcesz skorzystać z VCC (+5V) należy wybrać drugą opcje, lub wybierz ostatnią opcje dla wewnętrznego Vref.

Bit 5 – ADLAR – ADC Left Adjust Result – ustaw na „1” aby wyregulować w lewo ADC.

Bity 4:0 – MUX4:0 – Analog Channel and Gain Selection Bits. Aby wybrać któryś kanał analogowy należy ustawić odpowiednio te bity według tabelki:

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.22 V (V_{BG})	N/A		
11111	0 V (GND)			

Input Channel and Gain Selections

```
// set the analog reference (high two bits of ADMUX) and select the
// channel (low 4 bits). this also sets ADLAR (left-adjust result)
// to 0 (the default).
#if defined(ADMUX)
#if defined(__AVR_ATtiny25__) || defined(__AVR_ATtiny45__) || defi-
ned(__AVR_ATtiny85__)
ADMUX = (analog_reference << 4) | (pin & 0x07);
#else
ADMUX = (analog_reference << 6) | (pin & 0x07);
#endif
```

Kod obsługi rejestrów sprawdzaliśmy w bibliotece wiring_analog.c [6]

Jako wejście do ADC może być wybrany dowolny z pinów ADC, pin GND. ADC jest włączany poprzez ustawienie bitu ADC Enabled w ADCSRA.

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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	

ADCSRA Register

Bit 7 – ADEN – ADC Enable, włacza on funkcjonalność ADC. Dopóki ten bit nie jest włączony, operacje na ADC nie mogą być wykonywane.

Bit 6 – ADSC – ADC Start Conversion – należy ustawić „1” przed jakąkolwiek zamianą wartości. Wartość 1 pozostaje na tym bicie dopóki trwa zamiana wartości, po czym bit jest ustawiany na 0.

Bit 5 – ADATE – ADC Auto Trigger Enable – ustawienie tego bitu na „1” powoduje automatyczne włączenie ADC. ADC jest automatycznie włączane przy każdym pulsie zegara.

Bit 4 – ADIF – ADC Interrupt Flag – za każdym razem, kiedy konwersja jest skończona i rejestry są zaktualizowane, ten bit jest ustawiany automatycznie na „1”. Jest on używany czy konwersja się zakończyła czy nie.

Bit 3 – ADIE – ADC Interrupt Enable – kiedy ten bit jest ustawiony na „1” przerwania ADC są włączone.

Bity 2:0 – ADPS2:0 – ADC Prescaler Select Bits – Preskaler (współczynnik podziału między częstotliwością XTAL a częstotliwościami zegara ADC) wybieramy go tak jak w tabeli poniżej:

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

```
#if defined(ADCSRA) && defined(ADCL)

// start the conversion

sbi(ADCSRA, ADSC);



// ADSC is cleared when the conversion finishes

while (bit is set(ADCSRA, ADSC));
```

ADCL i ADCH – ADC Data Registers

Wyniki konwersji przechowywane są w tych rejestrach, ponieważ ADC ma wielkość 10 bitów, to jeden rejstr 8 bitowy byłby zamały. Więć potrzebujemy 2 rejestrów ADCL i ADCH (ADC Low byte i ADC High byte).

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	*ADLAR = 1*

Możemy tu zobaczyć wynik ustawienia bitu ADLAR (w rejestrze ADMUX), ustawienie go na „1” powoduje dostosowanie do lewej.

```
// we have to read ADCL first; doing so locks both ADCL
// and ADCH until ADCH is read. reading ADCL second would
// cause the results of each conversion to be discarded,

// as ADCL and ADCH would be locked when it completed.

low = ADCL;
high = ADCH;
#else
// we dont have an ADC, return 0
low = 0;
high = 0;
#endif

// combine the two bytes
return (high << 8) | low;
```

Niestety zauważyliśmy, że czujniki potrafią po ponownym uruchomieniu podawać inne dane niż wcześniej, dlatego zaimplementowaliśmy mechanizm auto-kalibracji, który jest odpalany zawsze na starcie modułu. Musimy najpierw zaczytać wartość na wyprostowanych palcach, następnie na zagiętych.

