# 10. Sterowniki PLC – metoda SFC

Metoda *sekwencyjnego schematu funkcjonalnego SFC* jest oparta na teorii sieci *Petriego* typu P/T (pozycja/tranzycja). Niektórzy autorzy nazywają *SFC* językiem i piszą, że należy on do grupy *języków sekwencyjnych schematów funkcjonalnych* lub inaczej do grupy *języków sterowania sekwencyjnego*. Norma IEC 61131 nie używa pojęcia język *SFC*, pisze o *SFC* jako o metodzie umożliwiającej organizację programu. Formalizm *SFC* został oparty na metodzie *Grafcet* opracowanej w 1977 roku przez firmę Telemecanique. *Grafcet* doczekał się modyfikacji w postaci np.: *GRAPH* 5, *GRAPH* 7, *Grafpol* i *SFC*. Do rozwoju języków tego typu przyczyniły się problemy związane ze stosowaniem klasycznych metod projektowania układów sekwencyjnych.

Sieć języka *SFC* to graf skierowany, który może mieć wierzchołki dwóch różnych typów:

- *kroki,*
- *tranzycje*.

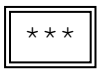
Wierzchołki grafu (kroki i tranzycje) łączone są krawędziami skierowanymi wskazującymi kierunek aktywowania kolejnych kroków. Pomiędzy dwoma kolejnymi krokami w grafie musi znajdować się dokładnie jedna tranzycja, a pomiędzy dwoma kolejnymi tranzycjami musi znajdować się dokładnie jeden *krok*. Domyślnym kierunkiem zmiany aktywności kroków jest kierunek z góry do dołu. W przypadku konieczności powrotu z kroku położonego na grafie niżej do kroku położonego wyżej norma, w celu zwiększenia przejrzystości, dopuszcza umieszczanie na połączeniach grotów wskazujących kierunek połączeń.

Sieć *SFC* definiuje ogólną strukturę programu. Szczegółowe działanie programu jest opisywane za pomocą *akcji* i *warunków przejść*, które mogą być zapisywane w jednym z czterech podstawowych języków normy (akcje mogą być zapisywane również w *SFC*). Z każdym krokiem może być skojarzony zbiór akcji (zbiór może być pusty), z każdą tranzycją związany jest jeden warunek przejścia.

## 10.1. Kroki

*Kroki* reprezentują elementarne etapy sterowanego procesu. Sterownik PLC rozpoczyna wykonywanie programu zapisanego w *SFC* od tzw. *kroku początkowego*. Sieć *SFC* powinna zwierać dokładnie jeden krok tego typu. Graficznie krok prezentowany jest w postaci prostokąta, a krok początkowy w postaci prostokąta z podwójną ramką (tab. 10.1).

*Tab. 10.1. Rodzaje kroków*

| *Symbol* | *Opis* |
|---|---|
|  | *krok początkowy* |
|  | *krok* |

gdzie (***) to nazwa kroku.

Kroki w trakcie działania programu są, zgodnie z regułami języka, *aktywowane* i *dezaktywowane* – krok jest aktywny od chwili rozpoczęcia jego wykonywania do chwili, w której zostanie rozpoczęte wykonywanie kroku następnego. W programie *SFC* z każdym krokiem związane są dwie zmienne:

`***.X` – zmienna typu `BOOL` określającą aktywność kroku, zmienna przyjmuje wartość `1` (`TRUE`) dla kroku aktywnego i `0` (`FALSE`) dla kroku nieaktywnego,

`***.T` – zmienna typu `TIME` określająca czas aktywności kroku,

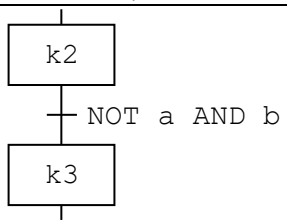
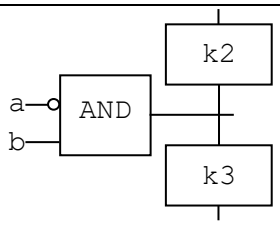
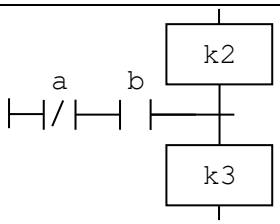
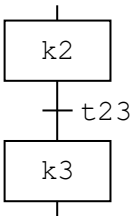
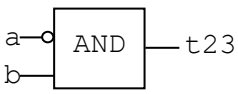
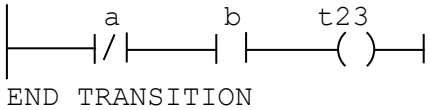
gdzie (`***`) to nazwa kroku.

## 10.2. Tranzycje

*Tranzycja* (*przejście*) opisuje warunki, których spełnienie kończy wykonywanie kroku lub kroków bezpośrednio poprzedzających tranzycję i rozpoczyna wykonywanie kroku lub kroków bezpośrednio następujących po tranzycji. Graficznie tranzycja rysowana jest jako poziomy odcinek przecinający połączenie kroków (połączenia rysowane są liniami pionowymi). Bezpośrednio obok symbolu może być zapisywana jej nazwa lub *warunek przejścia*.

*Warunek przejścia* jest wyrażeniem logicznym. Zgodnie z normą warunki mogą być zapisywane w językach: *IL*, *ST*, *FBD*, *LD* (tab. 10.2), a warunki, które są zawsze spełnione należy zapisywać jako `1` (`TRUE`).

*Tab. 10.2. Wybrane sposoby reprezentacji tranzycji i warunku przejścia*

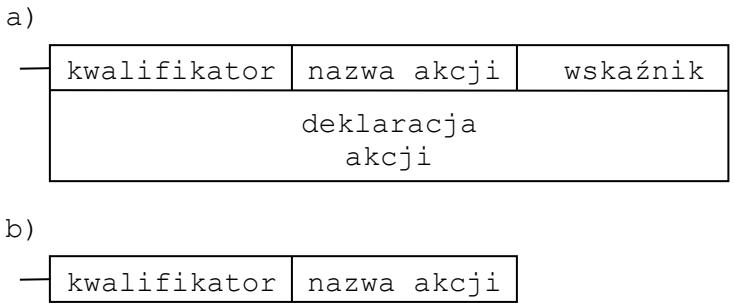
| *Przykład* | *Opis* |
|---|---|
|  | *tranzycja w języku ST* |
|  | *tranzycja w języku FBD* |
|  | *tranzycja w języku LD* |
|  | *tranzycja opisana nazwą (tutaj* `t23`*), zdefiniowana w wybranym języku, np:*<br>*ST*<br>`TRANSITION t23`<br>`:= NOT a AND b;`<br>`END_TRANSITION`<br>*FBD*<br>`TRANSITION t23:`<br>  <br>`END_TRANSITION`<br>*LD*<br>`TRANSITION t23:`<br>  <br>`END_TRANSITION` |

## 10.3. Akcje

*Akcje* to operacje wykonywane w czasie aktywności kroku, operacje te mogą np. przypisywać odpowiednie wartości zmiennym wyjściowym. Z każdym krokiem może być skojarzony zbiór akcji, dopuszczalne jest stosowanie kroków niepołączonych z akcjami – kroki tego typu oczekują na spełnienie warunku przejścia do kroku kolejnego. Zgodnie z normą akcja może być:

- zmienną boolowską,
- ciągiem instrukcji w języku tekstowym (*IL*, *ST*),
- zbiorem obwodów w języku graficznym (*FBD*, *LD*),
- siecią *SFC*.

Zostało przewidzianych kilka metod kojarzenia akcji z krokiem. W postaci graficznej akcja jest kojarzona z krokiem z pomocą *bloku akcji*. Rysunek 10.1 pokazuje blok akcji w postaci pełnej i uproszczonej.




*Rys.10.1. Blok akcji, postać: a) pełna, b) uproszczona*

Obowiązkowymi elementami bloku akcji są:

*kwalifikator* który określa warunki wykonania akcji definiujące kiedy i jak długo akcja będzie wykonywana,
*nazwa akcji* która określa nazwę kojarzonej akcji.

Opcjonalnymi elementami bloku akcji są:

*wskaźnik* to zmienna boolowska zdefiniowana wewnątrz akcji wykorzystywana do informowania o statusie akcji (np. *akcja aktywna*),
*deklaracja akcji* to zestaw instrukcji definiujących sposób działania akcji.

W dalszej części podręcznika bloki akcji będą przedstawiane w postaci uproszczonej. Przed narysowaniem sieci *SFC* dla pierwszego programu konieczne jest jeszcze zestawienie kwalifikatorów przewidzianych w normie do opisu warunków wykonywania akcji.

## 10.4. Kwalifikatory akcji

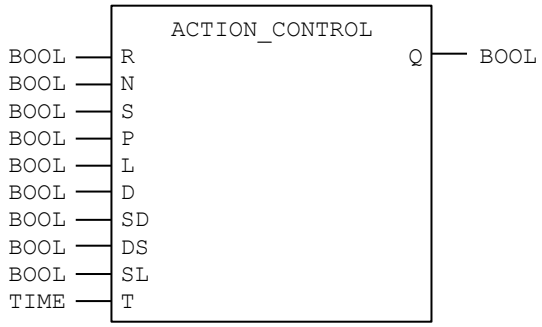
Planowaniem i nadzorowaniem przebiegu wykonania akcji zajmuje się system operacyjny sterownika, który określa warunki uruchomienia i zatrzymania każdej akcji. Warunki te są zależne od *kwalifikatorów* związanych z tą akcją. Lista kwalifikatorów została zestawiona w tab. 10.3.

