

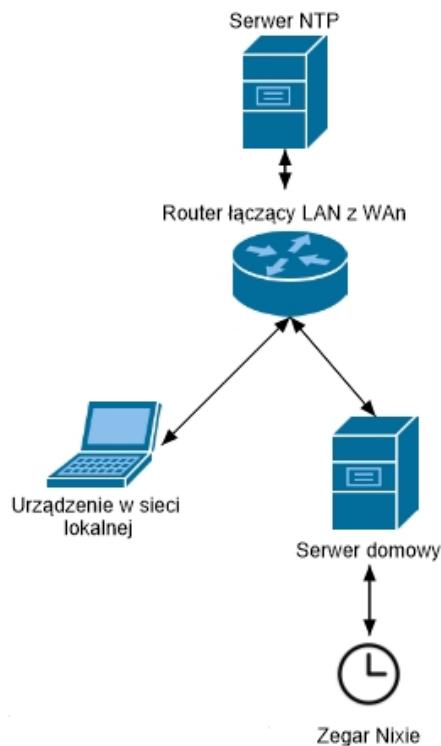
SPIS TREŚCI

1. Wprowadzenie.....	4
1.1. Cel i zakres pracy	4
1.2. Schemat blokowy układu	5
1.3. Opis działania układu i jego funkcjonalności	6
1.4 Wykorzystane protokoły komunikacji	7
2. Budowa zegara	8
2.1. Blok zasilania	8
2.2. Blok sterujący	10
2.3. Blok peryferyjny	15
2.4 Serwer domowy	19
3. Realizacja projektu	20
3.1. Zegar	20
3.2. Serwer	26
4. Eksperymenty i pomiary	28
4.1 Dobór częstotliwości zapalania lamp	28
4.2 Okres synchronizacji czasu	29
5. Uwagi końcowe	29
6. Repozytorium	30

1. WPROWADZENIE

1.1 Cel i zakres pracy

Celem pracy jest wykonanie zegara w stylu retro opartego na lampach elektronowych Nixie, mogącego pracować jako część infrastruktury inteligentnego domu. Oprócz zwykłych funkcji zegara posiada on możliwość synchronizacji czasu z domowym serwerem czasu poprzez interfejs Bluetooth (Rys.1.1).



Rys.1.1 Schemat struktury zarządzania zegarem

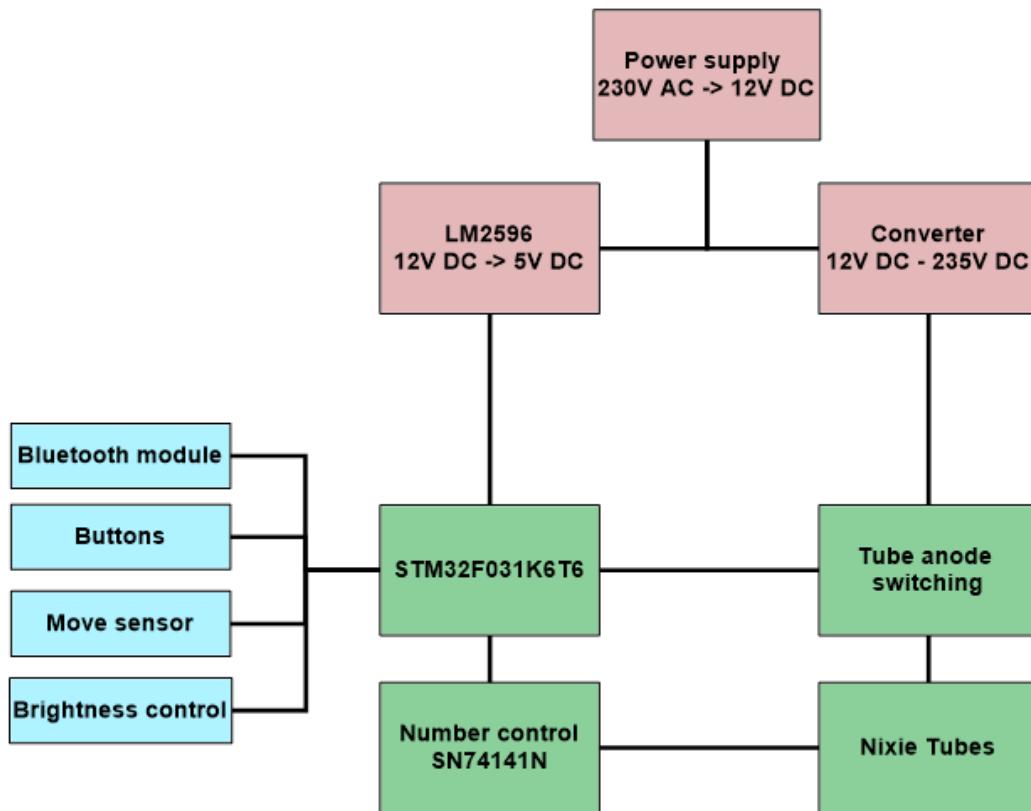
Do zakresu pracy należy:

- zaprojektowanie schematu elektronicznego zegara,
- implementacja oprogramowania mikrokontrolera zegara,
- fizyczne wykonanie, przetestowanie i uruchomienie zegara,
- zaimplementowanie testowego serwera domowego w celu obsługi funkcjonalności zegara.

1.2 Schemat blokowy układu

Na rysunku 1.2 znajduje się schemat blokowy zegara. Można wyróżnić trzy podstawowe bloki:

1. blok zasilania,
2. blok sterujący wyświetlaniem czasu,
3. blok układów peryferyjnych.



Rys. 1.2 Schemat blokowy zegara

Blok zasilania zapewnia źródło zasilania. Blok zamienia napięcie zasilania zegara (~230 V) na stabilizowane napięcie stałe (12 V). Napięcie 12 V jest następnie podawane na dwa konwertery napięcia DC:

- 12 V → 5 V; służący do zasilania płytka STM32 oraz układów peryferyjnych,
- 12 V → 235 V; służący do zasilania anody lamp Nixie.

Blok sterujący wyświetlaniem czasu zawiera:

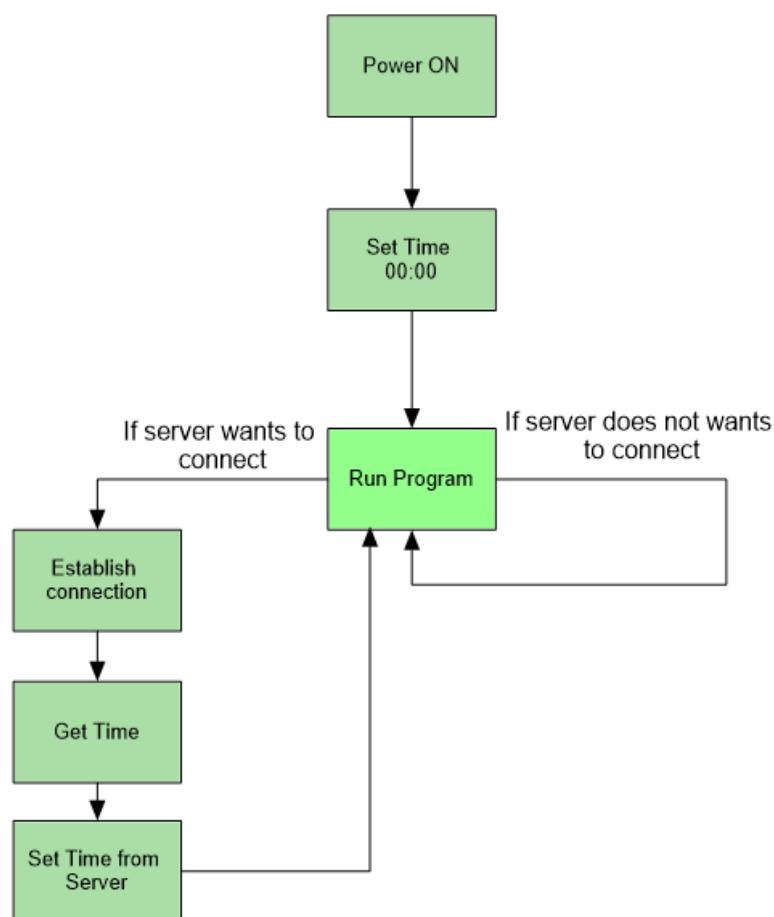
- mikroprocesor odpowiedzialny za sterowanie układu,
- układ przełączający napięcie 235 V pomiędzy anodami lamp,

- 4 bitowy dekoder służący do wysterowania odpowiedniej cyfry na lampie.

