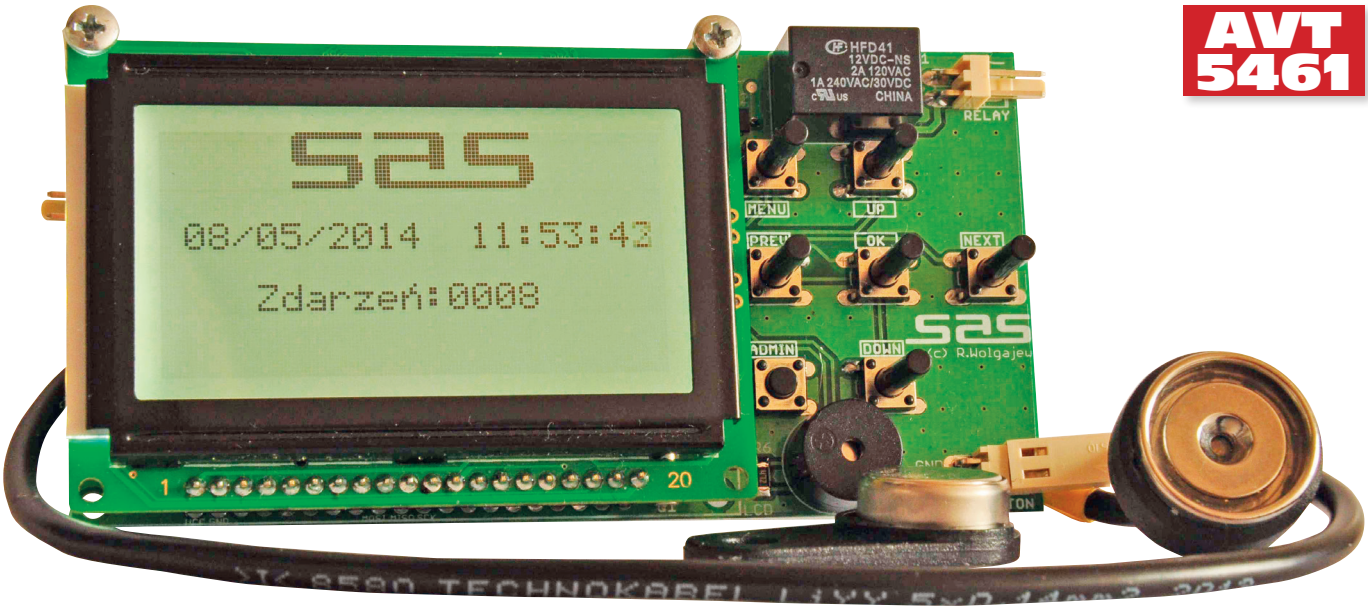
AVT 5461



SAS

Nieskomplikowany system kontroli dostępu

Jak to często bywa w naszym ciekawym hobby, inspirujące pomysły są niejednokrotnie wynikiem potrzeby chwili i często dają dodatkowe możliwości poszerzenia własnej wiedzy. Tak było i tym razem. Wykonując opisywane urządzenia mogłem poznać kilka nieznanych mi dotąd zagadnień jak i peryferiów, wszak każdy z nas uczy się przez całe życie. Wspomnianą potrzebą chwili okazała się sposobność zaprojektowania prostego systemu kontroli dostępu zapewniającego możliwość nadzorowanego ruchu osobowego w ramach dużego zakładu pracy.

Rekomendacje: *system przyda się do kontroli dostępu w firmie lub domku jednorodzinnym, jest też świetnym instruktażem, jak można samodzielnie zbudować podobne urządzenie.*

Jako najprostsze oraz również najtańsze z punktu widzenia inwestora, okazało się zastosowanie nieco już zapomnianych kluczy elektronicznych w postaci pastylek iButton, z unikalnym 64-bitowym numerem seryjnym. Zdaję sobie sprawę, że rozwiązanie jest dość leciwe jak również nie do końca bezpieczne (patrz mój projekt emulatora *cButton* z EP 2/2009), lecz czy istnieją zabezpieczenia stuprocentowe? Wszystko zależy od wymagań projektu i charakteru chronionego obiektu i w wielu wypadkach tego typu, proste i tanie rozwiązanie jest nader wystarczające. Idąc jednak dalej, w typowy dla siebie sposób, postanowiłem rozszerzyć nieco funkcjonalność takiego systemu o możliwość rejestrowania zdarzeń oraz zarządzania prawami dostępu uprawnionych użytkowników.

Pierwszym problemem, z którym musiałem się zmierzyć, był wybór medium dla rejestracji zdarzeń i parametrów użytkowników. W przypadku konfiguracji listy użytkowników mających prawa dostępu do obiektu, sprawa jest dość prosta, gdyż taka lista, jak i niezbędna konfiguracja, nie są wykonywane 9 modyfikowane zbyt często, więc wystarczy wbudowana w mikrokontroler nieulotna pamięć EEPROM o odpowiedniej wielkości. Nieco inaczej wy-

W ofercie AVT*
AVT-5461 A AVT-5461 B
AVT-5461 C

Podstawowe informacje:

- Napięcie zasilania: 9...12 V DC.
- Maksymalny prąd obciążenia (podświetlenie wyświetlacza wyłączone/załączone): 25 mA/45 mA.
- Maksymalna liczba użytkowników: 50 (+Administrator).
- Maksymalna liczba zdarzeń: 1000.
- Czas automatycznego wylogowania Administratora: 60 sekund.
- Czas automatycznego wygaszenia podświetlenia: 30 sekund bezczynności użytkownika.
- Czas automatycznego wyjścia do menu głównego: 30 sekund bezczynności użytkownika.
- Maksymalny prąd styków przekaźnika wykonawczego: 1 A przy 220 V AC (szczegóły w dokumentacji elementu).

Dodatkowe materiały na FTP:

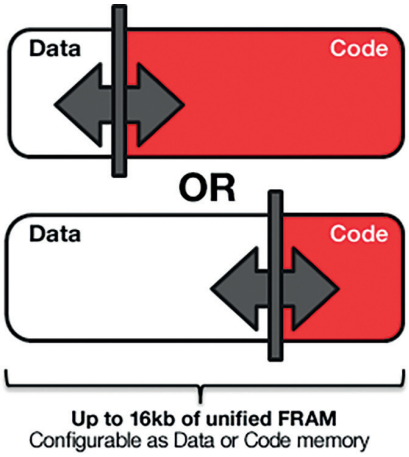
ftp://ep.com.pl, user: 26526, pass: 841uhx54

- wzory płytek PCB

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

AVT5171 cButton – Emulator pastylek Maxim-Dallas (EP 2/2009)
AVT-2847 Immobilizer z funkcją AntiCarJack (EdW 11/2007)

* 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 (niezbędne 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