*Tab. 10.3. Kwalifikatory akcji*

| *Kwalifikator* | *Opis* |
|---|---|
| *brak* lub `N` | *akcja nieprzechowywana* (ang. Non stored)<br>instrukcje akcji wykonywane są przez cały czas aktywności kroku w każdym cyklu programowym sterownika |
| `S` | *akcja zapamiętywana* (ang. Set)<br>akcja jest uruchamiana, gdy krok zyskuje aktywność, utrata aktywności nie przerywa akcji – akcja jest wykonywania do momentu skasowania w innym kroku |
| `R` | *akcja nadrzędnie kasowana* (ang. overriding Reset)<br>kasuje akcję uruchomioną w innym kroku |
| `P` | *akcja impulsowa* (ang. Pulse)<br>instrukcje akcji wykonywane są tylko raz, gdy krok zyskuje aktywność (w niektórych sterownikach *akcje impulsowe* są wykonywane gdy krok zyskuje i traci aktywność) |
| `L` | *akcja ograniczona w czasie* (ang. time Limited)<br>wykonywanie akcji kończy się albo po określonym czasie (który musi być podany bezpośrednio po kwalifikatorze akcji, np. `Lt#1s`) albo po utracie aktywności przez krok z którym akcja jest skojarzona |
| `D` | *akcja opóźniona w czasie* (ang. time Delayed)<br>wykonywanie akcji rozpoczyna się po określonym czasie (specyfikacja czasu jak przy kwalifikatorze `L` czyli np. `Dt#1s`), jeśli krok utraci aktywność przed uruchomieniem akcji to nie zostanie ona uruchomiona |
| `SD` | *akcja zapamiętywana i opóźniona* (ang. Stored and time Delayed) akcja jest uruchamiana po określonym czasie niezależnie od tego czy krok utracił aktywność (specyfikacja czasu jak przy kwalifikatorze `L` czyli np. `SDt#1s`) – akcja jest wykonywania do momentu skasowania w innym kroku |
| `DS` | *akcja opóźniona i zapamiętywana* (ang. Delayed and Stored)<br>akcja jest uruchamiana po określonym czasie pod warunkiem, że krok po upływie tego czasu jest jeszcze aktywny (specyfikacja czasu jak przy kwalifikatorze `L`, czyli np. `DSt#1s`) – akcja jest wykonywania do momentu skasowania w innym kroku |
| `SL` | *akcja zapamiętywana i ograniczona w czasie* (ang. Stored and time Limited)<br>wykonywanie akcji kończy się po określonym czasie (specyfikacja czasu jak przy kwalifikatorze `L`, czyli np. `SLt#1s`) niezależnie od aktywności skojarzonego kroku albo wcześniej jeśli zostanie skasowana |

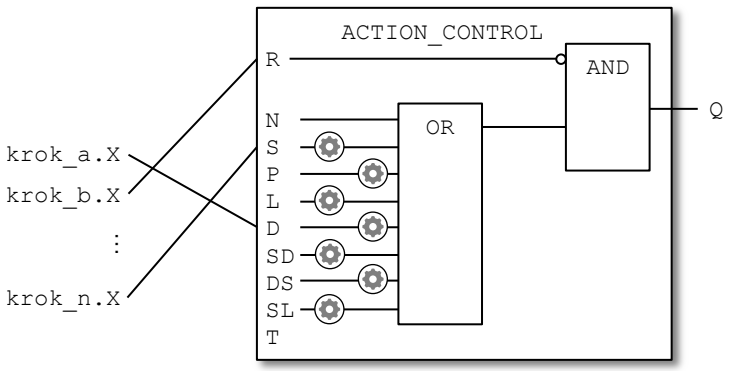
Z opisu w tab. 10.3 wynika, że akcja może występować na grafie *SFC* z różnymi kwalifikatorami (np. akcja zapamiętywana jest kasowana). Za wykonywanie akcji odpowiada, tworzony przez system operacyjny dla każdej akcji, specjalny blok zarządzający `ACTION_CONTROL`, którego interfejs pokazany jest na rys. 10.2. Blok ten ma wejścia boolowskie odpowiadające wszystkim możliwym kwalifikatorom akcji oraz wejście `T` typu `TIME`, na które podawany jest czas związany

z kwalifikatorami L, D, SD, DS lub SL. Wyjście bloku Q jest typu boolowskiego i określa czy dla podanych wartości wejściowych akcja powinna być uruchomiona czy zatrzymana, a w przypadku akcji reprezentowanej przez zmienną boolowską wartość Q jest przypisywana tej zmiennej.




*Rys. 10.2. Blok ACTION_CONTROL*

System operacyjny sterownika ustawia wartości wejść bloku na podstawie aktualnego stanu aktywności kroków w grafie. Jeżeli akcja została zadeklarowana z wybranym kwalifikatorem, to na odpowiadające wejście bloku podawany jest stan aktywności kroku (TRUE, FALSE lub 1, 0) związanego z daną deklaracją akcji. Sygnał ten jest następnie poddawany odpowiedniemu przetwarzaniu wynikającemu z typu kwalifikatora [6]. Wartość wyjściowa bloku ACTION_CONTROL ustalana jest jako koniunkcja zanegowanego wejścia związanego z kwalifikatorem R z wynikiem alternatywy pozostałych wejść bloku, zgodnie ze schematem pokazanym na rys. 10.3 (symbol ⚙ oznacza dodatkowe przetwarzanie sygnału wejściowego).




*Rys. 10.3. Działanie bloku ACTION_CONTROL przykładowej akcji skojarzonej z krokami a, b,..., n z kwalifikatorami D, R,..., S*

Ze sposobu funkcjonowania bloku zarządzającego wykonywaniem akcji wynika, że niedozwolone jest użycie akcji z kilkoma kwalifikatorami czasowymi, a kwalifikator R ma charakter nadrzędny.

Sterownik PLC przetwarza graf *SFC*, kierując się regułami [6]:

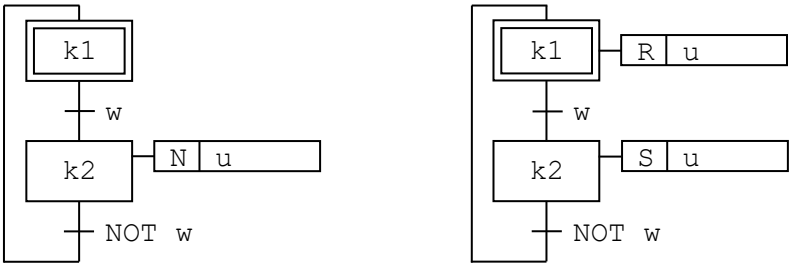
1) przy pierwszym wywołaniu programu aktywuje krok początkowy, w kolejnych cyklach dezaktywuje kroki, za którymi znajdują się tranzycje o spełnionych warunkach przejść i aktywuje kroki kolejne,
2) wykonuje akcje, których kwalifikatory wskazują, że nie będą już wykonywane w następnym cyklu,
3) wykonuje akcje, których kwalifikatory wymuszają ich wykonanie w bieżącym cyklu,
4) przetwarza wejścia i wyjścia, odczytując i zapisując wartości zmiennych do/z fizycznych wejść i wyjść sterownika,
5) oblicza wartości warunków przejść i kontynuuje, zaczynając od punktu 1.

## 10.5. Przykład 1

### 10.5.1. Wersja 1

Powyższy opis metody pozwala na interpretację programu przedstawionego na rys. 10.4. Program został narysowany na dwa sposoby, w wersji po lewej stronie wykorzystany został kwalifikator akcji `N`, a po prawej stronie kwalifikatory `R` i `S`. Niezależnie od wykorzystywanych kwalifikatorów w obydwu przypadkach działanie programu jest identyczne.

```
PROGRAM Przyklad1_1a
VAR
  w AT %IX1.1: BOOL;
  u AT %QX2.1: BOOL;
END_VAR
```

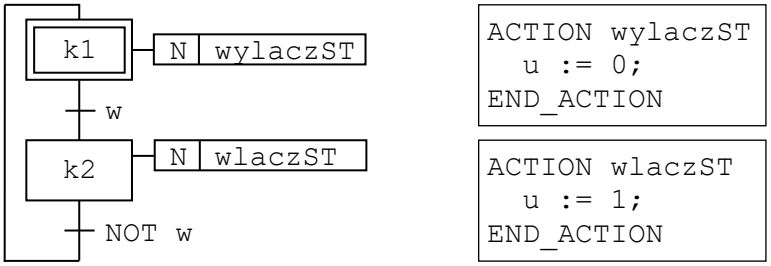



*Rys.10.4. Sterowanie pracą urządzenia wersja 1a*

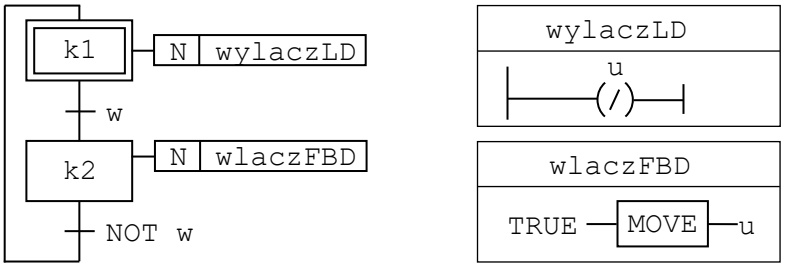
Program wykorzystuje dwie zmienne boolowskie: zmienna `w` odczytywana jest z pierwszego wejścia modułu wejściowego umieszczonego w pierwszym gnieździe sterownika, zmienna `u` ustawiana jest na pierwszym wyjściu modułu wyjściowego umieszczonego w drugim gnieździe sterownika. Warunki przejść zostały napisane w programie w języku *ST*, a akcje zostały zdefiniowane jako zmienne boolowskie. Program należy interpretować jako sterowanie pracą urządzenia włączanego przez operatora pojedynczym łącznikiem. Zmienna `u` określa stan urządzenia (`1` – pracuje, `0` – nie pracuje), zmienna `w` stan łącznika (`1` – włączony, `0` – wyłączony).

Korzystając z opisu bloku `ACTION_CONTROL`, przedstawionego na rys. 10.3 można określić wartość zmiennej `u` dla każdego kroku programu. Program w wersji pierwszej (lewy graf SFC) wykorzystuje akcję z kwalifikatorem `N` skojarzoną z krokiem `k2`. Kwalifikator `R` nie jest używany, więc na wejściu `R` bloku `ACTION_CONTROL` jest zawsze wartość `0`. Gdy aktywny jest krok `k1`, na wejście `N` podawana jest wartość `0`, więc wyjście bloku `Q`, a w konsekwencji zmienna `u` ma wartość `0`. W kroku `k2` na wejście `N` podawana jest wartość `1`, więc wyjście `Q` oraz zmienna `u` mają wartość `1`. W drugiej wersji programu (prawy graf SFC), gdy aktywny jest krok `k1` na wejście `R` podawane jest `1`, na `S` `0`, więc `u` jest `0`. Gdy aktywny jest krok `k2`, na wejścia `R` i `S` podawane są odpowiednio `0` i `1`, więc `u` ma wartość `1` (dodatkowo przetwarzanie sygnału wejściowego związane z kwalifikatorem `S` ustawia zmienną `u` trwale do momentu wystąpienia kwalifikatora `R`). Ostatecznie w kroku początkowym (`k1`) urządzenie jest wyłączone (`u = 0`). Po włączeniu łącznika (`w = 1`) krok `k1` traci aktywność i uaktywniany jest krok `k2`. W kroku `k2` urządzenie jest włączone (`u = 1`) dopóki operator nie zmieni stanu łącznika (`w = 0`), jeżeli taka zmiana nastąpi to krok `k2` utraci aktywność na rzecz kroku `k1`.

Na rys. 10.5 ten sam program został napisany z wykorzystaniem akcji zapisanych w języku *ST*, a na rys. 10.6 w językach *FBD* i *LD*. Deklaracje zmiennych nie zostały zmienione, więc pokazany został tylko graf *SFC* wraz z deklaracjami akcji. Jak wynika z poniższych przykładów, w przypadku gdy zadaniem akcji jest tylko ustawienie wartości zmiennej boolowskiej, wykorzystanie tej zmiennej w charakterze akcji wymaga znacznie mniejszego nakładu pracy (nie ma konieczności pisania akcji, zob. rys. 10.4).