Blok układów peryferyjnych składa się z układów peryferyjnych odpowiedzialnych za komunikację z użytkownikiem i serwerem

1.3 Opis działania układu i jego funkcjonalności

Zegar składa się z czterech lamp elektronowych (Nixie), służących do wyświetlania czasu w postaci godzin i minut. W celu ustawienia prawidłowego czasu łączy się on z serwerem za pośrednictwem zewnętrznego modułu Bluetooth 4.0. Podczas startu i inicjalizacji zegara serwer wykrywa go i synchronizuje jego czas ze swoim czasem. W razie niepowodzenia zegar ustawia czas 00:00.



Rys.1.3 Diagram pracy zegara

Zegar posiada również:

- **dwa przyciski B0 i B1**- pozwalające na konfigurację czasu oraz alarmu (w przypadku braku połączenia z serwerem), włączanie/wyłączanie funkcji alarmu, oraz wybór trybu pracy ‘Move Mode’ / ‘Continuous Mode’,
- **czujnik ruchu** - wykorzystywany w trybie ‘Move Mode’,
- **potencjometr** - służący do sterowania jasnością świecenia lamp.

Przygotowano dwa tryby pracy zegara:

- ‘**Continuous Mode**’ - tryb ciągłego wyświetlania czasu/alarmu,
 - ‘**Move Mode**’ - tryb chwilowego wyświetlania czasu/alarmu uruchamianego poprzez ruch w obszarze czujnika ruchu.

Ustawienie czasu za pomocą przycisków zostało zaprogramowane w sposób umożliwiający wykorzystanie sekwencji naciśnięć przycisku tj. pojedyncze kliknięcie, podwójne kliknięcie, oraz przytrzymanie przycisku.

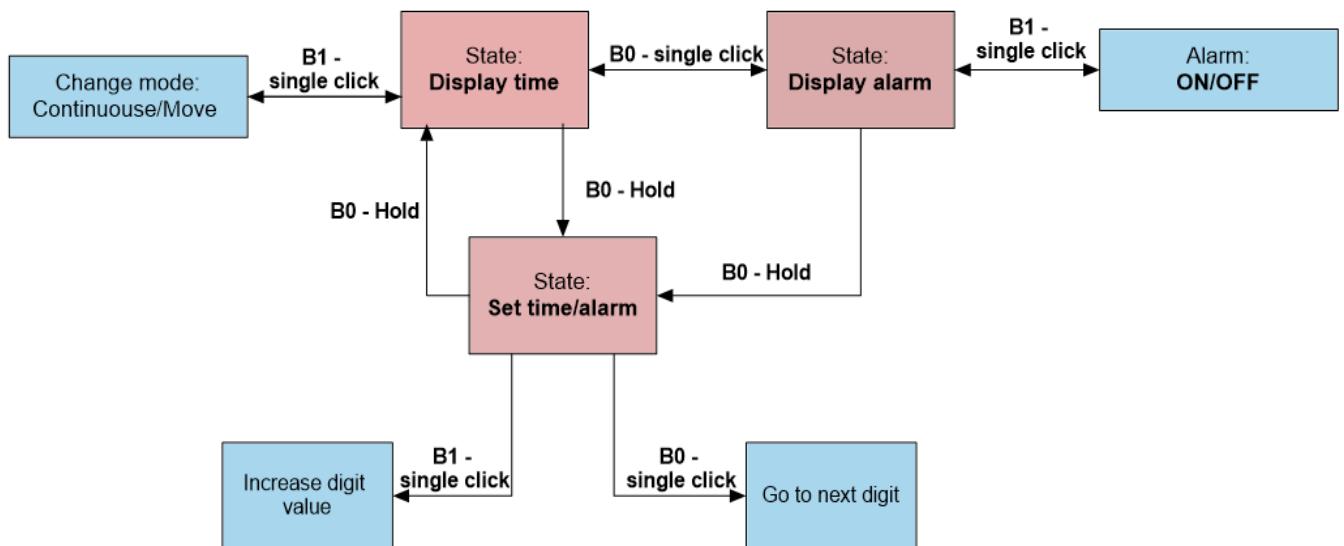


Fig. 1.4 Konfiguracja zegara za pomocą przycisków zegara

1.4 Wykorzystane oprogramowanie i protokoły komunikacyjne

STM32 CubeIDE

Środowisko programistyczne dedykowane dla mikrokontrolerów z rodziny STM32. Pozwala na szybką konfigurację mikrokontrolera bez konieczności zaznajamiania się z szczegółową budową mikrokontrolera.

Protokół USART

Jest to dwukierunkowy protokół komunikacji synchronicznej lub asynchronicznej, który umożliwia generowanie danych kompatybilnych z różnymi innymi protokołami np: RS-232, Smart Card, Modbus. Jest on rozwinięciem protokołu UART z zachowaną wsteczną kompatybilnością (jedynie transmisja asynchroniczna).

Protokół Bluetooth 4.0

Inaczej ***Bluetooth Low Energy*** to protokół charakteryzujący się bardzo niskim zużyciem energii podczas pracy, wspierający transmisję krótkich pakietów danych (od 8 do 27 oktetów) z predkością 1 Mbps. Wykorzystuje technikę ***Frequency Hopping*** pracującą na

paśmie 2.4 GHz. Dzięki 32 bitowej adresacji możliwa jest praca w systemie złożonym z bardzo dużej liczby urządzeń pracujących w tym standardzie. Używany jest do aplikacji, gdzie istnieje konieczność szybkiego łączenia i rozłączania się. Bluetooth 4.0 pracuje w topologii sieci typu *Mesh*.

2. BUDOWA ZEGARA

2.1 Blok zasilania

Z uwagi na konieczność zasilania mikrokontrolera i układów peryferyjnych napięciem 5 V, oraz specyfikę zastosowanych lamp elektronowych wymagających minimalnego napięcia zasilania o wartości 170 V DC, oraz prądu płynącego przez jedną lampa rzędu 2 mA do budowy układu zasilania zegara wykorzystano kilka modułów.:

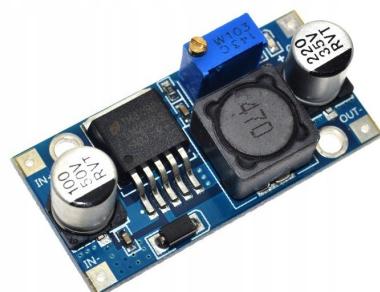
- Zasilacz impulsowy UP0301H-12PE 30W
2,5 A 12 V - zmieniający napięcie
zmienne 230 V na 12 V DC.



- Przetwornicy DC-DC LM2596 - w konfiguracji step-down zmniejszające napięcie z 12 V na 5 V.



