Centrala alarmowa z powiadomieniem GSM

AVT-526

PROJEKT Z OKŁADKI

Centralki alarmowe są bardzo często wyposażane w różnego rodzaju sygnalizację włamania. Najprostszym sposobem sygnalizacji jest włączenie sygnału dźwiękowego lub optycznego. Centralki bardziej zaawansowane konstrukcyjnie potrafią informować o włamaniu przez Internet lub telefon. Prezentowany w tym artykule system alarmowy jest amatorskim rozwiązaniem, jednak może przesłać informację tekstową o włamaniu za pomocą modułu GSM, czyli popularny SMS. ***Rekomendacje:*** *nowoczesne i bardzo skuteczne rozwiązanie zdalnej ochrony mienia, dzięki któremu informacja o włamaniu dotrze do Ciebie w dowolnym miejscu świata.*

W egzemplarzu modelowym wykorzystałem modem GSM firmy Wavecom (**fot. 1**), jednak w jego miejsce można użyć dowolnego telefonu komórkowego. Być może wykorzystanie innego modemu będzie wymagało rozwiązania problemu zasilania oraz sposobu kodowania SMS, jednak jest to możliwe i wykonalne w warunkach warsztatu elektronika-amatora.

Prezentowana centralka alarmowa jest jednym z urządzeń elektronicznych, których podstawowa wartość skupia się na odpowiednio napisanym oprogramowaniu. Układ elektryczny jest prosty (**rys. 2**), a płytka przypomina nieco płytkę prototypową przeznaczoną do projektowania układów z mikrokontrolerami. Jednostką sterującą jest mikrokontroler AT90S2313 taktowany wewnętrznym generatorem o częstotliwości stabilizowanej kwarcem 7,3728 MHz. Mikrokontroler ten ma 15 linii wejścia/wyjścia. Dwie z nich wykorzystywane są przez UART (RXD - 2, TXD - 3), pozostałe trzynaście steruje pracą układów peryferyjnych lub służy do komunikacji mikrokontrolera z otoczeniem. Port B ustawiany jest przez program jako wyjściowy z załączonymi rezystorami *pull-up* (DDRB=0xFF). Tryb ten nie ulega zmianie podczas pracy programu, mimo iż czasami przez ten port wyprowadzane są dane, a czasami służy on do ich odczytu. Jest to pewna szczególna cecha mikrokontrolera AVR, wrócimy jeszcze do niej w artykule. Podobnie jest w przypadku portu D, chociaż tutaj część linii jest liniami wyjściowymi (TXD, DIR), a część wejściowymi (RXD i bity 2 do 5). Raz ustawiony tryb pracy nie jest zmieniany podczas działania centralki alarmowej.

Jak wspomniałem, mikrokontroler AT90S2313 posiada 15 linii portów wejścia-wyjścia. Potrzeby układu alarmu są jednak znacznie większe. Posiada on 8 wejść przeznaczonych dla czujników, a wyświetlacz i klawiatura wymagają użycia odpowiednio co najmniej 6 i 7 wyprowadzeń portów. To tylko podstawowe układy otoczenia. Dodajmy do tego jeszcze co najmniej 1 linię do sterowania układem sygnalizacji alarmu i otrzymamy liczbę 22 wyprowadzeń portów niezbędnych do prawidłowego funkcjonowania układu alarmu. Konieczna więc była albo zmiana mikrokontrolera na posiadający większą liczbę portów, albo zastosowanie dodatkowych ukła-

dów scalonych. Zdecydowałem się na to drugie rozwiązanie.

Sygnał z czujników alarmowych jest podawany na doprowadzenia diody transoptora. Transoptor zapewnia izolację galwaniczną czujnika od centralki oraz eliminuje część niskonapięciowych zakłóceń, które mogą pojawić się na linii doprowadzającej sygnał z czujnika. Użyłem 2 elementów firmy SHARP o symbolu PC849 (odpowiednik funkcjonalny produkowany jest przez firmę LITEON pod oznaczeniem LTV849). W obudowie zawarto 4 transoptory składające się z fototranzystora i diody. Szeregowo połączone z diodami rezystory R1...R8 należy dobrać do parametrów czujników. Stosowane przeze mnie rezystory o wartości 1 kΩ dobrane zostały dla napięcia sterującego o wartości 9...12 V.

Osiem wejść czujników dołączonych jest do układu multipleksera, który w zależności od stanu logicznego sygnału przyłożonego na wejście A/B (wyprowadzenie 1) podaje na wyjścia 1Y...4Y odpowiednio stany logiczne z wejść 1A...4A lub 1B...4B. Umożliwia to 4-bitowemu portowi mikrokontrolera odczyt 8-bitowych danych w dwóch „porcjach" po 4 bity. Wyborem 4 wejść, z których odczytywany jest stan, steruje wyprowadzenie 6 portu D mikrokontrolera.

Niektóre z portów mikrokontrolera pełnią podwójną rolę. Oprócz tego, że są wyjściowymi, np. dla modułu wyświetlacza LCD, to są również wejściowymi dla wierszy klawiatury.

