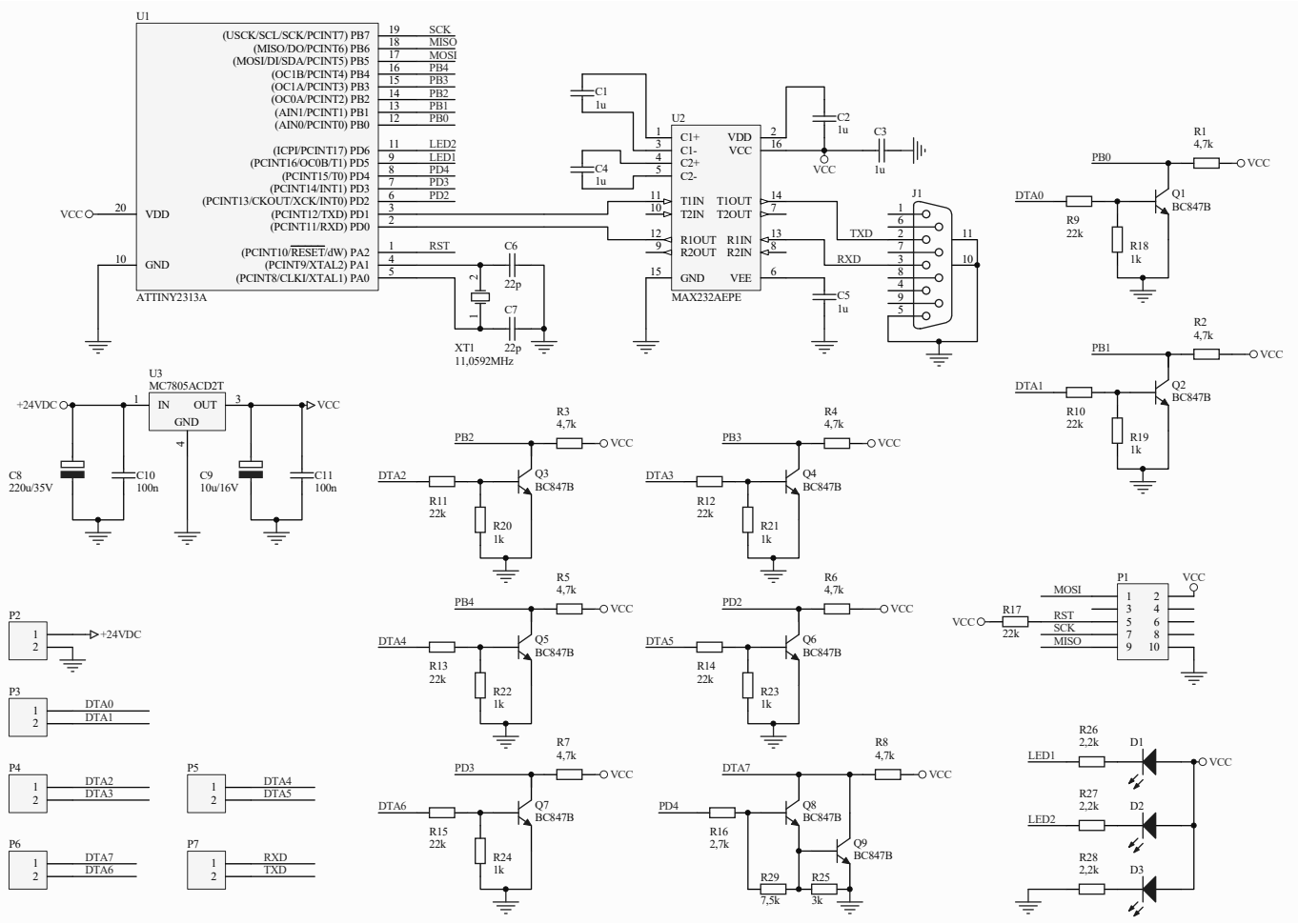
# Współpraca LOGO! z urządzeniem zewnętrznym – interfejs RS232

*W wielu zastosowaniach miniaturowy sterownik LOGO! jest bardzo dobrym wyborem. Ma obudowę, wyświetlacz LCD, nieskomplikowaną klawiaturę, wbudowany zegar RTC, a za pomocą LOGO! Soft Comfort można łatwo tworzyć aplikacje. Wejścia analogowe, wyjścia przekaźnikowe lub tranzystorowe świetnie nadają się do realizowania wielu aplikacji w automatyce domowej i nie tylko. Dodatkową zaletą jest możliwość dołączania modułów rozszerzeń, w tym również realizujących złożone funkcjonalności.*

Sterownik LOGO! ma jedną, niezmiernie cenną, ale za to bardzo często niezauważaną zaletę. Za umiarkowaną cenę otrzymujemy urządzenie elektroniczne z interfejsem użytkownika, które ma parametry i właściwości gwarantowane przez producenta – firmę Siemens. Trudno zaprzeczyć, że jest to jedna z najważniejszych firm wytwarzających urządzenia przemysłowe. Ponadto, jest nie tylko producentem, ale również kreuje nowe standardy. Można nawet zaryzykować twierdzenie, że jeśli firma Siemens wprowadza jakąś nową technologię, to wkrótce staje się ona standardem światowym. A więc używając LOGO! mamy wsparcie i gwarancję jakości Siemensa, co jest potwierdzone odpowiednimi certyfikatami.

