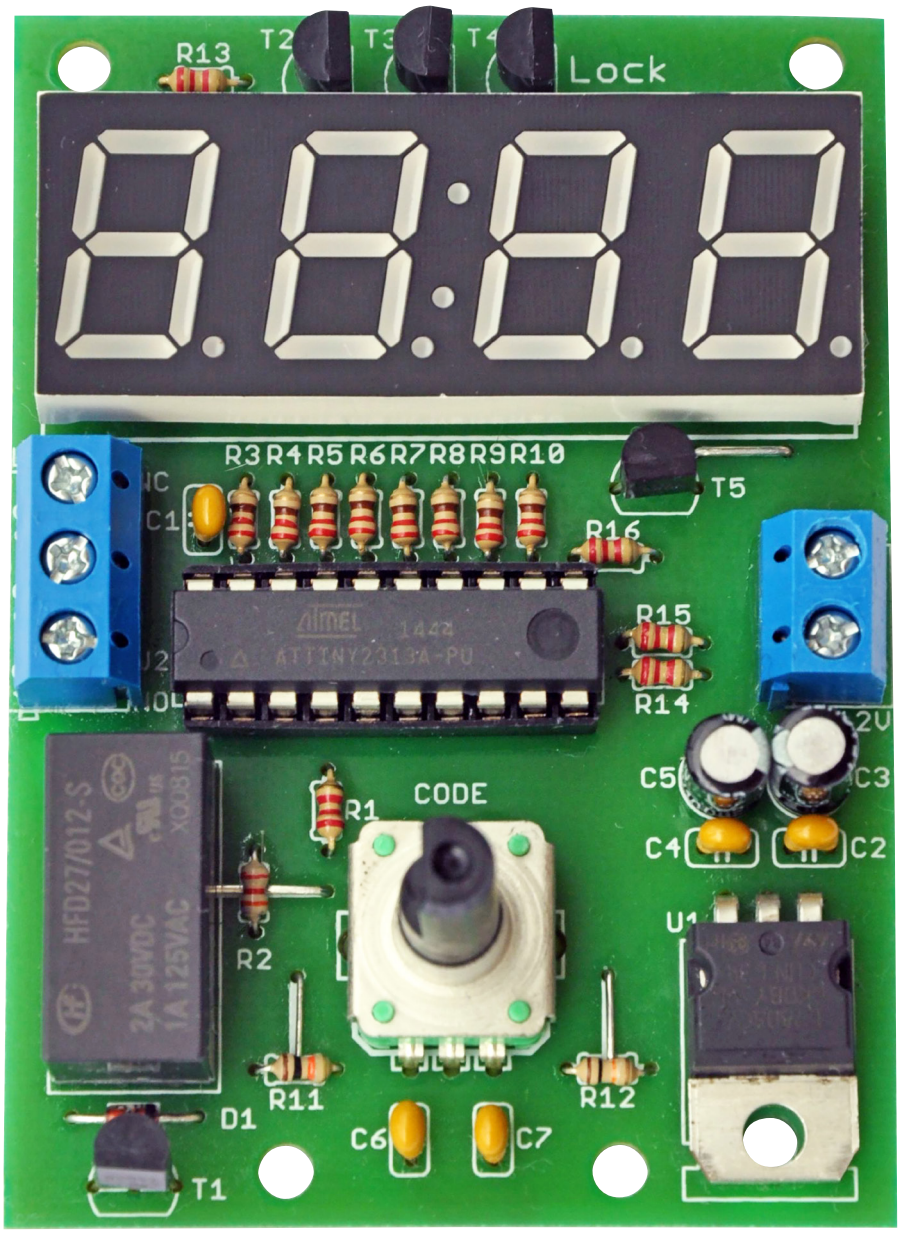
Zamek szyfrowy „Lock”

Projekt zamka szyfrowego, jak to często bywa, powstał w wyniku potrzeby chwili. Otóż dobry znajomy poprosił mnie o zaprojektowanie możliwie najtańszego i prostego w użyciu systemu kontroli dostępu, za którego pomocą można zabezpieczyć drzwi wejściowe obiektu wyposażone w rygiel elektromagnetyczny. Dodatkowym wymaganiem, który miało spełniać projektowane urządzenie, była konieczność wyposażenia go w przyjazny, a zarazem oryginalny interfejs użytkownika, którym miało się odróżniać od urządzeń dostępnych w handlu. Tak oto narodził się projekt niniejszego zamka szyfrowego, który, mam nadzieję, spełni zadanie wielu praktycznych zastosowań.