Zacznijmy opis od wyświetlacza. W swoim układzie użyłem modułu LCD (2 linie po 20 znaków w każdej). Wyświetlacz pracuje w trybie z interfejsem 4-bitowym z wykorzystaniem linii portu PB0...PB3 jako linii danych, PB4 jako linii wyboru rejestru RS i linii PB7 jako ENABLE. Wszystkie linie portów oprócz ENABLE są wykorzystywane podwójnie: jako sterujące wyświetlaczem oraz pracą klawiatury. Wykorzystywany jest fakt, że podczas trwania poziomu niskiego na linii ENABLE do rejestrów wyświetlacza nie są zapisywane żadne dane. Umożliwia to zmiany stanów pozostałych linii bez zakłócania pracy modułu.

Fot. 1. Modem GSM firmy Wavecom

Typowo, na wyprowadzenie 3 złącza wyświetlacza jest podane napięcie z suwaka potencjometru służące do regulacji kontrastu, a pomiędzy 15 i 16 doprowadzone jest napięcie dla podświetlenia tła (około 4,3 V). Układ można uprościć, zwierając napięcie regulacji kontrastu do masy. Dla większości wyświetlaczy LCD jest to wystarczające.

Klawiatura składa się z 12 przycisków i została zbudowana jako matryca 3 kolumn i 4 wierszy. Trzy bity portu B (4, 5 i 6) pracują jako wyjściowe, cztery pozostałe jako wejściowe. Wszystkie linie mają załączone wewnętrznie rezystory *pull-up*. Wyjątkowo bity 0 i 1 wymagają zewnętrznych rezystorów podciągających. Kierunek pracy portu nie jest zmieniany przez program sterujący. Linie pracują zawsze jako wyjściowe, a w razie potrzeby ich stan sprawdzany jest przez odczyt rejestru PIN. W architekturze mikrokontrolera AVR rejestr PIN zawiera adres, przy odczycie którego wyprowadzenie portu załączane jest przez specjalny bufor trójstanowy do wewnętrznej magistrali danych mikrokontrolera.

Rolę układu dopasowującego poziomy napięć interfejsu UART mikrokontrolera do RS232 pełni układ MAX232. Wykorzystane są tylko 2 drivery znajdujące się w strukturze układu. Sygnał interfejsu doprowadzony jest na męskie, 9-stykowe złącze DB.

Tab. 1. Opis funkcji realizowanych przez wyprowadzenia mikrokontrolera AT90S2313 w centrali alarmowej

Wyprowadzenie mikrokontrolera			Opis funkcji
Nazwa	Numer (obudowa DIL20)	Oznaczenie sygnału na schemacie	
PB0/AIN0	12	DB4	Linia danych DB4 modułu LCD, wiersz 2□klawiatury
PB1/AIN1	13	DB5	Linia danych DB5 modułu LCD, wiersz 3□klawiatury
PB2	14	DB6	Linia danych DB6 modułu LCD, wiersz 4□klawiatury
PB3/OC1	15	DB7	Linia danych DB7 modułu LCD, wiersz 1□klawiatury
PB4	16	RS	Linia RS (Register Select) modułu LCD, kolumna 1□klawiatury
PB5/MOSI	17	COL	Kolumna 2□klawiatury
PB6/MISO	18	LED	Linia załączająca przekaźnik i□diodę LED, kolumna 3□klawiatury
PB7/SCK	19	EN	Linia ENABLE modułu LCD (sterująca wprowadzaniem danych)
PD0/RXD	2	RXD	Linia odbiornika interfejsu UART mikrokontrolera
PD1/TXD	3	TXD	Linia nadajnika interfejsu UART mikrokontrolera
PD2/INT0	6	-	Wejście sygnału z□czujki zewnętrznej (X1 i□X5)
PD3/INT1	7	-	Wejście sygnału z□czujki zewnętrznej (X2 i□X6)
PD4/T0	8	-	Wejście sygnału z□czujki zewnętrznej (X3 i□X7)
PD5/T1	9	-	Wejście sygnału z□czujki zewnętrznej (X4 i□X8)
PD6/ICP	11	DIR	Sterowanie wyborem wejść multipleksera (U2)

Układu MAX można nie wlutowywać lub nie wykorzystywać. Zworki JP1 i JP2 znajdujące się na płytce umożliwiają odłączenie sygnałów RXD i TXD mikrokontrolera od układu MAX i doprowadzenie ich do inwerterów 74HCT14. Jest to układ dopasowujący poziomy do wymagań telefonu GSM (Siemens C35). Rezystory R20 i R21, tworzące dzielnik napięcia, są tak dobrane, aby napięcie wyjściowe nie przekraczało 3,3 V na poziomie wysokim. Bramka 74HCT14 pracująca jako odbiornik posiada w strukturze przerzutnik Schmitta, który powoduje, że napięcie powyżej 2,4 V traktowane jest jako wysokie. Zaznaczam jednak, że alarm testowany był wyłącznie z modemami firmy Siemens (M20T) oraz Wavecom (WMOD2), a interfejs dla telefonu GSM nie był wykorzystywany. Złącze dla „zwykłego" telefonu GSM oznaczone jest jako X14, a dla modemu GSM - X12.

