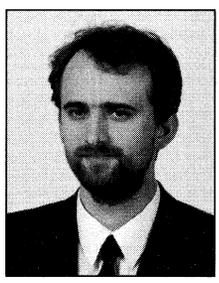
Wojciech SZYDEŁKO

POLITECHNIKA RZESZOWSKA
KATEDRA AUTOMATYKI I INFORMATYKI

Programowanie węzłów sieci LON

Mgr inż. Wojciech SZYDEŁKO
– asystent w Katedrze Automatyki i Informatyki Politechniki Rzeszowskiej im. Ignacego Łukasiewicza. Interesuje się sieciami komputerowymi, obliczeniami równoległymi i rozproszonymi oraz sieciami automatyki.



Streszczenie

W artykule przedstawiono zasady programowania węzłów sieci LON. Szczegółowo omówiono konstrukcję i działanie programu napisanego w języku Neuron C, zwłaszcza strukturę warunku *when*, obiektów we/wy oraz sposób deklarowania i użycia zmiennych sieciowych. Wszystkie zagadnienia zostały przedstawione w przykładzie kompletnego programu.

Abstract

This article presents rules of programming of LON network nodes. It describes in details construction and working of programs written in Neuron C language, particularly a structure of *when* statement. It also describes I/O objects and ways of declaring and use of network variables. Each problem is described in example of complete program.

Wstęp

W ostatnich latach pojawiły się sieci mikroprocesorowe, które w znaczny sposób uprościły tworzenie, zarządzanie i rozbudowę układów automatyki. Spośród wielu różnych rozwiązań bardzo dużym zainteresowaniem w świecie cieszy się sieć o nazwie LON (ang. Local Operating Network - Lokalna Sieć Operacyjna), stworzona przez amerykańską firmę Echelon. Odznacza się ona elastycznością, bezpieczeństwem, niezawodnością, interoperacyjnością, oraz prostotą programowania. Znajduje zastosowanie w automatyce przemysłowej oraz przede wszystkim w tzw. automatyce budynków.

Podstawowym elementem sieci LON jest węzeł, który zawiera specjalny procesor o nazwie Neuron składający się z trzech 8-bitowych procesorów komunikujących się przez wspólną pamięć, portu komunikacyjnego do podłączania transceivera, 11 pionowego portu we/wy do podłączania czujników i urządzeń wykonawczych i wbudowanej pamięci operacyjnej. Procesor Neuron może również zawierać port do podłączenia pamięci zewnętrznej. Ma ponadto wbudowany sprawdzony system operacyjny, specjalny protokół komunikacyjny LonTalk oraz biblioteki obsługi różnorakich urządzeń we/wy. Zastosowanie trzech procesorów w układzie Neuron pozwoliło na rozdzielenie zadań zgodnie z hierarchią 7-warstwowego modelu OSI/ISO. Pierwszy procesor odpowiada za warstwy 1 oraz 2 i nosi nazwę procesora MAC (ang. Media Access Control - sterowanie dostępem do mediów). Obsługuje on interfejs dostępu do medium transmisyjnego układu Neuron, załatwia tworzenie ramek telegramów i zabezpieczenie dostępu do medium. Procesor drugi nazywa się procesorem sieciowym. Jest on odpowiedzialny za warstwy 3-6. Do jego zadań należy generowanie i analiza informacji adresowych, procedury warstwy transportowej (potwierdzenia, kontrola, zarządzanie transakcjami i autentyzacja telegramów). Poza tym procesor ten wykonuje obszerne funkcje zarządzania siecią. Ostatni procesor przeznaczony jest na wykonywanie programu aplikacyjnego stworzonego przez użytkownika, łącznie z wywoływanymi przez ten program procedurami systemu operacyjnego.

Pisanie programu aplikacyjnego odbywa się przy pomocy języka Neuron C, który stanowi dialekt języka C zbudowany na bazie języka ANSI C, uzupełniony o konstrukcje specyficzne dla sieci LON.

Działanie węzła jest zdefiniowane przez program (aplikację) i informacje konfiguracyjne zawarte w pamięci węzła. Fragment pamięci węzła definiowany przez użytkownika składa się z dwóch części: obrazu sieci i obrazu aplikacji.

