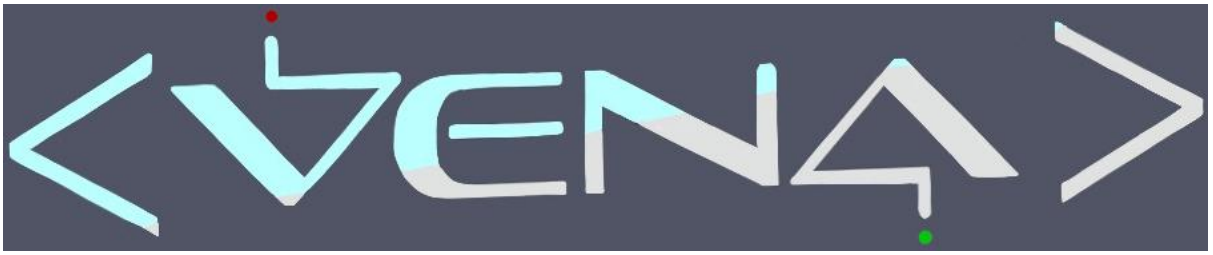



# Projekt “VENA”
# Sterownik PLC

**Autorzy:**
**Adam Więcek**
**Kacper Nowicki**

SCAN ME

## Opis projektu:

Sterowniki Vena stanowią programowalne kontrolery logiczne, specjalnie zaprojektowane do niezawodnej pracy w warunkach przemysłowych, nawet pod stałym obciążeniem. Programowane w języku tekstowym C++, są dostępne dla programistów w sposób intuicyjny. Charakteryzują się one uniwersalnością, umożliwiającą ich zastosowanie nie tylko w przemyśle, lecz także w instalacjach domowych. Ta wszechstronność produktów Vena pozwala użytkownikom na dostosowanie sterownika do swoich potrzeb poprzez konfigurację odpowiednich trybów wejścia i wyjścia (w przypadku wersji Pilot 3) lub wykorzystanie możliwości języka C++ do stworzenia dedykowanego programu zgodnego z wymaganiami.