*Rys.10.5. Sterowanie pracą urządzenia wersja 1b*

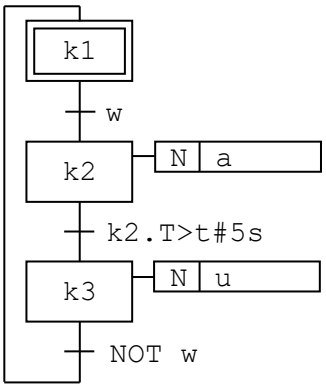



*Rys.10.6. Sterowanie pracą urządzenia wersja 1c*

### 10.5.2. Wersja 2

W zmodyfikowanej wersji programu założono, że przed włączeniem urządzenia na 5 sekund włącza się alarm. Do włączenia alarmu wykorzystane jest drugie wyjście modułu wyjściowego, z którym została związana zmienna `a`. Na rys. 10.7 przedstawione zostało przykładowe rozwiązane przygotowane na podstawie pierwszej wersji programu.

```
PROGRAM Przyklad1_2a
VAR
  w AT %IX1.1: BOOL;
  u AT %QX2.1: BOOL;
  a AT %QX2.2: BOOL;
END_VAR
```

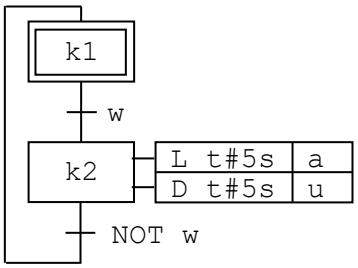



*Rys.10.7. Sterowanie pracą urządzenia wersja 2a*

W zaprezentowanym rozwiązaniu w kroku `k2` następuje włączenie alarmu realizowane przez *akcję nieprzechowywaną* w postaci zmiennej boolowskiej `a`. Tranzycja znajdująca się za tym krokiem z warunkiem przejścia `k2.T > t#5s` powoduje, że krok ten jest aktywny tylko przez 5 sekund, po tym czasie traci aktywność na rzecz kroku następnego, w którym następuje włączenie urządzenia (utrata aktywności przez krok `k2` powoduje wyłączenie alarmu).

Przedstawione rozwiązanie spełnia przyjęte założenia jednak, gdy operator włączy urządzenie i w czasie aktywności kroku `k2` zmieni zdanie – krok ten pozostanie aktywny przez założone 5 sekund. Dodatkowo krok `k3` nie zostanie całkowicie pominięty, musi zyskać aktywność, aby tranzycja znajdująca się za nim pozwoliła go dezaktywować i uaktywnić krok `k1`. Wykorzystując *akcje ograniczone i opóźnione w czasie* (kwalifikatory `L` i `D`), można przedstawić rozwiązanie pozbawione tych wad (rys. 10.8). Kwalifikator `L` powoduje wykonywanie akcji przez określony czas od chwili uaktywnienia kroku, kwalifikator `D` powoduje rozpoczęcie wykonywania akcji po określonym czasie.

Obydwa kwalifikatory przerywają wykonywanie akcji, gdy krok traci aktywność. W omawianym programie po uaktywnieniu kroku `k2` na 5 sekund włączany jest alarm (`a = 1`), a po jego wyłączeniu uruchamiane jest urządzenie (`u = 1`). Wyłączenie łącznika przez operatora skutkuje utratą aktywności kroku `k2`, co natychmiast przerywa wykonywaną akcję (wyłącza alarm lub urządzenie).

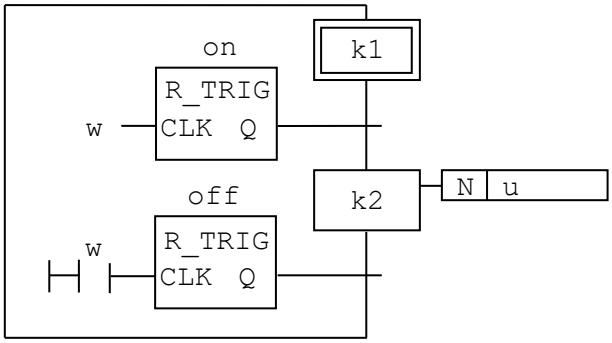



*Rys.10.8. Sterowanie pracą urządzenia wersja 2b*

### 10.5.3. Wersja 3

W kolejnej modyfikacji programu założono, że operator ma do dyspozycji łącznik monostabilny (przycisk), który służy zarówno do włączenia jak i wyłączenia urządzenia. W poprzednich wersjach operator sterował włączaniem za pomocą łącznika bistabilnego, który po naciśnięciu trwale zmieniał swój stan. Kolejna zmiana stanu łącznika mogła zostać wywołana tylko kolejnym działaniem operatora. Zamiana łącznika bistabilnego na przycisk, który jest w stanie *włączony* tylko tak długo jak jest wciskany, wymaga modyfikacji warunków przejścia w tranzycjach. W przykładzie, dla uproszczenia, powrócono do pierwotnej wersji programu z rys. 10.4.

Z przedstawionego opisu wynika, że urządzenie jest włączane i wyłączane w wyniku naciśnięcia przycisku, z którym w programie skojarzona jest zmienna `w`. Naciśnięcie przycisku powoduje zmianę zmiennej `w` z wartości `0` na `1`, może być więc wykryte za pomocą opisanego w punkcie 6.4.2 *detektora zbocza narastającego*. Rozwiązanie z wykorzystaniem tranzycji napisanych w językach *FBD* i *LD* zostało pokazane na rys. 10.9. Każda z tranzycji wykorzystuje własną instancję detektora zbocza – w języku *FBD* detektor otrzymał nazwę `on`, a w języku *LD* nazwę `off`.




*Rys.10.9. Sterowanie pracą urządzenia wersja 3*

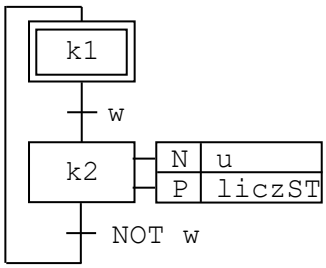
### 10.5.4. Wersja 4

W ostatniej modyfikacji programu założono, że każde włączenie urządzenia powinno być zliczane, ale dla uproszczenia powrócono do wersji programu z rys. 10.4. Rozwiązanie tak postawionego problemu wymaga wykorzystania dodatkowej zmiennej całkowitej pełniącej funkcję licznika włączeń, deklaracja zmiennych została uzupełniona o zmienną `ile` (rys. 10.10-10.12).

Do powiększania wartości zmiennej niezbędna jest w tym przypadku akcja, która powinna być skojarzona z krokiem `k2` odpowiedzialnym za włączanie urządzenia (powiększania zmiennej nie można zrealizować z pomocą zmiennej boolowskiej tak jak w przypadku włączania urządzenia). Ze względu na to, że powiększanie wartości zmiennej powinno odbywać się, tylko gdy krok zyskuje aktywność,

akcja powinna reagować na zbocze narastające sygnału opisującego aktywność kroku `k2`, co pozwala uniknąć powiększana wartości zmiennej w kolejnych cyklach aktywności kroku. W rozwiązaniu na rys. 10.10 wykorzystana została *akcja impulsowa* (zob. tab. 10.3) napisana w języku *ST* (akcja taka nie we wszystkich sterownikach będzie działać zgodnie z oczekiwaniami – zob. przypis w tab. 10.3).

```
PROGRAM Przyklad1_4a
VAR
  w AT %IX1.1: BOOL;
  u AT %QX2.1: BOOL;
  ile: INT:=0;
END_VAR
```

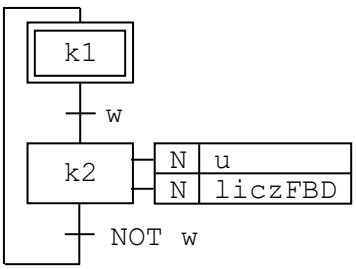


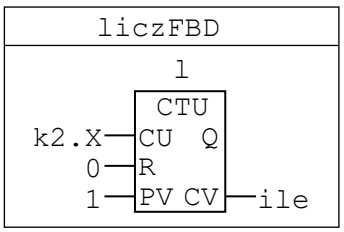

```
ACTION liczST
  ile := ile+1;
END_ACTION
```

*Rys.10.10. Sterowanie pracą urządzenia wersja 4a*

Alternatywne wersje programu przedstawiono na rys. 10.11 i 10.12. W rozwiązaniu na rys. 10.11 za detekcję zbocza narastającego odpowiada *licznik zliczający w górę* (zob. tab. 6.11), który z chwilą wykrycia zmiany stanu aktywności kroku `k2` inkrementuje automatycznie wartość zmiennej `ile`. W przypadku jawnego wykorzystania *detektora zbocza* tak jak na rys. 10.12 (zob. tab. 6.10) inkrementacja zmiennej jest realizowana z użyciem funkcji `ADD`.

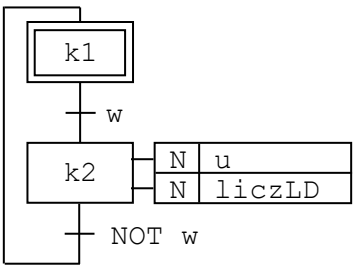
```
PROGRAM Przyklad1_4b
VAR
  w AT %IX1.1: BOOL;
  u AT %QX2.1: BOOL;
  ile: INT:=0;
  l: CTU;
END_VAR
```

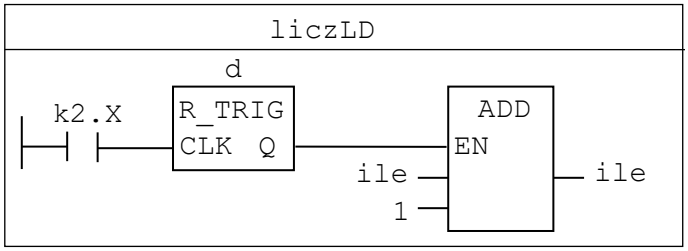



*Rys.10.11. Sterowanie pracą urządzenia wersja 4b*

```
PROGRAM Przyklad1_4c
VAR
  w AT %IX1.1: BOOL;
  u AT %QX2.1: BOOL;
  d: R_TRIG;
  ile: INT:=0;
END_VAR
```



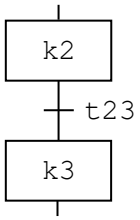

*Rys.10.12. Sterowanie pracą urządzenia wersja 4c*

## 10.6. Sekwencje

Zgodnie z regułami metody *SFC* niedozwolone jest łączenie elementów tego samego typu – w grafie *SFC* kroki i tranzycie muszą występować na przemian. Połączenie kroku i tranzycji tworzy tzw. *sekwencję*. Podobnie jak w klasycznych językach programowania dostępne są instrukcje sterujące, tak w metodzie SFC istnieją wzorcowe konstrukcje pozwalające na realizację typowych operacji takich jak wybór warunkowy lub pętla. Dodatkowo metoda SFC oferuje możliwość przetwarzania współbieżnego, dostarczając mechanizmów tworzenia i synchronizacji wątków.

### 10.6.1. Sekwencja pojedyncza

Najprostszą sekwencją jest *sekwencja pojedyncza* (rys. 10.13), w której po spełnieniu warunku przejścia w tranzycji za krokiem aktywowany jest krok następny.