Obraz sieci definiuje związki węzła z innymi węzłami i podaje unikatową pozycję węzła w sieci. Jest tworzony przy użyciu komend pochodzących ze specjalnego typu węzła zwanego narzędziem nadzorowania sieci.

Obraz aplikacji zawiera program węzła, który określa zdarzenia, na które węzeł odpowiada oraz działania, jakie podejmuje w odpowiedzi na pojawienie się zdarzenia.

Niniejsze opracowanie omawia sposób pisania programów węzła w języku Neuron C na przykładzie programu do sterowania ryglowaniem drzwi w budynku. Dokładnie omówiono konstrukcję i działanie programu, zwłaszcza użycie warunków *when*, obiektów we/wy oraz zmiennych sieciowych. Szczegóły rozwiązania poprzedzono niezbędnymi informacjami o charakterze ogólniejszym, by ułatwić Czytelnikowi przystąpienie do tworzenia własnych programów.

Zasady konstrukcji programu

Język Neuron C pozwala na wygodne przygotowanie programów użytkowych dla węzła sieci LON.

Program zarządzający obsługą zdarzeń

Programy napisane w Neuron C są zorientowane na obsługę zdarzeń. Aby to umożliwić, został wprowadzony specjalny warunek o nazwie ***when***. Natomiast do organizacji obsługi zdarzeń został stworzony specjalny program zarządzający - scheduler, który stanowi podstawową procedurę systemu operacyjnego wykonywaną na procesorze 3 (aplikacyjnym). Zadaniem tego programu jest cykliczne sprawdzanie warunków *when* i wywołanie zadań związanych z obsługą odpowiedniego zdarzenia. Zastosowano dwa poziomy priorytetów. Algorytm pracy schedulera jest następujący.

Po zresetowaniu procesora scheduler przebiega punkt wyjściowy. Tu dokonuje aktualizacji tzw. zmiennych sieciowych omówionych w dalszej części pracy i sprawdza stany liczników. Po tym jest gotowy do sprawdzania warunków ***when***. Rozpoczyna od warunków z wyższym priorytetem według kolejności ich występowania w tekście źródłowym. W razie spełnienia warunku, wykonywane jest odpowiednie zadanie i scheduler powraca do punktu wyjścia.

Gdy żaden z wariantów z wyższym priorytetem nie jest spełniony, scheduler przechodzi do algorytmu kołowego (Round_Robin) biorąc pod uwagę pierwszy z warunków ***when*** spośród warunków o niższym priorytecie, sprawdza go i w razie spełnienia wykonuje odpowiednie zadanie. W każdym przypadku powraca znowu do punktu wyjścia. Gdy i teraz nie wystąpi potrzeba obsługi zadania o wyższym priorytecie, przechodzi ponownie do algorytmu kołowego i sprawdza kolejny z warunków o niższym priorytecie.

Ponieważ cały program jest sterowany zdarzeniami, jego tekst składa się z występujących po sobie warunków ***when***.

Składnia wyrażenia „*when*"

W programie przykładowym (punkt) mamy szereg warunków ***when***. Składnia tego wyrażenia ma następującą postać.

```
[priority] when ( zdarzenie )
      .....
[priority] when ( zdarzenie )
{
  //Zadanie
}
```

Słowo **priority** stosujemy, gdy chcemy temu warunkowi przypisać wyższy priorytet, wyrażenie ***when*** jest obowiązkowe. W nawiasach okrągłych umieszcza się zdarzenie na jakie będzie reagował warunek. Mogą to być np. zmiany sygnału na pinach I/O, upływ czasu odmierzanego zegarem, pojawienie się zawartości zmiennej sieciowej. Tego typu zdarzenia nazywamy zdarzeniami predefiniowanymi. Mogą być również zastosowane zdarzenia zdefiniowane przez użytkownika, np. każde dopuszczalne wyrażenia języka Neuron-C. Zdarzenia predefiniowane zastosowane w programie przykładowym zostały opisane niżej. W nawiasach klamrowych wpisujemy obsługę warunku ***when***. Mogą tu być dowolne wyrażenia języka Neuron-C oprócz wyrażenia ***when***.