Rekomendacje: *zamek przyda się w wielu zastosowaniach i uwolni nas od konieczności noszenia przy sobie klucza.*



Schemat ideowy proponowanego rozwiązania zamka szyfrowego pokazano na **rysunku 1**. Jest to nieskomplikowany system mikroprocesorowy, którego sercem jest niewielki mikrokontroler ATtiny2313 realizujący całą założoną funkcjonalność. Mikrokontroler steruje pracą 7-segmentowego wyświetlacza LED (ze wspólną anodą), wykorzystując w tym celu wbudowany w strukturę układ czasowo-licznikowy Timer0 oraz przerwanie od porównania wartości licznika

DODATKOWE MATERIAŁY NA FTP:

ftp://ep.com.pl
USER: 92822, PASS: 37euo8qf

W ofercie AVT*
AVT-5601

Podstawowe informacje:

- Liczba kombinacji kodu: 65536.
- Napięcie zasilania: 9...12 V DC/100 mA.
- Prąd obciążenia (tryb bezczynności/praca/załączenie przekaźnika): 7/55/70 mA.

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

AVT-1919	Sterownik rygla elektromagnetycznego (EP 8/2016)
AVT-3129	Uniwersalny zamek elektroniczny – immobilizer (EdW 7/2015)
AVT-5186	Bezstykowy zamek RFID (EP 5/2009)
AVT-969	Bezstykowy zamek RFID (EP 2/2007)
AVT-522	Miniaturowy zamek cyfrowy. Immobilizer (EP 9/2003)

*** Uwaga! Elektroniczne zestawy do samodzielnego montażu. Wymagana umiejętność lutowania!**
Podstawową wersją zestawu jest wersja [**B**] nazywana potocznie KITem (z ang. zestaw). Zestaw w wersji [**B**] zawiera elementy elektroniczne (w tym [UK] – jeśli występuje w projekcie), które należy samodzielnie wlutować w dołączoną płytkę drukowaną (PCB). Wykaz elementów znajduje się w dokumentacji, która jest podlinkowana w opisie kitu.
Mając na uwadze różne potrzeby naszych klientów, oferujemy dodatkowe wersje:
- wersja [**C**] zmontowany, uruchomiony i przetestowany zestaw [**B**] (elementy wlutowane w płytkę PCB)
- wersja [**A**] płytka drukowana bez elementów i dokumentacja

