SCLL1/230V
Programowalny sterownik
8xPWM 4xWE
wersja 230V

Instrukcja obsługi

RoHS

Producent: EL KOSMITO Rafał Majewski
Ul. Kościuszki 21
68-320 Jasień
NIP 928-192-12-96
REGON 080936699

Kontakt: www.elkosmito.pl
info@elkosmito.pl

Spis treści

Opis ogólny....3
Cechy sterownika SCLL1....3
Możliwości zastosowania....4
Parametry techniczne....4
Wyprowadzenia i podłączenie....4
Programowanie układu....6
Wstęp do programowania....6
Menu sterownika SCLL1....7
Zasada działania wejść....7
Przeglądanie programu....7
Opcje menu programu....8
Tworzenie programów....8
Grupa I komend – sterowanie automatami PWM....9
START....9
STOP....9
CZEKAJ....10
USTAW....10
Grupa II komend – zmiana parametru Zegar automatów....10
ZEGAR....10
ZEGAR-....11
ZEGAR+....11
Grupa III komend – komendy oczekiwania....11
PAUZA....12
SYNCHR....12
Grupa I, II, III - przykłady....12
Grupa IV – komendy sterujące programem....15
POWT 15
KONIEC POWT....15
RESET....15
URUCHOM....15
KONIEC PROGRAMU....16
NIC NIE RÓB....16
Grupa IV – przykłady....16
Grupa V – komendy obsługujące wejścia/wyjścia....21
DEZAKT....21
TEST....21
ZERUJ....22
TEST STAN....22
!TEST STAN....22
CZAS....22
MAT....23
STOP_CZAS....23
START_CZAS....23
RESET_CZAS....24
JEZELI....24
WE_PROG....24
WE_DOMYSLNE....24
PWM_DOMYSLNE....24
Grupa V – przykłady....24
Najczęściej popełniane błędy w programie....37
Na zakończenie....38
Uwagi!....39

Opis ogólny.

Firma EL KOSMITO opracowała bardzo zaawansowany sterownik z 8 wyjściami PWM i 4 wejściami, które mogą spełniać bardzo różne funkcje. Wyjścia PWM w tym przypadku to wyprostowane napięcie sieciowe 230V (po wyprostowaniu osiąga ponad 300V). Napięcie to może posłużyć do sterowania oświetleniem LED różnych producentów pod warunkiem, że jest ono kompatybilne z naszym układem. Kompatybilność została opisana w dalszej części instrukcji. Można także zasilić zwykłe lampki żarówkowe pod warunkiem, że ich moc nie przekracza 55W na kanał.

Każde z wyjść PWM może się płynnie rozjaśniać, wygaszać oraz obsługuje do 55W. Sterownik jest programowalny. Oznacza to, że sekwencje wyjść możemy ustawić w bardzo szerokim zakresie. Można zbudować nawet 80 programów składających się z 1000 instrukcji każdy. Jest to gigantyczna ilość! Dla przykładu uzyskanie efektu, że 8 wyjść płynnie włącza się od wyjścia 1 do wyjścia 8 to zaledwie 16 instrukcji. Zaprogramowanie całej pamięci jest praktycznie nie możliwe. Obsługiwane funkcje pozwalają nie tylko sterować wyjściami, ale pozwalają także na budowanie prostych pętli (dzięki czemu niektóre części programu mogą się powtarzać), synchronizowanie, sterowanie tempem przetwarzania, uruchamianie programów jako nowych programów lub podprogramów, proste sterowanie przepływem programu (na podstawie wejść uruchamiać się mogą różne wyjścia), wbudowane 48 liczników czasu pozwala odmierzać czas lub przechowywać w licznikach wyniki prostych operacji matematycznych i porównywać je między sobą. Razem układ obsługuje 29 komend!

Sterownik SCLL1 wykonany jest w obudowie na szynę DIN. Na przednim panelu zamontowano wyświetlacz alfanumeryczny mogący wyświetlić po 16 znaków w dwóch liniach. Interfejs nie jest skomplikowany, jest w języku polskim a niniejsza instrukcja mamy nadzieję, że pomoże odkryć wszystkie tajniki programowania naszego układu. Dodatkowo dodano przydatne funkcje takie jak możliwość kopiowania istniejących programów, możliwość kopiowania tylko wybranych części itp. Powinno to ułatwić napisanie programu. Oczywiście należy się liczyć z tym, że dostępność 5 przycisków sterujących na panelu nie daje takich możliwości pisania programów jak w przypadku klawiatury, nie mniej jednak w tym przypadku to powinno wystarczyć z nawiązką.

Przykładowe programy mogłyby działać tak:

1) Możemy zaprogramować 4 różne sekwencje i zrobić układ 4 przycisków, którymi w dowolnych momencie będziemy przełączali daną funkcję na inną
2) Możemy zbudować zaawansowane sterowanie lampek choinkowych lub innych ozdobnych sekwencji/animacji
3) Możemy zaprogramować standardowe płynne rozjaśnianie do lampek umieszczonych przy schodach w ten sposób, że jeśli czujnik na początku schodów wykryje ruch to rozświetli lampki z dołu, a jak wykryje na końcu to rozświetli kolejno z góry
4) Możemy zablokować w punkcie 2) aby po uruchomieniu jednej sekwencji nie włączyła się żadna inna w tym czasie dopóki dana się nie wykona
5) Możemy zaprogramować tak, że na zewnątrz wyprowadzamy przyciski NASTĘPNY i POPRZEDNI i tymi przyciskami przełączamy wyjścia (widzimy tutaj, że układ pozwala uzyskać interaktywność pewnego rodzaju)
6) Odpowiednie sterowanie pozwala stworzyć np. dekoder BCD wyświetlacza 7-segmentowego (oczywiście złożonego z segmentów na 230V) działający w ten sposób, że na podstawie stanu wejść uruchamia się odpowiednia cyfra w taki sposób, że najpierw wygasza się poprzednia, a potem włącza nowa
7) Budować włączniki bistabilne, astabilne, takie i takie, ponieważ możliwa jest interakcja i odczytywanie stanu wejść w programach

Jak widzimy na powyższych przykładach funkcjonalność układu jest bardzo duża. Nie można w związku z tym wypisać wszystkich zastosowań, bo stosunkowo rozbudowana możliwość budowania kodu programu pozwala użytkownikowi na bardzo wiele.

Cechy sterownika SCLL1

- Zasilanie bezpośrednio 230V
- Separacja napięcia sieciowego dla zewnętrznych wejść (zewnętrzne wejścia nie są podpięte do napięcia sieciowego, działają niskonapięciowo)
- Wbudowany filtr sieciowy i zabezpieczenie przeciwprzepięciowe
- Wbudowane układy ograniczające zakłócenia na każdym z ośmiu kanałów PWM
- Bezpiecznik umieszczony z boku obudowy
- 8 wyjść PWM oraz 4 wejścia, które można dodatkowo oprogramować
- Wyjścia PWM z wyprostowanego napięcia sieciowego (powyżej 300V), przy czym PWM ogranicza skuteczne napięcie do 230V
- Sterowanie lampkami choinkowymi zwykłymi do 55W na kanał lub lampkami LED kompatybilnymi z układem
- Maksymalne obciążenie dla całego układu 2A
- Mikroprocesorowe sterowanie
- Wysoka częstotliwość PWM około 200Hz zapewnia, że diody LED i lampki nie widać jak migają (niektóre typy lampek świecą ładniej niż standardowo)
- Zaawansowane możliwości konfiguracji i budowania programów
- Dosyć łatwy interfejs w języku polskim
- Dostępne funkcje kopiowania całych programów i ich części dzięki czemu pisanie kodu może być prostsze
- Wyświetlacz na panelu oraz 5 przycisków sterujących
- Programowanie układu nie wymaga komputera, wszystko programujemy z panelu
- Obudowa na standardową szynę DIN35mm
- Możliwość budowania do 80 programów po 1000 instrukcji każda
- 29 komend sterujących układem
- wbudowane 48 liczników czasu (16 liczników odmierzających czas w ms, 16 w sekundach i 16 w minutach)
- możliwość wykonywania operacji matematycznych na licznikach czasu i wyjściach
- możliwość wykorzystania liczników czasu jako zmiennych

- Każdy kanał PWM może być ustawiony na poziom od 0 do 180
- Możliwość budowania pętli w programie (części programu, które się mają powtarzać)
- Wbudowane funkcje do prostego sterowania przepływem programu, czyli możliwość uruchamiania z jednego programu, drugiego, możliwość sprawdzania stanu wejść i przełączania programu na podstawie ich ustawienia itp.
- Dwustopniowy system regulacji płynnego rozjaśniania i wygaszania
- Wbudowana pamięć (wg producenta pamięci) o trwałości 20 lat oraz z możliwością wykonania min 100 000 cykli zapisu
- Szybkie przetwarzanie instrukcji kodu (około 5ms na instrukcję)
- Opcje śledzenia kodu ułatwiające sprawdzenie jak przetwarza się program

Możliwości zastosowania

Układ można wykorzystać do:

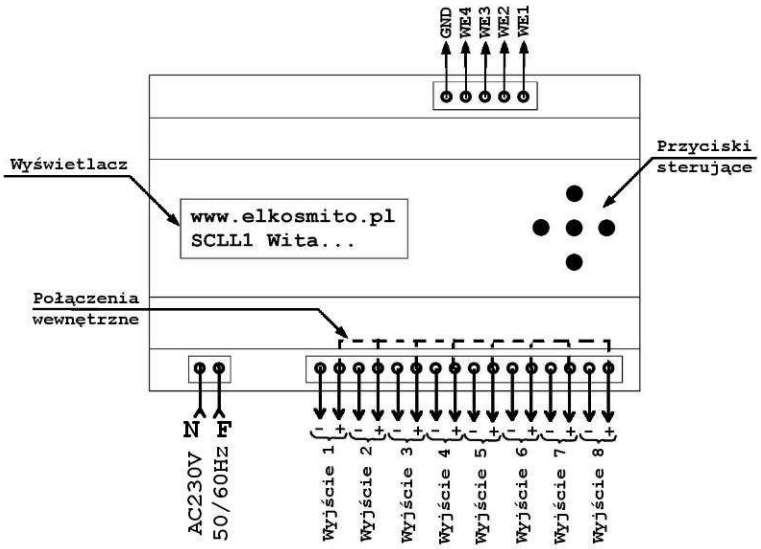
- płynny start diod LED lub lamp (lampek) kompatybilnych z układem
- budowanie układów płynnego startu oświetlenia
- stosunkowo zaawansowane układy włączające i sterujące z możliwością interakcji z użytkownikiem z zewnątrz
- układy sterowania bistabilnego i astabilnego
- sterowniki reklam świetlnych, podświetleń, efektów świetlnych
- odpowiednio sterowany układ może włączać styczniki a więc można budować układy przełączające reagujące na krańcówki itp.

Parametry techniczne

- zasilanie: AC230V 50-60Hz
- pobór prądu (bez wliczania obciążenia): poniżej 2W, przy czym przy wyłączonym wyświetlaczu około 1W
- wyjścia: PWM z napięcia sieciowego 230V wyprostowanego (stałe ok. 310-330V) ograniczone współczynnikiem wypełnienia do napięcia skutecznego 230V
- maksymalne obciążenie każdego wyjścia: 0,25A skutecznego (ok. 55W)
- maksymalne obciążenie wszystkich wyjść (suma): 2A skutecznego (ok. 460W)
- liczba programowalnych wyjść PWM: 8
- liczba programowalnych wejść: 4
- regulacja PWM od 0 do 180, co odpowiada regulacji od 0 do 100% stałego napięcia skutecznego ok 230V
- liczba instrukcji obsługiwanych przy budowie kodu: 29
- wbudowane liczniki czasu/zmienne: 48 (16 liczników odmierzających czas w ms, 16 w sekundach i 16 w minutach)
- wymiary 150x89x63mm
- obudowa na szynę DIN35mm
- bezpiecznik na wejściu 2,5A
- zakres temperatur pracy: -20°C do 40°C
- zalecany zakres temperatur programowania: 10°C do 40°C
- wykonanie IP00

Wyprowadzenia i podłączenie

Na Rys. 1 pokazano wyprowadzenia sterownika. Wyświetlacz i przyciski sterujące oraz podłączenie zasilania układu nie wymagają wyjaśnienia. Zaznaczono gdzie znajdują się połączenia wewnętrzne w układzie. Widzimy w takim razie, że plusy + są wszędzie połączone w środku układu, a więc każdy plus + wyjściowy to ten sam plus. Minusy – natomiast są sterowane i włączane zgodnie z zasadą działania układów PWM.



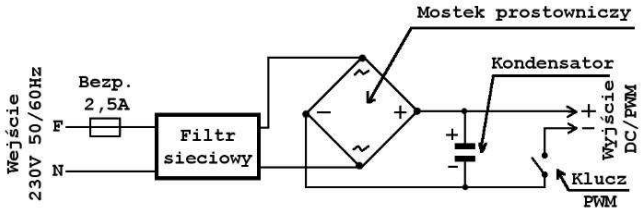
Rys. 1. Wyprowadzenia sterownika

Schemat obwodu wejściowego oraz jednego wyjściowego pokazano na Rys. 2 Widzimy, że napięcie sieciowe najpierw przez bezpiecznik trafia na filtr sieciowy, a następnie przez mostek prostowniczy staje się napięciem wyprostowanym. Kondensator filtruje to napięcie, a klucz PWM powoduje, że wyjście jest sterowane i można regulować współczynnik wypełnienia. Z Rys. 2 przy znajomości elektroniki w minimalnym stopniu można wysnuć wniosek, jakie obwody będą kompatybilne z naszym sterownikiem i można podłączać je pod wyjścia. Przykłady takich obwodów poprawnych i niepoprawnych pokazano na Rys. 3.

Na Rys. 3a pokazano podstawowy wariant podłączenia do wyjścia. Obciążeniem w tym przypadku jest standardowa żarówka na 230V o mocy nie przekraczającej dopuszczalnych parametrów sterownika SCLL1. Nie ma w tym przypadku znaczenie +/- i możemy podpiąć dowolnie.

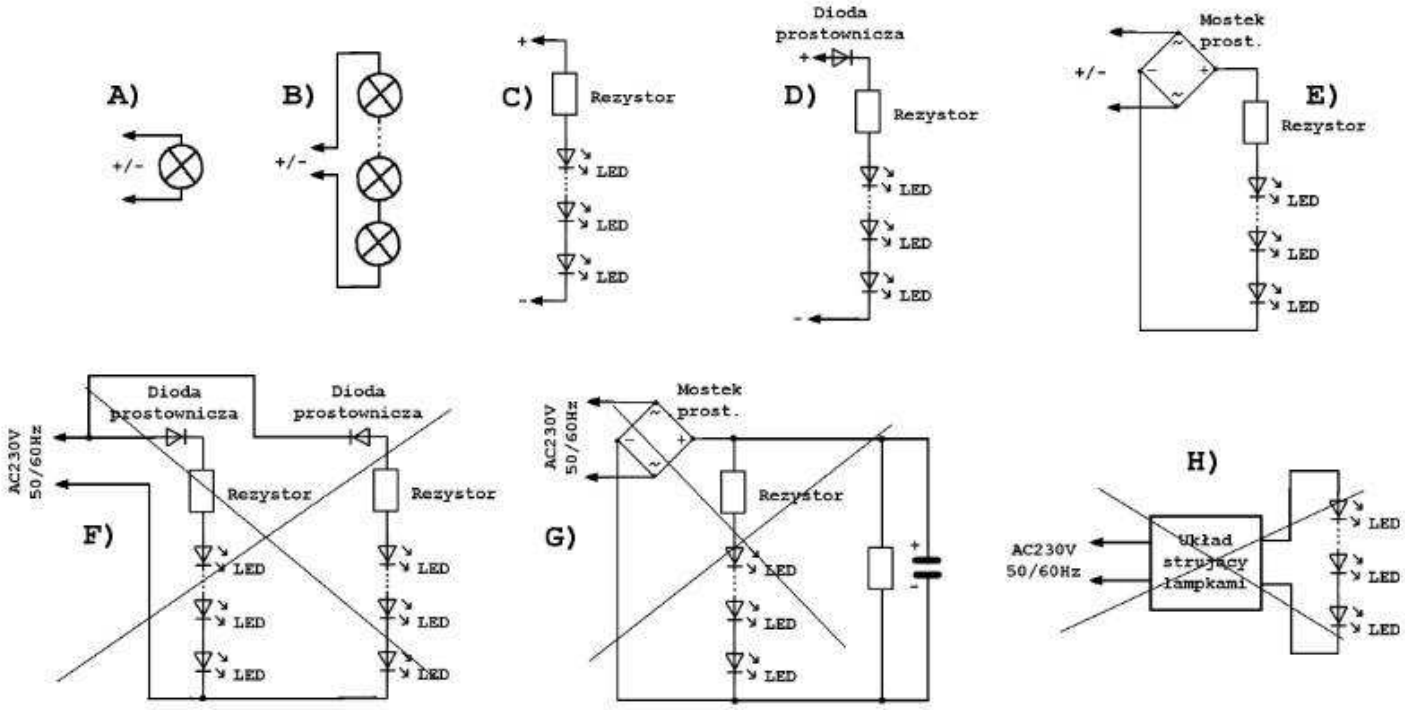
Na Rys. 3b pokazano wariant z podłączeniem kilku żarówek podłączonych szeregowo. Jeśli taki zestaw działa prawidłowo z sieci 230V oraz nie przekroczono dopuszczalnej mocy to sterownik SCLL1 może posłużyć do sterowania takim kompletem lampek.

Na Rys. 3c pokazano zestaw diod połączonych w szeregu wraz z rezystorem ograniczającym prąd. Taki układ raczej nie jest dostępny bezpośrednio na rynku, ale można go wykonać we własnym zakresie zachowując zasady bezpieczeństwa. Ważne jest aby zachować dokładnie polaryzację +/- przy podłączaniu. Odwrotne podłączenie może uszkodzić diody. Przy wykonaniu takiego układu należy przyjąć zasadę, że diody mają świecić bez przekraczania ich dopuszczalnego prądu a także rezystor powinien być odpowiedniej mocy. Dokładne zasady doboru tych elementów nie są przedmiotem niniejszej instrukcji.



Rys. 2.Schemat wejściowy wraz z jednym kluczem PWM

Na Rys. 3d pokazano ten sam wariant co na Rys. 3c, ale za to taki wariant może być dostępny na rynku u niektórych producentów. Taki układ może być dostosowany do pracy z napięcia sieci 230V 50/60Hz. Można go podłączyć do sterownika, ale



Rys. 3: Różne prawidłowe i nieprawidłowe warianty podłączenia dokładnie opisane w treści instrukcji

NIE WOLNO w takim przypadku przekraczać połowy maksymalnego współczynnika wypełnienia jaki można uzyskać w sterowniku czyli wartości 90. Ten wariant można również zbudować samemu podobnie jak z Rys. 3c. Wtedy można uzyskać pełen zakres regulacji współczynnika wypełnienia po odpowiednim dobraniu elementów. Niezależnie od tego czy kupny czy wykonany samemu, to należy zachować prawidłową polaryzację zasilania +/- aby układ działał poprawnie. Plusem tej wersji w stosunku do Rys. 3c jest zabezpieczenie przed odwrotnym podłączeniem (na rys. jest to dioda prostownicza).

Na Rys. 3e pokazano wariant dostępny na rynku. Układ składa się z mostka prostowniczego oraz diod i rezystora ograniczającego prąd. W tym wariancie nie ma znaczenia podłączenie +/-. W obu przypadkach podłączenie będzie prawidłowe. Taka wersja zestawu lampek LED dostosowanych do pracy z napięcia sieciowego 230V będzie pracowała poprawnie z układem SCLL1.

Na Rys. 3f pokazano wariant, który może być dostępny na rynku. Zestaw taki będzie działał poprawnie z siecią 230V (o ile na takie napięcie jest wyprodukowany) ale nie będzie poprawnie działał z układem SCLL1. Będzie paliła się tylko jedna część diod, druga część nie, a na dodatek będą takie same ograniczenia jak w wersji z Rys. 3d.

