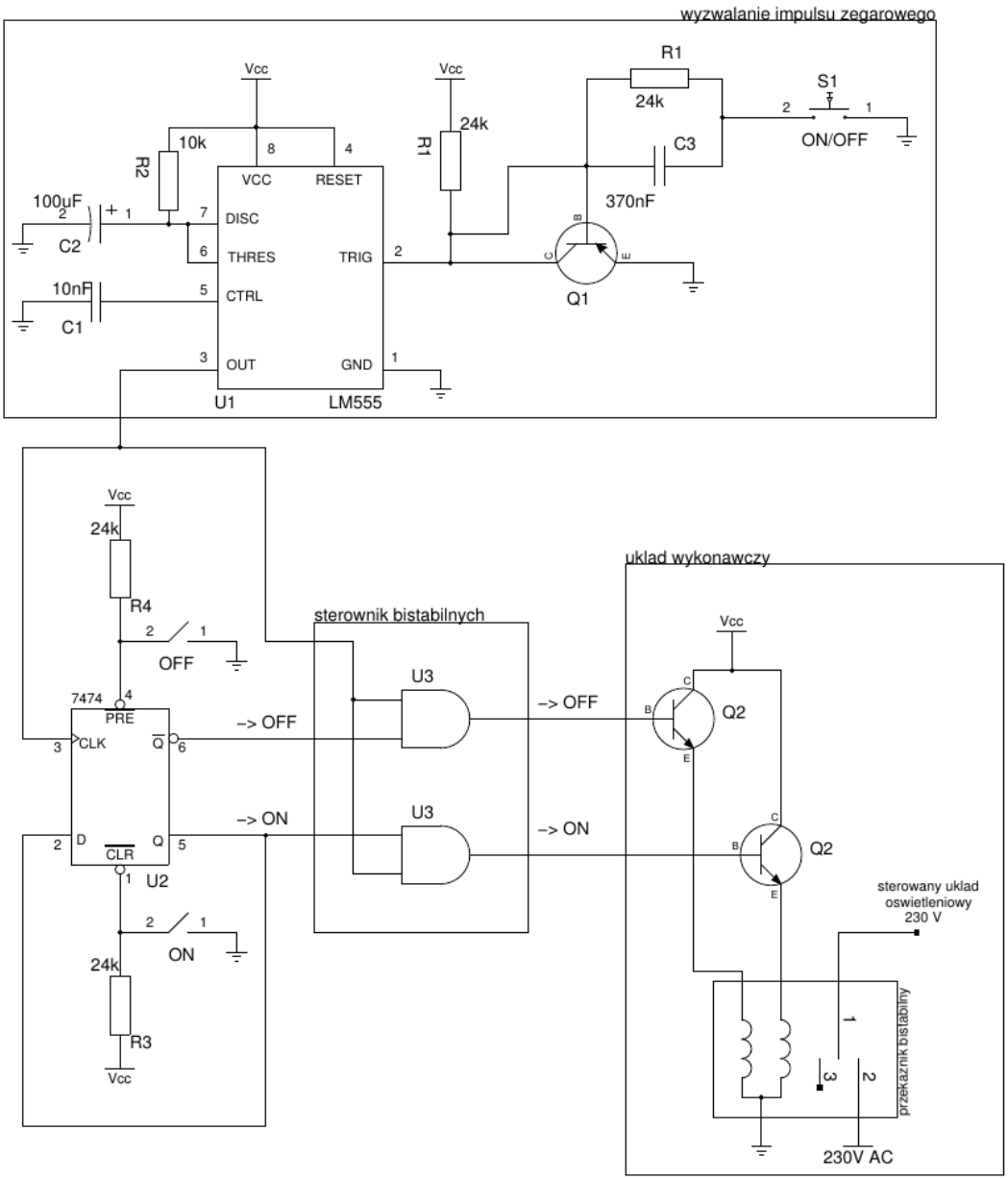
Ewolucja koncepcji projektów inteligentnego domu

Artykuł ma na celu prezentację rozwoju prezentowanych w serwisie projektów związanych z tzw. inteligentnym domem, czyli różnego rodzaju układów sterowania i automatyki domowej, a także przybliżyć interesujące z jakiś względów rozwiązania, które nie znalazły miejsca w aktualnych projektach.