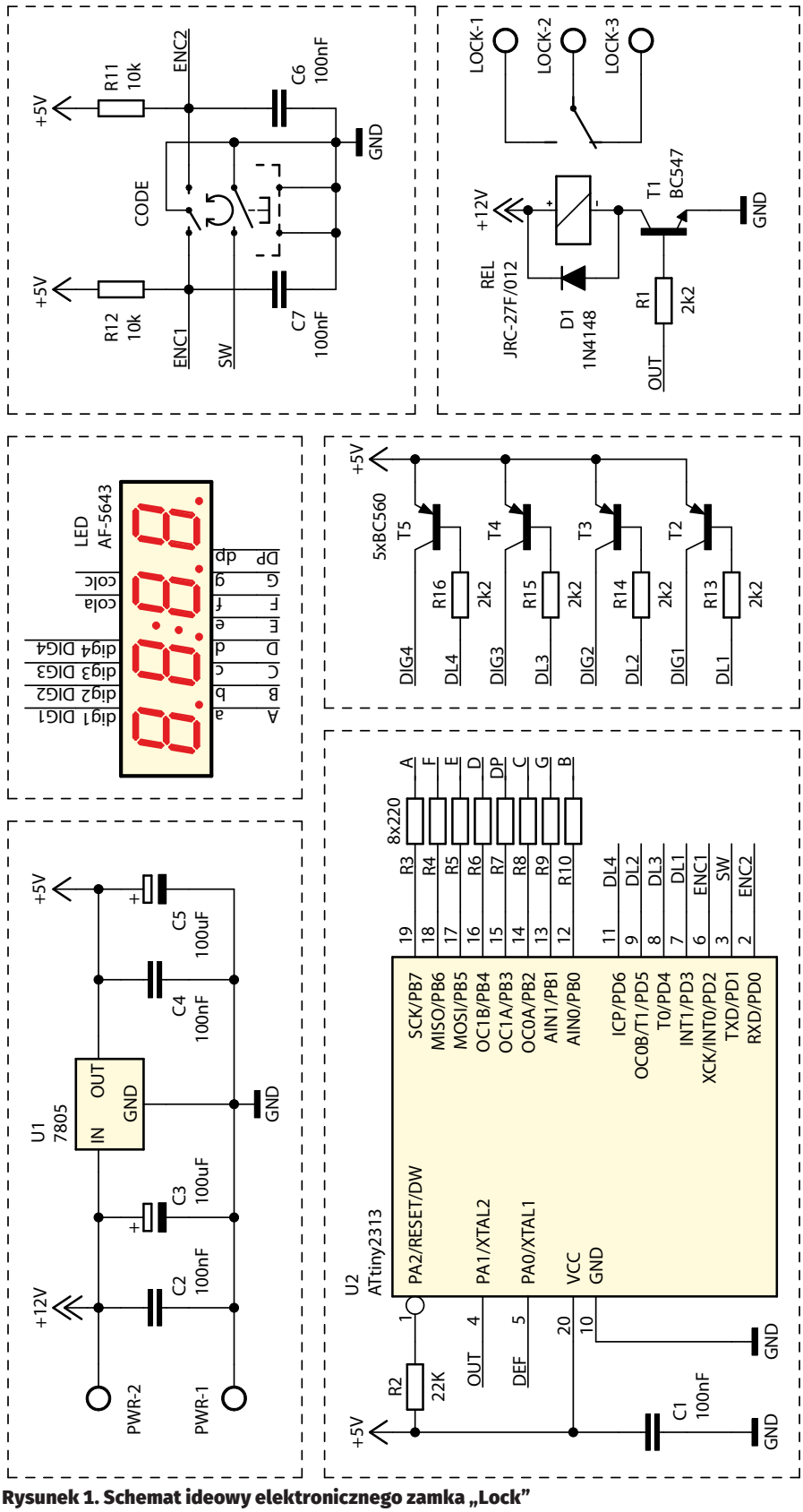
Kity w których występuje układ scalony wymagający zaprogramowania, posiadają następujące dodatkowe wersje:
- wersja [**A+**] płytka drukowana [**A**] + zaprogramowany układ [**UK**] i dokumentacja
- wersja [**UK**] zaprogramowany układ