## Vena Opis układu elektronicznego:

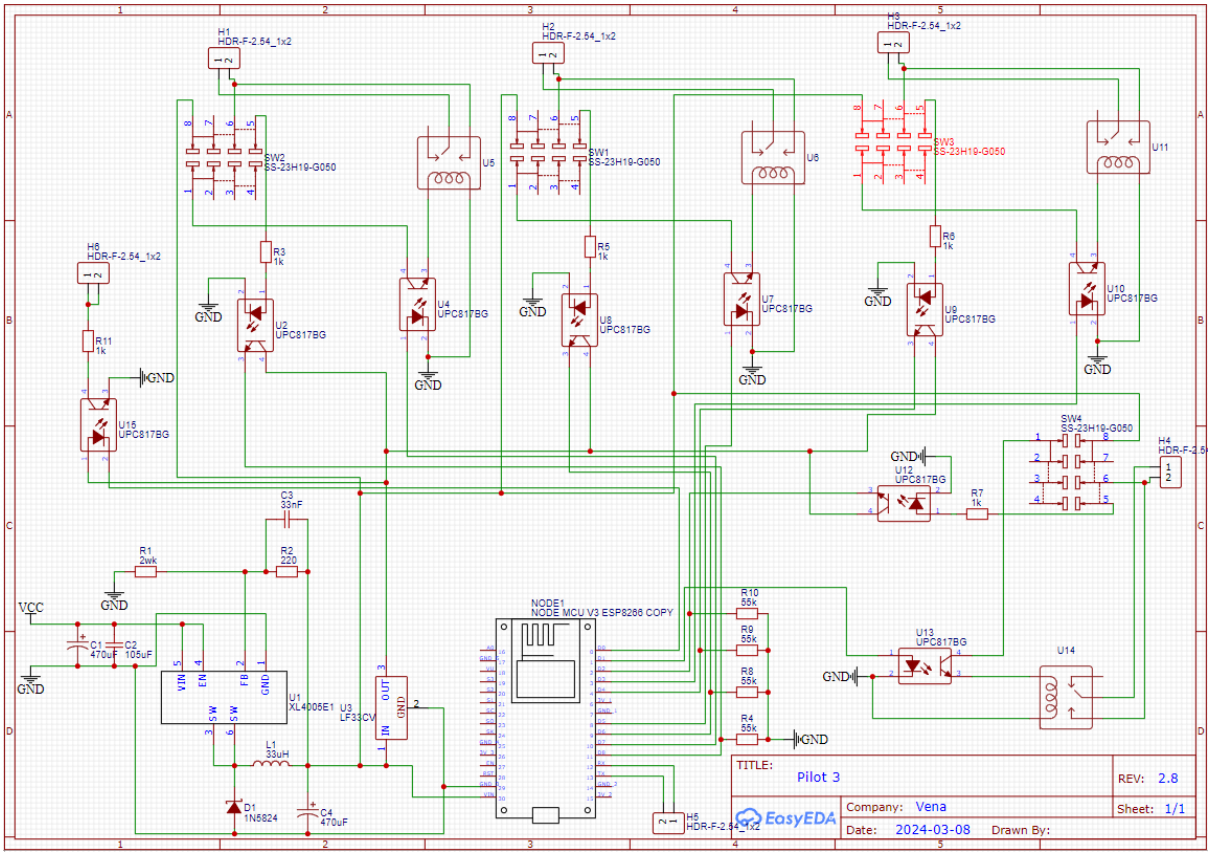
Wszystkie moduły sterujące Vena zostały wyposażone w galwanicznie izolowane wejścia, wykorzystujące transoptory w celu separacji między sterownikiem a sterowanym układem. Wyjścia tych modułów są oparte na przekaźnikach, umożliwiających przepływ prądu o maksymalnej wartości do 10A przy 30VDC oraz 10A przy 250VAC. Projekt sterowników został zoptymalizowany w celu minimalizacji elementów generujących ciepło, takich jak rezystory, co prowadzi do redukcji strat prądowych oraz obniżenia zużycia energii przez sam sterownik. Systemy zasilające wykorzystywane w produktach Vena oparte są na przetwornicach impulsowych, co sprawia, że są one bardziej efektywne energetycznie w porównaniu do układów liniowych, oraz generują mniej ciepła, co pozwala na ich niezawodne i długotrwałe użytkowanie.