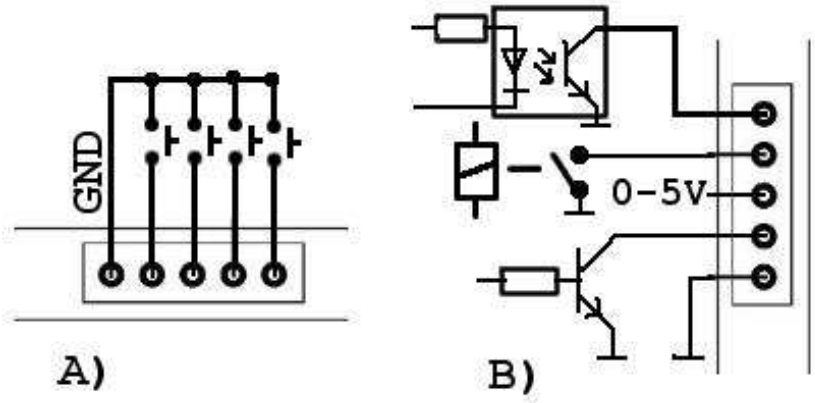
Na Rys. 3g pokazano wariant niepoprawny. Jest on modyfikacją wersji z Rys. 3e. Wadą układu z Rys. 3e jest migotanie diod, która jest widoczna przez wielu ludzi. Zastosowanie kondensatora w wersji Rys. 3g eliminuje problem migotania, ale uniemożliwia pracę takiej wersji ze sterownikiem SCLL1. Warto pamiętać, że jeśli w obwodzie ograniczenia prądu też znajduje się kondensator, to również taki układ nie może pracować ze sterownikiem SCLL1.

Na Rys. 3h pokazano układ lampek posiadających swój własny obwód sterujący. Może to być jakiś zasilacz, może to być układ stabilizacji prądu, może to być układ sterujący lampkami. We wszystkich tych przypadkach takiego zestawu nie należy podłączać do sterownika SCLL1.

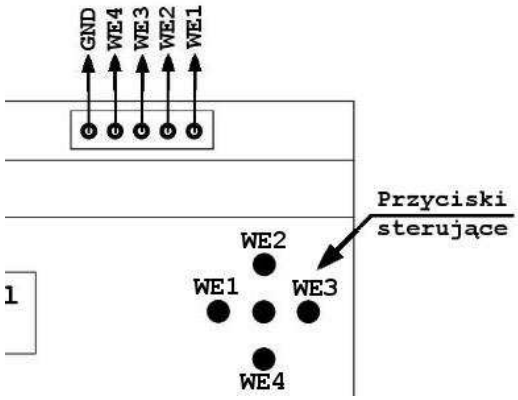
Wejścia działają na zasadzie zwierania do masy (GND) co ilustruje Rys. 4. Standardowo pod wejścia możemy podpiąć

włączniki zwierne tak jak pokazano na Rys. 4a. Opcjonalnie, bardziej zaawansowani użytkownicy mogą sterować układem na inne sposoby zaprezentowane na Rys. 4b. Widzimy tam, że wejście 4 podłączono poprzez transoptor (optoizolator) dzięki czemu sterowanie może odbywać się z separacją galwaniczną. Wejście 3 włącza i wyłącza przekaźnik, który może być innym sposobem separacji galwanicznej. Wejście 2 możemy sterować z zewnętrznego źródła napięcia od 0 do 5V, gdzie 0 oznacza, że wejście jest aktywne. Wejście 1 pokazuje załączanie przy pomocy tranzystora NPN.

Układ umożliwia sterowanie wejściami bez podłączania zewnętrznych włączników. Może to być bardzo korzystne podczas programowania, gdzie bez konieczności podłączania czegokolwiek pod listwę wejść, możemy sprawdzić jak one działają w danym momencie. Jak to się dzieje? To dość proste. Na panelu znajduje się 5 przycisków z czego 4 z nich są połączone równolegle z wejściami, więc wciskając te przyciski powodujemy, że układ będzie działał tak samo jakbyśmy mieli podłączone włączniki pod wejścia. Który przycisk odpowiada któremu wejściu pokazano na Rys. 5.



Rys. 4: Sterowanie wejściami.
a) standardowe
b) inne



Rys. 5: Przyciski sterujące i powiązanie ich z wejściami

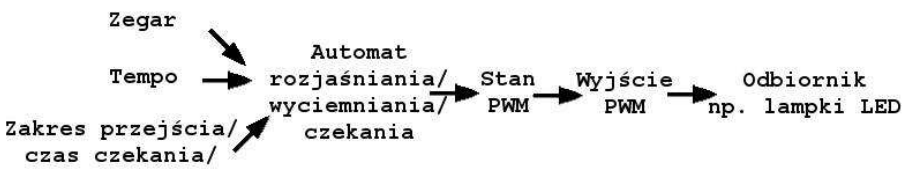
Programowanie układu

Wstęp do programowania

Nie pomijaj tego rozdziału. Jest on bardzo ważny do zrozumienia pracy układu.

Podstawową rzeczą, którą musisz wiedzieć o sterowniku SCLL1 to zakres regulacji wyjścia PWM. Każde z wyjść może być regulowane od 0 do 180, gdzie 0 oznacza, że wyjście jest całkowicie wyłączone, a 180 oznacza, że wyjście jest całkowicie włączone (dokładnie jest to PWM o takim współczynniku wypełnienia aby napięcie skuteczne było równe około 230V). Programista sterownika może ustawić dowolną wartość na wyjściu a także skorzystać z gotowych funkcji płynnego przechodzenia pomiędzy współczynnikami i dzięki temu uzyskać efekt płynnego rozjaśniania/wygaszania.

Aby zrozumieć jak działa układ rozjaśniania i wygaszania trzeba zrozumieć jak działa układ SCLL1 w tym zakresie. Dobrze ilustruje to Rys. 6. Najważniejszym elementem jest "Automat rozjaśniania/wyciemniania/czekania". To właśnie on zajmuje się automatyczną regulacją płynnego włączania i wyłączania wyjścia PWM, dzięki czemu nie musimy tworzyć złożonych programów, aby rozjaśnić jedno wyjście. Automatem można sterować poprzez zmienianie parametrów "Zegar", "Tempo" oraz decydować o zakresach płynnego przejścia lub czasie czekania.



Rys. 6: Zasada działania wyjścia PWM

Szczegóły tych operacji znajdują się w dalszej części instrukcji. Tutaj skupimy się na podstawowej kwestii.

Jak to się dzieje, że układ płynnie rozjaśnia lub wyciemnia?

Każde z wyjście ma swój własny parametr **Zegar** , **Tempo, Zakres przejścia/czas czekania**. Oznacza to, że każde z wyjść możemy skonfigurować osobno. **Zakres przejścia/czas czekania** oraz **Tempo** najczęściej reguluje się w jednej instrukcji, dzięki czemu możemy określić od razu w jakim tempie automat rozjaśni/wyciemni wyjście. Dodatkowo osobne instrukcje służą do zmian parametru **Zegar**. Oba parametry **Zegar** i **Tempo** decydują o szybkości rozjaśniania/wyciemniania. Dlaczego są dwa parametry? Odpowiedź jest dość prosta. Ta sama instrukcja nakazująca rozjaśnienie z określonym parametrem **Tempo** może wykonywać się z różną prędkością. Wystarczy zmienić parametr **Zegar**. Co to może znaczyć praktycznie? Wyobraźmy sobie, że zrobimy jeden program z jakąś ładną sekwencją. Dzięki możliwości zmiany parametru **Zegar** nie musimy pisać 4 różnych programów, aby zrobić, że sekwencja będzie wyświetlała się z różną prędkością w zależności od wciśniętego włącznika. Wystarczy, że zmodyfikujemy tylko parametr **Zegar** dla wszystkich lub tylko wybranych wyjść i uzyskamy zupełnie inne prędkości rozjaśniania nie modyfikując całej

sekwencji.

Ważne jest, żeby zapamiętać jak oblicza się czas wykonywania płynnego rozjaśniania. Podstawową jednostką czasu w układzie jest 1ms (milisekunda) czyli 0,001 sek. W praktyce 1000ms to jest 1 sekunda. Jeśli chcemy aby układ rozjaśnił wyjście od 0 do 180, parametr **Zegar** jest ustawiony na 2, a **Tempo** na 3, to czas płynnego rozjaśnienia będzie równy: (180-0)*2*3*1ms = 180*2*3*1ms=1080ms czyli takie rozjaśnianie potrwa około 1,1 sekundy. Spróbujmy policzyć ile potrwa najszybsze rozjaśnianie od 0 do 180. Oczywiście w takim przypadku **Zegar** jest ustawiony na 1 i **Tempo** również na 1. Wówczas uzyskujemy wynik (180-0)*1*1*1ms=180ms czyli około 1/5 sekundy.

Ile potrwa rozjaśnienie od 80 do 130 jeśli **Zegar=1**, **Tempo=10**? Policzmy: (130-80)*1*10*1ms=500ms=0,5sek.

Widzimy, że policzenie tego czasu jest banalnie proste i nie wymaga specjalnej znajomości matematyki.

Sterownik pozwala także ustawić natychmiast wyjście na określony poziom i wtedy układ nie rozjaśnia/wygasza płynnie tylko natychmiast zmienia stan.

Następną ważną rzeczą, którą musimy poznać jest sposób przetwarzania komend. Program wykonuje się niezależnie od pracy wyjść. A więc jeśli podamy automatowi wszystkie parametry, to nie musimy czekać aż on wypełni swoje zadanie. Program idzie od razu dalej. Oczywiście możemy w dowolnej chwili wymusić czekanie, ale nie jest to konieczne. Dzięki temu jeśli chcemy wykonać płynne rozjaśnianie wszystkich wyjść w różnym tempie wydajemy komendy rozjaśniania jedna po drugiej. Każda z nich wyzwala automat, który zaczyna wykonywać swoją rolę. Po wyzwoleniu sterownik SCLL1 idzie od razu do następnej komendy, a automat robi swoje. Jest to bardzo fajne rozwiązanie.

Menu sterownika SCLL1

Kiedy wyświetlacz jest wyłączony, układ pracuje tak jak został zaprogramowany. Aby wejść do MENU wystarczy wcisnąć środkowy klawisz. Włącza się wtedy podświetlenie wyświetlacza oraz komunikat powitalny. W głównym MENU mamy dostępne następujące funkcje:

1) Wyjście z MENU – wychodzi z MENU i powraca to trybu normalnej pracy
2) Domyślne uruchamianie programu – opcja pozwala wybrać program jaki będzie wykonywał się domyślnie po uruchomieniu sterownika
3) Programowanie – opcje programowania układu, pisanie programów itd.
4) Program wejścia 1 – opcja pozwala wybrać program jaki uruchomi się po aktywowaniu wejścia 1
5) Program wejścia 2 – opcja pozwala wybrać program jaki uruchomi się po aktywowaniu wejścia 2
6) Program wejścia 3 – opcja pozwala wybrać program jaki uruchomi się po aktywowaniu wejścia 3
7) Program wejścia 4 – opcja pozwala wybrać program jaki uruchomi się po aktywowaniu wejścia 4
8) Przywróć ustawienia domyślne i wykasuj pamięć – przywraca ustawienia domyślne i kasuje pamięć. Ustawia również domyślny program o numerze 80, który jest sekwencją testową płynnego rozjaśniania i wygaszania kolejnych wyjść. Program ten można zmienić w dowolnej chwili

W większości przypadków przechodzenie pomiędzy MENU odbywa się przy pomocy przycisków PRAWO/LEWO/GÓRA/DÓŁ. Środkowy przycisk jest wykorzystywany głównie przy programowaniu i spełnia on rolę akceptacji lub wyświetlania dodatkowych opcji.

Aby przechodzić pomiędzy opcjami menu posługujemy się przyciskami GÓRA/DÓŁ. Aby przejść dalej wybieramy przycisk PRAWO, aby cofnąć przycisk LEWO. W niektórych przypadkach przycisk LEWO ma dwie funkcje. Jedna z nich może przesuwać dane na wyświetlaczu, a druga jest to właśnie cofanie do wcześniejszego menu. W takim przypadku aby cofnąć po prostu przytrzymujemy przycisk około 0,5sek.

Posługiwanie się klawiszami nie jest specjalnie skomplikowane. Można popróbować i zobaczyć jak to działa.

Zasada działania wejść

Wejścia w układzie zostały zaprojektowane tak, że mogą działać na impulsy i przetwarzaniem ich zajmuje się sam sterownik, który uruchamia odpowiednie programy z pamięci zapisane w konfiguracji lub jeśli wykorzystujemy ich stan w trakcie przetwarzania programu to mogą pracować jako wejścia impulsowe lub jako wejścia NO/NC (można sprawdzić w jakim są aktualnie stanie). Służą do tego wszystkiego proste funkcje.

Domyślnie układ kontroluje wejścia z poziomu głównego zarządzania SCLL1. Oznacza to, że wejścia te po pojawieniu się impulsu włączają taki program jak jest ustawiony w menu głównym. Istnieje jednak możliwość wyłączenia domyślnych opcji a co za tym idzie wyłączenia kontroli głównego zarządzania wejściami przez SCLL1. Wyłączenie takie może obywać się z poziomu programu napisanego przez użytkownika w dowolnym momencie. W dowolnym momencie można też któreś wejścia włączyć. W programie można także zmienić przypisany program do danego wejścia, co pozwala na stworzenie bardziej ciekawych interaktywnych programów i poszerza zakres zastosowań układu do bardzo wielu funkcji!

Przeglądanie programu

Sterownik SCLL1 pozwala na przeglądanie istniejących programów. Jeśli przywrócimy ustawienia do domyślnych z MENU głównego, to ostatni 80-ty program jest programem przykładowym, który możemy zobaczyć. Aby to zrobić włączamy MENU i przechodzimy do opcji **Programowanie**. Teraz wybieramy przyciskami GÓRA/DÓŁ program 80 i przechodzimy do niego

klawiszem PRAWO. Mamy kod programu na ekranie wyświetlacza. Kod składa się z komend. Na początku każdej linii wyświetla się jej trzycyfrowy numer od 000 do 999. Pierwsza linia wyświetlacza, to zawsze linia komenda na której „stoimy". Aby przejść dalej wystarczy posługiwać się przyciskami GÓRA/DÓŁ. Aby zobaczyć wszystkie parametry komend, jeśli nie mieszczą się w linii wyświetlacza, wystarczy posłużyć się przyciskami PRAWO/LEWO. Jeśli chcemy wyjść z przeglądania, wciskamy przycisk LEWO na około 0,5sek.

Opcje menu programu

W menu programowanie możemy wybrać program, który nas interesuje i przyciskiem PRAWO przejść do niego. Ale zanim przejdziemy, zobaczmy jakie jeszcze możliwości oferuje ten poziom MENU. Aby otworzyć listę opcji dla danego programu, na którym stoimy wciśnijmy przycisk środkowy. Otworzy się nam lista opcji:

- Uruchom program – uruchamia program, na którym stoimy. Uruchomienie odbywa się bez wyświetlania komend na wyświetlaczu. W czasie przetwarzania programu klawisze PRAWO/LEWO/GÓRA/DÓŁ działają zgodnie z wejściami. Aby przerwać przetwarzanie programu wciskamy przycisk środkowy lub czekamy na koniec programu
- Zaznacz/odznacz – zaznacza program lub odznacza jeśli był zaznaczony (opcja służy np. do zaznaczenia w celu skopiowania)
- Wstaw – zastępuje istniejący program tym, który jest zaznaczony. Nie ma możliwości cofnięcia tej opcji, ale wymaga ona dodatkowego potwierdzenia, że chcemy to zrobić
- Dodaj na koniec – dodaje zaznaczony program lub wycinek programu na koniec programu tego, na którym stoimy.
- Dodaj na począt. - dodaje zaznaczony program lub wycinek programu na początek programu tego, na którym stoimy. Nie kasuje istniejących instrukcji w programie.

Wejdźmy do programu testowego 80. Teraz możemy przyciskiem środkowym otworzyć dodatkowe opcje:

- Komenda – pozwala zmodyfikować komendę, na której stoimy
- Dodaj przed – dodaje komendy **Nic nie rób** przed instrukcją, na której stoimy
- Dodaj za - dodaje komendy **Nic nie rób** za instrukcją, na której stoimy
- Zaznacz start – układ pozwala na zaznaczenie kawałka kodu i np. skopiowanie go lub usunięcie. Ta opcja pozwala określić początek zaznaczenia (które potem będzie widoczne na wyświetlaczu)
- Zaznacz stop – analogicznie jak do opcji Zaznacz start, ta opcja pozwala na określenie końca zaznaczonego obszaru kodu
- Zaznacz ten – pozwala zaznaczyć tylko jedną instrukcję, na której stoimy
- Kopiuj przed – kopiuje zaznaczony obszar przed instrukcją, na której stoimy
- Kopiuj za – kopiuje zaznaczony obszar za instrukcją, na której stoimy
- Czyść zaznaczone – zamienia zaznaczone komendy na komendę **Nic nie rób**
- Usuń zaznaczone – usuwa zaznaczone komendy
- Usuń zaznaczenie – usuwa zaznaczenie, obszar zaznaczony przestaje być zaznaczony
- Wyczyść program – usuwa wszystkie instrukcje programu i pozostawia jedną standardową **Nic nie rób**
- Uruchom LCD – uruchamia program z możliwością śledzenia go na wyświetlaczu LCD
- Uruchom LCD 0,5s – uruchamia program z możliwością wolnego śledzenia go na wyświetlaczu LCD
- Uruchom program – uruchamia program z pominięciem wyświetlacza LCD

Z powyższych opcji należy szczególną uwagę zwrócić na opcje uruchamiania programu. Czas przetwarzania każdej instrukcji programu zależy od tego w jakiej opcji go uruchomiliśmy. Standardowo jeśli wyświetlacz jest uruchomiony to przetworzenie jednej komendy zajmuje około 5ms (oczywiście komendy takie jak czekanie określonego czasu przetwarzają się dłużej). Jeśli skorzystamy z opcji **Uruchom LCD** to wtedy czas przetwarzania każdej komendy wzrośnie do 18ms, ponieważ konieczne jest jeszcze ich wypisywanie na ekranie. Natomiast przy włączeniu opcji **Uruchom LCD 0,5s** komendy będą przetwarzane przez czas około 0,5sek i wyświetlane na wyświetlaczu. Wszystko to ma znaczenie, bo program testowany z LCD będzie przetwarzał się wolniej niż kiedy układ będzie pracował w normalnym trybie przy wyłączonym LCD. Pamiętajmy o tym podczas testowania programów. Zawsze możemy przetestować poprzez **Uruchom program**, ponieważ wtedy nie ma wyświetlania komend na wyświetlaczu.

Tworzenie programów

Komendy jakie obsługuje układ można podzielić na komendy:

1) Grupa I – 4 komendy sterujące automatem wyjść PWM opisanym wcześniej w instrukcji. Do tych komend zaliczamy:
    1. START **[START]**
    2. STOP **[STOP]**
    3. CZEKAJ **[CZEKAJ]**
    4. USTAW **[USTAW]**
2) Grupa II – 3 komendy sterujące parametrem **Zegar** automatu
    1. ZEGAR **[ZEGAR]**
    2. ZEGAR- **[ZEGAR-]**
    3. ZEGAR+ **[ZEGAR+]**

3) Grupa III – 2 komendy oczekiwania
    1. PAUZA **[PAUZA]**
    2. SYNCHRONIZUJ **[SYNCHR]**
4) Grupa IV – 6 komend sterujących programem
    1. POWTÓRZ **[POWT]**
    2. KONIEC POWTÓRZ **[KONIEC POWT.]**
    3. RESET **[RESET]**
    4. URUCHOM **[URUCHOM]**
    5. KONIEC PROGRAMU **[KONIEC PROGRAMU]**
    6. NIC NIE RÓB **[NIC NIE RÓB]**
5) Grupa V – 14 komend obsługujących wejścia/wyjścia (niektóre także sterują programem)
    1. DEZAKTYWACJA WEJŚĆ **[DEZAKT]**
    2. TEST **[TEST]**
    3. ZERUJ **[ZERUJ]**
    4. TEST STAN **[TEST STAN]**
    5. TEST STAN ODWROTNY **[!TEST STAN]**
    6. CZAS **[CZAS]**
    7. MAT **[MAT]**
    8. STOP CZAS **[STOP_CZAS]**
    9. START CZAS **[START_CZAS]**
    10. RESET CZAS **[RESET_CZAS]**
    11. JEŻELI **[JEZELI]**
    12. PROGRAM WEJŚCIA **[WE_PROG]**
    13. DOMYŚLNE USTAWIENIA WEJŚĆ **[WE_DOMYSLNE]**
    14. DOMYŚLNE USTAWIENIA KANAŁÓW PWM **[PWM_DOMYSLNE]**

W dalszej części poznamy kolejne grupy wraz z opisem każdej instrukcji. Po zapoznaniu się z komendami Grupy I, II, i III przejdziemy to praktyczniejszych prób napisania programu.