5.2MG-996R – serwomotor

Pwm, biblioteka servo

void write(int value); // if value is < 200 its treated as an angle, otherwise as pulse width in microseconds

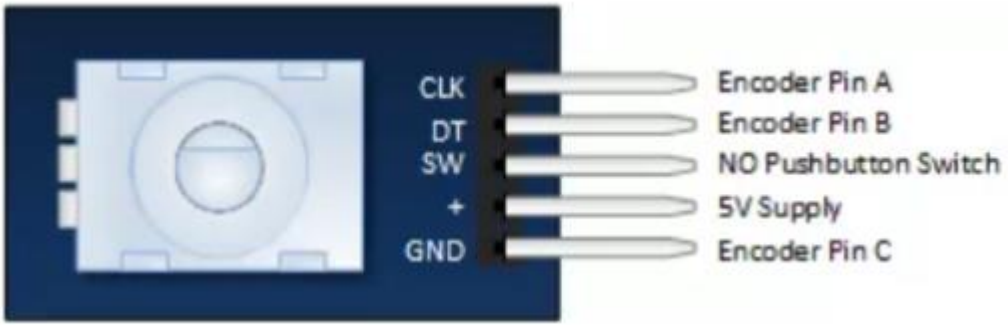
uint8_t attach(int pin); // attach the given pin to the next free channel, sets pinMode, returns channel number or 0 if failure

5.3 KY-040 – enkoder rotacyjny

KY-040 - Moduł zawierający enkoder (impulsator) obrotowy. Dzięki zastosowanemu przetwornikowi, moduł wysyła sygnały określające kierunek i zakres obrotu. Dodatkowo moduł posiada przycisk, wbudowany w oś enkodera, dzięki czemu użyliśmy go do obsługi wyświetlacza w naszym układzie.

Czujnik posiada 5 pinów:

Nazwa	Opis
VCC	Napięcie zasilania modułu w zakresie 2.5 -5V.
GND	Masa układu.
SW	zwiera pin do GND gdy gałka zostanie wciśnięta
DT	puls kierunku
CLK	puls zegara

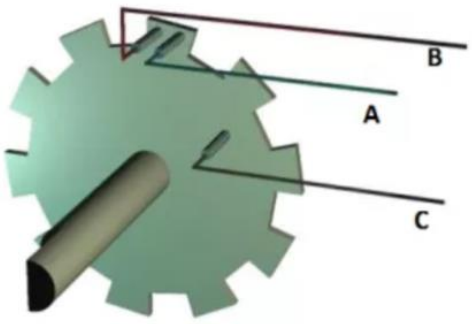


Enkoder obrotowy ma stałą liczbę pozycji na obrót, można je wyczuć jako kliknięcia, gdy obracamy enkoderem.

Po jednej stronie przełącznika są 3 piny, określane zazwyczaj jako A, B, C, tak jak na poniższym obrazku:

Wewnątrz enkodera znajdują się dwa przełączniki. Jeden przełącznik łączy pin A z pinem C, a drugi przełącznik łączy pin B z C. W każdej pozycji enkodera oba przełączniki są albo otwarte, albo zamknięte. Każde kliknięcie powoduje że te przełączniki zmieniają stany:

- Jeśli oba przełączniki były zamknięte, obrócenie enkodera w prawo lub w lewo o jedną pozycje spowoduje otwarcie obu przełączników
- Jeśli oba przełączniki są otwarte, obrócenie enkodera w prawo lub w lewo spowoduje zamknięcie obu przełączników



Powyższa ilustracja przedstawia konstrukcję przełącznika.

Obrót przełącznika zgodnie z ruchem wskazówek zegara spowoduje, że stan najpierw zmieni przełącznik łączący A i C, natomiast obrót enkodera w kierunku przeciwnym do wskazówek zegara spowoduje, że jako pierwszy zmieni stan przełącznik łączący B i C.