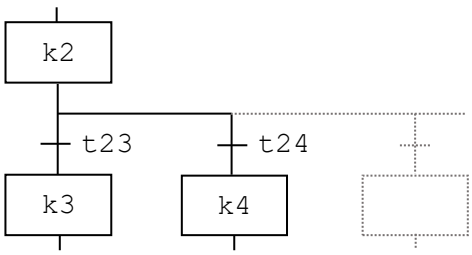



jeżeli aktywny jest krok `k2` i spełniony jest warunek przejścia `t23`, to uaktywniany jest krok `k3`

*Rys.10.13. Sekwencja pojedyncza*

### 10.6.2. Sekwencja wyboru

Jeżeli po wykonaniu określonego kroku zachodzi potrzeba realizacji jednej z kilku możliwych sekwencji kroków, to po symbolu kroku należy na grafie narysować *pojedynczą linię poziomą*, a pod nią zestaw tranzycji odpowiadających możliwym wyborom (rys. 10.14).




jeżeli aktywny jest krok `k2` i spełniony warunek przejścia:

- `t23` to uaktywniany jest krok `k3`
- `t24` to uaktywniany jest krok `k4`

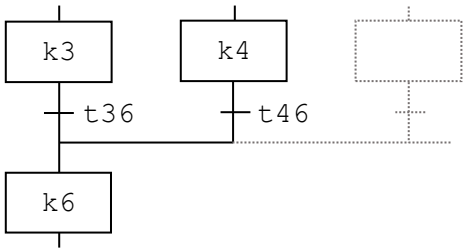
*Rys.10.14. Sekwencja wyboru – rozbieżność*

Sprawdzanie warunków przejścia jest przeprowadzane:

- od lewej strony (najpierw sprawdzana jest prawdziwość warunku pierwszego od lewej, jeżeli warunek jest fałszywy, sprawdzany jest kolejny itd.),
- na podstawie liczbowego priorytetu przypisanego przez użytkownika gałęziom (najpierw sprawdzana jest prawdziwość warunku gałęzi o najniższym numerze, jeżeli warunek jest fałszywy, sprawdzany jest warunek gałęzi o kolejnym numerze itd.).

Jeżeli jeden z warunków przejścia jest spełniony, to krok przed symbolem rozbieżności traci aktywność, a uaktywniany jest krok pod tranzycją ze spełnionym warunkiem przejścia. Krok ten pozostaje aktywny do spełnienia warunku tranzycji znajdującej się bezpośrednio pod nim. Jeżeli położone poniżej kroku gałęzie grafu nie będą ponownie prowadziły do kroku znajdującego się przed symbolem rozbieżności, to kroki znajdujące się w alternatywnych gałęziach grafu nigdy nie uzyskają aktywności.

Alternatywne gałęzie grafu można połączyć w jeden ciąg kroków, wykorzystując symbol zbieżności rysowany, podobnie jak symbol rozbieżności, w postaci *pojedynczej linii poziomej* (rys. 10.15).



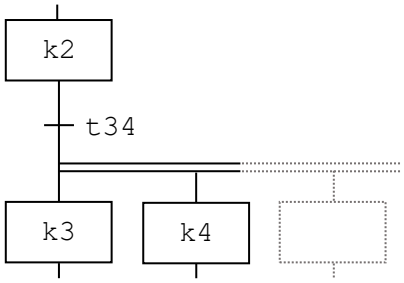

krok k6 jest uaktywniany, gdy:
- aktywny jest krok k3 i spełniony jest warunek przejścia t36
- aktywny jest krok k4 i spełniony jest warunek przejścia t46

*Rys.10.15. Sekwencja wyboru – zbieżność*

### 10.6.3. Sekwencja współbieżności

Jeżeli po wykonaniu określonego kroku zachodzi potrzeba jednoczesnej realizacji kilku sekwencji kroków, to po symbolu tranzycji związanej z tym krokiem należy na grafie narysować *podwójną linię poziomą*, a pod nią zestaw odpowiednich kroków (rys. 10.16).

Równoległe gałęzie grafu można połączyć w jeden ciąg kroków, używając symbolu zbieżności rysowanego w postaci *podwójnej linii poziomej* (rys. 10.17). Połączenie takie sprowadza się do synchronizacji operacji wykonywanych w krokach współbieżnych: kroki w gałęziach równoległych oczekują na spełnienie warunku tranzycji znajdującej pod symbolem zbieżności, a po jego spełnieniu tracą aktywność na rzecz kroku znajdującego się za tranzycją.




jeżeli aktywny jest krok k2 i spełniony warunek przejścia t34 to uaktywniane są jednocześnie kroki:

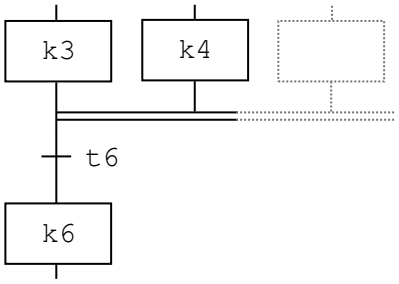
k3, k4, …

po uaktywnieniu kroków:

k3, k4, …

przetwarzanie dalszych kroków w gałęziach równoległych odbywa się niezależnie

*Rys.10.16. Sekwencja współbieżności – rozbieżność*




krok k6 jest uaktywniany gdy aktywne są kroki:

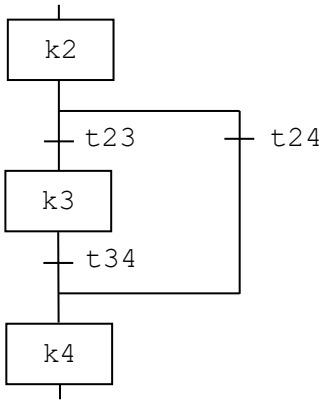
k3, k4, …

i spełniony jest warunek przejścia t6

*Rys.10.17. Sekwencja współbieżności – zbieżność*

### 10.6.4. Pomijanie sekwencji kroków

Jeżeli po wykonaniu określonego kroku, przy spełnieniu określonych warunków, zachodzi potrzeba pominięcia sekwencji kroków, to kroki te można pominąć, wykorzystując pustą gałąź alternatywną (rys. 10.18).

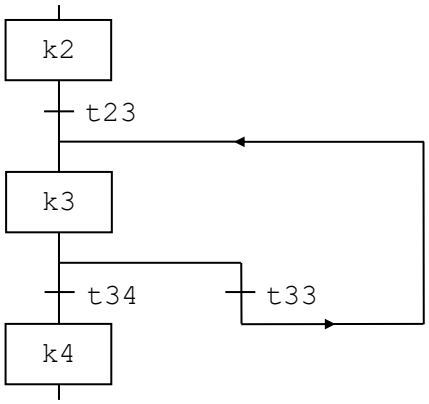



krok `k3` jest pomijany, gdy aktywny jest krok `k2` i spełniony jest warunek `t24`

*Rys.10.18. Pomijanie sekwencji*

### 10.6.5. Pętle

Pętlami norma nazywa sekwencje wyboru, w których występują gałęzie prowadzące do kroków położonych wyżej. W celu zwiększenia przejrzystości grafu, dopuszcza się w takim przypadku umieszczanie na połączeniach grotów wskazujących kierunek połączeń (rys. 10.19).




jeżeli aktywny jest krok `k3` i spełniony warunek przejścia:

- `t34` to uaktywniony zostanie krok `k4`
- `t33` to nastąpi powrót do kroku `k3`

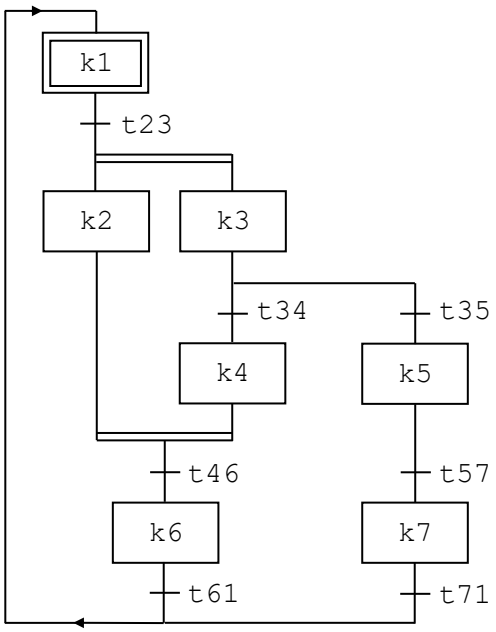
*Rys.10.19. Pętla*

### 10.6.6. Błędy

Omówione reguły metody *SFC* nie zabezpieczają użytkownika przed narysowaniem „*niebezpiecznego*” grafu. Możliwe błędy w budowie grafu dotyczą łączenia sekwencji współbieżności z sekwencją wyboru. Opuszczenie sekwencji współbieżności gałęzią sekwencji wyboru może prowadzić do:

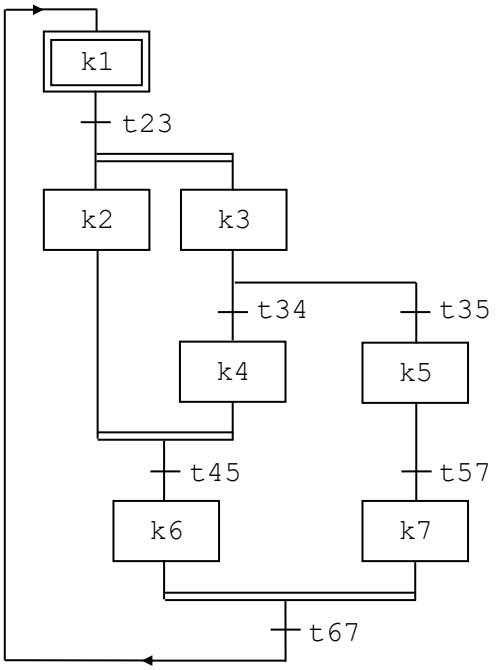
- niekontrolowanej liczby kroków aktywnych (rys. 10.20 góra),
- blokady programu (rys. 10.20 dół).