Zdarzenia predefiniowane

Zawsze stosowanym zdarzeniem predefiniowanym jest zdarzenie ***reset***. Występuje gdy procesor zostanie zresetowany i przy pierwszym włączeniu procesora. Następne zdarzenie to ***io_changes()***. Występuje gdy nastąpi zmiana wartości obiektu I/O (zmiana na porcie wejściowym układu Neuron). W nawiasach okrągłych podaje się nazwę obiektu we/wy. Trzecie zdarzenie predefiniowane to ***nv_update_occurs()*** informuje, że wejściowa zmienna sieciowa węzła została zmodyfikowana. W nawiasach okrągłych występuje nazwa zmiennej sieciowej, którą obserwujemy.

Komunikacja z urządzeniami wejścia/wyjścia (obiekty I/O)

Zewnętrzne urządzenia i układy elektroniczne podłącza się do układu Neuron Chip za pośrednictwem jedenastu pinów układu o nazwach **IO_0** do **IO_10**. Piny te mogą być konfigurowane na różne sposoby, dzięki czemu w łatwy sposób możemy stworzyć skomplikowane funkcje wejścia i wyjścia przy minimalnym zaangażowaniu dodatkowego sprzętu. Piny konfigurujemy programowo przy pomocy języka Neuron C. Umożliwia on programiście w prosty sposób zadeklarowanie jednego lub więcej pinów jako obiektu wejściowego lub wyjściowego. Dzięki wprowadzeniu obiektów wejścia/wyjścia programista odwołuje się do odpowiedniego typu obiektów, zamiast odwoływać się bezpośrednio do pinów i formatować dane. Program podczas wykonywania odwołuje się za pośrednictwem wywołań funkcji **io_in()** i **io_out()** do zdefiniowanych wcześniej obiektów wejściowych i wyjściowych.

W systemie układu Neuron Chip znajdują się 34 różne obiekty wejścia i wyjścia. Większość jest dostępna z wewnętrznej pamięci ROM układu. W razie potrzeby użytkownik może przygotować dodatkowe obiekty i zapisać je w pamięci EEPROM. Wszystkie obiekty możemy podzielić na cztery podstawowe grupy:

- **bezpośrednie** - bazują na poziomie logicznym pinów I/O. Obiekty te nie używają żadnych timerów/liczników sprzętowych układu Neuron i mogą być używane wielokrotnie.
- **timer/counter** - używają układu timer/counter układu Neuron Chip. Układ Neuron posiada 2 liczniki, przy czym na wejściu jednego może być sygnał multipleksowany pochodzący z pinów IO4 - IO7, a na wejściu drugiego z pinu IO0.
- **szeregowe** - używane do transmisji szeregowych danych poprzez pojedynczy pin lub zbiory pinów. Ze względu na ograniczenia sprzętowe tylko jeden obiekt tego typu, (wejściowy lub wyjściowy) może być zadeklarowany.
- **równoległe** - używane do dwukierunkowej transmisji o dużej prędkości. Obiekty te wykorzystują wszystkie piny procesora.

Składnia deklaracji obiektów I/O

Składnia obiektów I/O zależy od ich typu. Jednak w każdej deklaracji występują nazwy pinów (słowa ***IO_0, IO_1*** itd. aż do ***IO_10***) oraz w zależności od kierunku słowo ***input*** lub ***output***. W przykładowym programie stosujemy tylko obiekty typu bitowego, więc do nich się ograniczymy. Składnię tego obiektu przedstawiono dalej. Słowo kluczowe bit oznacza, że pin, którego nazwa znajduje się na początku będzie traktowany jak jednobitowy port binarny.

O_7 **input bit** nazwa_obiektu_io;
O_0 **output bit** nazwa_obiektu_io [= wartość_inicjująca].

Pierwsze wyrażenie określa ósmy pin jako wejście bitowe a drugie wyrażenie określa pin pierwszy jako binarny port wyjściowy. W drugim przypadku obiektowi przypisana jest opcjonalnie wartość początkowa.