Z punktu widzenia konstruktora, który chciałby użyć LOGO! w jakimś tworzonym przez siebie urządzeniu, ma ono jedną wadę – jest systemem zamkniętym. Owszem, istnieją programy, które w jakiś sposób pobierają dane z pamięci sterownika, ale robią to za pośrednictwem interfejsu Ethernet, a więc wymagają LOGO! v7 lub v8. Ponadto, „wydobycie" danych za pośrednictwem Ethernetu wymaga odpowiednich zasobów sprzętowych i programowych po stronie urządzenia, które ma zamiar z tych danych korzystać. Owszem, uzyskujemy w ten sposób sporo ciekawych możliwości, ale czas opracowania gotowego produktu znacznie wydłuża się, ponieważ musimy wykonać stos TCP/IP i rozszyfrować protokół komunikacyjny przy skąpej dostępności do informacji. Jeszcze trudniej jest w wypadku




**Rysunek 1. Schemat ideowy modułu interfejsowego**

lokalnego protokołu komunikacyjnego interfejsu, którego LOGO! używa do wymiany danych z zewnętrznymi modułami rozszerzeń. Gdyby firma Siemens udostępniła ten protokół, to zapewne znalazłoby się wiele firm i osób, które budowałyby własne moduły rozszerzeń. Niestety, dane są utajnione i dlatego trzeba uciekać się do pewnych „sztuczek". Jedną z nich opisano dalej.

## Własny interfejs do LOGO?

„Uzbrojenie" LOGO! w interfejs szeregowy np. RS485 lub RS232, nawet jedynie przesyłający dane z LOGO! do urządzenia zewnętrznego, takiego jak wyświetlacz, moduł GSM lub inne, nie jest łatwe. Jak wspomniano, protokół komunikacyjny, którym CPU porozumiewa się z modułami jest tajemnicą producenta. Dlatego też – posługując się metodą standardową – jedyną możliwością jest użycie ogólnie dostępnych wejść/wyjść sterownika oraz wykonanie odpowiedniego oprogramowania.

Sterownik LOGO! ma tylko wejścia/wyjścia binarne, a niektóre wersje mają też wejścia analogowe. Dołączając dodatkowe moduły można dodać do LOGO! wyjścia analogowe, napięciowe lub prądowe, wejścia binarne lub analogowe, dołączyć termistory PT100 i PT1000. Można również do wyjść przekaźnikowych dodać tranzystorowe, do tranzystorowych - przekaźnikowe itd. Niestety, wśród modułów na próżno szukać takich z interfejsem szeregowym RS485 lub RS232. Siłą rzeczy, nie na też dla nich wsparcia w środowisku LOGO! Soft Comfort.

Dodając interfejs zewnętrzny trzeba też zadać sobie pytanie o to, do czego będzie używany. Nie da się za pomocą wejść/wyjść binarnych uzyskać dostępu do wszystkich zmiennych zawartych w pamięci CPU. Nie da się też przesyłać całych tablic, chociaż można sobie wyobrazić implementację takiego oprogramowania. Z powodzeniem można natomiast przesłać do modułu zewnętrznego stan licznika lub komunikat SMS, ale... tego też nie da się zrobić w „zwykły sposób". Tu jednak mogą nam przyjść w sukurs techniki rodem z nie tak bardzo odległej przeszłości i przyda się wiedza z zakresu kombinatoryki układów logicznych.

Korzystając z opisanej metody można z powodzeniem wykonać np. interfejs do modułu GSM, ale cały interpreter komend będzie musiał być zaimplementowany na zewnątrz, a nasz moduł będzie musiał porozumiewać się za pomocą wejść/wyjść binarnych, więc też raczej będzie miał ograniczone możliwości. I znowuż – z powodzeniem da się za jego pomocą przesłać jakieś liczby, pojedyncze wartości, ale raczej nie złożone zmienne, pobierane z różnych obszarów pamięci. Tu trzeba by było sięgnąć do aplikacji komunikujących się z LOGO! za pomocą Ethernetu.