Kontrola poprawności grafu powinna być przeprowadzana przez system rzeczywistego sterownika PLC, nie dopuszczając do uruchomienia programu zawierającego błędy.




po uaktywnieniu kroków:
k2 i k3
jeśli spełniony będzie warunek t35, to kolejno uaktywnione zostaną kroki:
k5, k7, k1,
a krok k2 będzie ciągle czekał na spełnienie warunku t46,

jeśli spełniony będzie warunek t23, to krok k2 zostanie ponownie uaktywniony (krok k2 może być uaktywniany wielokrotnie)




po uaktywnieniu kroków:
k2 i k3
jeśli spełniony będzie warunek t35, nie będzie możliwa synchronizacja sekwencji, kolejno uaktywnione zostaną kroki:
k5, k7,
więc warunek przejścia tranzycji t67 nigdy nie będzie sprawdzony (nie jest możliwe aktywowanie kroku k6) – krok k7 zyska aktywność i nie odda jej żadnemu innemu krokowi programu)

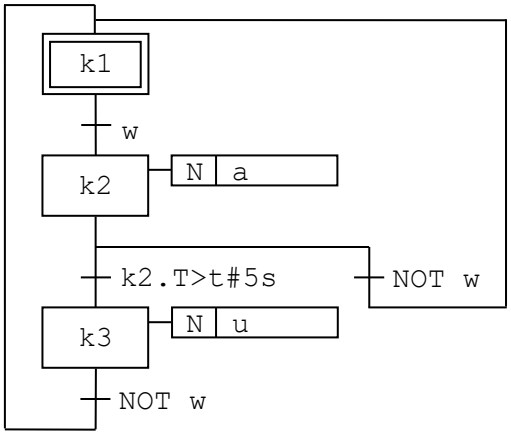
*Rys.10.20. Błędy w SFC*

## 10.7. Przykład 2

W drugiej wersji programu z przykładu 1 założono, że przed włączeniem urządzenia na 5 sekund włącza się alarm. Rozwiązanie przedstawione na rys. 10.7 nie pozwalało operatorowi na cofnięcie operacji włączenia przed upływem 5 sekund, w czasie których włączony był alarm.

Wadę tę wyeliminowało wykorzystanie *akcji ograniczonych i opóźnionych w czasie* (zob. rys. 10.8). Problem ten można również rozwiązać, korzystając z *sekwencji wyboru*, tak jak zostało to pokazane na rys. 10.21.

```
PROGRAM Przyklad2
VAR
  w AT %IX1.1: BOOL;
  u AT %QX2.1: BOOL;
  a AT %QX2.2: BOOL;
END_VAR
```

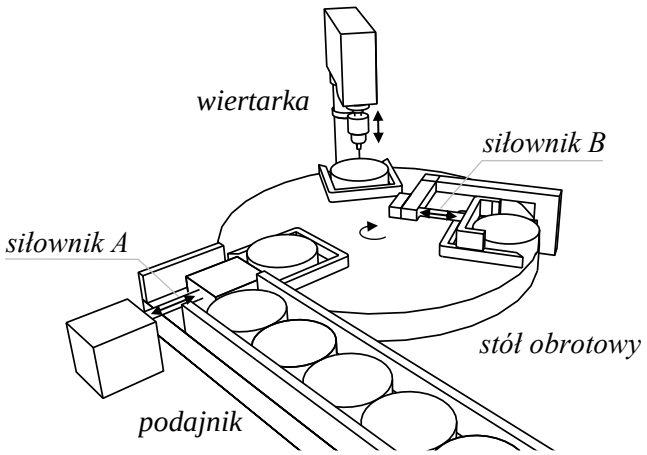



*Rys.10.21. Sterowanie pracą urządzenia wersja 5*

## 10.8. Przykład 3

W ramach podsumowania omówionych reguł metody *SFC* rozważony zostanie proces zautomatyzowanego wiercenia detali przedstawiony schematycznie na rys. 10.22. Każdy z detali jest kolejno obsługiwany na trzech stanowiskach:

- stanowisko 1 jest stanowiskiem załadunku, tutaj *siłownik A* przemieszcza detal z podajnika na *stół obrotowy*,
- stanowisko 2 jest właściwym miejscem, na którym odbywa się obróbka, *wiertarka* pracująca na tym stanowisku jest opuszczana, nawierca otwór, a na koniec jest podnoszona do pozycji startowej,
- stanowisko 3 jest stanowiskiem rozładunku, *siłownik B* zrzuca detal ze stołu do podstawionego tam pojemnika.



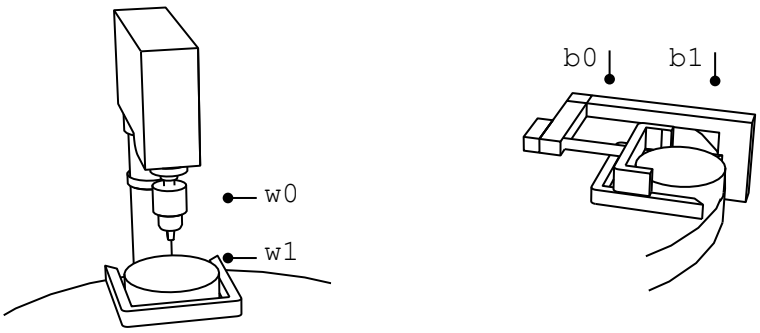

*Rys.10.22. Schemat dla procesu zautomatyzowanego wiercenia detali*

Każde ze stanowisk jest wyposażone w zestaw sensorów. Do sterowania pracą każdego z siłowników wykorzystywane są po dwa zawory pozwalające na wykonanie ruchu roboczego i powrotnego, sterowanie pracą wiertarki jest związane z włączeniem silnika i ustaleniem czy ruch powinien odbywać się „w dół" czy „w górę". Operacje na stanowiskach wykonywane są równolegle, po zakończeniu najdłuższej, stół jest ustawiany w kolejnej pozycji, umożliwiając obróbkę kolejnego detalu. Obrót stołu jest realizowany z pomocą napędu pozwalającego na zmianę jego ustawienia oraz czujnika wskazującego prawidłowe ustawienie stołu względem stanowisk. Lista wszystkich dostępnych sensorów wraz z sygnałami sterującymi została zestawiona w tab. 10.4, a usytuowanie sensorów na stanowiskach obróbczym i rozładunku zostało pokazane na rys. 10.23.

*Tab. 10.4. Dostępne sensory i sygnały sterujące*

| *Symbol* | *Opis* |
|---|---|
| *siłownik A* | |
| a0 | pozycja „wsunięty” |
| a1 | pozycja „wysunięty” |
| *siłownik B* | |
| b0 | pozycja „wsunięty” |
| b1 | pozycja „wysunięty” |
| *wiertarka* | |
| w0 | pozycja „góra” |
| w1 | pozycja „dół” |
| | |
| *stół* | |
| ok | pozycja robocza |

| *Symbol* | *Opis* |
|---|---|
| *siłownik A* | |
| Ar | ruch roboczy |
| Ap | ruch powrotny |
| *siłownik B* | |
| Br | ruch roboczy |
| Bp | ruch powrotny |
| *wiertarka* | |
| Wn | napęd |
| Wd | ruch „w dół” |
| Wg | ruch „w górę” |
| *stół* | |
| So | obrót |




*Rys.10.23. Usytuowanie sensorów*

Proces wiercenia:

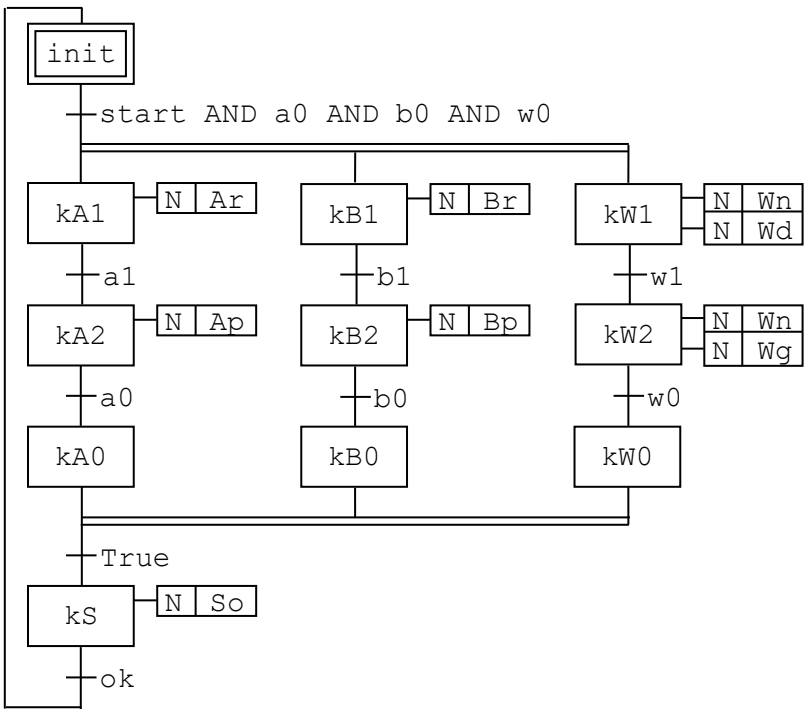
- zaczyna się od pozycji górnej sygnalizowanej przez czujnik `w0` (`w0 = 1`),
- następuje włączenie napędu (`Wn = 1`) i ustawienie ruchu „w dół” (`Wd = 1`),
- ruch „w dół” jest wyłączany (`Wd = 0`) w chwili gdy pozycję „dolną” zasygnalizuje czujnik `w1` (`w1 = 1`),
- po ustaniu ruchu „w dół” ustawiany jest ruch „w górę” (`Wg = 1`),
- napęd oraz ruch „w górę” jest wyłączany (`Wn = 0`, `Wg = 0`), w chwili gdy pozycję „górną” zasygnalizuje czujnik `w0` (`w0 = 1`).

Praca siłowników na stanowiskach załadunku i rozładunku przebiega podobnie. Na stanowisku rozładunku:

- siłownik na starcie ustawiony jest w pozycji „wsunięty” sygnalizowanej przez czujnik `b0` (`b0 = 1`),
- zawór sterujący pracą siłownika rozpoczyna ruch roboczy w wyniku ustawienia sygnału `Br` (`Br = 1`),
- ruch roboczy jest wyłączany, w chwili gdy pozycję „wysunięty” zasygnalizuje czujnik `b1` (`b1 = 1`),
- po ustaniu ruchu roboczego siłownik rozpoczyna ruch powrotny w wyniku ustawienia sygnału `Bp` (`Bp = 1`),
- ruch powrotny jest wyłączany, w chwili gdy pozycję „wsunięty” zasygnalizuje czujnik `b0` (`b0 = 1`).