Grupa I komend – sterowanie automatami PWM

Komendy z tej grupy wyzwalają automat do wykonywania określonej roli. Podczas przetwarzania programu, kiedy układ natrafia na taką komendę, powoduje wyzwolenie automatu jeśli automat w tym czasie nic nie robił lub przerywa działania aktualnie wykonywanej operacji i uruchamia nową. Posługiwanie się automatami jest bardzo korzystne. Dzięki temu program po wyzwoleniu automatu idzie dalej. Dzięki temu także możemy zakończyć przed czasem wykonywaną funkcję przez automat i zmienić ją na inną.

START

Komenda wyzwala automat do płynnego rozjaśniania (płynnego startu) wyjścia. Komenda ta składa się z następujących parametrów, które programujemy

```
START        --------    OD    DO    TEMPO
```

Widzimy, że na początku jest podana komenda i jej nazwa. Następnie znajduje się 8 znaków – . Znaki te określają kolejne wyjścia PWM, których dotyczy ta komenda. Jeśli jest znak – to dane wyjście PWM nie otrzyma tego wyzwalacza automatu. Jeśli natomiast jest znak X to dane wyjście otrzyma ten wyzwalacz i zacznie realizować określoną funkcję. Wyjścia licząc od lewej są zgodne z wyjściami na złączach, a więc pierwszy z lewej to pierwsze wyjście PWM z lewej strony na złączach czyli Wyjście 1.

Po określeniu, których wyjść dotyczy ta komenda, określamy od jakiego poziomu ma rozpocząć się rozjaśnianie (parametr **OD**). Następnie określamy do jakiego poziomu ma odbywać się rozjaśnianie (parametr **DO**). Na koniec podajemy parametr **TEMPO**, który ustawi znany nam wcześniej współczynnik prędkości płynnego rozjaśniania/wygaszania.

Zakresy parametrów:

- **OD : 0-180**
- **DO : 0-180**
- **TEMPO : 1-65535**

Widzimy, że maksymalny parametr **TEMPO** jest bardzo wysoki. W praktyce rzadko kiedy będzie potrzebna aż taka wartość, ale jest ona możliwa do ustawienia. Przy tworzeniu dynamicznych sekwencji raczej nie wyjdziemy poza zakres od 0 do 20

STOP

Komenda wyzwala automat do płynnego wygaszania (płynnego stopu) wyjścia. Komenda ta składa się z następujących parametrów, które programujemy

```
STOP         --------    OD    DO    TEMPO
```

Widzimy, że na początku jest podana komenda i jej nazwa. Następnie znajduje się 8 znaków – . Znaki te określają kolejne wyjścia PWM, których dotyczy ta komenda. Jeśli jest znak – to dane wyjście PWM nie otrzyma tego wyzwalacza automatu. Jeśli natomiast jest znak X to dane wyjście otrzyma ten wyzwalacz i zacznie realizować określoną funkcję. Wyjścia licząc od lewej są zgodne z wyjściami na złączach, a więc pierwszy z lewej to pierwsze wyjście PWM z lewej strony na złączach czyli Wyjście 1.

Po określeniu, których wyjść dotyczy ta komenda, określamy od jakiego poziomu ma rozpocząć się wygaszanie (parametr **OD**). Następnie określamy do jakiego poziomu ma odbywać się wygaszanie (parametr **DO**). Na koniec podajemy parametr **TEMPO**, który ustawi znany nam wcześniej współczynnik prędkości płynnego rozjaśniania/wygaszania.

Zakresy parametrów:

- **OD : 0-180**
- **DO : 0-180**
- **TEMPO : 1-65535**

Widzimy, że maksymalny parametr **TEMPO** jest bardzo wysoki. W praktyce rzadko kiedy będzie potrzebna aż taka wartość, ale jest ona możliwa do ustawienia. Przy tworzeniu dynamicznych sekwencji raczej nie wyjdziemy poza zakres od 0 do 20

CZEKAJ

Komenda wyzwala automat dla określonych wyjść, który nie zmienia ich stanu, ale powoduje, że można zaprogramować na danym wyjściu czas oczekiwania. Może to być przydatne w pewnych sytuacjach. Komenda ta składa się z następujących parametrów, które programujemy:

```
CZEKAJ       --------     ILOSC_CYKLI
```

Tak jak poprzednio, na początku znajduje się komenda oraz 8 znaków określających, który automat ma zostać wyzwolony. Nie będziemy kolejny raz tego analizować, bo to już jest jasne.

Ostatnim parametrem jest **ILOSC_CYKLI**. Ten parametr spowoduje, że po wyzwoleniu danego automatu zostanie odliczony czas będący iloczynem parametru **ILOSC_CYKLI** oraz parametru **Zegar** danego wyjścia. Spójrzmy na przykład:

- wyjście 1 ma parametr **Zegar=2**
- wyjście 2 ma parametr **Zegar=5**
- wyjście 4 ma parametr **Zegar=7**
- wyjście 5 ma parametr **Zegar=1**

Jeśli użyjemy teraz komendy:

```
CZEKAJ       XX--X---     140
```

to dla wyjść 1, 2 i 5 zostanie uruchomiony automat, który zacznie odliczać czas. Dla wyjścia 1 odliczanie będzie trwało 2*140*1ms=280ms, dla wyjścia 2 potrwa 5*140*1ms=700ms, dla wyjścia 5 potrwa 1*140*1ms=140ms. Wyjście 4 nie zostało uwzględnione, więc nie zostanie wyzwolony dla niego automat.

Pamiętajmy, że jak każdy wyzwalacz, tak samo i ten przerywa aktualnie wykonywaną operację przez automat i rozpoczyna nową (w tym przypadku po prostu oczekiwanie, ale możemy dzięki temu zrobić też zatrzymanie automatu).

Zakresy parametrów:

- **ILOSC_CYKLI : 1-65535**

USTAW

Komenda wyzwala automat do płynnego wygaszania/rozjaśniania wyjścia. Składa się z następujących parametrów, które programujemy:

```
USTAW        --------     STAN  TEMPO
```

Na początku jest podana komenda i jej nazwa oraz znany z wcześniejszych opisów zestaw określający, którego wyjścia/wyjść dotyczy ta komenda.

Następny parametr to wartość współczynnika wypełnienia na wyjściu PWM. Ostatnim parametrem jest znane z wcześniejszych instrukcji **TEMPO**.

Zakresy parametrów:

- **STAN: 0-180**
- **TEMPO : 1-65535** lub **- - - - -**

Tym razem parametr **TEMPO** może przybierać nie tylko wartości od 1-65535, ale także można ustawić `-----`. Po ustawieniu takiej wartości, komenda **USTAW** powoduje, że na wyjściu PWM pojawi się natychmiast ten **STAN**, który określono. Nie ma przy tym płynnego rozjaśniania ani wygaszania danego wyjścia.

Jeśli natomiast parametr **TEMPO** jest w zakresie od 1 do 65535 to funkcja powoduje, że płynnie zmieni poziom wypełnienia z tego, który jest aktualnie na jej wyjściu na to, które jest podane w parametrze **STAN**. W dalszej części instrukcji poznamy przykład działania tej komendy.

Grupa II komend – zmiana parametru Zegar automatów

Komendy z tej grupy nie są wyzwalaczami automatu. Ich zadaniem jest modyfikacja stanu parametru **Zegar** określonych wyjść. Domyślnie wszystkie wyjścia mają ustawione **Zegar=**1. Przy pomocy tej instrukcji możemy wpływać na szybkość przetwarzania się automatów. Przykłady poznamy w dalszej części opisu komend.

ZEGAR

Komenda ustawia parametr **Zegar** automatów tych wyjść, które określimy. Składnia tej komendy wygląda następująco:

```
ZEGAR        --------     WARTOŚĆ
```

Podobnie jak poprzednio rozpoznajemy tutaj standardowe 8 znaków -, które już omówiliśmy.

Ostatnim parametrem jest **WARTOŚĆ**, czyli nowa wartość parametru **Zegar** automatu dla tych wyjść, których dotyczy ta komenda.

Zakresy parametrów:

- **WARTOŚĆ: 1-65535**

Widzimy jak przy poprzednich funkcjach, że parametr **Zegar** może być aż 65535. W praktyce rzadko kiedy będziemy korzystali z tak wysokich wartości. Najczęściej będzie to zakres od 0 do 20. Dlaczego? To proste. Załóżmy, że parametr **Zegar** dla wyjścia 1 wynosi 20. Teraz wykonamy instrukcję:

```
START       X-------    0     180   20
```

Ile czasu potrwa rozjaśnianie? Widzimy, że **TEMPO=20**, wiemy, że **Zegar=20** dla wyjścia 1. Obliczamy (180-0)*20*20*1ms=72000ms=72sek. Nie trudno tutaj zauważyć, że wstawienie wartości rzędu 65535 do tych obliczeń dałoby kolosalne czasy przetwarzania automatu.

ZEGAR-

Parametr **Zegar** nie koniecznie musi mieć ustawioną z góry wartość. Możemy coś odjąć od aktualnej wartości parametru **Zegar**. W jakim celu? Jeśli chcemy wykonać jakąś część programu kilka razy, ale tak, żeby było coraz szybciej, to możemy za każdym razem zmienić tylko **Zegar** bez konieczności zmiany parametrów **TEMPO** w każdej instrukcji wyzwalaczów. Składnia komendy wygląda następująco

```
ZEGAR-      --------    -WARTOŚĆ    min=MINIMUM
```

Jak poprzednio wiadomo do czego służy 8 znaków -. Kolejny parametr to **WARTOŚĆ**. Tutaj możemy określić ile będziemy odejmować od aktualnego stanu zegara. Ostatnim parametrem jest **MINIMUM**. Parametr ten oznacza, że jeśli po odjęciu od **Zegar** liczby **WARTOŚĆ** wynik będzie mniejszy od **MINIMUM** to układ ustawi **Zegar** na wartość **MINIMUM** i nie zejdzie poniżej. Przykład:

- **Zegar** dla wyjścia 1 jest równy 30
- **Zegar** dla wyjścia 2 jest równy 50
- **Zegar** dla wyjścia 5 jest równy 5

Jeśli teraz wykonamy instrukcję:

```
ZEGAR-      XX--X---    -20   min=15
```

to po tej instrukcji wartości parametru zegar zmienią się i wyniosą:

- **Zegar** dla wyjścia 1 = 15, bo 30-20=10, a to mniej niż 15, więc ustawiono 15
- **Zegar** dla wyjścia 2 = 30, bo 50-20=30
- **Zegar** dla wyjścia 5 = 15, bo 5-20=-15, a to mniej niż 15, więc ustawiono 15

Zakresy parametrów:

- **WARTOŚĆ: 1-65535**
- **MINIMUM: 1-65535**

ZEGAR+

Analogicznie jak w przypadku komendy **ZEGAR-** mamy komendę **ZEGAR+**, która powoduje dodanie do aktualnego stanu parametru **Zegar** jakiejś wartości. Składnia komendy wygląda następująco

```
ZEGAR+      --------    +WARTOŚĆ    max=MAKSIMUM
```

Jak poprzednio wiadomo do czego służy 8 znaków -. Kolejny parametr to **WARTOŚĆ**. Tutaj możemy określić ile będziemy dodawać od aktualnego stanu zegara. Ostatnim parametrem jest **MAKSIMUM**. Parametr ten oznacza, że jeśli po dodaniu do **Zegar** liczby **WARTOŚĆ** wynik będzie większy od **MAKSIMUM** to układ ustawi **Zegar** na wartość **MAKSIMUM** i nie pozwoli na więcej. Przykład:

- **Zegar** dla wyjścia 1 jest równy 30
- **Zegar** dla wyjścia 2 jest równy 50
- **Zegar** dla wyjścia 5 jest równy 5

Jeśli teraz wykonamy instrukcję:

```
ZEGAR+      XX--X---    +20   max=45
```

to po tej instrukcji wartości parametru zegar zmienią się i wyniosą:

- **Zegar** dla wyjścia 1 = 45, bo 30+20=50, a to więcej niż 45, więc ustawiono 45
- **Zegar** dla wyjścia 2 = 45, bo 50+20=70, a to więcej niż 45, więc ustawiono 45
- **Zegar** dla wyjścia 5 = 25, bo 5+20=25

Zakresy parametrów:

- **WARTOŚĆ: 1-65535**
- **MINIMUM: 1-65535**

Grupa III komend – komendy oczekiwania

Komendy z tej grupy służą do robienia przerw oraz do zsynchronizowania automatów (czekania aż zakończą one

przetwarzanie komendy).

PAUZA

Komenda pozwala zrealizować przerwę liczoną w milisekundach. Składania komendy jest dość prosta:

```
PAUZA        WARTOSC ms
```

Jako wartość podajemy czas w ms. Układ po napotkaniu na taką komendę zatrzymuje przetwarzanie programu. Pamiętajmy jednak, że przetwarzanie instrukcji trwa 5ms, a więc jeśli przerwa zostanie ustawiona na 1ms, to jej przetworzenie potrwa około 6ms.

Zakresy parametrów:

- **WARTOŚĆ: 1-65535**

SYNCHR

Komenda może zatrzymywać realizację programu lub sprawdzać stan automatu wyjść PWM czy nie zakończyły swoich operacji. Składnia tej komendy jest następująca:

```
SYNCHR       --------     (NUMER_PROGRAMU)
```

Z poprzednich rozdziałów wiemy do czego służy 8 znaków -, więc nie będziemy tego powtarzać. Dodatkowo możemy określić parametr numer programu/podprogramu/opcje, która ma się wykonać jeśli warunek synchronizacji będzie spełniony. Czym jest program i podprogram dowiemy się przy komendzie **URUCHOM**. Dodatkowo można ustawić parametr `NUMER_PROGRAMU` określający opcję:

`***` zadaniem tej opcji jest opuszczenie pętli jeśli warunek synchronizacji będzie spełniony

`---` zadaniem tej opcji jest czekanie dopóki nie nastąpi synchronizacja

Tutaj zaznaczmy sobie najważniejszą rzecz, że synchronizacja czeka jeśli wybrano parametr NUMER_PROGRAMU na `---`. W pozostałych przypadkach komenda ta nie blokuje działania programu i po sprawdzeniu idzie dalej. Jak dokładnie działa ta instrukcja dowiemy się w przykładach w następnych rozdziałach.

Grupa I, II, III - przykłady

W tym rozdziale zobaczymy kilka prostych przykładów zastosowania komend z grup I, II i III, które pozwolą zrozumieć jak działają te komendy.

Przykład 1. Mamy napisany program:

```
000     START       X-------    0     180   1
001     START       -X------    0     180   1
002     START       --X-----    0     180   1
003     START       ---X----    0     180   1
004     START       ----X---    0     180   1
005     START       -----X--    0     180   1
006     START       ------X-    0     180   1
007     START       -------X    0     180   1
```

Widzimy jaki był cel piszącego program, ale... on nie zadziała poprawnie. Piszącemu program chodziło zapewne o to, aby wyjścia po kolei załączały się płynnie w taki sposób, że najpierw pierwsze wyjście, potem drugie, potem trzecie itd. Z pozoru wydaje się, że wszystko powinno działać, ale nie działa. Dlaczego? To proste... Czas wykonania pierwszej instrukcji 000 nie wynosi tyle samo co czas rozjaśnienia pierwszego wyjścia. Pamiętajmy, że ta instrukcja to jedynie wyzwalacz automatu, więc zaraz po wykonaniu instrukcji 000 zaczyna się wykonywać instrukcja 001, potem 002... w praktyce przy wyłączonym wyświetlaczu wszystkie te instrukcje wykonają się w czasie około 40ms, podczas gdy rozjaśnienie potrwa kilka razy tyle... aby osiągnąć oczekiwany cel musimy posłużyć się komendą synchronizacji, która spowoduje, że układ będzie czekał, aż określone automaty przetworzą zadane im zadanie. Poprawiamy program do postaci:

```
000     START       X-------    0     180   1
001     SYNCHR      X-------    ---
002     START       -X------    0     180   1
003     SYNCHR      -X------    ---
004     START       --X-----    0     180   1
005     SYNCHR      --X-----    ---
006     START       ---X----    0     180   1
007     SYNCHR      ---X----    ---
008     START       ----X---    0     180   1
009     SYNCHR      ----X---    ---
010     START       -----X--    0     180   1
011     SYNCHR      -----X--    ---
012     START       ------X-    0     180   1
```

```
013     SYNCHR      ------X-    ---
014     START       -------X    0     180   1
015     SYNCHR      -------X    ---
```

Ten program zadziała tak jak chcieliśmy, dlatego, że instrukcja 000 uruchamia automat na wyjściu 1, który zaczyna rozjaśniać od 0 do 180. Następnie instrukcja 001 **czeka** aż automat wyjścia 1 zakończy rozjaśnianie. Rozumiemy już zasadę działania komend **START** i **SYNCHR**? Warto zwrócić uwagę, że komenda SYNCHR w tym przypadku była z końcową opcją ---, czyli była komendą blokującą/czekającą na zakończenie operacji.

Spójrzmy jeszcze na zmodyfikowaną wersję tego samego kodu do postaci:

```
000     START       X-------    0     180   1
001     SYNCHR      XXXXXXXX    ---
002     START       -X------    0     180   1
003     SYNCHR      XXXXXXXX    ---
004     START       --X-----    0     180   1
005     SYNCHR      XXXXXXXX    ---
006     START       ---X----    0     180   1
007     SYNCHR      XXXXXXXX    ---
008     START       ----X---    0     180   1
009     SYNCHR      XXXXXXXX    ---
010     START       -----X--    0     180   1
011     SYNCHR      XXXXXXXX    ---
012     START       ------X-    0     180   1
013     SYNCHR      XXXXXXXX    ---
014     START       -------X    0     180   1
015     SYNCHR      XXXXXXXX    ---
```

Jak zadziała ten kod? Dokładnie tak samo jak poprzedni. Dlaczego? Dlatego, że synchronizujemy po prostu wszystkie wyjścia. Jeśli wyjścia nic nie robią to znaczy, że automaty są wolne. A więc tylko oczekujemy na te wyjścia, które w danym czasie mają uruchomiony automat.

Pamiętamy, że automat pracuje z uwzględnieniem dwóch parametrów **Zegar** oraz **TEMPO**. W tym przypadku domyślnie **Zegar=1** a komendach ustawiano **TEMPO** również na wartość 1. W efekcie czego płynne rozjaśnienie każdego wyjścia potrwa (180-0)*1*1*1ms=180ms czyli około 1/5 sekundy.