Rysunek 1. Struktura pamięci mikrokontrolerów firmy Texas Instruments z serii MSP430 FRAM Series

gląda sprawa z rejestrowaniem zdarzeń. Tutaj jest niezbędna pamięć o nieograniczonej (teoretycznie) liczbie możliwych do przeprowadzenia operacji zapisu (w zasadzie, kasowania, w przypadku pamięci EEPROM), a jak wiadomo pamięci EEPROM mają ograniczoną liczbę cykli zapisu. W takim wypadku, najprostszym rozwiązaniem byłoby zastosowanie statycznej pamięci SRAM, ale ta z kolei wymagałaby zastosowania jakiegoś mechanizmu podtrzymania zasilania, by informacje w niej zawarte nie ulegały skasowaniu po jego wyłączeniu. To nie byłoby optymalne i łatwe rozwiązanie biorąc pod uwagę wymóg niezawodności urządzenia. W związku z tym musiałem wybrać inne rozwiązanie, które byłoby jak najprostsze, niezawodne i co ważne – tanie. Dość szybko okazało się, że idealnym peryferium do tego typu zastosowań jest pamięć FRAM (*Ferroelectric Random Access Memory*). Co prawda, te pamięci są znane już od ponad 20 lat, jednak ich dotychczasowa cena jak i dostępność znacznie ograniczała pole ewentualnych zastosowań.

Pamięć taka jest rodzajem pamięci RAM, w której do zapamiętywania bitów słowa danych wykorzystano efekt ferroelektryczny specjalnego kryształu ferromagnetycznego, co wiąże się między innymi z tą ważną cechą, iż taka pamięć nie wymaga specjalnych zabiegów podczas zapisu, jak to ma miejsce w pamięci EEPROM. Bardzo krótki, w odróżnieniu od pamięci EEPROM, jest też czas zapisu pamięci FRAM. Jako, że w Internecie można bez problemu znaleźć sporo szczegółowych informacji na temat budowy i zasady działania tego typu peryferiów (polecam zwłaszcza stronę *http://goo.gl/djjB9M*) nie będę rozwijał tego, skądinąd ciekawego zagadnienia, a skupię się na zaletach tego typu pamięci, które to można zdefiniować w kilku, poniższych punktach:

- Zgodność wyprowadzeń z pamięciami szeregowymi EEPROM i równoległymi SRAM.
- Liczba cykli odczytu/zapisu rzędu 100 bilionów, więc praktycznie nieograniczona.
- Bardzo duża szybkość zapisu, rzędu nanosekund.
- Mały pobór prądu, ok. 1/60 energii potrzebnej dla pamięci EEPROM do zapisu lub 1/400 zapotrzebowania pamięci FLASH w trakcie zapisu.
- Wysoka stabilność w czasie ~40lat @75°C.
- Znacznie większą odporność na promieniowanie niż pamięci Flash/EEPROM, jak również na działanie pola magnetycznego (ważne w zastosowaniach medycznych i wojskowych).
- Duża szybkość interfejsów komunikacyjnych w dostępnych rozwiązaniach rynkowych, np. dla pamięci firmy RAMTRON szybkość interfejsu I²C może dochodzić do niestandardowego 1 MHz.

Jak widać, pamięć FRAM jest niemalże idealnym peryferium pamięciowym i z powodzeniem łączy zalety szybkich, ale ulotnych pamięci SRAM oraz wolnych, ale nieulotnych pamięci EEPROM/Flash. Nie zostało to niezauważone przez producentów mikrokontrolerów, bo przecież jest to również idealne rozwiązanie, jeśli chodzi o wbudowaną, nieulotną pamięć mikrokontrolera! Zastępując wbudowaną pamięć EEPROM pamięcią FRAM zwalniamy na przykład programistę z potrzeby stosowania różnych „sztuczek" programowych jak np. wielozapis (tzw. *wear-leveling*) by wydatnie zwiększyć „czas życia" komórek pamięci EEPROM, co przekłada się na skrócenie kodu wynikowego jak i sprawniejsze działania aplikacji. Co więcej, firma Texas Instruments, lider w technologii pamięci FRAM, poszła jeszcze dalej. Po co wbudowywać w mikrokontroler wiele typów, niezbędnej pamięci, jak Flash (treść programu aplikacji), RAM (pamięć danych) i EEPROM (nieulotna pamięć ogólnego przeznaczenia), by użytkownik wybierał pomiędzy różnymi typami układów o zróżnicowanej objętości każdej z nich? Nie lepiej zastosować jeden rodzaj pamięci, w tym przypadku FRAM, i dać użytkownikowi możliwość definiowania zakresów każdej ze znanych dotychczas, typowych pamięci mikrokontrolera w zakresie całej, dostępnej pamięci FRAM? Mrzonki? Wcale nie! Wystarczy spojrzeć do not katalogowych mikrokontrolerów tejże firmy, z rodziny MSP430 FRAM Series, a ujrzymy następującą grafikę pokazaną na **rysunku 1** oraz opis „*Universal Memory = Program + Data + Storage*".

Jak dla mnie, ideał! Pozostawiając kwestie doskonałości pamięci FRAM, wróćmy zatem do konstrukcji naszego urządzenia, którego to schemat pokazano na **rysunku 2**.

Jak widać, jest to system mikroprocesorowy, którego „sercem" jest nowoczesny mikrokontroler ATmega32A. Przy udziale sprzętowego interfejsu TWI (kompatybilnego z I²C) zapewnia on obsługę pamięci FRAM (U3) oraz obsługę układu DS1338 (U4) realizującego funkcjonalność dokładnego zegara czasu rzeczywistego (RTC), wyposażonego w zintegrowany, automatyczny mechanizm podtrzymania zasilania (co zapewnia bateria CR2032). Mikrokontroler ten jest odpowiedzialny również za obsługę interfejsu użytkownika zbudowanego z wykorzystaniem kilku przycisków, buzzer-a piezoelektrycznego i niedrogiego, graficznego wyświetlacza LCD o rozdzielczości 128×64 piksele, wyposażonego w dość znany i prosty w obsłudze sterownik KS108A.