Działanie całego systemu rozpoczyna wciśnięcie przez operatora łącznika `START`, przerwanie może nastąpić po każdym przestawieniu stołu, o ile łącznik ten zostanie wyłączony. Sieć *SFC* realizującą opisane zadanie przedstawiono na rys. 10.24. Operacje na każdym ze stanowisk zostały zrealizowane z pomocą trzech kroków, odpowiednio: (`kA1`, `kA2`, `kA0`) – siłownik A, (`kB1`, `kB2`, `kB0`) – siłownik B i (`kW1`, `kW2`, `kW0`) – wiertarka. Krok trzeci (`kA0`, `kB0` i `kW0`) w każdym z przypadków, nie został związany z żadną akcją, co oznacza, że żaden z sygnałów sterujących nie jest w tym czasie ustawiony. Kroki `kA0`, `kB0` i `kW0` pozwalają na realizację oczekiwania na zakończenie operacji na pozostałych stanowiskach. Warunek przejścia za symbolem zbieżności jest zawsze prawdziwy, ponieważ do synchronizacji operacji konieczne jest jedynie zakończenie czynności w gałęziach równoległych. Po zakończeniu tych operacji przestawiany jest stół obrotowy i proces może rozpocząć się od początku.

```
PROGRAM Przyklad3
VAR
  start AT %IX1.1: BOOL;
  a0 AT %IX1.2: BOOL;    a1 AT %IX1.3: BOOL;
  b0 AT %IX1.4: BOOL;    b1 AT %IX1.5: BOOL;
  w0 AT %IX1.6: BOOL;    w1 AT %IX1.7: BOOL;
  ok AT %IX1.8: BOOL;
  Ar AT %QX2.1: BOOL;    Ap AT %QX2.2: BOOL;
  Br AT %QX2.3: BOOL;    Bp AT %QX2.4: BOOL;
  Wn AT %QX2.5: BOOL;
  Wd AT %QX2.6: BOOL;    Wg AT %QX2.7: BOOL;
  So AT %QX2.8: BOOL;
END_VAR
```




*Rys.10.24. Wiercenie detali*

## 10.9. Tłumaczenie grafu SFC

Programy zapisane w postaci grafów *SFC* w sposób czytelny i precyzyjny opisują algorytmy sterowania procesów z pomocą sterowników PLC. Metoda *SFC* jest opisywana w normie IEC 61131-3, jednak oprogramowanie sterowników nie zawsze udostępnia możliwość jej wykorzystania do zapisu algorytmu sterowania. W takim przypadku przydatne stają się techniki tłumaczenia grafu *SFC* na pozostałe języki programowania sterowników [2]. Tłumacząc program zapisany w *SFC*, należy zadbać o to, żeby każdy krok grafu, który zostanie uaktywniony w bieżącym cyklu pozostał aktywny do końca tego cyklu – zob. reguły przetwarzania grafu *SFC* opisane na końcu p. 10.4.

### 10.9.1. Tłumaczenie na język LD

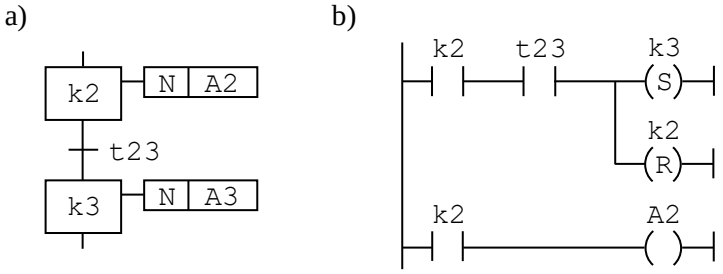
Graf *SFC* można przetłumaczyć na język *LD,* wykorzystując [2]:

- pomocnicze zmienne boolowskie reprezentujące aktywność każdego z kroków grafu (np. aktywność kroku `k2` będzie opisywana zmienną `k2`, która w przypadku gdy krok zostanie jest uaktywniony otrzyma wartość `1` (`TRUE`), a gdy utraci aktywność otrzyma wartość `0` (`FALSE`)),
- cewki zatrzaskiwane: ustawiającą i kasującą.

Dla uproszczenia, przy opisie zasad tłumaczenia przyjęto, że akcje skojarzone z krokami są zmiennymi boolowskimi. Przedstawiona metoda może być stosowana również, w przypadku gdy akcje zapisane są w postaci funkcji czy bloków funkcjonalnych – w takim przypadku muszą być one aktywowane w miejscu ustawiania wartości zmiennych reprezentujących akcje.

Tłumacząc sekwencję prostą, zapisuje się dwa obwody (rys. 10.25):

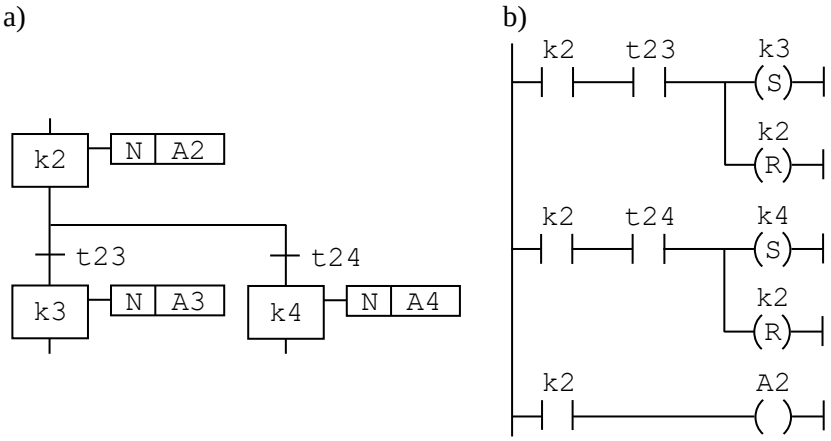
- pierwszy obwód pełni funkcję *sterującą* – zapisuje działanie zmieniające aktywność kroków, w chwili gdy aktywny jest krok `k2` i spełniony jest warunek przejścia tranzycji znajdującej się za krokiem,
- drugi obwód pełni funkcję *wykonawczą* – wykonuje akcję skojarzoną z krokiem `k2`, jeżeli krok ten jest krokiem aktywnym.



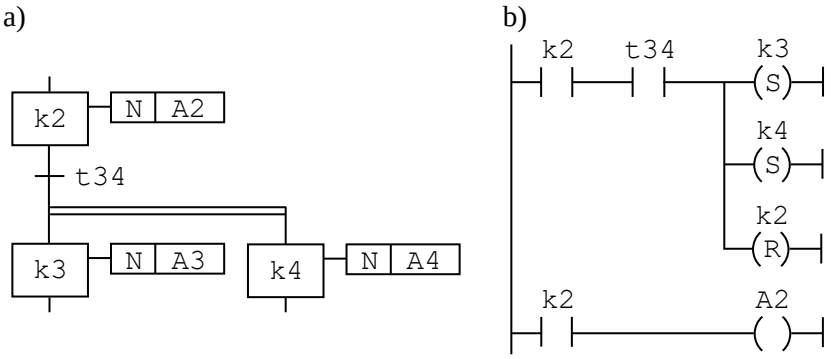

*Rys.10.25. Tłumaczenie sekwencji prostej na język LD*

Podobne zasady wykorzystywane są do tłumaczenia sekwencji wyboru i sekwencji współbieżności (rys. 10.26, 10.27 i 10.28). W każdym przypadku rysowane są obwody pełniące funkcję sterującą i wykonawczą. Na rysunkach nie pokazano sposobu tłumaczenia symbolu zbieżności dla sekwencji wyboru, w tym przypadku należy wykorzystywać identyczne zasady jak podczas tłumaczenia sekwencji pojedynczej, które powinny być zastosowane dla wszystkich łączonych gałęzi alternatywnych.

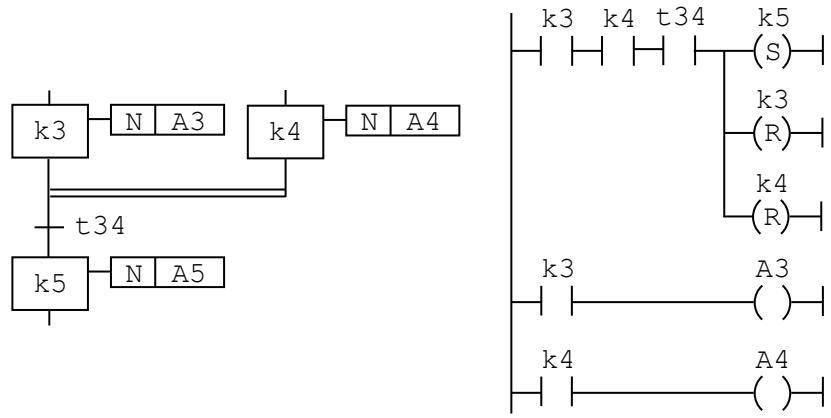
Stosowanie opisanych reguł nie gwarantuje jeszcze całkowitej zgodności programu w języku *LD* z oryginalnym programem zapisanym w postaci grafu *SFC*. Jak pokazano w [2] zasady tłumaczenia należy uzupełnić, wprowadzając dodatkowe reguły zabezpieczające przed *przeskokami* (*lawinami*).




*Rys.10.26. Tłumaczenie sekwencji wyboru na język LD*



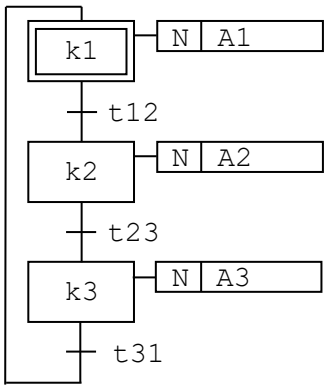

*Rys.10.27. Tłumaczenie rozbieżności współbieżności na język LD*




*Rys.10.28. Tłumaczenie zbieżności współbieżności na język LD*

Załóżmy na początek, że do przetłumaczenia jest prosty program zbudowany z trzech kroków odpowiadający grafowi z rys. 10.7. Graf ten w uproszczonej wersji został pokazany na rys. 10.29. Oryginalne warunki przejść zostały zastąpione zmiennymi boolowskimi `t12`, `t23` i `t31`, a kroki skojarzono ze zmiennymi `A1`, `A2` i `A3`.

```
PROGRAM Przyklad4
VAR
  t12,t23,t31: BOOL;
  A1, A2, A3: BOOL;
END_VAR
```




*Rys.10.29. Graf z krokami połączonymi szeregowo*

Stosując omówione zasady, graf ten został przetłumaczony na język *LD* (rys. 10.30a). Wprowadzono pomocnicze zmienne boolowskie (`k1`, `k2` i `k3`) reprezentujące aktywność każdego z kroków grafu. W bloku deklaracji zmiennych zmiennej `k1` przypisano wartość początkową `TRUE`, co nadało krokowi pierwszemu aktywność na początku wykonania programu.

Program w tej wersji nie jest jednak zabezpieczony przed przeskokami. Oznacza to, że w przypadku gdy aktywny jest krok `k1` i spełnione są warunki przejść w tranzycjach `t12` i `t23`, to w jednym cyklu programowym krok `k2` zyska i straci aktywność na rzecz kroku `k3`.

Jeżeli kolejne pary tranzycji programu mają charakter wykluczający, tzn. nie mogą być jednocześnie spełnione, działanie programu w postaci grafu *SFC* i programu przetłumaczonego na język *LD* z zachowaniem omówionych reguł tłumaczenia będzie identyczne. W przypadku gdy kolejne tranzycie mogą być jednocześnie spełnione, potrzebne są dodatkowe reguły zabezpieczające program przed przeskokami. W pracy [2] autor proponuje:

- tłumaczenie grafu w kierunku od dołu do góry (eliminuje to przeskoki pomiędzy krokami położonymi wyżej a krokami położonymi niżej),
- wprowadzenie pomocniczych zmiennych blokujących przeskoki po krokach uaktywnianych z kroków położonych niżej, blokowanie przeskoku powinno być również ustawione dla pierwszego uruchomienia programu (sposób korzystania z blokad zostanie opisany poniżej),
- podział programu na *część sterującą* z obwodami odpowiedzialnymi za zmianę aktywności kroków i *część wykonawczą* z obwodami realizującymi akcje skojarzone z krokami, w *części wykonawczej* należy umieścić tyle obwodów, ile jest zmiennych ustawianych w poszczególnych krokach (a nie tyle, ile jest kroków), zapisując wyrażenia zapewniające odpowiednią wartość każdej zmiennej, pozwala to uniknąć błędu programu, w przypadku gdy określona zmienna jest wykorzystywana w charakterze akcji w więcej niż jednym kroku.