Projektując własny interfejs sprzętowy trzeba też zwrócić uwagę na to, z którą wersją LOGO! będzie współpracował. Jeśli mamy do dyspozycji LOGO! 24RCE z wyjściami przekaźnikowymi, to trzeba będzie „uważać" na drgania styków, a za pomocą przekaźników też nie uda się uzyskać dużej prędkości przekazywania danych. Mają one jednak tę zaletę, że można je zasilać niemal dowolnym napięciem i w ten sposób uniknąć konieczności konwersji poziomów napięcia i zapewnić izolację galwaniczną. Jeśli zastosujemy LOGO! 24CE z wyjściami tranzystorowymi, to będzie można uzyskać większą prędkość transmisji, ale trzeba będzie zadbać o dopasowanie poziomów logicznych, ponieważ na wyjściach tej wersji LOGO! występuje napięcie 24 V DC. Podobnie wejścia binarne – napięcie niższe od 18 V DC jest uznawane za zero logiczne. Masa będzie musiała być wspólna dla naszego interfejsu i modułów LOGO!, więc nie można też mówić o separacji galwanicznej obwodów. Warto też pamiętać, że pewne wartości można zakodować lub przekazać analogowo, za pomocą stopniowanego napięcia lub prądu.

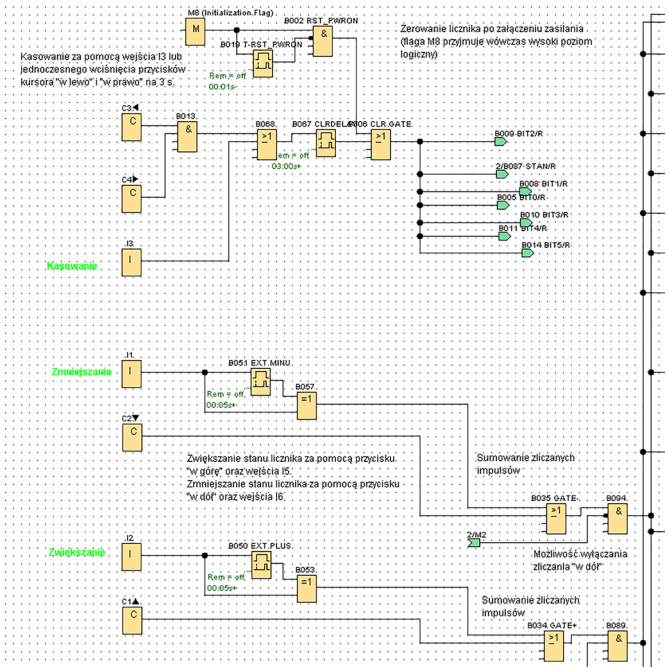
## Przykład – interfejs szeregowy dla LOGO!

Przykładowy, opisywany interfejs szeregowy trzeba podzielić na dwie części. Pierwsza z nich, sprzętowa, musi zapewniać współpracę modułu sterownika z wbudowanymi układami elektronicznymi. Jak łatwo się domyślić, jeśli mowa o zamianie danych binarnych na protokół transmisji szeregowej, to trzeba będzie użyć mikrokontrolera lub układu programowalnego, a te zwykle są zasilane napięciem z zakresu 2,5…5 V. Nasz układ konwersji poziomów logicznych będzie musiał zapewnić translację 24 V DC na np. 5 V DC lub mniej, a jeśli będziemy potrzebowali podawać coś na wejścia LOGO!, to również w drugą stronę. Rozwiązania przesuwników poziomu napięcia są dobrze znane i nie są trudne do wykonania.

Druga część, to oprogramowanie. Dane w LOGO! są dostępne w postaci, którą trzeba będzie zamienić na nadające się do transmisji poza LOGO! dane binarne. Z racji tego, że nie ma możliwości robienia wstawek programowych, jak w innych, „większych" sterownikach, to zadanie nie zawsze będzie wykonalne i trywialne. Niekiedy trzeba będzie troszkę pokombinować i sięgnąć do różnych źródeł wiedzy.

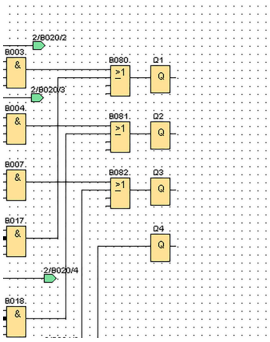
Schemat ideowy przykładowego rozwiązania interfejsu szeregowego RS232 pokazano na **rysunku 1**. Moduł ma więcej wejść, niż będziemy potrzebowali w tym przykładzie. Dane binarne są odbierane i zamieniane na postać szeregową przez mikrokontroler U1 (ATtiny2313). Odbiera on też echo transmisji szeregowej, potwierdza poprawność transmisji i może retransmitować dane w razie potrzeby. Do współpracy z płytką przeznaczyłem sterownik LOGO! 24CE zasilany napięciem stałym 24 V i dlatego poziomy logiczne występujące na wejściach proponowanej płytki interfejsowej to napięcie zbliżone do masy dla „0" logicznego i prawie 24 V dla „1" logicznej. Upraszcza to budowę konwertera poziomów dla mikrokontrolera – wykonano je na tranzystorach Q1…Q7, które mają kolektory zasilane takim samym napięciem, jak napięcie zasilania mikrokontrolera U1 (+5 V), a do bazy mają dołączone dzielniki rezystorowe 22 kΩ/1 kΩ. Dzięki temu, gdy LOGO! podaje „1" logiczną, to na bazie tranzystora występuje napięcie ok. 1 V wprowadzając go w stan przewodzenia, co mikrokontroler odczytuje jako „0" logiczne. Nie jest jednak problemem, aby „odwrócić" poziom logiczny we własnej aplikacji, programie napisanym dla mikrokontrolera AVR. Moduł ma również jedno wyjście, za którego pomocą może np. sygnalizować awarię linii transmisyjnej lub dołączonego urządzenia – w tym wypadku wyświetlacza LED z interfejsem RS232. Wówczas LOGO! może wyświetlić odpowiedni komunikat sygnalizując konieczność interwencji serwisu. Wykonano je w oparciu o tranzystory Q8 i Q9. Samo rozwiązanie przesuwnika jest zbliżone do zastosowanego w układzie ULN2003A.