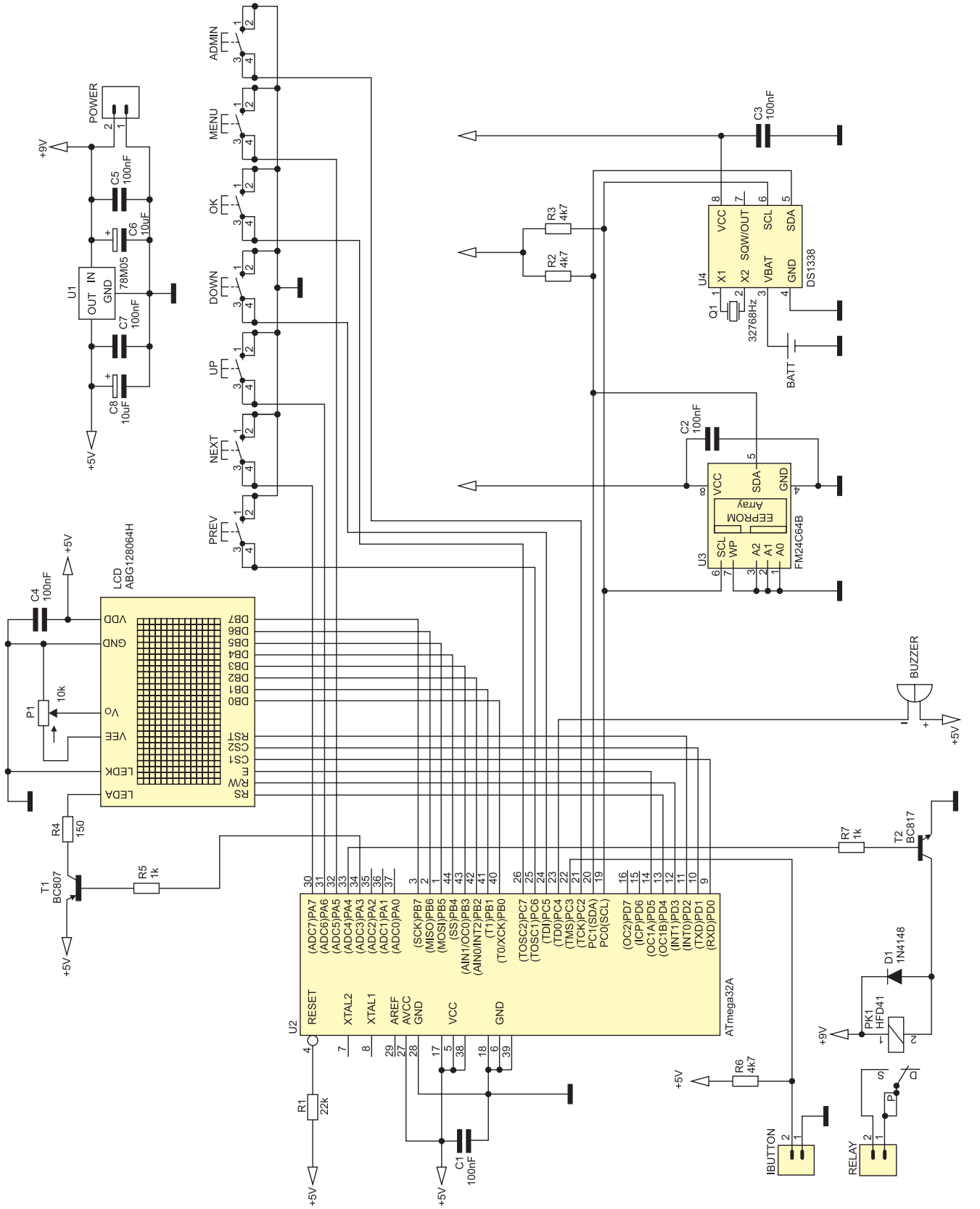
Wybór tego konkretnego typu mikrokontrolera nie był bynajmniej podyktowany niezbędną pojemnością pamięci Flash, gdyż program obsługi aplikacji napisany za pomocą kompilatora AVR-GCC zajmuje około 9 kB, z czego pokaźna część przypada na dwujęzykowe teksty interfejsu użytkownika zapisane w pamięci Flash (w tym definicje czcionki 5×7 pikseli). Wybór ten wynika wyłącznie z potrzeby dobrania mikrokontrolera o odpowiednio dużej liczbie wyprowadzeń,

Wykaz elementów

Rezystory: (SMD 1206)
R1: 22 kΩ
R2, R3, R6: 4.7 kΩ
R4: 100…150 Ω
R5, R7: 1 kΩ
P1: 10 kΩ/A (pot. montażowy)
Kondensatory: (SMD 1206)
C1…C5, C7: 100 nF (X5R)
C6, C8: 10 µF/16 V (SMD „B", EIA 3528-21R)
Półprzewodniki:
U1: 78M05 (DPAK)
U2: ATmega32A (TQFP44)
U3: FM24C64B (SOIC08)
U4: DS1338 (SOIC08)
T1: BC807 (SOT23-BEC)
T2: BC817 (SOT23-BEC)
D1: 1N4148 (MINIMELF)
Inne:
LCD – wyświetlacz graficzny LCD-AG-128064H-DIW W/KK-E6 (lub inny, ze sterownikiem KS108A o zgodnych wyprowadzeniach i wymiarach zewnętrznych)
PK1: przekaźnik HFD41-9V (napięcie cewki zależne od napięcia zasilającego urządzenie)
Q1: 32768 Hz (rezonator kwarcowy, zegarkowy)
BATT: koszyczek baterii CR2032
BUZZER: buzzer piezoelektryczny z generatorem 5 V
PREV, NEXT, UP, DOWN, OK, MENU: microswitch z ośką 16 mm do montażu przewlekanego
ADMIN: microswitch z ośką 1 mm do montażu przewlekanego
IBUTTON, POWER, RELAY: złącze kątowe 2-pin (NSL25-2W)
Adapter do pastylek iButton (tzw. touch probe) np. DS9092+/PL,
Pastylki iButton DS1990 (dowolna wersja) w liczbie zależnej od wielkości systemu (plus pastylka Administratora)

niezbędnej z uwagi na rodzaj zastosowanego interfejsu użytkownika. Niemniej jednak, ważnym elementem przy wyborze zastosowanego mikrokontrolera był także wymóg wielkości wbudowanej pamięci EEPROM, która w naszym musi pomieścić około 1024 bajtów. Co prawda można by zastosować dodatkową pamięć zewnętrzną lub, tak jak napisałem wcześniej, wykorzystać część pamięci FRAM do przechowywania informacji o użytkownikach, lecz zastosowane rozwiązanie wydawało mi się optymalne, jeśli chodzi o możliwości urządzenia i cenę.

W implementacji programu obsługi aplikacji nie mogło również zabraknąć miejsca na realizację programowej obsługi magistrali 1-wire (w tym kontroli i kalkulacji sum kontrolnych typu CRC8), dzięki której możliwa stała się obsługa pastylek/kluczy iButton jak i implementacji prostego sterownika wyświetlacza graficznego, który to został napisany w najprostszy z możliwych sposobów, to znaczy taki, by emulował niejako wyświetlacz alfanumeryczny o organizacji 8



Rysunek 2. Schemat ideowy sterownika systemu SAS

Listing 1. Ciała funkcji odpowiedzialnych za obsługę interfejsu TWI mikrokontrolera ATmega32A

```
void TWI_Start(void)
{
    TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTA);
    while (!(TWCR&(1<<TWINT)));
}

void TWI_Stop(void)
{
    TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);
    while (TWCR&(1<<TWSTO));
}

void TWI_WriteByte(uint8_t Byte)
{
    TWDR = Byte;
    TWCR = (1<<TWINT)|(1<<TWEN);
    while ( !(TWCR&(1<<TWINT)));
}

uint8_t TWI_ReadByte(uint8_t ACK_NACK)
{
    TWCR = (1<<TWINT)|(ACK_NACK<<TWEA)|(1<<TWEN);
    while ( !(TWCR & (1<<TWINT)));
    return TWDR;
}
```

Listing 2. Struktura danych odpowiedzialna za przechowywanie informacji o zdarzeniu

```
typedef union
{
    struct
    {
        dateTimeType dateTime;
        uint8_t userID;
        uint8_t Status;
    };
    uint8_t index[8];
} eventType;
```

Listing 3. Struktura danych odpowiedzialna za przechowywanie informacji o bieżącej dacie i godzinie zegara RTC

```
typedef union
{
    struct
    {
        uint8_t Day;
        uint8_t Month;
        uint8_t Year;
        uint8_t Hour;
        uint8_t Minute;
        uint8_t Second;
    };
    uint8_t index[6];
} dateTimeType;
```

linii po 21 znaków o rozmiarze 5×7 pikseli. Przejdźmy, zatem do szczegółów programowych naszej aplikacji.