Zmodyfikujmy nasz kod dodając na początku komendy wpływające na **Zegar**.

```
000     ZEGAR       X-X-X-X-    3
001     START       X-------    0     180   1
002     SYNCHR      XXXXXXXX    ---
003     START       -X------    0     180   1
004     SYNCHR      XXXXXXXX    ---
005     START       --X-----    0     180   1
006     SYNCHR      XXXXXXXX    ---
007     START       ---X----    0     180   1
008     SYNCHR      XXXXXXXX    ---
009     ZEGAR-      X-X-X-X-    1     1
010     START       ----X---    0     180   1
011     SYNCHR      XXXXXXXX    ---
012     START       -----X--    0     180   1
013     SYNCHR      XXXXXXXX    ---
014     START       ------X-    0     180   1
015     SYNCHR      XXXXXXXX    ---
016     START       -------X    0     180   1
017     SYNCHR      XXXXXXXX    ---
```

Jaki efekt teraz uzyskamy? Komenda 000 spowoduje, że **Zegar** dla wyjść 1, 3, 5, 7 zostanie ustawiony na 3. Oznacza to, że wyzwolenie instrukcją 001 spowoduje, że automat będzie rozjaśniał wyjście 1 w czasie (180-0)*3*1*1ms=540ms czyli około 0,5sek. Następnie instrukcja 003 spowoduje, że automat będzie rozjaśniał wyjście 2 w czasie (180-0)*1*1*1ms=180ms, czyli trzy razy szybciej niż wyjście 1. Potem komenda z linii 005 spowoduje, że automat wyjście 3 rozjaśni w czasie takim jak wyjście 1, a komenda z linii 007, że automat wyjście 4 rozjaśni w czasie takim jak wyjście 2. Dodając tę jedną instrukcję spowodowaliśmy właśnie, że wybrane wyjścia rozjaśniają się wolniej, chociaż komendy rozjaśniania wcale się nie zmieniły.

Dochodzimy do komend 009. Jaki efekt uzyskamy? Zgadza się... wyjścia 5 i 7 rozjaśnione zostaną w czasie (180-0)*2*1*1ms=360ms, a wyjścia 6, 8 w czasie 180ms. Dlaczego? Bo zegar wejść 1, 3, 5, 7 wynosił 3 przed linią 009, a po linii 009 wynosi już 2 dla tych wejść.

Spróbujmy napisać program, który w połowie rozjaśniania jednego wyjścia zaczyna rozjaśniać wyjście następne.

```
000     START       X-------    0     180   1
001     PAUZA       90ms
002     START       -X------    0     180   1
003     PAUZA       90ms
004     START       --X-----    0     180   1
005     PAUZA       90ms
006     START       ---X----    0     180   1
007     PAUZA       90ms
008     START       ----X---    0     180   1
009     PAUZA       90ms
010     START       -----X--    0     180   1
011     PAUZA       90ms
012     START       ------X-    0     180   1
013     PAUZA       90ms
014     START       -------X    0     180   1
015     PAUZA       90ms
```

Zastosowaliśmy tutaj zwykłą sztuczkę. Skoro rozjaśnianie automatu wyzwolonego w linii 000 potrwa 180ms, to po prostu wykonamy pauzę 90ms (pamiętajmy, że w rzeczywistości będzie to około 95ms) i uruchamiamy drugie wyjście. W efekcie w połowie rozjaśniania wyjścia 1 rozpocznie się rozjaśnianie wyjścia 2.

Wiemy już jak działają komendy **START**, **PAUZA** i **SYNCHR**. W analogiczny sposób do **START** działa komenda **STOP**. Wiemy także już jak działa komenda **ZEGAR** oraz **ZEGAR-**. Analogicznie do **ZEGAR-** działa komenda **ZEGAR+**. Do omówienia została jeszcze komendy **USTAW** i **CZEKAJ**. Spójrzmy na poniższy kod:

```
000     START       X-------    0     180   1
001     PAUZA       90ms
002     CZEKAJ      X-------    90
003     SYNCHR      XXXXXXXX    ---
004     START       -X------    0     180   1
005     SYNCHR      XXXXXXXX    ---
006     START       --X-----    0     180   1
007     SYNCHR      XXXXXXXX    ---
```

Pierwsza komenda jest już nam znana. Uruchomi ona w tym przypadku automat, który rozjaśni od 0 do 180 wyjście 1 w czasie 180ms. Komenda 001 robi przerwę 90ms, a więc komenda 002 zacznie się wykonywać około 95ms od czasu wystartowania automatu wyjścia 1, co oznacza, że po dojściu do 002 wyjście 1 rozjaśni się zaledwie do połowy. Co się wtedy stanie? Rozjaśnianie zostanie zatrzymane, a automat zacznie odliczać czas 90 cykli zegara. Jeśli **Zegar=1**, to 90 cykli daje czas 90*1*1ms=90ms. Komenda z linii 003 będzie czekała aż upłynie te 90ms. Jaki efekt dały komendy z linii 000, 001, 002, 003? Rozjaśniliśmy do połowy w czasie 90ms oraz czekaliśmy 90ms i wyjście nadal pozostało rozjaśnione tylko do połowy. Wnioski:

- Komenda **CZEKAJ** zatrzymuje automat jeśli coś rozjaśniał/wygaszał
- Komenda **CZEKAJ** pozwala odliczyć czas dla danego wyjścia z uwzględnieniem parametru **Zegar**, a więc można tworzyć przerwy zależne od parametru **Zegar** a nie stałe przerwy tak jak w przypadku komendy **PAUZA**
- Komenda **CZEKAJ** działa dla konkretnego wyjścia i też wyzwala automat po prostu do czekania, a więc jeśli chcemy wykorzystać czas czekania, musimy zsynchronizować tak jak pokazano to w linii 003

W dalszej części instrukcji pokażemy jak wykorzystać komendę **CZEKAJ** bardziej praktycznie. Tymczasem spójrzmy jeszcze na przykład:

```
000     START       X-------    0     180   1
001     PAUZA       90ms
002     STOP        X-------    180   0     1
003     SYNCHR      XXXXXXXX    ---
```

W tym programie ideą miało być, żeby diody rozjaśniły się do połowy a potem wygasiły. Widzimy, że tutaj jest źle napisane. Oczywiście moglibyśmy to poprawić zmieniając parametry 180 na 90, ale nie zawsze wiemy jaką wartość ma wyjście PWM w danej

chwili, a więc posługiwanie się stałymi konkretnymi poziomami na starcie nie zawsze jest korzystne. Jeśli chcemy aby z nieznanego poziomu nastąpiło przejście do innego poziomu posłużmy się komendą USTAW:

```
000     START       X-------    0     180   1
001     PAUZA       90ms
002     USTAW       X-------    0     1
003     SYNCHR      XXXXXXXX    ---
```

Teraz efekt jest lepszy do przewidzenia. Komenda 000 spowoduje start automatu. Komenda 001 spowoduje przerwę 90ms. Po dojściu do komendy 002 wyjście rozjaśni się mniej więcej do 90, a po wykonaniu komendy 002 zacznie się automatycznie zmniejszać do 0. Widzimy, że komenda **USTAW** nie wymaga od nas znajomości stanu początkowego wyjść PWM.

Spójrzmy na przykład, gdzie wyjścia 1, 2, 3 najpierw płynnie włączymy, a potem płynnie wyłączymy właśnie przy pomocy komendy **USTAW**

```
000     USTAW       X-------    180   1
001     SYNCHR      XXXXXXXX    ---
002     USTAW       -X------    180   1
003     SYNCHR      XXXXXXXX    ---
004     USTAW       --X-----    180   1
005     SYNCHR      XXXXXXXX    ---
006     USTAW       X-------    0     1
007     SYNCHR      XXXXXXXX    ---
008     USTAW       -X------    0     1
009     SYNCHR      XXXXXXXX    ---
010     USTAW       --X-----    0     1
011     SYNCHR      XXXXXXXX    ---
```

Posługiwanie się komendą **USTAW** jest bardzo wygodne.

I tak oto poznaliśmy zasady pisania kodu z użyciem komend I, II i III grupy.

Grupa IV – komendy sterujące programem

Zadaniem komend z tej grupy jest sterowanie programem takie jak uruchamianie innego programu, powtarzanie, resetowanie itd.

POWT

Komenda **POWT** służy do tworzenia powtórzeń, czyli tzw. pętli. Dzięki pętlom możemy pewne części programu wykonywać wielokrotnie lub nieskończoną ilość razy. Jest to bardzo wygodne, ponieważ nie musimy kopiować części programu tylko po to aby wykonał się dany kawałek 5 razy. Wystarczy dodać komendę **POWT**. Składnia komendy wygląda następująco

```
POWT        ILOŚĆ
```

Jako parametr podajemy ilość powtórzeń wybranej części programu.

Zakresy parametrów:

- **ILOŚĆ: 1-65535 lub ZAWSZE**

Jeśli ustawimy **ILOŚĆ** na opcję **ZAWSZE** to nasza pętla będzie powtarzać się w nieskończoność, chyba że ją w inny sposób przerwiemy (ale o tym dalej).

KONIEC POWT.

Komenda ta służy do zakończenia bloku, który miał się powtarzać przy pomocy komendy **POWT**. Co to oznacza? Oczywiście, żeby coś powtarzać, to trzeba określić co chcemy powtarzać. W przeciwnym razie stosowanie komendy **POWT** było by bez sensu. Powtarzanie przy użyciu komendy **POWT** dotyczy tego bloku programu, który zawiera się pomiędzy komendami **POWT** i **KONIEC POWT.** Co to oznacza w praktyce dowiemy się w kolejnych przykładach.

RESET

Komenda służy do wystartowania aktualnie uruchomionego programu od początku. Start programu odbywa się od początku, ale należy pamiętać, że stan wyjść nie zostaje zmodyfikowany. Podobnie nie zostaje zmieniony stan **Zegar** dla wyjść, ani nie zostają zatrzymane żadne automaty. Tak samo nie zostają przywrócone domyślne ustawienia wejść (co to oznacza, poznamy dalej).

Jeśli natomiast przed komendą **RESET** występowały jakieś pętle, które nie zostały zakończone, to oczywiście one zostaną teraz zakończone automatycznie przed ponownym uruchomieniem programu.

URUCHOM

Komenda służy do uruchomienia innego programu jako nowego programu lub jako podprogramu (jaka jest różnica, dowiemy się za chwilę). Podobnie jak w przypadku komendy **RESET**, tak i w tym przypadku nie zostają dokonane zmiany stanu automatów PWM, parametrów **Zegar**, ani nie przywracane są domyślne ustawienia wejść (co to oznacza, poznamy dalej).

Jeśli program zostaje uruchomiony jako nowy, to wszystkie pętle oraz wcześniejsze uruchomienia podprogramów zostają anulowane. Dokładniej wyjaśnią to przykłady.

Składnia komendy **URUCHOM** wygląda następująco

```
URUCHOM     NR_PROGRAMU
```

lub

```
URUCHOM     (NR_PROGRAMU)
```

Obie składnie wyglądają podobnie. Różnią się jedynie nawiasami, w których podany jest numer programu. Jaka jest różnica pomiędzy komendą z nawiasami, a komendą bez nawiasów? Jeśli mamy komendę z nawiasami, to uruchomiony program jest **podprogramem**, czyli jeśli nowy program dojdzie do końca (zostanie zakończony) to układ powróci do wcześniejszego programu, który przetwarzał i który uruchamiał ten podprogram. Jeśli natomiast nie ma nawiasów, to układ uruchomi program, jako nowy program i zapomni o aktualnym programie, wszystkich pętlach itd. Zobaczymy na przykładach jak to dokładnie działa.

KONIEC PROGRAMU

Mamy tutaj do czynienia z komendą kończącą dany program. W praktyce nie jest konieczne umieszczanie jej na końcu kodu. Jeśli jej nie umieścimy, to program i tak się zakończy. Zachodzą jednak przypadki, gdzie mamy dłuższy kod, ale chcemy go zakończyć szybciej z jakiegoś powodu. Przykłady zastosowania tej komendy poznamy w kolejnych przykładach.

NIC NIE RÓB

Najprostsza komenda w wyjaśnieniu, to komenda **Nic nie rób**. Ta komenda po prostu nie robi nic. Do czego więc służy? W układzie nie ma możliwości dopisywania komend bezpośrednio. Można je jedynie modyfikować. A to oznacza, że jeśli chcemy zbudować program, który składać się będzie np. ze 100 komend, to dodajemy z poziomu MENU 100 komend **Nic nie rób** a potem modyfikujemy je do postaci takiej jaką chcemy. Dodatkowo komenda **Nic nie rób** może być przydatna, jeśli na szybko chcemy jakąś komendę usunąć. Wystarczy wtedy wpisać tę komendę, a układ podczas przetwarzania pominie ją.

Grupa IV – przykłady

Tutaj poznamy proste przykłady sterowania wyjściami z użyciem komend z grupy IV.

Zacznijmy od prostego przypadku. Załóżmy, że chcemy, aby wyjście 1 rozjaśniło się i wygasiło trzy razy, a potem to samo zrobiło wyjście 3. Gdyby nie możliwość powtarzania nasz program musiałby wyglądać tak:

```
000     USTAW       X-------    180   1
001     SYNCHR      XXXXXXXX    ---
002     USTAW       X-------    0     1
003     SYNCHR      XXXXXXXX    ---
004     USTAW       X-------    180   1
005     SYNCHR      XXXXXXXX    ---
006     USTAW       X-------    0     1
007     SYNCHR      XXXXXXXX    ---
008     USTAW       X-------    180   1
009     SYNCHR      XXXXXXXX    ---
010     USTAW       X-------    0     1
011     SYNCHR      XXXXXXXX    ---
012     USTAW       -X------    180   1
013     SYNCHR      XXXXXXXX    ---
014     USTAW       -X------    0     1
015     SYNCHR      XXXXXXXX    ---
016     USTAW       -X------    180   1
017     SYNCHR      XXXXXXXX    ---
018     USTAW       -X------    0     1
019     SYNCHR      XXXXXXXX    ---
020     USTAW       -X------    180   1
021     SYNCHR      XXXXXXXX    ---
022     USTAW       -X------    0     1
023     SYNCHR      XXXXXXXX    ---
```

Widzimy, że aby uzyskać taki efekt, potrzeba aż 24 instrukcji, czyli sporo. Z wykorzystaniem pętli otrzymamy krótszy program realizujący to samo:

```
000     POWT        3
001     USTAW       X-------    180   1
002     SYNCHR      XXXXXXXX    ---
003     USTAW       X-------    0     1
004     SYNCHR      XXXXXXXX    ---
005     KONIEC POWT.
006     POWT        3
007     USTAW       -X------    180   1
008     SYNCHR      XXXXXXXX    ---
009     USTAW       -X-------   0     1
010     SYNCHR      XXXXXXXX    ---
011     KONIEC POWT.
```

W tym przypadku z zastosowaniem pętli program jest krótszy o połowę. W dodatku łatwiej wprowadzić zmiany. W powyższym przykładzie widzimy wyraźnie, że najpierw powtarzają się komendy 001, 002, 003, 004, a potem komendy 007, 008, 009, 010. Na tym przykładzie wyraźnie widać do czego służy komenda **KONIEC POWT.** To właśnie komendy **POWT** i **KONIEC POWT.** definiują część kodu, która ma być powtarzana.

Od razu nasuwa się pytanie, czy można powtórzyć coś co już się powtarza? Czy mogę powtórzyć powyższy cały program np. 5 razy? Odpowiedź brzmi: TAK. Poniżej przedstawiamy taki kod:

```
000     POWT        5
001           POWT        3
002                 USTAW       X-------    180   1
003                 SYNCHR      XXXXXXXX    ---
004                 USTAW       X-------    0     1
005                 SYNCHR      XXXXXXXX    ---
006           KONIEC POWT.
007           POWT        3
008                 USTAW       -X------    180   1
009                 SYNCHR      XXXXXXXX    ---
010                 USTAW       -X-------   0     1
011                 SYNCHR      XXXXXXXX    ---
012           KONIEC POWT.
013     KONIEC POWT.
```

W powyższym kodzie zastosowaliśmy takie formatowanie, aby lepiej było widać, które bloki obejmują które pętle. Dzięki temu dobrze widać co powtórzy się ile razy. Z powyższego kodu wynika, że pętla 000 obejmuje instrukcje od 001 do 012... A więc można budować pętle w pętli i nie ma z tym problemu.

Chcemy zrobić, aby wyjście 1 płynnie się włączało i wyłączało przez 5 razy, ale na początku wolniej, a potem coraz szybciej i szybciej. Aby to zrobić znów możemy napisać funkcję, która będzie wszystko wykonywała po kolei, ale możemy także skorzystać z pętli oraz możliwości modyfikacji parametru **Zegar**. Spójrzmy na poniższy kod programu:

```
000     ZEGAR       X-------    5
001     POWT        5
002     USTAW       X-------    180   1
003     SYNCHR      XXXXXXXX    ---
004     USTAW       X-------    0     1
005     SYNCHR      XXXXXXXX    ---
006     ZEGAR-      X-------    -1    1
007     KONIEC POWT.
```

Jak działa poniższy kod? Na początku ustawiamy parametr **Zegar** dla wyjścia 1 na wartość 5. Następnie powtarzamy 5 razy instrukcje 002, 003, 004, 005, 006. Instrukcja 002 włącza wyjście 1, a 003 czeka aż się rozjaśni do końca. Instrukcja 004 wyłącza wyjście 1, 005 czeka aż wygasi się do końca. Co robi instrukcja 006? Zmniejsza ona o 1 wartość parametru **Zegar**. W praktyce spowoduje, to, że po pierwszym wykonaniu części programu, druga wykona się szybciej. Policzmy szybko jak to zadziała.

1) Za pierwszym razem **Zegar=5**, tempo rozjaśniania **TEMPO=1** a więc czas rozjaśnienia to 180*5*1*1ms=900ms
2) Za drugim razem **Zegar**=4, tempo rozjaśniania **TEMPO**=1 a więc czas rozjaśnienia to 180*4*1*1ms=720ms
3) Za drugim razem **Zegar**=3, tempo rozjaśniania **TEMPO**=1 a więc czas rozjaśnienia to 180*3*1*1ms=540ms
4) Za drugim razem **Zegar**=2, tempo rozjaśniania **TEMPO**=1 a więc czas rozjaśnienia to 180*2*1*1ms=360ms

5) Za drugim razem **Zegar**=1, tempo rozjaśniania **TEMPO**=1 a więc czas rozjaśnienia to 180*1*1*1ms=180ms

W ten sposób, kilka umiejętnie połączonych instrukcji dało bardzo ciekawy efekt. A jaki efekt będzie w przypadku poniższego programu?

```
000     ZEGAR       X-------    5
001     POWT        5
002     USTAW       X-------    180   1
003     SYNCHR      XXXXXXXX    ---
004     PAUZA       1000ms
005     USTAW       X-------    0     1
006     SYNCHR      XXXXXXXX    ---
007     PAUZA       1000ms
008     ZEGAR-      X-------    -1    1
009     KONIEC POWT.
```

Efekt będzie taki, że po pełnym rozjaśnieniu nastąpi przerwa 1 sek, potem pełne wygaszenie i przerwa 1 sek. Pętla też powtórzy się 5 razy, ale za każdym razem czas przerwy będzie taki sam, a czas płynnego rozjaśniania i wygaszania będzie się zmieniał.

A co jeśli chcemy, aby czas przerw też się zmieniał? Z pomocą przychodzi komenda **CZEKAJ** tak jak w poniższym przykładzie:

```
000     ZEGAR       X-------    5
001     POWT        5
002     USTAW       X-------    180   1
003     SYNCHR      XXXXXXXX    ---
004     CZEKAJ      X-------    200
005     SYNCHR      XXXXXXXX    ---
006     USTAW       X-------    0     1
007     SYNCHR      XXXXXXXX    ---
008     CZEKAJ      X-------    200
009     SYNCHR      XXXXXXXX    ---
010     ZEGAR-      X-------    -1    1
011     KONIEC POWT.
```

Co dzieje się w tym kodzie? Tym razem komenda 004 spowoduje, że uruchomi się oczekiwanie na wyjściu 1. Ponieważ oczekiwanie na wyjściu 1 uwzględnia stan parametru **Zegar** to mamy w ten sposób zrobione zmienne oczekiwanie. Za pierwszym razem przerwa wyniesie 200*5*1ms=1000ms=1sek, za drugim wyniesie 200*4*1ms=800ms, za 5 razem wyniesie 200*1*1ms=200ms.