Mikrokontroler jest taktowany za pomocą rezonatora kwarcowego XT1 o częstotliwości 11,0592 MHz zapewniającego minimalny błąd




**Rysunek 2. Wejścia sterownika LOGO!**

zegara transmisji szeregowej. Płytkę wyposażono w driver interfejsu szeregowego U2 (MAX232 lub odpowiednik) zapewniający obsługę transmisji RS232. Oryginalnie, płytka jest przeznaczona do montażu w obudowie na szynę TH35. W moim rozwiązaniu złącze interfejsu RS232 (DB9) jest dostępne z boku obudowy, w której wyfrezowano odpowiedni otwór.




**Rysunek 3. Wyjścia sterownika LOGO!**

Płytkę wyposażono w 3 diody LED służące do sygnalizowania statusu: D3 sygnalizuje załączenie zasilania, natomiast D1 i D2 mogą być dowolnie użyte przez aplikację sterującą np. do sygnalizacji poprawności odebranego echa transmisji itp.

Płytka jest zasilana napięciem +24 V DC, tym samym, którym jest zasilane LOGO! i jego moduły rozszerzeń. Dzięki niewielkiemu poborowi prądu, a tym samym i małym, spodziewanym stratom mocy, do zasilania płytki interfejsu zastosowano stabilizator liniowy 7805ACD2T (U3) w obudowie TO252 do montażu powierzchniowego. Interfejs wykonałem na płytce 2-warstwowej, na której stabilizator zamontowano na warstwie dolnej, która jednocześnie stanowi radiator dla stabilizatora.

W artykule opisano przykładowe rozwiązanie, pewien pomysł, więc nie udostępniono dokumentacji płytek drukowanych. Jest ono dopasowane do konkretnej obudowy i konkretnego zastosowania. W materiałach dodatkowych jest dostępny przykładowy program, więc nic nie stoi na przeszkodzie, aby samodzielnie wykonać podobną płytkę interfejsową dla dostępnej obudowy, wykorzystując artykuł, jako opis projektu referencyjnego.

## Zasada działania

Zadaniem płytki interfejsowej było przesyłanie na odległość stanu licznika. Jak wspomniano, zrobiono to za pomocą wyjść binarnych, co wymagało wykonania licznika z elementów dyskretnych, a nie wykorzystanie gotowego modułu programowego dostępnego w LOGO! Soft Comfort. Szczerze powiedziawszy, część sprzętowa interfejsu jest łatwa do wykonania – przysłowiowe „schody" zaczynają się przy wykonaniu oprogramowania.

W pierwszym kroku musimy zdecydować się na jakiś protokół komunikacyjny służący do wymiany danych pomiędzy CPU a naszym modułem. Dla potrzeb tego przykładu wykonano licznik 6-bitowy oraz zdecydowano się na użycie LOGO! v6. Osoba chcąca zaadoptować program dla LOGO! v7 lub v8 nie będzie z tym miała najmniejszego problemu – wystarczy, że załaduje program do środowiska LOGO! Soft Comfort i po naciśnięciu „Ctrl+H" (Select Hardware) wskaże nowy typ sterownika. Program sprawdzono ze sterownikami mającymi wyjścia tranzystorowe i przekaźnikowe z tym, że w wypadku tych drugich konieczne było wydłużenie czasów tworzonych przez timery B086 i B090 o około 1 sekundę (przy zachowaniu warunku, że B086 jest dłuższy od B090), aby przekaźnik zdążył zewrzeć i rozewrzeć styki. W praktyce LOGO! z przekaźnikami nadaje się w tej aplikacji jedynie do testów, ponieważ częste załączanie/wyłączanie przekaźników może spowodować ich szybkie zużycie się – pamiętajmy, że mają one ograniczoną liczbę cykli załączenia. Dodatkowo, trzeba pamiętać o zasilaniu jednego z zestyków przekaźników, aby załączał on napięcie +24 V DC.