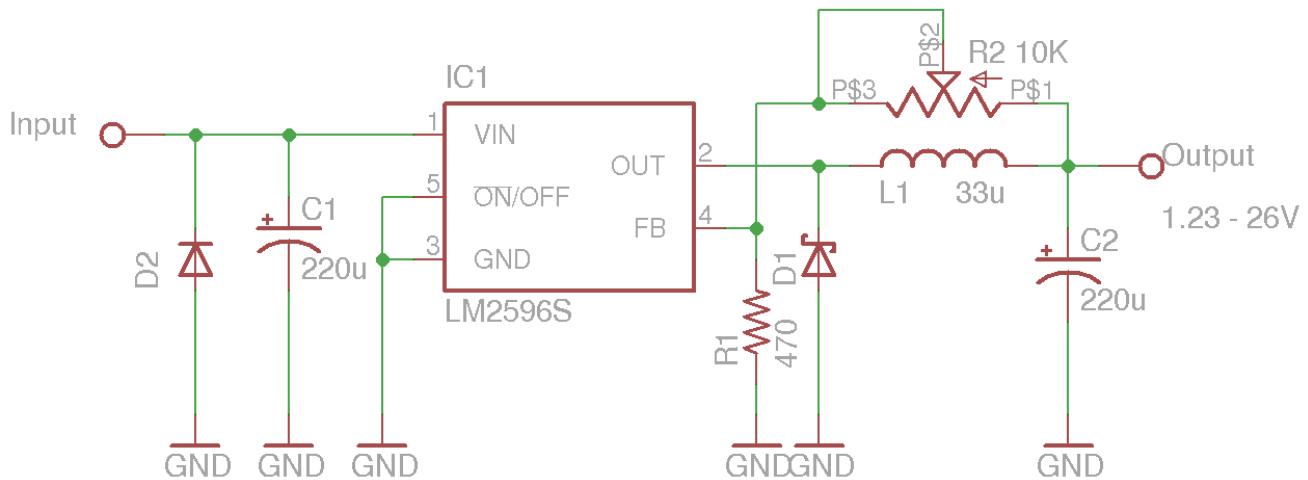
- Wysokonapięciowa przetwornica NCH6100HV DC-DC - zwiększająca napięcie 12 V na 235 V.



Przetwornica DC-DC LM2596-ADJ

Impulsowy regulator step-down o częstotliwości przełączania 150 kHz pracujący z zakresem napięć wejściowych 3 V - 30 V i generując na wyjściu napięcie z zakresu około

1,5 V - 26 V. Maksymalny prąd wyjściowy wynosi 3 A, oraz maksymalna sprawność modułu może osiągnąć 92%. Schemat przetwornicy przedstawiono na rysunku 2.1.



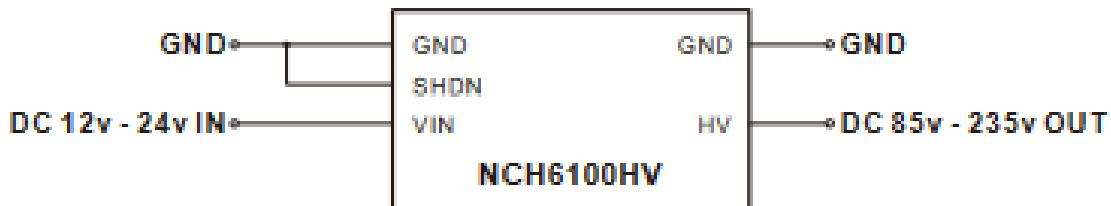
Rys.2.1 Schemat przetwornicy step-down z wykorzystaniem układu LM2596

Opis układu LM2596S:

- **VIN (1)** - wejście regulatora LM2596, na które jest podawane napięcie wejściowe **Input**. W celu zminimalizowania fluktuacji napięcia na wejściu zastosowano bocznikujący kondensator C1 oraz diodę prostowniczą;
- **OUT (2)** - wyjście wewnętrznego przełącznika układu LM2596, podłączone do cewki podrzymującej przepływ prądu. Dioda D1, oraz kondensator C2 zapobiegają powstawaniu tętnień na wyjściu;
- **GND (3)** - masa układu;
- **FB (4)** - wejście sterujące pętlą sprzężenia zwrotnego. Za pomocą regulowanego dzielnika napięciowego sterowana jest pętla sprzężenia zwrotnego układ LM2596 umożliwiająca ustalenie pożądanej wartości napięcia na wyjściu z zakresu;
- **ON/OFF (5)** - wejście pozwalające na włączanie/wyłączanie układu podając na nie poziom logiczny 0 (ON) lub 1 (OFF).

Wysokonapięciowa przetwornica DC-DC - NCH6100HV

Konwerter step-up pracujący z zakresem napięć wejściowych 12 V - 24 V, z regulowanym, za pomocą potencjometru, napięciem wyjściowym 85 V - 235 V. Układ został zaprojektowany *stricto* pod zasilanie lamp elektronowych.



Rys.2.2 Schemat przetwornicy step-up NCH6100HV

Opis końcówek układu NCH6100HV:

- **VIN** - napięcie wejściowe z zakresu 12 V - 24 V;
- **GND** - masa układu;
- **HV** - wyjście układu;
- **SHDN** - wejście pozwalające na włączanie/wyłączenie układu podając na nie poziom logiczny 0 (ON) lub 1 (OFF).
-

2.2 Blok sterujący

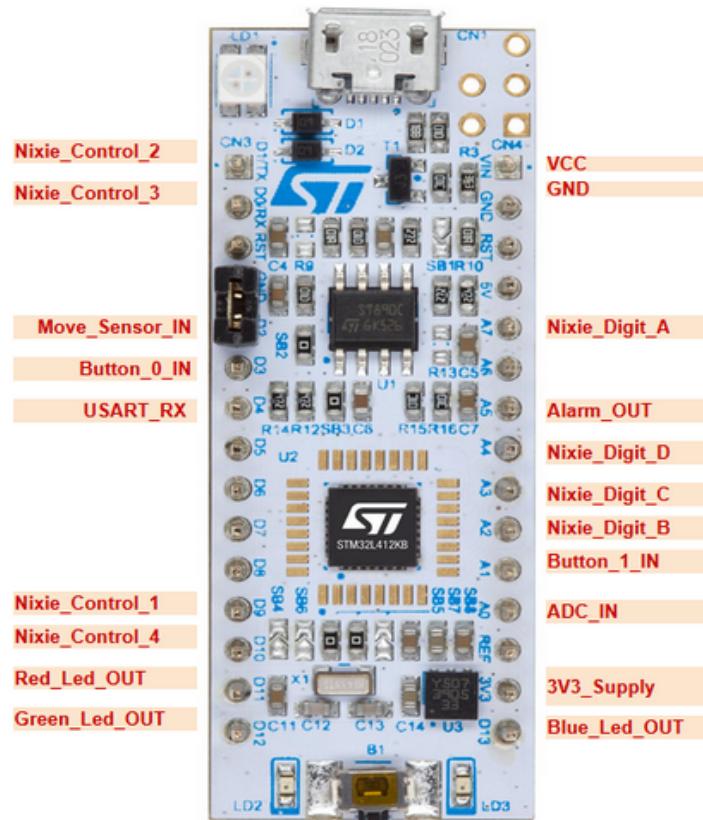
Blok sterujący składa się z:

- płytka rozwojowa z mikrokontrolerem STM32F031K6T6,
- układu sterującego zasilaniem lamp,
- dekodera 74141 służącego do wysterowania określonej cyfry na lampie..

Za sterowanie całym zegarem odpowiada mikrokontroler STM32F031K6T6 z procesorem ARM Cortex M0 - 48 MHz, 32 Kb pamięcią Flash i 4Kb pamięcią SRAM.

Z płytka mikrokontrolera wyprowadzone są piny (Rys.2.3):

- ***Vcc*** - zasilanie mikrokontrolera,
- ***3V3_supply*** - zasilanie 3,3 V,
- ***GND*** - masa,
- ***Nixie_Digit_X*** - wyjścia odpowiedzialne za sterowanie układem dekodera 74141,
- ***Nixie_Control_X*** - wyjścia odpowiedzialne za włączenie odpowiedniej lampy,
- ***Alarm_OUT*** - wyjście sygnalizujące wystąpienie alarmu budzika,
- ***Button_X_IN*** - wejście przycisków sterujących zegarem,
- ***ADC_IN*** - wejście przetwornika ADC służącego do kontroli jasności świecenia lamp za pomocą potencjometru,
- ***Move_Sensor_IN*** - wejście odpowiedzialne za obsługę czujnika ruchu,
- ***USART_RX*** - wejście USART odbierające dane pochodzące z modułu Bluetooth,
- ***Red_Led_OUT, Green_Led_OUT, Blue_Led_OUT*** - wyjścia sterujące diodą RGB.