Z powyższych przykładów wiemy już jak działają pętle i że można tworzyć pętle w pętli. Czy jest jakieś ograniczenie? Takie ograniczenie istnieje. Maksymalna ilość pętli w pętli (tzw. zagnieżdżeń) wynosi 100, przy czym ilość tych zagnieżdżeń zmniejsza się jeśli będziemy wykonywać podprogramy. Nie mniej jednak ilość 100 jest to tak dużo, że praktycznie nie ma możliwości aby to przekroczyć w prawidłowo napisanym kodzie.

Pętle nieskończone, czyli **POWT ZAWSZE** działają tak samo jak zwykłe pętle, tylko po prostu nigdy się nie kończą same. Nie będziemy więc tutaj specjalnie opisywać ich na przykładzie.

Teraz sprawdźmy jak działa komenda **RESET**. Jeśli weźmiemy kod:

```
000     USTAW       X-------    180   1
001     SYNCHR      XXXXXXXX    ---
002     USTAW       X-------    0     1
003     SYNCHR      XXXXXXXX    ---
004     ZEGAR+      X-------    +1    10
005     RESET
```

W tym kodzie komendy 000, 001, 002, 003 to standardowe wcześniej opisane komendy płynnie rozjaśniające i wygaszające wyjście 1. Komenda 004 powiększa **Zegar**, ale w tym programie wcześniej zegara nikt nie ustawia, a więc domyślnie za pierwszym razem wynosił on 1. Po wykonaniu komend 004 wyniesie 2 dla wyjścia 1. Kolejna komenda 005 spowoduje wykonywanie się programu od początku, przy czym stan wyjść nie został zmieniony, podobnie jak stan parametru **Zegar**. Oznacza to, że za drugim razem **Zegar** wyniesie 2, za trzecim 3 itd. za 10 razem wyniesie 10 i za każdym kolejnym będzie wynosił już tyle samo, bo ustawiono maksimum na wartość 10.

Można zadać sobie pytanie, co będzie jeśli wykonamy taki program:

```
000     POWT        5
001     USTAW       X-------    180   1
002     SYNCHR      XXXXXXXX    ---
003     USTAW       X-------    0     1
004     SYNCHR      XXXXXXXX    ---
005     ZEGAR+      X-------    +1    10
006     RESET
007     KONIEC POWT.
```

Oczywiście, w powyższym przykładzie kod dojdzie do linii 006, a potem uruchomi się od początku i nie będzie brał pod uwagę żadnych wcześniej rozpoczętych pętli. Można także zadać sobie pytanie, czy taka rozpoczęta pętla będzie wliczana do wcześniej omówionego ograniczenia zagnieżdżeń? Nie, taka pętla nie jest liczona. Uruchomiony na nowo program zapomina o wcześniej rozpoczętych pętlach dla tego programu.

Jak zakończyć program przed jego normalnym końcem? Mamy kod:

```
000     ZEGAR       X-------    5
001     POWT        5
002     USTAW       X-------    180   1
003     SYNCHR      XXXXXXXX    ---
004     CZEKAJ      X-------    200
005     SYNCHR      XXXXXXXX    ---
006     USTAW       X-------    0     1
007     SYNCHR      XXXXXXXX    ---
008     CZEKAJ      X-------    200
009     SYNCHR      XXXXXXXX    ---
010     ZEGAR-      X-------    -1    1
011     KONIEC POWT.
```

Szybko skopiowaliśmy ten kod z innego programu ale chcemy żeby tylko raz wykonało się płynne rozjaśnienie i wygaszenie, potem program się zakończył. Nic innego nas nie interesuje. W praktyce musielibyśmy pousuwać wszystkie instrukcje, które są niepotrzebne i otrzymalibyśmy program:

```
000     ZEGAR       X-------    5
001     USTAW       X-------    180   1
002     SYNCHR      XXXXXXXX    ---
003     CZEKAJ      X-------    200
004     SYNCHR      XXXXXXXX    ---
005     STAW        X-------    0     1
006     SYNCHR      XXXXXXXX    ---
007     CZEKAJ      X-------    200
008     SYNCHR      XXXXXXXX    ---
```

A nie da się szybciej?

Da się... wystarczy, że we wcześniejszym programie (000-011) zamienimy instrukcję 010 na **KONIEC PROGRAMU** i w prosty sposób program w tym miejscu się zakończy. Nie ma praktycznego znaczenia, że pozostały komendy **POWT** i **KONIEC POWT**. bo komenda **KONIEC PROGRAMU** po prostu zakończy program w tym miejscu, gdzie się pojawiła.

A co jeśli mam kod taki jak poniżej, ale nie chcę aby wykonała się tylko instrukcja 000 oraz 010? Skopiowany kod z innego programu i te instrukcje się tam znalazły. Czy trzeba je koniecznie usuwać, tak żeby całkowicie znikły z programu?

```
000     ZEGAR       X-------    5
001     POWT        5
002     USTAW       X-------    180   1
003     SYNCHR      XXXXXXXX    ---
004     CZEKAJ      X-------    200
005     SYNCHR      XXXXXXXX    ---
006     USTAW       X-------    0     1
007     SYNCHR      XXXXXXXX    ---
008     CZEKAJ      X-------    200
009     SYNCHR      XXXXXXXX    ---
010     ZEGAR-      X-------    -1    1
011     KONIEC POWT.
```

Nie, nie jest konieczne. Wystarczy, zastąpić te instrukcje komendą **Nic nie rób** tak jak pokazano to na poniższym kodzie.

```
000     Nic nie rób
001     POWT        5
002     USTAW       X-------    180   1
003     SYNCHR      XXXXXXXX    ---
004     CZEKAJ      X-------    200
005     SYNCHR      XXXXXXXX    ---
006     USTAW       X-------    0     1
007     SYNCHR      XXXXXXXX    ---
008     CZEKAJ      X-------    200
009     SYNCHR      XXXXXXXX    ---
010     Nic nie rób
011     KONIEC POWT.
```

Instrukcje **Nic nie rób** zostaną pominięte podczas przetwarzania programu, ale pozostało nam miejsce na wstawienie innych instrukcji jeśli zmienimy zdanie i zechcemy coś dodać.

Jedną z ciekawszych możliwości programu jest opcja uruchamiania innych programów lub podprogramów. Jakie to ma praktyczne zastosowanie? Załóżmy, że tworzymy 4 programy. Pogrubioną czcionką podano numer programu:

```
PROGRAM 20
000     USTAW       X---X---    180   1
001     USTAW       -XXX-XXX    0     1
002     SYNCHR      XXXXXXXX    ---

PROGRAM 21
000     USTAW       -X---X--    180   1
001     USTAW       X-XXX-XX    0     1
002     SYNCHR      XXXXXXXX    ---

PROGRAM 22
000     USTAW       --X---X-    180   1
001     USTAW       XX-XXX-X    0     1
002     SYNCHR      XXXXXXXX    ---

PROGRAM 23
000     USTAW       ---X---X    180   1
001     USTAW       XXX-XXX-    0     1
002     SYNCHR      XXXXXXXX    ---
```

Co robią te programy? Weźmy przykładowo program 23. Jego zadaniem jest płynnie rozjaśnić wyjścia 4 i 8 a pozostałe wyjścia w tym czasie mają zostać płynnie wygaszone.

Mając te programy, które nie robią zbyt wiele, możemy teraz napisać inny program, który będzie wykorzystywał w ciekawy sposób to co tutaj zrobiliśmy. Piszemy program np. 3:

```
PROGRAM 03
000     POWT  5
001     URUCHOM (20)
002     URUCHOM (21)
003     URUCHOM (22)
004     URUCHOM (23)
005     KONIEC POWT.
006     POWT  5
007     URUCHOM (23)
008     URUCHOM (22)
009     URUCHOM (21)
010     URUCHOM (20)
011     KONIEC POWT.
012     POWT  5
013     URUCHOM (20)
014     URUCHOM (21)
015     URUCHOM (22)
016     URUCHOM (23)
017     URUCHOM (22)
```

```
018     URUCHOM (21)
019     KONIEC POWT.
```

Domyślamy się pewnie co robi ten program. Ten program zaoszczędza nam czas i ułatwia pisanie programu jeśli niektóre jego części powtarzają się w nietypowy sposób. Zobaczmy, że pierwsza pętla powtarzająca instrukcje od 001 do 004 powoduje, że najpierw wystartuje program z linii 001, czyli program 20. Ponieważ numer programu jest w nawiasach, to wykona się ten program jako podprogram 20 a po jego wykonaniu układ wróci do programu 03 i będzie go kontynuował od linii 002. Linia 002 uruchamia podprogram 21 a po jego wykonaniu układ wróci do programu 03 i będzie go kontynuował od linii 003. Jaki efekt w tym przypadku osiągniemy? Najpierw 5 razy wykona się sekwencja:

- płynne włączenie wyjścia 1 i 5 a wyłączenie pozostałych
- płynne włączenie wyjścia 2 i 6 a wyłączenie pozostałych
- płynne włączenie wyjścia 3 i 7 a wyłączenie pozostałych
- płynne włączenie wyjścia 4 i 8 a wyłączenie pozostałych

Potem wykona się 5 razy sekwencja odwrotna czyli

- płynne włączenie wyjścia 4 i 8 a wyłączenie pozostałych
- płynne włączenie wyjścia 3 i 7 a wyłączenie pozostałych
- płynne włączenie wyjścia 2 i 6 a wyłączenie pozostałych
- płynne włączenie wyjścia 1 i 5 a wyłączenie pozostałych

A następnie wykona się 5 razy sekwencja... pomyślcie sami :).

Jaka jest różnica pomiędzy komendami **URUCHOM** i **URUCHOM ( )**? Załóżmy, że mamy nasze programy 03, 20, 21, 22, 23. Teraz napiszemy program 04.

```
PROGRAM 04
000     POWT  5
001     URUCHOM (20)
002     URUCHOM (23)
003     URUCHOM (21)
004     URUCHOM (22)
005     KONIEC POWT.
006     URUCHOM 03
007     Pauza 1000ms
008     URUCHOM (20)
```

Jeśli uruchomimy program 04 to wykona on najpierw 5 razy to co jest w pętli. Jak to działa to już wiemy. Następnie zostanie uruchomiony program 03, ale tym razem jak program 03 się skończy, to nie nastąpi powrót do programu 04 do linii 007. W tej sytuacji jak koniec to koniec i układ po prostu skończy dalsze przetwarzanie. Przy uruchamianiu bez nawiasów zostaje zapomniany stan pętli i wcześniejszych informacji o powrotach. Po prostu start jest jak nowego programu z tą tylko różnicą, że nie są zresetowane parametry wejść i wyjść automatów PWM.

Grupa V – komendy obsługujące wejścia/wyjścia

Do tej pory poznaliśmy już komendy zajmujące się budowaniem prostych programów. Nie mieliśmy natomiast możliwości interakcji z użytkownikiem np. poprzez przyciski, czy też jakieś krańcówki, które sterują układem. Jedyna interakcja jaka była możliwa to domyślne sterowanie programami dostępne do ustawienia z poziomu MENU głównego układu. Jednak nie są to wszystkie możliwości jakie mamy przy budowaniu interaktywnych programów.

DEZAKT

Jak wspomnieliśmy wcześniej układ posiada 4 wejścia. Domyślnie po uruchomieniu sterownika wszystkie one pracują reagując na impulsy. Po pojawieniu się impulsu, układ sprawdza co jest ustawione w MENU głównym i uruchamia program, który jest przypisany pod dany klawisz. Nie oznacza to jednak, że w programie nie możemy sprawdzić, czy nie nastąpiło krótkotrwałe załączenie wejścia (tzw. impuls na wejściu). Jednak żeby mieć dostęp do możliwości sprawdzania czy na danym wejściu nie nastąpił impuls, musimy koniecznie wyłączyć sprawdzanie tego wejścia przez domyślne ustawienia sterownika SCLL1. Do tego służy właśnie komenda **DEZAKT**, której składania przedstawia się następująco:

```
DEZAKT      ----
```

Tym razem komenda ma 4 znaki -. Dlaczego 4? Dlatego, że są 4 wyjścia. Pierwszy z lewej znak – to wejście 4, drugi to 3 itd. Kolejność jest taka sama jak kolejność wejść przedstawiona na Rys. 1. Jeśli danemu wyjściu przyporządkujemy znak – to układ od tego momentu przestaje sprawdzać domyślne działanie wejść i będziemy mogli robić to ręcznie. Jeśli natomiast ustawimy znak X to wyjście pracuje z domyślnym przetwarzaniem.

TEST

Wejścia można sprawdzać na dwa sposoby:

1) jaki jest ich stan w danej chwili, ale także

2) czy nie nastąpiło krótkie wciśnięcie (impuls)

Ta komenda pozwala sprawdzić, czy nie nastąpiło krótkie wciśnięcie. Tak dokładniej, to zostanie rozpoznane zarówno krótkie wciśnięcie jak również jeśli ktoś wciśnie i będzie trzymał klawisz na danym wejściu. Jednak rozpoznawanie impulsowe z jakim tutaj mamy do czynienia ma tę ciekawą zaletę, że jedno wciśnięcie zostanie rozpoznane zawsze tylko jeden raz, a więc jeśli ktoś będzie trzymał ciągle, to po pierwszym teście układ stwierdzi, że został wciśnięty klawisz, ale przy drugim teście już nie rozpozna wciśniętego klawisza dopóki nie puścimy i nie naciśniemy go ponownie. Składnia komendy wygląda następująco:

```
TEST    NR_WEJSCIA (DLA TAK: NUMER_PROGRAMU/OPCJA) (DLA NIE: NUMER_PROGRAMU/OPCJA)
```

Komenda wymaga określenia numeru wejścia do którego się odnosi, czyli od 1 do 4. Następnie określamy numer programu/podprogramu/opcję (czyt. dalej) jeśli na wejściu był impuls. Ostatni parametr to określenie numeru programu/podprogramu/opcję jeśli na wejściu nie było impulsu. Widzimy, że posługiwanie się tą komendą nie jest specjalnie skomplikowane. Należy jednak pamiętać, że aby komenda działała poprawnie koniecznie trzeba zdezaktywować domyślne przetwarzanie wejść komendą **DEZAKT**.

Co możemy wpisać dla TAK i NIE? Zarówno dla TAK jak i NIE możemy ustawić:

- **numer programu**, który ma się zacząć wykonywać jeśli na wejściu był impuls lub jeśli go nie było
- **numer podprogramu**, który ma się zacząć wykonywać jeśli na wejściu był impuls lub jeśli go nie było, pod warunkiem, że numer będzie w nawiasach (czyli tak samo jak w przypadku komendy **URUCHOM** podanie w nawiasach uruchamia podprogram, który jak się zakończy to układ będzie kontynuował przetwarzanie wcześniejszego programu)
- **symbol ---**, który oznacza, że nic się ma nie dziać, dla tej opcji (np. chcemy aby coś się wykonało jeśli został naciśnięty klawisz, a jeśli nie został to nic się ma nie dziać)
- **symbol *****, który oznacza, że należy wyjść z pętli dla tej opcji (np. chcemy aby układ opuścił natychmiast pętle po naciśnięciu klawisza)

Szczegóły tych symboli poznamy w dalszej części instrukcji.

ZERUJ

Zadaniem tej komendy jest wyzerowanie stanu wejść impulsowych rozpoznawanych w trakcie komendy **TEST**. Dokładne zastosowanie tej komendy poznamy w przykładach. Składnia tej komendy wygląda następująco:

```
ZERUJ       ----
```

Tak jak poprzednio wiemy co oznaczają znaki – i nie będziemy tego tutaj ponownie omawiać. Tam gdzie jest znak X to stan wejścia impulsowego zostaje wyzerowany.

TEST STAN

Komenda ta działa podobnie do komendy **TEST** z tą jednak różnicą, że nie sprawdza działania impulsowego wejść. Ona sprawdza dokładnie taki stan jaki jest ustawiony na wejściach w danej chwili. Dodatkowo test odbywa się dla wszystkich 4 wejść jednocześnie. Składnia komendy wygląda następująco:

```
TEST STAN       ----        (DLA TAK: NUMER_PROGRAMU/OPCJA)
```

Widzimy, że komenda wymaga określenia jaki stan mają mieć wejścia. Jeśli ustawiony stan będzie na wejściach ma uruchomić się program/podprogram/opcja. W miejscu znaków – może być symbol:

- `-` oznacza, że na wejście ma być rozwarte z masą (GND)
- `X` oznacza, że wejście ma być zwarte z masą (GND)
- `?` oznacza, że wejście może być zwarte lub rozwarte, czyli jego stan nie ma znaczenia podczas testu

Podobnie jak w przypadku komendy **TEST** podanie numeru programu/podprogramu/opcji odbywa się na takich samych zasadach.

Warto przy tym zwrócić uwagę, że komenda **TEST STAN** oraz !**TEST STAN** nie wymaga dezaktywacji wejść poleceniem **DEZAKT**, ponieważ w tym przypadku nie mamy do czynienia z testowaniem impulsów wejściowych, a z testowaniem bezpośrednio stanu jaki jest na wejściu.

!TEST STAN

Komenda ta działa prawie tak samo jak komenda **TEST STAN**. Jedyna różnica polega na tym, że **TEST STAN** przechodził do programu/podprogramu/opcji kiedy test zakończył się zgodnością stanu wejść z założeniami, a komenda **!TEST STAN** przechodzi do programu/podprogramu/opcji kiedy test nie zakończył się zgodnością stanu wejść z założeniami. Składnia komendy wygląda następująco:

```
!TEST STAN      ----        (DLA NIE: NUMER_PROGRAMU/OPCJA)
```

Ponieważ różnica pomiędzy **TEST STAN** i **!TEST STAN** jest nieznaczna, nie będziemy dodatkowo omawiać znaczenia każdego z symboli, bo to zostało już omówione.

CZAS

Komenda ta pozwala ustawić jeden z liczników czasu (lub stan wyjść!) na określoną wartość. Układ posiada 48 liczników czasu. Liczniki o numerach:

- od 1 do 16 mierzą czas w milisekundach [ms] i liczą do 60000ms=60sek
- od 17 do 32 mierzą czas w sekundach [s] i liczą do 60000s = 16 godzin 40 min

- od 33 do 48 mierzą czas w minutach [min] i liczą do 60000min = 41 dni 16 godzin

Komenda ma składnię:

```
CZAS        NUMER_LICZNIKA/NUMER_WYJSCIA  WARTOSC/NUMER_LICZNIKA/NUMER_WYJSCIA
```

Jako pierwszy parametr podajemy numer wyjścia W1 do W8 lub numer licznika od C1 do C48. Oznacza to, że możemy ustawić zarówno wyjście PWM na określony stan natychmiast bez zatrzymywania aktualnie działającego automatu lub stan licznika czasu.

Jako drugi parametr podajemy konkretną wartość od 0 do 59999 lub również numer wyjścia W1 do W8 albo numer licznika od C1 do C48.

Dzięki tej komendzie można zarówno ustawić wyjście jak i licznik czasu. Można także przenieść wartość z jednego licznika czasu do drugiego licznika.

Uwaga! Liczniki liczą od 0 do 60000, czyli nie osiągają większych wartości od 60000! Na tej wartości się zatrzymują!

Uwaga! Próba ustawienia wyjścia PWM na wartość większą niż 180 spowoduje, że i tak zostanie ustawiona wartość 180 na wyjściu.

Uwaga! Liczniki działają niezależnie od reszty systemu, a więc użytkownik musi wiedzieć kiedy chce je zresetować, wystartować i zatrzymać. W praktyce oznacza to, że jeśli włączamy układ do prądu, to domyślnie wszystkie liczniki są wyzerowane i ustawione na wartość 0. Jeśli uruchamiamy jakiś program, który włączy licznik czasu C3, to nawet po zakończeniu tego programu licznik C3 nadal jest uruchomiony. Nawet jeśli uruchomimy inny program, to licznik C3 i tak nadal idzie, chyba, że w nowym programie sami go wyłączymy.

