SPRAWOZDANIE			PROSZĘ PODAĆ NR GRUPY:					
			ZIISS1	3	5	1	2	IO

IMIĘ	NAZWISKO	Temat ćwiczenia zgodny z wykazem tematów:	PONIŻEJ PROSZĘ PODAĆ TERMIN ZAJĘĆ:						ROK:
MICHAŁ	WARSZAWSKI	**Podstawy programowania mikroprocesorów Dekodery**	PN	WT	SR	CZ	PT	SB	ND	2023 r.
			GODZINA ROZPOCZĘCIA ZAJĘĆ:							11 : 30

UWAGA !!! Wypełniamy tylko białe pola. W **punkcie 1**, proszę zakreślić odpowiednie pola i podać godzinę w której odbywają się zajęcia, zgodnie z planem zajęć.

Wprowadzenie teoretyczne:

Opisz rodzaje kodów spotykanych w elektronice cyfrowej Binarny, Gray'a Johnsona, Heksadecymalny,, Aikena, (będzie Ci to potrzebne w dalszej części zadania).

- Kod Binarny:

Kod binarny stanowi fundamentalny system liczbowy w elektronice cyfrowej, bazujący na dwóch symbolach: 0 i 1. Każda cyfra w tym systemie reprezentuje bit, najmniejszą jednostkę informacyjną w komputerze. Kody binarne stanowią podstawę wszystkich operacji wykonywanych przez komputery, gdzie instrukcje i dane są reprezentowane w postaci binarnej.

- Kod Gray'a:

Kod Gray'a, znany również jako kod jednokierunkowy, to metoda przedstawiania liczb, w której dwie sąsiadujące wartości różnią się tylko jednym bitem. Jest to użyteczne w sytuacjach, gdzie istotne jest minimalizowanie błędów przy zmianie jednej wartości na drugą. Przykładowo, dla liczby 6 (110 w kodzie binarnym), kod Gray'a to 10.

- Kod Johnsona:

Kod Johnsona, zwany także kodem balansowanym, to rodzaj kodu liniowego, w którym sekwencja bitów przemieszcza się symetrycznie względem pewnego punktu. Powszechnie stosowany w transmisji danych i rejestrach przesuwnych, przykłady to kod NRZ (Non-Return-to-Zero) i kod FM (Frequency Modulation).

- Kod Heksadecymalny:

Kod heksadecymalny to system liczbowy oparty na 16 symbolach, 0-9 oraz A-F (reprezentujących liczby dziesiętne od 10 do 15). Powszechnie używany w programowaniu i reprezentacji kolorów w grafice komputerowej ze względu na swoją krótką formę i łatwość konwersji na system binarny.

- Kod Aikena:

Kod Aikena, znany również jako kod jednostkowy Aikena, to specyficzny kod binarny stosowany w konstrukcji komputerów. W odróżnieniu od standardowego kodu binarnego, gdzie każdy bit reprezentuje wartość logiczną, kod Aikena wprowadza dodatkowy bit kontroli. Kombinacja bitu danych i bitu kontroli umożliwia wykrywanie błędów, co jest istotne w systemach, gdzie precyzja i niezawodność są kluczowe.

Analiza różnych rodzajów kodów ukazuje ich zastosowania w elektronice cyfrowej. Kod binarny, mimo bycia podstawowym, jest zazwyczaj nieczytelny dla ludzi, co skłania do stosowania bardziej zwięzłych reprezentacji, np. kodu Gray'a, szczególnie w obszarach, gdzie minimalizacja błędów jest kluczowa. Kod Johnsona jest używany w komunikacji danych do utrzymania równowagi między stanami logicznymi, a kod heksadecymalny jest popularny w programowaniu i reprezentacji kolorów ze względu na swoją kompaktową formę. Natomiast kod Aikena wyróżnia się przez wprowadzenie bitu kontroli, co zwiększa odporność na błędy transmisji danych, co ma szczególne znaczenie w systemach, gdzie precyzja i niezawodność są kluczowe.