Dostęp do obiektów I/O

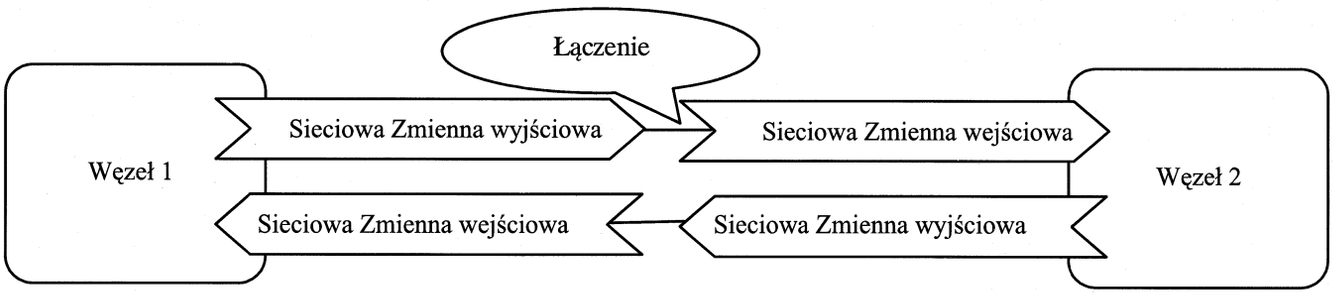
Do wysyłania na port danych (port wyjściowy) i do sczytywania stanu portu (port wejściowy) stosowane są dwie predefiniowane funkcje ***io_out()*** i ***io_in()***. Funkcja pierwsza nie zwraca żadnych wartości. Pierwszym jej parametrem jest nazwa obiektu, na którym będzie operować. Drugi parametr to wartość wyjściowa. Może tu być wstawiona nazwa zmiennej której wartość będzie wysyłana do portu. Druga funkcja (io_in()) zwraca wartość przeczytaną z portu. Jej parametrem wywołania jest nazwa obiektu, na którym będzie operować. W zależności od typu obiektu dodatkowo argumentami obu funkcji mogą być parametry tych obiektów. Ogólna składnia tych funkcji jest następująca.

wartość_zwracana = **io_in** (nazwa_obiektu_io
[,argumenty]);

io_out (nazwa_obiektu_io, wartość_wyjściowa
[,argumenty]);

Komunikacja między węzłami, zmienne sieciowe

Dla uproszczenia programowania wymiany danych między węzłami w sieci LON wprowadzono tzw. zmienne sieciowe. Z punktu widzenia sieci są to wejścia i wyjścia węzła, które umożliwiają współ-



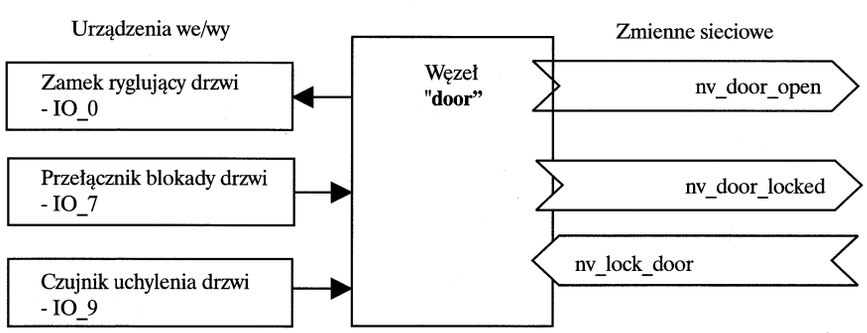
Rys 1. Wymiana danych przez zmienne sieciowe

dzielenie danych. Dzięki temu programista przygotowujący aplikację skupia się tylko na przygotowaniu kodu dla pojedynczego węzła. Przy projektowaniu wymiany danych musi pamiętać jedynie o zadeklarowaniu odpowiedniej zmiennej sieciowej, do której będzie się odwoływał jak do każdej innej zmiennej w programie. Do wysyłania danych z węzła musi zadeklarować zmienną wyjściową (**output**), natomiast do przyjmowania danych zmienną wejściową (**input**). Operując na zmiennych sieciowych programista nie musi interesować się mechanizmem sieciowym, który załatwia protokół LonTalk. Dzięki temu możemy projektować węzły niezależnie od siebie. Co więcej, możemy wykorzystywać ten sam program na wielu węzłach w sieci. Zmienne sieciowe są przypisane do węzłów, ich lista jest zapisywana w pamięci EEPROM układu Neuron.