Uwaga! Liczniki zatrzymane można używać do obliczeń a także do przechowywania informacji z innych np. zakończonych programów.

Uwaga! Odliczanie czasu trwa nawet jak liczniki są zatrzymane. W czasie zatrzymania liczników system nadal prowadzi odliczanie, chociaż nigdzie wyniki tego odliczania nie są przechowywane dla użytkownika. Jest to jednak ważna informacja, ponieważ wyobraźmy sobie co będzie jeśli napiszemy program, w którym wewnętrzny licznik odlicza czas dla licznika C35 (licznik minut). Ten wewnętrzny licznik odliczył np. do 35sekund. Teraz my ustawiamy licznik C35 na 0 a potem go uruchamiamy. Po jakim czasie licznik C35 będzie wynosił 1 minutę? Po około 25sekundach... dopiero kolejna minuta będzie cała. Wszystko przez to, że wewnętrzne odliczanie trwa cały czas. To samo dotyczy liczników sekund. Wyjątek stanowią liczniki milisekund, bo milisekunda jest podstawową jednostką czasu w układzie SCLL1. Za to w przypadku liczników w milisekundach pamiętajmy o czasie wykonywania instrukcji wynoszącym około 5ms.

MAT

Komenda pozwala na dodanie/odjęcie/podzielenie/pomnożenie jakiejś wartości z licznika czasu (lub do stanu wyjść!) z jakąś inną wartością/licznikiem czasu/stanem wyjść. Składnia komendy wygląda następująco:

```
MAT         NUMER_LICZNIKA/NUMER_WYJSCIA  +-*/  NUMER_LICZNIKA/NUMER_WYJSCIA/WARTOSC
```

Jako pierwszy parametr podajemy numer wyjścia W1 do W8 lub numer licznika czasu od C1 do C48. Jako drugi parametr określamy rodzaj działania matematycznego jakie chcemy wykonać. Ostatnim parametrem jest również numer licznika/numer wyjścia lub jakaś wartość.

Przykład: `MAT C4 + 10`

Komenda spowoduje, że do licznika czasu C4 zostanie dodana wartość 10.

Przykład: `MAT C40 + W1`

Komenda spowoduje, że do licznika czasu C40 zostanie dodana wartość aktualnego stanu PWM na wyjściu 1.

Przykład: `MAT W1 + W2`

Komenda spowoduje, że do wyjścia W1 zostanie dodana wartość aktualnego stanu PWM na wyjściu 2.

Należy pamiętać, że liczniki czasu osiągają maksymalnie 60000, a więc tej wartości wyniki nie przekroczą. Najmniejsza wartość to 0, a więc przy próbie wykonania operacji dającej mniejszy wynik, to i tak zakończy się ona wartością 0. To same ograniczenia obejmują także wyjścia tyle, że one przyjmują wartości od 0 do 180.

Na szczególną uwagę zasługuje operacja dzielenia.

Przykład: `MAT C1 / 23`

Zakładamy, że C1=540. Jaki wynik jest dzielenia 540/23? 540/23=23,4782... Jaka będzie nowa wartość licznika C1? Nowa wartość licznika będzie wynikiem **po odcięciu części po przecinku**, a więc po tej operacji licznik C1=23.

STOP_CZAS

Komenda ta zatrzymuje licznik czasu o określonym numerze. Składania komendy wygląda następująco:

```
STOP_CZAS       NUMER_LICZNIKA
```

Jako parametr podajemy numer licznika czasu zgodnie z wcześniejszymi ustaleniami.

START_CZAS

Komenda ta uruchamia licznik czasu o określonym numerze. Składania komendy wygląda następująco:

START_CZAS NUMER_LICZNIKA

Jako parametr podajemy numer licznika czasu zgodnie z wcześniejszymi ustaleniami.

RESET_CZAS

Komenda resetuje wszystkie liczniki czasu. Aby zresetować wybrany licznik możemy skorzystać z komendy **CZAS**. Czasami trzeba jednak wykasować wszystkie liczniki jednocześnie. Do tego właśnie służy ta komenda.

JEZELI

Komenda ta pozwala wykonać określony program/podprogram/opcję jeśli stan wyjścia lub licznik czasu w danym momencie spełnia określony warunek. Składnia ma następującą postać:

```
JEZELI NUMER_WYJSCIA/LICZNIK_CZASU <=> NUMER_WYJSCIA/LICZNIK_CZASU/WARTOSC (DLA TAK: NUMER_PROGRAMU/OPCJA)
```

Pierwszym parametrem numer od W1 do W8 dla wyjść lub numer od C1 do C48 dla liczników czasu.

Kolejnym parametrem jest określony warunek. Dostępne są wszystkie kombinacje znaków <=>. Możemy ustawić dowolny z tych znaków jako włączony lub wyłączony.

Trzecim parametrem jest **NUMER_WYJSCIA/LICZNIK_CZASU/WARTOSC**, czyli poziom, który określamy dla danego porównania. Parametrem tym może być zarówno konkretna wartość od 0 do 59999 jak również numer innego wyjścia (W1-W8) lub numer licznika czasu (C1-C48)

Ostatnim parametrem jest numer programu, podprogramu lub opcja wyjścia z pętli. Tak jak w przypadku komendy **TEST** program podajemy jako liczbę bez nawiasów, podprogram jako liczbę z nawiasami, a podanie *** oznacza wyjście z pętli w przypadku spełnienia tego warunku.

Przykład 1:

```
JEZELI C18  > 100     (20)
```

Tak skonstruowana komenda spowoduje sprawdzenie licznika czasu C18 (licznik o numerze 18 liczy w sekundach o czym wspomniano wcześniej). A więc jeśli naliczył więcej niż 100 sekund to uruchamiamy podprogram 20.

Przykład 2:

```
JEZELI C18  > C19     (20)
```

Tak skonstruowana komenda spowoduje sprawdzenie licznika czasu C18 i jeśli jego wynik jest większy od C19 to uruchamiamy podprogram 20.

WE_PROG

Komenda ta pozwala na zmianę numeru programu uruchamianego przez domyślny system na inny lub na żaden po aktywowaniu wejścia. Składnia komendy wygląda następująco:

```
WE_PROG     NUMER_WEJŚCIA     NUMER_PROGRAMU
```

Numer wejścia określamy jak zwykle numerem od 1 do 4. Następnie podajemy numer programu lub ustawiamy --- jeśli nie chcemy aby domyślny system dla danego klawisza uruchamiał jakiś program.

WE_DOMYSLNE

Komenda pozwala na przywrócenie domyślnych programów przypisanych do wejść, ale należy pamiętać, że wejścia jeśli zostały dezaktywowane komendą **DEZAKT** to ta komenda ich nie przywróci i trzeba zrobić to samemu przy pomocy komendy **DEZAKT**.

PWM_DOMYSLNE

Komenda pozwala na przywrócenie domyślnego stanu automatów PWM. W praktyce oznacza to, że dla wszystkich automatów wyjściowych parametr **Zegar** będzie równy 1, wszystkie wyjścia PWM zostaną ustawione na 0, a także automaty zostaną zatrzymane.

Grupa V – przykłady

W tej części instrukcji znajdują się przykłady pokazujące działanie programów z instrukcjami obsługującymi wejścia i wyjścia. Pokażemy przy okazji ciekawsze zastosowanie komendy **SYNCHR** kiedy nie ma blokowania programu, czyli kiedy ostatni jej parametr jest inny niż ---. Będą też przykłady zastosowania prostych operacji matematycznych, liczników czasu i innych elementów. Ciekawe są też proste triki, które można używać w programach do realizacji różnych efektów.

Na początek coś prostego. Zadaniem układu będzie zareagować na wciśnięcie włącznika wejścia 1 i zapalenie wszystkich wyjść, a po wciśnięciu włącznika wejścia 2 wyłączenie wszystkich wyjść. Wydaje się, że podstawą realizacji takiego programu jest ustawienie w menu głównym programu uruchamianego przez wejście 1 i drugiego programu przez wejście dwa. Następnie piszemy takie programy:

```
PROGRAM 01    {zapala dla wejścia 1}
000     START       XXXXXXXX    0     180   1
001     SYNCHR      XXXXXXXX    ---
002     POWT ZAWSZE
```

```
003     KONIEC POWT.

PROGRAM 02    {gasi dla wejścia 2}
000     STOP        XXXXXXXX    180   0     1
001     SYNCHR      XXXXXXXX    ---
002     POWT ZAWSZE
003     KONIEC POWT.
```

Tak napisany program nie spełni naszych oczekiwań. Powodów jest kilka:

- Jeśli dwukrotnie wciśniemy przycisk wejścia 1 to nastąpi dwukrotnie rozjaśnienie, od 0 do 180, ale po co rozjaśniać dwa razy to samo? To samo dotyczy wygaszania.
- Na dodatek przy próbie wygaszenia będziemy mieli krótkie wyłączenie całkowite i dopiero potem włączenie i wygaszenie

Dlaczego tak jest? Główną przyczyną jest to, że program przy uruchamianiu domyślnym z klawiszy zawsze dostaje taki sam stan zresetowanych wyjść, a więc nie ma możliwości kontynuowania tego co było na wyjściach z poprzedniej sekwencji.

Jak wybrnąć z tego problemu? Wystarczy napisać takie programy:

```
PROGRAM 01:   {program główny, kontroluje stan wejść 1 i 2}
000     DEZAKT      ----
001     POWT  ZAWSZE
002     TEST  1     (2)   ---
003     TEST  2     (3)   ---
004     KONIEC POWT.

PROGRAM 02:   {podprogram zapala}
000     USTAW       XXXXXXXX    180   1

PROGRAM 03:   {podprogram gasi}
000     USTAW       XXXXXXXX    0     1
```

W systemie sterownika ustawiamy aby domyślnie uruchamiał się program 01. Zadaniem tego programu jest w linii 000 dezaktywacja domyślnego odczytywania stanu wejść. Następnie w linii 001 jest pętla nieskończona, która obejmuje linie 002 i 003. Linia 002 sprawdza czy pojawił się impuls na wejściu 1. Jeśli się pojawił, to wykonywany zostanie podprogram 2, którego zadaniem jest uruchomienie automatu i rozjaśnianie wszystkich wyjść. Po skończeniu podprogramu, zacznie wykonywać się dalej program główny od linii 003. Linia sprawdza czy nie pojawił się impuls na wejściu 2. Jeśli się pojawi, to wykona się podprogram 03.

W ten prosty sposób osiągnęliśmy ten efekt, który był zamierzony.

Spróbujmy zrobić program, który będzie powodował, że jeden raz wciskamy klawisz wyjście zostaje włączone, drugi raz wciskamy zostaje wyłączone. W tym celu ustawiamy program domyślny na 01, a klawiszom nie przypisujemy żadnych domyślnych programów. Następnie piszemy kod:

```
PROGRAM 01    {wygaszanie}
000     DEZAKT      ----
001     USTAW       XXXXXXXX    0     1
002     ZERUJ       XXXX
003     POWT        ZAWSZE
004     TEST        1     02    ---
005     KONIEC POWT.

PROGRAM 02    {rozjaśnianie}
000     USTAW       XXXXXXXX    180   1
001     ZERUJ       XXXX
002     POWT        ZAWSZE
003     TEST        1     01    ---
004     KONIEC POWT.
```

Domyślnie uruchomi się kod z programu 01. Widzimy, że w programie 01 najpierw zostają dezaktywowane wszystkie wejścia. Dzięki temu będziemy mogli m. in. sprawdzać ich stan poprzez komendę **TEST**. Potem ustawiamy wszystkie wyjścia, żeby zostały płynnie wygaszone. Nie musimy czekać, aż to się stanie. W linii 002 zerujemy stan wejść impulsowych. Potem w linii 003 uruchamiamy nieskończoną pętle, której zadaniem będzie testowanie czy na wejściu 1 był impuls. Jeśli impuls się nie pojawi, to układ dochodzi do linii 005 i potem znów wraca do linii 004 i ponawia test. Tak się będzie działo, dopóki nie podamy impulsu na wejście 1. Pojawienie się takiego impulsu spowoduje, że komenda TEST przejdzie do programu 02. Uruchomi go od nowa. Nowo uruchomiony program nadal ma ustawione te same parametry wejść, które w programie 01 zostały dezaktywowane. Podobnie nowo uruchomiony program nadal ma taki sam stan wyjść PWM. Program 02 rozjaśnia wyjścia, zeruje wejścia impulsowe i czeka podobnie jak program 01 na pojawienie się impulsu. Kiedy impuls się pojawi, to zostanie uruchomiony na nowo program 01, który wygasi płynnie wyjścia i znów będzie czekał na impuls na wejściu 1... w ten oto sposób przy pomocy dwóch prostych programów

zbudowaliśmy układ bistabilnego włączania i wyłączania.

Zmodyfikujmy nasz poprzedni program w taki sposób, aby czas płynnego rozjaśniania i wygaszania można było kontrolować przy pomocy wejść 2 (zwiększa czas) i 4 (zmniejsza czas). Jak to zrobić? Zobaczmy:

```
PROGRAM 01    {wygaszanie}
000     DEZAKT      ----
001     USTAW       XXXXXXXX    0     1
002     ZERUJ       XXXX
003     POWT        ZAWSZE
004     TEST        1     02    ---
005     TEST        2     (03)  ---
006     TEST        4     (04)  ---
007     KONIEC POWT.

PROGRAM 02    {rozjaśnianie}
000     USTAW       XXXXXXXX    180   1
001     ZERUJ       XXXX
002     POWT        ZAWSZE
003     TEST        1     01    ---
004     TEST        2     (03)  ---
005     TEST        4     (04)  ---
006     KONIEC POWT.

PROGRAM 03    {zwiększanie czasu}
000     ZEGAR+      XXXXXXXX    +1    10

PROGRAM 04    {zmniejszenie czasu}
000     ZEGAR-      XXXXXXXX    -1    1
```

Widzimy, że doszły dwa nowe programy 03 i 04, które w kodzie programów 01 i 02 są uruchamiane z nawiasami przez funkcję **TEST**, czyli są uruchamiane jako podprogramy, a więc po ich wykonaniu układ wróci z powrotem tam skąd nastąpiło przejście do podprogramu. Prawda, że nie jest to skomplikowane?

Zwróćmy uwagę, że zegar zmieniony w podprogramie zostanie potem już dla innych programów, chyba że zostanie uruchomiony program z domyślnego systemu przełączania programów lub zostanie wywołana odpowiednia funkcja.

W naszych programach zagospodarowaliśmy wejścia 1, 2, 4. Zostało nam wejście 3. Zróbmy aby wejście 3 przywracało nam domyślne ustawienia kanałów PWM.

```
PROGRAM 01    {wygaszanie}
000     DEZAKT      ----
001     USTAW       XXXXXXXX    0     1
002     ZERUJ       XXXX
003     POWT        ZAWSZE
004     TEST        1     02    ---
005     TEST        2     (03)  ---
006     TEST        4     (04)  ---
007     TEST        3     05    ---
008     KONIEC POWT.

PROGRAM 02    {rozjaśnianie}
000     USTAW       XXXXXXXX    180   1
001     ZERUJ       XXXX
002     POWT        ZAWSZE
003     TEST        1     01    ---
004     TEST        2     (03)  ---
005     TEST        4     (04)  ---
006     TEST        3     05    ---
007     KONIEC POWT.

PROGRAM 03    {zwiększanie czasu}
000     ZEGAR+      XXXXXXXX    +1    10

PROGRAM 04    {zmniejszenie czasu}
```

```
000     ZEGAR-      XXXXXXXX    -1    1

PROGRAM 05    {przywrócenie domyślnych ustawień PWM}
000     PWM_DOMYSLNE
001     URUCHOM 01
```

Jak działa ten program nie trzeba tłumaczyć. Warto tylko zauważyć, że po przywróceniu domyślnych ustawień każdego kanału PWM wszystkie wyjścia są wyłączone, a więc powinniśmy uruchomić potem program 01, bo jego zadaniem jest dopiero płynne rozjaśnienie.

Dla formalności zobaczmy jeszcze jak zrealizować w jednym programie przełączanie bistabilne:

```
PROGRAM 01    {przełączanie bistabilne wejściem 1}
000     DEZAKT      ----
001     USTAW       XXXXXXXX    0     1
002     ZERUJ       XXXX
003     POWT        ZAWSZE
004     TEST        1     ***   ---
005     KONIEC POWT.
006     USTAW       XXXXXXXX    180   1
007     POWT        ZAWSZE
008     TEST        1     ***   ---
009     KONIEC POWT.
010     RESET
```

Jak to działa? To dość proste. Linie 000, 001, 002 już znamy z poprzednich przykładów. W linii 003 jest nieskończona pętla. Wewnątrz niej jest komenda **TEST**, która sprawdza stan wejścia impulsowego 1. Jeśli nic nie wykryto na wejściu 1 to zostaje przetwarzana komenda 005, a potem znowu 004 i tak w kółko. Jeśli natomiast pojawi się impuls na wejściu to symbol `***` ustawiony oznacza, że układ opuści pętle, a więc przejdzie do linii 006, która płynnie włącza wyjścia. Analogicznie jest w dalszej części programu, którą już rozumiemy.

Poznaliśmy już ciekawe sposoby na zaprogramowanie różnych sekwencji. Co jeśli jednak chcielibyśmy mieć coś bardziej złożonego? Może spróbujmy zrobić tak, że mamy pod dwa wejścia 1 i 3 podpięte klawisze „poprzedni” (wejście 1) i „następny” (wejście 3). Przy pomocy tych klawiszy chcemy mieć możliwość przełączania kolejnych żarówek na wyjściach. Co to znaczy? Załóżmy, że na początku świeci żarówka 1. Jeśli wciśniemy „następny” to zaświeci się żarówka 2, jeśli wciśniemy „następny” to zaświeci się 3, jeśli wciśniemy „poprzedni” to zaświeci się 2... jeśli świeci się żarówka 1 i wciśniemy poprzedni to zaświeci się żarówka 8, a jeśli świeci się żarówka 8 i wciśniemy następny to zaświeci się żarówka 1... mamy więc tutaj do czynienia z sekwencyjnym przełączaniem wyjść przy pomocy dwóch przycisków.

Piszemy więc następujące programy. Ustawiamy domyślnie uruchamiany numer programu na 01.

```
PROGRAM 01    {włączanie wyjścia 1}
000     DEZAKT      ----
001     USTAW       -XXXXXXX    0     1
002     USTAW       X-------    180   1
003     POWT        ZAWSZE
004     TEST        1     8     ---
005     TEST        3     2     ---
006     KONIEC POWT.

PROGRAM 02    {włączanie wyjścia 2}
000     USTAW       X-XXXXXX    0     1
001     USTAW       -X------    180   1
002     POWT        ZAWSZE
003     TEST        1     1     ---
004     TEST        3     3     ---
005     KONIEC POWT.

PROGRAM 03    {włączanie wyjścia 3}
000     USTAW       XX-XXXXX    0     1
001     USTAW       --X-----    180   1
002     POWT        ZAWSZE
003     TEST        1     2     ---
004     TEST        3     4     ---
005     KONIEC POWT.
```

```
PROGRAM 04    {włączanie wyjścia 4}
000     USTAW       XXX-XXXX    0     1
001     USTAW       ---X----    180   1
002     POWT        ZAWSZE
003     TEST        1     3     ---
004     TEST        3     5     ---
005     KONIEC POWT.

PROGRAM 05    {włączanie wyjścia 5}
000     USTAW       XXXX-XXX    0     1
001     USTAW       ----X---    180   1
002     POWT        ZAWSZE
003     TEST        1     4     ---
004     TEST        3     6     ---
005     KONIEC POWT.

PROGRAM 06    {włączanie wyjścia 6}
000     USTAW       XXXXX-XX    0     1
001     USTAW       -----X--    180   1
002     POWT        ZAWSZE
003     TEST        1     5     ---
004     TEST        3     7     ---
005     KONIEC POWT.

PROGRAM 07    {włączanie wyjścia 7}
000     USTAW       XXXXXX-X    0     1
001     USTAW       ------X-    180   1
002     POWT        ZAWSZE
003     TEST        1     6     ---
004     TEST        3     8     ---
005     KONIEC POWT.

PROGRAM 08    {włączanie wyjścia 8}
000     USTAW       XXXXXXX-    0     1
001     USTAW       -------X    180   1
002     POWT        ZAWSZE
003     TEST        1     7     ---
004     TEST        3     1     ---
005     KONIEC POWT.
```

Jak działają te programy? Weźmy program 01. Linia 001 uruchamia automat wygaszania wszystkich wyjść poza pierwszym. Linia 002 włącza wyjście pierwsze. Następnie jest pętla, w której są dwie linie **TEST**, które sprawdzają klawisze „następny” i „poprzedni” i w zależności od tego, który wybierzemy uruchamiają określony program. Widzimy, że nawet taka spora interaktywność nie jest trudna do napisania.