## **Specyfikacja prototypów**:

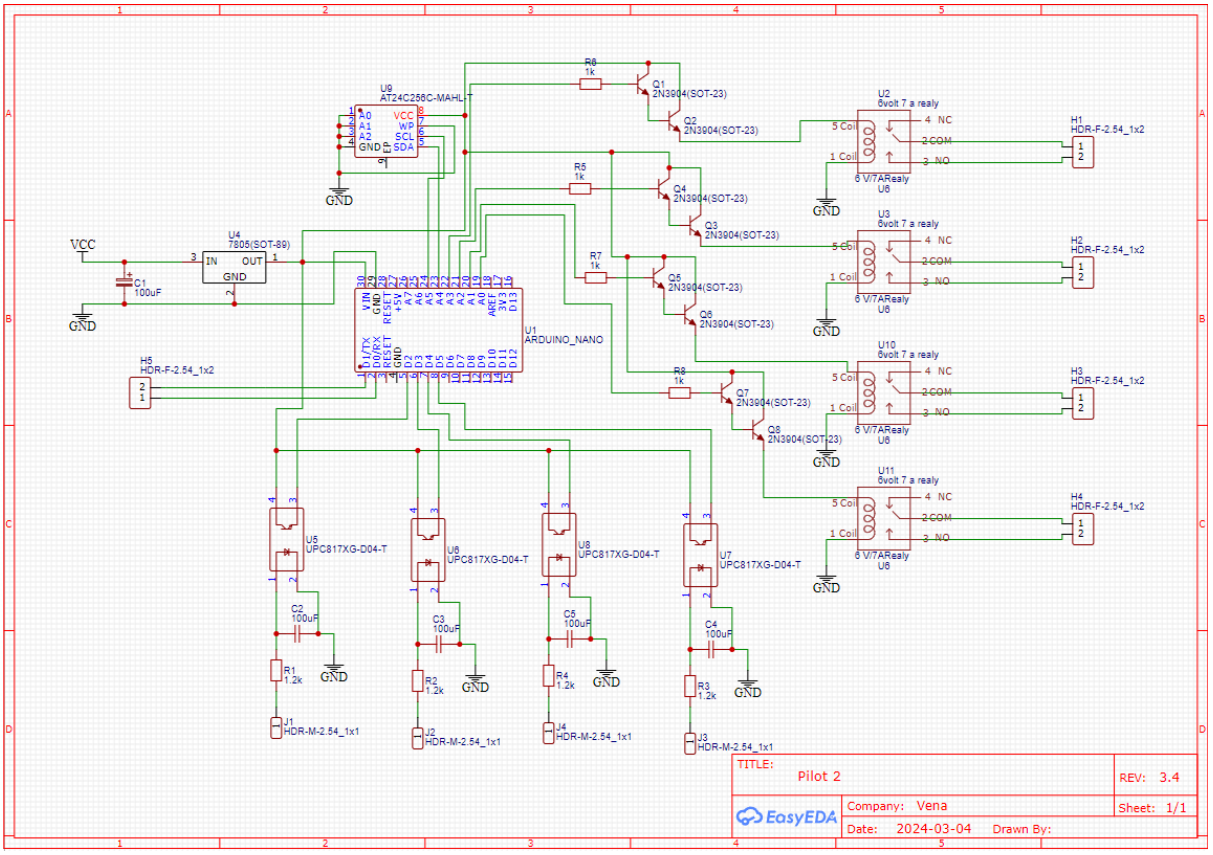
- **Pilot 2**
    - Procesor: ATmega328P
        - Architektura - 8Bit,
        - Prędkość procesora - 16Mhz,
        - Pamięć flash - 32KB,
        - Pamięć EEPROM - 1KB (zewnętrzny moduł rozszerzający pamięć EEPROM 24C256 posiadający 256 KBitów pamięci),
        - Pamięć SRAM - 2KB,
    - Wejścia/Wyjścia - 4 przyłącza pełniące funkcje wejść (przyjmują sygnały binarne o wartości 24V) oraz 4 pełniące funkcje wyjść (podają sygnały binarne o wartości 24V).
    - Zasilanie - 24VDC poprzez zasilacz liniowy.
    - Komunikacja - specjalny port do komunikacji szeregowej umożliwiający skomunikowanie sterownika z innymi układami Vena.

- **Pilot 3**
    - Procesor ESP-8266
        - Architektura - 32Bit,
        - Prędkość procesora - 80Mhz,
        - Pamięć flash - 4MB,
        - Pamięc EEPROM - 4KB,
        - Pamięć SRAM - 64KB,
    - Wejścia/Wyjścia - 4 przyłącza z możliwością dostosowania ich do potrzeb użytkownika i zmiany trybu pracy na wejście lub wyjście w zależności od pozycji przełącznika suwakowego.
    - Zasilanie - 24VDC za pomocą przetwornicy impulsowej pozwalającej na długotrwałe działanie bez wydzielania ciepła oraz strat energii.
    - Komunikacja - Wyposażony w port do komunikacji szeregowej umożliwiający skomunikowanie sterownika z innymi układami Vena.
    - Przycisk stop - dedykowane przyłącze dla przycisku stop wyłączający cały układ.

# Pilot 3 schemat elektryczny:




# Pilot 2 schemat elektryczny:



## Vena opis programowania:

Sterowniki Vena są programowane przy użyciu języka tekstowego C++ za pomocą środowiska programistycznego Arduino IDE oraz dedykowanej biblioteki vena.h. Wykorzystanie powszechnego języka C++ do tworzenia oprogramowania dla naszych sterowników przekłada się na ich wszechstronność i ułatwia dostęp do niezbędnej wiedzy potrzebnej do stworzenia programu spełniającego oczekiwania użytkownika. Stworzenie własnej biblioteki vena.h usprawniło proces programowania, uczyniło go bardziej intuicyjnym i logicznym, a także skróciło czas tworzenia programów. Poprzez odzwierciedlenie w bibliotece vena.h komend znanych z języków LADDER lub FBD, proces nauki programowania przy użyciu naszej biblioteki będzie bardziej intuicyjny dla osób zaznajomionych z tymi językami.

