Watch

DODATKOWE MATERIAŁY NA FTP:
ftp://ep.com.pl
USER: 60086, PASS: sjh7zycq

W ofercie AVT*
AVT-5525 A, UK

Podstawowe informacje:
- Napięcie zasilania: 5 V (USB).
- Maksymalny prąd obciążenia (wyświetlacz załączony/wyłączony): 20 mA/1 mA.
- Średni czas pracy na zasilaniu akumulatorowym: 7 dni.
- Zegar, kalendarz, stoper, budzik (z planem tygodniowym), barometr, termometr, prognoza pogody.
- Płytka przystosowana do zamocowania paska zegarka.

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

AVT-5522	Zegar ustawiany za pomocą GPS (EP 9/2015)
AVT-3132	Prosty zegar LED (EdW 7/2015)
AVT-5377	Mega stoper – wielofunkcyjny licznik, nie tylko czasu (EP 12/2012)
AVT-513	Zegar ze stuletnim kalendarzem i termometrem (EP 10-11/2011)
AVT-5281	„Inteligentny" zegar z wyświetlaczem LED (EP 3/2011)
AVT-5273	Zegar cyfrowy z analogowym sekundnikiem (EP 1/2011)
AVT-2849	Tiny clock (EdW 1/2008)
AVT-2721	Mikroprocesorowy zegar (EdW 4/2004)
AVT-2632	Gigantyczny zegar (EdW 5/2002)
AVT-5022	Programowany zegar z DCF77 (EP 6-7/2001)
AVT-5002	Zegar cyfrowy z wyświetlaczem analogowym (EP 3/2001)

* Uwaga:
Zestawy AVT mogą występować w następujących wersjach:
AVT xxxx UK to zaprogramowany układ. Tylko i wyłącznie. Bez elementów dodatkowych.
AVT xxxx A płytka drukowana PCB (lub płytki drukowane, jeśli w opisie wyraźnie zaznaczono), bez elementów dodatkowych.
AVT xxxx A+ płytka drukowana i zaprogramowany układ (czyli połączenie wersji A i wersji UK) bez elementów dodatkowych.
AVT xxxx B płytka drukowana (lub płytki) oraz komplet elementów wymienionych w załączniku pdf
AVT xxxx C to nic innego jak zmontowany zestaw B, czyli elementy wlutowane w PCB. Należy mieć na uwadze, że o ile nie zaznaczono wyraźnie w opisie, zestaw ten nie ma obudowy ani elementów dodatkowych, które nie zostały wymienione w załączniku pdf
AVT xxxx CD oprogramowanie (nieczęsto spotykana wersja, lecz jeśli występuje, to niezbędne oprogramowanie można ściągnąć, klikając w link umieszczony w opisie kitu)
Nie każdy zestaw AVT występuje we wszystkich wersjach! Każda wersja ma załączony ten sam plik pdf! Podczas składania zamówienia upewnij się, którą wersję zamawiasz! (UK, A, A+, B lub C). http://sklep.avt.pl

Zegarek naręczny (1)

Współczesny świat pędzi szybciej, niż kiedykolwiek wcześniej. Szybki rozwój technologii, elektroniki i najnowsze obszary ich zastosowań powodują, że już nie wyobrażamy sobie życia bez wielu urządzeń, które jeszcze do niedawna uważalibyśmy za zbędne gadżety. ***Rekomendacje:*** *własnoręcznie wykonany smartwatch przyciągnie spojrzenie wielu ciekawskich oczu.*

Po fazie euforii dotyczącej telefonii komórkowej nadeszła era wszechobecnych smartfonów, które to bardziej przypominają przenośne komputery niż telefony. Przychodzi mi na myśl obraz poranka w komunikacji miejskiej, gdy większość osób spogląda na ekrany swoich smartfonów, by w ten sposób być ciągle online i nie wypaść „z obiegu". Wszak nie bez powodu powstało obiegowe stwierdzenie, że „*jeśli nie masz konta na portalu społecznościowym...to po prostu nie istniejesz*". Jako anegdotę mogę opowiedzieć, że mój dobry znajomy informatyk, wykształcony i posiadający wiele branżowych certyfikatów, na jednym z portali związanych z rozwojem kariery zawodowej, które to dają możliwość zaprezentowania własnego CV, w zakładce „*Inne osiągnięcia*" umieścił wpis „*Brak konta na Facebook'u*". Znamienne!

Nową falą wspomnianej mody, moim zdaniem napędzaną chęcią zysku producentów elektroniki użytkowej, jest moda na tak zwane smartwatch'e, czyli ni mniej, ni więcej, tylko mniejsze wersje smartfonów, które to udają zegarki naręczne. Nie rozumiem tego pędu i tęsknię za czasami, gdy ludzie wysyłali do siebie zwyczajne, papierowe listy, zaś wymianom poglądów towarzyszyły normalne spotkania. Mam nadzieję, że wróci to wcześniej czy później, tak jak wraca moda na urządzenia w stylu retro, czy tęsknota za tym, co miało swój styl i jakość. W świetle tego, tym bardziej nie rozumiem, po co w naręcznym zegarku integrować funkcjonalność telefonu komórkowego i komputera w jednym. Czyż nie wystarczyłoby, żeby był to po prostu zwykły, acz efektowny zegarek elektroniczny? Tak, jak za dawnych czasów, tylko we współczesnym wykonaniu?

Idąc tym właśnie tropem postanowiłem zbudować takie urządzenie, które to nazwałem po prostu „**Watch**". Nie ukrywam, iż przyczynkiem do powstania tego projektu było natknięcie się na doskonałej jakości, niewielki wyświetlacz OLED o przekątnej 0,96" i rozdzielczości

128×64px, który to można kupić za tak niewygórowaną kwotę, że grzechem byłoby niewykorzystanie go w swoich konstrukcjach. Jakby tego było mało, na portalach aukcyjnych dostępne są „wygodne" moduły wykorzystujące tenże element, wyposażone w złącze goldpin (o różnej liczbie wyprowadzeń, w zależności od zastosowanego interfejsu komunikacyjnego SPI lub I²C), co znacznie ułatwia implementację we własnych urządzeniach. Sam wyświetlacz jest oferowany w różnych kolorach i wersjach, gdzie dla przykładu, pierwsze 16 linii (licząc od góry) jest w kolorze żółtym, a kolejne 48 w kolorze seledynowym (cyjan), co czyni go jeszcze bardziej atrakcyjnym wizualnie i właśnie tego typu element wykorzystano w opisywanym projekcie.

Sterownikiem ekranu, zastosowanym w każdej wersji tegoż wyświetlacza, jest układ firmy Solomon Systech Limited typu SSD1306. Sterownik ten jest łatwy w obsłudze programowej, choć warto zaznaczyć, że w wersji z interfejsem SPI (nasz przypadek) umożliwia wyłącznie zapis do sterownika ekranu, bez możliwości odczytu (wynika to z wyprowadzeń dostępnych na złączu goldpin). Co ważne, wybierając konkretny moduł dostępny w handlu należy zakupić wersję wyposażoną w następujące sygnały sterujące: **CLK** (sygnał zegarowy magistrali SPI), **MOSI** (wejście danych magistrali SPI), **RST** (wejście zerowania sterownika SSD1306) oraz **DC** (wejście, decydujące o charakterze wysyłanych danych: 1 → dane pamięci obrazu, 0 → rozkaz sterujący). Równie ważne jest rozmieszczenie sygnałów zasilających, jako że moduły dostępne w handlu mają częstokroć zamienione miejscami sygnały zasilania (VCC) i masy (GND). Reasumując, konfiguracja sterownika SSD1306 sprowadza się do ustawienia niezbędnych rejestrów sterujących, które to odpowiedzialne są za sprzętowe parametry układu wynikające z organizacji pamięci obrazu jak i właściwości obsługiwanego panelu OLED. Przechodząc do konkretów, zamieszczę w pierwszej kolejności funkcje odpowiedzialne za programową obsługę interfejsu SPI, w świetle wysyłania danych do pamięci obrazu i rozkazów sterujących, których to ciała pokazano na **listingu 1**.

Aby umożliwić obsługę zdefiniowanych przez użytkownika czcionek ekranowych wprowadzono nowy typ danych, którego definicję pokazano na **listingu 2**. Bazując na zdefiniowanej strukturze, wprowadzono funkcję, która korzystając z globalnej zmiennej *static FontDescription CurrentFont* pozwala na ustawienie bieżącej czcionki ekranowej, której ciało pokazano na **listingu 3**. Na **listingu 4** pokazano z kolei funkcję, która pozwala na inicjalizację sterownika OLED naszego panelu. Dalej, na **listingu 5** pokazano funkcje narzędziowe odpowiedzialne za: ustawienie aktywnego obszaru ekranu, w ramach którego przeprowadzany jest zapis do pamięci ekranu sterownika SSD1306, funkcję pozwalającą na ustawienie kontrastu wyświetlacza OLED oraz funkcję odpowiedzialną za wymazanie zawartości pamięci ekranu sterownika w zakresie współrzędnych zdefiniowanych argumentami wywołania tejże funkcji.

Czas na funkcje odpowiedzialne za rysowanie prostych elementów graficznych, to jest funkcję odpowiedzialną za wyświetlanie obrazków na ekranie wyświetlacza oraz funkcję pozwalającą na rysowanie znaków, z użyciem bieżącej czcionki ekranowej. Wspomniane funkcje pokazano na **listingu 6**. Na koniec dość nietypowa funkcja, której zadaniem jest wyświetlenie znaku, którego wzorzec przesunięty jest o zdefiniowaną parametrem wywołania funkcji liczbę pixeli w pionie (*uint8_t*pixelShift). Pozwala ona na „przewijanie" znaków na ekranie, dzięki czemu w dość łatwy sposób możemy uzyskać efekt animacji przypominający swoim działaniem pracę starych liczników mechanicznych, gdzie zmianie znaku towarzyszyło przesunięcie się jednego znaku w górę i „wskoczenie" na jego miejsce znaku kolejnego (w przypadku liczników były to oczywiście cyfry). Efekt taki wykorzystano w implementacji funkcji stopera naszego urządzenia i muszę

przyznać, że wygląda to nadspodziewanie efektownie. Ciało funkcji zamieszczono na **listingu 7**. Na koniec listing pliku nagłówkowego związanego z obsługą naszego wyświetlacza OLED, bez którego trudno byłoby zrozumieć działanie wcześniej przedstawionych funkcji. Zawartość wspomnianego pliku nagłówkowego pokazano na **listingu 8**.

To tyle, jeśli chodzi o obsługę naszego, niezmiernie ciekawego wyświetlacza OLED. Przejdźmy zatem do tytułowego urządzenia, w którego konstrukcji musiałem zmierzyć się z kilkoma istotnymi kwestiami wynikającymi z przyjętych założeń konstrukcyjnych, jeśli chodzi o jego docelową funkcjonalność. Podstawowe założenia, jakie sobie postawiłem były następujące:

- Mały pobór mocy zapewniający długą pracę bez potrzeby ładowania wbudowanego akumulatora.
- Autonomiczne zasilanie akumulatorowe i możliwość ładowania z portu USB.
- Niewielkie wymiary.
- Nieskomplikowana budowa i mały koszt implementacji.

Listing 1. Funkcje odpowiedzialne za obsługę programowego interfejsu SPI

```
inline void SPIsendByte(uint8_t Byte)
{
    //Wysyłamy bajt do układu Slave przesuwając kolejne bity na wyjscie MOSI
począwszy od MSB
    //Zbocze rosnące na wyprowadzeniu SCK powoduje zatrzasnięcie kolejnego
bitu - CLK_TICK = 0->1->0
    if(Byte&0x80) SET_MOSI; else RESET_MOSI; CLK_TICK;
    if(Byte&0x40) SET_MOSI; else RESET_MOSI; CLK_TICK;
    if(Byte&0x20) SET_MOSI; else RESET_MOSI; CLK_TICK;
    if(Byte&0x10) SET_MOSI; else RESET_MOSI; CLK_TICK;
    if(Byte&0x08) SET_MOSI; else RESET_MOSI; CLK_TICK;
    if(Byte&0x04) SET_MOSI; else RESET_MOSI; CLK_TICK;
    if(Byte&0x02) SET_MOSI; else RESET_MOSI; CLK_TICK;
    if(Byte&0x01) SET_MOSI; else RESET_MOSI; CLK_TICK;
}

inline void SSD1306writeCmnd(uint8_t Command)
{
    RESET_DC;
    SPIsendByte(Command);
}

inlinevoid SSD1306writeData(uint8_t Data)
{
    SET_DC;
    SPIsendByte(Data);
}
```

Listing 2. Definicja nowego typu danych odpowiedzialnego za przechowywanie parametrów bieżącej czcionki ekranowej

```
//Deklaracja struktury przechowującej parametry bieżącej czcionki ekranowej
typedef struct
{
    uint8_tWidth;  //Rzeczywista szerokosc znaku (px)
    uint8_tHeight;  //Rzeczywista wysokosc znaku (bajty)
    uint8_tInterspace;  //Odstęp pomiędzy znakami (px)
    uint8_tBytesPerChar;  //Liczba bajtów danych tablicy wzorców na 1 znak
    uint8_tFirstCharCode;  //Kod ASCII pierwszego znaku
    constuint8_t *Bitmap;  //Wskaźnik to tablicy zawierającej wzorce
poszczególnych znaków
} fontDescription;
```

Listing 3. Funkcja odpowiedzialna za ustawienie bieżącej czcionki ekranowej

```
void OLEDsetFont(const fontDescription *Font)
{
    CurrentFont.Width = pgm_read_byte(&Font->Width);  //Rzeczywistaszerokoscczcionki
    CurrentFont.Height = pgm_read_byte(&Font->Height);  //Rzeczywistawysokoscczcionki
    CurrentFont.Interspace = pgm_read_byte(&Font->Interspace);  //Odstęp pomiędzy znakami
    CurrentFont.BytesPerChar = pgm_read_byte(&Font->BytesPerChar);  //Iloscbajtow na definicje pojedyńczego znaku
    CurrentFont.FirstCharCode = pgm_read_byte(&Font->FirstCharCode);  //Kod ASCII definicji pierwszego znaku
    CurrentFont.Bitmap = (uint8_t*)pgm_read_word(&Font->Bitmap);  //Wskaźnik do tablicy wzorców tej czcionki
}
```

- Integracja następującej funkcjonalności: zegar, kalendarz, stoper, budzik (z planem tygodniowym), barometr, termometr, prognoza pogody.
- Łatwość obsługi przy udziale minimalnej liczby elementów sterujących.

Tak oto powstał projekt urządzenia „Watch", którego schemat ideowy pokazano na **rysunku 1**. Jego „sercem" jest niewielki mikrokontroler Atmega168 taktowany wewnętrznym, wysokostabilnym generatorem RC o częstotliwości 1 MHz (dla zmniejszenia poboru mocy) odpowiedzialny za realizację pełnej, założonej funkcjonalności urządzenia. Mikrokontroler steruje pracą wyświetlacza OLED, zaś za pomocą wbudowanego interfejsu I²C (nazywanego TWI w wykonaniu firmy Atmel) realizuje współpracę ze scalonym barometrem firmy *Bosch Sensortec* pod postacią układu scalonego BMP180. Wybór tego, konkretnego typu scalonego barometru z szerokiej palety układów dostępnych na rynku podyktowany był faktem, iż w jego obudowie zintegrowano dokładny termometr scalony.

Dociekliwy Czytelnik zapewne dostrzeże fakt braku jakiegokolwiek układu realizującego funkcjonalność zegara czasu rzeczywistego (RTC), który to przecież jest podstawową funkcjonalnością, jaką realizuje nasz projekt. Rzeczywiście, na etapie projektowania systemu zastanawiałem się nad zastosowanie jednego ze znanych i tanich układów zegarów RTC pracujących na magistrali I²C, lecz

Listing 4. Funkcja odpowiedzialna za inicjalizację sterownika SSD1306

```
void OLEDinit(void)
{
    //Wszystkie porty jako wyjściowe - domyślnie stan „0", reset układu SSD1306
    OLED_DDR |= (1<<OLED_CLK)|(1<<OLED_MOSI)|(1<<OLED_RST)|(1<<OLED_DC);
    _delay_ms(1);
    SET_RST;  //Koniecresetu
    _delay_ms(10);
    SSD1306writeCmnd(DISPLAY_OFF_CMD);
    SSD1306writeCmnd(SET_START_LINE_CMD | 0x00);
    SSD1306writeCmnd(SET_CONTRAST_CMD);
    SSD1306writeCmnd(0xCF); //Contrast
    SSD1306writeCmnd(SET_SEG_REMAPING_CMD | 0x01);
    SSD1306writeCmnd(SET_COM_SCAN_REMAPPED_CMD);
    SSD1306writeCmnd(NORMAL_DISPLAY_CMD);
    SSD1306writeCmnd(SET_MULTIPLEX_RATIO_CMD);
    SSD1306writeCmnd(0x3F);  //1/64 duty
    SSD1306writeCmnd(SET_DISPLAY_OFFSET_CMD);
    SSD1306writeCmnd(0x00);  //Not offset
    SSD1306writeCmnd(SET_DISPLAY_CLOCK_DIV_CMD);
    SSD1306writeCmnd(0x80);  //Set divide ratio, Set Clock as 100 Frames/Sec
    SSD1306writeCmnd(SET_PRECHARGE_PERIOD_CMD);
    SSD1306writeCmnd(0xF1);  //Set Pre-Charge as 15 Clocks & Discharge as 1 Clock
    SSD1306writeCmnd(SET_COM_PINS_CMD);
    SSD1306writeCmnd(0x12);
    SSD1306writeCmnd(SET_VCOMH_DESELECT_CMD);//--set vcomh
    SSD1306writeCmnd(0x40);  //Set VCOM Deselect Level
    SSD1306writeCmnd(MEMORY_MODE_CMD);
    SSD1306writeCmnd(0x00);  //Horizontal addressing mode
    SSD1306writeCmnd(CHARGEPUMP_CMD);
    SSD1306writeCmnd(0x14);  //Disable
    SSD1306writeCmnd(DISPLAY_ALLON_RESUME_CMD);
    SSD1306writeCmnd(NORMAL_DISPLAY_CMD);
    SSD1306writeCmnd(DISPLAY_ON_CMD);
}
```

Listing 5. Funkcje narzędziowe sterownika SSD1306

```
//Column: 0...127, Page: 0...7
void OLEDsetActiveWindow(uint8_t startColumn, uint8_t startPage, uint8_t
endColumn, uint8_t endPage)
{
    SSD1306writeCmnd(SET_COLUMN_ADDR_CMD);
    SSD1306writeCmnd(startColumn);
    SSD1306writeCmnd(endColumn);
    SSD1306writeCmnd(SET_PAGE_ADDR_CMD);
    SSD1306writeCmnd(startPage);
    SSD1306writeCmnd(endPage);
}

void OLEDsetContrast(uint8_t Contrast)
{
    SSD1306writeCmnd(SET_CONTRAST_CMD);
    SSD1306writeCmnd(Contrast);
}

void OLEDclearArea(uint8_t startColumn, uint8_t startPage, uint8_t endColumn,
uint8_t endPage)
{
    registeruint16_tbytesToSend = (endColumn-startColumn+1)*(endPage-
startPage+1);
    OLEDsetActiveWindow(startColumn, startPage, endColumn, endPage);
    while(bytesToSend--) SSD1306writeData(0x00);
}
```

ostatecznie zrezygnowałem z tego pomysłu z kilku istotnych powodów. Po pierwsze, zastosowanie zewnętrznego zegara RTC zwiększyłoby koszt i pobór mocy całego układu, co nie wpisywałoby się w założenia projektu. Po drugie i najważniejsze, jakikolwiek z dostępnych na rynku układów realizujących funkcjonalność zegara RTC nie zapewniałby możliwości zrealizowania funkcjonalności stopera, która to była jednym z założeń projektu. Jak rozwiązałem tenże problem? Bardzo prosto! Skorzystałem z możliwości wbudowanego w strukturę mikrokontrolera Atmega168 układu czasowo-licznikowego Timer2, która to pozwala na jego asynchroniczną pracę i taktowanie zewnętrznym, wysokostabilnym, zegarkowym rezonatorem kwarcowym (w naszym przypadku o częstotliwości 40 kHz) dołączonym

Wykaz elementów

Rezystory: (SMD 0603)
R1, R8: 10 kΩ
R2: 100 kΩ
R3: 12 kΩ
R4: 5,1 kΩ
R5: 47 kΩ
R6, R7: 4,7 kΩ
Kondensatory: (SMD 0603)
C1, C7…C13: 100 nF (X7R)
C2…C4, C6: 10 µF/10 V (SMD „A")
C5: 10 nF (X7R)
Półprzewodniki:
U1: MCP73832 (SOT-23-5)
U2: APE8865Y5-27-HF-3 (SOT-23-5)
U3: ATmega168PA (TQFP32)
U4: BMP180 (LGA7)
T1: AP2301AGN (SOT23)
D1: MBR0520L (SOD123)
Inne:
OLED: wyświetlacz graficzny OLED 0,96" ze sterownikiem SSD1306
L1: dławik 10 µH (SMD 0805)
Q1: rezonator 40 kHz
MENU, EXIT: przyciski SMD typu DTSM-31N
USB: gniazdo micro USB-B typu ATTEND 207A-BBA0-R
PIEZZO: sygnalizator piezoelektryczny LD--BZPN-2030
ACCU: akumulator Li-Po 3,7 V/250 mA CELLEVIA BATTERIES L502030

Listing 6. Funkcje odpowiedzialne za rysowanie prostych elementów graficznych (obrazków i znaków)

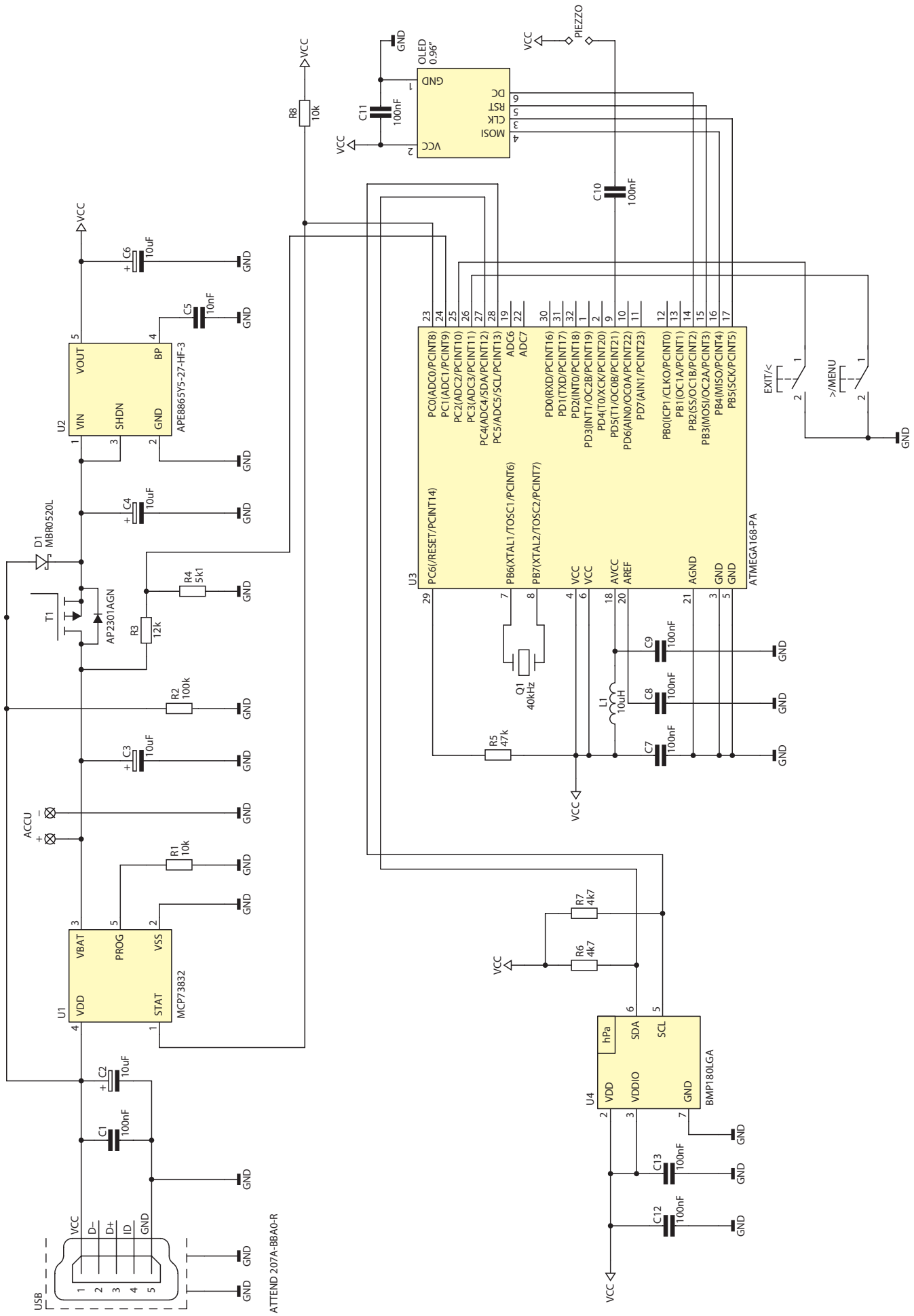
```
void OLEDdrawBitmap(uint8_t Column, uint8_t Page, constuint8_t *Bitmap)
{
    registeruint8_t Width, Height;
    registeruint16_tbytesToSend;
    Width = pgm_read_byte(Bitmap++);  //Pierwszybajttablicy Bitmap to szerokość: 0...127
    Height = pgm_read_byte(Bitmap++)>>3;  //Drugibajttablicy Bitmap to wysokość: 8, 16, 24...64 ->przeliczamynabajty
    bytesToSend = Width*Height;  //Liczba bajtów przeznaczonych do wysłania do OLEDa
    OLEDsetActiveWindow(Column, Page, Column+Width-1, Page+Height-1);
    while(bytesToSend--) SSD1306writeData(pgm_read_byte(Bitmap++));
}

void OLEDdrawChar(char Character, uint8_t Column, uint8_t Page, uint8_t Inverted)
{
    register uint8_treadByte;
    const uint8_t *dataPointer;
    register uint8_tbytesToSend;

    //Ustalamy adres początku wzorca znaku ASCII, który zamierzamy wyświetlić
    dataPointer = &CurrentFont.Bitmap[(CurrentFont.BytesPerChar*(Character-CurrentFont.FirstCharCode))];
    //Określamy okno zapisu by uprościć samą procedurę zapisu
    OLEDsetActiveWindow(Column, Page, Column+CurrentFont.Width-1, Page+CurrentFont.Height-1);
    //Określamy liczbę bajtów do wysłania
    bytesToSend = CurrentFont.BytesPerChar;
    while(bytesToSend--)
    {
        if(Character == ' ') readByte = 0x00; elsereadByte = Inverted? ~pgm_read_byte(dataPointer++):pgm_read_
byte(dataPointer++);
        SSD1306writeData(readByte);
    }
}
```

Listing 7. Funkcja pozwalająca na animację znaków

```
void OLEDdrawCharShifted(char Character, uint8_t Column, uint8_t Page, uint8_tpixelShift)
{
    constuint8_t *dataPointer;
    registeruint8_tbytesToSend, rowsToShift, prevByte, nextByte;
    //Ustalamy adres początku wzorca znaku ASCII, który to znak zamierzamy wyświetlić
    dataPointer = &CurrentFont.Bitmap[(CurrentFont.BytesPerChar*(Character-CurrentFont.FirstCharCode))];
    //Przesuwamy tenże wskaźnik o wartość pełnych strony, jeśli argument pixelShift jest większy od 7
    dataPointer += (pixelShift / 8) * CurrentFont.Width;
    //Teraz obliczamy ilośćpixeli przesunięcia wzroca znaku w ramach pojedyńczej strony
    rowsToShift = pixelShift % 8;
    //Określamy okno zapisu by uprościć samą procedurę zapisu do pamięci obrazu sterownika
    OLEDsetActiveWindow(Column, Page, Column+CurrentFont.Width-1, Page+CurrentFont.Height-1);
    //Określamy liczbę bajtów przeznaczonych do wysłania
    bytesToSend = CurrentFont.BytesPerChar;
    while(bytesToSend--)
    {
        prevByte = pgm_read_byte(dataPointer) >>rowsToShift;
        nextByte = pgm_read_byte(dataPointer + CurrentFont.Width) << (8-rowsToShift);
        SSD1306writeData(prevByte | nextByte);
        dataPointer++;
    }
}
```



Rysunek 1. Schemat ideowy urządzenia Watch.

do wyprowadzeń TOSC1/TOSC2. Dzięki temu, moduł Timer2 taktowany jest za pomocą wspomnianego rezonatora kwarcowego zupełnie niezależnie od sygnału zegarowego (1 MHz) taktującego procesor, co zapewnia realizację bardzo dokładnego zegara czasu rzeczywistego. Co oczywiste, w takim wykonaniu niezbędne było napisanie odpowiednich funkcji, które realizują całą, założona funkcjonalność zegara.

Listing 8. Listing pliku nagłówkowego związanego z obsługą wyświetlacza OLED opartego o sterownik ekranu SSD1306

```
//OLED RESOLUTION
#define OLED_WIDTH 128
#define OLED_HEIGHT 64
//OLED port definitions (incl. SPI).
#define OLED_PORT PORTB
#define OLED_DDR DDRB
#define OLED_CLK PB5  //SERIAL CLOCK
#define OLED_MOSI PB4  //SERIAL INPUT
#define OLED_RST PB3  //CHIP RESET
#define OLED_DC PB2  //DATA(1)/COMMAND(0)
#define SET_CLK OLED_PORT |= (1<<OLED_CLK)
#define RESET_CLK OLED_PORT &= ~(1<<OLED_CLK)
#define SET_MOSI OLED_PORT |= (1<<OLED_MOSI)
#define RESET_MOSI OLED_PORT &= ~(1<<OLED_MOSI)
#define SET_CS OLED_PORT |= (1<<OLED_CS)
#define RESET_CS OLED_PORT &= ~(1<<OLED_CS)
#define SET_RST OLED_PORT |= (1<<OLED_RST)
#define RESET_RST OLED_PORT &= ~(1<<OLED_RST)
#define SET_DC OLED_PORT |= (1<<OLED_DC)  //DDRAM MODE
#define RESET_DC OLED_PORT &= ~(1<<OLED_DC)  //COMMAND MODE
#define CLK_TICK SET_CLK; RESET_CLK
//OLED Commands
#define SET_CONTRAST_CMD 0x81
#define DISPLAY_ALLON_RESUME_CMD 0xA4
#define DISPLAY_ALLON_CMD 0xA5
#define NORMAL_DISPLAY_CMD 0xA6
#define INVERSE_DISPLAY_CMD 0xA7
#define DISPLAY_OFF_CMD 0xAE
#define DISPLAY_ON_CMD 0xAF
#define SET_DISPLAY_OFFSET_CMD 0xD3
#define SET_COM_PINS_CMD 0xDA
#define SET_VCOMH_DESELECT_CMD 0xDB
#define SET_DISPLAY_CLOCK_DIV_CMD 0xD5
#define SET_PRECHARGE_PERIOD_CMD 0xD9
#define SET_MULTIPLEX_RATIO_CMD 0xA8
#define SET_START_LINE_CMD 0x40
#define MEMORY_MODE_CMD 0x20
#define SET_COLUMN_ADDR_CMD 0x21
#define SET_PAGE_ADDR_CMD 0x22
#define SET_COM_SCAN_NORMAL_CMD 0xC0
#define SET_COM_SCAN_REMAPPED_CMD 0xC8
#define SET_SEG_REMAPING_CMD 0xA0
#define CHARGEPUMP_CMD 0x8D
```

Nie było to zbyt trudne, więc tym bardziej dziwnym wydaje się fakt, iż nie spotkałem się dotychczas z tego typu rozwiązaniem wertując dziesiątki stron internetowych, więc zacznę od zaprezentowania pliku nagłówkowego związanego z modułem RTC, którego zawartość pokazano na **listingu 9**.

Teraz, pora na przedstawienie kilku kluczowych funkcji. Pierwsza z nich odpowiedzialna jest za inicjalizację Timera2 realizującego funkcjonalność zegara RTC oraz Timera0, który wykorzystany jest w naszym urządzeniu do generowania periodycznego przebiegu o częstotliwości 3 kHz (na wyprowadzeniu OC0B mikrokontrolera) w celu obsługi prostego sygnalizatora piezoelektrycznego (tzw. blaszkowego) używanego do emitowania dźwięku budzika. Ciało wspomnianej funkcji pokazano na **listingu 10**. Kolejna z funkcji, niezbędna przy realizacji zaawansowanego zegara RTC, to funkcja, która na podstawie argumentów wywołania (rok i miesiąc) ustala liczbę dni miesiąca uwzględniając fakt, czy dany rok jest rokiem przestępnym czy tez nie. Funkcję pokazano na **listingu 11**.

Funkcja *getMonthDaysLimit()* korzysta ze zmiennej *monthsDays[]* umieszczonej w pamięci programu, której to definicja jest następująca: *const uint8_t monthsDays[12] PROGMEM = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};*

Robert Wołgajew, EP

Listing 9. Listing pliku nagłówkowego związanego z realizacją zegara czasu rzeczywistego

```
//Definicje typów modułu odpowiedzialnych za obsługę timera, zegara i budzika
typedef struct
{
    volatile uint8_t Activity, Flag, Minute, Second, Hundredth;
} timerType;

typedef struct
{
    volatile uint8_t Flag, WeekDay, Year, Month, Day, Hour, Minute, Second;  //WeekDay: 0: Poniedziałek, 1: Wtorek...
} clockType;

typedefstruct
{
    volatile uint8_t Activity, WeekDays, Hour, Minute;  //WeekDays: bit0: Poniedziałek, bit1: Wtorek...
} alarmType;

//Zmienneglobalnemodułu
extern timerType Timer;
extern clockType Clock;
extern alarmType Alarm;
extern const uint8_t monthsDays[12];  //Liczba dni dla poszczególnych miesięcy
//Prototypyfunkcji
void megaRTCinit(void);
uint8_t getMonthDaysLimit(uint8_t Year, uint8_t Month);
//Definicje flag aktywności poszczególnych funkcjonalności zegara RTC
#define ACTIVE 1
#define INACTIVE 0
//Definicje flag związanych ze zmianą wartości zmiennych, które reprezentują
#define TIMER_ALL_VALUES 0b00111111
#define TIMER_MINUTE10 (1<<5)
#define TIMER_MINUTE (1<<4)
#define TIMER_SECOND10 (1<<3)
#define TIMER_SECOND (1<<2)
#define TIMER_HUNDREDTH10 (1<<1)
#define TIMER_HUNDREDTH (1<<0)
#define CLOCK_ALL_VALUES 0b01111111
#define CLOCK_WEEKDAY (1<<6)
#define CLOCK_YEAR (1<<5)
#define CLOCK_MONTH (1<<4)
#define CLOCK_DAY (1<<3)
#define CLOCK_HOUR (1<<2)
#define CLOCK_MINUTE (1<<1)
#define CLOCK_SECOND (1<<0)
//Obsługa Timera0 generującegoprzebieg 3000Hz
#define START_SOUND TCCR0B = (1<<CS00)
#define STOP_SOUND TCCR0B = 0
```