Zadanie 1

W programie Simulide zaprojektuj system mikroprocesorowy zamieniający kod binarny na kod wyświetlacza siedmiosegmentowego

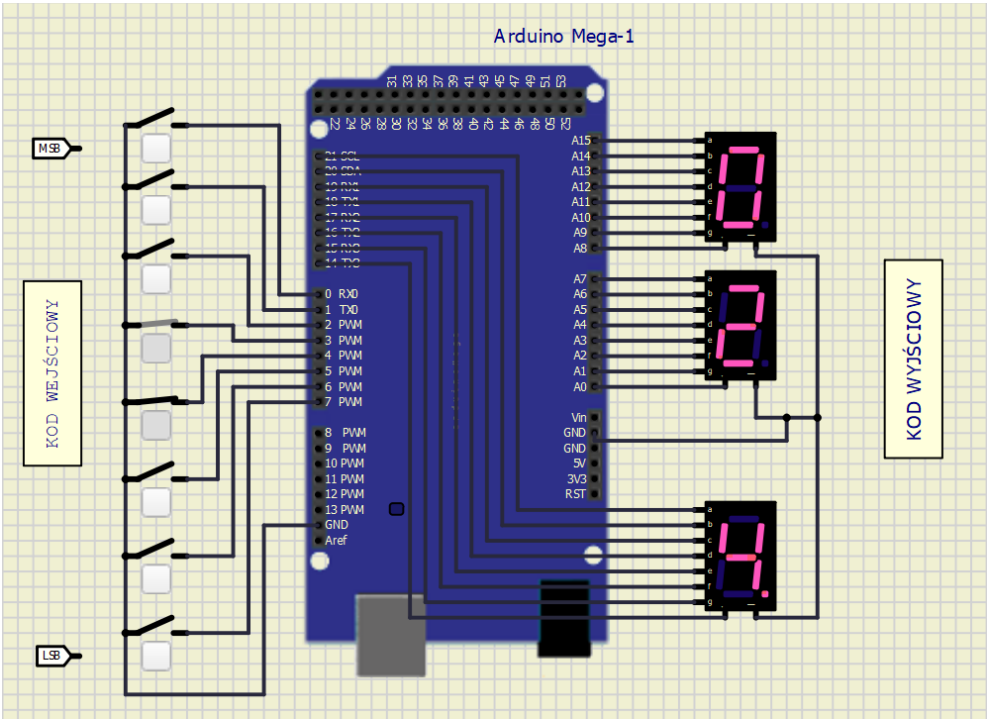
W przypadku gdy suma liczb występujących w Twoim numerze indeksu daje liczbę na wyświetlaczu „Dziesiątek” zapal kropkę a w przypadku liczb nieparzystych zapal kropkę na wyświetlaczu: „Jedności”