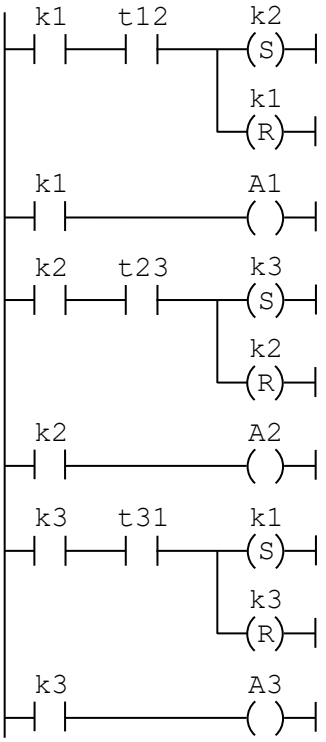
W programie na rys. 10.30b pierwszych pięć obwodów realizuje funkcje *sterujące*, ostatnie trzy funkcje *wykonawcze*. Dwa ostatnie obwody w części *sterującej* są częścią systemu blokowania: blokując potencjalny przeskok na starcie programu i przeskok z kroku `k3` do kroku `k1`.

Blokada przeskoku na starcie programu realizowana jest z pomocą zmiennej `k01`, której przypisano wartość początkową `TRUE`. Zmienna ta w części sterującej zeruje swoją wartość i ustawia zmienną `k1`, co aktywuje krok `k1`. Dodatkowo aktywowanie kroku `k1` na końcu części sterującej zapobiega wykonaniu obwodu, który mógłby uaktywnić krok kolejny.

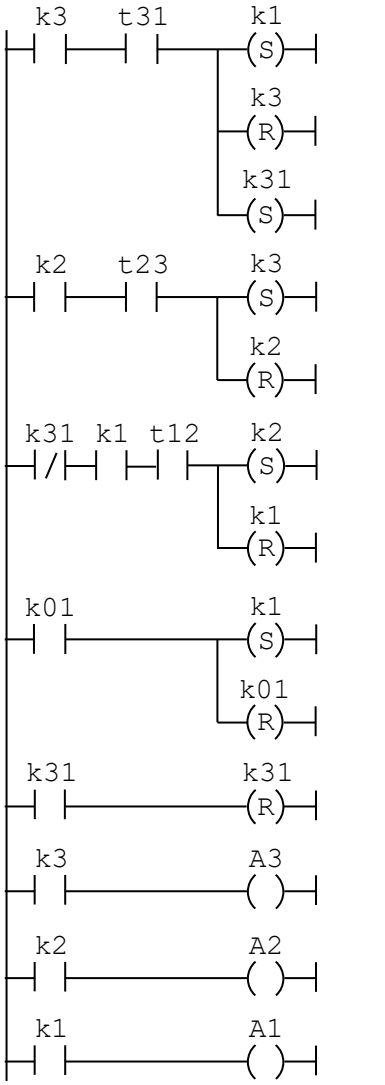
Przeskok z kroku `k3` do kroku `k1` blokuje zmienna `k31`, która jest:

- ustawiana w części sterującej w obwodzie, który opisuje przejście z kroku `k3` do kroku `k1` (pierwszy obwód *części sterującej*),
- wykorzystywana w części sterującej w obwodzie, który opisuje przejście z kroku `k1` do kroku `k2` (trzeci obwód *części sterującej*),
- zerowana przez ostatni obwód *części sterującej*, co pozwala na wykonanie przejścia z kroku `k1` do kroku `k2` w kolejnym cyklu realizacji programu.

```
PROGRAM Przyklad4a
VAR
  t12,t23,t31: BOOL;
  A1, A2, A3: BOOL;
  k1: BOOL := TRUE;
  k2, k3: BOOL;
END_VAR
```




```
PROGRAM Przyklad4b
VAR
  t12,t23,t31: BOOL;
  A1, A2, A3: BOOL;
  k01: BOOL := TRUE;
  k1,k2,k3,k31: BOOL;
END_VAR
```



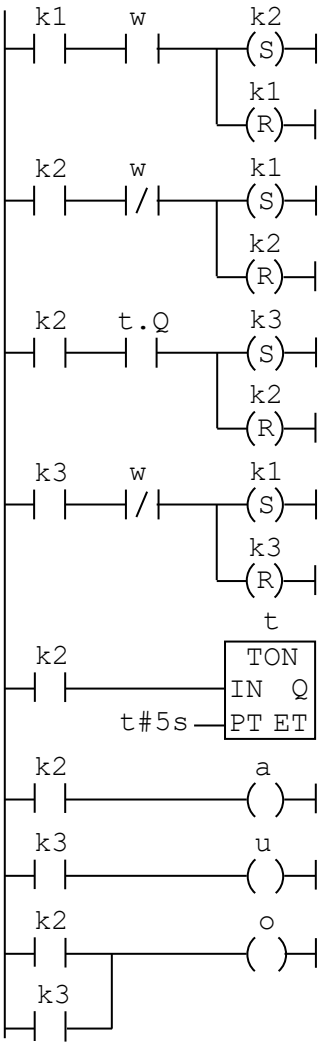

*Rys.10.30. Program z rys. 10.29 w wersji a) niezabezpieczonej, b) zabezpieczonej przed przeskokami*

Po wyjaśnieniu podstawowych zasad dotyczących tłumaczenia na język *LD* przetłumaczony zostanie program z przykładu 2 przedstawiony w punkcie 10.7. Operator zmieniając stan łącznika miał możliwość włączania i wyłączania urządzenia. Włączenie urządzenia było poprzedzane trwającym 5 sekund alarmem, a zmiana stanu łącznika po zainicjowaniu alarmu pozwalała na przerwanie operacji włączania urządzenia.

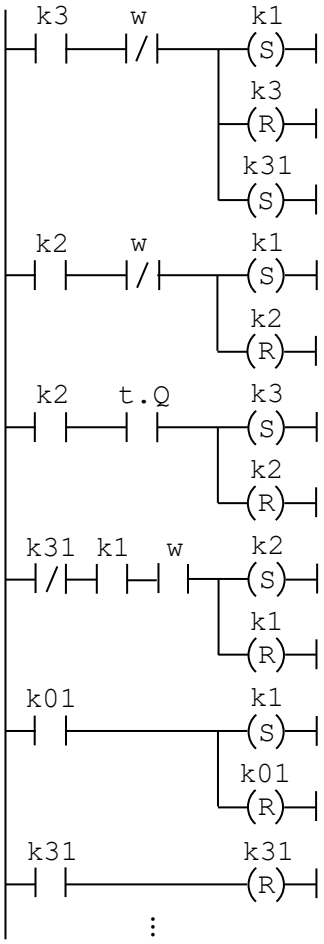
Realizację dwóch alternatywnych sposobów umożliwiających przerwanie kroku odpowiedzialnego za sygnalizację alarmu zrealizowano, wykorzystując *sekwencję wyboru* (zob. rys. 10.21), co pozwoliło na wykorzystanie w przykładzie reguły tłumaczenia sekwencji rozbieżności (zob. rys. 10.26). W przedstawionym dalej programie założono również, że w trakcie działania alarmu i w trakcie pracy

urządzenia włączany jest dodatkowy ostrzegawczy sygnał świetlny. Takie uzupełnienie założeń pozwoliło na zastosowanie reguły dotyczącej wykorzystania tej samej zmiennej w charakterze akcji w więcej niż jednym kroku. Program został przedstawiony w dwóch wersjach: niezabezpieczonej i zabezpieczonej przed przeskokami (rys. 10.31a i 10.31b). W rozważanym przypadku wystąpienie przeskoku nie jest jednak możliwe, a więc obydwie wersje programu są prawidłowe.

```
PROGRAM Przyklad5a
VAR
  w AT %IX1.1: BOOL;
  u AT %QX2.1: BOOL;
  a AT %QX2.2: BOOL;
  o AT %QX2.3: BOOL;
  k1: BOOL := TRUE;
  k2,k3: BOOL; t: TON;
END_VAR
```




```
PROGRAM Przyklad5b
VAR
  w AT %IX1.1: BOOL;
  u AT %QX2.1: BOOL;
  a AT %QX2.2: BOOL;
  o AT %QX2.3: BOOL;
  k01: BOOL := TRUE;
  k31: BOOL;
  k1,k2,k3: BOOL;
  t: TON;
END_VAR
```




*Rys.10.31. Sterowanie pracą urządzenia – przykład 2, wersje: a) niezabezpieczona i b) zabezpieczona przed przeskokami*

*Część sterującą* programu w wersji niezabezpieczonej tworzą pierwsze cztery obwody z obwodami drugim i trzecim realizującymi *sekwencję wyboru*. Obwód piąty pełni *funkcję pomocniczą*, uruchamiając *timer TON* w chwili uaktywnienia kroku `k2`. Zastosowanie *timera* pozwala na zmierzenie czasu aktywności tego kroku: czas od uruchomienia *timera* (tzn. od włączenia sygnału na wejściu `IN`) jest dostępny na jego wyjściu `ET` i może być wykorzystany do skonstruowania odpowiedniego warunku przejścia (alarm zgodnie z założeniami powinien trwać 5 sekund) – takie podejście wymagałoby jednak wykorzystania bloku realizującego porównania arytmetyczne. W programie z rys. 10.31a w obwodzie realizującym przejście z kroku `k2` do kroku `k3` (trzeci obwód *części sterującej*) wykorzystany został sygnał z wyjścia `Q` *timera* – wyjście to jest włączane po czasie `PT` (tzn. po 5 sekundach) do włączenia sygnału na wejściu `IN` *timera*, sygnał ten daje więc wartość `TRUE` (`1`), gdy krok `k2` powinien stracić swoją aktywność. *Część wykonawczą* programu tworzą ostatnie trzy obwody. Warto zauważyć, że zgodnie z opisanymi tutaj regułami, *część wykonawcza* programu składa się z tylu obwodów, ile jest zmiennych wyjściowych. Rozdzielenie ostatniego obwodu, odpowiedzialnego za aktywność dodatkowego ostrzegawczego sygnału świetlnego, na dwa oddzielne obwody spowodowałoby błędne działanie programu (sygnał ostrzegawczy byłby włączony tylko w czasie aktywności kroku `k3`).