Aby określić kierunek w którym enkoder jest obracany, musimy określić który przełącznik zmienił stan jako pierwszy:

- Jeżeli A zmienił stan jako pierwszy, przełącznik obraca się w kierunku zgodnym do ruchu wskazówek zegara
- Jeżeli B zmienił stan jako pierwszy, przełącznik obraca się w kierunku przeciwnym do ruchu wskazówek zegara

Moduł jest zaprojektowany tak, że na wyjściu jest stan LOW, gdy przełączniki są zamknięte, a HIGH kiedy są otwarte. Stan LOW generowany jest poprzez umieszczenie masy na pinie C i przekazanie go do pinów CLK i DT, gdy przełączniki są zamknięte. Stan HIGH natomiast generowany jest przez wejście zasilania 5V i rezystory, tak że CLK I DT mają stan HIGH kiedy przełączniki są otwarte.

```
#define outputA 4
#define outputB 3                          //definicja pinów od enkodera
#define selectButton 2
…
void setup() {
  pinMode(outputA,INPUT_PULLUP);
  pinMode(outputB,INPUT_PULLUP);       //ustawienie pinów enkodera jako wejście
  pinMode(selectButton, INPUT_PULLUP);  //podciągnięte do zasilania
…
aLastState = digitalRead(outputA);  //odczytaj stan pinu enkodera
```

Przypisujemy odpowiednie piny w kodzie do odpowiadającym im pinom w enkoderze, ustawiamy je w tryb INPUT_PULLUP co oznacza, że kiedy stan jest HIGH to przełącznik jest wyłączony, a kiedy stan jest LOW to przełącznik jest włączony. W metodzie setup odczytujemy również aktualny stan wyjścia na pinie A, żeby móc go później porównać i stwierdzić czy nastąpiła zmiana.

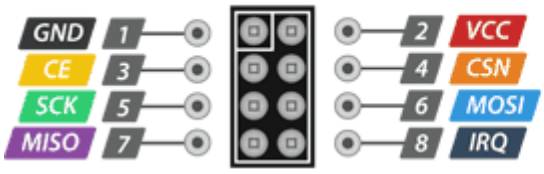
```
void loop() {

  aState = digitalRead(outputA);     //odczytaj stan pinu enkodera
  if (aState != aLastState){        //jeżeli odczytany stan jest różny od
poprzedniego stanu
    if (digitalRead(outputB) != aState) {
      plus = true;                 //plus ustawiamy na 1
    }
  }
  aLastState = aState;           //poprzedni stan = stan

  if(plus == true){             //jeżeli plus = 1
    menu++;                     //kolejna strona menu
    drawMenu();                 //pokaż menu na ekranie
    delay(100);                 //czekaj 0,1s
    while(!plus);               //czekaj do póki !plus
    plus=false;                 //ustaw plus na 0
  }

  if (!digitalRead(selectButton)){
    delay(200);
    display.clearDisplay();
    do{
      executeAction();
    }while(digitalRead(selectButton) == HIGH);
    drawMenu();
    while (!digitalRead(selectButton));
  }

  display.display();
}
```

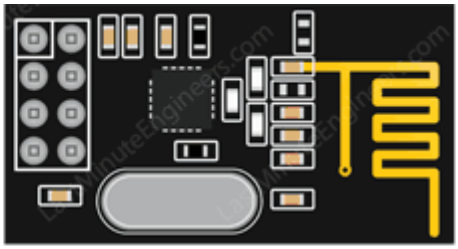
Sprawdzamy czy stan na pinach enkodera jest taki sam jak poprzedni, jeżeli tak to nic nie robimy, jeżeli nie, to przełączamy zaznaczenie opcji w menu na następną. Kiedy naciśniemy przycisk enkodera dostaniemy sygnał LOW na pinie od przycisku i wtedy chcemy wywołać metodę *executeAction().* Do póki pin przycisku będzie dawał sygnał HIGH (przycisk enkodera nie został wciśnięty), chcemy powtarzać wywoływanie metody *executeAction()*, jak dostaniemy sygnał LOW, printujemy na wyświetlaczu menu i wychodzimy z pętli.

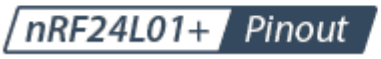
5.4 SSD1306 – wyświetlacz

5.5 nRF24L01+ – moduł do transmisji radiowej

Ręka jak i rękawica komunikują się za pomocą modułu nRF24L01+. Jest to moduł radiowy, który komunikuje się w paśmie 2,4GHz i komunikuje się z Arduino po interfejsie SPI.