Listing programu:

```
#include <stdlib.h>

// Definicje pinów dla mikrokontrolera i wyświetlacza siedmiosegmentowego
const int inputPins[] = {0, 1, 2, 3, 4, 5, 6, 7};
const int hundretPins[] = {A15, A14, A13, A12, A11, A10, A9, A8};
const int decimalPins[] = {A7, A6, A5, A4, A3, A2, A1, A0};
const int unityPins[] = {21, 20, 19, 18, 17, 16, 15, 14};
const int numbers[] = {1, 2, 4, 8, 16, 32, 64, 128};

void setup() {
	// Inicjalizacja pinów wejściowych i wyjściowych
	for (int i = 0; i < 8; ++i) {
		pinMode(inputPins[i], INPUT_PULLUP);
	}

	for (int i = 0; i < 8; ++i) {
		pinMode(hundretPins[i], OUTPUT);
		digitalWrite(hundretPins[i], LOW);
		pinMode(decimalPins[i], OUTPUT);
		digitalWrite(decimalPins[i], LOW);
		pinMode(unityPins[i], OUTPUT);
		digitalWrite(unityPins[i], LOW);
	}

	// Sprawdzenie, czy suma indeksu jest parzysta i ustawienie kropki
	if((2+2+3+1+9+4)%2==0){
		digitalWrite(decimalPins[7], HIGH);
	}
	else{
		digitalWrite(unityPins[7], HIGH);
	}
}

void loop() {
	int number = getSignals();
	int lp = sizeof(numbers) / sizeof(numbers[0]);
	char digits[lp];

	// Podział liczby dziesiętnej na trzy cyfry w tablicy
	int tempNumber = number;
	for (int i = 0; i < lp; ++i) {
		digits[i] = tempNumber % 10;
		tempNumber /= 10;
	}

	// Wyświetlanie cyfr na poszczególnych wyświetlaczach
	displayDigit(digits[0], unityPins);
	displayDigit(digits[1], decimalPins);
	displayDigit(digits[2], hundretPins);
}

void displayDigit(int digit, int pins[]) {
	// Mapowanie cyfr na reprezentację siedmiosegmentową
	switch (digit) {
		case 0:
			digitalWrite(pins[0], HIGH);
			digitalWrite(pins[1], HIGH);
			digitalWrite(pins[2], HIGH);
			digitalWrite(pins[3], HIGH);
			digitalWrite(pins[4], HIGH);
			digitalWrite(pins[5], HIGH);
			digitalWrite(pins[6], LOW);
		break;
		case 1:
			digitalWrite(pins[0], LOW);
			digitalWrite(pins[1], HIGH);
			digitalWrite(pins[2], HIGH);
			digitalWrite(pins[3], LOW);
			digitalWrite(pins[4], LOW);
			digitalWrite(pins[5], LOW);
			digitalWrite(pins[6], LOW);
		break;
		case 2:
			digitalWrite(pins[0], HIGH);
```

```
                        digitalWrite(pins[1], HIGH);
                        digitalWrite(pins[2], LOW);
                        digitalWrite(pins[3], HIGH);
                        digitalWrite(pins[4], HIGH);
                        digitalWrite(pins[5], LOW);
                        digitalWrite(pins[6], HIGH);
                break;
                case 3:
                        digitalWrite(pins[0], HIGH);
                        digitalWrite(pins[1], HIGH);
                        digitalWrite(pins[2], HIGH);
                        digitalWrite(pins[3], HIGH);
                        digitalWrite(pins[4], LOW);
                        digitalWrite(pins[5], LOW);
                        digitalWrite(pins[6], HIGH);
                break;
                case 4:
                        digitalWrite(pins[0], LOW);
                        digitalWrite(pins[1], HIGH);
                        digitalWrite(pins[2], HIGH);
                        digitalWrite(pins[3], LOW);
                        digitalWrite(pins[4], LOW);
                        digitalWrite(pins[5], HIGH);
                        digitalWrite(pins[6], HIGH);
                break;
                case 5:
                        digitalWrite(pins[0], HIGH);
                        digitalWrite(pins[1], LOW);
                        digitalWrite(pins[2], HIGH);
                        digitalWrite(pins[3], HIGH);
                        digitalWrite(pins[4], LOW);
                        digitalWrite(pins[5], HIGH);
                        digitalWrite(pins[6], HIGH);
                break;
                case 6:
                        digitalWrite(pins[0], HIGH);
                        digitalWrite(pins[1], LOW);
                        digitalWrite(pins[2], HIGH);
                        digitalWrite(pins[3], HIGH);
                        digitalWrite(pins[4], HIGH);
                        digitalWrite(pins[5], HIGH);
                        digitalWrite(pins[6], HIGH);
                break;
                case 7:
                        digitalWrite(pins[0], HIGH);
                        digitalWrite(pins[1], HIGH);
                        digitalWrite(pins[2], HIGH);
                        digitalWrite(pins[3], LOW);
                        digitalWrite(pins[4], LOW);
                        digitalWrite(pins[5], LOW);
                        digitalWrite(pins[6], LOW);
                break;
                case 8:
                        digitalWrite(pins[0], HIGH);
                        digitalWrite(pins[1], HIGH);
                        digitalWrite(pins[2], HIGH);
                        digitalWrite(pins[3], HIGH);
                        digitalWrite(pins[4], HIGH);
                        digitalWrite(pins[5], HIGH);
                        digitalWrite(pins[6], HIGH);
                break;
                case 9:
                        digitalWrite(pins[0], HIGH);
                        digitalWrite(pins[1], HIGH);
                        digitalWrite(pins[2], HIGH);
                        digitalWrite(pins[3], HIGH);
                        digitalWrite(pins[4], LOW);
                        digitalWrite(pins[5], HIGH);
                        digitalWrite(pins[6], HIGH);
                break;
        }
}

int getSignals() {
        // Odczytywanie sygnałów z pinów wejściowych i konwersja na liczbę
        int result = 0;
        for (int i = 0; i < 8; ++i) {
                if (digitalRead(inputPins[i]) == LOW) {
                        result += numbers[i];
                }
        }
        return result;
}
```

Schemat układu:



Wnioski:

Program przetwarzaa kod binarny na kod wyświetlacza siedmiosegmentowego. Program odczytuje sygnały z pinów wejściowych, konwertuje je na liczbę i wyświetla trzy cyfry na odpowiednich wyświetlaczach, mapując każdą cyfrę na reprezentację siedmiosegmentową. Demonstruje praktyczne zastosowanie systemu mikroprocesorowego do obsługi wyświetlacza siedmiosegmentowego.