Rozwiązania oparte na układach cyfrowych małej skali integracji

Pierwszym z projektów automatyki domowej był impulsowy sterownik oświetlenia w oparciu o układy cyfrowe serii 74 (TTL) oraz przekaźniki bistabilne. Umożliwiał on także zastosowanie zwykłych przekaźników (po wyeliminowaniu części układu odpowiedzialnego za impulsowość sterowania. Głównym problemem takiej realizacji było to iż układ wymagałaby użycia wielu elementów (dla sterownika 6-cio kanałowego: 6x NE555, 3x 4747, 3x 4708 oraz wiele elementów biernych) a co za tym idzie rozbudowanego i zajmującego sporo miejsca PCB. Zamieszczam tutaj powstały wtedy projekt takiego sterownika, jednak należy pamiętać że projekt nigdy nie doczekał się realizacji czy nawet testów, więc ma prawo nie działać.



Mikrokontrolery AVR

Celem zmniejszenia rozmiarów oraz uproszczenia budowy zdecydowałem się na zastosowanie mikrokontrolera ATmega8. Układ taki miał też większe możliwości (w szczególności prostsze zarządzanie zdalne z wykorzystaniem

RS-485 lub RS-422A). Równocześnie w miejsce przekaźników zdecydowałam się także na zastosowanie zestawu optotriak + triak. Pierwotna wersja sterownika posiadała na swoim PCB autonomiczne zasilacze sieciowe (niezależne dla każdego sterownika). Podyktowane zostało to rozwiązaniem zasilania instalacji oświetleniowej (układ zabezpieczeń nad i różnicowoprądowych) nie bardzo pozwalającym na centralne zasilenie (wyłączenie jego zabezpieczenia rozwalałoby cały system, który obecnie podzielony jest na strefy z niezależnymi zabezpieczeniami). Został zrealizowany jeden prototypowy sterownik (działający p;o dziś dzień) wg tego projektu.

Porównanie z koncepcją realizacji na elementach dyskretnych pokazuje przewagę rozwiązań opartych na mikrokontrolerze (gdzie elektronika na układach dyskretnych pełni tylko pomocniczą funkcję). Jest ona spowodowana ogromnymi możliwościami mikrokontrolerów oraz drastycznym uproszczeniem budowy układu, zmniejszeniem rozmiarów, a często także kosztów w stosunku co do realizacji analogicznych układów cyfrowych w oparciu o bramki, przerzutniki itp..

W drugiej wersjach sterownika scentralizowane zostało ich zasilanie, pojawiła się funkcja regulacji jasności wykonana w technologii Pulse Width Modulation dla jednej linii, a także (obok typowej "płaskiej" wersji) została zaprojektowana wersja dwupoziomowa do umieszczenia w typowej obudowie na szynę DIN.