Pokażmy sobie jak działa jeszcze komenda **WE_PROG** przy okazji poprzedniego układu przełączania sekwencyjnego. Załóżmy, że chcemy zrobić aby migała nam lampa 1. Kiedy wciśniemy przycisk następny to miga lampa 2. Czyli zasada działania podobna jak poprzednio, ale lampka miga. Nie ma natomiast płynnego wygaszania poprzedniej lampki, która gaśnie natychmiast. Piszemy w takim razie następujące programy:

```
PROGRAM 01    {włączanie wyjścia 1}
000     DEZAKT      XXXX
001     WE_PROG     1     8
002     WE_PROG     3     2
003     POWT  ZAWSZE
004     USTAW       X-------    180   1
005     SYNCHR      XXXXXXXX    ---
006     USTAW       X-------    0     1
007     SYNCHR      XXXXXXXX    ---
008     KONIEC_POWT.
```

```
PROGRAM 02    {włączanie wyjścia 2}
000     DEZAKT      XXXX
001     WE_PROG     1     1
002     WE_PROG     3     3
003     POWT  ZAWSZE
004     USTAW       -X------    180   1
005     SYNCHR      XXXXXXXX    ---
006     USTAW       -X------    0     1
007     SYNCHR      XXXXXXXX    ---
008     KONIEC_POWT.

PROGRAM 03    {włączanie wyjścia 3}
000     DEZAKT      XXXX
001     WE_PROG     1     2
002     WE_PROG     3     4
003     POWT  ZAWSZE
004     USTAW       --X-----    180   1
005     SYNCHR      XXXXXXXX    ---
006     USTAW       --X-----    0     1
007     SYNCHR      XXXXXXXX    ---
008     KONIEC_POWT.

PROGRAM 04    {włączanie wyjścia 4}
000     DEZAKT      XXXX
001     WE_PROG     1     3
002     WE_PROG     3     5
003     POWT  ZAWSZE
004     USTAW       ---X----    180   1
005     SYNCHR      XXXXXXXX    ---
006     USTAW       ---X----    0     1
007     SYNCHR      XXXXXXXX    ---
008     KONIEC_POWT.

PROGRAM 05    {włączanie wyjścia 5}
000     DEZAKT      XXXX
001     WE_PROG     1     4
002     WE_PROG     3     6
003     POWT  ZAWSZE
004     USTAW       ----X---    180   1
005     SYNCHR      XXXXXXXX    ---
006     USTAW       ----X---    0     1
007     SYNCHR      XXXXXXXX    ---
008     KONIEC_POWT.

PROGRAM 06    {włączanie wyjścia 6}
000     DEZAKT      XXXX
001     WE_PROG     1     5
002     WE_PROG     3     7
003     POWT  ZAWSZE
004     USTAW       -----X--    180   1
005     SYNCHR      XXXXXXXX    ---
006     USTAW       -----X--    0     1
007     SYNCHR      XXXXXXXX    ---
008     KONIEC_POWT.

PROGRAM 07    {włączanie wyjścia 7}
000     DEZAKT      XXXX
001     WE_PROG     1     6
002     WE_PROG     3     8
003     POWT  ZAWSZE
004     USTAW       ------X-    180   1
```

```
005     SYNCHR      XXXXXXXX    ---
006     USTAW       ------X-    0     1
007     SYNCHR      XXXXXXXX    ---
008     KONIEC_POWT.

PROGRAM 08    {włączanie wyjścia 8}
000     DEZAKT      XXXX
001     WE_PROG     1     7
002     WE_PROG     3     1
003     POWT  ZAWSZE
004     USTAW       -------X    180   1
005     SYNCHR      XXXXXXXX    ---
006     USTAW       -------X    0     1
007     SYNCHR      XXXXXXXX    ---
008     KONIEC_POWT.
```

Ustawiamy domyślne uruchamianie programu na 01, a przyciskom nie musimy przypisywać żadnych programów, które się będą uruchamiały, bo zrobiliśmy to w trakcie kodzie. Spójrzmy na kod programu 01. W kodzie tym w linii 000 dostają uaktywnione wszystkie wejścia impulsowe i ich przełączanie domyślne. Nie musimy tego robić, ale dla formalności i czytelności zrobiliśmy. W linii 001 przypisujemy program wejściu 1, który ma się uruchomić po jego włączeniu. W linii 002 przypisujemy program, który ma się uruchomić po włączeniu wejścia 2. Dalsza część nie wymaga komentarza. Teraz jeśli pojawi się impuls na wejściu 1, to aktywny system sprawdzi jaki program jest przypisany do wejścia 1. Jest to program 8, a więc zostanie natychmiast przerwane wykonywanie programu 01, następnie zostaną wyzerowane wszystkie wyjścia PWM i przywrócone domyślne ich parametry. Zostanie wykasowana także lista programów przypisana do wejść. Po tym wszystkim uruchomi się program 08. I tu widzimy, że tym razem inny program został przypisany do wejścia 1... Logika tego kodu dalej wydaje się dość jasna.

Spróbujmy zrobić coś ciekawego. Zróbmy układ astabilny, czyli popularną czasówkę, ale działającą w ten sposób, że jeśli wciśniemy przycisk podłączony do wejścia 1 to zapalamy żarówkę. Żarówka świeci się przez 10 minut, ale jeśli w tym czasie jeszcze raz wciśniemy przyciski to żarówka zgaśnie szybciej.

```
PROGRAM 01
000     DEZAKT      ----
001     USTAW       X-------    0     1
002     POWT        ZAWSZE
003     TEST        1     ***   ---
004     KONIEC POWT.
005     USTAW       X-------    180   1
006     POWT        ZAWSZE
007     SYNCHR      X-------    ***
008     TEST        1     01    ---
009     KONIEC POWT.
010     ZEGAR       X-------    1000
011     CZEKAJ      X-------    600
012     POWT        ZAWSZE
013     SYNCHR      X-------    ***
014     TEST        1     ***   ---
015     KONIEC POWT.
016     ZEGAR       X-------    1
017     RESET
```

Widzimy, że nasz program nie jest zbyt długi, jednak składa się już z bardziej zróżnicowanych instrukcji. Przyjrzyjmy się mu dobrze linijka po linijce. Na początku 000 dezaktywuje wejścia, abyśmy mogli analizować ich stan impulsowy w programie. Potem linia 001 ustawia wyjście 1, żeby było wyłączone. Następnie mamy pętle nieskończoną, podczas której powtarza się jedynie jedna instrukcja

```
003     TEST        1     ***   ---
```

Co robi ta instrukcja? Znamy już ją. Ona czeka aż pojawi się impuls na wejściu 1 i kiedy się pojawi to symbol `***` oznacza, że należy opuścić pętle. A więc jeśli na początku mieliśmy wyjście wyłączone, to teraz po wciśnięciu klawisza układ opuszcza pętla i przejdzie do instrukcji 005. Instrukcja ta włącza automat to płynnego uruchamiania wejścia 1. Wyjście już mamy włączone teraz i kolejne instrukcje to znów pętla nieskończona, która ciągle wykonuje dwie instrukcje:

```
007     SYNCHR      X-------    ***
008     TEST        1     01    ---
```

Tutaj w tych instrukcjach po raz pierwszy zobaczymy komendę **SYNCHR** z innym ostatnim parametrem niż `---`. Tutaj

ostatni parametr to ***, który oznacza, że jeśli w tym przypadku wyjście 1 nie skończyło jeszcze przetwarzać to będzie instrukcja pominięta i przejdziemy do linii 008, ale jeśli wyjście 1 skończyło przetwarzać to nastąpi opuszczenie pętli i przejdziemy do linii 010. Po co w takim razie była ta pętla? Dlatego, że rozjaśnianie trwa ułamek sekundy (lub więcej jeśli zmienimy ustawienia w programie). Jeśli w tym czasie użytkownik ponownie wciśnie przycisk, to komenda **TEST** z linii 008 uruchomi nam od początku nasz program 01, który co zrobi? A no w linii 001 wygasi nam naszą żarówkę...

Jeśli żarówka rozjaśniła się do końca to program doszedł do linii 010. W tej linii zostaje ustawiony parametr **Zegar** wyjścia 1 na wartość 1000. Następnie zostaje uruchomiony automat oczekiwania na wyjściu 1 z parametrem 600. Oznacza to, że na wyjściu 1 oczekiwanie potrwa 1000*600*1ms=600 000ms=600sek=10min. Po uruchomieniu wyzwalacza, układ znów zaczyna przetwarzać pętlę nieskończoną z komendami:

```
013     SYNCHR      X-------    ***
014     TEST        1     ***   ---
```

Co robią te komendy w tej pętli? Komenda 013 sprawdza czy na wyjściu 1 upłynął już czas. Jeśli nie upłynął, to komenda zostaje pominięta i przetwarza się 014. Ta komenda sprawdza czy został wciśnięty klawisz. Widzimy, że pętla zostanie przerwana w dwóch przypadkach. W pierwszym jeśli czas upłynie (linia 013), w drugim jeśli zostanie wciśnięty klawisz podpięty pod wejście 1. Po przerwaniu pętli układ dochodzi do komendy 016, która przywraca standardową wartość parametru **Zegar** dla wyjścia 1, a potem komenda 017 resetuje program i cały cykl zaczyna się od początku.

No tak, ale przecież mamy w układzie komendy umożliwiające odliczanie czasu. Dlaczego z nich nie skorzystać w tym przypadku? Oczywiście jest to możliwe. Ten sam program przy użyciu liczników czasów wyglądałby w następujący sposób:

```
000     DEZAKT      ----
001     USTAW       X-------    0     1
002     POWT        ZAWSZE
003     TEST        1     ***   ---
004     KONIEC POWT.
005     USTAW       X-------    180   1
006     POWT        ZAWSZE
007     SYNCHR      X-------    ***
008     TEST        1     01    ---
009     KONIEC POWT.
010     CZAS        C17 = 0
011     START_CZAS  C17
012     POWT        ZAWSZE
013     JEZELI      C17 > 600   ***
014     TEST        1     ***   ---
015     KONIEC POWT.
016     ZEGAR       X-------    1
017     RESET
```

Ten program jest praktycznie identyczny z poprzednim. Główna zmiana jest w liniach 010, 011 i 013. W linii 010 zresetowany został licznik czasu C17, który zgodnie z wcześniejszymi ustaleniami liczy czas w sekundach. Kolejna linia 011 uruchamia licznik czasu (jeśli nie był uruchomiony). Od tego momentu na pewno trwa odliczanie i licznik C17 co jedną sekundę zwiększa swoją wartość. Teraz w linii 013 sprawdzamy czy licznik C17 przekroczył wartość 600 sekund (czyli 10 minut). Jeśli przekroczył to kończymy pętlę. Jeśli nie przekroczył to idziemy do linii 014 testując wejście 1... itd. To samo moglibyśmy uzyskać na liczniku minut.

W ten oto sposób kilkoma prostymi komendami zbudowaliśmy układ, który robi coś bardziej interaktywnego. Postawmy jednak poprzeczkę jeszcze wyżej. Chcemy aby jeden sterownik SCLL1 działał jako 4 niezależne układy astabilne chodzące wg takiej zasady jak poprzednio. Jak to zrobić? Tutaj jest już trudniej, ale nie jest to niewykonalne. Wręcz przeciwnie. Tworzymy następujące założenia:

- wejście 1 włącza/wyłącza wyjście 1
- wejście 2 włącza/wyłącza wyjście 2
- wejście 3 włącza/wyłącza wyjście 3
- wejście 4 włącza/wyłącza wyjście 4
- licznik czasu C17 odlicza czas dla wyjścia 1
- licznik czasu C18 odlicza czas dla wyjścia 2
- licznik czasu C19 odlicza czas dla wyjścia 3
- licznik czasu C20 odlicza czas dla wyjścia 4

```
PROGRAM 40:
000     DEZAKT      ----
001     USTAW       XXXXXXXX    0     1
002     START_CZAS  C17
003     START_CZAS  C18
```

```
004     START_CZAS  C19
005     START_CZAS  C20
006     RESET_CZAS
007     POWT  ZAWSZE
008     TEST        1     (41)  ---
009     TEST        2     (42)  ---
010     TEST        3     (43)  ---
011     TEST        4     (44)  ---
012     POWT  1
013     JEZELI      C17 < 600   ***
014     USTAW       X-------    0     1
015     KONIEC POWT.
016     POWT  1
017     JEZELI      C18 < 600   ***
018     USTAW       -X------    0     1
019     KONIEC POWT.
020     POWT  1
021     JEZELI      C19 < 600   ***
022     USTAW       --X-----    0     1
023     KONIEC POWT.
024     POWT  1
025     JEZELI      C20 < 600   ***
026     USTAW       ---X----    0     1
027     KONIEC POWT.
028     KONIEC POWT.

PROGRAM 41:
000     CZAS        C17 = 0
001     POWT        1
002     JEZELI      W1 <> 0     ***
003     USTAW       X-------    180   1
004     KONIEC PROGRAMU
005     KONIEC POWT.
006     USTAW       X-------    0     1

PROGRAM 42:
000     CZAS        C18 = 0
001     POWT        1
002     JEZELI      W2 <> 0     ***
003     USTAW       -X------    180   1
004     KONIEC PROGRAMU
005     KONIEC POWT.
006     USTAW       -X------    0     1

PROGRAM 43:
000     CZAS        C19 = 0
001     POWT        1
002     JEZELI      W3 <> 0     ***
003     USTAW       --X-----    180   1
004     KONIEC PROGRAMU
005     KONIEC POWT.
006     USTAW       --X-----    0     1

PROGRAM 44:
000     CZAS        C20 = 0
001     POWT        1
002     JEZELI      W4 <> 0     ***
003     USTAW       ---X----    180   1
004     KONIEC PROGRAMU
005     KONIEC POWT.
006     USTAW       ---X----    0     1
```

Domyślnie ustawiamy, aby program 40 uruchamiał się po włączeniu układu SCLL1 do sieci. Analizując program 40 widzimy, że linie od 000 do 006 po prostu przygotowują liczniki czasu do działania. Następnie linia 007 określa, że ma się powtarzać część kodu od linii 008 do 027 w kółko (komenda **POWT** ma swój koniec w linii 028).

Linie 008 do 011 sprawdzają czy został wciśnięty klawisz na jednym z 4 wejść. Jeśli został, to zostaje uruchomiony jeden z podprogramów, którymi zajmiemy się za chwilę. Następnie dla każdego z wyjść wykonywany jest podobny test:

```
012     POWT  1
013     JEZELI      C17 < 600   ***
014     USTAW       X-------    0     1
015     KONIEC POWT.
```

Idea tego testu to pewna sztuczka. Widzimy, że w linii 012 jest pętla, ale powtarza się ona tylko 1 raz, więc po co taka pętla skoro powtarza się 1 raz? A po to, że z takiej pętli można wyjść przed jej końcem. I tak oto robimy jeśli zostaje spełniony warunek C17 < 600, czyli jeśli licznik czasu C17 jest mniejszy od 10 minut to wychodzimy z pętli, a skoro wychodzimy, to nie wykona się instrukcja z linii 014. Z powyższego wynika, że dopóki licznik C17 jest mniejszy od 600 to linia 014 się nie wykonuje. Jeśli licznik dojdzie do 600, wtedy linia 014 się wykona i zostanie wyłączone wyjście (jeśli było włączone to się wyłączy, a jeśli było wyłączone to nic się nie zmieni). Po wyjściu z pętli przetwarza się kolejna instrukcja.

Powtarzając tę samą sztuczkę dla wszystkich wyjść uzyskamy rezultat, że jeśli czas na którymś wyjściu przekracza 10 minut to wyłączamy dane wyjście. Proste?

Pozostaje jeszcze rozważyć jak działają programy 41, 42, 43, 44. Wystarczy rozważyć jeden z nich aby wiedzieć o co chodzi. Spójrzmy na program 44. Wiemy, że uruchamia się on, jeśli został wciśnięty klawisz podpięty pod wejście 4. Załóżmy, że wyjście 4 jest wyłączone w czasie kiedy wciśnięto ten klawisz. Co się wówczas dzieje? Linia 000 zresetuje licznik czasu C20 (ten dla wyjścia 4). Linia 001 rozpoczyna pętlę, która znów ma za zadanie odegrać tę samą wygodną sztuczkę, którą poznaliśmy wcześniej. W linii 002 odbywa się sprawdzenie czy wyjście 4 jest różne od 0 (w naszym przypadku jest równe 0, bo przecież jest wyłączone, więc nie będzie wyjścia z pętli i przejdziemy do instrukcji 003). Instrukcja 003 włącza wyjście 4 a instrukcja 004 kończy podprogram i wróci do programu wcześniejszego czyli 40 do linii 012. Teraz mamy załączone wyjście 4. Ponownie wciskamy klawisz 4 i znów zostanie uruchomiony podprogram 44. Linia 000 resetuje czas, linia 001 to ta sztuczka, linia 002 sprawdza czy wyjście jest różne od 0. Wiemy, że teraz już jest różne, więc wychodzimy z pętli, czyli do linii 006. Co robi linia 006 nie trzeba już tłumaczyć. Potem podprogram się kończy i wracamy do linii 012 w programie 40.

Nic skomplikowanego, a mamy już program bardziej interaktywny wykorzystujący wszystkie wejścia wraz z licznikami czasu.

No dobrze, ale co z operacjami matematycznymi? Może jakiś prosty przykład z ich wykorzystaniem. Zróbmy aby:

- wejście 3 po każdym wciśnięciu rozjaśniało o 1 wyjście 1
- wejście 1 po każdym wciśnięciu wygaszało o 1 wyjście 1
- wejście 2 po każdym wciśnięciu rozjaśniało o 10 wyjście 1
- wejście 4 po każdym wciśnięciu wygaszało o 10 wyjście 1

Rozwiązaniem tego programu są następujący program:

```
PROGRAM 50:
000     DEZAKT      ----
001     USTAW       X-------    0     -----
002     POWT  ZAWSZE
003     POWT 1
004     TEST 3      ---   ***
005     MAT         W1+1
006     KONIEC POWT.
007     POWT 1
008     TEST 2      ---   ***
009     MAT         W1+10
010     KONIEC POWT.
011     POWT 1
012     TEST 1      ---   ***
013     MAT         W1-1
014     KONIEC POWT.
015     POWT 1
016     TEST 4      ---   ***
017     MAT         W1-10
018     KONIEC POWT.
019     KONIEC POWT.
```

W powyższym przykładzie w linii 000-001 mamy standardowe operacje, która znamy. Następnie w linii 002 jest nieskończona pętla, która powtarza ciągle linie od 003 do 018. Linie te można podzielić na 4 części, obsługujące każda osobne wejście. Nie musimy analizować wszystkiego. Wystarczy, że przeanalizujemy jedną część:

```
011     POWT 1
012     TEST 1      ---   ***
```

```
013     MAT         W1-1
014     KONIEC POWT.
```

Po raz kolejny tutaj zastosowano trik, z pętlą. Widzimy, że w linii 012 jeśli na wejściu 1 nie było impulsu to wychodzimy z pętli i idziemy do linii 015. Jeśli natomiast był impuls, to komenda **TEST** nic nie robi a program idzie do linii 013, a linia ta powoduje pomniejszenie o 1 wyjścia 1. Jeśli wyjście byłoby równe 0 to oczywiście po wykonaniu tej komendy nadal byłoby równe 0. To samo przy dodawaniu. Jeśli dodawalibyśmy 1 a na wyjściu byłoby już 180, to po tej operacji nadal byłoby równe 180.