Rys.2.3 Schemat wyprowadzeń pinów mikrokontrolera

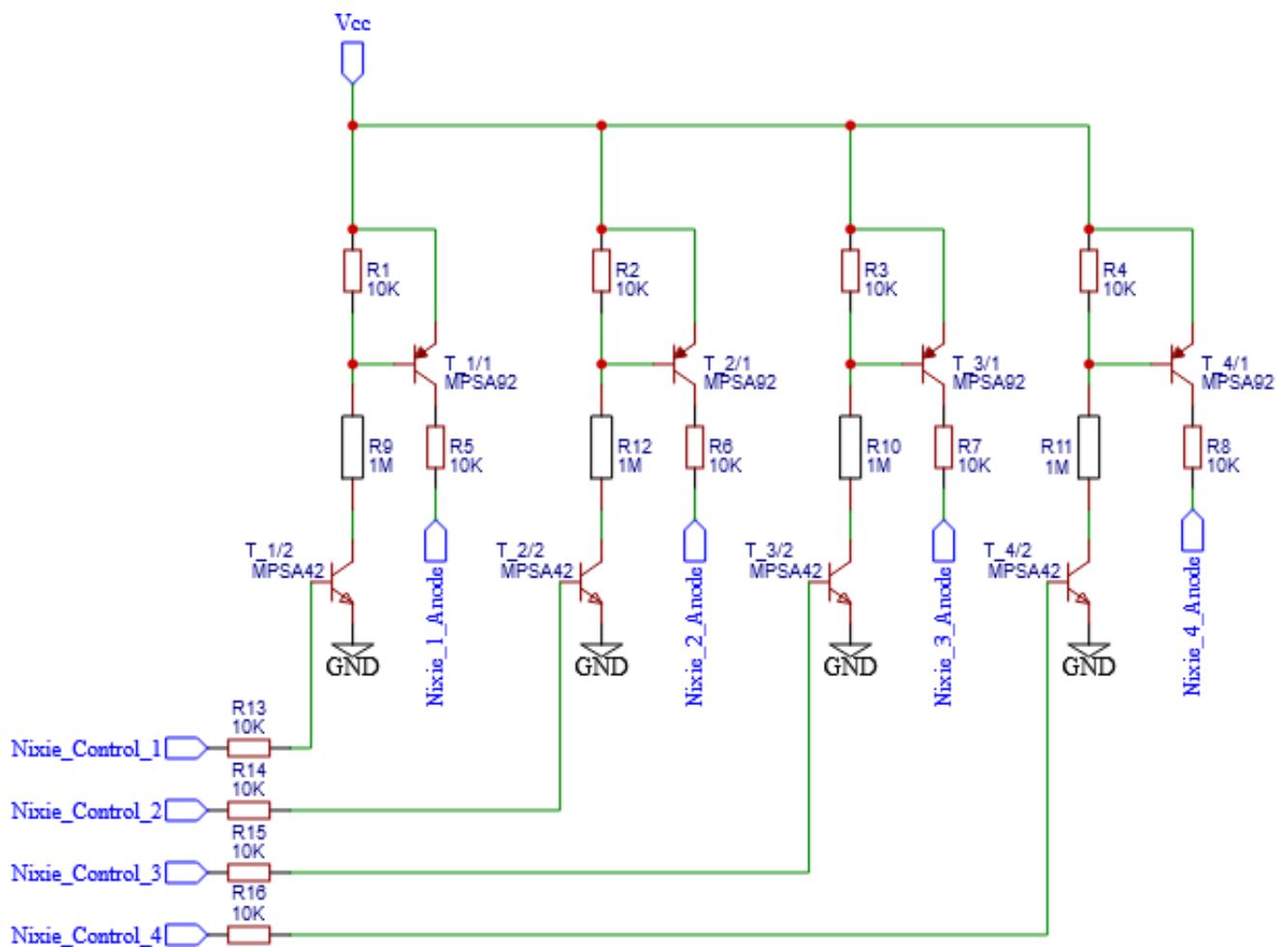
Zastosowane lampy LC-516 charakteryzują się napięciem zapłonu równym 170 V, oraz prądem płynącym przez pojedynczy segment o wartości 2 mA. Posiadają one 11 nóżek: jedną wspólną anodę, oraz 10 katod. Każda katoda jest odpowiedzialna za osobną cyfrę od 0 do 9.



Rys.2.4 Widok lampy od boku

Z powodu wykorzystania jednego układu dekodera 74141, w danej jednostce czasu możliwe jest sterowanie tylko jednego wyświetlacza. W celu uzyskania efektu ciągłego świecenia wszystkich lamp i poprawnego wyświetlania czasu konieczna była multipleksacja lamp i zapalanie każdej lampy z osobna z częstotliwością minimalnie 50 Hz (dodatkowo w celu sterowania jasnością świecenia lamp sterowane są one sygnałem PWM o różnym współczynniku wypełnienia). Przed zapłonem kolejnej lampy gaszona jest poprzednia, oraz następnie zmieniana jest wyświetlana cyfra. Za zapalanie, oraz gaszenie danej lampy odpowiedzialny jest układ do którego podłączone są wyjścia **Nixie_Control_X** (Rys. 2.6).

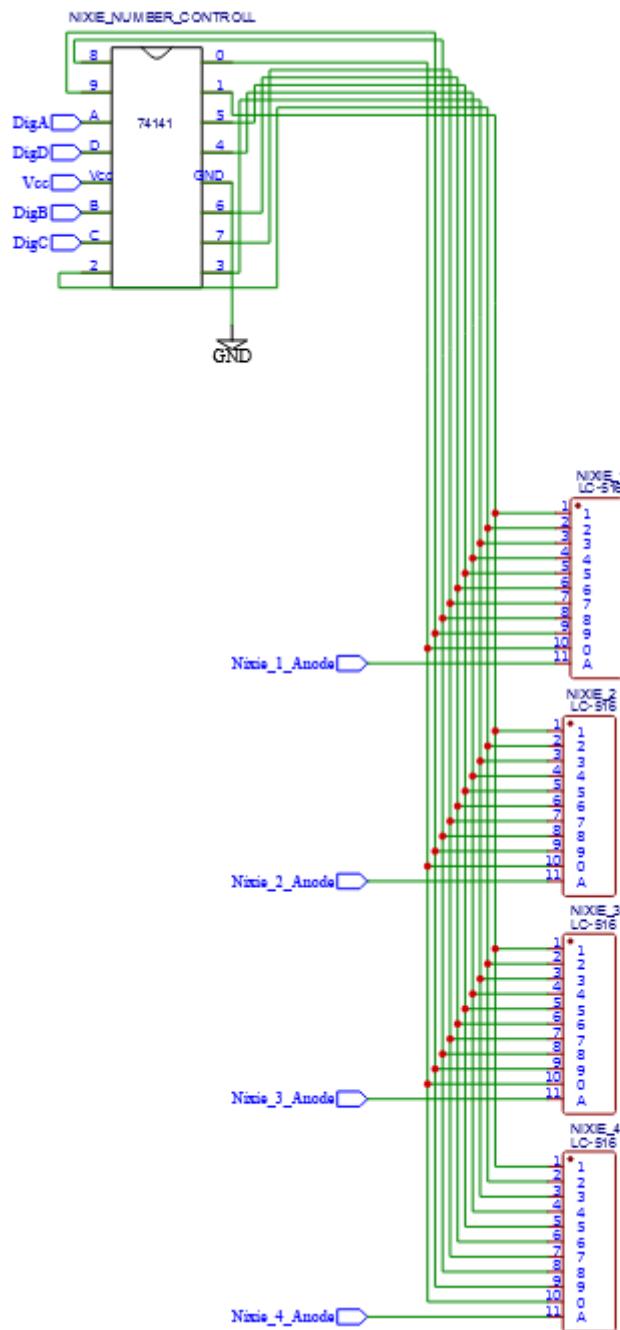
Każdą lampa steruje osobna para tranzystorów, **T_X/1 (MPSA92)** oraz **T_X/2 (MPSA42)**, zaprojektowanych do pracy w układach wymagających dużego napięcia przechodzenia. W wyniku podania logicznego stanu ‘1’ na wejście **Nixie_Control_1** wzrośnie potencjał bazy tranzystora **T_1/2** i spowoduje jego przejście w stan nasycenia. W wyniku przewodzenia tranzystora **T_1/2** i spadku napięcia na rezystorze **R1** potencjał bazy tranzystora **T_1/1** zmniejszy się, co spowoduje przewodzenie tranzystora **T_1/1**. Na anodę lampy zostanie podane napięcie **Vcc** pomniejszone o napięcie kolektor-emiter tranzystora **T_1/1**, oraz spadek na rezystorze **R5**.