Bramki układu inwertera pracują również jako układy wyjściowe sterujące pracą przekaźnika oraz diody LED.

Układ jest zasilany napięciem 5 V z umieszczonego na płytce układu stabilizatora 7805 (U3). Do poprawnej pracy wymaga on podania co najmniej 7 V na zaciski złącza X10 (uwaga na polaryzację!). Kondensatory C9...C12 pełnią rolę filtrowania napięcia zasilającego. Kondensatory C10 i C11 powinny być umieszczone jak najbliżej wyprowadzeń stabilizatora. Rezystor R11 i kondensator C4 to obwód RC wytwarzający po włączeniu sygnał zerujący dla mikrokontrolera.

Elementem wykonawczym alarmu jest przekaźnik RL1. Na doprowadzeniach jego cewki znajduje się dioda D3 mająca za zadania eliminowanie przepięć powstających w czasie załączania i wyłączania cewki. Jak wspomniałem, rolę układu dopasowującego pełnią bramki inwertera połączone równolegle w celu zapewnienia większej wydajności prądowej. Przekaźnik sterowany jest wspólnie z diodą LED. Załączenie przekaźnika jest sygnalizowane zaświeceniem tej diody.

W pewnych momentach pracy układu dioda LED jest wykorzystywana również do sygnalizacji. Wówczas jednak jej błyski są na tyle krótkie, że nie powodują załączenia przekaźnika. Również w czasie przeglądania wierszy klawiatury starałem się dobrać czasy na tyle krótkie, aby nie powodować załączania urządzeń zewnętrznych.

Opis programu sterującego

Początkowo zamierzałem napisać program sterujący pracą alarmu w języku Bascom lub GCC. Okazało się jednak, że mikrokontroler AT90S2313 ze swoimi 2 kB pamięci programu jest zbyt „skromny" jak na wymagania kompilatora języka wysokiego poziomu.

Zupełnie zaskoczył mnie kompilator GCC. Przeniosłem do niego bibliotekę funkcji, napisaną co prawda dla 8051, lecz po drobnych modyfikacjach kompilator GCC przetworzył źródło na kod wynikowy bez większego problemu. Jednak rozmiar tego kodu był dla mnie zupełnym zaskoczeniem! Wykonałem wyłącznie procedury obsługi LCD w języku C i te po kompilacji zajęły 3 kB! Dla porównania analogiczny funkcjonalnie program wykonany w Bascom AVR zajął jedynie ok. 500 bajtów, podobnie zresztą jak w RC-51. Być może nie umiałem włączyć jakiejś opcji optymalizacji.

Tak więc 2 kB to za mało pamięci do wykonania tego typu aplikacji w języku wysokiego poziomu. Musiałem napisać program sterujący w asemblerze. Przy uruchamianiu i testowaniu programu posługiwałem się AVR Studio 4 firmy Atmel.