4 wyjścia LOGO! mogą przekazać stan 6-bitowego licznika tylko w porcjach. Łatwo domyślić się, że będzie to 2× po 3 bity. Pozostaje tylko kwestia sposobu wyprowadzenia stanu licznika, podzielenia tej wartości na dwie porcje oraz zasygnalizowania tego faktu interfejsowi zewnętrznemu.

Osoby zajmujące mikrokontrolerami na pewno pamiętają procesory z rdzeniem 8051. Miały one wyprowadzone wyjścia 8 bitów adresu, a mogły adresować komórki pamięci zewnętrznej leżące w przestrzeni 16-bitowej. Adres był dzielony na dwie połówki, a o tym, z którą połówką mamy do czynienia informował sygnał ALE (Address Latch Enable). Sterował on też zewnętrznym, 8-bitowym rejestrem typu „zatrzask", zapamiętującym młodszy bajt adresu. Na podobne rozwiązanie trzeba zdecydować się też tutaj – w roli sygnału rozróżniającego połówki liczby użyto wyjścia Q4. Nasz protokół komunikacyjny, który będzie używany przez moduł zewnętrzny będzie wyglądał następująco:

- Jeśli wyjście Q4 będzie miało poziom logiczny niski, to na wyjściach Q1…Q3 będą dostępne 3 starsze bity wartości licznika.
- Jeśli wyjście Q4 będzie miało poziom logiczny wysoki, to na wyjściach Q1…Q3 będą dostępne 3 młodsze bity wartości.

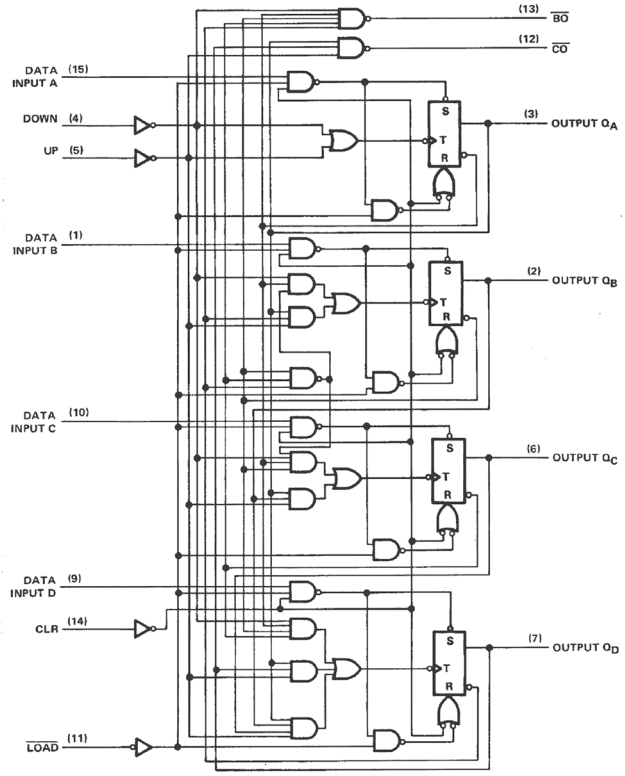
Mając tak sformułowane założenia protokołu komunikacyjnego oraz opracowaną część sprzętową modułu interfejsu, możemy przystąpić do tworzenia oprogramowania. Jak wspomniano, składa się ono z dwóch części – wykonanej dla LOGO! oraz dla mikrokontrolera ATtiny2313. Zacznijmy od łatwiejszej – od oprogramowania dla mikrokontrolera.

## Oprogramowanie modułu interfejsu

Pętlę główną programu dla mikrokontrolera zamieszczono na **listingu 1**. Zakładam, że wyprowadzenia:

- Q1 sterownika LOGO! dołączono do DTA0 interfejsu (PB0),
- Q2 do DTA1 (PB1),
- Q3 do DTA2 (PB2),
- Q4 do DTA3 (PB3),
- DTA4…DTA6 pozostawiono niepodłączone lub dołączono je do masy.

Po ustawieniu trybu pracy wyprowadzeń, zainicjowaniu UART i wyłączeniu diod LED, a następnie zaświeceniu LED1, program oczekuje na zmianę poziomu na wyprowadzeniu PB3 zapamiętując stan wejść mikrokontrolera w zmiennej *STEROWANIE* (port B jest „narzucony", numer bitu określa definicja *IMPULS_ZEGAROWY*). Po wystąpieniu poziomu wysokiego na PB3, program gasi LED1, neguje zawartość zmiennej *STEROWANIE* i umieszcza ją w zmiennej




**Rysunek 4. Schemat wewnętrzny licznika SN74193 firmy Texas Instruments zaczerpnięty z dokumentacji układu**

*LICZBA*. Teraz maskowane są bity od 3 do 7 i program oczekuje na wyzerowanie wejścia PB3, które to jest sygnałem, że na wejściach jest dostępna młodsza połówka bajtu. Po jej odebraniu, starsza połówka jest przesuwana w prawo o 3 miejsca i sumowana z młodszą. Tak uformowana liczba jest wysyłana do wyświetlacza LED, a powrót do trybu czuwania jest sygnalizowany zaświeceniem LED1.