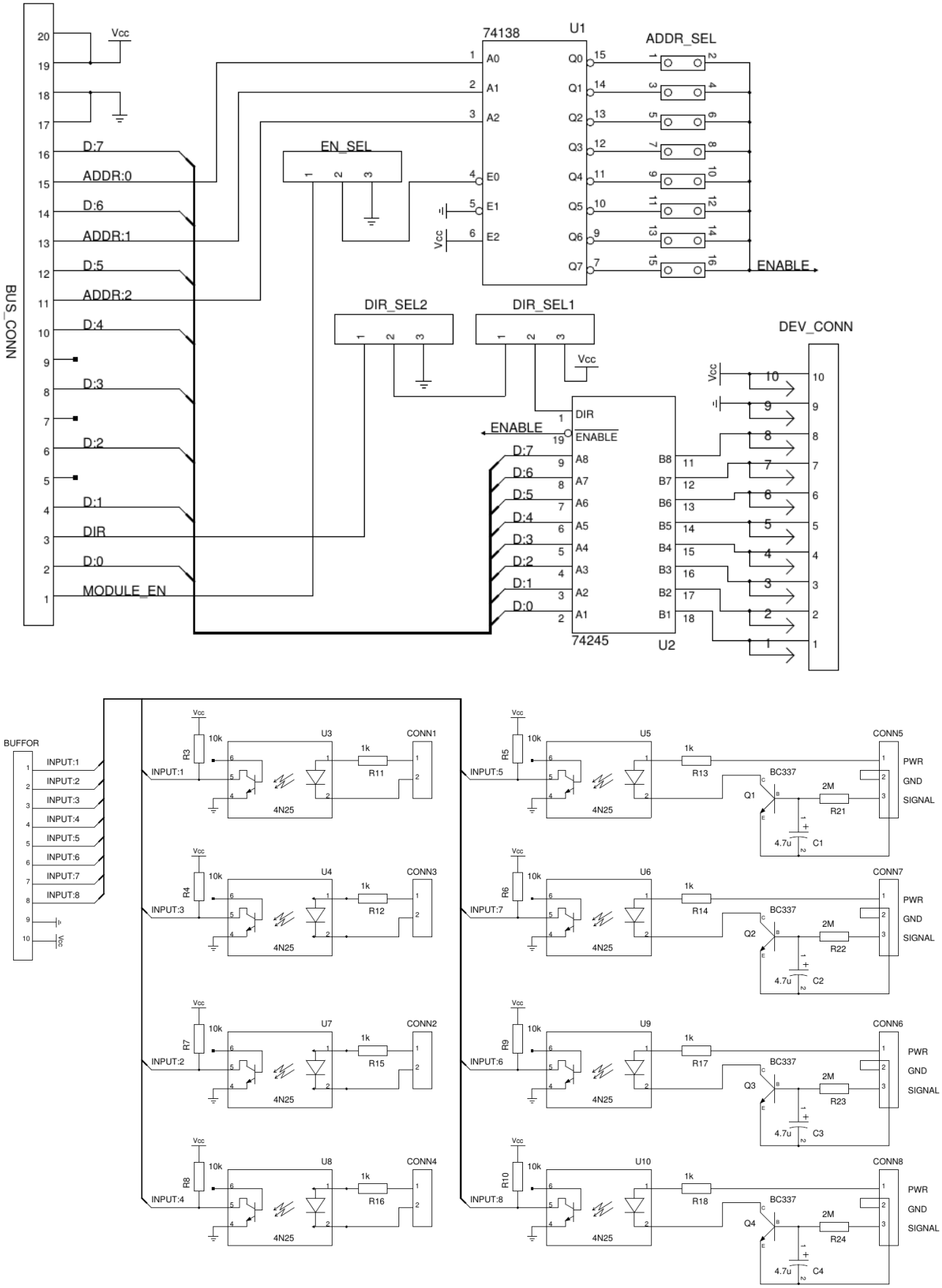
W trzeciej wersji sterowanie PWM zostało rozszerzone na wszystkie linie, a dzięki przeniesieniu obsługi linii sterujących z przełączników do innego modułu (pozostało tylko sterowanie po RS-485) zwiększono liczbę linii do 15tu. W (nie zrealizowanej) erracie tej wersji wprowadzone zostały także opcjonalne układy gasikowe (umożliwiające poprawną pracę z obciążeniem indukcyjnym - świetlówki) i dławiki. Z wersją trzecią zmieniona została także sama metoda projektowania - wykorzystana została powtarzalność układu do hierarchizacji projektu - patrz hierarchiczne schematy w gEDA.

Sterownik w trzeciej wersji zaprojektowany został do wykorzystania w 19" systemie rozdzielnic elektrycznych opisywanym w elektroniczny dom. Sterownik przystosowany był do montażu (max po dwa moduły sterownika) w obudowach rack 19" 3U/250mm mających od frontu szyna DIN do montażu zabezpieczeń (opcjonalnie można też umieścić szynę DIN do montażu dodatkowej aparatury prostopadle do frontowej wewnątrz obudowy panelu). Sterownik montowany powinien być na przestrzeni środkowego 1U. Sterownik może być montowany w pod-obudowie posiadającej otwarte wyprowadzenie kabli podłączeniowych od tyłu (do złączek szynowych montowanych na tylnej ścianie szafy) i zdejmowalną górną pokrywę. Na przedniej maskownicy aparatury modułowej można zamontować dodatkowo diody kontrolne (zasilanie sterownika + napięcie kontrolne sterownika + stan każdej linii) oraz gniazdo RJ-11 dla RS485 i gniazdo zasilania DC. Moduł sterownika odpowiedzialny za obsługę sygnałów wejściowych jest montowany w osobnej obudowie 1U, może być także zintegrowany z centralką alarmową itp urządzeniami. Zobacz także zdjęcia prototypu