Zadanie 2

W programie Simulide zaprojektuj system mikroprocesorowy zamieniający kod Graya na kod wyświetlacza siedmiosegmentowego

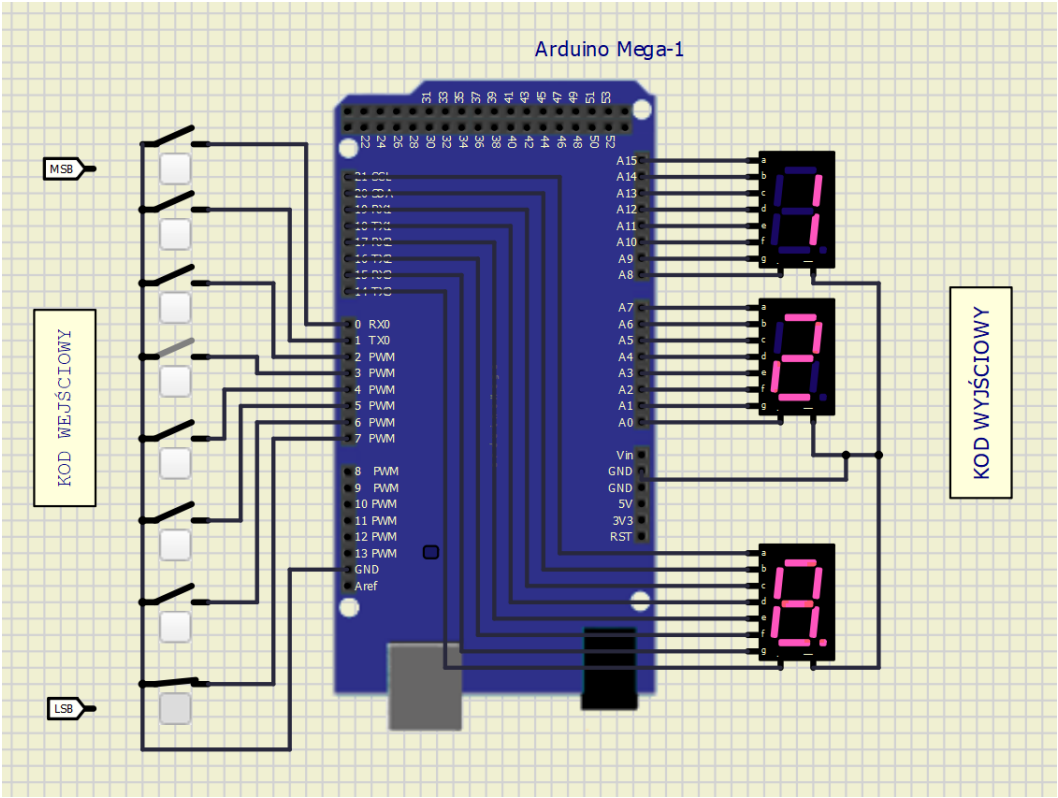
W przypadku liczb parzystych występujących w Twoim numerze indeksu na wyświetlaczu „Dziesiątek” zapal kropkę a w przypadku liczb nieparzystych zapal kropkę na wyświetlaczu: „Jedności”