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

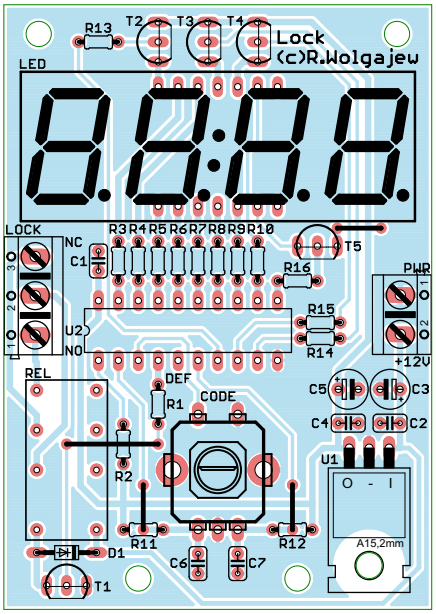
z wartością rejestru OCR0A, dzięki czemu zapewniono realizację dobrze znanego mechanizmu multipleksowania kolejnych cyfr wyświetlacza (częstotliwość przerwania wynosi 240 Hz, co daje 60 Hz/cyfrę), realizuje obsługę elementu regulacyjnego, jakim jest enkoder z wbudowanym przyciskiem, wykorzystując w tym celu przerwanie zewnętrzne INT0 inicjowane wystąpieniem zbocza opadającego na wyprowadzeniu PD2/INT0 mikrokontrolera oraz steruje pracą przekaźnika REL (za pomocą tranzystora) stanowiącego element wykonawczy. Jak wspomniano, sterowanie pracą wyświetlacza LED odbywa się sekwencyjnie, dzięki czemu do realizacji tej funkcjonalności niezbędna stała się mniejsza liczba wyprowadzeń mikrokontrolera. W tym rozwiązaniu katody wyświetlacza LED podłączono, poprzez rezystory ograniczające prąd, bezpośrednio do portu PORTB mikrokontrolera, zaś cztery wspólne anody, poprzez typowe stopnie tranzystorowe (PNP), do portu PORTD. W przerwaniu od porównania wartości licznika Timer0 z wartością rejestru OCR0A wywoływanym co 4,167 ms (ok. 240 Hz) wysyłana jest przez PORTB wartość kolejnej cyfry przeznaczonej do wyświetlenia, po czym jest załączana (poprzez stopień tranzystorowy sterujący portem PORTD) odpowiednia wspólna anoda wyświetlacza LED i proces powtarza się sekwencyjnie dla każdej z cyfr. To typowe rozwiązanie stosowane powszechnie w systemach mikroprocesorowych. Niemniej jednak, poniżej przedstawię bardzo czytelną realizację programową wspomnianego mechanizmu. Na początek niezbędne definicje, które pokazano na **listingu 1**. Dwie tablice umieszczone w pamięci Flash upraszczają funkcję obsługi przerwania odpowiedzialnego za mechanizm multipleksowania, jednocześnie zwiększając czytelność samego kodu. Pierwsza z tablic (DIGITS[17]) przechowuje kombinację segmentów wyświetlacza LED (A…G) odpowiadających obrazom poszczególnych cyfr oraz liter A…F (plus kombinacja odpowiedzialna za wygaszenie wyświetlacza LED), natomiast druga (*COMS[4]*) przechowuje wartości, których przepisanie na port wspólnych anod wyświetlaczy LED, PORTD powoduje załączenie odpowiedniej cyfry wyświetlacza (0…3, liczone od prawej). Dodatkowo zadeklarowano niewielką tablicę w pamięci RAM *Digit[4]*, która przechowuje wartość kolejnych cyfr przeznaczonych do wyświetlenia

Wykaz elementów:
Rezystory: (miniaturowe 1/8 W)
R1, R13...R16: 2,2 kΩ
R2: 22 kΩ
R3...R10: 220 Ω
R11, R12: 10 kΩ
Kondensatory: (raster 2,5 mm)
C1, C2, C4, C6, C7: 100 nF (ceram.)
C3, C5: 100 μF/16 V (elektrolit.)
Półprzewodniki:
U1: 7805 (TO-220)
U2: ATtiny2313 (DIP20)
T1: BC547 (TO-92)
T2...T5: BC560 (TO-92)
D1: 1N4148 (DO35)
LED: wyświetlacz LED np. typu AF-5643
Inne:
CODE: enkoder z wbudowanym przyciskiem
PWR: złącze śrubowe, 2 pola, raster 5 mm
LOCK: złącze śrubowe, 3 pola, raster 5 mm
REL: przekaźnik JRC-27F/012

Ustawienia fusebitów:
CKSEL3...0: 0100
SUT1...0: 10
CKDIV8: 0
EESAVE: 0



Rysunek 1. Schemat ideowy elektronicznego zamka „Lock"



Rysunek 2. Schemat montażowy elektronicznego zamka „Lock”