Jak wspomniałem, program do obsługi aplikacji musi zawierać co najmniej 2 rodzaje struktur danych zaopatrzonych w odpowiednie indeksy: strukturę opisującą dane użytkownika umieszczaną w pamięci EEPROM mikrokontrolera oraz strukturę opisująca zdarzenie (z udziałem użytkownika) umieszczaną w nieulotnej pamięci FRAM, w naszym przypadku 64-kilobitowej pamięci FM24C64B produkcji firmy Ramtron. Można byłoby, co oczywiste, obie struktury umieścić w pamięci FRAM, ale chciałem zmaksymalizować liczbę pamiętanych zdarzeń, w związku z czym prawie cała pojemność tejże pamięci została wykorzystana właśnie w tym celu. Dodatkową, niewspomnianą dotychczas strukturą danych, znacznie ułatwiającą tworzenie programu aplikacji, jest również struktura opisująca bieżące nastawy zegara czasu rzeczywistego RTC. Zanim jednak przejdę do szczegółów implementacyjnych, przedstawię podstawowe funkcje, dzięki którym możliwa jest komunikacja przy udziale interfejsu TWI mikrokontrolera, przy czym należy zaznaczyć, iż są to podstawowe implementacje poszczególnych funkcjonalności niewyposażone np. w mechanizm obsługi błędów, który moim zdaniem, nie jest elementem niezbędnym w tak prostych i niewielkich systemach mikroprocesorowych. Ciała funkcji odpowiedzialnych za obsługę interfejsu TWI pokazano na **listingu 1**.

Myślę, że nie wymagają one dodatkowego komentarza, gdyż ich nazwy są na tyle wymowne, iż nie pozostawiają wątpliwości, co do realizowanych przez nie funkcjonalności. Przejdźmy, zatem do obsługi pamięci FRAM. Na początek, struktura danych, która odpowiedzialna jest za przechowywanie informacji o zdarzeniu. Jej budowę pokazano na **listingu 2**. Jest to unia z osadzoną, tzw. strukturą anonimową, to znaczy taką, której nie nadano nazwy i jest dostępna wyłącznie w ramach deklarowanej unii. Tego typu konstrukcja zapewnia łatwy dostęp do pól struktury za pomocą ich nazw jak i indeksu, co upraszcza adresowanie lub nadawanie wartości poszczególnym elementom. Elementem tej struktury jest pole *dateTime* o zdefiniowanych wcześniej typie *dateTimeType*, którego konstrukcję pokazano na **listingu 3**, a które to, jak łatwo się domyślić, jest przeznaczone do przechowywania informacji o bieżącej dacie i godzinie (odczytanej, czy też przeznaczonej do zapisania) zegara RTC.

Znając podstawowe, utworzone przez nas, typy danych, przejdźmy do funkcji odpowiedzialnych za obsługę pamięci FRAM w zakresie zapisywania i odczytywania zdarzeń opisanych zadeklarowanymi typami, choć przyznać trzeba, że funkcje te mogę służyć do zapisywania/odczytywania dowolnych bloków danych z zewnętrznej pamięci

Listing 4. Ciała funkcji odpowiedzialnych za obsługę pamięci FRAM

```
void FRAMwriteBlock(uint16_t blockNr, eventType *Data)
{
    register uint8_t bankSize = sizeof *Data;  //Określamy rozmiar bloku danych Data
    register uint16_t startAddress = blockNr*bankSize;  //Adres początku obszaru przewidzianego do zapisu
    register uint8_t i = 0;
    TWI_Start();
    TWI_WriteByte(FRAM_WRITE_ADDR);  //Adres zapisu pamięci FRAM (bit W/R = 0)
    TWI_WriteByte(startAddress>>8);  //Adres początku obszaru przewidzianego do zapisu - MSB
    TWI_WriteByte(startAddress & 0xFF);  //Adres początku obszaru przewidzianego do zapisu - LSB
    while(bankSize--) TWI_WriteByte(Data->index[i++]);
    TWI_Stop();
}


void FRAMreadBlock(uint16_t blockNr, eventType *Data)
{
    register uint8_t bankSize = sizeof *Data;  //Określamy rozmiar bloku danych Data
    register uint16_t startAddress = blockNr*bankSize;  //Adresu początku obszaru przewidzianego do odczytu    register
uint8_t i = 0;
    TWI_Start();
    TWI_WriteByte(FRAM_WRITE_ADDR);  //Adres zapisu pamięci FRAM (bit W/R = 0)
    TWI_WriteByte(startAddress>>8);  //Adresu początku obszaru przewidzianego do odczytu - MSB
    TWI_WriteByte(startAddress & 0xFF);  //Adresu początku obszaru przewidzianego do odczytu - LSB
    TWI_Start();
    TWI_WriteByte(FRAM_READ_ADDR);  //Adres odczytu pamięci FRAM (bit W/R = 1)
    while(bankSize--) Data->index[i++] = TWI_ReadByte(bankSize? ACK:NACK);
    TWI_Stop();
}
```

FRAM/EEPROM. Wspomniane ciała tychże funkcji pokazano na **listingu 4**.

Pora na przedstawienie funkcji odpowiedzialnych za obsługę zegara czasu rzeczywistego, które tak naprawdę operują na wewnętrznych rejestrach układu DS1338 przewidzianych do operacji odczytu/aktualizacji czasu RTC. Nie wchodzę, w tym miejscu, w szczegóły implementacyjne obsługi układów tego typu, gdyż po pierwsze, zasady komunikacji z tymi peryferiami nie odbiegają od standardowych rozwiązań dotyczących magistrali I²C, a po drugie, w Internecie dostępna jest szczegółowa dokumentacja producenta opisująca te zagadnienia. Funkcje, o których mowa przedstawiono na **listingu 5**.