Listing programu:

```
#include <stdlib.h>

// Definicje pinów dla wejść, wyjść i segmentów wyświetlacza
const int inputPins[] = {0, 1, 2, 3, 4, 5, 6, 7};
const int hundretPins[] = {A15, A14, A13, A12, A11, A10, A9, A8};
const int decimalPins[] = {A7, A6, A5, A4, A3, A2, A1, A0};
const int unityPins[] = {21, 20, 19, 18, 17, 16, 15, 14};
const int numbers[] = {128, 64, 32, 16, 8, 4, 2, 1};

void setup() {
	// Konfiguracja pinów wejściowych z rezystorem pull-up
	for (int i = 0; i < 8; ++i) {
		pinMode(inputPins[i], INPUT_PULLUP);
	}
	// Konfiguracja pinów wyjściowych dla setek, dziesiątek i jedności
	for (int i = 0; i < 8; ++i) {
		pinMode(hundretPins[i], OUTPUT);
		digitalWrite(hundretPins[i], LOW);
		pinMode(decimalPins[i], OUTPUT);
		digitalWrite(decimalPins[i], LOW);
		pinMode(unityPins[i], OUTPUT);
		digitalWrite(unityPins[i], LOW);
	}
}

void loop() {
	// Odczyt sygnałów wejściowych
```

```
	int number = getSignals();
	int lp = sizeof(numbers) / sizeof(numbers[0]);
	char digits[lp];
	// Rozdzielenie liczby na poszczególne cyfry
	int tempNumber = number;
	for (int i = 0; i < lp; ++i) {
		digits[i] = tempNumber % 10;
		tempNumber /= 10;
	}

	// Wybór, który segment wyświetlacza będzie miał włączoną kropkę
	if((2+2+3+1+9+4)%2==0){
		displayDigit(digits[0], unityPins, false);
		displayDigit(digits[1], decimalPins, true);
	}
	else{
		displayDigit(digits[0], unityPins, true);
		displayDigit(digits[1], decimalPins, false);
	}
	displayDigit(digits[2], hundretPins, false);
}

void displayDigit(int digit, int pins[], bool showDot) {
	// Mapa bitowa dla każdej cyfry
	const byte digitMap[] = {
		B00111111, // 0
		B00000110, // 1
		B01011011, // 2
		B01001111, // 3
		B01100110, // 4
		B01101101, // 5
		B01111101, // 6
		B00000111, // 7
		B01111111, // 8
		B01101111 // 9
	};
	// Wyświetlanie cyfry na odpowiednich wyświetlaczach
	for (int i = 0; i < 8; ++i) {
		digitalWrite(pins[i], bitRead(digitMap[digit], i));
	}
	// Włączenie kropki dla dziesiątek, jeśli showDot jest true
	for (int i = 0; i < 8; ++i) {
		if (i == 7 && showDot) {
			digitalWrite(pins[i], HIGH);
		} else {
			digitalWrite(pins[i], bitRead(digitMap[digit], i));
		}
	}
}

// Odczyt sygnałów wejściowych i przekształcenie ich na liczbę
int getSignals() {
	int gray[8];
	int binary[8];
	int result = 0;
	// odczytanie kodu Gray'a
	for (int i = 0; i < 8; ++i) {
		if (digitalRead(inputPins[i]) == LOW) {
			gray[i] = 1;
		}
	}
	// zmiana kodu Gray'a na binarny
	binary[0] = gray[0];
	for (int i = 1; i < 8; ++i) {
		if (binary[i-1] == gray[i]){
			binary[i] = 0;
		}
		else{
			binary[i] = 1;
		}
	}
	// zamiana kodu binarnego na liczbę dziesiętną
	for (int i = 0; i < 8; ++i) {
		if (binary[i] == 1) {
			result += numbers[i];
		}
	}
	return result;
}
```

Schemat układu:



Wnioski:

Program Simulide konwertuje kod Graya na kod binarny a następnie na liczbę dziesiętną i ostatecznie na kod wyświetlacza siedmiosegmentowego. Użyto pinów wejściowych i wyjściowych, a działanie obejmuje odczyt sygnałów i rozdzielenie cyfr. Program reaguje na parzystość/nieparzystość liczb, zapalając kropkę na odpowiednim segmencie. Program wykorzystuje mapę bitową dla cyfr binarnych.

Zadanie 3

Rozbuduj układ tak aby przyciski podłączone do pinów od 31 do 35, które mogą przyjmować w kodzie binarnym wartości wartości 0, 1, 2, 3, 4, 5 zaprogramuj tak aby wartość ustawiona na górnym przycisku transformowała kody:

0 – transformował kod binarny na kod dziesiętny wyświetlając go na wyświetlaczu siedmiosegmentowym (ten program już masz gotowy)

1 – transformował kod Graya wyświetlał kod dziesiętny na wyświetlaczu siedmiosegmentowym (ten program masz gotowy)

2 – transformował kod binarny i wyświetlał na wyświetlaczu kod szesnastkowy (jedności pierwszy półbajt, dziesiątki drugi półbajt)

3 – transformował kod Graya i wyświetlał go na wyświetlaczu w postaci kodu szesnastkowego.