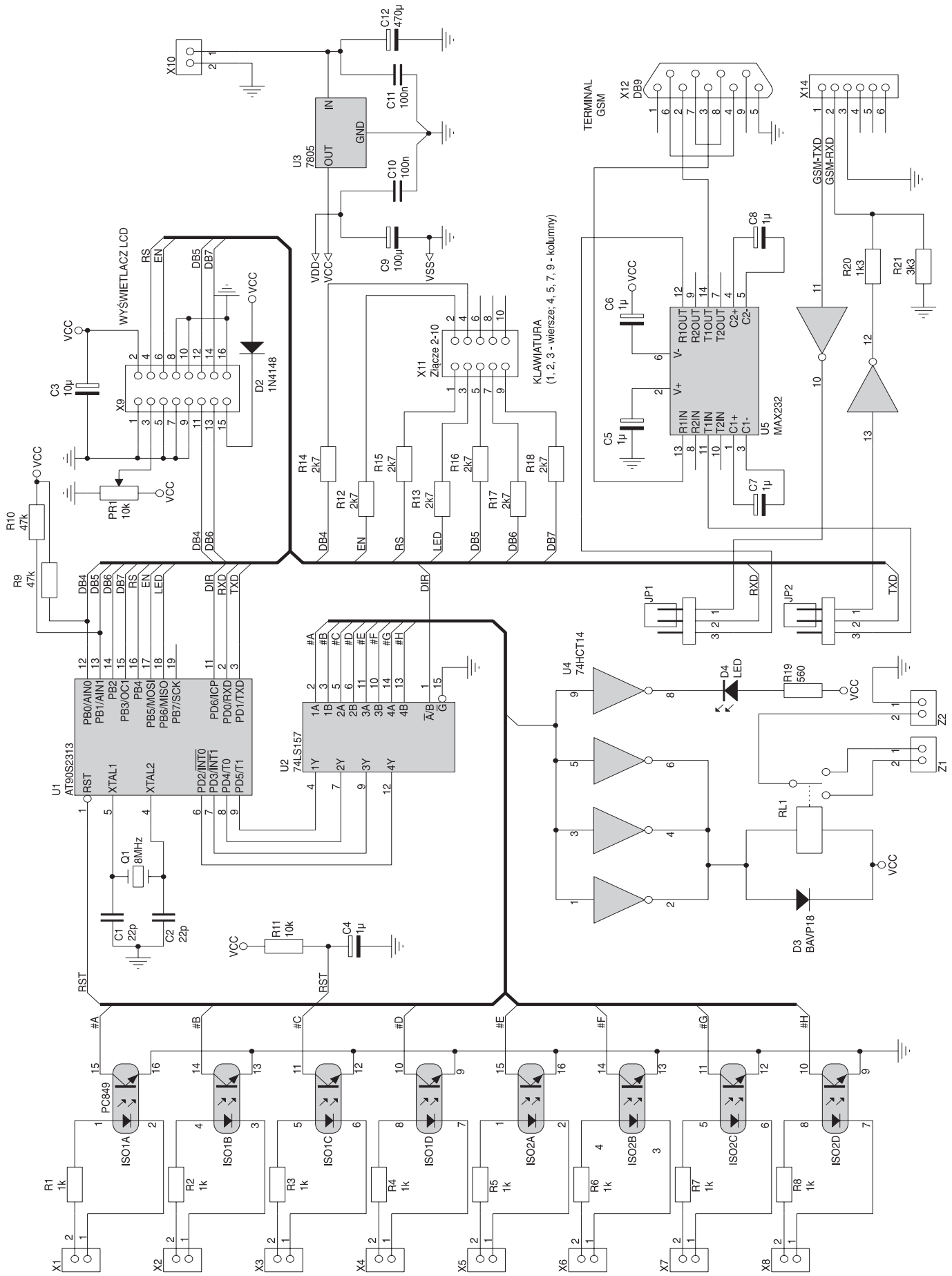
Aby omówić działanie programu, muszę wyjaśnić kilka podstawowych reguł, którymi kieruję się przy pisaniu tego rodzaju aplikacji. Dotyczą one głównie stałych i zmiennych oraz sposobu ich interpretacji przez funkcje. Zacznijmy od liczb. Zgodnie z notacją przyjętą przez firmę Atmel, każdą liczbę rozpoczyna bajt mniej znaczący, a kończy bardziej znaczący. Jeśli liczba jest 4-bajtową umieszczoną w rejestrach R0 - R01 - R02 - R03, to najmniej znaczący bajt jest zawarty w rejestrze R0, a najbardziej znaczący w R03.

Każdy łańcuch tekstowy wysyłany na wyświetlacz zawiera tylko kody znaków ASCII o wartościach większych od 0x20 (32 dziesiętnie, kod znaku odstępu), a kończy się znakiem o kodzie 0x00. Podobnie jest z ciągami znaków wysyłanych do modemu, ale podlegających przetwarzaniu przed ich wysłaniem. Na przykład polecenie zapisujące kod PIN zawiera część stałą *AT+CPIN=* oraz zmienny - zależny od karty SIM - poufny kod PIN. Inaczej jest z poleceniami przesyłanymi wprost do modemu. Te kończy sekwencja kodów CR (powrót kursora do początku linii, 0x0A) oraz LF (nowa linia, 0x0C).

Reasumując - każdy tekst wyświetlany lub przetwarzany kończy kod 0x00, a każdy tekst przesyłany do modemu kończą kody CR-LF. Ten sposób zapamiętywania danych stosowany jest konsekwentnie w całym programie z wyjątkiem przechowywania kodu załączenia i wyłączenia alarmu oraz kodu PIN.

Program główny jest bardzo krótki. Na początku ustawiane są tryby pracy portów B i D. Do rejestru DDRB wpisywana jest wartość 0xFF, co oznacza, że wszystkie wyjścia mają załączone rezystory pull-up i pracują jako wyjściowe. Do rejestru DDRD wpisywana jest wartość 0xC0, która ustawia bity 6 i 7 portu D jako wyjściowe, a pozostałe jako wejściowe. Linia TXD portu C (bit 1) pracuje jako wyjściowa, jednak wpisanie do rejestru DDRD „1" na pozycję odpowiednią dla tego bitu powoduje, że pracuje ona jako zwykły port wyjściowy i traci swoją funkcję jako TXD. Aby linia ta pracowała poprawnie jako wyjście interfejsu UART, należy do DDRD na pozycji 1 wpisać „0" lub pozostawić stan taki, jaki jest po załączeniu zasilania.