Przykładowy program napisano w sposób mający ułatwić zrozumienie działania. Nie zastosowano w nim przerwań, mikrokontroler nie oszczędza energii i nie jest wprowadzany w tryb uśpienia, z którego mogłaby go wybudzić zmiana poziomu lub zbocze na PB3. Pozostawiam to inwencji czytelników, jeśli oczywiście takie rozwiązanie będzie miało sens i nie będzie jedynie kwestią mody na energooszczędność, ponieważ pobór prądu przez interfejs jest naprawdę znikomy.

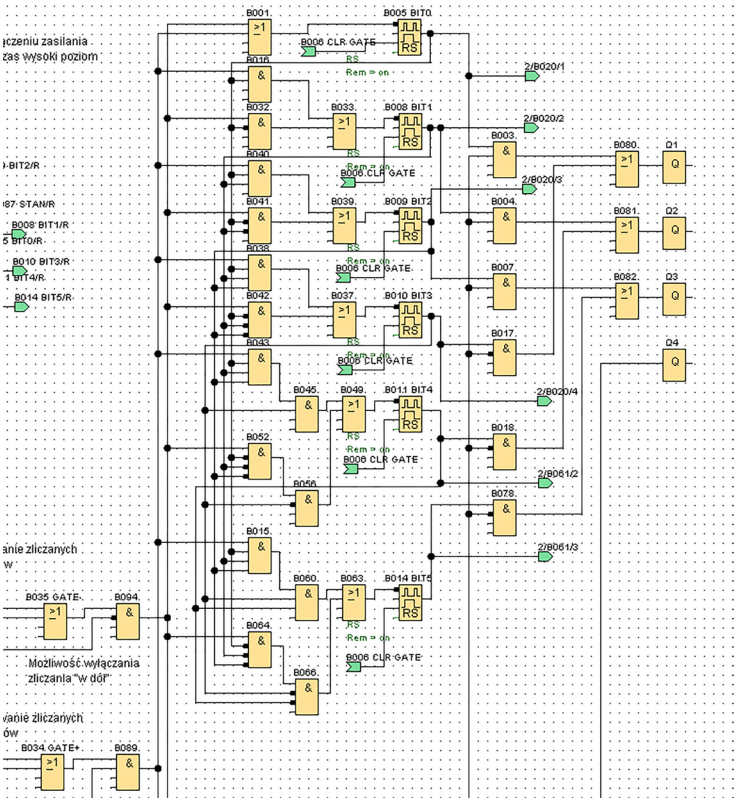
**Listing 1. Główna pętla programu dla mikrokontrolera ATtiny2313**

```
/*****************************************************************
  PROGRAM GŁÓWNY (OBSŁUGA SYGNAŁÓW POBUDZAJĄCYCH)
 *****************************************************************/
#define IMPULS_ZEGAROWY  0x08  //impuls zegarowy na PB3


int main(void)
{
  ports_init();  //inicjowanie portów I/O
  WYJSCIE_WYL;
  LED_WYL;
  USART_Init();  //inicjowanie USART
  LED1_ZAL;  //załączenie diody LED

/***************************
       PĘTLA GŁÓWNA
 ***************************/
  for (;;)
  {
    STEROWANIE = PINB;  //odczyt portu B - sygnały sterujące
    STEROWANIE = ~STEROWANIE;  //negowanie zmiennej, żeby aktywne były „1”
    if (STEROWANIE & IMPULS_ZEGAROWY)
    {
#ifdef PRZEKAZNIK
      _delay_ms(50);
#endif
      LED1_WYL;  //zgaszenie diody LED
      LICZBA = ~STEROWANIE;  //w zmiennej STEROWANIE już jest PINB
      //bity danych są ustalone po ustawieniu linii IMPULS_ZEGAROWY
      LICZBA &= 0x07;  //maskowanie bitów od 3 do 7
      do
      {
        STEROWANIE = PINB;
        STEROWANIE = ~STEROWANIE;
      } while (STEROWANIE & IMPULS_ZEGAROWY);

      //dane przesyłane poracjami po 3 bity, stąd przesuniecie
      //o 3 miejsca w lewo
      STEROWANIE = STEROWANIE << 3;
      //maskowanie bitów b0...b3, b6...b7
      STEROWANIE = STEROWANIE & 0b00111000;
      //obliczenie liczby wysłanej przez sterownik PLC
      LICZBA = LICZBA + STEROWANIE;
      if (LICZBA > 0) LED_Wyslij(LICZBA);
      _delay_ms(300);
      LED1_ZAL;
    }
  }
}
```