Nieco prościej przedstawia się sprawa odczytu/zapisu danych użytkowników, ponieważ te dane są zapisane w pamięci EEPROM mikrokontrolera, a środowisko AVR-GCC dostarcza gotowych rozwiązania do wygodnej obsługi tychże operacji, w dodatku dla różnych typów danych, co znacznie ułatwia życie programistom. W pierwszym kroku, podobnie jak poprzednio, deklarujemy nowy typ przeznaczony do przechowywania danych użytkownika, a następnie definiujemy dwie zmienne: jedną w pamięci RAM jako bufor operacji zapisu/odczytu, a drugą jako zmienną w pamięci EEPROM mikrokontrolera, w tym wypadku tablicę naszego typu o rozmiarze równym liczbie użytkowników. Tak proste podejście do tematu jest możliwe jest dzięki przemyślanej konstrukcji pliku nagłówkowego *eeprom.h*, standardowo dostarczanego ze wspomnianym środowiskiem. Rzeczowe deklaracje/definicje przedstawiono na **listingu 6**.

Jako, że potrzebujemy odczytywać całe bloki danych a nie pojedyńcze bajty, do tego typu operacji wykorzystamy dwie, wbudowane, niestandardowe funkcje, których użycie przedstawia się nastepująco:

```
eeprom_read_block(&User,
&UserEE[index], sizeof(User));
eeprom_write_block(&User,
&UserEE[index], sizeof(User));
```

Zmienna *index* określa kolejny numer elementu w tablicy użytkowników. Prawda, że proste? Zresztą, polecam wszystkim dokładną lekturę dokumentacji dostarczanej ze środowiskiem AVR-GCC (np. pod postacią dokumentu *avr-libc-user-manual.pdf*), gdyż zawiera ona dokładny opis wielu gotowych funkcji i makr związanych bezpośrednio z architekturą mikrokontrolerów AVR.

Listing 5. Ciała funkcji odpowiedzialnych za obsługę układu DS1338

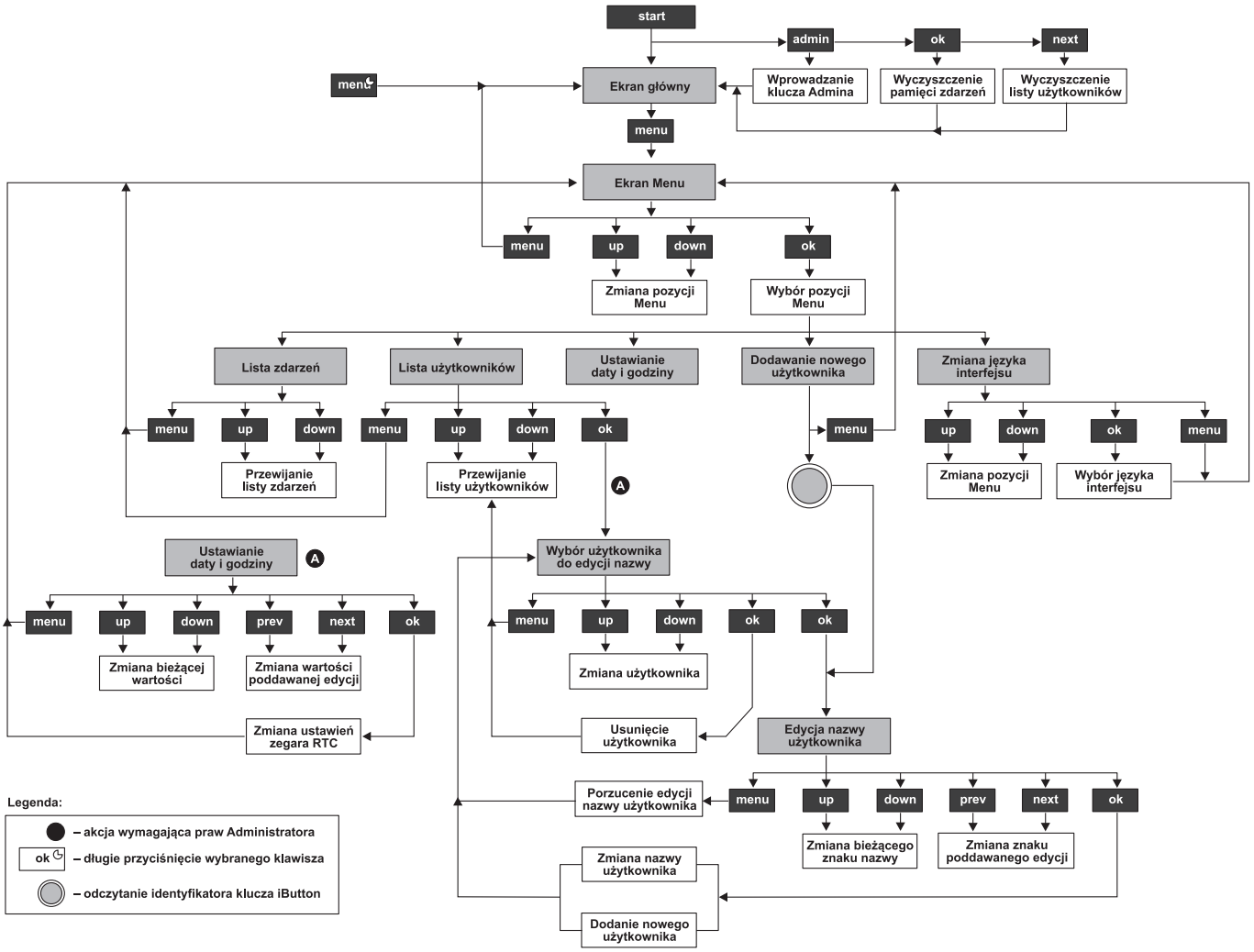
```
void RTCwriteDateTime(dateTimeType *Data)
{
    TWI_Start();
    TWI_WriteByte(DS1338WRITE_ADDR); //Adres zapisu układu DS1338 (bit W/R = 0)
    TWI_WriteByte(0x00);  //Adresu początku obszaru przewidzianego do zapisu
    TWI_WriteByte(((Data->Second/10)<<4) | (Data->Second%10));  //Sekundy w zapisie BCD
    TWI_WriteByte(((Data->Minute/10)<<4) | (Data->Minute%10));  //Minuty w zapisie BCD
    TWI_WriteByte(((Data->Hour/10)<<4) | (Data->Hour%10));  //Godziny w zapisie BCD
    TWI_WriteByte(0x00);  //Dzień tygodnia - niewykorzystywany
    TWI_WriteByte(((Data->Day/10)<<4) | (Data->Day%10));  //Dzień w zapisie BCD
    TWI_WriteByte(((Data->Month/10)<<4) | (Data->Month%10));  //Miesiąc w zapisie BCD
    TWI_WriteByte(((Data->Year/10)<<4) | (Data->Year%10));  //Rok w zapisie BCD
    TWI_Stop();
}

void RTCreadDateTime(dateTimeType *Data)
{
    register uint8_t temp;
    TWI_Start();
    TWI_WriteByte(DS1338WRITE_ADDR); //Adres zapisu układu DS1338 (bit W/R = 0)
    TWI_WriteByte(0x00);  //Adresu początku obszaru przewidzianego do odczytu
    TWI_Start();
    TWI_WriteByte(DS1338READ_ADDR); //Adres odczytu układu DS1338 (bit W/R = 1)
    temp = TWI_ReadByte(ACK);  //Sekundy w zapisie BCD
    Data->Second = ((temp>>4)*10) + (temp&0x0F);
    temp = TWI_ReadByte(ACK);  //Minuty w zapisie BCD
    Data->Minute = ((temp>>4)*10) + (temp&0x0F);
    temp = TWI_ReadByte(ACK);  //Godziny w zapisie BCD
    Data->Hour = ((temp>>4)*10) + (temp&0x0F);
    temp = TWI_ReadByte(ACK);  //Pomijany dzień tygodnia
    temp = TWI_ReadByte(ACK);  //Dzień w zapisie BCD
    Data->Day = ((temp>>4)*10) + (temp&0x0F);
    temp = TWI_ReadByte(ACK);  //Miesiąc w zapisie BCD
    Data->Month = ((temp>>4)*10) + (temp&0x0F);
    temp = TWI_ReadByte(NACK);  //Rok w zapisie BCD
    Data->Year = ((temp>>4)*10) + (temp&0x0F);
    TWI_Stop();
}
```