## Vena opis techniczny programu:

- Biblioteki:
    - Arduino.h: Biblioteka standardowa Arduino, zapewniająca podstawowe funkcje obsługi płytki.
    - EEPROM.h: Biblioteka obsługi pamięci EEPROM, umożliwiająca zapis i odczyt danych z pamięci nieulotnej.
    - SoftwareSerial.h: Biblioteka do obsługi komunikacji szeregowej z użyciem programowego UART-a.
- Struktury:
    - Klasa Vena zawiera struktury Pin, Timer i ImpulseGenerator do przechowywania informacji o pinach, timerach i generatorach impulsów.

- Metody:
    - Klasa Vena zawiera metody do obsługi pinów (check(), out()), generatorów impulsów (generateImpulse()), operacji logicznych (l_and(), l_or(), itd.), obsługi zmiennych dodatkowych (setMarker(), getMarker(), itd.), obsługi timerów (startTimer(), checkTimer()), obsługi komunikacji szeregowej (sendMsg(), readMsg()), oraz dodatkowe metody pomocnicze (reset(), handleInterruptStatic()).
    - Metody te umożliwiają sprawdzanie stanu pinów, generowanie impulsów, wykonywanie operacji logicznych, manipulowanie zmiennymi dodatkowymi, zarządzanie timerami oraz komunikację szeregową.
- Komunikacja szeregowa:
    - Klasa umożliwia komunikację szeregową poprzez interfejs UART z użyciem biblioteki SoftwareSerial.
    - Metody sendMsg() i readMsg() służą odpowiednio do wysyłania i odbierania komunikatów.
- Obsługa pamięci EEPROM:
    - Klasa zapewnia możliwość zapisu i odczytu zmiennych dodatkowych do pamięci EEPROM za pomocą metod setExtraVar() i getExtraVar().
    - Zdefiniowano metodę findVariableAddress() do znajdowania adresu zapisu dla danej zmiennej w pamięci EEPROM.
- Obsługa przerwań:
    - Klasa obsługuje przerwania za pomocą metody handleInterruptStatic() oraz zmiennych stopFunction i stopFunction.
- Wykorzystanie millis():
    - Wykorzystanie millis() pozwoliło ograniczyć wszystkie opóźnienia do prędkości procesora.

## Dokumentacja komend:

Nazwa Funkcji: l_or
Parametry Wejściowe: bool x - pierwszy operand, bool y - drugi operand
Wartość Zwracana: bool - wynik operacji logicznego OR
Opis: Wykonuje operację logicznego OR na dwóch operandach x i y. Zwraca true, jeśli przynajmniej jeden z operandów jest true, w przeciwnym razie zwraca false.

Nazwa Funkcji: l_nand
Parametry Wejściowe: bool x - pierwszy operand, bool y - drugi operand
Wartość Zwracana: bool - wynik operacji logicznego NAND
Opis: Wykonuje operację logicznego NAND na dwóch operandach x i y. Zwraca false, jeśli oba operandy są true, w przeciwnym razie zwraca true.

Nazwa Funkcji: l_nor
Parametry Wejściowe: bool x - pierwszy operand, bool y - drugi operand
Wartość Zwracana: bool - wynik operacji logicznego NOR
Opis: Wykonuje operację logicznego NOR na dwóch operandach x i y. Zwraca true, jeśli oba operandy są false, w przeciwnym razie zwraca false.

Nazwa Funkcji: l_xor
Parametry Wejściowe: bool x - pierwszy operand, bool y - drugi operand
Wartość Zwracana: bool - wynik operacji logicznego XOR
Opis: Wykonuje operację logicznego XOR na dwóch operandach x i y. Zwraca true, jeśli operandy są różne, w przeciwnym razie zwraca false.