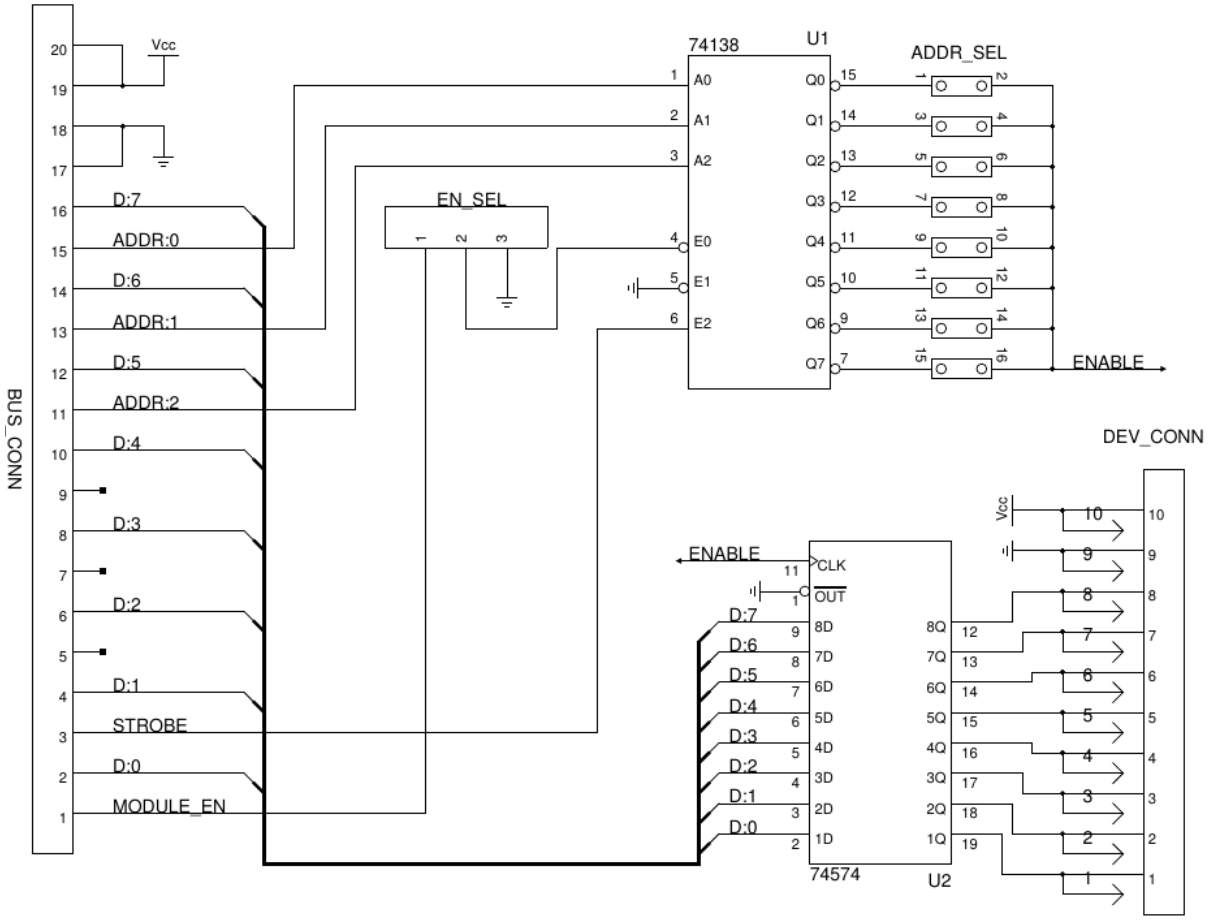
Centralka

Jak wspomniałem funkcje obsługi wejść (przełaczników) w wrsji trrzeciej zostały przekazane do innego modułu - była nim centralka alarmowa, oparta na mikrokontrolerze Atmega16. Projekt oparty był na modułach podłączanych do mikrokontrolera poprzez magistralę równoległą utworzoną z portu B mikrokontrolera (8 bitów danych) i części portu A (6 bitów adresowych). Detekcja adresu w modułach realizowana była w oparciu 3 najmłodsze bity (i sygnał zezwolenia) adresu na układzie 74hc183. 3 najstarsze bity adresu mogłyby·ć wykorzystane przez switch magistrali oparty także na układzie 74hc183 do generowania sygnału zezwolenia.:

Moduł wejścia-wyjścia (oparty na buforze dwukierunkowym) i bazujący na nim moduł wejść optoizolowanych (4 zwykłe, 4 z filtrem rc):



Moduł wyjścia (oparty na rejestrze 8 bitowym) i bazujący na nim moduł kluczy NPN oraz NPN (sterowanie układem od strony masy) + P-MOSFET (sterowanie układem od strony bieguna dodatniego):



Centralka - wnioski po-wdrożeniowe

- nie najlepszym pomysłem jest bezpośrednie przyłączanie przewodów wyjściowych (w wdrażanym rozwiązaniu były one zakończone wtykami RJ45 podłączanymi do panelu krosowego) do modułów ze względu na konieczność podpięcia przewodów od jednego urządzenia do kilku modułów wejściowych (np. od czujki p.poz z termometrem 1-wire były to moduły: wyjść mosfetowych, filtrowanych wejść optoizolowanych, wejściowo-wyjściowy)
  - należy rozważyć zastosowanie modułów podłączania kabli - z złączami śrubowymi do podłączania przewodów i/lub bezpośrednio dolutowanymi przewodami oraz złączami taśmowymi IDC do podłączania kilku (np. czterech lub ośmiu modułów centralki), dodatkowo w ramach tego złącza taśmowego należy przewidzieć dystrybucję dodatkowych zasilań 5V, 9V, 12V na potrzeby podłączanego modułu centralki
- pewną uciążliwością charakteryzuje się brak możliwości podpięcia poszczególnych elementów na modułach (np. wejść optoizolowanych) do wspólnych mas/zasilań bez lutowania dodatkowych przewodów na tych modułach
  - należy rozważyć bezpośrednie wprowadzenie tych linii na złącze sygnałów wejściowych i zapewnienie jakiegoś konfigurowalnego mechanizmu podłączania do szyn mas/zasilania na module przyłączeniowym
- problematyczne było podłączanie zewnętrznych magistral RS485
  - należy przewidzieć stosowne miejsca w module wejściowym i piny dla magistrali RS485 w złączu taśmowym wejść lub w złączu taśmowym magistrali
- problem stanowił też brak standaryzacji wielkości modułów i mechanizmu ich mocowania do wspólnej płyty
- można rozważyć realizację magistrali w oparciu o złącza krawędziowe na płytkach i płytę bazową zamiast obecnie stosowanego 20 pinowego złącza taśmowego
- można rozważyć wykorzystanie złącz RJ45 umieszczanych bezpośrednio na module do łączenia modułów umieszczanych wewnątrz szafki z centralką (np. sterowników oświetlenia) - w tym wypadku możliwe byłoby pominięcie w ich podłączaniu panelu krosowego przy zachowaniu jednolitego standardu podłączeń
- przyjęta koncepcja magistrali z rejestrami i buforami okazała się trudna do pogodzenia z stosowaną koncepcją linii sterowania jasnością gdzie na tym samym przewodzie mamy sygnał wyjściowy o możliwości regulacji i wejściowy zadający regulację - rozwiązanie to wymaga możliwości zmiany kierunku z zapamiętaniem wartości wyjściowej i rezystorami podciągającymi w wariancie wejściowym
  - możliwe jest przeznaczenie na to bezpośrednich wejść mikrokontrolera, jednak ogranicza to znacznie ilość takich linii
  - innym rozwiązaniem jest skonstruowanie specjalnego modułu magistralowego realizującego tą funkcjonalność
  - możliwe jest też rozwiązanie tego problemu przez zmianę koncepcji linii sterowania jasnością i rozdzielenie funkcji wejściowej od wyjściowej
  - jeszcze innym (i chyba najprostszym) jest wydelegowanie tej funkcjonalności do modułu mikrokontrolerowego

komunikującego się po RS485, wariant taki umożliwiłby także sygnalizację nastrojenia jasności (a nie tylko możliwego kierunku regulacji) z wykorzystaniem PWM
- interesująca koncepcją jest stosowanie rozproszonych systemów wejść wyjść opartych na projekcie manipulatora alarmowego, gdzie np. zamiast LCD podłączone byłyby dodatkowe linie na bezpośrednie przyciski, a sterowanie dowolną linią mogłoby odbywać się z klawiatury
- warto rozważyć przy realizacji w centralce tylko jednego portu UART rozdzielenie linii nadawczej i odbiorczej - zapewnia to niezależność transmisji od centralki w stosunku co do transmisji do niej - komunikacja z układami wykonawczymi centralki tylko za jej pośrednictwem