Listing 6. Deklaracje typu i definicje zmiennych związane z obsługą wbudowanej pamięci EEPROM mikrokontrolera

```
typedef union
{
    struct
    {
        uint8_t userID;
        uint8_t buttonAddress[8];
        char userName[8];
    };
    uint8_t index[17];
} userType;
userType User;  //Bufor tablicy Użytkownika
userType UserEE[50] EEMEM; //Dane użytkowników w pamięci EEPROM mikrokontrolera
```

Kończąc tematykę rozwiązań programowych przedstawię jeszcze dwa zagadnienia, które dość często pojawiają się na wszelkiego rodzaju forach poświęconych mikrokontrolerom, sprawiając problemy osobom początkującym w tej tematyce. Na początek przedstawię funkcję odpowiedzialną za odczytanie i weryfikację numeru klucza iButton. Nie będę jednak rozpisywał się, jak to zwyczajowo bywa, na temat poszczególnych funkcji narzędziowych niezbędnych dla inicjalizacji magistrali 1-wire (wysłanie sygnału RESET i kontrola obecności sygnału PRESENCE) czy odczytu/zapisu poszczególnych bitów, bo bez problemu znajdziemy darmowe implementacje tych funkcjonalności w Internecie (nie mówiąc już nawet o tym, że jest to zagadnienie dość proste) tylko zaprezentuję gotowe rozwiązanie dotyczące układów DS1990 w szczególności. Wspomnianą funkcję pokazano na **listingu 7**.

Przedmiotowa funkcja *uint8_t readIbutton(uint8_t *readID)* zwraca w wyniku swojego działania rezultat procesu odczytu: 0x00, gdy wszystko zakończyło się powodzeniem i OW_ERROR (0x01), gdy brak podłączonego klucza iButton, lub wystąpił błąd odczytu jego numeru ID. Jednocześnie w zmiennej tablicowej (8 bajtów), do której adres przekazano argumentem wywołania funkcji, znajdzie się odczytany numer ID. Warto wspomnieć, że użyta funkcja *_crc_ibutton_update()* wymaga dołączenia pliku nagłówkowego *crc16.h*,



Rysunek 3. Diagram prezentujący kompletny algorytm obsługi systemu menu sterownika systemu SAS

Listing 7. Funkcja odpowiedzialna za odczytanie i weryfikację numeru klucza iButton

```
uint8_t readIbutton(uint8_t *readID)
{
    register uint8_t idx, calculatedCRC8 = 0;
    //Sprawdzamy obecność układu iButton na magistrali 1wire - brak odpowiedzi pastylki zwróci OW_ERROR czyli 0x01
    if(ow_reset() == OW_ERROR) return OW_ERROR;
    //Wysyłamy rozkaz odczytu numeru ID
    ow_write_byte(OW_READ_ROM);
    //Odczytujemy 8 bajtów numeru ID. Ostatni bajt to suma kontrolna CRC8 z siedmiu pierwszych bajtów
    for(idx=0; idx<8; idx++) readID[idx] = ow_read_byte();
    //Na podstawie pierwszych 7-miu bajtów obliczamy własną sume kontrolną CRC8
    for(idx=0; idx<7; idx++) calculatedCRC8 = _crc_ibutton_update(calculatedCRC8, readID[idx]);
    //Porównujemy obliczoną sumę kontrolną CRC8 z 8-mym bajtem zawierającym przesłaną sumę kontrolną CRC8
    if(calculatedCRC8 != readID[7]) return OW_ERROR; else return 0x00;
}
```

gdyż właśnie tam znajduje się jej definicja. Drugie z zagadnień to implementacja wielojęzykowych systemów Menu. W tym przypadku, podobnie jak poprzednio, z pomocą przychodzą nam dostarczane, gotowe funkcje pakietu AVR-GCC, tym razem dostępne po dołączeniu pliku nagłówkowego *pgmspace.h*, który zawiera funkcje i makra upraszczające dostęp do pamięci Flash mikrokontrolera. W pierwszym kroku zdefiniujmy sobie kilka ciągów znakowych (każdy przechowujący ten sam napis, ale w różnym języku), tzw. c-stringów, w pamięci Flash mikrokontrolera posiłkując się specyfikatorem PROGMEM (dostępnym we wspomnianym wcześniej pliku nagłówkowym):

```
char menu1en[] PROGMEM = „EVENT
LIST”;
char menu2en[] PROGMEM = „USER
LIST”;
char menu3en[] PROGMEM = „SET
DATE/TIME”;
char menu1pl[] PROGMEM = „LISTA
ZDARZEN”;
char menu2pl[] PROGMEM = „LISTA
UZYTKOWNIKOW”;
char menu3pl[] PROGMEM = „USTAW
DATE/GODZINE”;
```

