



# Androidowy zegar Nixie (1)

Na łamach czasopism elektronicznych oraz stronach internetowych opublikowano wiele projektów zegarków z lampami Nixie, w najróżniejszych konfiguracjach i opcjach. Były zarówno projekty oparte na popularnych mikrokontrolerach, były zegarki zbudowane na układach serii 74HC, a nawet zrealizowane na układach programowalnych. Jednak ten zegarek jest inny, ponieważ zastosowano w nim najnowsze zdobycze techniki, takie jak sensor laserowy lub sterowanie za pomocą smartfona.

**Rekomendacje:** zegarek może być ozdobą niejednego wnętrza, a ze względu na swoją funkcjonalność może stać się nietuzinkowym prezentem.

Na tle innych, podobnych urządzeń, opisany zegarek wyróżnia się brakiem przycisków mechanicznych, ponieważ sterowanie odbywa się za pomocą aplikacji dla systemu Android oraz laserowego czujnika zbliżeniowego. Wykorzystanie smartfona z dużym czytelnym wyświetlaczem jest o wiele wygodniejsze niż obsługa za pomocą przycisków mechanicznych oraz ładnego, aczkolwiek mało funkcjonalnego, wyświetlacza Nixie. Dodatkowo zyskujemy źródło czasu pochodzącego z sieci GSM, który będzie służył do synchronizacji zegara. Czujnik laserowy stanowi uzupełnienie interfejsu i umożliwia szybki dostęp do funkcji, takich jak wyświetlanie daty, godziny alarmu oraz zapewnia szybkie wyłączenie budzika. Rezygnacja z fizycznych przycisków daje możliwość wykonania estetycznej obudowy z pleksi lub szkła, pozbawionej otworów w widocznych miejscach. Zastosowanie skompensowanego generatora sygnału zegarowego 32 kHz umożliwi uzyskanie dobrej dokładności zegara, nawet przy niezbyt częstej synchronizacji z telefonem.

W urządzeniu zastosowano najpopularniejsze i najłatwiej dostępne w kraju lampy LC-513 produkcji zakładów Unitra Dolam o wielkości cyfr 15,5 mm. Lampy i podstawki są stosunkowo łatwo dostępne na aukcjach internetowych, więc każdy zainteresowany może wykonać zegarek we własnym zakresie.

Średnia trwałość lamp LC-513 jest określona na około 100 tys. godzin. Dodatkowe funkcje, takie jak: automatyczne dostosowanie poziomu jasności do warunków oświetleniowych, możliwość zaprogramowania przedziału godzin w cyklu dobowym (wtedy wyświetlacz może być wyłączony) i funkcja odtruwania katod, pozytywnie wpływają na trwałość wyświetlacza.

## Budowa urządzenia

Zegar zbudowano w oparciu o dwie płytki drukowane: płytę główną zawierającą mikrokontroler oraz większość innych układów (rysunek 1) oraz płytę wyświetlacza z układem sterującym (rysunek 2). Całością steruje mikrokontroler STM32F103R8T6 (U5) w obudowie TQFP64, zawierający 64 kB pamięci Flash oraz 20 kB pamięci RAM. Wybór tego układu był podyktowany jego niską ceną oraz wewnętrznym zegarem RTC, który jest 32-bitowym licznikiem zwiększającym zawartość co 1 sekundę. Dzięki takiemu rozwiązaniu układ RTC pozwala odmierzać czas systemowy bezpośrednio w formacie UTS (Unix Time Stamp). Niestety, późniejsze wersje układów rodziny STM32 mają typowy zegar RTC zliczający w BCD, co zdaniem autora jest krokiem wstecz. Rdzeń mikrokontrolera jest taktowany wewnętrznym generatorem RC (8 MHz), natomiast licznik RTC jest taktowany za pomocą dodatkowego sygnału o częstotliwości

## DODATKOWE MATERIAŁY NA FTP:

<ftp://ep.com.pl>

USER: 44747, PASS: 3qwdwa8u

## W ofercie AVT\*

AVT-5582

## Podstawowe informacje:

Androidowy zegar Nixie ma następujące parametry:

- Wyświetlacz Nixie o wielkości cyfr 15,5 mm.
- Odchyłka autonomicznego pomiaru czasu max  $\pm 1$  min/rok.
- Automatyczna zmiana czasu lato/zima.
- Wygodne sterowanie za pomocą interfejsu Bluetooth i aplikacji dla systemu Android dostępnej w sklepie Play.
- Podstawowa obsługa pozbawiona przycisków mechanicznych za pomocą zbliżeniowego czujnika laserowego.
- Funkcja alarmu w trybie tygodniowym.
- Automatyczne sterowanie poziomem jasności w zależności od natężenia oświetlenia, z możliwością ustawienia stałego poziomu podświetlania z aplikacji.
- Efekt płynnego przejścia pomiędzy cyframi.
- Możliwość zaprogramowania przedziału godzin, kiedy zegar ma być wyłączony np. w nocy, aby nie przeszkadzać oraz przedłużać żywotność lamp.
- Algorytm automatycznego odtruwania katod, przedłużający żywotność lamp.
- Przyjemny dla ucha sygnał alarmu.
- Zasilanie z zasilacza wtyczkowego 12 V, pobór mocy ok. 2 W (przy 50% jasności).

## Projekty pokrewne na FTP:

(wymienione artykuły są w całości dostępne na FTP)

AVT-3141	Zegar Nixie z sekundami (EdW 6/2016)
AVT-3097	Zegar Nixie (EdW 7/2014)
AVT-5145	Zegar retro na lampach Nixie (EP 9/2008)

\* Uwaga! Elektroniczne zestawy do samodzielnego montażu.

### Wymagana umiejętność lutowania!

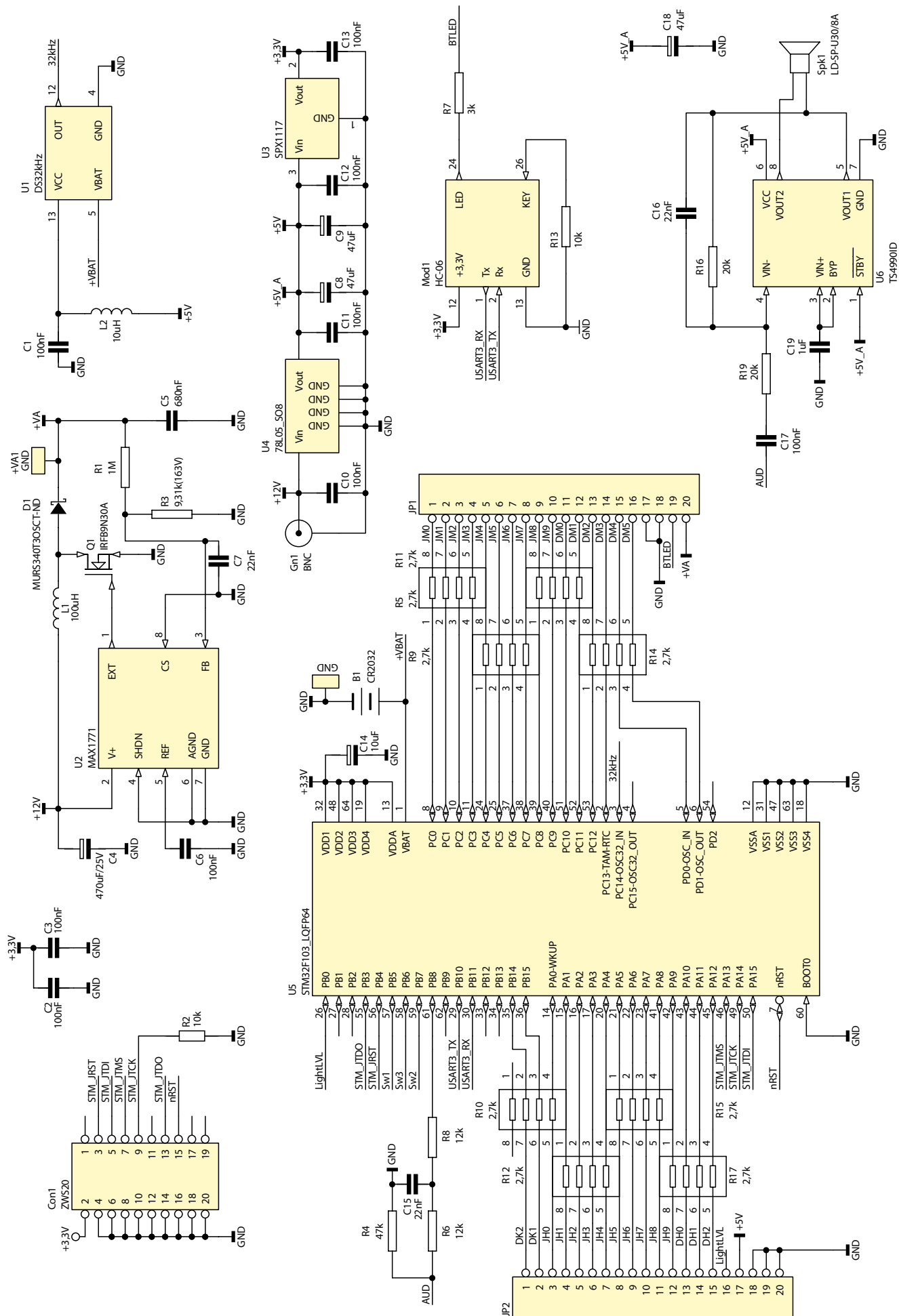
Podstawową wersją zestawu jest wersja [B] nazywana potocznie KiTem (z ang. zestaw). Zestaw w wersji [B] zawiera elementy elektroniczne (w tym [UK] – jeśli występuje w projekcie), które należy samodzielnie wlutować w dołączoną płytkę drukowaną (PCB). Wykaz elementów znajduje się w dokumentacji, która 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] płytką drukowaną bez elementów i dokumentacja

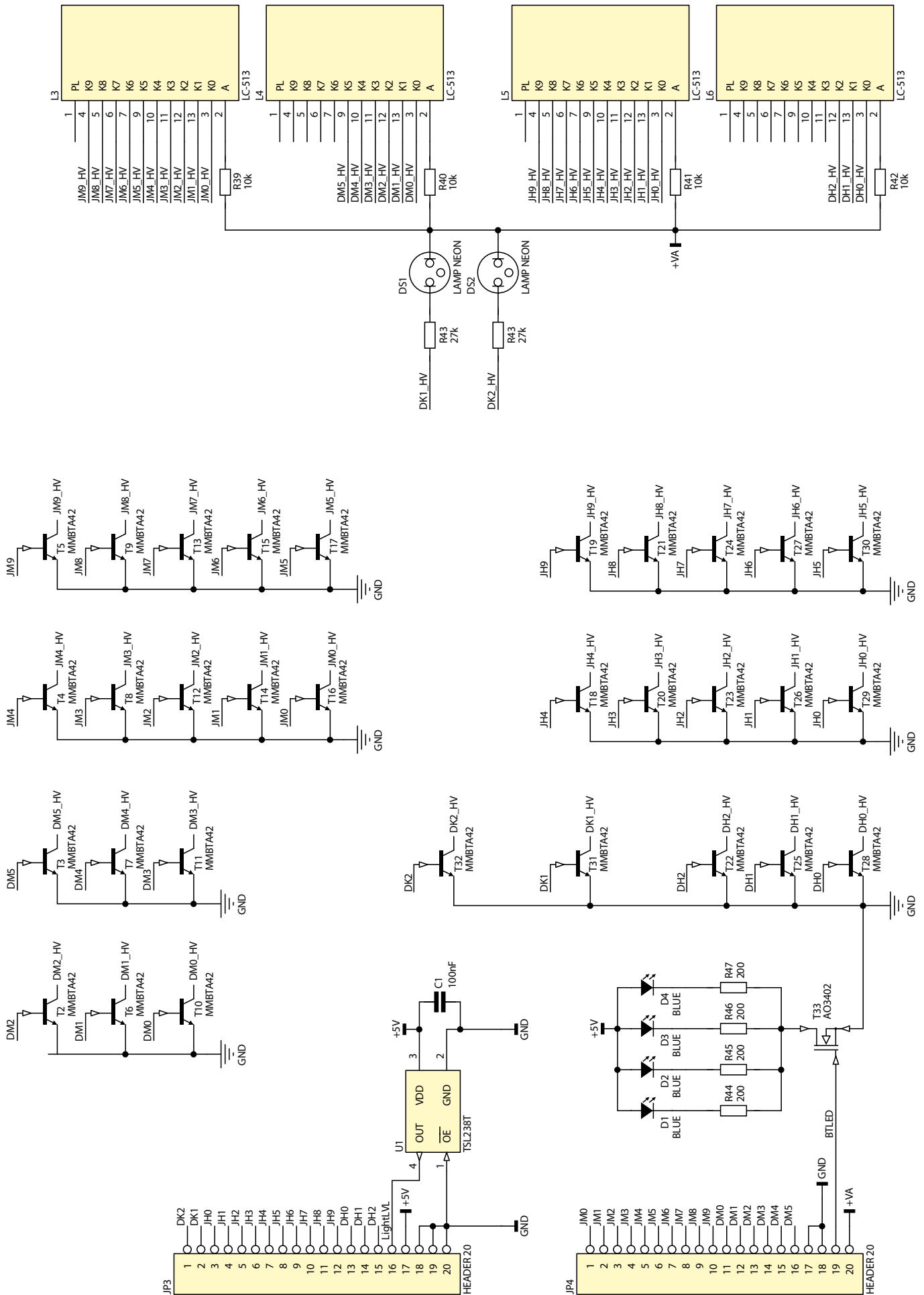
Kity w których występuje układ scalony wymagający zaprogramowania, posiadają następujące dodatkowe wersje:

- wersja [A+] płytką drukowaną [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>



Rysunek 1. Schemat ideowy płytki sterującej zegara



Rysunek 2. Schemat ideowy płytki wyświetlacza

32 kHz, dostarczanego z generatora DS32KHZ (U1), który jest skompensowanym temperaturowo generatorem kwarcowym zapewniającym maksymalną odchyłkę czasu 1 minutę na rok oraz charakteryzującym się poborem prądu rzędu 1  $\mu$ A.

Do linii portu szeregowego UART3\_TX, RX dołączono popularny moduł Bluetooth HC-05, który zapewnia komunikację ze światem zewnętrznym. Moduł ten jest konwerterem protokołu Bluetooth RFCOMM na RS232, który od strony mikrokontrolera może być sterowany za pomocą komend AT.

Do linii PB6 i PB7 stanowiących linie SDA, i SCL układu peryferyjnego I2C1 dołączono laserowy czujnik odległości VL6180x odpowiedzialny za wykrywanie gestów. Do portu PB5 dołączono sygnał przerwania GPIO0 czujnika, który jest używany do sygnalizowania zdarzeń generowanych przez czujnik, dzięki czemu nie ma konieczności ciągłego odpytywania układu.

Linia PB8, stanowiąca wyjście 3 sygnału PWM TIM3, za pomocą filtru dolnoprzepustowego złożonego z rezystora R8 i kondensatora C15 jest dołączona do wejścia wzmacniacza audio TS4490ID (U6). Wzmacniacz ten pracuje w układzie mostkowym w klasie AB i przy zasilaniu 5 V dostarcza sygnał o mocy 1 W przy obciążeniu 8  $\Omega$ .

Linie PB1 dołączono do linii STBY wzmacniacza, umożliwiając redukcję poboru prądu przez układ wzmacniacza do około 10 nA w stanie nieaktywności. Wyjście wzmacniacza dołączono do niewielkiego głośniczka LD-SP-U30/8A, który jest odpowiedzialny za generowanie dźwięku alarmów. Dzięki zastosowaniu trybu PWM do generowania audio możemy w pamięci Flash mikrokontrolera zapisać bardziej „przyjazne” dla ucha dźwięki niż przy zastosowaniu typowego buzzera.

Zasilacz niskiego napięcia zrealizowano z użyciem stabilizatorów liniowych. Układ

78L05 (U4) dostarcza napięcie +5 V potrzebne do zasilania wzmacniacza audio, natomiast układ SPX1117-3.3 jest odpowiedzialny za dostarczanie napięcia +3,3 V stanowiącego zasilanie mikrokontrolera. Do linii zasilającej VBAT mikrokontrolera, oraz generatora DS32KHZ dołączono zasilanie z baterii litowej CR2032 (B1), zapewniając podtrzymanie zliczania czasu po zaniku napięcia sieciowego. Blok zasilacza wysokonapięciowego dostarczającego napięcie +165 V, do zasilania lamp LC513, zrealizowano w topologii boost z zastosowaniem kontrolera MAX1171 (U1). Układ ten charakteryzuje się pracą z częstotliwością 300 kHz, zapewnia dużą sprawność przetwarzania (rzędu 90%) oraz bardzo mały pobór prądu rzędu 300  $\mu$ A w stanie bez obciążenia. Zawiera także zintegrowany sterownik zewnętrznego tranzystora N-MOSFET znacznie zmniejszający liczbę potrzebnych elementów zewnętrznych. Wyjście EXT steruje tranzystorem IRFB9N30A (Q1) o rezystancji kanału 0,45  $\Omega$ , napięciu wstecznym 300 V i prądzie przewodzonego 9 A. Cewka L1, dioda D1 i tranzystor Q1 stanowią fragment przetwornicy w topologii boost. Pętlę sprzężenia zwrotnego zrealizowano za pomocą rezystorów R1 i R3, natomiast napięcie wyjściowe jest filtrowane za pomocą kondensatora 680 nF/450 V (C5). Linie PA0-15, PB13-15, PC0-12, PD0-1 są używane do sterowania lampami Nixie. Za pomocą drabinek rezystorowych zostały dołączone do złączy JP1 i JP2, których użyto do połączenia płyty kontrolera z płytą wyświetlacza.

Na płycie wyświetlacza zamontowano układ TSL238T (U1) będący czujnikiem natężenia światła, który zapewnia automatyczny dobór jasności świecenia cyfr. Układ ten dostarcza sygnał wyjściowy o częstotliwości proporcjonalnej do natężenia oświetlenia, który poprzez złącze JP3 jest podawany na wejście PB0 mikrokontrolera.

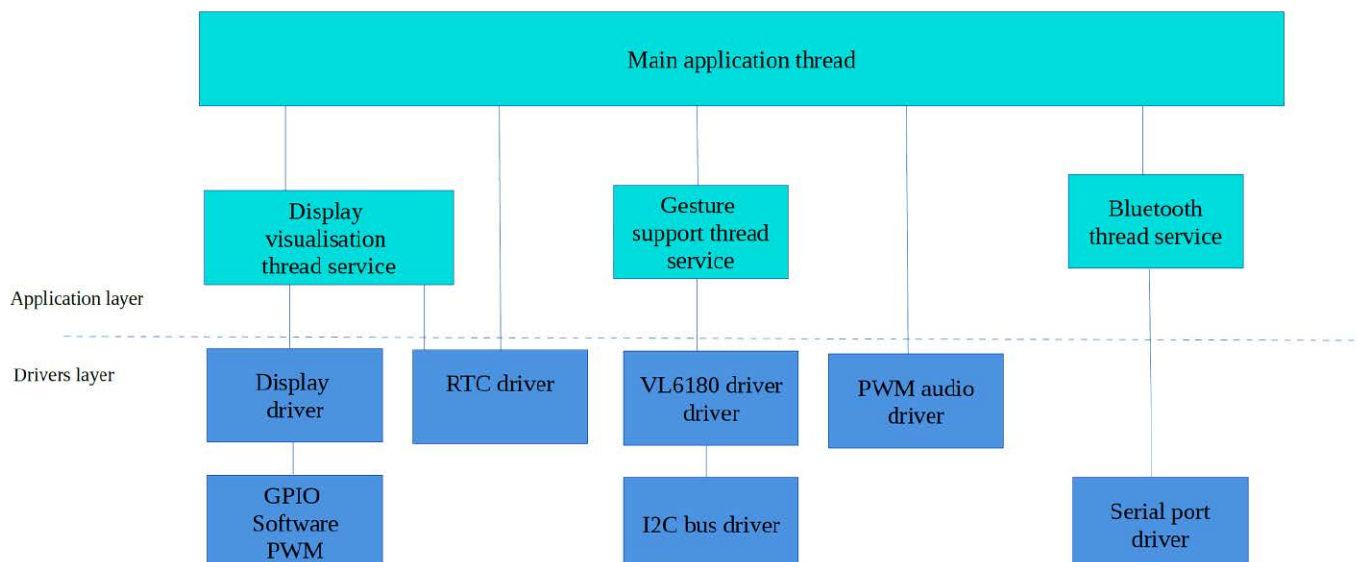
Choć czujnik laserowy ma zintegrowany czujnik światła dziennego, celowo zastosowano osobny czujnik umieszczony na froncie po to, aby mierzył poziom światła padającego bezpośrednio na wyświetlacze. Za sterowanie elektrodami Nixie są odpowiedzialne tranzystory bipolarne NPN typu MMBTA42 o maksymalnym napięciu wstecznym 300 V i prądzie przewodzenia do 100 mA. Wyświetlacze umieszczono w specjalnych podstawkach dla lamp LC513. Pomiędzy lampami L3, L4 a L5, L6 znajdują się dwie neonówki DS1 i DS2 umieszczona jedna nad drugą. Służą one do wyświetlania dwukropka pomiędzy minutami i godzinami.

## Oprogramowanie

Oprogramowanie mikrokontrolera napisano w języku C++14 za pomocą kompilatora `gcc` w wersji 6.2 pracującego pod kontrolą systemu operacyjnego ISIX-RTOS. Jest ono dostępne na otwartej licencji GPL. Oprogramowanie można pobrać z repozytorium znajdującego się pod adresem <https://www.boff.pl/cgit/public>.

Budowę firmware'u zegara przedstawiono na **rysunku 3**. Oprogramowanie podzielono na warstwę sterowników oraz warstwę aplikacji. Sterowniki są odpowiedzialne za obsługę: wyświetlacza Nixie, zegara RTC, czujnika laserowego, audio PWM oraz portu szeregowego.

Część aplikacyjną zrealizowano w postaci 4 wątków systemu ISIX. Wątek wizualizacji odpowiada za wyświetlanie danych na wyświetlaczu Nixie. W zależności od kontekstu jest to data, czas lub godzina alarmu. Wątek interpretacji gestów odczytuje stan czujnika VL6180 i w zależności od wykrycia odpowiednich gestów polegających na zbliżeniu ręki do czujnika i zatrzymaniu w pewnej odległości generuje zdarzenia, które są interpretowane przez wątek główny. Wykrywa on dwa rodzaje gestów: gest odczytu daty



Rysunek 3. Budowa firmware'u zegara

polegający na energicznym zbliżeniu ręki do czujnika i zatrzymaniu jej w odległości większej niż 5 cm oraz gest odczytu godziny alarmu polegający na energicznym zbliżeniu ręki i dotknięciu obudowy zegara.

Wątek Bluetooth jest odpowiedzialny za zarządzanie i konfigurację modułu HC-05 oraz interpretację rozkazów otrzymywanych z aplikacji zainstalowanej na smartfonie. Po otrzymaniu stosownego komunikatu generuje on zdarzenie interpretowane przez wątek główny aplikacji.

Wróćmy do opisu warstwy sterowników. Z uwagi na to, że w projekcie przyjęto założenie płynnej regulacji jasności oraz efekt płynnego przejścia między cyframi, sterownik wyświetlacza musi każdą cyfrą sterować za pomocą sygnału PWM. W mikrokontrolerach STM32 mamy limitowaną liczbę wyprowadzeń, które mogą pełnić funkcję wyjścia sprzętowego sygnału PWM, natomiast do sterowania wyświetlaczem potrzebujemy aż 27 linii wyjściowych. Co prawda, jedynie 5 linii jest aktywnych w tym samym czasie i można pokusić się o dołączenie zewnętrznych demultiplekserów, jednak niepotrzebnie skomplikowałoby to budowę urządzenia. W związku z powyższym zdecydowano się na rozwiązanie w postaci programowego generowania sygnału PWM. Za generowanie sygnału PWM odpowiada klasa *soft\_pwm* (plik *soft\_pwm.cpp/hpp*), która wykorzystuje sygnał przepełnienia od układu czasowo-licznikowego T1 do wytwarzania przerwania. Licznik skonfigurowano tak, aby przerwanie było wywoływane z częstotliwością 25,6 kHz. Przy 8-bitowym liczniku

PWM zapewnia to uzyskanie częstotliwości odświeżania wyświetlacza równej 100 Hz, co jest wartością wystarczającą, aby zapobiec nieprzyjemnemu migotaniu cyfr. Przerwanie odpowiedzialne za generowanie sygnału PWM jest wywoływane ze stosunkowo dużą częstotliwością i nie powinno być blokowane, dlatego zostało skonfigurowane z najwyższym priorytetem i może wywłaszczać pozostałe przerwania. Priorytet przerwania *T1\_OVERFLOW* znajduje się również powyżej priorytetu przerwania blokowanych za pomocą rejestru *BASEPRI* przy wejściu do sekcji krytycznych systemu ISIX, w związku z tym wszystkie operacje związane z tym przerwaniem nie mogą wykorzystywać API systemu, a jedyną możliwością synchronizacji i komunikacji jest wykorzystanie operacji atomowych. Procedura obsługi przerwania od układu czasowo-licznikowego i najważniejsze elementy programowego generowania PWM są przedstawione na **listingu 1**.

Każde utworzenie nowej instancji klasy *soft\_pwm* powoduje wywołanie konstruktora, który za pomocą zmiennej atomowej *g\_locked* blokuje dostęp do obiektu *std::list*, a następnie dopisuje na koniec listy wskaźnik na nowo utworzony obiekt klasy. Jeśli przy próbie dopisania elementu lista jest pusta oznacza to, że układ czasowo-licznikowy nie został jeszcze uruchomiony. W związku z tym jest wywoływana funkcja, której zadaniem jest konfiguracja układu czasowo-licznikowego T1, aby generował przerwanie z częstotliwością 25,6 kHz. Podobnie, gdy obiekt klasy *soft\_pwm* kończy swój cykl życia, zostaje wywołany destruktory, który

ustawia blokadę listy, a następnie kasuje element z listy odpowiadający temu obiektowi. Procedura obsługi przerwania *tim1\_up\_isr\_vector()* jest funkcją zaprzyjaźnioną z klasą *soft\_pwm* i cyklicznie z częstotliwością 25,6 kHz wywołuje metody *isr\_handler()* dla wszystkich zarejestrowanych instancji klas. Wektor przerwania został zdefiniowany z dodatkowymi atrybutami nakazującymi kompilatorowi włączenie dla tej funkcji agresywnej optymalizacji. Wykorzystanie listy obiektów podłączanych pod procedurę przerwania umożliwia elastyczne tworzenie instancji klas *soft\_pwm*, zapewniając generowanie dowolnej liczby sygnałów PWM. Publiczne API klasy *soft\_pwm* pokazano na **listingu 2**.

**Listing 1. Procedura obsługi przerwania od układu czasowo-licznikowego i najważniejsze elementy programowego generowania PWM**

```
namespace {
    //Vectors for PWM handler callbacks
    std::list<soft_pwm*> g_handlers;
    std::atomic<bool> g_locked;
    volatile soft_pwm::value_type m_pwm_cnt;
}

//! Constructor
soft_pwm::soft_pwm()
{
    g_locked = true;
    if( g_handlers.empty() )
    {
        timer_create();
    }
    g_handlers.push_back( this );
    g_locked = false;
}

//! Destructor
soft_pwm::~soft_pwm()
{
    g_locked = true;
    g_handlers.remove( this );
    if( g_handlers.empty() )
    {
        timer_destroy();
    }
    g_locked = false;
}

//ISR handler
extern "C" {
    __attribute__((interrupt,optimize(,"-O3"))) void tim1_up_isr_vector()
    {
        stm32::tim_clear_it_pending_bit( TIM1, TIM_IT_Update );
        if( g_locked ) return;
        for( auto h : g_handlers )
        {
            h->isr_handler();
        }
        ++m_pwm_cnt;
    }
}
```

**Wykaz elementów:  
Płytkę zegara**

- Rezystory:** (SMD 0805)  
 R1: 1 MΩ  
 R2, R13: 10 kΩ  
 R3: 9,31 kΩ  
 R4: 47 kΩ  
 R5, R9...R12, R14, R15, R17: 4×2,7 kΩ (drabinka rezystorowa)  
 R6, R8: 12 kΩ  
 R7: 3 kΩ  
 R16, R19: 20 kΩ  
**Kondensatory:** (SMD 0805)  
 C1...C3, C6, C10...C13, C17: 100 nF  
 C4: 470 μF/25 V  
 C5: 680 nF/400 V (THT)  
 C7, C15, C16: 22 nF  
 C8, C9, C18: 47 μF (SMD „D”)  
 C14: 10 μF (SMD)  
 C19: 1 μF  
**Półprzewodniki:**  
 D1: MURS340T3OSCT-ND (diody Schottky'ego, DO214AB)  
 Q1: IRFB9N30A (TO-220)  
 U1: DS32KHZ  
 U2: MAX1771 (SO8)  
 U3: SPX1117 (TO-252)  
 U4: 7805 (SO8)  
 U5: STM32F103 (LQFP64)  
 U6: TS4990ID (SO8)  
**Inne:**  
 B1: bateria CR2032 z gniazdem  
 L1: 100 μH (EPCOS 19×15)  
 L2: 10 μH (SMD 1206)  
 MOD1: moduł Bluetooth HC-06  
 SPK1: głośniczek LD-SP-U30/8A

**Płytkę wyświetlacza**

- Rezystory:**  
 R39...R42: 10 kΩ (SMD 1206)  
 R43: 27 kΩ (SMD 1206)  
 R44, R45, R46, R47: 200 Ω (SMD 0603)  
**Kondensatory:**  
 C1: 100 nF (SMD 0805)  
**Półprzewodniki:**  
 D1...D4: LED 5 mm  
 T2...T32: MMBTA42 (SOT-23)  
 T33: AO3402 (SOT-23, MOSFET N)  
 U1: TSL238T  
**Inne:**  
 JP3, JP4: złącze 2×10  
 L3...L6: lampa Nixie LC-513  
 DS1, DS2: neonówka

```

Listing 2. Publiczne API klasy soft_pwm
// !Replace pin and change pwm level
bool change_pin( const out& new_pin );
//! Change PWM bright value
void level( value_type pwm );
//! Set fast mode
void fast( bool m );

```

```

Listing 3. Publiczne metody klasy sterownika wyświetlacza
/** Put character on the display */
int putc( char ch );
/** Set display position */
int setpos( int x, int );
//! Set brightness
void brightness( bright_t val );
//! Get brightness
bright_t brightness() const;
//! Clear the display
void clear();

```

Wywołanie metody *change\_pin()* z referencją do obiektu *out* zawierającym nazwę i numer portu GPIO powoduje płynne zmniejszenie współczynnika wypełnienia sygnału PWM poprzednio przypisanego wyprowadzenia i jednocześnie płynne zwiększenie współczynnika wypełnienia dla nowego portu. Zastosowanie takiego triku umożliwia osiągnięcie ciekawego efektu wizualnego polegającego na płynnym efekcie przejścia z jednej cyfry w drugą. Metoda *level()* służy do ustawienia poziomu jasności wyświetlacza, należy tutaj podać wartości w zakresie od 0 do 255. Metoda *fast* umożliwia określenie, czy efekt przejścia między cyframi ma być szybki (0,1 s), czy powolny (1 s). Metoda *fast* ustawiana jest tylko dla sygnałów PWM odpowiedzialnych za dwukropki.

Klasa *soft\_pwm* jest klasą usługową wykorzystywaną przez właściwy sterownik wyświetlacza realizowany przez klasę *nixie\_disp* (pliki *nixie\_disp.cpp/hpp*). Sterownik wyświetlania udostępnia wysokopoziomowe API do obsługi cyfr. Na początku tworzy on 5 obiektów klas *soft\_pwm* odpowiadający odpowiednio cyfrom godzin, dwukropkowi oraz cyfrom minut *soft\_pwm m\_pwm[\_pwm\_eof\_]i*. Publiczne metody klasy sterownika wyświetlacza zgodne są z biblioteką strumieniową dla wyświetlaczy znakowych z *libfoundation* (listing 3).

Metoda *putc()* służy do wyświetlania znaku na bieżącej pozycji kursora, metoda *setpos()* ustawia kursor znaku na żądanej pozycji, metoda *brightness()* umożliwia ustawienie poziomu podświetlenia, natomiast

```

Listing 4. Fragmenty funkcji obsługi wyświetlacza
namespace {
//Display digit config
const constexpr out ports[][10]
{
    //H1
    { out(out::A,10), out(out::A,11), out(out::A,12) },
    //H2
    { out(out::A,0), out(out::A,1), out(out::A,2), out(out::A,3),
      out(out::A,4), out(out::A,5), out(out::A,6), out(out::A,7),
      out(out::A,8), out(out::B,13) },
    //M1
    { out(out::C,10), out(out::C,11), out(out::C,12), out(out::C,13),
      out(out::D,0), out(out::D,1) },
    //M2
    { out(out::C,0), out(out::C,1), out(out::C,2), out(out::C,3),
      out(out::C,4), out(out::C,5), out(out::C,6), out(out::C,7),
      out(out::C,8), out(out::C,9) },
    //DOT lo
    { out(out::B,15) },
    // Dot hi
    { out(out::B,14) },
};
//Off port marker
constexpr out port_off;

// Handle nixie digit
void nixie_disp::handle_digit( int val, size_t pos )
{
    const dev::out& d = val!=e_inval?ports[pos][val]:port_off;
    //! Wait for fadein fade out
    while ( m_pwm[pos].change_pin( d ) )
    {
        isix::wait_ms( 250 );
    }
}
}

```

metoda *clear()* zeruje wyświetlacz i ustawia kursor na pozycji początkowej. Pełna zgodność z biblioteką wyświetlaczy znakowych z *libfoundation* umożliwia wykorzystanie przeciążonego operatora „<<”. Naturalnie, z uwagi na ograniczenia samego wyświetlacza, na pozycjach 0, 1, 3, 4 mogą być wyświetlane tylko cyfry, natomiast na pozycji 2 odpowiadającej za dwukropek interpretowane są jedynie znaki „.”, „:” oraz spacja. Najciekawsze fragmenty implementacji klasy wyświetlacza przedstawiono na listingu 4.

Tablica *ports* jest tablicą obiektów *dev::out* o wymiarach 5×10 zawierającą mapowanie poszczególnych portów mikrokontrolera na odpowiadające cyfry wyświetlacza. Zadeklarowano ją jako wyrażenie stałe, znane podczas kompilacji, więc kompilator umieści ją w pamięci Flash. Klasa *out* przechowuje informacje o nazwie portu oraz numerze pinu w postaci prywatnego pola o wielkości jednego bajtu i zawiera metody *port()/pin()*, które umożliwiają zdekodowanie informacji do postaci akceptowanej przez funkcję *gpio\_set()/gpio\_clr()*. Wyświetlenie znaku na wybranej pozycji realizowane jest przez prywatną

metodę *handle\_digit()*. Działanie tej metody polega na wybraniu z tablicy odpowiedniego elementu oznaczającego numer linii GPIO na podstawie cyfry oraz pozycji, a następnie wywołaniu metody *change\_pin()* z odpowiadającego obiektu klasy *soft\_pwm*. Jak już wspomniano, priorytet obsługi przerwania realizującego PWM znajduje się powyżej priorytetu maskowania przerw sekcji krytycznych systemu, zatem nie ma możliwości realizacji powiadomień np. za pomocą semaforów systemowych. Aby zapobiec wykonaniu kolejnej sekwencji przenikania cyfr poprzez niespodziewanie rozpoczęcie nowego cyklu, metoda *change\_pin()* zwraca wartość **true**, gdy sterownik *soft\_pwm* znajduje się w trakcie realizacji algorytmu przenikania. Jeśli wywołanie metody *change\_pin()* nie powiodło się, wówczas podejmujemy kolejną próbę po odczekaniu 250 ms. Nie jest to rozwiązanie zbyt eleganckie, ale dzięki takiemu podejściu przerwanie realizujące PWM nie jest nigdy wstrzymywane.

Lucjan Bryndza, EP

Prenumerujesz  
„Elektronikę Praktyczną”  
i „Elektronikę dla Wszystkich”?

Masz prawo do  
bezpłatnej prenumeraty  
miesięcznika „Elektronik”  
w promocji 1+1=3

[www.avt.pl/prenumerata](http://www.avt.pl/prenumerata)