Następne linie programu ustalają częstotliwość, z jaką wywoływane będą przerwania Timerów 0 i 1. Pierwszy to licznik 8-bitowy generujący przerwanie po zliczeniu 255 impulsów. Kolejny, 256 impuls zegarowy ustawia flagę przepełnienia OVF0 i powoduje przejście do obsługi przerwania. Przy częstotliwości rezonatora kwarcowego 7,3728 MHz, preskalerze ustawionym na wartość 1024 oraz początkowej wartości licznika 0 (licznik Timera 0 liczy w górę), przerwania będą wywoływane



Rys. 2. Schemat elektryczny centrali

z częstotliwością 7372800: 1024: 256 = 28,125 Hz. Timer 1 również ma preskaler ustawiony na wartość 1024, jednak do rejestrów wpisywana jest wartość inicjująca, od której Timer zlicza impulsy zegarowe w górę, aż do przepełnienia. Timer 1 w odróżnieniu od Timera 0 jest 16-bitowy, toteż liczy do wartości 65535. Impuls numer 65536 powoduje przepełnienie, ustawienie flagi OVF1 i przejście do procedury obsługi przerwania. Tu przerwanie wywoływane jest z częstotliwością 7372800: 1024: (65535 - 58335) = 1 Hz. Przerwania są zablokowane po uruchomieniu mikrokontrolera, jednak pozostawiony („na wszelki wypadek") rozkaz CLI wyłącza je i blokuje aż do momentu, gdy konieczne będzie ich użycie. Zostawmy na razie procedury obsługi przerwań - będzie jeszcze okazja do nich wrócić.

Od etykiety *STATUS_CHECK* rozpoczyna się część programu mająca za zadanie rozpoznanie trybu, w którym znajdował się mikrokontroler przed wyłączeniem napięcia zasilania. Wywoływana zostaje funkcja odczytująca bajt spod adresu 0x00 w pamięci EEPROM. Wartość tego bajtu odpowiada statusowi urządzenia. Jeśli komórka ta zawiera wartość różną od 0xF0 lub 0x0F, to program uznaje, że jest to pierwsze załączenie alarmu i zażąda wykonania wszystkich niezbędnych nastaw, to jest: podania kodu załączenia i wyłączenia, podania kodu PIN dla używanej karty SIM, wprowadzenia numeru telefonu oraz opisów dla lokalizacji alarmu i wejść czujników. Wybrana metoda sprawdzenia statusu urządzenia jest o tyle pewna, że każdy zapis programu do pamięci Flash mikrokontrolera AT90S2313 powoduje usunięcie zawartości EEPROM (cały EEPROM zostaje zapisany wartością 0xFF). Po wprowadzeniu nastaw do komórki *STATUS* zostaje zapisana wartość odpowiednia dla stanu wyłączenia alarmu.

Poświęćmy teraz kilka chwil wysyłanym do modemu rozkazom i ich znaczeniu. Interfejs UART pracuje z szybkością 9600 bitów na sekundę. Identyczną prędkość powinniśmy ustawić w posiadanym telefonie czy modemie. Aby dokonać niezbędnych nastaw, należy za pomocą dowolnego programu terminala nawiązać połączenie z modemem i wydać polecenie *AT+IPR=9600;&W*. Ustawi ono i zapisze w pamięci konfiguracji modemu parametr prędkości transmisji. W przypadku użycia telefonu GSM może się okazać, że nie zaakceptuje powyższej komendy. Wówczas prawdopodobnie łatwiej będzie nam zmienić prędkość z jaką pracuje UART mikrokontrolera aniżeli telefonu. Można to zrobić, poprawiając funkcję *uart_init*, a konkretnie wartość wpisywaną do rejestru UBRR. Należy dobrać ją w zależności od używanego rezonatora i wymaganej prędkości transmisji. Gotowe przykłady można znaleźć w notach aplikacyjnych na stronie producenta *http://www.atmel.com* lub wyliczyć samodzielnie na podstawie wzorów znajdujących się w kartach katalogowych mikrokontrolerów AVR. Gdy sprawdzimy, że modem (telefon) gotowy jest do pracy i komunikuje się poprawnie używając prędkości 9600 bd, możemy podłączyć go do naszego układu.

Program sterujący wysyła jako pierwszą komendę *ATZ*. Powoduje ona ustawienie domyślnego trybu pracy - przywrócone zostają również wszystkie te parametry, które nie zostały zapamiętane na stałe jako domyślne. Następny rozkaz wprowadza numer PIN właściwy dla karty, na przykład: *AT+CPIN=9889*. Jako kolejne wysyłane są polecenia *ATE0* wyłączające tzw. echo komendy oraz *AT+CPMS=SM* ustawiające miejsce przechowywania wiadomości SMS na kartę SIM. Jednocześnie z przyjmowaniem tych poleceń modem włącza się do sieci. W przypadku modemu Wavecom gotowość do pracy sygnalizowana jest przez migoczącą z częstotliwością około 1 Hz diodę LED. W przypadku Siemens MT20 jest inaczej - dioda LED świeci się stale.

Po każdym poleceniu wysłanym do modemu wywoływana jest funkcja *gets* odbierająca ciąg znaków z UART. Oczekiwane jest, że modem odpowie, wysyłając (co najmniej) kody CR i LF. W przypadku braku odpowiedzi od modemu program przestanie pracować już w fazie inicjacji. Można w ten sposób łatwo stwierdzić, że brak jest poprawnej łączności z modemem. Niestety, nie wykonałem w programie detekcji odłączenia modemu. W przypadku, gdy modem zostanie odłączony tuż przed wysłaniem SMS z powiadomieniem o załączeniu alarmu, alarm zostanie włączony, a następnie program będzie oczekiwał na połączenie z modemem i odpowiedź na przesłane polecenia. Jeśli jej nie otrzyma - zawiesi się sprawdzając w nieskończoność stan flagi gotowości do odbioru znaku (bajtu) z UART.