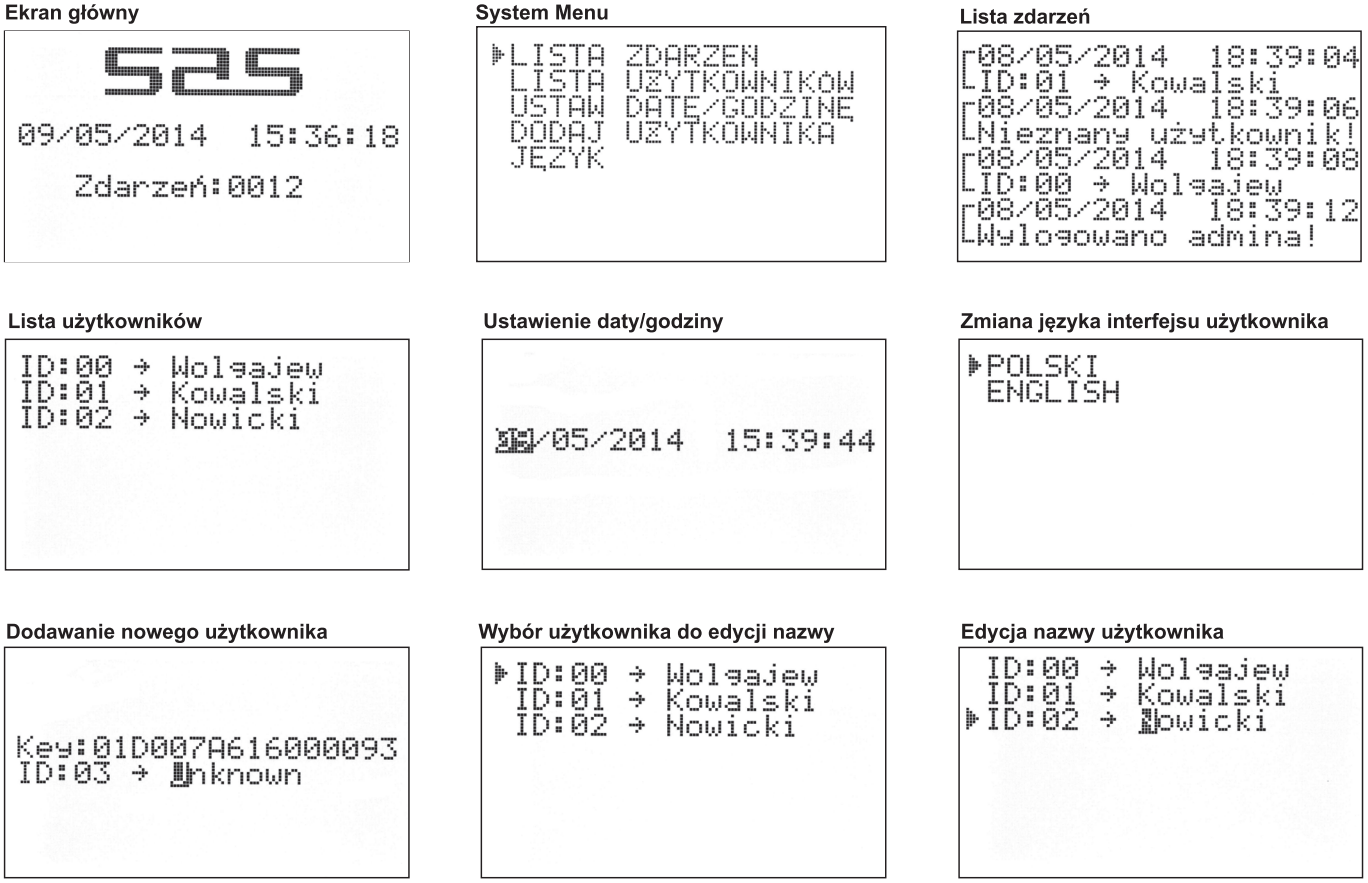
Teraz, zdefiniujemy dwuwymiarową, bo przygotowujemy 2 wersje językowe, tablicę w pamięci Flash, zawierająca wskaźniki do powyższych c-stringów *char* Texts[2][3] PROGMEM = {{menu1pl, menu2pl, menu3pl},{menu1en, menu2en, menu3en}};*. Załóżmy, w tym miejscu, że zdefiniowaliśmy specjalną zmienną *uint8_t* Language przechowującą aktualnie obowiązującą wersję językową oraz pewną funkcję o symbolicznej nazwie *lcd_str_P(const char *string)*, która realizuje wyświetlanie napisów z pamięci Flash, do których to wskaźnik zawiera argument jej wywołania opisany *const char *string*. Wywołanie naszej symbolicznej funkcji *lcd_str_P((**char** *)pgm_read_word(&Texts[Language][1]));* spowoduje wyświetlenie napisu „*USER LIST*” lub „*LISTA UZYTKOWNIKOW*”, w zależności od wartości zmiennej *Language*.

Co robi wbudowana funkcja *pgm_read_word*? Funkcja ta powoduje odczytanie słowa (*word* = 16 bitów) z pamięci Flash mikrokontrolera spod adresu podanego jako argument jej wywołania (w tym wypadku jest to wskaźnik na element tablicy *Texts*). Czemu słowa a nie bajtu? To proste. Adresy w mikrokontrolerach AVR tego typu są wartościami 16-bitowymi i stąd potrzeba zastosowania funkcji *pgm_read_word* a nie dla przykładu *pgm_read_byte*. Jednocześnie, aby nie nastąpiła niezgodność typów docelowych wskaźników, musimy wykonać rzutowanie typu odczytanej wartości na wskaźnik na typ char, co symbolizuje zapis (*char* *). W ten prosty sposób możemy przygotować wielojęzykowy system menu, w całości umieszczony w pamięci Flash mikrokontrolera, a więc nieangażujący nadmiernie pamięci RAM. Prawda, że proste? W tym miejscu widać raz jeszcze, że warto drobiazgowo przestudiować dokumentację pakietu AVR-GCC.

Obsługa

Projektując Menu systemu SAS oraz sposób obsługi tegoż urządzenia przyjąłem, że ergonomia i prostota jego użytkowania jak i czytelność interfejsu użytkownika powinna być najważniejszym kryterium przy konstruowaniu stosownych procedur sterujących. Zgodnie z tymi podstawowymi założeniami, na płytce sterownika przewidziano aż 7 przycisków sterujących dających bezpośredni dostęp do podstawowej funkcjonalności. Jako, że Menu obsługi urządzenia udostępnia wiele funkcji, stosowne przyciski maja różnoraką funkcjonalność zależną od miejsca w układzie Menu, przy czym ich podstawowe znaczenie przedstawia się następująco:

- Przyciski oznaczone jako ***NEXT, PREV*** służą do zmiany pozycji edytowanego elementu.
- Przyciski oznaczone jako ***UP***, ***DOWN*** służą do zmiany wartości edytowanego elementu oraz poruszania się po kolejnych opcjach Menu urządzenia.
- Przycisk oznaczony jako ***OK*** służy do zatwierdzania wyboru zarówno w zakresie opcji Menu jak i zmiany parametrów poddawanych edycji.
- Przycisk oznaczony jako ***MENU*** służy do wejścia w system Menu jak i wyjścia do Menu nadrzędnego bez zmiany parametrów edytowanego elementu (w przypadku Menu umożliwiającego edycję). Dodatkowo, długie



Rysunek 4. Wygląd ekranu interfejsu użytkownika dla głównych trybów pracy menu sterownika systemu SAS

Ustawienia ważniejszych fusebitów:
CKSEL3...0: 0100
SUT1...0: 10
JTAGEN: 1
EESAVE: 0

przytrzymanie tego przycisku powoduje każdorazowo wyjście do ekranu głównego aplikacji.

- Przycisk oznaczony jako ***ADMIN*** służy do realizacji funkcji administracyjnych: kasowania całej pamięci zdarzeń, kasowania całej pamięci definicji użytkowników i wejścia w tryb zapamiętywania klucza Administratora. Wszystkie wymienione operacje możliwe są do wykonania wyłącznie podczas włączania urządzenia, zaś sam przycisk ***ADMIN*** z założenia nie powinien być dostępny na panelu obsługi urządzenia (stąd też zastosowano dla niego przełącznik microswitch z bardzo krótką oską).