Listing 1. Plik nagłówkowy mechanizmu multipleksowania wyświetlaczy LED

```
#define SEG_DDR DDRB
#define SEG_PORT PORTB
//Definicje konfiguracji poszczególnych segmentów (katod)
#define SEG_A PB7
#define SEG_B PB0
#define SEG_C PB2
#define SEG_D PB4
#define SEG_E PB5
#define SEG_F PB6
#define SEG_G PB1
#define SEG_DP PB3
//Port katod, jako port wyjściowy
#define SEG_AS_OUTPUT SEG_DDR = 0xFF
//Wszystkie segmenty (katody) wygaszone (aktywny stan „0”, gdyż sterujemy bezpośrednio katodami diod LED)
#define SEG_BLANK SEG_PORT = 0xFF
#define COM_DDR DDRD
#define COM_PORT PORTD
//Definicje konfiguracji poszczególnych wspólnych anod
#define COM_DIG0 PD6
#define COM_DIG1 PD4
#define COM_DIG2 PD5
#define COM_DIG3 PD3
//Port wspólnych anod, jako port wyjściowy
#define COM_AS_OUTPUT COM_DDR |= (1<<COM_DIG3)|(1<<COM_DIG2)|(1<<COM_DIG1)|(1<<COM_DIG0)
//Wszystkie wspólne anody wyłączone (aktywny stan „0”, gdyż sterujemy bazami tranzystorów PNP)
#define COM_BLANK COM_PORT |= (1<<COM_DIG3)|(1<<COM_DIG2)|(1<<COM_DIG1)|(1<<COM_DIG0)
//Definicja bitu odpowiedzialnego za miganie cyfry
#define BLINKING_BIT 0b10000000
//Index wygaszonej cyfry w tablicy DIGITS
#define BLANK_DIGIT_NR 16
```

Listing 2. Niezbędne definicje zmiennych mechanizmu multipleksowania wyświetlaczy LED

```
//Definicje cyfr wyświetlacza (aktywny stan „0”, gdyż sterujemy bezpośrednio katodami diod LED)
const uint8_t DIGITS[17] PROGMEM =
{
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_B)|(1<<SEG_C)|(1<<SEG_D)|(1<<SEG_E)|(1<<SEG_F)),  //0
    (uint8_t) ~((1<<SEG_B)|(1<<SEG_C)),  //1
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_B)|(1<<SEG_D)|(1<<SEG_E)|(1<<SEG_G)),  //2
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_B)|(1<<SEG_C)|(1<<SEG_D)|(1<<SEG_G)),  //3
    (uint8_t) ~((1<<SEG_B)|(1<<SEG_C)|(1<<SEG_F)|(1<<SEG_G)),  //4
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_C)|(1<<SEG_D)|(1<<SEG_F)|(1<<SEG_G)),  //5
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_C)|(1<<SEG_D)|(1<<SEG_E)|(1<<SEG_F)|(1<<SEG_G)),  //6
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_B)|(1<<SEG_C)),  //7
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_B)|(1<<SEG_C)|(1<<SEG_D)|(1<<SEG_E)|(1<<SEG_F)|(1<<SEG_G)),  //8
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_B)|(1<<SEG_C)|(1<<SEG_D)|(1<<SEG_F)|(1<<SEG_G)),  //9
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_B)|(1<<SEG_C)|(1<<SEG_E)|(1<<SEG_F)|(1<<SEG_G)),  //A
    (uint8_t) ~((1<<SEG_C)|(1<<SEG_D)|(1<<SEG_E)|(1<<SEG_F)|(1<<SEG_G)),  //b
    (uint8_t) ~((1<<SEG_D)|(1<<SEG_E)|(1<<SEG_G)),  //c
    (uint8_t) ~((1<<SEG_B)|(1<<SEG_C)|(1<<SEG_D)|(1<<SEG_E)|(1<<SEG_G)),  //d
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_D)|(1<<SEG_E)|(1<<SEG_F)|(1<<SEG_G)),  //E
    (uint8_t) ~((1<<SEG_A)|(1<<SEG_E)|(1<<SEG_F)|(1<<SEG_G)),  //F
    0xFF  //Wyświetlacz wygaszony
};

//Definicje dla portu sterującego wspólnymi anodami wyświetlaczy LED (aktywny stan „0”, gdyż sterujemy bazami tranzystorów PNP)
const uint8_t COMS[4] PROGMEM =
{
    (uint8_t) ~(1<<COM_DIG0),  //Wspólna anoda cyfry 0 (pierwsza z prawej)
    (uint8_t) ~(1<<COM_DIG1),  //Wspólna anoda cyfry 1
    (uint8_t) ~(1<<COM_DIG2),  //Wspólna anoda cyfry 2
    (uint8_t) ~(1<<COM_DIG3)  //Wspólna anoda cyfry 3 (pierwsza z lewej)
};

volatile uint8_t Digit[4];  //Zmienna przechowująca wartość wyświetlaną na wyświetlaczu LED
volatile uint8_t dpON;  //Zmienna odpowiedzialna za wyświetlanie kropki dziesiętnej na pozycji
```