Od etykiety *main* rozpoczyna się główna pętla programu. Na jej początku ponownie odczytywany jest bajt statusu i w zależności od jego wartości podejmowana jest odpowiednia akcja. Polega ona bądź to na wywołaniu procedury obsługi alarmu w stanie czuwania, bądź to obsługi alarmu w stanie wyłączenia.

Stan wyłączenia

Za jego obsługę odpowiedzialna jest funkcja *alarm_off*. Po jej wywołaniu czyszczony jest ekran wyświetlacza LCD, a następnie wyświetlona zostaje informacja z pamięci ROM umieszczona w niej pod nazwami symbolicznymi *whenoff* oraz *set_1*. Pierwsza umieszczona zostaje w pierwszej linii ekranu - ja zadeklarowałem ją jako *ALARM WYŁĄCZONY*, ta druga umieszczona w drugiej linii LCD. Zadeklarowałem ją jako *Kod wyłączenia?* Później zmiennej *visible* nadana zostaje wartość *UCANTSEE* a zmiennej *sensors* wartość *SENSOROFF*. Po tych operacjach wywoływana jest funkcja odczytu kodu wprowadzanego za pomocą klawiatury. Kod jest zawsze 4-cyfrowy. Wynik funkcji zwracany jest w rejestrach o nazwach symbolicznych *cchar1* do *cchar4*. Uwaga! Kod załączenia musi być inny niż „0000". Jest on zarezerwowany do wprowadzenia urządzenia w tryb serwisowy!

Funkcja *compare_codes* porównuje 4 bajty pobrane z pamięci EEPROM spod adresu podanego w rejestrze *eeaddr* z odczytanymi z klawiatury. Jeśli wprowadzony kod jest identyczny z zapamięta-

nym, do komórki zawierającej bajt statusu w EEPROM zapisana zostaje wartość informująca program o tym, że alarm jest załączony i następuje powrót do programu głównego. Teraz na podstawie zawartości komórki *status* pamięci EEPROM wywołana zostaje funkcja obsługująca stan załączenia alarmu.

Stan załączenia

Wszystkie funkcje związane ze stanem czuwania alarmu umieszczono w podprogramie *alarm_on*. Po wejściu do podprogramu załączane jest przerwanie Timera 1. Fragment programu zawierający kod jego obsługi umieściłem na **list. 1**. Procedura ta ma do wykonania dwa podstawowe zadania. Ma zwiększyć wartość zmiennej - licznika *goon* oraz odświeżyć zawartość rejestrów licznika Timera 1. Do tych dwóch zadań dodałem jeszcze trzecie - krótki, trwający około 1 milisekundy, błysk diody LED. Sygnalizuje ona w ten sposób, że odmierzany jest czas do załączenia. Pętla rozpoczynająca się od etykiety *al_on_wait* trwa dokładnie tyle czasu odmierzanego w sekundach, ile podano w stałej *DELAYTIME*. Jako typową wartość zadeklarowałem 30 sekund. Jest to czas, przed upływem którego osoba załączająca alarm musi się oddalić z obszaru zadziałania czujników.

Po upływie tego czasu rozkaz CLI wyłącza obsługę przerwań a przerwanie pochodzące od Timera 1 jest blokowane. Teraz wyświetlony zostaje komunikat o załączeniu alarmu (*whenon* i *set_2*), a następnie do rejestru kontrolnego przerwań (TIMSK) wprowadzana jest nastawa załączająca przerwanie na skutek przepełnienia Timera 0. Jest ono odpowiedzialne za odpytywanie stanu czujników (sensorów) alarmu. Rozkazy składające się na procedurę obsługi przerwania umieściłem na **list. 2**. Ze względu na pozorną złożoność poświęcimy jej trochę uwagi.

Rozkaz *push temp* przechowuje wartość rejestru *temp* na stosie. Następnie zerowany jest bit 6 portu D oznaczony na schemacie jako DIR. Wartość „0" tego bitu oznacza, że odczytane zostaną czujniki podłączone do wejść 1A, 2A, 3A i 4A mikrokontrolera. Tworzą one

List. 1. Procedura obsługi przerwania Timera 1

```
;-------------------------------
;obsługa przerwania timera 1
;odmierzanie czasu opóźnienia do zadziałania alarmu
;-------------------------------
tim1_irq:
   sbi portb,ALARMOUT      ;zalaczenie diody LED (migotanie w czasie „ALARMDELAY")
   push temp               ;przechowanie zawartosci „temp" na stosie
   ldi temp,TIM1RLDH
   out TCNT1H,temp
   ldi temp,TIM1RLDL
   out TCNT1L,temp         ;odswiezenie zawartosci Timer'a 1
   inc goon                ;zwiekszenie licznika „goon"
   ldi temp,0xFF
tim1_loop:
   rcall delay1ms          ;opóznienie czasowe aby migotanie LED bylo widoczne
   pop temp
   cbi portb,ALARMOUT      ;wylaczenie diody LED
   reti
```