Rys.2.5 Widok lampy z przodu

Rys.2.6 Schemat układu sterującego zasilaniem lamp

Za sterowanie cyframi lampy odpowiedzialny jest układ NTE74141, dedykowany dla lamp Nixie (Rys.2.6.1). Wyjścia mikrokontrolera (*Nixie_Digit_X*) podłączone są do wejść dekodera. Jest to dekoder BCD na kod dziesiętny. Podając różne kombinacje stanów na wejścia dekodera (*A*, *B*, *C*, *D*) jesteśmy w stanie załączyć odpowiednią katodę lampy (Rys.2.7).



Rys.2.6.1 Schemat połączenia układu NTE74141 z lampami

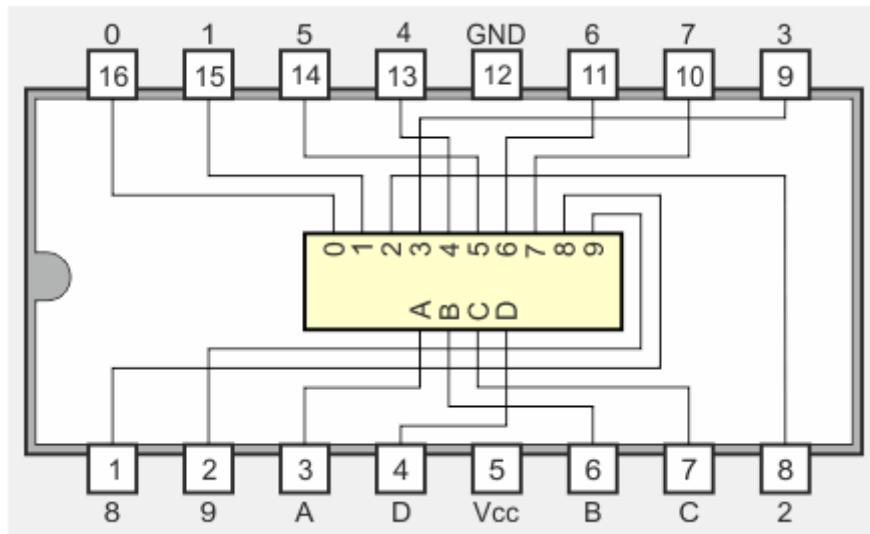
Wejścia				Wyjścia									
D	C	B	A	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	N	N	N	N	N	N	N	N	N
0	0	0	1	N	0	N	N	N	N	N	N	N	N
0	0	1	0	N	N	0	N	N	N	N	N	N	N
0	0	1	1	N	N	N	0	N	N	N	N	N	N
0	1	0	0	N	N	N	N	0	N	N	N	N	N
0	1	0	1	N	N	N	N	N	0	N	N	N	N
0	1	1	0	N	N	N	N	N	N	0	N	N	N
0	1	1	1	N	N	N	N	N	N	N	0	N	N
1	0	0	0	N	N	N	N	N	N	N	N	0	N
1	0	0	1	N	N	N	N	N	N	N	N	N	0
1	0	1	0	N	N	N	N	N	N	N	N	N	N
1	0	1	1	N	N	N	N	N	N	N	N	N	N
1	1	0	0	N	N	N	N	N	N	N	N	N	N
1	1	0	1	N	N	N	N	N	N	N	N	N	N
1	1	1	0	N	N	N	N	N	N	N	N	N	N
1	1	1	1	N	N	N	N	N	N	N	N	N	N

Rys.2.7 Tabela stanów dekodera 74141

Dla stanu wyłączonego napięcie wyjściowe ma wartość 60 V (**Fig.2.8**), ponieważ do zapłonu lampy LC-516 wymagane jest napięcie przewodzenia około 170 V (nie zostanie ono osiągnięte przy wykorzystanym napięciu zasilania 200 V). Dla stanu włączonego napięcie wynosi 2,5 V powodując zapłon danego segmentu.

	Opis parametru	Wartość
VCC	Napięcie zasilania	4,75...5,25 V
VIH	Napięcie wejściowe dla stanu 1	2 V
VIL	Napięcie wejściowe dla stanu 0	0,8 V
VO(on)	Napięcie wyjściowe w stanie włączonym	2,5 V
VO(off)	Napięcie wyjściowe w stanie wyłączonej	60 V

Rys.2.8 Parametry układu 74141

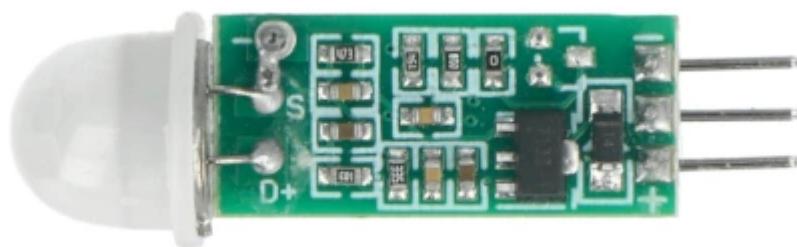


Rys.2.9 Wyrowadzenia układu scalonego 74141

2.3 Układy peryferyjne

Czujnik ruchu PIR HC-SR505

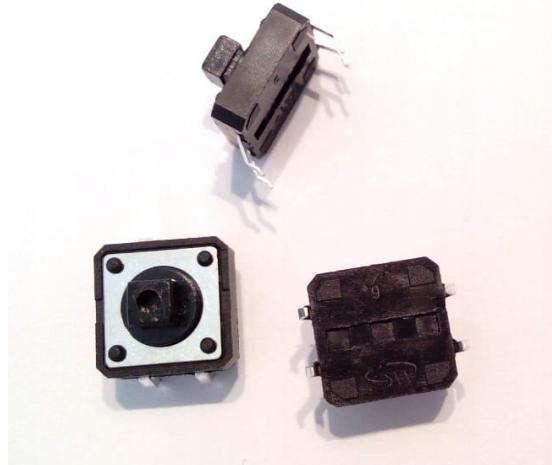
Czujnik ruchu (Rys.2.9) wykorzystywany jest w celu detekcji ruchu w obszarze zegara w celu zapalenia lamp w trybie '**Move Mode**'. Podłączony jest on do wejścia **Move_Sensor_IN**. Charakteryzuje się on zakresem pomiarowym maksymalnie do 3 m, oraz kątem widzenia do 100 °. Posiada on 3 piny - zasilanie Vcc (5 V), masa, (GND), oraz wyjście sygnalizujące (OUT). W przypadku wykrycia ruchu sygnalizuje on wykrycie stanem wysokim (3,3 V) na wyjściu OUT , oraz utrzymuje ten stan przez 8 sekund. Brak wykrycia obiektu sygnalizowany jest stanem niskim (0 V). Jego praca nie ma wpływu na działanie zegara w trybie '**Continuous Mode**'.