Do prawidłowego funkcjonowania całej rozproszonej aplikacji sieciowej nie wystarczy zaprogramować węzły (wpisać im aplikacje) i włączyć do sieci fizycznej. Musimy jeszcze połączyć ze sobą odpowiednie zmienne wejściowe i wyjściowe. Służy do tego specjalne narzędzie o nazwie „binder" wchodzący w skład narzędzi zarządzających LONBUILDER, rys. 1. Informacje o połączeniu tych zmiennych zostają zapisane w węźle w pamięci EEPROM i od tej pory nie potrzebujemy już programu zarządzającego. Sieć jest skonfigurowana. Gdy węzeł wysyłający dane będzie chciał przesłać je do węzła czytającego, to po prostu modyfikuje swoją wyjściową zmienną sieciową a mechanizmy protokołu LonTalk powodują przesłanie tej wartości do wszystkich związanych z nią zmiennych wejściowych na innych węzłach, rys. 2).

Zmienne wyjściowe mogą być połączone z wieloma zmiennymi wejściowymi innych węzłów. Wówczas jednym urządzeniem wejściowym np. wyłącznikiem możemy sterować wieloma urządzeniami wyjściowymi np. lampami.

Dla ujednolicenia wymiany danych między węzłami różnych typów i różnych producentów został wprowadzony specjalny typ zmiennych sieciowych o nazwie „standardowe zmienne sieciowe" w skrócie SNVT (Standard Network Variable Type). Zmienne te zawierają jednostki, zakresy, krok zmienności oraz numer identyfikacyjny (ID) z zakresu od 1 do 255. Dostępne są zmienne do przekazywania temperatury, wilgotności itd. a nawet zmienne sterujące wybieraniem numerów telefonicznych. Producenci sprzętu, stosując do wymiany danych zmienne typu SNVT, mogą być pewni, że ich urządzenia będą poprawnie współpracować z produktami innych producentów. Aby zachować interoperacyjność między węzłami, zaleca się stosowanie zmiennych SNVT, kiedy to tylko jest możliwe.



Rys 2. Urządzenia we/wy i zmienne sieciowe węzła „door"

Deklarowanie zmiennych sieciowych

Deklaracja zmiennych sieciowych jest bardzo podobna do deklaracji innych zmiennych w Neuron-C. Uproszczona składnia deklaracji jest następująca.

network input | output typ identyfikator
[= wartość_inicjująca]

Słowo kluczowe ***network*** zawsze poprzedza słowo ***input*** lub ***output*** i oznacza zmienną sieciową. Słowo ***input*** jest stosowane do deklaracji zmiennej wejściowej (węzeł czyta wartość zmiennej), natomiast słowo ***output*** do deklaracji zmiennej wyjściowej (węzeł zapisuje do niej wartości). Słowo „typ " określa typ zmiennej. Może to być na przykład typ „int ", ale często w tym miejscu wpisuje się jakiś typ standardowej zmiennej sieciowej (SNVT). Następne słowo „identyfikator" to nazwa zmiennej sieciowej a „wartość_inicjująca" to wartość początkowa zmiennej.

Przykład: automatyczne zamykanie drzwi

Przedstawiony zostanie program, w treści którego na bieżąco będą wyjaśniane wszystkie jego elementy oraz sposób działania. W ramkach zawarty będzie opis kolejnych fragmentów programu. Znaczenie poszczególnych linii objaśniają komentarze.

Sformułowanie zadania

Program przeznaczony jest do automatycznego zamykania (ryglowania) wszystkich drzwi systemu przez zamknięcie tylko jednych. System może być zastosowany w samochodach oraz w budynkach. Zamknięcie (zaryglowanie) wydzielonych głównych drzwi powoduje zablokowanie pozostałych drzwi. Odblokowanie głównych drzwi powoduje otwarcie wszystkich, natomiast odblokowanie innych niż główne powoduje otwarcie jedynie tych drzwi.