4 - transformował kod AIKENA wyświetlał go na wyświetlaczu w postaci kodu dziesiętnego

5 – transformował kod AIKENA i wyświetlał go na wyświetlaczu w postaci kodu szesnastkowego.

6 – transformował kod JOHNSONA i wyświetlał go na wyświetlaczu w postaci kodu dziesiętnego

7- transformował kod JOHNSONA i wyświetlał go na wyświetlaczu w postaci kodu szesnastkoweg

Listing programu:

```
#include <stdlib.h>

// Definicje pinów dla wejść, wyjśćs i segmentów wyświetlacza
const int inputPins[] = {0, 1, 2, 3, 4, 5, 6, 7};
const int hundredPins[] = {A15, A14, A13, A12, A11, A10, A9, A8};
const int decimalPins[] = {A7, A6, A5, A4, A3, A2, A1, A0};
const int unitsPins[] = {21, 20, 19, 18, 17, 16, 15, 14};
const int powersOfTwo[] = {128, 64, 32, 16, 8, 4, 2, 1};
```

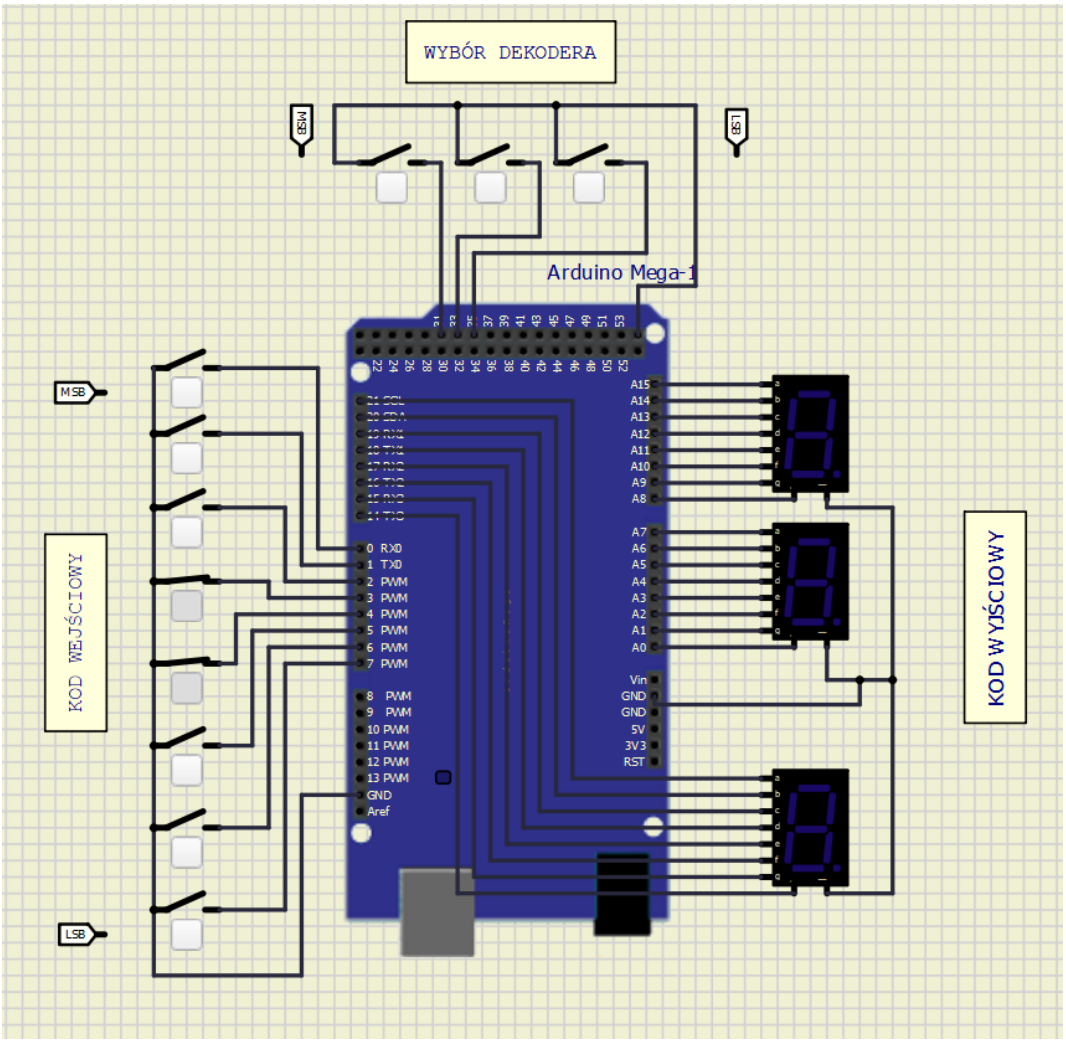
```
const int decoderPins[] = {31, 33, 35};

void setup()
{
    // Konfiguracja pinów wejściowych z rezystorem pull-up
    for (int i = 0; i < 8; ++i)
    {
        pinMode(inputPins[i], INPUT_PULLUP);
    }

    // Konfiguracja pinów wejściowych z rezystorem pull-up
    for (int i = 0; i < 3; ++i)
    {
        pinMode(decoderPins[i], INPUT_PULLUP);
    }

    // Konfiguracja pinów wyjściowych dla wyświetlaczy
    for (int i = 0; i < 8; ++i)
    {
        pinMode(hundredPins[i], OUTPUT);
        digitalWrite(hundredPins[i], LOW);
        pinMode(decimalPins[i], OUTPUT);
        digitalWrite(decimalPins[i], LOW);
        pinMode(unitsPins[i], OUTPUT);
        digitalWrite(unitsPins[i], LOW);
    }
}

void loop()
{
    // Odczyt sygnałów sterujących
    int number = getDecoderPins();


    // Wyświetlanie odpowiednich danych
    switch (number)
    {
    // Kod Binarny -> Kod Dziesiętny
    case 0:;
        displayDecimalNumber(getBinaryCode());
        break;
    // Kod Gray'a -> Kod Dziesiętny
    case 1:
        displayDecimalNumber(getGrayCode());
        break;
    // Kod Binarny -> Kod Szesnastkowy
    case 2:
        displayHexNumber(convertDecimalToHexadecimal(getBinaryCode()));
        break;
    // Kod Gray'a -> Kod Szesnastkowy
    case 3:
        displayHexNumber(getGrayCode());
        break;
    // Kod Aikena -> Kod Dziesiętny
    case 4:
        displayDecimalNumber(getAikenCode());
        break;
    // Kod Aikena -> Kod Szesnastkowy
    case 5:
        displayHexNumber(getJohnsonCode());
        break;
    // Kod Johnsona -> Kod Dziesiętny
    case 6:
        displayDecimalNumber(getJohnsonCode());
        break;
    // Kod Johnsona -> Kod Szesnastkowy
    case 7:
        displayHexNumber(getJohnsonCode());