No dobrze, ale powyższy program ma wadę. Raz wciskam przycisk, wykonuje się tylko jedna operacja wygaszenia/rozjaśnienia. A jak zrobić to samo, żeby działało powtarzanie klawisza, czyli dopóki trzymam to rozjaśnia/wygasza. Spójrzmy na poniższy program:

```
PROGRAM 51:
000     DEZAKT      ----
001     USTAW       X-------    0     -----
002     POWT  ZAWSZE
003     POWT 1
004     !TEST STAN  -X--  ***
005     MAT         W1+1
006     KONIEC POWT.
007     POWT 1
008     !TEST STAN  --X-  ***
009     MAT         W1+10
010     KONIEC POWT.
011     POWT 1
012     !TEST STAN  ---X  ***
013     MAT         W1-1
014     KONIEC POWT.
015     POWT 1
016     !TEST STAN  X---  ***
017     MAT         W1-10
018     KONIEC POWT.
019     KONIEC POWT.
```

W powyższym kodzie zmieniliśmy jedynie komendę **TEST** na **!TEST STAN**. Różnica między tymi komendami jest taka, że **TEST** sprawdza impulsy na wejściu, a **TEST STAN** i **!TEST STAN** sprawdzają co dokładnie jest na wejściu w danej chwili. W tym przypadku zastosowanie komendy **!TEST STAN** powoduje, że jeśli stan na wejściu się nie zgadza z tym ustalonym to mamy wyjście z pętli np. w linii 012 aby układ nie wyszedł z pętli musi być stan na wejściu taki, że wejścia 2,3,4 są wyłączone, wejście 1 włączone.

Ale to nie wszystkie możliwości komendy **TEST STAN**. Spróbujmy zrobić, że wciśnięcie jednocześnie włącznika 1 i 3 spowoduje wyłączenie wyjścia 1. Modyfikujemy nasz program i dopisujemy linie.

```
PROGRAM 52:
000     DEZAKT      ----
001     USTAW       X-------    0     -----
002     POWT  ZAWSZE
003     POWT 1
004     !TEST STAN  -X--  ***
005     MAT         W1+1
006     KONIEC POWT.
007     POWT 1
008     !TEST STAN  --X-  ***
009     MAT         W1+10
010     KONIEC POWT.
011     POWT 1
012     !TEST STAN  ---X  ***
013     MAT         W1-1
014     KONIEC POWT.
015     POWT 1
016     !TEST STAN  X---  ***
017     MAT         W1-10
018     KONIEC POWT.
019     POWT 1
020     !TEST STAN  -X-X  ***
021     CZAS        W1=0
022     KONIEC POWT.
023     KONIEC POWT.
```

Może spróbujmy jeszcze zrobić coś co rozpozna długość wciśnięcia klawisza i w zależności od tego ile czasu był włączony włączy lub wyłączy odpowiednie wyjście. Założenia:

- krótkie wciśnięcie wejścia 1 poniżej 0,5sek powoduje załączenie/wyłączenie wyjścia 1
- załączenie od 1 do 2 sekund powoduje załączenie/wyłączenie wyjścia 2
- załączenie od 2 do 3 sekund powoduje załączenie/wyłączenie wyjścia 3
- załączenie powyżej 3 sekund wyłącza wyjścia 1, 2, 3

Operacja załączenie/wyłączenie działa na tej zasadzie, że za pierwszym razem dane wyjście zostaje załączone, za kolejnym wyłączone i tak na zmianę.

Aby sprawdzać czas wciśnięcia klawisza, zaangażujemy do tego licznik czasu C1 (liczy w ms).

Piszemy programy:

```
PROGRAM 60: {program główny}
000   DEZAKT      ----
001   STOP_CZAS   C1
002   CZAS        C1 = 0
003   POWT ZAWSZE
004   TEST STAN   ???X  (61)  {sprawdzamy czy wciśnięto klawisz}
005   !TEST STAN  ???X  (62)  {sprawdzamy czy nie wciśnięto klawisza}
006   KONIEC POWT.

PROGRAM 61: {dla wciśniętego klawisza}
000   START_CZAS  C1
001   POWT 1
002   JEZELI      C1 < 3000   ***
003   USTAW       XXX-----    0     1
004   KONIEC PROGRAMU
005   KONIEC POWT.

PROGRAM 62: {dla nie wciśniętego klawisza}
000   STOP_CZAS   C1
001   POWT 1
002   JEZELI      C1 <> 0     ***
003   KONIEC PROGRAMU
004   KONIEC POWT
005   POWT 1
006   JEZELI      C1 > 500    ***
007   URUCHOM     (71)
008   KONIEC PROGRAMU
009   KONIEC POWT
010   POWT 1
011   JEZELI      C1 > 1000   ***
012   CZAS        C1=0
013   KONIEC PROGRAMU
014   KONIEC POWT
015   POWT 1
016   JEZELI      C1 > 2000   ***
017   URUCHOM     (72)
018   KONIEC PROGRAMU
019   KONIEC POWT
020   POWT 1
021   JEZELI      C1 > 3000   ***
022   URUCHOM     (73)
023   KONIEC PROGRAMU
024   KONIEC POWT
025   USTAW       XXX-----    0     1
026   CZAS        C1=0

PROGRAM 71: {odwracanie stanu na wyjściu 1}
000   CZAS        C1 = 0
001   POWT  1
002   JEZELI      W1 <> 0     ***
003   USTAW       X-------    180   1
```

```
004    KONIEC PROGRAMU
005    KONIEC POWT.
006    USTAW       X-------    0     1

PROGRAM 72: {odwracanie stanu na wyjściu 2}
000    CZAS        C1 = 0
001    POWT  1
002    JEZELI      W2 <> 0     ***
003    USTAW       -X------    180   1
004    KONIEC PROGRAMU
005    KONIEC POWT.
006    USTAW       -X------    0     1

PROGRAM 73: {odwracanie stanu na wyjściu 3}
000    CZAS        C1 = 0
001    POWT  1
002    JEZELI      W3 <> 0     ***
003    USTAW       --X-----    180   1
004    KONIEC PROGRAMU
005    KONIEC POWT.
006    USTAW       --X-----    0     1
```

Nadszedł czas na analizę kodu. Program główny to 60, który powinniśmy uruchomić. W programie głównym mamy komendy, które już znamy. Zadaniem tego kodu jest sprawdzanie czy jest wciśnięty klawisz, czy nie jest i jeśli jest to uruchamia się podprogram 61, jeśli nie jest to podprogram 62. Co się dzieje, kiedy klawisz wciskamy i trzymamy? Wtedy w kółko będzie uruchamiał się kod podprogram 61. Przeanalizujmy go.

W podprogramie 61 linia 000 startuje zegar C1, którego rolą będzie odmierzanie czasu, kiedy przycisk na wejściu jest wciśnięty. Musimy znać czas wciśnięcia przycisku aby określić, które wyjście mamy włączyć/wyłączyć. Jeśli zegar już wcześniej wystartował, to ta komenda nic nie zmienia i zegar dalej idzie. Kolejna linia to znana nam sztuczka z pętlą. Wewnątrz niej jest sprawdzany warunek **JEZELI C1<3000** to wtedy jest wyjście z pętli. Komenda ta powoduje, że nie wykonuje się linia 003 i 004 jeśli czas wciśnięcia klawisza nie wyniósł jeszcze 3000ms=3sek. Jeśli klawisz będzie przytrzymany dłużej, to wykona się linia 003 i 004. Co robią te linie? Oczywiście 003 wyłącza wyjścia 1, 2, 3, a 004 kończy podprogram i wraca do programu głównego.

Podprogram 62 nie jest uruchamiany dopóki przycisk 1 jest wciśnięty. Dopiero kiedy go puścimy, to zostanie uruchomiony podprogram 62. Wiemy, że podprogram 61 uruchomił licznik czasu. Licznik ten liczył do tej pory cały czas, kiedy przycisk był wciśnięty. Teraz podprogram 62 zatrzymuje C1. Następnie linie:

- 001-004 – sztuczka z pętlą sprawdza czy licznik C1 w ogóle coś naliczył (może być przecież tak, że licznik nic nie liczył i wynosi 0). Jeśli nic nie naliczył, to wykona się komenda 003 i zakończy się podprogram
- 005-009 – sztuczka z pętlą sprawdza czy licznik był C1 większy od 500 (jeśli nie był, to znaczy, że przycisk wciśnięto na mniej niż 0,5sek i uruchomi się podprogram 71, który przełączy stan wyjścia na przeciwny temu co jest obecnie)
- 010-014 – sztuczka z pętlą, która sprawdza czy licznik C1 był większy od 1000 (jeśli nie był, a także program dotarł aż tutaj, więc musiał zawierać się od 501 do 1000, a to oznacza, że w takim przypadku licznik tylko zerujemy i nic nie robimy, bo zgodnie z tym co napisaliśmy wcześniej, dla tego przedziału czasu nic się nie dzieje)
- 015-019 – sztuczka z pętlą, która sprawdza czy licznik C1 był większy od 2000 (jeśli nie był, a także program dotarł aż tutaj, więc musiał zawierać się od 1001 do 2000, a to oznacza, że w takim przypadku zmieniamy stan na wyjściu 2 uruchamiając podprogram 72
- 020-024 – sztuczka z pętlą, która sprawdza czy licznik C1 był większy od 3000 (jeśli nie był, a także program dotarł aż tutaj, więc musiał zawierać się od 2001 do 3000, a to oznacza, że w takim przypadku zmieniamy stan na wyjściu 3 uruchamiając podprogram 73

Jeśli czas był większy od 3000 to program doszedł do linii 025. Linia ta wyłącza wszystkie 3 wyjścia a linia 026 resetuje czas licznika C1.

Działanie podprogramów 71, 72, 73 powinno być nam już znane z wcześniejszych przykładów.

Na koniec jeszcze jakaś prosta sekwencja. Podłączamy lampki pod wyjścia i uruchamiamy naszą sekwencję. Jak ona działa, pozostawiamy państwu do oceny :).

```
PROGRAM 01:
000     ZEGAR       XXXXXXXX    1
001     POWT  4
002     START       ---XX---    0     180   1
003     SYNCHR      XXXXXXXX    ---
004     START       --X--X--    0     180   1
005     SYNCHR      XXXXXXXX    ---
006     START       -X----X-    0     180   1
```

```
007     SYNCHR      XXXXXXXX    ---
008     START       X------X    0     180   1
009     SYNCHR      XXXXXXXX    ---
010     STOP        ---XX---    180   0     1
011     SYNCHR      XXXXXXXX    ---
012     STOP        --X--X--    180   0     1
013     SYNCHR      XXXXXXXX    ---
014     STOP        -X----X-    180   0     1
015     SYNCHR      XXXXXXXX    ---
016     STOP        X------X    180   0     1
017     SYNCHR      XXXXXXXX    ---
018     ZEGAR+      XXXXXXXX    +1    5
019     KONIEC POWT.
020     POWT 10
021     USTAW       X-------    180   -----
022     PAUZA       50ms
023     USTAW       -X------    180   -----
024     PAUZA       50ms
025     USTAW       --X-----    180   -----
026     PAUZA       50ms
027     USTAW       ---X----    180   -----
028     PAUZA       50ms
029     USTAW       ----X---    180   -----
030     PAUZA       50ms
031     USTAW       -----X--    180   -----
032     PAUZA       50ms
033     USTAW       ------X-    180   -----
034     PAUZA       50ms
035     USTAW       -------X    180   -----
036     PAUZA       50ms
037     USTAW       X-------    0     -----
038     PAUZA       50ms
039     USTAW       -X------    0     -----
040     PAUZA       50ms
041     USTAW       --X-----    0     -----
042     PAUZA       50ms
043     USTAW       ---X----    0     -----
044     PAUZA       50ms
045     USTAW       ----X---    0     -----
046     PAUZA       50ms
047     USTAW       -----X--    0     -----
048     PAUZA       50ms
049     USTAW       ------X-    0     -----
050     PAUZA       50ms
051     USTAW       -------X    0     -----
052     PAUZA       50ms
053     KONIEC POWT.
```

Najczęściej popełniane błędy w programie

Niestety nie ma jednej metody na napisanie dobrego programu. Natomiast zawsze można popełnić błąd. Błędów w sposobie myślenia można zrobić wiele. O wielu rzeczach można również zapomnieć.

- **Jeśli coś nam nie działa z wejściami**, to powinniśmy pomyśleć czy przypadkiem nie są one aktywne i nie trzeba ich dezaktywować, a może mamy ustawione jakieś programy domyślne lub przypisane klawiszom, które mają złe numery.
- **Jeśli coś nam nie działa z licznikami czasu,** to warto pomyśleć czy przypadkiem nie brakuje komend, które by je uruchamiały, zatrzymywały, czy nie zapomnieliśmy ich wyzerować, czy pamiętamy, że raz uruchomiony licznik czasu idzie nawet wtedy kiedy już program się zakończył i robimy coś zupełnie innego i że taki licznik jeśli chcemy używać to powinniśmy sami zadbać aby został wyzerowany, uruchomiony, zatrzymany
- **Jeśli program się nagle kończy, a nie powinien**, bo został uruchomiony podprogram, to sprawdź czy nie zapomniałeś nawiasu chcąc uruchomić podprogram
- **Jeśli program nie działa tak jak chcemy,** to może trzeba pomyśleć i dokładnie zastanowić się czy dobrze to napisaliśmy

- W podprogramie korzystasz z uruchamiania programów a nie podprogramów. Nowy program uruchomiony z podprogramu powoduje utratę informacji o wszystkich powrotach z podprogramów, pętlach itd. a więc układ tworzy od początku wszystkie adresy powrotów i zapomina o tym co było wcześniej
  W kodzie można popełnić także inne błędy, które układ zasygnalizuje, bo będzie to dla niego niewykonalne:

```
PROGRAM 01:
000    Nic nie rób
001    Nic nie rób
002    Nic nie rób
003    Nic nie rób
004    USTAW       XXXXXXXX    0     1
005    KONIEC POWT.
006    Nic nie rób
```

Od razu widać niedorzeczność w tym kodzie. W linii 005 jest komenda **KONIEC POWT.** a gdzie w takim razie jest **POWT**? W takim przypadku układ zasygnalizuje nam taki błąd wypisując na wyświetlaczu:

```
KONIEC POWT. wystąpiło w złym miejscu!
```

W drugiej linii pojawi się dodatkowo w jakim programie oraz w której linii taki błąd wystąpił.

Innym rodzajem błędu, którego nie da się wykonać jest przekroczenie ilości zagnieżdżeń pętli lub podprogramów. Spójrzmy na prosty przykład:

```
PROGRAM 01:
000    URUCHOM     (02)

PROGRAM 02:
000    URUCHOM     (03)

PROGRAM 03:
000    URUCHOM     (01)
```

Jeśli uruchomimy program 01 to on uruchomi podprogram 02, ale przy tym musi zapamiętać gdzie wrócić. Po uruchomieniu podprogramu 02, uruchomiony zostaje podprogram 03 a także znów zapamiętany adres powrotu. Podprogram 03 startuje podprogram 01 i już mamy zapamiętany 3 adres powrotu... i cykl znów się powtarza, a ilość zapamiętanych informacji rośnie. Efekt? Pojawia się komunikat w pewnym momencie:

```
Błąd stosu! Przekroczono ilość zagnieżdżeń!
```

Czym jest stos? Stos to takie coś co pamięta gdzie trzeba wrócić. Może się zdarzyć, że pętlami lub podprogramami przekroczymy ilość pamięci i układ nie będzie w stanie zapamiętać wszystkich danych powrotu.

Ale za to kod:

```
PROGRAM 01:
000    URUCHOM     (02)

PROGRAM 02:
000    URUCHOM     (03)

PROGRAM 03:
000    URUCHOM     01
```

wykona się poprawnie. Dlaczego? Bo po dojściu do podprogramu 3, nastąpi przejście do programu 1 i zostaje zapomniany cały poprzedni stos z informacjami o powrocie, pętlach itd.

Na zakończenie

Pamiętaj, że to co zaprogramujesz zależy od Ciebie. Prawie każdy problem można rozwiązać na wiele różnych sposobów. Notuj sobie poprawnie zaprogramowany kod i nie zgub go, aby w przypadku awarii układu móc go potem odtworzyć.

Mamy nadzieje, że przykłady, które zaprezentowaliśmy pozwolą zrozumieć co można osiągnąć przy pomocy **SCLL1**. Początkowo sterownik miał być tylko zwykłym sterownikiem z wyjściami, gdzie można ułożyć tylko sekwencję i nic więcej. Jednak plany się zmieniły i teraz prócz zwykłych sekwencji można budować znacznie bardziej złożone systemy. Sterownik **SCLL1** stał się dzięki temu prostym sterownikiem PLC lub jak kto woli Sterownikiem Kompaktowym.

Nie pozostaje nam nic innego, jak życzyć powodzenia w Państwa konstrukcjach.

Uwagi!

Uwaga! Koniecznie pamiętaj o zachowaniu niezbędnej wentylacji, jeśli układ zostanie zamknięty w jakiejś szafie instalacyjnej!

Uwaga! Zachowaj odległość pomiędzy urządzeniami mogącymi wpływać negatywnie na stabilność pracy układu SCLL1 np. falownikami, stycznikami.

Uwaga! Przewody niskonapięciowe (sterujące) nie powinny iść razem z przewodami wysokonapięciowymi 230V! Zachowaj odległość pomiędzy tymi przewodami minimum 10cm. Nie tylko jest to kwestia bezpieczeństwa, ale także jest to niezbędna zasada przy projektowaniu instalacji, w której przewody można podzielić na zasilające i sterujące. Dodatkowo jeśli przewody sterujące przecinają się z przewodami zasilającymi to staraj się, aby przecięcie następowało pod kątem prostym.

Uwaga! Urządzenie elektroniczne! Nieprawidłowe użytkowanie urządzenia może grozić uszkodzeniem odbiornika lub innymi poważniejszymi konsekwencjami w tym porażeniem prądem! Zachowaj szczególną ostrożność!

Uwaga! Instalacja urządzenia powinna obywać się w taki sposób, aby urządzenia nie można było dotykać w czasie kiedy nie jest to niezbędne do programowania. Prawidłowo zainstalowany układ jest wtedy, kiedy znajduje się w miejscu niedostępnym, dzięki czemu nikomu nie grozi porażenie prądem.

Uwaga! Sterownik przeznaczony jest do załączania urządzeń nie zagrażających bezpośrednio życiu takich jak np. żarówki. W przypadku chęci wykorzystania sterownika do załączania urządzeń takich jak silniki itd. należy zastosować go jako element pomocniczy ułatwiający włączanie i wyłączanie a dodatkowo należy zamontować włącznik bezpieczeństwa, w który powinno być wyposażone każde zagrażające życiu elektryczne narzędzie pracy.

Uwaga! Uruchamianie układu na świeżo położonych tynkach może spowodować, że układ nie będzie działał poprawnie, jednak nie jest to regułą. W takim przypadku należy odczekać aż ściany wyschną i podłączyć układ.

Uwaga! Jako producent nie określamy szczegółowo kolorów kabli i tego jak taka instalacja ma być wykonana. Najważniejsze to aby instalacja była wykonana w sposób bezpieczny, niezagrażający nikomu, solidny i zgodny ze sztuką i przepisami!

Uwaga! Instalacji układu powinna prowadzić osoba posiadająca odpowiednią wiedzę i uprawnienia, gdyż nieprawidłowe posługiwanie się urządzeniem może grozić porażeniem i poważnymi skutkami zdrowotnymi jak w przypadku każdego urządzenia zasilanego z sieci 230V.