Aktualna wersje projektów (w szczególności dostosowane do wymogów projektowych opisanych w mikrokontrolery) znajdują się w dziale z projektami związanymi z inteligentnym domem: sterownik oświetlenia, centralka alarmowa. W pierwotnych założeniach przewijało się stosowanie magistral równoległych w niektórych modułach (np. sterownikach tranzystorowych) do zwiększania ilości wejść/wyjść. Pomysł ten nie został jednak zrealizowany ze względu na powyżej opisane trudności, zamiast tego zostały zastosowane moduły i2c podłączone do głównego sterowniki (w szczególności moduł oparty na mikrokontrolerze pełniącym funkcję slave'a i2c). W ogólności takie moduły można stosować jako (zaawansowane) rozszerzenia wejść wyjść mikrokontrolera (zaletą standaryzacja, niższy koszt na pin niż dedykowane układy i2c, oszczędność 3 pinów w stosunku co do wersji modbusowej).

Komunikacja poprzez RS485

Wraz z użyciem mikrokontrolerów pojawiła się idea zdalnego sterowania poprzez RS485 z wykorzystaniem wbudowanych układów UART. Wiązało się to z koniecznością przyjęcia/stworzenia jakiegoś protokołu do przekazywania komend. W pierwszych rozwiązaniach stosowany był prosty binarny format komunikatów - komunikaty były 4 bajtowe, a dwa pierwsze bity oznaczały typ bajtu w komunikacie (identyfikator urządzenia, identyfikator linii, komenda, odpowiedz). Implementacja tego protokołu zawarta jest w my_rs485_binary.c, a przykład użycia w sterownik_oswietlenia_v2.0.c. Rozwiązanie to ograniczało przestrzeń dostępnych adresów pod-adresów i komunikatów do 6 bitów, a także uniemożliwiało w prosty sposób przekazywania argumentu do komendy (co stało się potrzebne przy sterowaniu jasnością większej ilości linii). Niedogodnością była też niemożność prostego używania terminala (/dev/ttySx) do sterowania urządzeniami.

Kolejnym rozwiązaniem była komunikacja tekstowa ASCII w postaci: `@adres#subadres#komenda#parametr;` lub `@adres#subadres#komenda;`, gdzie adres, subadres, komenda, parametr są liczbami z zakresu 0-255 zapisanymi dziesiętne. Implementacja tego protokołu zawarta jest w my_rs485_ascii.c, a przykład użycia w sterownik_oswietlenia_v3.1.c.

W obu wypadkach całość dekodowania i (typowo) wykonywania komendy odbywała się w ramach przerwań związanych z otrzymaniem znaku z portu szeregowego. Niestety ze względu na konieczność stosowania znacznych opóźnień między-znakowych w komunikacji RS485, wynikłą z potrzeby dostosowania się do najwolniejszych modułów, które muszą mieć czas na analizę otrzymanego komunikatu system ulega stosunkowo łatwo zamulaniu. W związku z tym rozważana była zmiana koncepcji odbioru komunikatów z stosowanej analizy w trakcie odbioru na szybki odbiór analizę po jego zakończeniu i potwierdzenie wykonania po odebraniu komendy, ale wymagałoby to stosowania dodatkowego timeoutu na oczekiwaniu na potwierdzenie wykonania.

Kolejnym problemem była chęć umożliwienia nadawania komend (a nie tylko odpowiadania na nie) wielu urządzeniom. Wymusiła ona konieczność stosowania jakiejś formy arbitrażu. Zastosowane w pierwotnej wersji centralki alarmowej indywidualne linie zezwolenia na nadawanie nie zdały niestety egzaminu. Rozważane były m.in.:

- zastosowanie protokołu żetonowego z żetonem przekazywanym przy pomocy komendy
- system oparty na centralnym masterze odpytujący każde inne urządzenie o to czy ma coś do nadania
- system z stałym oknem czasowym na nadawanie (a nie rozpoczęcie nadawania) zadawanym indywidualna linią zezwolenia
- jakieś usprawnienia obecnego systemu niedopuszczające do dyskryminacji i zagładzania urządzeń