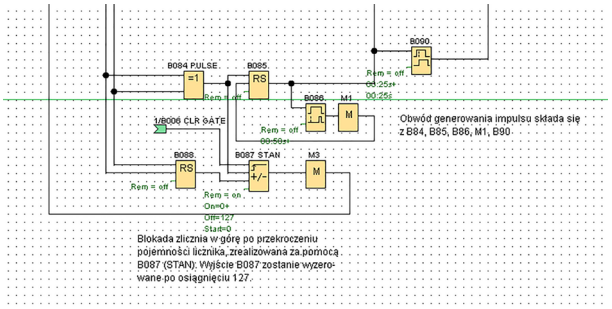

**Rysunek 5. Dwukierunkowy, 6-bitowy licznik wykonany w LOGO! Soft Comfort**

## Oprogramowanie LOGO!

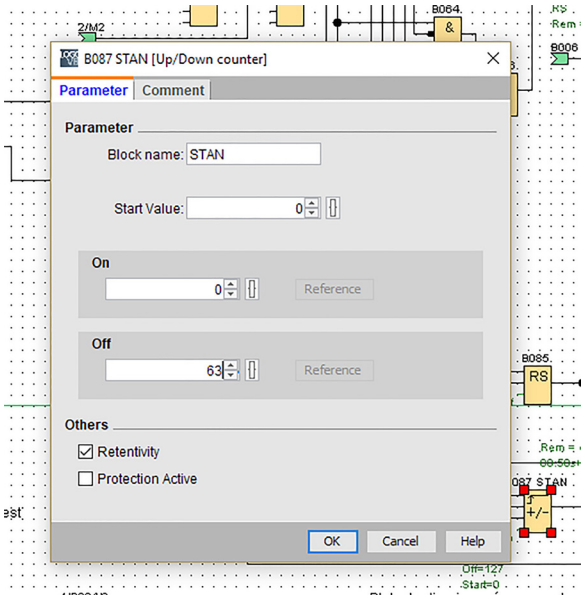
W materiałach dodatkowych można znaleźć cały plik programu z zaimplementowanym licznikiem. Najlepiej załadować go do środowiska LOGO! Soft Comfort, nacisnąć F3 i prześledzić działanie.

Założyłem, że licznik będzie liczył w górę i w dół, od 0 do 63, nie dopuszczając liczb ujemnych. Poszczególnym wejściom/wyjściom LOGO! nadano następujące funkcje:

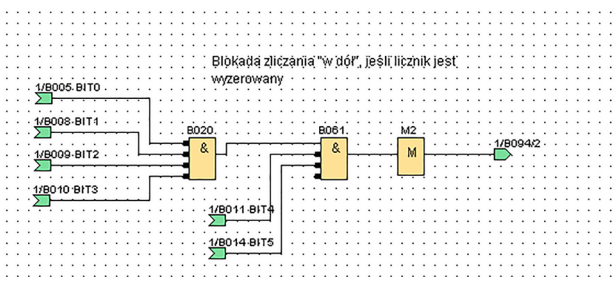
- I1 – zmniejszanie stany licznika (połączony z kursorem „w dół”).




**Rysunek 6. Blok licznika z generatorem impulsu strobującego**




**Rysunek 7. Okno właściwości bloku Up/Down Counter**




**Rysunek 8. Zabezpieczenie przed zliczaniem poniżej zera**

- I2 – zwiększanie stanu licznika (połączony z kursorem „w górę”).
- I3 – zerowanie stanu licznika po naciśnięciu i przytrzymaniu przez 3 sekundy.
- Q1…Q3 – wyjścia danych licznika.

Funkcje wyprowadzeń pokazano na **rysunkach 2** i **3**. Wejścia wyposażono w nieskomplikowane filtry eliminujące krótkie impulsy. I na tym w zasadzie kończy się najłatwiejsza część programu dla LOGO!

Środowisko LOGO! Soft Comfort ma bardzo funkcjonalne moduły liczników **Up/Down Counter**, z których można pobierać dane przez referencje. Dostępna jest w ten sposób cała zawartość licznika, a my dla potrzeb interfejsu musimy wydostać ją bit po bicie. Dlatego potrzebujemy wykonać licznik binarny z bramek i zaimplementować go w pamięci LOGO! Tu właśnie musimy cofnąć się do wiadomości z podstaw techniki cyfrowej.

W dokumentacji firmy Texas Instruments można znaleźć schemat wewnętrzny układu scalonego, dwukierunkowego licznika binarnego SN74193 (**rysunek 4**). Ma on nie tylko możliwość liczenia w górę i w dół, ale również zadania wartości, od której nastąpi zliczanie. My nie potrzebujemy tej funkcjonalności, więc wykonując licznik pozbywamy się wejść *LOAD*, *DATA INPUT A…D*. Przerzutniki typu „T” możemy zastąpić blokami **Pulse Relay** dostępnymi w LOGO! Soft Comfort. Powielając schemat połączeń bramek dodajemy dwa bity rozszerzając pojemność licznika z 4 do 6 bitów. Jeśli zabraknie nam wejść funktorów logicznych, to łatwo zwiększyć ich liczbę łącząc 2 lub więcej bramek. Dwukierunkowy, 6-bitowy, pozbawiony wejść równoległych licznik, wykonany na potrzeby tego artykułu pokazano na **rysunku 5**. Licznik ma oddzielne wejście „+” (Up) i oddzielne „–” (Down). Na jego wejściach włączono bramki B84 i B89, za których pomocą można zablokować zliczane impulsy zewnętrzne.