na wyświetlaczu LED. Definicje, o których mowa powyżej, pokazano na **listingu 2**.

Mamy już niezbędne definicje – pora na pierwszą funkcję, której zadaniem jest inicjalizacja procesu multipleksowania składająca się z konfiguracji portów sterujących oraz konfiguracji i uruchomienia układu czasowo-licznikowego Timer0, w tym jego przerwania od porównania wartości licznika z wartością rejestru OCR0A. Wspomnianą funkcję inicjalizacyjną zamieszczono na **listingu 3**. Pora na ostatni element „układanki”, tj. funkcję obsługi przerwania układu Timer0 odpowiedzialną za realizację mechanizmu multipleksowania wyświetlaczy LED, którą pokazano na **listingu 4**.

Zgodnie z tym, co obiecałem we wstępie, dzięki wprowadzeniu zmiennych *DIGITS[17]* i *COMS[4]* (w szczególności) wspomniana funkcja stała się bardzo krótka i czytelna, co powinno być główną zasadą w wypadku implementacji jakichkolwiek funkcji obsługi przerwań systemowych. Co więcej, wprowadzono dodatkową funkcjonalność w postaci obsługi migania cyfr wyświetlacza LED wykorzystywaną w programie obsługi urządzenia do sygnalizacji faktu edycji wybranej pozycji. Funkcjonalność tę uruchamiamy niezależnie dla każdej z cyfr wyświetlacza poprzez ustawienie najbardziej znaczącego bitu odpowiadającej jej zmiennej *Digit[4]*.

To tyle, jeśli chodzi o szczegóły implementacyjne mechanizmu multipleksowania cyfr wyświetlacza LED. Z jednej strony jest to zagadnienie dość nieskomplikowane, z drugiej, nie zawsze rozumiane przez początkujących elektroników.

Montaż

Na **rysunku 2** pokazano schemat montażowy płytki zamka elektronicznego. Montaż należy rozpocząć od wlutowania 4 zworek, których położenie zaznaczono graficznie na obwodzie drukowanym. Następnie lutujemy rezystory i kondensatory, kolejno elementy półprzewodnikowe, a na samym końcu elementy mechaniczne, jak złącza PWR i LOCK, przekaźnik REL oraz enkoder CODE. Podczas montażu wyświetlacza LED należy odsunąć go od powierzchni płytki urządzenia o około 2 mm, aby górna powierzchnia jego obudowy znajdowała się na tym samym poziomie, co górna powierzchna obudowy przekaźnika REL.

Poprawnie zmontowane z użyciem zaprogramowanego mikrokontrolera urządzenie nie wymaga żadnej regulacji i powinno działać od razu po włączeniu zasilania. W zależności od zastosowanego koloru wyświetlacza LED niezbędnym może okazać się dobór wartości rezystorów ograniczających prąd poszczególnych jego segmentów (R3…R10). Domyślny kod użytkownika wynosi „0000”. Kod ten możemy awaryjnie przywrócić przez zwarcie wyprowadzenia nr 5 mikrokontrolera (oznaczonego na płytce jako DEF) do masy zasilania podczas włączania urządzenia (wyłącznie).