Nazwa Funkcji: l_xnor
Parametry Wejściowe: bool x - pierwszy operand, bool y - drugi operand
Wartość Zwracana: bool - wynik operacji logicznego XNOR
Opis: Wykonuje operację logicznego XNOR na dwóch operandach x i y. Zwraca true, jeśli operandy są takie same, w przeciwnym razie zwraca false.

Nazwa Funkcji: l_not
Parametry Wejściowe: bool x - operand
Wartość Zwracana: bool - wynik operacji logicznego NOT
Opis: Wykonuje operację logicznego NOT na operandzie x. Zwraca true, jeśli operand x jest false, w przeciwnym razie zwraca ture.

Nazwa Funkcji: setMarker
Parametry Wejściowe: String name - nazwa markera, int value - wartość do ustawienia dla markera
Wartość Zwracana: Brak
Opis: Ustawia wartość określonego markera.

Nazwa Funkcji: getMarker
Parametry Wejściowe: const char* name - nazwa markera
Wartość Zwracana: int - wartość określonego markera
Opis: Pobiera wartość określonego markera.

Nazwa Funkcji: setCounter
Parametry Wejściowe: String name - nazwa licznika, int value - wartość do ustawienia dla licznika
Wartość Zwracana: Brak
Opis: Ustawia wartość określonego licznika.

Nazwa Funkcji: getCounter
Parametry Wejściowe: const char* name - nazwa licznika
Wartość Zwracana: long int - wartość określonego licznika
Opis: Pobiera wartość określonego licznika.

Nazwa Funkcji: addToCounter
Parametry Wejściowe: const char* name - nazwa licznika, int value - wartość do dodania do licznika
Wartość Zwracana: Brak
Opis: Dodaje określoną wartość do określonego licznika.

Nazwa Funkcji: checkCounter
Parametry Wejściowe: const char* name - nazwa licznika
Wartość Zwracana: bool - zwraca true, jeśli licznik zgadza się z jego wartością, false w przeciwnym przypadku
Opis: Sprawdza, czy licznik zgadza się z jego wartością.

Nazwa Funkcji: resetCounter
Parametry Wejściowe: const char* name - nazwa licznika
Wartość Zwracana: Brak
Opis: Resetuje określony licznik.

Nazwa Funkcji: startTimer
Parametry Wejściowe: String name - nazwa timera, unsigned long duration - czas trwania timera
Wartość Zwracana: Brak
Opis: Rozpoczyna odliczanie timera o określonym czasie trwania.

Nazwa Funkcji: checkTimer
Parametry Wejściowe: String name - nazwa timera, unsigned long duration (opcjonalne) - określa alternatywny czas trwania
Wartość Zwracana: bool - zwraca true, jeśli timer został zakończony, false w przeciwnym przypadku
Opis: Sprawdza, czy timer o określonej nazwie został zakończony lub sprawdza czy okreslony czas minoł.

Nazwa Funkcji: showInOut
Parametry Wejściowe: bool inputs - określa czy wyświetlić stany pinów wejściowych, bool outputs - określa czy wyświetlić stany pinów wyjściowych
Wartość Zwracana: Brak
Opis: Wyświetla stany pinów wejściowych i/lub wyjściowych na porcei szeregowym.

Nazwa Funkcji: sendMsg
Parametry Wejściowe: const char* message - wiadomość do wysłania
Wartość Zwracana: Brak
Opis: Wysyła wiadomość przez komunikację szeregową do drugiego podłaczonego sterownika.

Nazwa Funkcji: readMsg
Parametry Wejściowe: Brak
Wartość Zwracana: String - odczytana wiadomość
Opis: Odczytuje wiadomość z komunikacji szeregowej od drugiego podłaczonego sterownika.

Nazwa Funkcji: reset
Parametry Wejściowe: Brak
Wartość Zwracana: Brak
Opis: Resetuje sterownik.