Równolegle do wejść liczących dołączono przerzutnik RS (B088) sterujący wejściem kierunku zliczania bloku **Up/Down Counter** (B087). Ten blok jest rozwiązany inaczej, niż SN74193, na którym się wzorowaliśmy. Ma jedno wejście kierunku zliczania oraz wejście zliczanych impulsów. Dlatego na jego wejście kierunku włączono przerzutnik RS, który jest ustawiany lub zerowany zależnie od tego, na którym wejściu (ustawiającym czy zerującym) wystąpi zliczany impuls. Pomimo obaw o czasy propagacji, rozwiązanie świetnie spełnia swoje zadanie. Impulsy z wejść *I1* oraz *I2* są sumowane za pomocą bramki *OR* (B084) i doprowadzone do wejścia liczącego (**rysunek 6**).

Po co nam ten blok licznika, skoro wykonaliśmy 6-bitowy licznik z funktorów logicznych? Posłuży on do zapewnienia dwóch funkcjonalności. Po pierwsze, ustali górny zakres zliczanych liczb. We właściwościach bloku licznika można zdefiniować wartości „On” i „Off”. To nic innego jak liczby, po których osiągnięciu wyjście zostanie ustawione (On) lub wyzerowane (Off). Użyłem tylko tej drugiej funkcjonalności deklarując, że wyjście ma być wyzerowane po osiągnięciu przez licznik 63 (**rysunek 7**) i łącząc je z bramką blokującą kierunek zliczania w górę. Po drugie, moduł licznika pozwoli nam na łatwe pokazanie liczby zliczonych impulsów na wyświetlaczu LCD lub panelu LOGO! TD. Niestety, wykorzystując poziomy dostępne na wyjściach Q1…Q4 moglibyśmy co najwyżej pokazać dane binarne, a moduł licznika pracując równolegle z wykonanym przez nas licznikiem dyskretnym zapewni „konwersję” na liczbę łatwo czytelną dla użytkownika.

Jak pamiętamy, musimy jeszcze zabezpieczyć licznik przed zliczaniem wartości poniżej 0. Tę funkcjonalność uzyskano za pomocą bramek B020 i B061 (**rysunek 8**). Połączone funktory AND tworzą bramkę, na której wyjściu jest dostępna „1” logiczna, jeśli wszystkie wejścia są wyzerowane. Te wejścia dołączono do wyjść poszczególnych bitów licznika, a jej wyjście poprzez flagę M3 jest doprowadzone do zanegowanego wejścia bramki AND (B094) włączonej na wejściu „–”.

Wróćmy jeszcze do rysunku 6, ponieważ oprócz połączeń samego modułu licznika zawiera on także schemat generatora impulsu strobującego. Każdy impuls docierający na wejście zliczające modułu licznika powoduje ustawienie przerzutnika **RS** (B085). Jego wyjście jest połączone z wejściami bramek B003, B004, B007, B017, B018 i B078 multipleksujących dane z liczników oraz z wejściem timera **On Delay** (B086). Jeśli wyjście przerzutnika jest ustawione, to na wyjściach bramek są dostępne 3 młodsze bity, a jeśli wyzerowane, to 3 starsze.

Przerzutnik **RS** jest zerowany przez timer **On Delay** (B086) opóźniający impuls wyjściowy o 50 ms. Ten timer wprowadza krótką zwłokę tworzącą „szerokość” impulsu strobującego. Timer **On/Off Delay** (B090) wprowadza krótkie opóźnienie (przesunięcie) zboczy impulsu na wyjściu Q4, aby impuls strobujący występował po zwłoce na ustalenie się poziomów na wyjściach sterownika.

## Podsumowanie

Sterownik LOGO! v8 umożliwia nam wykonywanie podobnych programów z aż 400 bloków funkcjonalnych! To naprawdę ogromna liczba funktorów, których można użyć na wiele, wiele sposobów. Autor artykułu wykonał np. urządzenie liczące zawierające 2 liczniki 9-bitowe, do którego jest dostęp poprzez Internet. Nieskomplikowany interfejs użytkownika umożliwia łatwe modyfikowanie wartości i wprowadzanie potrzebnych nastaw. Samodzielnie wykonanie takiego urządzenia od podstaw wymaga mnóstwo czasu, podczas gdy LOGO! oferuje nam rozwiązanie gotowe do użycia. Jeśli urządzenie nie ma być produkowane na skalę masową, a ma zapewnić rozwiązanie jednostkowe, to warto rozważyć zastosowanie mocarnego LOGO!

**Jacek Bogusz, EP**