mniej znaczącą połówkę bajtu stanu czujników. Jest ona odczytywana z rejestru PIND, przesunięta dwukrotnie w prawo (dwa polecenia *asr temp*) i zostaje zapamiętana w zmiennej *sensors*. Następnie, po ustawieniu poziomu wysokiego na wyprowadzeniu PORTD.6 pobrana zostaje bardziej znacząca połowa bajtu stanu czujników, przesunięta dwukrotnie w prawo i zsumowana z mniej znaczącą. Wynik sumowania zostaje zapamiętany w zmiennej *sensors*. Po odtworzeniu wartości *temp* procedura kończy się. Inne funkcje sprawdzają odczytany stan czujników i podejmują odpowiednią akcję. W przypadku stwierdzenia na jednym z wejść informacyjnych mikrokontrolera (PD2...PD5) poziomu niskiego obsługa przerwania Timera 0 kończy się - zostaje ono zablokowane. Dalsze przeglądanie czujników nie jest potrzebne. Ponownie zerowany jest licznik czasu *goon* i załączone zostaje przerwanie Timera 0, tym razem odmierzając czas upływający od momentu zadziałania czujnika do momentu załączenia alarmu. Jest to także czas podany w sekundach o wartości *DELAYTIME*. Opóźnienie zostało wprowadzone po to, aby użytkownik miał możliwość wprowadzenia kodu wyłączającego alarm.

Procedura *alarm_off* sprawdza kod wyłączenia wprowadzany przez użytkownika oraz kończy pracę, ustawiając bajt statusu urządzenia w EEPROM i powracając do głównej pętli programu w przypadku zgodności wprowadzonego kodu z zapisanym w EEPROM. Decyzję o podejmowanej przez centralkę akcji podejmuje główna pętla programu na podstawie zawartości komórki pamięci zawierającej bajt statusu centralki. W związku z tym, że jest to pamięć nieulotna, urządzenie po odłączeniu i ponownym włączeniu napięcia zasilania zostanie wprowadzone w ten sam stan, w którym znajdowało się przed jego wyłączeniem.

Odczyt kodu

Jedną z dwóch „trudnych" procedur jest procedura odczytu kodu wprowadzanego z klawiatury. Zwłaszcza kilka początkowych linii wymaga dokładniejszego omówienia. Umieściłem je na **list. 3**.

Od etykiety *read_code_next* rozpoczyna się właściwa procedura rozpoznawania wciśniętych klawiszy i nadawania wartości komórkom *cchar1*...*cchar4*. Pamiętaj-

List. 2. Procedura obsługi przerwania Timera 0

```
;-------------------------------
;obsluga przerwania timera 0
;odpytywanie wejsc alarmu
;-------------------------------
tim0_irq:
   push temp               ;zapamietanie „temp" na stosie
   cbi portd,HIGHNIBBLE    ;odczyt mlodszej polówki bajtu wejsc czujników
   nop                     ;przerwa na czasy propagacji ukladów
   nop
   in temp,pind            ;odczyt mlodszej polówki bajtu
   asr temp                ;"dopasowanie" bitów
   asr temp
   mov sensors,temp
   sbi portd,HIGHNIBBLE
   nop                     ;przerwa na czasy propagacji ukladów
   nop
   in temp,pind            ;odczyt starszej polówki bajtu
   asr temp                ;"dopasowanie" bitów
   asr temp
   andi temp,0x0F
   swap temp
   or sensors,temp
   pop temp
   reti
```

List. 3. Fragment funkcji odczytu kodu *read_code*

```
read_code_loop:
   brid read_code_next     ;jesli przerwania sa wylaczone, przejdz do odczytu klawiatury
                           ;i nie sprawdzaj stanu sensorów
   mov temp,sensors        ;sprawdzenie stanu czujników!!! konieczne tutaj,
                           ;bo funkcja nie przerywa pracy do momentu odczytu
                           ;pelnego, 4-cyfrowego kodu
   cpi temp,SENSORSOFF
   breq read_code_next     ;jesli zaden sensor nie jest aktywny omin
                           ;ponizsze rozkazy
   brts read_code_act      ;jesli flaga T jest ustawiona, to nie wykonuj
                           ;nastaw przerwania Timera 1
   cli                     ;wylaczenie przerwan
   clr goon                ;kasowanie licznika czasu opóznionego zalaczenia
                           ;(pracuje w przerw.Timer'a 1)
   ldi temp,0x80
   out TIMSK,temp          ;zalaczenie przerwan Timer'a 1 (co 1 sek.)
   sei                     ;zalaczenie przerwan
read_code_act:
   set                     ;ustawienie flagi „T"
   push zl                 ;przechowanie wskaznika do bufora w RAM
   rcall alarm_active
   pop zl
read_code_next:
   rcall kbd_read          ;odczyt klawiatury, znak zwracany w zmiennej "char"
```