Komunikacja binarna i switche - koncepcje niezrealizowane

W efekcie opisanych powyżej problemów z opóźnieniami w transmisji zrodził się pomysł **wykorzystania echa (odsyłania odebranego bajtu) do** weryfikacji odbioru komunikatu, jej wykonania oraz odbioru odpowiedzi, a przede wszystkim do **sterowania prędkością nadawania** (czekamy na potwierdzenie odbioru i przetworzenia poprzedniego bajtu). Równocześnie nastąpił powrót do komunikatów binarnych (mniejszy narzut na transmisję oraz prostsza analiza komendy). Komunikat miał składać się z 6 bajtów:

- pierwszy bajt - adres urządzenia (0-32) 0b000x xxxx
- drugi bajt - komenda (0-32) 0b001x xxxx
- 3+4. dana 1 (subadres)
  - młodsza polowa bajtu: 0b0100
  - starsza polowa bajtu: 0b0101
- 5+6. dana 2 (wartość)
  - młodsza polowa bajtu: 0b0110
  - starsza polowa bajtu: 0b0111

Kodowanie typu bajtu w każdym z bajtów poprawia czytelność protokołu, ułatwia interpretację komunikatu (a także umożliwia detekcję zagubionego bajtu), a także restart transmisji (znacznik pierwszego bajtu).

Algorytm odbioru komunikatu zapisany w pseudo-c:

```
uint8_t rd, cmd, d1, d2;
uint8_t mask = MY_ADDRESS;
#ifdef USE_INTERPUT
procedura_obslugi przerwania() {
        rd = UDR;
        // zakladamy ze nadawca czeka na potwierdzenie wiec nie bedzie nadawl gdy przetwarzamy
        odblokuj_przerwania();
#else
while (1) {
        rd = wait_for_data();
#endif
        if ((rd & mask) == rd) {
                switch (mask) {
                        /// adres
                        case MY_ADDRESS:
                                mask = 0x20 | 0x1f;
                                cmd = 0x00;
                                break;
                        /// komenda
                        case 0x20 | 0x1f:
                                mask = 0x40 | 0x0f;
                                cmd = rd &0x1f;
                                break;
                        /// dana 1
                        case 0x40 | 0x0f:
                                mask = 0x40 | 0x10 | 0x0f;
                                d1 = rd &0x0f;
                                break;
                        case 0x40 | 0x10 | 0x0f:
                                mask = 0x60 | 0x0f;
                                d1 = d1 | ( ( rd << 4 ) & 0x0f);
                                break;
                                cmd = execute_cmd1(cmd, d1, &d2); // funkcja wykonuje komendę która ma odpowiedź i zapisuje ją w d2,
jeżeli się udało (była taka komenda zwraz 0xff, w przeciwnym razie cmd)
                        /// dana 2
                        case 0x60 | 0x0f:
                                mask = 0x60 | 0x10 | 0x0f;
                                if (cmd == 0xff) {
                                        send( 0x60 | (d2 &0xf) );
                                } else {
                                        d2 = rd &0x0f;
                                }
                                break;
                        case 0x60 | 0x10 | 0x0f:
                                mask = MY_ADDRESS;
                                if (cmd == 0xff) {
                                        send( 0x60 | ((d2<<4) &0xf) );
                                } else {
                                        d2 = d2 | ( ( rd << 4 ) & 0x0f);
                                        execute_cmd2(cmd, d1, d2); // funkcja wykonuje komendę która nie ma odpowiedzi
                                }
                                break;
                }
                if (cmd != 0xff)
                        send(rd);
        } else {
                mask = MY_ADDRESS;
        }
}
```