- Ground (GND) – Uziemienie
- Voltage Supply (VCC) – Pin zasilający (od 1,9V do 3,9V). W naszym przypadku wykorzystujemy zasilanie 3V3 (3,3V), które udostępnia nam Arduino.
- Chip Enable (CE) – aktywny w stanie wysokim. Kiedy ten pin jest ustawiony na „1”, moduł jest aktywny i będzie nadawał/odbierał dane, w zależności jak jest skonfigurowany.
- Chip Select Not (CSN) – aktywny w stanie niskim, lecz podczas normalnej pracy w stanie wysokim. Gdy ten pin jest ustawiony na stan niski, moduł przechodzi w tryb nasłuchiwania i odczytuje dane z szyny SPI po czym wewnętrznie je przetwarza. Z punktu widzenia mikroprocesora, tutaj podłączamy pin SS (Slave select)
- Serial Clock (SCK) – pin zegarowy, który akceptuje pulsy od Master’a szyny SPI.
- Master Out Slave In (MOSI) – wejście dla danych do modułu
- Master In Slave Out (MISO) – wyjście dla danych z modułu
- Interrupt (IRQ) – pin przerwania, którym może poinformować Master’a, że pojawiły się nowe dane do przetworzenia (w naszym przypadku niewykorzystywane).

W naszym kodzie korzystamy z dedykowanej biblioteki RF24.

Nadajnik (rękawica):

Na początku tworzymy zmienną globalną typu *RF24*, podając położenie pinów CE (Chip Enable) oraz CSN (Chip Select Not).

```
RF24 radio(9, 10);                          //tworzymy instancje komunikacji
```

Następnie wywołujemy serię metod ustawiających pracę modułu.

```
radio.begin();                              //uruchom moduł
radio.setDataRate(RF24_2MBPS);              //ustaw prędkość transmisji na 2Mb/s
radio.setPALevel(RF24_PA_HIGH);             //ustaw wzmocnienie modułu radiowego na
wysokie
radio.openWritingPipe(pipe);                //rozpocznij transmisję
```

Linia `radio.begin()` w naszym przypadku wywołuje jedynie metodę *begin()* na obiekcie klasy SPI (opis dokładny w sekcji o SPI).

Linia `radio.setDataRate(RF24_2MBPS)` powoduje nawiązanie komunikacji po SPI z modułem i pobranie bieżącej konfiguracji i dobranie wartości opóźnień na podstawie danych o częstotliwości taktowania głównego mikroprocesora (czy Arduino pracuje w trybie 8 czy 16MHz), a następnie wysłanie ich do modułu.

```
bool RF24::setDataRate(rf24_datarate_e speed)
{
    bool result = false;
    uint8_t setup = read_register(RF_SETUP);

    // HIGH and LOW '00' is 1Mbs - our default
    setup &= ~(_BV(RF_DR_LOW) | _BV(RF_DR_HIGH));

    #if !defined(F_CPU) || F_CPU > 20000000
    txDelay = 280;
    #else //16Mhz Arduino
    txDelay=85;
    #endif
    if (speed == RF24_250KBPS) {
        // Must set the RF_DR_LOW to 1; RF_DR_HIGH (used to be RF_DR) is
already 0
        // Making it '10'.
        setup |= _BV(RF_DR_LOW);
        #if !defined(F_CPU) || F_CPU > 20000000
        txDelay = 505;
        #else //16Mhz Arduino
        txDelay = 155;
        #endif
    } else {
        // Set 2Mbs, RF_DR (RF_DR_HIGH) is set 1
        // Making it '01'
        if (speed == RF24_2MBPS) {
            setup |= _BV(RF_DR_HIGH);
            #if !defined(F_CPU) || F_CPU > 20000000
            txDelay = 240;
            #else // 16Mhz Arduino
            txDelay = 65;
            #endif
        }
    }
    write_register(RF_SETUP, setup);

    // Verify our result
    if (read_register(RF_SETUP) == setup) {
        result = true;
    }
    return result;
}
```

5.6 Obsługa I2C

5.7 Obsługa SPI

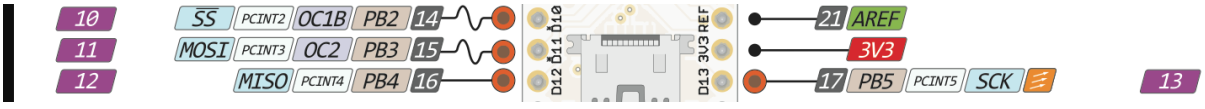
SPI (ang. Serial Peripheral Interface) to interfejs szeregowy, najczęściej wykorzystywany do dwukierunkowej (full-duplex) transmisji danych. Dane są przesyłane synchronicznie w układzie master-slave(-slave-slave…). Wykorzystywane są do tego 8-bitowe rejestry przesuwne po każdej ze stron oraz sygnał taktujący, którego źródłem jest Master (urządzenie zarządzające, nadrzędne).