Cały system składa się z jednego typu węzłów na którym jest realizowany program o nazwie „door.nc ". Do węzła podłączone są dwa czujniki binarne: czujnik uchylenia drzwi do

pinu IO_9 oraz czujnik zaryglowania (blokada drzwi) do pinu IO_7. Podłączony jest również sterowany binarnie rygiel (zamek) do pinu IO_0, rys. 2.

Do pinu IO_0 podłączony jest dwustabilny rygiel, który powoduje zablokowanie drzwi. Na pinie IO_7 zamontowany jest przełącznik, który steruje zamknięciem drzwi, natomiast na pinie IO_9 czujnik uchylenia drzwi sprawdzający, czy drzwi nie są uchylone.

W węźle zadeklarowane są trzy zmienne sieciowe, których funkcje podano w Tab.

Tab. Funkcje zmiennych sieciowych użytych w przykładzie

Zmienna	Kierunek	Funkcja
nv_door_open	wyjściowa	Informuje inne węzły że drzwi są uchylone przyjmując wartość ST_ON, gdy są zamknięte przyjmyje ST_OFF.
nv_door_locked	wyjściowa	Zmienna, za pośrednictwem której przekazywana jest informacja o zaryglowaniu lub odryglowaniu drzwi. Gdy drzwi mają być odryglowane, przyjmuje wartość ST_OFF, gdy mają być zaryglowane, przyjmuje wartość ST_ON.
nv_lock_door	wejściowa	Zmienna stosowana do automatycznego zamykania i otwierania drzwi na komendę z węzła sterującego. Aby spełniła swą funkcję, musi być połączona na etapie konfiguracji sieci ze zmienną *nv_door_locked* (zob. rys. 3)

Stan ST_ON i ST_OFF zostały zdefiniowane w pliku nagłówkowym „snvt_lev.h ", który należy dołączyć do programu przy pomocy wyrażenia „#include snvt_lev.h ". W pliku tym zdefiniowany jest typ standardowych zmiennych sieciowych (SNVT) jako typ wyliczeniowy. Stan ST_ON oznacza włączenie, a ST_OFF wyłączenie, stan 1 lub 0.

Konfigurowanie sieci

Aby była możliwa komunikacja pomiędzy węzłami, musimy przy pomocy narzędzi konfiguracyjnych (np. Lon Maker) odpowiednio połączyć zmienne sieciowe. Połączenie zmiennych sieciowych z rozważanego przykładu przedstawione jest na rys. 3.

Wykorzystano tu tylko po jednej zmiennej sieciowej dla węzła. W tym przypadku nie jest wykorzystywana tylko zmienna „nv_door_open ", która może informować inne węzły o uchyleniu drzwi, co jest równoznaczne z niemożliwością zamknięcia.

Sieć może również być skonfigurowana w inny sposób, ale wówczas zmieni się jej funkcjonowanie.

Kod źródłowy programu

```
#include <io_types.h>
#include <snvt_lev.h>
```

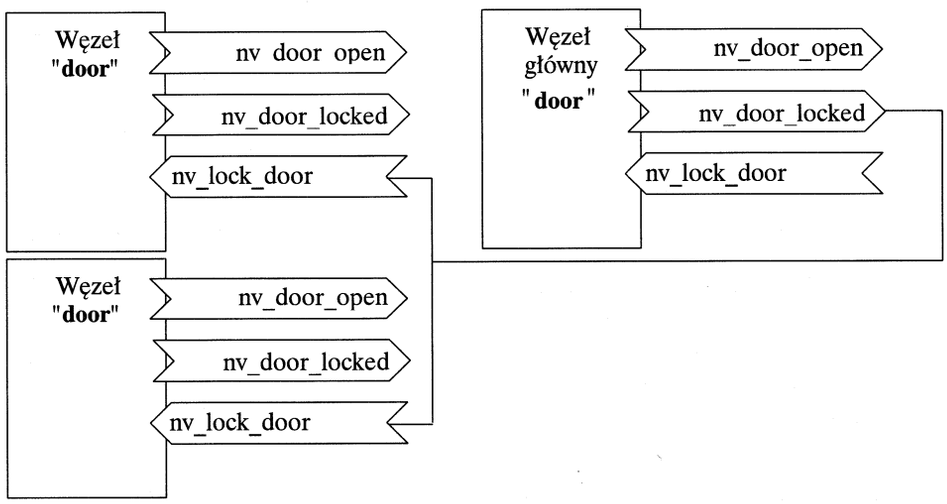
Pierwszy plik zawiera deklaracje różnych typów danych stosowanych dla obiektów we/wy. Są tam zdefiniowane takie typy danych jak: bit_t, byte_t i inne. Drugi plik zawiera deklaracje standardowych zmiennych sieciowych typu wyliczeniowego. Są tam zdefiniowane między innymi stałe ST_OFF i ST_ON.

```
#include <netdbg.h>   // plik nagłówkowy umożliwiający
                      // "debugowanie" programu
```