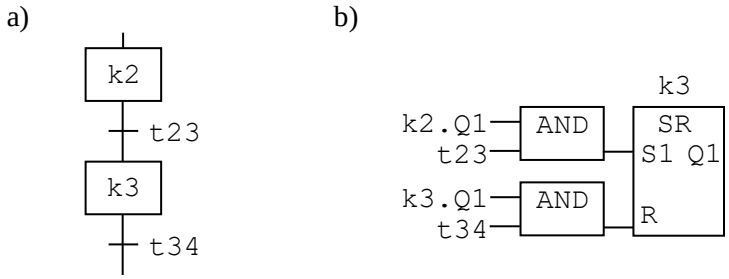
W programie z rys. 10.31b pokazana została tylko *część sterująca*, *części pomocnicza* i *wykonawcza* są w takie same jak na rys. 10.31a, oznacza to, że program ten należy uzupełnić, dodając cztery ostatnie obwody z wersji niezabezpieczonej przed przeskokami.

W omówionej metodzie tłumaczenia wykorzystane zostały jedynie *akcje nieprzechowywane* w postaci zmiennych boolowskich. Nie omówiono zasad tłumaczenia w przypadku akcji z innymi kwalifikatorami, które musiałyby być najpierw zastąpione *akcjami nieprzechowywanymi*. Nie opisano również problemów, które mogą się pojawić, w przypadku gdy graf *SFC* wykorzystuje liczniki i timery czy problemów przy tłumaczeniu algorytmów wielografowych. Zagadnienia te są szczegółowo omówione w pracy [2].

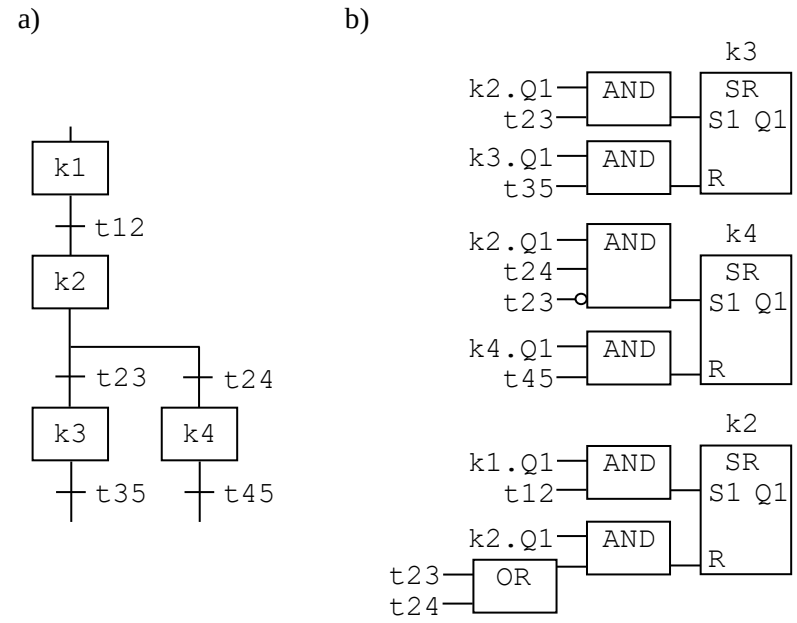
### 10.9.2. Tłumaczenie na język FBD

W przypadku tłumaczenia grafu *SFC* na język *FBD* do opisu zmian aktywności kroków wykorzystywane są przerzutniki SR. Wykorzystywane są zarówno przerzutniki z dominującym wejściem zerującym, jak i z dominującym wejściem ustawiającym. W pracy [2] autor wykorzystuje przerzutniki z dominującym ustawianiem w języku *LD* sterownika, który nie zawiera cewek ustawiającej i kasującej. Przedstawioną tam metodę można wprost zastosować do tłumaczenia grafu na język *FBD*.

Na rys. 10.32 i 10.33 pokazane zostały reguły tłumaczenia umożliwiające odpowiednią zmianę aktywności kroków dla sekwencji prostej i sekwencji wyboru. Przedstawione obwody pełnią więc *funkcję sterującą* (zob. p. 10.9.1). Aktywność kroków w tym podejściu reprezentowana jest przez stan sygnału wyjściowego przerzutnika, którego nazwa odpowiada nazwie odpowiedniego kroku grafu *SFC*. Warunki na wejściach ustawiającym i zerującym przerzutnika odpowiadają warunkom przejść powodującym odpowiednio aktywowanie i dezaktywowanie kroku.




*Rys.10.32. Tłumaczenie sekwencji prostej na język FBD*




*Rys.10.33. Tłumaczenie sekwencji wyboru na język FBD*

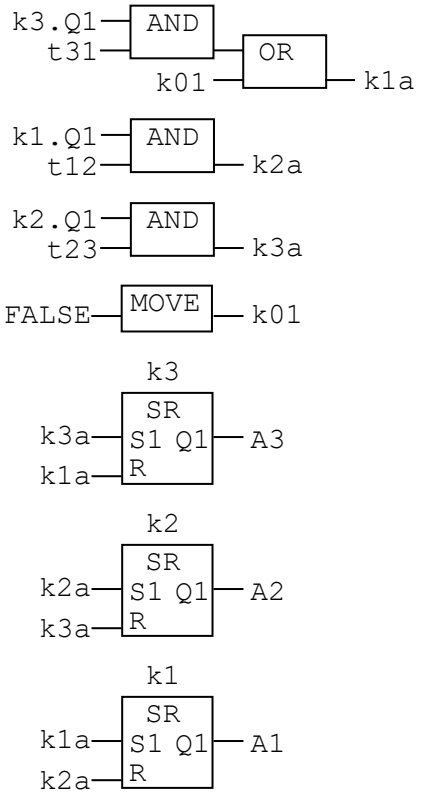
W podobny sposób przebiega również tłumaczenie sekwencji współbieżności. Przeskoki na gałęziach prowadzących w górę można eliminować, tak jak przy tłumaczeniu grafu na język *LD*, z pomocą dodatkowych zmiennych blokujących.

Zastosowanie opisanych reguł dla grafu z rys. 10.29 prowadzi do programu przedstawionego na rys. 10.34. W programie tym w celu wyeliminowania konieczności powielania tych samych wyrażeń wykorzystywanych do aktywowania i dezaktywowania kroków wprowadzono pomocnicze zmienne boolowskie (`k1a`, `k2a` i `k3a`) reprezentujące warunki uaktywniania kroków. Początkowe obwody programu, w których zmiennym tym nadawane są wartości odpowiadające aktualnemu stanowi grafu tworzą jego *część przygotowawczą*. Podobnie jak w programie z rys. 10.30, tutaj też wykorzystywana jest zmienna `k01` będąca znacznikiem pierwszego uruchomienia programu. Zmiennej tej przypisano wartość początkową `TRUE`, a w kolejnych cyklach wartość zmiennej jest ustawiana na `FALSE`. W programie z rys. 10.34 zrezygnowano natomiast z pisania oddzielnej *części wykonawczej*, zmienne boolowskie `A1`, `A2` i `A3`, będące akcjami grafu z rys. 10.29, otrzymują wartości na podstawie stanów sygnałów wyjściowych przerzutników reprezentujących kroki grafu.

Na rys. 10.35 przedstawiony został program realizujący zadanie opisane w punkcie 10.7, którego graf SFC został pokazany na rys. 10.21. W zadaniu tym, podobnie jak w punkcie poprzednim, w krokach `k2` i `k3` generowany jest dodatkowy ostrzegawczy sygnał świetlny. Program został przetłumaczony z wykorzystaniem tych samych zasad, które zostały użyte do tłumaczenia programu z rys. 10.34. Podobnie jak przy tłumaczeniu na język LD, do odtworzenia warunku przejścia w tranzycji za krokiem `k2` konieczne było zastosowanie *timera*. Sygnał z wyjścia `Q` *timera* został wykorzystany w trzecim obwodzie programu do nadania wartości pomocniczej zmienne boolowskiej `k3a`, reprezentującej warunki uaktywniania kroku `k3`. Warto zauważyć, że wyjścia przerzutników reprezentujących drugi i trzeci krok grafu *SFC* zostały wykorzystane bezpośrednio, podobnie jak na rys. 10.34, do ustalenia wartości sygnałów wyjściowych sterujących pracą alarmu i urządzenia. W tym przypadku konieczne jednak było napisanie *części wykonawczej* (ostatni obwód programu), w której przypisana została odpowiednia wartość zmiennej reprezentującej dodatkowy sygnał ostrzegawczy.

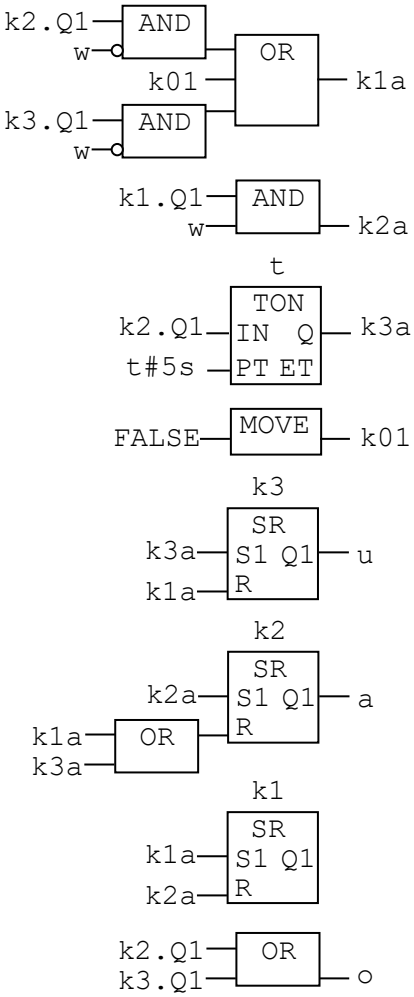
Przedstawiony sposób tłumaczenia grafu *SFC* na język *FBD* nie jest jedynym możliwym. W pracy [22] aktywność kroków modyfikowana jest z pomocą przerzutników SR, których wyjścia połączone są z detektorami zbocza opadającego. Na wejście ustawiające przerzutnika reprezentującego krok następny podawany jest sygnał z detektora kroku poprzedniego, co powoduje jego uaktywnienie po utracie aktywności przez krok poprzedni. Niektórzy autorzy wykorzystują do opisu aktywności kroku przerzutnik D, który można zaimplementować, wykorzystując przerzutnik SR i blok `NAND`.

```
PROGRAM Przyklad4c
VAR
  t12,t23,t31: BOOL;
  A1, A2, A3:  BOOL;
  k01: BOOL := TRUE;
  k1a,k2a,k3a: BOOL;
  k1, k2, k3:  SR;
END_VAR
```




*Rys.10.34. Program z rys. 10.29*

```
PROGRAM Przyklad5c
VAR
  w AT %IX1.1: BOOL;
  u AT %QX2.1: BOOL;
  a AT %QX2.2: BOOL;
  o AT %QX2.3: BOOL;
  k01: BOOL := TRUE;
  k1a,k2a,k3a: BOOL;
  k1, k2, k3:  SR;
END_VAR
```




*Rys.10.35. Program z rys. 10.21*

## 10.10. Zadania

Przemyśl ponownie zadania z punktów 7.6 i 8.6 napisz programy, wykorzystując metodę SFC.