Algorytm nadawania komunikatu zapisany w pseudo-c:

```
send_msg(uint8_t addr, uint8_t cmd, uint8_t d1, uint8_t d2) {
        uint8_t buf[6], i, ret, reply = 0;
        buf[0]=addr;
        buf[1]=0x20 | cmd;
        buf[2]=0x40 | (d1 & 0x0f);
        buf[3]=0x40 | 0x10 | ((d1<<4) & 0x0f);
        buf[4]=0x60 | (d2 & 0x0f);
        buf[5]=0x60 | 0x10 | ((d2<<4) & 0x0f);
        for (i=0; i<6; i++);
                send(buf[i]);
                ret = wait_for_data_with_timeout();
                if (i == 4 && d2 == 0xff) {
                        if (ret & 0xf0 ! = 0x60) {
                                // błąd transmisji
                                return i - 20;
                        }
                        reply = ret & 0xf;
                } else if (i == 5 && d2 == 0xff) {
                        if (ret & 0xf0 ! = 0x70) {
                                // błąd transmisji
                                return i - 20;
                        }
                        reply = reply | ( (ret<<4) & 0xf );
                } else if (ret != d[i]) {
                        // błąd transmisji
                        return i - 10;
                }
        }
```

```
        return reply;
}
```

Niestety nie rozwiązywało to problemu rywalizacji o dostęp do medium transmisyjnego, rozważano w tej kwestii:

- użycie `0b111 | ADRES` jako tokenu uprawniającego do nadawania
- użycie linii open-collector informacja że właśnie nadaje
- rezygnacja z magistrali na rzecz łącz punkt(switch)-punkt(slave):
  - gwiazda z hubem/switchem (mniej lub bardziej inteligentnym) wybierającym (w oparciu o zgłoszenia na dedykowanej linii) tego kto aktualnie nadaje do wszystkich pozostałych (ew. może także rozpoznawać adresy i nadawać tylko na właściwe porty), przykładowe realizacje takiego huba: w oparciu o sterowanie z mikrokontrolera i bez mikrokontrolera sterującego
  - wariant rs_ttl full-duplex: nadawanie i odbiór poprzez bufor (74hc125) do wskazanego z użyciem dekodera (74hc138, zerowe wyjście nie używane) slave'a
  - wariant rs_485 full-duplex: nadawanie i odbiór poprzez trancriver (sn75176) do wskazanego z użyciem dekodera (74hc138, zerowe wyjście nie używane) slave'a
  - wariant rs_485 half-duplex: nadawanie do wszystkich (osobne sn75176) słuchanie tylko z wybranego z użyciem dekodera (74hc138, zerowe wyjście nie używane) slave'a; nadajniki w sn75176 (DE) sterowane z 74hc238 (zerowe wyjście nie używane), odbiorniki (~RE) sterowane z 74hc138 (zerowe wyjście nie używane), wejście E i ~E dekoderów podłączone do linii sterującej zapis/odczyt
  - linia sygnalizująca chęć nadawania wprowadzona na 8in encoder (74hc148, zerowe wejście podłączone do masy) i do mastera decydującego komu dać prawo nadawania uC
  - sygnał zgody na nadawanie (włączenia danego odbiornika) przekazywany do urządzenia slave w formie linii open-collector (hc7407)
    - stan niski nie ustawiony przez slave informuje o tym że można nadawać
    - stan niski nie ustawiony przez mastera oznacza prośbę o transmisję odpowiedzi (aby móc wykryć zgodę na transmisję odpowiedzi musi on być przerywany celem odczytu stanu linii, zatem nie ma potrzeby osobnej zgody na transmisje odpowiedzi)
    - linię taka można uzyskać na Atmega8 (nadawanie 1: PORTx = 0; DDRx = 0; nadawanie 0: PORTx = 0; DDRx = 1;)
  - dla każdego slave (1-7) switch ma określoną wartość dopuszczalnego czasu trwania zgody na nadawanie (timeout)

```
for (i = 1; i < 8; ++i) {
        mask = 0x01 << i;
        if (read_oczekujacy_mask() & mask) {
                wykryto_koniec_ramki=0;                                 /// opcjonalnie możemy śledzić koniec ramki i limitować
ilość ramek od urządzenia ....
                start_timer(timeout[i]);
                do {
                        odbieraj_dane(i);                                   /// transmituj zapytanie
                        while(! (reply = read_oczekujacy_reply_mask()) ); /// czekaj na zgłoszenie chęci transmisji
odpowiedzi
                        odbieraj_dane(reply);                               /// transmituj odpowiedź
                        while(read_oczekujacy_reply_mask() != 0);         /// czekaj na koniec transmisji odpowiedzi
                } while (!timer_timeout && !wykryto_koniec_ramki && read_oczekujacy_mask() & mask);
                stop_timer(timeout[i]);
        }
}
```