        break;
    default:
        break;
    }
}

// --> POBRANIE SYGNAŁÓW STERUJĄCYCH
int getDecoderPins()
{
    int result = 0;
    for (int i = 0; i < 3; ++i)
    {
        if (digitalRead(decoderPins[i]) == LOW)
        {
            result += powersOfTwo[i + 5];
        }
```

```
    }
    return result;
}

// --> POBIERANIE DANYCH WEJŚCIOWYCH W ODPOMIEDNIM KODZIE
int getBinaryCode()
{
    int decimalNumber = 0;
    for (int i = 0; i < 8; ++i)
    {
        if (digitalRead(inputPins[i]) == LOW)
        {
            decimalNumber += powersOfTwo[i];
        }
    }
    return decimalNumber;
}

int getGrayCode()
{
    int gray[8];
    // Odczytanie kodu Gray'a
    for (int i = 0; i < 8; ++i)
    {
        if (digitalRead(inputPins[i]) == LOW)
        {
            gray[i] = 1;
        }
        else
        {
            gray[i] = 0;
        }
    }

    // Zmiana kodu Gray'a na binarny
    int binary[8];
    binary[0] = gray[0];
    for (int i = 1; i < 8; ++i)
    {
        if (binary[i - 1] == gray[i])
        {
            binary[i] = 0;
        }
        else
        {
            binary[i] = 1;
        }
    }

    // Zamiana kodu binarnego na liczbę dziesiętną
    return convertBinaryToDecimal(binary);
}

int getAikenCode()
{
    return 0;
}

int getJohnsonCode()
{
    return 0;
}

// --> ZAMIANA SYSTEMÓW LICZBOWYCH
int convertBinaryToDecimal(int binary[])
{
    int decimalNumber = 0;
    for (int i = 0; i < 8; ++i)
    {
        if (binary[i] == 1)
        {
            decimalNumber += powersOfTwo[i];
        }
    }

    return decimalNumber;
}

char *convertDecimalToHexadecimal(int decimalNumber)
{
    int quotient;
    int i = 1, j, temp;
    char hexNumber[3];
    quotient = decimalNumber;
```

```
    while (quotient != 0)
    {
        temp = quotient % 16;
        if (temp < 10)
        {
            temp = temp + 48;
        }
        else
        {
            temp = temp + 55;
        }

        hexNumber[i++] = temp;
        quotient = quotient / 16;
    }

    char resultArray[3];
    resultArray[0] = hexNumber[2];
    resultArray[1] = hexNumber[1];
    resultArray[2] = hexNumber[0];

    return resultArray;
}

// --> WYŚWIETLANIE DANYCH NA WYŚWIETLACZU
void displayDecimalNumber(int decimalNumber)
{
    int hundreds = decimalNumber / 100;
    int decimals = (decimalNumber % 100) / 10;
    int units = decimalNumber % 10;

    // Mapa bitowa dla każdej cyfry
    const byte digitDecMap[] = {
        B00111111, // 0
        B00000110, // 1
        B01011011, // 2