Rys.2.9 Czujnik ruchu PIR HC-SR505

Przyciski sterujące

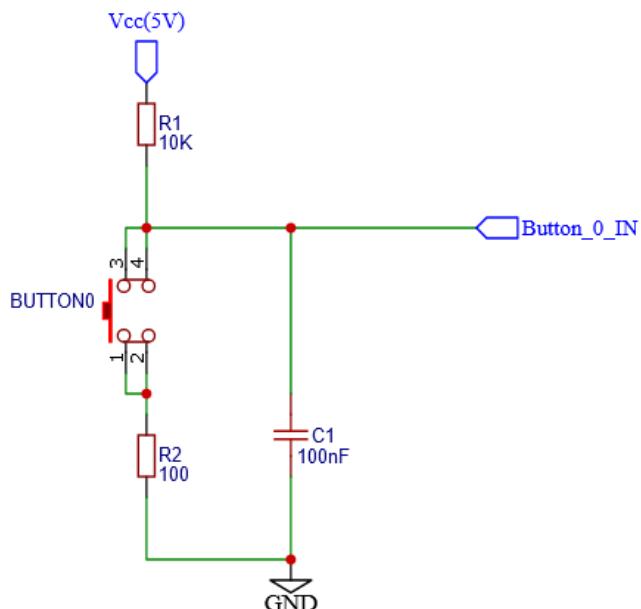
W celu manualnej konfiguracji zegara zastosowane zostały dwa przyciski (Rys.2.10) podłączone do wejść mikrokontrolera **Button_0_IN** oraz **Button_1_IN**. Pozwalają one na zmianę czasu, alarmu oraz trybu pracy.



Rys.2.10 Przyciski sterujące zegarem

W celu obsługi takiej liczby funkcji konieczne było zaimplementowanie programowej wielofunkcjonalności przycisków tzn. zależnie od aktualnego trybu pracy zegara, oraz czasu, czy liczby wciśnięć przycisk spełnia inną funkcję (Fig.1.4).

W celu eliminacji drgań styków została wykorzystana sprzętowa eliminacja drgań (Rys.2.11).



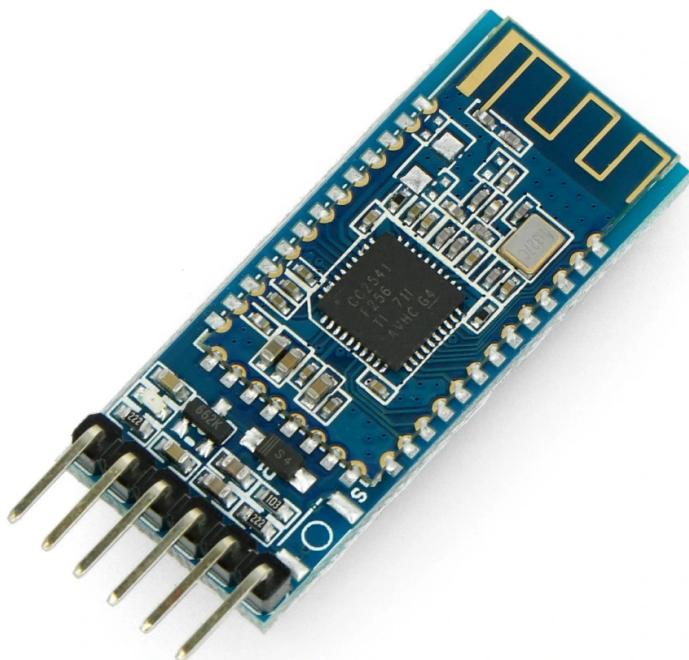
Rys.2.11 Układ sprzętowej eliminacji drgań styku

Równolegle do przycisku dodany został kondensator **C1** w celu eliminacji drgań, oraz szeregowo z przyciskiem połączony został rezystor **R2** w celu ograniczenia prądu rozładowania kondensatora.

Moduł Bluetooth MLT-BT05 HC-10

W celu komunikacji zegara z domowym serwerem został wykorzystany moduł Bluetooth pracujący w standardzie 4.0 (Bluetooth Low Energy) o zasięgu do 10m (Rys.2.12)

W celu wysyłania danych do zegara przez moduł jego wyjście TX podłączone jest do wejścia **USART_RX** mikrokontrolera. Pracuje on w trybie **slave** co pozwala na inicjowanie połączenia przez serwer w dogodnym dla niego czasie np. co 12 h w celu synchronizacji czasu.



Rys.2.12 Moduł Bluetooth MLT-BT05 HC-10

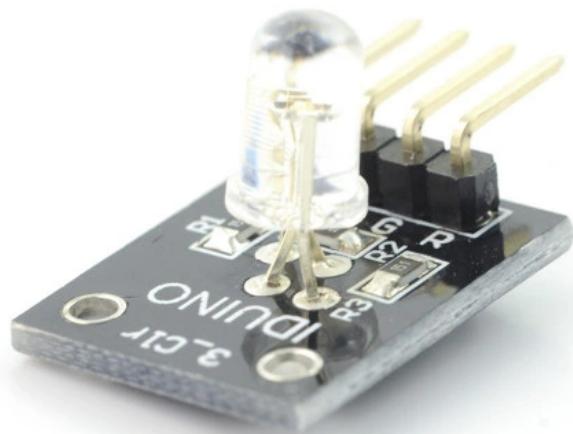
Dioda RGB

Diodę RGB (Rys.2.13) wykorzystano do sygnalizacji obecnego trybu pracy zegara. Poszczególne kolory sterowane są przez komplementarne wyjścia mikrokontrolera **Red_Led_OUT**, **Green_Led_OUT**, **Blue_Led_OUT**.

Znaczenia kolorów:

- **niebieski** - zegar aktualnie wyświetla czas,
- **czerwony** - zegar wyświetla czas alarmu oraz sygnalizuje, że jest on wyłączony,

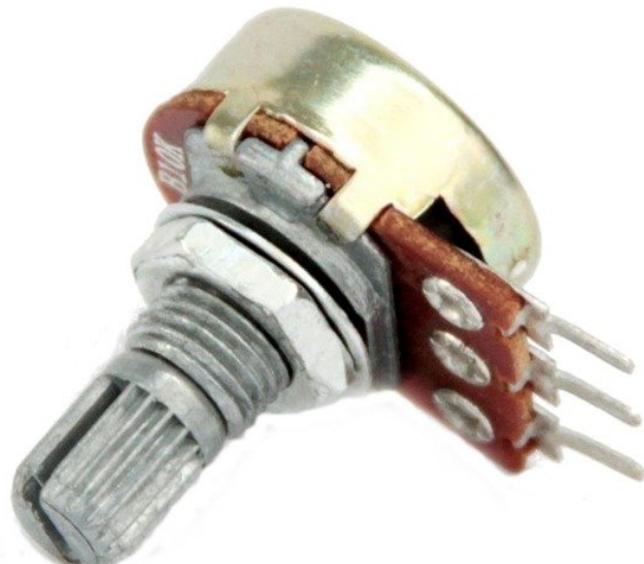
- **zielony** - zegar wyświetla czas alarmu oraz sygnalizuje, że jest on włączony.



Rys.2.13 Dioda RGB

Potencjometr jasności świecenia

Do regulacji jasności świecenia został wykorzystany potencjometr (Rys.2.14) podłączony do wejścia **ADC_IN** mikrokontrolera. Napięcie jest regulowane w zakresie 0 - 3,3 V, oraz przetwarzane przez mikrokontroler w celu sterowania wypełnieniem sygnału PWM zapalającego lampy. Rezystancja potencjometru wynosi $10\text{ k}\Omega$ co pozwala płynnie sterować jasnością świecenia.



Rys.2.14 Potencjometr sterujący jasnością świecenia lamp

Sygnalizator alarmu/budzika

Jest to wyjście mikrokontrolera **ALARM_OUT**. Mikrokontroler sygnalizuje wystąpienie alarmu budzika stanem wysokim, które podawane jest na diodę LED, oraz na zewnętrzne wyjście. Umożliwia ono podpięcie innego układu sterującego np. oświetleniem w pomieszczeniu.