Plik nagłówkowy „netdbg.h" zawiera niezbędne dyrektywy kompilatora umożliwiające uruchomienie debagera. Gdy przetestujemy program, możemy tę linię wykasować.

```
// Deklaracja zmiennych sieciowych. Wszystkie
// zadeklarowane zmienne są standardowymi
// zmiennymi sieciowymi

network output SNVT_lev_disc nv_door_open;
                      // Przyjmuje ST_ON
                      // gdy drzwi są uchylone
network output SNVT_lev_disc nv_door_locked;
                      // Przyjmuje ST_ON
                      // gdy drzwi są zaryglowane
network input SNVT_lev_disc nv_lock_door = ST_OFF;
                      // Przyjmuje ST_ON
                      //po to aby zaryglować drzwi
```



Rys 3. Połączenie zmiennych sieciowych

// Deklaracja obiektów we/wy

Na wstępie zadeklarowane są dwie stałe pomocnicze typu bit_t. Ten typ zmiennych zdefiniowano w pliku io_types.h

```
const bit_t DOOR_LOCK_COMMAND = 0;
const bit_t DOOR_UNLOCK_COMMAND = 1;
```

Właściwa deklaracja we/wy. Jest to obiekt typu ***bit***, który przypisany jest do pinu IO_0. Zadaniem jego jest sterowanie ryglem elektrycznym zmykającym drzwi.

```
IO_0 output bit io_lock_control = 0;
                        // Używany do sterowania
                        // automatycznego ryglowania drzwi
```

Deklaracja obiektu wejściowego typu bit przywiązanego do pinu IO_7, do którego podłączony jest przełącznik zamykający drzwi (klucz).

```
IO_7 input bit io_lock_sensor;  //Podłączony jest
                                //przełącznik zamykający
                                //drzwi (klucz)
```

Deklaracja stałych pomocniczych

```
const bit_t DOOR_OPEN_STATE = 0;
const bit_t DOOR_CLOSED_STATE = 1;
```

Deklaracja obiektu wejściowego typu ***bit*** przywiązanego do pinu IO_9, do którego podłączony jest czujnik uchylenia drzwi.

```
IO_9 input bit io_open_sensor;  // Sprawdza czy drzwi
                                // są uchylone
```

// Prototyp funkcji

```
void set_lock(boolean lock_door);
```

Obsługa zdarzeń

// Zadanie obsługi zdarzenia "reset"

```
when (reset)
{
 // Sprawdzanie czy drzwi są uchylone
 nv_door_open = (io_in(io_open_sensor)
== DOOR_OPEN_STATE) ? ST_ON : ST_OFF;

 // Sprawdzanie czy drzwi są zaryglowane
 set_lock(io_in(io_lock_sensor)
== DOOR_LOCKED_STATE) ;
// Inicjacja zdarzenia io_change wartością aktualną na
// porcie. Dopiero od niej będzie obserwowana zmiana.
 io_change_init(io_open_sensor);
 io_change_init(io_lock_sensor);
}
```

// Obsługa zdarzenia związanego z uchyleniem drzwi

```
when (io_changes(io_open_sensor) )
{ // Przeczytany stan obiektu "io_open_sensor"
  // jest wysyłany przez zmienną sieciową
  // nv_door_open
  nv_door_open = (input_value ==
DOOR_OPEN_STATE) ? ST_ON : ST_OFF;
}
```

// Obsługa zdarzenia obsługi rygla

```
when (io_changes(io_lock_sensor))
{ // Sprawdza, czy został przekręcony "klucz" w węźle.
  // Obsługa tego zdarzenia polega na
  // wywołaniu funkcji "set_lock()", która powoduje
  // zamknięcie lub otwarcie zamka węzła,
  // jako parametr jest użyty warunek, którego wynikiem
  // jest PRAWDA lub FAŁSZ.

  set_lock(input_value == DOOR_LOCKED_STATE);
}
```

// Obsługa zdarzenia modyfikacji zmiennej
// sieciowej nv_lock_door

```
when (nv_update_occurs(nv_lock_door))
{ // Reaguje na zmianę wartości zmiennej sieciowej,
  // która podaje informacje o zamknięciu
  // lub otwarciu drzwi przychodzącą z innego węzła.
  // Samą operacją zamknięcia lub otwarcia drzwi
  // zajmuje się funkcja "set_lock()".

  set_lock(nv_lock_door == ST_ON);
}
```

//Funkcja przeznaczona do zamykania i otwierania
//drzwi.

Gdy parametr wywołania ma wartość PRAWDA, drzwi zostają zamknięte, gdy FAŁSZ to są otwarte.

```
void set_lock(boolean lock_door)
{
 if ( lock_door ) {
  io_out ( io_lock_control,DOOR_LOCK_COMMAND);
                    // Zamknięcie drzwi.
  nv_door_locked=ST_ON; // Wysłanie przez zmienną
                        // sieciową informacji o
                        // zamknięciu drzwi.
 } else {
  io_out (io_lock_control,DOOR_UNLOCK_COMMAND);
                        // Otwarcie drzwi.

  nv_door_locked = ST_OFF;// Wysłanie przez zmienną
                        // sieciową informacji
                        // o otwarciu drzwi.
 }
}
```

Podsumowanie

Programowanie węzłów sieci LON wymaga znajomości języka Neuron C. Jest on jednak bardzo podobny do języka ANCI C. Wprowadzone w Neuron C dodatkowe elementy, takie jak obiekty we/wy, zmienne sieciowe oraz przeznaczenie języka do obsługi zdarzeń powodują, że nawet mając mniejsze doświadczenie w programowaniu procesorów, można w łatwy sposób tworzyć skomplikowane aplikacje.

Nieniejsza praca wyjaśnia podstawowe zasady programowania bazujące na reprezentacyjnym przykładzie z zakresu tzw. automatyki budynków. Programowanie przetestowano w systemie Node-Builder [4]. Przedstawione rozwiązanie można łatwo zmodyfikować i rozbudować do innych zastosowań.

Interoperacyjność systemu LON pozwala na wspólną pracę urządzeń różnych producentów. Sieć automatyki LON mająca szerokie możliwości zastosowań w różnych dziedzinach spotykają się z coraz szerszym zainteresowaniem w Polsce. Pojawiają się układy wykorzystujące oprócz zachodnich, również własne rozwiązania węzłów [3]. Umiejętność konstruowania węzłów, zarówno pod względem sprzętowym i programowym, jest więc bardzo aktualna.

Literatura

[1] W. BOROŃ: Lokalna sieć sterowania LON. PAK 10/1993.
[2] T. GOSZCZYŃSKI: Sieci mikroprocesorowe od podstaw. PAR 4-5/1998.
[3] Technologia LonWorks a polskie urządzenia automatyki. PAK 11/1998.
[4] Echelon Corporation: LonWorks Engineering Bulletins, January 1995.
[5] Echelon Corporation: Neuron C Programmer's Guide, 1995.
[6] Echelon Corporation: Neuron C Reference Guide, 1995.
[7] Echelon Corporation: Node Builder User's Guide, 1995.
[8] Echelon Corporation: The SNVT Master List and Programmer's Guide.

Artykuł recenzowany