        B01001111, // 3
        B01100110, // 4
        B01101101, // 5
        B01111101, // 6
        B00000111, // 7
        B01111111, // 8
        B01101111, // 9
    };

    // Wyświetlanie cyfry na odpowiednich wyświetlaczach
    for (int i = 0; i < 8; ++i)
    {
        digitalWrite(hundredPins[i], bitRead(digitDecMap[hundreds], i));
        digitalWrite(decimalPins[i], bitRead(digitDecMap[decimals], i));
        digitalWrite(unitsPins[i], bitRead(digitDecMap[units], i));
    }
}

void displayHexNumber(char *hexNumber)
{
    // Mapa bitowa dla każdej cyfry szesnastkowej
    const byte digitHexMap[] = {
        B01111110, // 0
        B00110000, // 1
        B01101101, // 2
        B01111001, // 3
        B00110011, // 4
        B01011011, // 5
        B01011111, // 6
        B01110000, // 7
        B01111111, // 8
        B01111011, // 9
        B01110111, // A
        B00011111, // B
        B01001110, // C
        B00111101, // D
        B01001111, // E
        B01000111  // F
    };

    // Wyświetlanie cyfry szesnastkowej na odpowiednich wyświetlaczach
    for (int i = 0; i < 3; ++i)
    {
        // Sprawdzenie, czy wartość jest poprawną cyfrą szesnastkową
        if ((hexNumber[i] >= '0' && hexNumber[i] <= '9') || (hexNumber[i] >= 'A' && hexNumber[i] <= 'F'))
        {
            // Konwersja cyfry szesnastkowej na odpowiadającą jej liczbę
```

```
            int index = (hexNumber[i] >= '0' && hexNumber[i] <= '9') ? (hexNumber[i] - '0') : (hexNumber[i] - 'A' +
10);

            // Wyświetlanie poszczególnych bitów cyfry szesnastkowej na wyświetlaczach
            digitalWrite(hundredPins[i], bitRead(digitHexMap[index], 6));
            digitalWrite(decimalPins[i], bitRead(digitHexMap[index], 5));
            digitalWrite(unitsPins[i], bitRead(digitHexMap[index], 4));
        }
        else
        {
            // Jeśli wartość nie jest poprawną cyfrą szesnastkową, wyłącz wyświetlacz
            digitalWrite(hundredPins[i], LOW);
            digitalWrite(decimalPins[i], LOW);
            digitalWrite(unitsPins[i], LOW);
        }
    }
}
```

Schemat układu:



Wnioski:

Układ obejmuje przyciski od 31 do 35, przyjmujące wartości binarne 0-7. Programy dla tych przycisków realizują różne transformacje kodów, m.in. binarnego na dziesiętny, Graya na dziesiętny, binarnego na szesnastkowy, Graya na szesnastkowy, AIKENA na dziesiętny, AIKENA na szesnastkowy, JOHNSONA na dziesiętny, JOHNSONA na szesnastkowy. Transformacje wyświetlane są na siedmiosegmentowych wyświetlaczach.