2.3 Serwer domowy

Jako testowy serwer domowy została wykorzystana płytka **ESP32-WROOM-32D** (Rys.2.15). Charakteryzuje się ona wbudowanym układem WiFi 802.11BGN, oraz modułem Bluetooth Low Energy, dzięki którym jesteśmy w stanie zaimplementować funkcję testowego serwera domowego spełniającego wymagania dotyczące tego projektu tzn. pobierać czas z serwera czasu rzeczywistego, oraz synchronizować z nim nasz zegar.

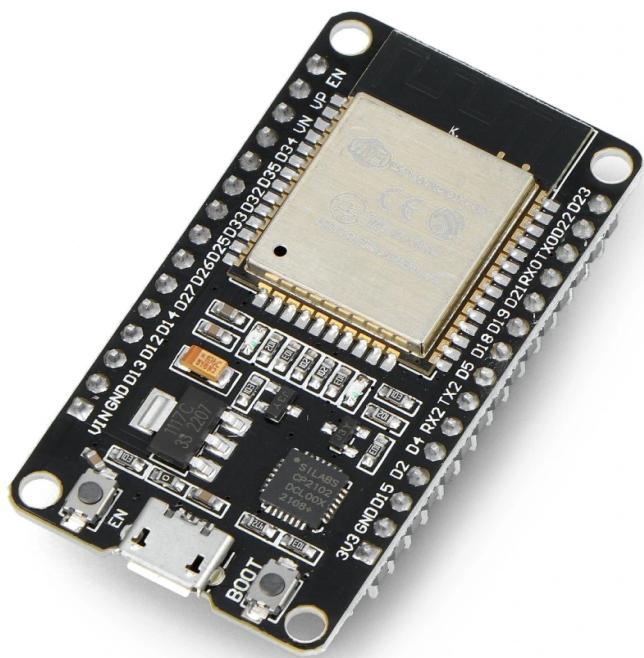


Fig.2.15 Testowy serwer domowy ESP32-WROOM-32D

W czasie swojej pracy ESP32 aktywnie skanuje przestrzeń wyszukując urządzenia domowe i sprawdzając, czy konieczne jest przesłanie danych do któregoś z nich. W przypadku wykrycia zegara, sprawdza kiedy zostało wykonane ostatnie połączenie synchronizujące czas. Synchronizuje on czas zegara w przypadku kiedy zegar nie był wykrywany przez serwer i z powrotem pojawi się on w tablicy wykrytych urządzeń (np. w

przypadku odłączenia zegara od zasilania i ponownym jego wpięciu), lub w przypadku kiedy od ostatniej synchronizacji upłynęło przynajmniej 12 h.

3. Realizacja projektu

3.1 Zegar

Wykorzystane biblioteki

W projekcie została wykorzystana biblioteka HAL (ang. Harware Abstraction Layer) dla mikrokontrolera STM32. Jest to wysokopoziomowa biblioteka interfejsu programowania. Dostarcza on API (ang. Application Programming Interface) do komunikacji mikrokontrolera z układami peryferyjnymi. Pozwala ona na prostą inicjalizację, konfigurację interfejsów, zarządzanie przerwaniami, błędami i transmisją danych. Dzięki zastosowaniu biblioteki HAL jesteśmy w stanie przenieść kod na inną platformę sprzętową z rodziną STM32, bez konieczności zmiany całego kodu.

Struktury oraz typy zmiennych

W celu poprawienia przejrzystości, oraz czytelności kodu programu mikrokontrolera zostały zdefiniowane następujące typy zmiennych:

- **LightUpTube** - typ wyliczeniowy określający wybraną lampa

```
typedef enum {
    All,
    Nixie1,
    Nixie2,
    Nixie3,
    Nixie4,
    None
} LightUpTube ;
```

- **ClkState** - typ wyliczeniowy określający stan pracy zegara

```
typedef enum {
    Time,           //wyświetlanie czasu
    Alarm,          //wyświetlanie czasu alarmu
    TimeSet,         //wyświetlanie podczas ustawiania czasu
    AlarmSet        //wyświetlanie podczas ustawianiu czasu
alarmu
} ClkState;
```

- **Mode** - typ wyliczeniowy określający tryb pracy zegara

```

typedef enum {
    ContinuousMode,
    MoveMode,
} Mode;

```

- **ButtonStates** - typ wyliczeniowy określający stan przycisku

```

typedef enum {
    NotPressed,
    SingleClick,
    DoubleClick,
    Hold
} ButtonStates;

```

- **AlarmState** - typ wyliczeniowy określający stan Alarmu

```

typedef enum {
    ON,
    OFF
} AlarmState;

```

- **MultiButton** - struktura przechowująca zmienne potrzebne do obsługi wielofunkcjonalności przycisku

```

typedef struct {
    volatile bool flag; //flaga wystąpienia przerwania
    uint32_t timer; //ilość systemowych tików
    uint8_t buttonTimerEnable; //stan licznika
    przytrzymania
    uint8_t buttonHoldCounter; //wartość licznika
    ButtonStates buttonState; //stan przycisku
} MultiButton;

```

- **TimeHolder** - struktura przechowująca czas

```

typedef struct {
    uint8_t Hours;
    uint8_t Minutes;
} TimeHolder;

```

- **Clk** - struktura zegara przechowująca wszystkie jego niezbędne stany

```

typedef struct {
    volatile Mode DispMode; //tryb wyświetlania
    volatile ClkState ClockState; //stan pracy zegara
    volatile AlarmState Alarm; //stan alarmu
    volatile LightUpTube Tube; //aktualnie zapalona lampa
}

```

```

    volatile LightUpTube NextNixie; // kolejna lampa do
    zapalenia
    TimeHolder ClockTime; // czas zegara
    TimeHolder AlarmTime; // czas alarmu
} Clk;

```

- **DisplayTime** - struktura przechowująca czas w formacie przygotowanym do wyświetlenia go na lampach

```

typedef struct {
    uint8_t HoursDecimal;
    uint8_t HoursUnity;
    uint8_t MinutesDecimal;
    uint8_t MinutesUnity;
} DisplayTime;

```

Obsługa przycisków

Wejście przycisków są skonfigurowane do pracy w trybie GPIO z zewnętrznym wyzwalaniem przerwania zboczem opadającym. W przypadku wcisnięcia jednego z przycisku wejście zostanie zwarte do masy wywołując w ten sposób przerwanie. W przerwaniu zostaje wywołana funkcja void **HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)** pozwalająca na obsługę przerwania przez użytkownika:

```

if(GPIO_Pin == B0_Pin) {
    currentTicksB0 = HAL_GetTick();
    // eliminacja ewentualnych drgań z styku
    if((currentTicksB0 - previousTicksB0) > 200) {
        previousTicksB0 = currentTicksB0;
        B0.flag = true; // ustawienie flagi
                        // wystąpienia przerwania
    }
}

if(GPIO_Pin == B1_Pin) {
    ...
}

```

Ustawienie flagi wystąpienia przerwania spowoduje wykonanie kodu w pętli głównej. W celu obsługi wielofunkcjonalności konieczne było zastosowanie licznika zliczającego czas wystąpienia poprzedniego wcisnięcia oraz czas przytrzymania przycisku.

```

if(B0.flag && !alarmFlag) {
    B0.flag = false;      // wyczyszczenie flagi wyst. przerwania
}

```

```

if (!B0.buttonTimerEnable) {
    B0.buttonTimerEnable = SET;
    B0.buttonState = SingleClick;
    B0.timer = HAL_GetTick();
}
else{
    B0.buttonState = DoubleClick;
}

```

W przypadku gdy po raz pierwszy zostanie wciśnięty przycisk do zmiennej **B0.timer** zostanie wpisana aktualna wartość czasu pracy programu, oraz stan przycisku zostanie zmieniony na **SingleClick**. Kiedy w ciągu kolejnych 600 ms ponownie zostanie wciśnięty ten sam przycisk stan przycisku zostanie zmieniony na **DoubleClick**.