Na połączenia interfejsu SPI składają się:

- MOSI (Master out Slave in) – jest to wyprowadzenie, które służy jako wyjście danych gdy urządzenie jest w trybie „mastera” LUB wejściem danych gdy jest od „slavem”
- MISO (Master in slave out) – jest to wyprowadzenie, które służy jako wejście danych gdy urządzenie jest w trybie „mastera” LUB wyjściem danych gdy jest od „slavem”
- SCK (Serial Clock) – sygnał taktujący wysyłany przez „mastera”. Służy do synchronizacji operacji odczytu danych
- SS (Slave Select) – linia wyboru aktywnego „slave”. W praktyce wykorzystuje się piny GPIO do ustawiania „0” na wyprowadzeniu docelowego „slave” by zakomunikować mu, że zamierzamy mu wysłać dane (lub od niego odebrać). Stan wysoki „1” oznacza, że urządzenie ma być nieaktywne (nie reagować na dane na linii MOSI, nie ustawiać linii MISO).

W naszym przypadku, interfejs SPI znajduje się na wyprowadzeniach oznaczonych jako:

MISO – D12
MOSI – D11
SCK – D13
SS – D10



Do kontroli, sprawdzenia stanu oraz wysyłania/odbierania danych służą 3 rejestry:

SPCR – SPI Control Register – Rejestr kontrolny. Służy do konfiguracji i sterowania pracą interfejsu SPI

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
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	

- SPIE – Włączenie przerwań. Przerwanie zostanie wygenerowane, jeżeli flaga SPIF w rejestrze SPSR jest ustawiona.
- SPE – Wpisanie 1 włącza SPI. Wykonanie jakichkolwiek operacji na SPI wymaga tego ustawienia.
- DORD – Kolejność przesyłanie danych. 1 oznacza że LSB (Least Significant Bit) jest wysyłany jako pierwszy. 0 odwrotnie – pierwsze zostanie wysłane MSB

- MSTR – Wpisanie 1 na tą pozycję oznacza, że urządzenie będzie działać jako master. 0 jako slave
- CPOL – wybór polaryzacji zegara taktującego. 1 oznacza, że w stanie bezczynności zegara, zegar ma stan wysoki, 0 – niski.
- CPHA – wybór fazy próbkowania. 1 oznacza, że dane są próbkowane na zboczu narastającym, a ustawiane na opadającym. 0 odwrotnie
- SPR1, SPR2 – ustawienie tych dwóch bitów steruje częstotliwością taktowania na pinie SCK, co ma bezpośredni związek z prędkością przesyłania danych. Polega ona na zmianie dzielnika częstotliwości wg tabeli poniżej

Table 18-5. Relationship Between SCK and the Oscillator Frequency

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

SPSR – SPI Status Register – Rejestr kontrolny. Służy do kontroli stanu.

Bit	7	6	5	4	3	2	1	0	
0x2D (0x4D)	**SPIF**	**WCOL**	**–**	**–**	**–**	**–**	**–**	**SPI2X**	**SPSR**
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- SPIF – flaga sygnalizująca zakończenie wysyłania danych poprzez ustawienie jej na 1. Powoduje ona skok do procedury przerwania, a następnie jej wyzerowanie. Stan 1 na tej fladze oznacza również stan zajętości – do tego rejestru można zapisywać dane tylko wtedy, gdy nie jest ona ustawiona
- WCOL – flaga kolizji. Jest ustawiana na 1, gdy rejestr danych SPDR został zmieniony podczas transmisji danych. Jej wyzerowanie następuje pod odczycie SPDR.
- SPI2X – wpisanie tutaj logicznej jedynki spowoduje podwojenie zegara taktowania SCK

SPDR – SPI Data Register – Rejestr danych. Wykorzystywany do przechowywania odbieranych/wysyłanych danych.