Obsługa

Tworząc oprogramowanie, starałem się maksymalnie uprościć obsługę zamka, jednocześnie zachowując wymaganą funkcjonalność. Ma to o tyle znaczenie, że jedynym elementem regulacyjnym jest enkoder z wbudowanym przyciskiem, więc dla uzyskania dużej ergonomii obsługi stosowne założenia musiały być dobrze przemyślane.

Po pierwsze, wprowadzony tryb konfiguracyjny, którego uruchomienie jest możliwe wyłącznie podczas włączania urządzenia dzięki wciśnięciu przycisku wbudowanego w oskę enkodera. Ten tryb pozwala na ustawienie czasu załączenia (0…5 sekund) przekaźnika towarzyszącego odblokowaniu/zablokowaniu zamka szyfrowego. W wypadku ustawienia czasu 0 s przekaźnik pozostaje cały czas załączony po zablokowaniu zamka szyfrowego, zaś wyłączony po odblokowaniu. W wypadku ustawienia wartości z zakresu 1…5 s przekaźnik jest załączany chwilowo na czas odpowiadający ustawieniu (1…5 s) przy każdym zablokowaniu/odblokowaniu naszego zamka szyfrowego.

Wyjście z trybu konfiguracyjnego jest możliwe dzięki ponownemu wciśnięciu przycisku wbudowanego w oskę enkodera. W tym momencie przechodzimy do trybu bezczynności urządzenia (wyświetlacz wygaszony). Co ważne, domyślnie, po włączeniu zasilania (lub wyjściu z trybu konfiguracyjnego) zamek pozostaje odblokowany.

Wciśnięcie oski enkodera powoduje wejście w tryb wprowadzania kodu użytkownika w celu zablokowania zamka. Towarzyszy temu miganie pierwszej cyfry od prawej oraz możliwość jej zmiany dzięki kręceniu oską enkodera (w zakresie „0”…„9”, „A”…„F”). Zatwierdzenia wprowadzanej cyfry dokonujemy poprzez wciśnięcie przycisku wbudowanego w oskę enkodera. W tym momencie cyfra ta przesunie się w lewo o jedną pozycję i przestanie migać, zaś cyfra pierwsza od prawej, jak poprzednio, zacznie ponownie migać, sugerując możliwość jej edycji. Innymi słowy, kod wprowadzamy od lewej do prawej (patrząc na kod, zaczynając od cyfry bardziej znaczącej), przy czym bieżąco edytowana cyfra jest wyświetlana zawsze na pozycji pierwszej od prawej, zaś poprzednio wprowadzone cyfry przesuwane o odpowiednią liczbę pozycji w lewo. Kroki te powtarzamy 4 razy, aż do momentu wprowadzenia całego kodu, po którym nastąpi zablokowanie zamka szyfrowego (i stosowna reakcja przekaźnika), zapamiętanie kodu w nieulotnej pamięci EEPROM mikrokontrolera oraz przejście w tryb bezczynności urządzenia.

Fakt zablokowania zamka jest sygnalizowany przez świecenie kropki dziesiętnej na pozycji 0 wyświetlacza nawet w czasie bezczynności urządzenia. Warto również podkreślić, że dla wygody użytkownika wprowadzono dodatkowy mechanizm, dzięki któremu możemy pominąć wprowadzanie kodu użytkownika służącego do zablokowania zamka, przyjmując domyślnie wcześniej używany i zapamiętany kod blokady. W tym celu po wejściu w tryb wprowadzania kodu użytkownika (ze stanu bezczynności) należy przycisnąć i przytrzymać na dłużej oskę enkodera, po czym nastąpi zablokowanie zamka szyfrowego (i stosowna reakcja przekaźnika) oraz przejście w tryb bezczynności. W tym momencie (po zablokowaniu zamka) wciśnięcie oski enkodera powoduje wejście w tryb wprowadzania kodu użytkownika (jak poprzednio), lecz tym razem w celu odblokowania zamka.