Dodatkowo w programie obsługi aplikacji przyjęto następujące założenia:

- Zmiana ustawień zegara czasu rzeczywistego RTC, edycja nazwy oraz usuwanie zdefiniowanego wcześniej użytkownika możliwe jest wyłącznie po zalogowaniu klucza Administratora.
- Usuwanie zdefiniowanych wcześniej użytkowników możliwe jest wyłącznie wtedy, gdy na liście zdarzeń nie zanotowano żadnego zdarzenia z identyfikatorem usuwanego użytkownika.
- Nie jest możliwe usuwanie zdarzeń z listy zdarzeń, zaś sama lista ma charakter kołowy, tzn. gdy wyczerpana zostanie maksymalna liczba zarejestrowanych zdarzeń (1000), nowo rejestrowane zdarzenia zastępują zdarzenia najstarsze.
- Rejestrowane są również zdarzenia zalogowania/wylogowania Administratora jak i zdarzenia prób logowania niezarejestrowanych użytkowników (używających niezarejestrowanych kluczy).
- Wbudowano mechanizm samowylogowywania się klucza Administratora po czasie 60 sekund, co zapobiega nieautoryzowanym zmianom, w przypadku niewylogowania klucza Administratora.
- Wbudowano mechanizm automatycznego powrotu do ekranu głównego urządzenia po czasie 30 sekund bezczynności ze strony użytkownika.
- Dodawanie nowego użytkownika poprzedzone jest każdorazowym sprawdzeniem obecności identyfikatora nowego klucza na liście użytkowników.
- W procesie dodawania nowego użytkownika przyznawany jest automatycznie najniższy, wolny identyfikator ID z listy użytkowników zachowanej w pamięci EEPROM mikrokontrolera.
- Obsługa logowania kluczy dostępna jest jedynie z Menu głównego urządzenia. Każda inna pozycja Menu automatycznie uniemożliwia sprawdzanie podłączanych kluczy iButton.
- Wbudowano mechanizm wygaszania podświetlenia wyświetlacza graficznego po czasie 30 sekund bezczynności klawiatury w celu ograniczenia poboru mocy.

W związku z powyższym aplikacja programu obsługi wyświetla dodatkowe komunikaty dotyczące następujących zdarzeń:

- Braku uprawnień do zmiany nastaw przeznaczonych wyłącznie dla Administratora.
- Istnieniu użytkownika na liście użytkowników o numerze klucza, który ma zostać dodany jako nowy.
- Osiągnięciu maksymalnej, dopuszczalnej liczby (50) użytkowników na liście zdefiniowanych użytkowników.
- Braku możliwości usunięcia użytkownika z listy użytkowników w przypadku istnienia identyfikatora tegoż użytkownika na liście zdarzeń.
- Zalogowaniu i wylogowaniu (w tym automatycznym wylogowaniu) klucza administratora.
- Wyczyszczeniu listy zdarzeń, listy użytkowników lub zapamiętaniu klucza administratora (wyłącznie podczas włączania urządzenia).
- Braku zdarzeń na liście zdarzeń.
- Braku użytkowników na liście użytkowników.

Ponieważ lista dostępnych opcji systemu Menu urządzenia SAS jest dość obszerna, na **rysunku 3** pokazano diagram prezentujący kompletny algorytm obsługi, natomiast na **rysunku 4** wygląd ekranu interfejsu użytkownika dla głównych trybów pracy systemu Menu.

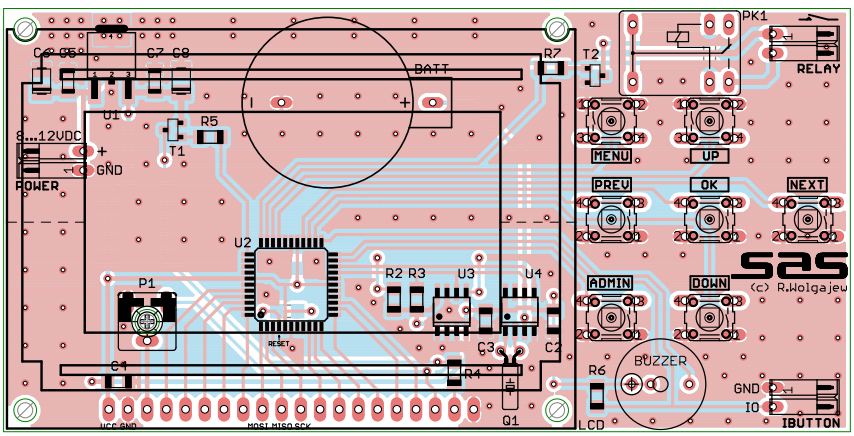
Montaż

Schemat montażowy systemu SAS pokazano na **rysunku 5.** Zaprojektowano bardzo zwartą konstrukcję obwodu drukowanego ze zdecydowaną przewagą elementów SMD po to, aby wymiary całego urządzenia nie przekraczały niezbędnego, minimalnego obszaru dla wykonania interfejsu użytkownika. W celu zminimalizowania zakłóceń na płytce poprowadzono obszerne pola masy po obu stronach obwodu drukowanego oraz zastosowano szereg przelotek pomiędzy nimi w celu zmniejszenia pojemności pasożytniczych. Warto podkreślić, iż elementy SMD oznaczone C1, R1 i D1 należy zamocować po stronie wyprowadzeń (BOTTOM).

Z uwagi na zastosowanie niewielkich elementów SMD montaż najlepiej przeprowadzić z użyciem stacji lutowniczej wyposażonej w grot o niewielkiej średnicy, odpowiedniej jakości topników lutowniczych oraz mając spore doświadczenie. Montaż rozpoczynamy od przylutowania wszystkich układów scalonych, tranzystorów oraz rezonatora kwarcowego. Następnie lutujemy diody, rezystory, kondensatory a na końcu mikroprzyciski oraz pozostałe elementy, w tym gniazdo baterii CR2032, buzzer, przekaźnik, złącza oraz potencjometr regulacji kontrastu wyświetlacza graficznego (P1). Wyświetlacz należy zamocować w odpowiedniej odległości od obwodu drukowanego, najlepiej za pomocą tulei dystansowych, wykorzystując przewidziane w tym celu otwory, zaś same połączenie należy wykonać przy użyciu listwy goldpin (gniazdo-wtyk) lub zwykłej taśmy wieloprzewodowej.

Poprawnie zmontowany układ nie wymaga żadnych regulacji (poza regulacją kontrastu wyświetlacza LCD) i powinien działać tuż po włączeniu zasilania. Jedynym zabiegiem, jaki należy wykonać przed użytkowaniem urządzenia jest sformatowanie i wyczyszczenie pamięci FRAM, do czego przewidziano odpowiednią opcję Menu (dostępną wyłącznie podczas włączania urządzenia).

Robert Wołgajew, EP



Rysunek 5. Schemat montażowy sterownika systemu SAS