my, że procedura jest wykorzystywana zarówno w trybie wyłączenia alarmu, jak i w trybie załączenia. Tryby te różnią się pomiędzy sobą przede wszystkim tym, że w stanie załączenia aktywne jest przerwanie Timera 0 przeglądające stan czujników. W związku z tym rozkaz *brid read_code_next* powoduje natychmiastowe przejście do funkcji interpretacji klawiszy, gdy nie jest załączona obsługa przerwania. Zrobiłem tak po to, aby wartość zmiennej *sensors* nie wpływała na stan alarmu. Dlaczego? Prześledźmy dalsze linie programu. Sekwencja rozkazów:

```
movtemp,sensors
cpitemp,SENSORSOFF
```

sprawdza, czy któryś z czujników alarmu jest aktywny. Obojętne jest, czy w stanie wyłączenia, czy załączenia podejmowana jest akcja na podstawie stanu zmiennej *sensors*! W związku z tym możliwe jest uaktywnienie sygnalizacji dźwiękowej i powiadomienia SMS w stanie wyłączenia alarmu. Omawiane wcześniej polecenie *brid* ma uchronić przed tym system alarmowy.

Również w tej procedurze konieczne jest jednokrotne zainicjowanie wartości *goon* i załączenie przerwań Timera 1 w celu odmierzania czasu opóźnionego załączenia. Inicjalizacja przeprowadzana jest w zależności od stanu flagi T. Rozkaz *brts read_code_act* omija polecenia nadające wartości zmiennym w przypadku, gdy flaga jest ustawiona. Polecenia *set* ustawia flagę T po przeprowadzeniu inicjalizacji.

Załączenie alarmu

Na **list. 4** umieściłem fragment kodu źródłowego programu odpowiedzialnego za uaktywnienie alarmu. Rozpoczyna się ona od etykiety *alarm_active* rozkazami testującymi stan zmiennej *goon* odmierzającej czas do załączenia alarmu. Jeśli wartość *goon* jest mniejsza lub równa *DELAYTIME*, to nie jest podejmowana żadna akcja. Jeśli wartość *goon* jest większa od *DELAYTIME*, rozkaz *sbi portb,ALARMOUT* załącza sygnalizację dźwiękową a polecenie *rcall send_SMS* wywołuje podprogram wysyłający SMS pod zaprogramowany i zapamiętany w EEPROM numer telefonu. SMS będzie wysłany, jeżeli bit 0 zmiennej *sms_sent* nie jest ustawiony. Dodatkowo blokowane są przerwania, co powoduje zaniechanie sprawdzania stanu czujników oraz odmierzania czasu. Alarm pozostaje załączony do momentu wprowadzenia kodu wyłączenia lub odłączenia napięcia zasilania.

List. 4. Procedura obsługi załączenia alarmu

```
;------------------------------
;alarm aktywny - sygnalizacja dzwiekowa, powiadomienie za pomoca SMS,
;oczekiwanie na kod wylaczenia
;------------------------------
alarm_active:
   mov temp,goon
   subi temp,DELAYTIME       ;sprawdzenie,czy uplynal juz czas opóznienia
   brsh active_and_on
   rjmp active_exit
active_and_on:
   cli                       ;wylaczenie przerwan - juz nie sa potrzebne
   sbi portb,ALARMOUT        ;zalaczenie wyjscia alarmu
   tst smssent               ;jesli wysylano juz SMS,to nie wysylaj powtórnie
   brne active_exit
   rcall send_SMS            ;wyslanie sms z powiadomieniem
active_exit:
   ret
```

Jacek Bogusz, AVT
jacek.bogusz@ep.com.pl

Wzory płytek drukowanych w formacie PDF są dostępne w Internecie pod adresem: ***http://www.ep.com.pl/?pdf/wrzesien03.htm***.

WYKAZ ELEMENTÓW

Rezystory
R1...R8: 1kΩ
R9...R11: 10kΩ
R14, R16..R18: 33Ω
R19: 560Ω
R20: 270Ω
R21: 750Ω
PR1: 10...22kΩ

Kondensatory
C1, C2: 22pF
C3: 10µF/16V
C4...C8: 1µF /50V
C9: 100µF/16V
C10, C11: 100nF/63V
C12: 470µF/25V

Półprzewodniki
D1: dioda LED czerwona
D2: dioda krzemowa średniej mocy, np. 1N4007
D3...D6: dioda krzemowa małej mocy, np. 1N4148
ISO1, ISO2: PC849 SHARP (LTV849 LITEON)
U1: AT90S2313 (zaprogramowany)
U2: 74LS157
U3: 7805
U4: 74HCT14
U5: MAX232

Różne
Z1, Z2, X1...X8, X10: ARK2 o rastrze 3,5 mm
X9: komplet wtyk wlutowywany w płytkę, prosty, 16 wypr. + gniazdo zaciskane na przewodach
X11: komplet wtyk wlutowywany w płytkę, prosty, 10 wypr. + gniazdo zaciskane na przewodach
X12: złącze DSUB-9/M (męskie)
X14: złącze dla telefonu komórkowego
SW1..SW12: przyciski zwierne
RL1: przekaźnik z cewką 5V, np. ITT RZ2H-5
LCD1: wyświetlacz LCD 2x20
Q1: kwarc 7,3728MHz