Wprowadzenie poprawnego kodu powoduje odblokowanie zamka (i stosowną reakcję przekaźnika), zaś błędnego, przejście do trybu bezczynności urządzenia w oczekiwaniu na wprowadzanie nowego kodu odblokowującego. Pozostawanie w trybie wprowadzania kodu użytkownika (migająca pierwsza cyfra od prawej) i nieukończenie tego procesu w czasie 5 s powoduje wyjście do trybu bezczynności urządzenia. Czas ten liczony jest każdorazowo od nowa po dokonaniu jakiejkolwiek edycji. Ponadto, w celu sygnalizacji stanu pracy, urządzenie nasze wyświetla kilka informacji tekstowych, których zestawienie, łącznie z ich znaczeniem, umieszczono w **tabeli 1**.

Robert Wołgajew, EP

Listing 3. Funkcja inicjalizacyjna mechanizmu multipleksowania wyświetlaczy LED

```
void initMultiplex(void)
{
    //Porty wspólnych anod i katod, jako wyjściowe ze stanami nieaktywnymi na wyjściach
    SEG_BLANK;
    SEG_AS_OUTPUT;
    COM_BLANK;
    COM_AS_OUTPUT;
    //Konfiguracja układu Timer0 w celu generowania przerwania do obsługi multipleksowania wyświetlacza LED (240 Hz)
    TCCR0A = (1<<WGM01);  //Tryb CTC
    TCCR0B = (1<<CS01)|(1<<CS00);  //Preskaler = 64
    OCR0A = 64;  //240 Hz (co 4.167ms)
    TIMSK |= (1<<OCIE0A);  //Uruchomienie przerwania Output Compare Match A
    for(uint8_t i=0; i<4; ++i) Digit[i] = BLANK_DIGIT_NR;  //Wygaszenie wszystkich cyfr
}
```

Listing 4. Funkcja obsługi przerwania układu Timer0 odpowiedzialna za realizację mechanizmu multipleksowania wyświetlaczy LED

```
ISR(TIMER0_COMPA_vect)
{
    static uint8_t Nr;  //Numer kolejnej cyfry przeznaczonej do wyświetlenia
    static uint8_t timer4ms;  //Timer programowy 4ms służący do obsługi migania cyfr wyświetlacza LED
    register uint8_t currDigit = Digit[Nr];  //Optymalizacja volatile
    COM_BLANK;  //Wyłączenie wspólnych anod wyświetlaczy LED
    if(currDigit & BLINKING_BIT)  //Sprawdzamy, czy dla danej cyfry uruchomiono funkcję migania (ustawiony 7. bit zmiennej)
    {
        if(++timer4ms & 0x20) SEG_PORT = pgm_read_byte(&DIGITS[currDigit & (~BLINKING_BIT)]); else SEG_BLANK;  //Migamy
    }
    else SEG_PORT = pgm_read_byte(&DIGITS[currDigit & (~BLINKING_BIT)]);  //Pobranie kolejnej cyfry na port katod
    if(Nr==0 && dpON) SEG_PORT &= ~(1<<SEG_DP);  //Obsługa kropki dziesiętnej na pozycji 0
    COM_PORT &= pgm_read_byte(&COMS[Nr]);  //Włączenie odpowiedniej wspólnej anody (aktywny stan „0”)
    Nr = (Nr+1) & 0x03;
}
```

Tabela 1. Informacje tekstowe wyświetlane przez elektroniczny zamek „Lock”

Komunikat	Znaczenie
dEF	Przywrócenie domyślnego kodu użytkownika tj. „0000”
Err	Sygnalizacja błędnie wprowadzonego kodu
OPEn	Sygnalizacja odblokowania zamka (poprawnie wprowadzony kod)
cLOSE	Sygnalizacja zablokowania zamka (napis przewijany)