Bit	7	6	5	4	3	2	1	0	
0x2E (0x4E)	**MSB**							**LSB**	**SPDR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

Transfer jest inicjowany poprzez wpisanie danych do tego rejestru.

Inicjalizacja SPI na Arduino następuje po wywołaniu metody *begin()* na obiekcie klasy **SPIClass**

```
void SPIClass::begin()
{
	digitalWrite(SS, HIGH);
	pinMode(SS, OUTPUT);
	SPCR |= _BV(MSTR);
	SPCR |= _BV(SPE);
	pinMode(SCK, OUTPUT);
	pinMode(MOSI, OUTPUT);
}
```

Wpisanie logicznej jedynki na pin SS oraz ustawienie go w trybie wyjścia jest przygotowaniem na ustawienie SPI w tryb mastera. Następnie następuje ustawienie SPI w tryb master oraz włączenie SPI. Na końcu piny SCK oraz MOSI zostają ustawione w tryb wyjścia.

6. Analiza skutków awarii

Awaria	Prawdopodobieństwo	Znaczenie	Automatyczne wykrywanie	Reakcja
Awaria serwomotora		Krytyczne	(Teoretyczne) Sprawdzić czy Attach() zwraca 0	Brak (funkcjonalność mocno ograniczona, możliwe tylko wyświetlanie informacji na wyświetlaczu)
Awaria enkodera rotacyjnego		Średnie	(Teoretyczne) ustawienie countera sprawdzającego czy nastąpiła zmiana stanu enkodera.	Ograniczenie funkcji modułu tylko do naśladowania, bez możliwości zmiany na cokolwiek innego.
Awaria wyświetlacza		Średnie	Sprawdzenie czy inicjalizacja wyświetlacz się powiodła czy nie.	Ograniczenie funkcji modułu tylko do naśladowania, bez możliwości zmiany na cokolwiek innego.
Czujnik ugięcia		Krytyczne	Brak	Brak (funkcjonalność mocno ograniczona, możliwe tylko wyświetlanie informacji na wyświetlaczu i zagranie w papier, kamień, nożyce, bez zaczytywania danych z rękawiczki)
Moduł transmisji radiowej		Krytyczne	Brak	Brak (praktycznie zerowa funkcjonalność, możliwość jedynie zaczytywania informacji z rękawiczki na wyświetlaczu)
Rozkalibrowanie czujnika ugięcia		Średnie	(Teoretyczne) Auto kalibracja czujników przy każdym uruchomieniu	Przy każdym uruchomieniu będziemy prosić użytkownika o kalibrację czujników, najpierw dla wyprostowanych palców, a później dla zgiętych.

7. Wnioski

Do opisów rejestrów i bibliotek Arduino wykorzystaliśmy datasheet do ATmega328P [7].

8. Bibliografia

[1] „Hand and Forarm – InMoov”. http://inmoov.fr/hand-and-forarm/ (udostępniono cze. 11, 2021).

[2] „KEYES Rotary encoder module KY-040.pdf”. Udostępniono: cze. 12, 2021. [Online]. Dostępne na: https://ir.edu.pl/sites/default/files/KEYES%20Rotary%20encoder%20module%20KY-040.pdf

[3] „FLEXSENSOR(REVA1)-1129347.pdf”. Udostępniono: cze. 12, 2021. [Online]. Dostępne na: https://pl.mouser.com/datasheet/2/830/FLEXSENSOR(REVA1)-1129347.pdf

[4] „MG996R_Tower-Pro.pdf”. Udostępniono: cze. 12, 2021. [Online]. Dostępne na: https://www.electronicoscaldas.com/datasheet/MG996R_Tower-Pro.pdf

[5] „Digital Pins | Arduino”. https://www.arduino.cc/en/Tutorial/Foundations/DigitalPins (udostępniono cze. 13, 2021).

[6] „Arduino/wiring_analog.c at ide-1.5.x · arduino/Arduino · GitHub”. https://github.com/arduino/Arduino/blob/ide-1.5.x/hardware/arduino/avr/cores/arduino/wiring_analog.c (udostępniono cze. 13, 2021).

[7] „Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf”. Udostępniono: cze. 13, 2021. [Online]. Dostępne na: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf