iControlSensor

Czujnik do pomiaru temperatury dla iControl

Na łamach Elektroniki Praktycznej (część 1 – EP 1/2015, część 2 – EP 2/2015) przedstawiłem projekt systemu typu „inteligentny dom”, który w roli medium transmisyjnego używa sieci energetycznej (technologia Power Line Communication). W artykule opisano podstawowe elementy systemu nazwanego iControl, tj. urządzenia sterującego iControlMaster oraz urządzenia wykonawczego iControlSwitch pełniącego rolę wyłącznika jedno- lub dwubiegunowego. Kolejnym rodzajem modułu, jaki obsługiwany jest przez nasz system „inteligentnego budynku” jest moduł iControlSensor przeznaczony do pomiaru temperatury.

Rekomendacje: *czujnik temperatury do systemu iControl. Może przydać się do pomiaru lub sterowania ogrzewaniem.*

Podstawowe informacje:

- Maksymalnie 64 modułów wykonawczych (typu Slave) w ramach jednej sieci systemu iControl.
- Maksymalnie 16 modułów sterujących (typu Master) wyposażonych w interfejs użytkownika z wyświetlaczem TFT.
- Adresy logiczne modułów wykonawczych nadawane są automatycznie przez moduły sterujące podczas konfigurowania sieci, zaś adresy logiczne modułów sterujących nadawane są przez użytkownika za pomocą interfejsu użytkownika GUI.
- Każdy moduł sterujący może zapamiętać i zaadresować 64 moduły wykonawcze.
- Wszystkie moduły wykonawcze zapamiętane przez dany moduł sterujący mogą zostać połączone w maksymalnie 8 grup, dowolnie podczas konfiguracji sieci, reprezentujących pomieszczenia, nad którymi moduł ten ma kontrolę (np. pokoje).
- Kilka modułów sterujących może mieć kontrolę nad jednym modułem wykonawczym.
- W ramach graficznego interfejsu użytkownika modułu sterującego każdy moduł wykonawczy jest identyfikowany przez unikalną nazwę.
- Każda z 8 możliwych grup, w które mogą być łączone moduły wykonawcze może mieć nadaną nazwę, aktywowana lub wyłączona.
- Przewidziano 5 rodzajów modułów wykonawczych: wyłącznik 1-biegunowy, wyłącznik 2-biegunowy, ściemniacz, sensor temperatury, sterownik oświetlenia RGB LED.
- System iControl sygnalizuje dołączenie nowych, jeszcze nieskonfigurowanych modułów wykonawczych oraz wystąpienie błędów transmisji.
- System iControl umożliwia usuwanie modułów wykonawczych z sieci, a co za tym idzie – rekonfigurację sieci.

Dodatkowe materiały na FTP:

ftp://ep.com.pl, user: 11877, pass: ragjkdt9

- wzory płytek PCB

Projekty pokrewne na FTP:
(wymienione artykuły są w całości dostępne na FTP)

AVT-5490	iControl – system automatyki domowej (1, 2) (EP 1-2/2015)
AVT-5313	IntelliDom – System sterowania inteligentnego budynku z interfejsem ZigBee (EP 10-11/2011)
AVT-5126	IDom – System automatyki domowej (EP 3/2008)

* Uwaga:
Zestawy AVT mogą występować w następujących wersjach:

AVT xxxx UK	to zaprogramowany układ. Tylko i wyłącznie. Bez elementów dodatkowych.
AVT xxxx A	płytka drukowana PCB (lub płytki drukowane, jeśli w opisie wyraźnie zaznaczono), bez elementów dodatkowych.
AVT xxxx A+	płytka drukowana i zaprogramowany układ (czyli połączenie wersji A i wersji UK) bez elementów dodatkowych.
AVT xxxx B	płytka drukowana (lub płytki) oraz komplet elementów wymienionych w załączniku pdf
AVT xxxx C	to nic innego jak zmontowany zestaw B, czyli elementy wlutowane w PCB. Należy mieć na uwadze, że o ile nie zaznaczono wyraźnie w opisie, zestaw ten nie ma obudowy ani elementów dodatkowych, które nie zostały wymienione w załączniku pdf
AVT xxxx CD	oprogramowanie (nieczęsto spotykana wersja, lecz jeśli występuje, to niezbędne oprogramowanie można ściągnąć, klikając w link umieszczony w opisie kitu)

Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik pdf! Podczas składania zamówienia upewnij się, którą wersję zamawiasz! (UK, A, A+, B lub C). *http://sklep.avt.pl*

Schemat czujnika temperatury do systemu iControl pokazano na **rysunku 1**. „Sercem” urządzenia jest mikrokontroler ATmega8 odpowiedzialny za realizację całej założonej funkcjonalności. Mikrokontroler za pomocą wbudowanego, sprzętowego interfejsu TWI steruje pracą modemu CY8CPLC10. Dane są transmitowane oraz analizowane z użyciem przerwania zewnętrznego INT1. Ponadto wykorzystano programową implementację obsługi interfejsu 1-Wire (wyprowadzenie PD2 mikrokontrolera), dzięki czemu zrealizowano współpracę sterownika ze scalonym termometrem typu DS18S20 zapewniając dokładny pomiar temperatury otoczenia. Część analogowa iControlSensor jest w zasadzie taka sama, jak w module iControlSwitch, więc nie będę powielał szczegółowego opisu zastosowanych rozwiązań sprzętowych odsyłając dociekliwych Czytelników do artykułu wspomnianego we wstępie. Jedyne, co należy podkreślić to fakt, iż rozbudowana aplikacja toru analogowego wynika wyłącznie z chęci sprostania wymogom norm CENELEC EN50065-1:2001 oraz FCC part 15 (w jej konstrukcji posłużono się zresztą schematem aplikacyjnym firmy Cypress – nota aplikacyjna CY3274).

Z uwagi na fakt, że moduł iControlSensor „zadowala się” znacznie mniejszym zapotrzebowaniem na energię, niż wspomniany moduł iControlSwitch, w jego zasilaczu zastosowano typowy transformator do druku

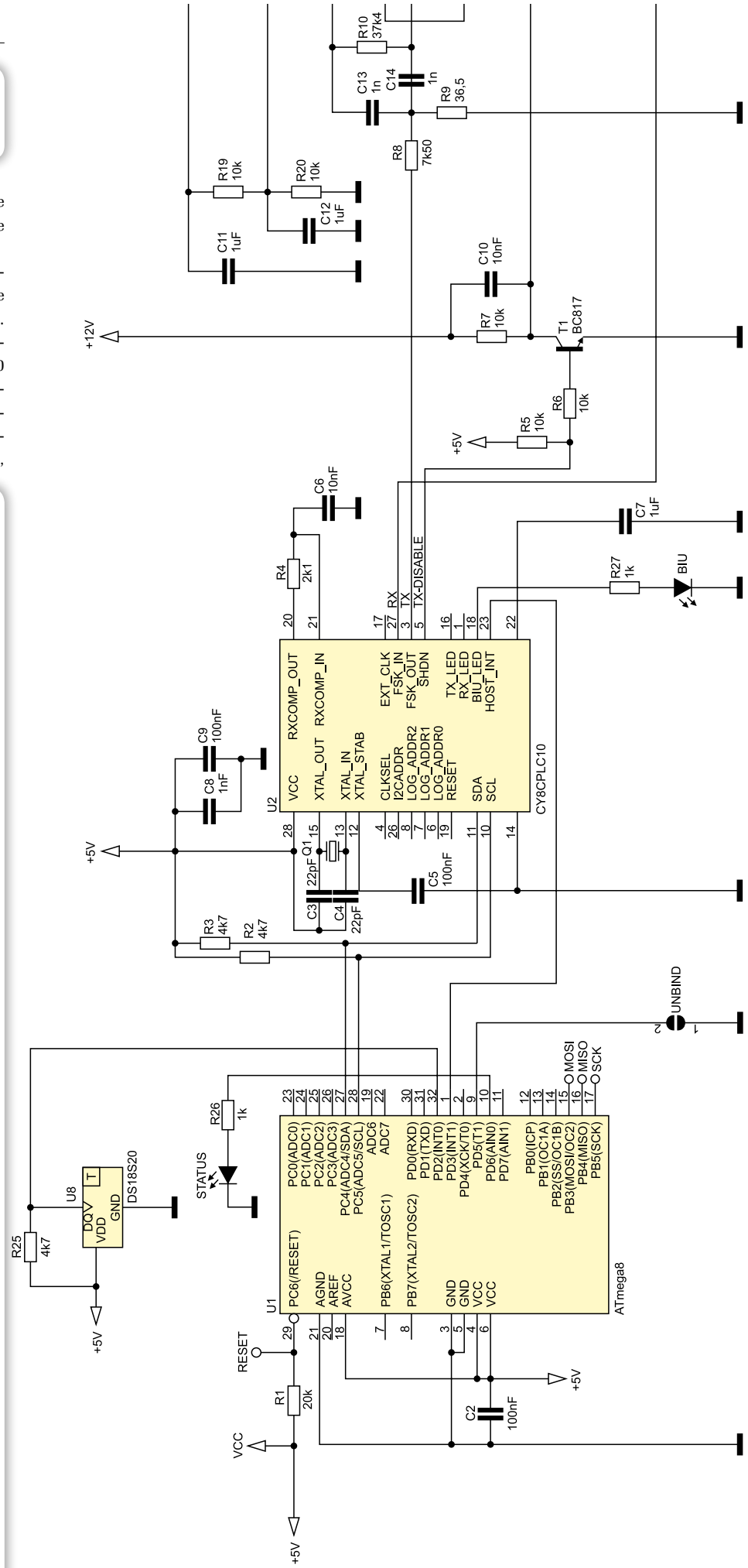
Ustawienie ważniejszych fusebitów:
CKSEL3...0: 0100
SUT1...0: 10
CKOPT: 1
EESAVE: 0

(15 V/2,6 VA) oraz dwa stabilizatory liniowe (78M12 i 78M05) zapewniającą odpowiednie napięcie zasilające.

Przejdźmy do szczegółów implementacyjnych programu obsługi aplikacji, które nie zostały opisane w poprzednim artykule. Wszystkie moduły systemu iControl wykorzystują wbudowany w modem CY8CPLC10 mechanizm detekcji zajętości magistrali danych (*Band In Use*), więc wysłanie pojedynczego pakietu danych, w zależności od parametrów transmisji i stanu tejże magistrali,

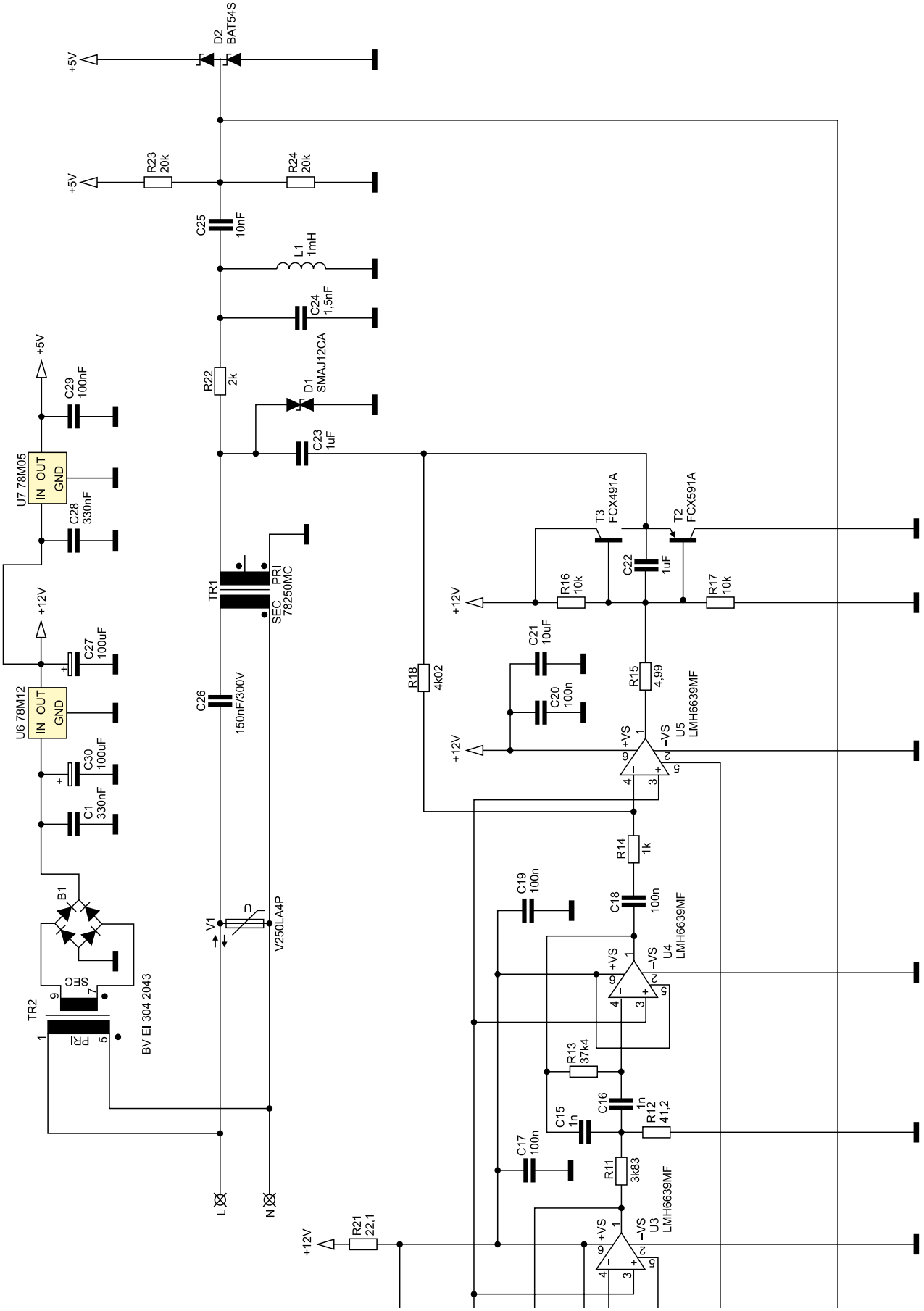
Wykaz elementów

Rezystory: (obudowy SMD 0805)
R1: 20 kΩ
R2, R3, R25: 4,7 kΩ
R4: 2,1 kΩ/1%
R5...R7, R16, R17, R19, R20: 10 kΩ
R8: 7,5 kΩ/1%
R9: 36,5 Ω/1%
R10, R13: 37,4 kΩ/1%
R11: 3,.83 kΩ/1%
R12: 41,2 Ω/1%
R14, R26, R27: 1 kΩ/1%
R15: 4,99 Ω/1%
R18: 4,02 kΩ/1%
R21: 22,1 Ω/1%
R22: 2 kΩ/1%
R23, R24: 20 kΩ
Kondensatory: (SMD 0805):
C1, C28: 330 nF
C2, C5, C9, C17...C20, C29: 100 nF
C3, C4: 22 pF
C6, C10, C25: 10 nF
C7, C11, C12, C22, C23: 1 µF
C8, C13...C16: 1000 pF/1%
C21: 10 µF (SMD 1206)
C24: 1,5 nF
C26: 150 nF/300 V (polipropylenowy klasy X1, r=15 mm)
C27: 100 µF/16 V (r=2,5 mm)
C30: 100 µF/25 V (r=2,5 mm)
Półprzewodniki:
U1: ATmega8 (TQFP32)
U2: CY8CPLC10 (SSOP28)
U3...U5: LMH6639MF (SOT23-6)
U6: 78M12 (DPAK)
U7: 78M05 (DPAK)
U8: DS18S20 (TO-92)
T1: BC817 (SOT23)
T2: FCX591A (SOT89)
T3: FCX491A (SOT89)
D1: SMAJ12CA (DO-214AC)
D2: BAT54S (SOT23)
STATUS: zielona dioda LED (SMD 0805)
BIU: czerwona dioda LED (SMD 0805)
B1: mostek prostowniczy 1A/50 V (raster 5 mm)
Inne:
TR1: transformator izolacyjny SMD Murata 78250MC
TR2: transformator do druku HAHN BV EI 304 2043 (15 V/2,6 VA)
L1: dławik SMD 1 mH (1812)
Q1: rezonator kwarcowy CFPX217 (SMD 32768 Hz)
V1: warystor V250LA4P

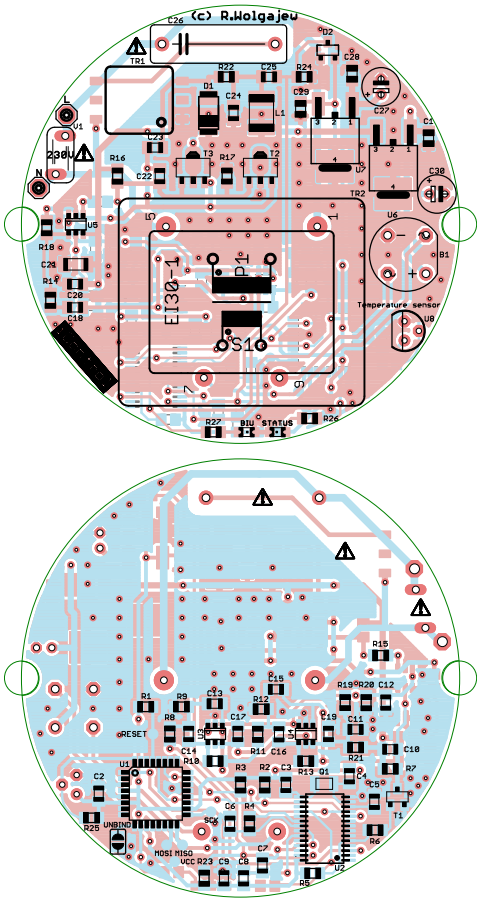


Rysunek 1. Schemat ideowy modułu iControlSensor

może zając nawet 1,5 sekundy. Dlatego rozwiązanie przesyłania danych w postaci inicjacji transmisji danych i oczekiwania na rezultat jej wykonania byłoby nieefektywne. Z punktu widzenia programu sterującego wstrzymywałoby obsługę innych, ważnych procesów. W związku z tym w aplikacji programu obsługi urządzeń systemu iControl zaimplementowano specjalny mechanizm wysyłania, synchronizacji i kontroli pakietów danych. Po pierwsze, zastosowano specjalny bufor kołowy pakietów transmisji



Rysunek 1. c.d.



Rysunek 2. Schemat montażowy modułu iControlSensor

(z odpowiednimi wskaźnikami miejsca zapisu i odczytu), którego elementem jest struktura danych opisująca kompletną ramkę transmisji, a której konstrukcję (wraz z odpowiednimi zmiennymi) pokazano na **listingu 1**. Dodatkowo, przewidziano 2 funkcje obsługi przerwań ISR: od Timera1 (dokładnie, od porównania wartości Timera1 z wartością rejestru ORC1A), wywoływaną cyklicznie co 100 milisekundy, której zadaniem jest sprawdzanie czy są jakiekolwiek dane przeznaczone do wysłania, inicjacja procesu wysłania danych, jego nadzór i obsługa bufora kołowego oraz funkcję obsługi przerwania zewnętrznego INT1, której zadaniem jest zarówno odbiór „zwykłych" danych przesyłanych przez moduł sterujący iControlMaster oraz odbiór statusów bieżącej i zainicjowanej we wcześniej wspomnianej procedurze obsługi przerwania, transmisji danych. W tym, drugim przypadku, funkcja obsługi przerwania zewnętrznego INT1 sprawdza status wysyłania bieżącej transmisji danych i w zależności od jego stanu modyfikuje bieżąca strukturę danych dając asumpt do ponowienia tejże transmisji (w przypadku jej niepowodzenia i nie przekroczenia maksymalnej liczby retransmisji) lub też kończąc jej przebieg poprzez ustawienie odpowiednich flag w strukturze danych (przeprowadzenia transmisji zakończonego powodzeniem lub niepowodzeniem) jak i flag dla pętli głównej aplikacji, dzięki czemu program główny informowany jest na bieżąco o aktualnym stanie systemu. W ten prosty sposób unika się niepożądanego wstrzymywania pracy aplikacji w przypadku problemów z komunikacją po magistrali PLC. W celu zrealizowania powyższej funkcjonalności, po pierwsze, przewidziano prostą funkcję narzędziową, której zadaniem jest dodanie do nadawczego bufora kołowego, danych przeznaczonych do wysłania, a którą pokazano na **listingu 2**.

Kolejnym krokiem jest implementacja funkcji, której cykliczne wywoływanie (co 100 ms) zapewnia sprawdzanie czy są jakiekolwiek dane przeznaczone do wysłania, inicjacja procesu wysłania danych, jego nadzór i obsługa bufora kołowego, a którą to pokazano na **listingu 3**.

Jak widać, przedstawiona implementacja wspomnianej funkcji zapewnia wykonanie 3 prób wysyłania kompletnej wiadomości PLC, przy czym dla każdej z wymienionych prób zmieniana jest wartość progu dla mechanizmu BIU, który to dla kolejnych transmisji przyjmuje następujące ustawienia:

- próba pierwsza: 96 dBμVrms,
- próba druga: 99 dBμVrms,
- próba trzecia: wyłączenie mechanizmu BIU.

Po trzech, nieudanych próbach wysłania pakietu PLC, aplikacja główna otrzymuje

Listing 1. Realizacja bufora kołowego i deklaracje zmiennych odpowiedzialnych za proces wysyłania danych PLC

```
//Definicja typu bufora kołowego danych przeznaczonych do wysłania
typedef struct
{
    volatile uint8_t Status;  //Status transmisji bieżącego pakietu danych
    volatile uint8_t Trials;  //Liczba prób retransmisji
    volatile uint8_t Command;  //Rozkaz sterujący towarzyszący przesyłanym danym
    volatile uint8_t Data[10];  //Tablica danych przeznaczonych do transmisji
    volatile uint8_t Bytes;  //Liczba danych przeznaczonych do transmisji
    volatile uint8_t DAtype;  //Rodzaj adresu urządzenia docelowego: Logiczny/Grupowy
}txBufferType;

//Definicja statusów procesu wysyłania
#define BUF_EMPTY 0
#define BUF_READY_TO_SEND 1
#define BUF_WAITING_FOR_STATUS 2
#define BUF_SENT 3
#define BUF_FAILED 4

//Definicje rozmiaru bufora nadawczego
#define TX_BUF_SIZE 16  //Rozmiar bufora nadawczego
#define TX_BUF_MASK (TX_BUF_SIZE - 1)  //Maska bufora nadawczego

//Deklaracje zmiennych odpowiedzialnych za obsługę bufora
//nadawczego jak i samego procesu nadawania
txBufferType txBuffer[TX_BUF_SIZE];  //Nadawczy bufor kołowy
static volatile uint8_t txBufferWriteTo;  //Wskaźnik bieżącego miejsca w buforze przeznaczonego do zapisu
static volatile uint8_t txBufferReadFrom;  //Wskaźnik bieżącego miejsca w buforze przeznaczonego do odczytu
static volatile uint8_t txMessages;  //Aktualna liczba wiadomości przeznaczonych do wysłania
```

Listing 2. Funkcja odpowiedzialna za dodanie danych do nadawczego bufora kołowego

```
//Funkcja dodaje do bieżącego miejsca zapisu bufora nadawczego nową
//strukturę danych przeznaczonych do wysłania
void addToTransmitBuffer(uint8_t Command, volatile uint8_t *Data, uint8_t Bytes, uint8_t DAtype)
{
    register uint8_t i = 0, writePointer = txBufferWriteTo;
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
    {
        txBuffer[writePointer].Status = BUF_READY_TO_SEND;
        txBuffer[writePointer].Trials = 0;
        txBuffer[writePointer].Command = Command;
        txBuffer[writePointer].Bytes = Bytes;
        while(Bytes--) {txBuffer[writePointer].Data[i]= Data[i]; i++;}
        txBuffer[writePointer].DAtype = DAtype;
        txBufferWriteTo = (writePointer+1) & TX_BUF_MASK;  //Nowy adres miejsca zapisu
        txMessages++;
    }
}
```

Fotografia 3. Widok obwodu drukowanego modułu iControlSensor od strony BOTTOM.

informację o błędzie transmisji danych, co pozwala podjąć odpowiednie akcje. Ostatnią funkcją, niezbędną z punktu widzenia zastosowanego mechanizmu transmisji, jest funkcja obsługi przerwania zewnętrznego INT1, której zadaniem jest odbiór „zwykłych" danych przesyłanych przez moduł sterujący iControlMaster oraz odbiór statusów bieżącej i zainicjowanej we wcześniej wspomnianej funkcji obsługi przerwania (list. 3), transmisji danych. Funkcji pokazano na **listingu 4**. Funkcję inicjalizacji modułu iControlSensor zawiera odpowiednie procedury konfigurujące obsługę przerwań odpowiedzialnych za realizację procesu wysyłania danych – pokazano ją na **listingu 5**.

Montaż

Schemat montażowy modułu iControlSensor pokazano na **rysunku 2**. Sposób wykonania obwodu drukowanego umożliwia zamontowanie gotowego urządzenia w typowej puszcze elektroinstalacyjnej o średnicy 60 mm i głębokości 40 mm. Z uwagi na sporą liczbę

Listing 3. Funkcja odpowiedzialna za inicjowanie i nadzór procesu wysyłania danych PLC

```
ISR(TIMER1_COMPA_vect)
{
    //Lokalne bufory zmiennych globalnych typu volatile - optymalizacja funkcji ISR
    uint8_t Bytes, Command, Trials, DAtype, readPointer = txBufferReadFrom;
    if(txMessages)  //Sprawdzenie czy są jakieś dane do wysłania
    {
        Bytes = txBuffer[readPointer].Bytes;
        Command = txBuffer[readPointer].Command;
        Trials = txBuffer[readPointer].Trials;
        DAtype = txBuffer[readPointer].DAtype;
        //W zależności od statusu bieżącej operacji, podejmujemy różne akcje
        switch(txBuffer[readPointer].Status)
        {
            case BUF_READY_TO_SEND:
            //W zależności od liczby wykonanych prób, podejmujemy różne akcje
                switch(Trials)
                {
                    //Inicjowanie transmisji
                    case 0:
                        //Aby nie czytać rejestru PLC_MODE_REG by
                        //uaktywniać BIU to wczytujemy do niego parametry startowe
                        writePLCregister(PLC_MODE_REG, TX_ENABLE|RX_ENABLE|RX_OVERRIDE|ENABLE_BIU|CHECK_DA|VERIFY_
PACKET_CRC8);
                        //Ustawienie poziomu sygnału dla mechanizmu CSMA
                        writePLCregister(THRESHOLD_NOISE_REG, BIU_TRESHOLD_96DBUV);
                        //Ustawiamy typ adresu przeznaczenia. Typ adresu nadajnika - zawsze logiczny
                        setPLCtxAddrType(TX_SA_TYPE_LOGICAL, DAtype);
                        //Ustawiamy adres DA: grupowy Masterów lub LA powiązanego Mastera
                        if(DAtype==TX_DA_TYPE_GROUP) setPLCtxDA(TX_DA_TYPE_GROUP, &(uint8_t) {MASTER_GROUP_ADDR});
                        else setPLCtxDA(TX_DA_TYPE_LOGICAL, (uint8_t *) &Node.bindMasterLA);
                        writePLCregister(TX_COMMAND_ID_REG, Command);
                        writePLCregisters(TX_DATA_REG, (uint8_t *) txBuffer[readPointer].Data, Bytes);
                        //Inicjacja procesu transmisji, w tym specyfikacja liczby
                        //bajtów przeznaczonych do wysłania
                        writePLCregister(TX_MESSAGE_LENGTH_REG, Bytes|SEND_MESSAGE);
                        break;
                    //Zwiększenie wartości BIU i restart transmisji
                    case 1:
                        writePLCregister(THRESHOLD_NOISE_REG, BIU_TRESHOLD_99DBUV);
                        writePLCregister(TX_MESSAGE_LENGTH_REG, Bytes|SEND_MESSAGE);
                        break;
                    //Wyłączenie BIU dla Trials = 2  i restart transmisji
                    case 2:
                        writePLCregister(PLC_MODE_REG, TX_ENABLE|RX_ENABLE|RX_OVERRIDE|DISABLE_BIU|CHECK_DA|VERIFY_
PACKET_CRC8);
                        writePLCregister(TX_MESSAGE_LENGTH_REG, Bytes|SEND_MESSAGE);
                        break;
                    }
                txBuffer[readPointer].Status = BUF_WAITING_FOR_STATUS;  //Nie robimy nic
                break;
            //Nie robimy nic, gdyż czekamy na rezultat bieżącej transmisji
            case BUF_WAITING_FOR_STATUS: break;
            case BUF_FAILED:
            case BUF_SENT:
            //Usunięcie elementu z bufora - zarówno w przypadku pomyślnego jak i niepomyślnego wysłania
                txBuffer[readPointer].Status = BUF_EMPTY;
                //Przesunięcie ogona bufora, bo zdjęliśmy jeden element
                txBufferReadFrom = (readPointer+1) & TX_BUF_MASK;
                txMessages--;  //Zmniejszenie liczby wiadomości przeznaczonych do wysłania
                break;
        }
    }
}
```

elementów i małą powierzchnię, zastosowano dość gęsty montaż elementów SMD po obu stronach laminatu. Na płytce urządzenia poprowadzono obszerne pola masy po obu stronach obwodu drukowanego oraz zastosowano szereg przelotek pomiędzy nimi w celu zmniejszenia pojemności pasożytniczych. Z uwagi na zastosowanie niewielkich elementów SMD, montaż tego typu układu najlepiej jest przeprowadzić z użyciem stacji lutowniczej wyposażonej w grot o niewielkiej średnicy, odpowiedniej jakości topników lutowniczych oraz dysponując sporym doświadczeniem w tej kwestii. Jak zwykle, montaż taki zaczynamy od przylutowania wszystkich półprzewodników w obudowach SMD (po obu stronach laminatu), następnie lutujemy rezystory, kondensatory, elementy indukcyjne, a na samym końcu wszystkie elementy przeznaczone do montażu przewlekanego. Wygląd obwodu drukowanego modułu od strony BOTTOM pokazano na **fotografii 3**. W układzie prototypowym, w odróżnieniu od projektu docelowego, zastosowano rezonator kwarcowy przeznaczony do montażu przewlekanego.

Co ważne, poprawnie zmontowany układ nie wymaga żadnych regulacji i powinien działać tuż po włączeniu zasilania. Tak jak w poprzednim module typu Slave, na płytce drukowanej modułu iControlSensor przewidziano specjalną zworkę oznaczoną „UNBIND". Służy ona do wyzerowania stanu modułu wykonawczego do stanu wyjściowego (moduł nieskonfigurowany), przy czym stan jej sprawdzany jest wyłącznie podczas włączania urządzenia, zaś potwierdzeniem zadziałania tego mechanizmu jest zaświecenie diody STATUS przez 0,5 sekundy. Funkcji tej można użyć na przykład w przypadku, gdybyśmy odłączyli zalogowany wcześniej i skonfigurowany moduł wykonawczy od sieci (z nadanym adresem LA) i chcieli go użyć w innej sieci. W takim przypadku, jako że moduł ten miałby już nadany adres LA, nie zgłosiłby się automatycznie w nowej sieci i jedynym sposobem „zmuszenia" go do tej czynności byłoby sprzętowe, lokalne zresetowanie jego stanu. Niemniej jednak, nie polecam tego scenariusza postępowania, gdyż konfiguracja czy rekonfiguracja sieci powinna być procesem przemyślanym i wykonywanym z poziomu modułów sterujących.

Listing 4. Funkcja ISR INT1 współodpowiedzialnej za obsługę procesu wysyłania danych PLC

```
ISR(INT1_vect)
{
    //Odebrane dane (co najwyżej 1 bajt, gdyż nie ma rozkazów,
    //którym towarzyszyło by więcej danych)
    uint8_t packetData;
    uint8_t receivedCmnd, packetLength, whatHappened, readPointer = txBufferReadFrom, saveNodeSettings = 0;
    //W pierwszej kolejności sprawdzamy czy INT1 zostało wywołane
    //statusem wysyłania danych czy nadchodzącą wiadomością
    whatHappened = readPLCintRegister() & ~(STATUS_VALUE_CHANGE|STATUS_RX_PACKET_DROPPED);
    if(whatHappened & STATUS_RX_DATA_AVAIBLE)  //Odebrano nową wiadomosc
    {
        //Odczytujemy cała, przesłaną wiadomość by ustalić jej typ: rodzaj
        //rozkazu, towarzyszące dane i ich ilość
        readPLCrxPacket(&receivedCmnd, &packetData, &packetLength);
        //Teraz w zależności od rodzaju tejże wiadomości, podejmujemy
        //określone kroki, sygnalizując co trzeba programowi głównemu
        switch(receivedCmnd)
        {
            case CMD_SELECT_NODE: Node.State = NODE_SELECTED; break;
            case CMD_UNSELECT_NODE: Node.State = NODE_AVAILABLE; break;
            case CMD_SET_LOGICAL_ADDR:
            //Wyłączamy diodę STATUS
                STATUS_LED_OFF;
                //Aktualizacja parametrów Node, zapisanie ich do EEPROM
                //jak i aktualizacja adresu LA układu Slave
                Node.State = NODE_ACTIVE;
                Node.LA = packetData;
                //Ustalenie LA Mastera, który nadaje nam nasz własny adres LA
                Node.bindMasterLA = getPLCrxSA();
                //Ustawienie adresu logicznego modułu Slave
                setPLCnodeLA(Node.LA);
                saveNodeSettings = 1;
                break;
            case CMD_UNBIND_NODE:
                //Aktualizacja parametrów Node, zapisanie ich do EEPROM
                //oraz aktualizacja adresu LA układu Slave
                Node.State = NODE_AVAILABLE;
                Node.LA = UNCONFIGURED_SLAVE_ADDR;
                Node.bindMasterLA = MASTER_GROUP_ADDR;
                //Ustawienie adresu logicznego na wartość nieskonfigurowanego
                //układu Slave (LA=0x00)
                setPLCnodeLA(UNCONFIGURED_SLAVE_ADDR);
                saveNodeSettings = 1;
                break;
            case CMD_MAKE_SHARED_NODE:
                //Aktualizacja parametrów Node i zapisanie ich do EEPROMa
                Node.State = NODE_ACTIVE_SHARED;
                saveNodeSettings = 1;
                break;
            case CMD_GET_NODE_STATUS:
                //Master zażądał przesłania stanu Node, więc dokonamy
                //tego w pętli głównej
                sendNodeStatus = 1;
                break;
            }
            //Jeśli żądano aktualizacji danych Node w EEPROMie to zapisujemy je
        if(saveNodeSettings) eeprom_write_block(&Node, &NodeEE, sizeof(Node));
    }
    else  //Odebrano status wysyłania wiadomosci
    {
        //Wysłanie udane
        if(whatHappened & STATUS_TX_DATA_SENT) txBuffer[readPointer].Status = BUF_SENT;
        //Magistrala zajęta - podejmujemy kolejną próbę, jeśli nie przekroczono liczby prób
        else if(whatHappened & STATUS_BUSY)
        {
            if(++txBuffer[readPointer].Trials>2) txBuffer[readPointer].Status = BUF_FAILED;
            else txBuffer[readPointer].Status = BUF_READY_TO_SEND;
        }
        //STATUS_TX_NO_ACK, STATUS_TX_NO_RESP -> Nieudane wysłanie
        else txBuffer[readPointer].Status = BUF_FAILED;
    }
}
```

Na płytkach obu układów zbudowano kompletne układy zasilające zasilane napięciem sieciowym 230 V AC i dlatego istnieje niebezpieczeństwo porażenie elektrycznego, co stanowi zagrożenie dla życia i zdrowia. W związku z tym, montaż układów w tym zakresie powierzyć należy osobie posiadającej uprawnienia elektryczne w zakresie eksploatacji urządzeń o napięciu do 1 kV. Miejsca na obwodach płytek drukowanych, gdzie występuje wysokie napięcie groźne dla życia i zdrowia oznaczone zostały odpowiednimi opisami.

Robert Wołgajew, EP

Listing 5. Funkcja odpowiedzialna za inicjalizację modułu iControlSensor

```
void initPLCdevice(void)
{
    //Konfiguracja interfejsu I2C (TWI)
    i2cInit();
    //Uruchomienie i podstawowa konfiguracja modemu PLC
    writePLCregister(PLC_MODE_REG, TX_ENABLE|RX_ENABLE|RX_OVERRIDE|ENABLE_BIU|CHECK_DA|VERIFY_PACKET_CRC8);
    //Ustawienie poziomu sygnału dla mechanizmu CSMA zapewniającego
    //wielodostęp do medium transmisyjnego
    writePLCregister(THRESHOLD_NOISE_REG, BIU_TRESHOLD_96DBUV);
    //Konfiguracja parametrów transmisji
    writePLCregister(MODEM_CONFIG_REG, MODEM_BPS_2400|MODEM_FSK_BAND_DEV_3KHZ);
    //Uruchomienie przerwań dla wybranych zdarzeń (aktywny poziom niski
    //na wyprowadzeniu HOST_INT)
    writePLCregister(INTERRUPT_ENABLE_REG, INT_POLARITY_LOW|INT_UNABLE_TO_TX|INT_TX_DATA_SENT|INT_TX_NO_ACK|INT_TX_NO_
RESP|INT_RX_DATA_AVAILABLE);
    //Ustawienie trybu potwierdzania pakietów danych oraz liczby
    //prób transmisji = 3 (domyślne logiczne typy adresów SA i DA)
    writePLCregister(TX_CONFIG_REG, TX_SERVICE_ACKNOWLEDGED|0x03);
    //Ustawienie wzmocnienia dla modułu nadajnika PLC
    writePLCregister(TX_GAIN_REG, TX_GAIN_LEVEL_1550MV);
    //Ustawienie czułości dla modułu odbiornika PLC
    writePLCregister(RX_GAIN_REG, RX_GAIN_LEVEL_250UV);
    //Przyporządkowanie Slave'a do grupy SLAVE_GROUP_ADDR (0x10)
    setPLCnodeGA(SLAVE_GROUP_ADDR);
   //Konfigurowanie i załączenie przerwania INT1 odpowiedzialnego za
    //ustawianie flagi zmiany stanu rejestru INTERRUPT_STATUS_REG PLC
    HOST_INT_PORT |= (1<<HOST_INT_NR);  //Podciągnięcie INT1 do VCC
    MCUCR |= (1<<ISC11);  //Zbocze opadające na INT1 inicjuje przerwanie
    GICR |= (1<<INT1);  //Zezwolenie na przerwanie INT1
    //Konfigurowanie przerwania CompareA Timera1 wyznaczającego czas
     //obsługi wysyłania wiadomości - 100ms
    OCR1A = 782;  //Przerwanie co 100ms dla oscylatora 8MHz
    TCCR1B |= (1<<WGM12)|(1<<CS12)|(1<<CS10);  //Uruchomienie Timer1: Tryb CTC, Preskaler = 1024
    TIMSK |= (1<< OCIE1A);  //Zezwolenie na przerwanie od porównania - CompareA Timera1
}
```