Następnie co 10ms sprawdzany logiczny na wejściu przycisku i jeżeli jest on nadal wciśnięty zwiększany jest licznik przytrzymania przycisku. Jeżeli będzie on wciśnięty przez 600 ms to stan przycisku zmieni się na **Hold**.

```

if( checkTimer(&B0.timer, 10 * B0.buttonHoldCounter) &&
    B0.buttonTimerEnable
) {

    if(!HAL_GPIO_ReadPin(B0_GPIO_Port, B0_Pin)) {
        ++B0.buttonHoldCounter;

    }
    B0.buttonState = (B0.buttonHoldCounter >= 60) ? Hold :
    B0.buttonState;
}

```

Po upływie 600 ms zostanie sprawdzony stan przycisku i zostanie wykonana instrukcja obsługi przycisku dla danego stanu.

```

if(checkTimer(&B0.timer, 600) && B0.buttonTimerEnable) {

    switch(B0.buttonState) {
        case SingleClick:
            ...
            ...
            break;
        case DoubleClick:
            ...
            ...
            break;
        case Hold:
            ...
            ...
            break;
    }
}

```

```

        default:
            break;
    }
}

```

Wyświetlanie czasu

W celu wprowadzenia możliwości sterowania jasnością świecenia lamp do ich sterowania, zamiast stałego sygnału, został zastosowany sygnał PWM z zmienną wartością współczynnika wypełnienia. Regularnie wywoływanie jest przerwanie od przetwornika ADC, wykonujące pomiar wartości napięcia na wejściu, oraz wpisując przekonwertowaną wartość do zmiennej globalnej **volatile uint8_t ADC_val**:

```

if( h == &hadc ) {
    ADC_val = HAL_ADC_GetValue(&hadc);
}

```

Następnie jest ona mapowana na wartość z zakresu <64, 512> w celu uniemożliwienia całkowitego zgaszenia lamp.

```
Luminosity1 = ADC_val * 8 + 64 - ADC_val;
```

Lampy sterowane są za sygnału PWM licznika **TIM1** - posiada on cztery kanały, każdy odpowiedzialny za inną lampa. Licznik **TIM14** wywołuje z częstotliwością 200 Hz przerwanie ustawiające flagę **timerFlag**, której wartość sprawdzana jest w pętli głównej i powodujące wywołanie funkcji obsługi świecenia lamp.

```

if(timerFlag) {
    timerFlag = 0;
    NixieLightUp();
}

```

W funkcji sprawdzany jest numer lampy która ma zostać zaświecona, a następnie wartość sterująca jasnością zostaje wpisana do rejestru ustalającego cykl pracy sygnału PWM (CCR). Następnie gaszona jest poprzednia lampa (wyłączenie sygnału sterującego PWM lampy poprzedniej), zmieniana wartość wyświetlonej cyfry oraz załączana kolejna lampa. Numer kolejnej lampy do zaświecenia zostaje zmieniony na następną, w wyniku lampy są cyklicznie zapalane i gaszone z częstotliwością 50 Hz każda.

```

void NixieLightUp() {
    ...
    ...
    ...

    switch (Clock.NextNixie) {
        case Nixie1 :

```

```

    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1,
Luminosity1);
    if( Clock.Tube == All || Clock.Tube == Nixie1 ){
        HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_4);
        for(int i = 0;i<190;i++);
        WriteDigit(&TimeToDisp.HoursDecimal);
        HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
        Clock.NextNixie = Nixie2;
    }
    else{
        HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_4);
        Clock.NextNixie = Nixie2;
    }
    break;
case Nixie2 :
    ...
    ...
    ...
    break;
case Nixie3 :
    ...
    ...
    ...
    break;
case Nixie4 :
    ...
    ...
    ...
    break;
default:
    break;
}
HAL_ADC_Start_IT(&hadc);
}

```

Odbieranie danych

W przypadkuinicjalizacji połączenia przez serwer w celu synchronizacji czasu łączy się on z modułem Bluetooth, który odbiera przesłane dane, oraz przesyła je do zegara przez interfejs USART. Dane przesyłane są w formacie 5 bajtów, gdzie kolejne bajty oznaczają:

- pierwszy oznacza typ danych - 'T' czas,
- kolejne cztery zawierają czas w postaci 'HHMM'.

Po otrzymaniu 5 bajtów wywoływana jest funkcja powrotu z przerwania i ustawiana jest flaga **flagUsart**, gdzie następnie w pętli głównej wykonywana jest instrukcja sprawdzająca poprawność danych.

```

if( flagUsart ){
    flagUsart = 0;
    if( Received[0] == 84){
        HoursUsart = ((int)Received[1] - 48)*10 + ((int)Received[2] - 48);
        MinutesUsart = ((int)Received[3] - 48)*10 + ((int)Received[4] - 48);
        if( HoursUsart > 23 || HoursUsart < 0){
            HAL_UART_Receive_IT(&huart1, Received, 5);
        }
        else{
            Clock.ClockTime.Hours = HoursUsart;
            Clock.ClockTime.Minutes = MinutesUsart;
            set_time(HoursUsart, MinutesUsart, 0);
            HAL_UART_Receive_IT(&huart1, Received, 5);
        }
    }
}

```

Jeżeli dane są poprawne co do zakresu wartości czas zegara zostaje nadpisany otrzymaną wartością z serwera.

3.2 Serwer

Zadaniem serwera jest monitorowanie urządzeń znajdujących się w sieci domowej i zarządzanie nimi poprzez krótkie połączenie i wymianę danych.

Po startie serwer podłącza się do określonej sieci Wifi oraz synchronizuje swój czas z serwerem czasu rzeczywistego (pool.ntp.org) - przy braku połączenia z siecią Wifi serwer nie rozpocznie swojej pracy próbując się z nią połączyć do skutku. W celu kontroli działania serwera można się z nim połączyć poprzez interfejs serialowy, gdzie wysyłane są komunikaty dotyczące bieżącej pracy serwera.

```

ntpServer = "pool.ntp.org"
...
...
Serial.printf("Connecting to %s ", ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println(" CONNECTED TO WiFi");
configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
while (1) {
    if ( getTime(&timeinfo) ) {
        localTime = timeinfo;
        break;
    }
}

```

```

        else {
            Serial.println(" Can't get time .... retry");
            delay(100);
        }
    }
}

```

Co 2 sekundy zegar skanuje przestrzeń i sprawdza tabele wyszukanych urządzeń i porównuje je z adresami podlegających mu urządzeń:

```
BLEScanResults foundDevices = pBLEScan->start(scanTime, false);
```

W przypadku wykrycia pojawienia się urządzenia, lub po upłynięciu określonego okresu od ostatniego połączenia z urządzeniem serwer nawiązuje połączenie z zegarem, oraz przesyła mu czas pobrany z serwera czasu rzeczywistego:

```

if (connected)
{
    Serial.println("Send Data");
    while (1) {
        if (getTime(&localTime) ) {
            timeinfo = localTime;
            break;
        }
        else {
            Serial.println(" Can't get time .... retry");
            delay(100);
        }
    }
    lastConnection = timeinfo.tm_hour * 3600 + timeinfo.tm_min
*60;
    if (int(timeinfo.tm_hour) < 10) {
        hour = "0" + String(timeinfo.tm_hour);
    }
    else {
        hour = String(timeinfo.tm_hour);
    }
    if (int(timeinfo.tm_min) < 10) {
        minutes = "0" + String(timeinfo.tm_min);
    }
    else {
        minutes = String(timeinfo.tm_min);
    }

    String newValue = "T" + hour + minutes;
    Serial.println(newValue);
    pRemoteCharacteristic->writeValue(newValue.c_str(),
                                      newValue.length());
}

```

```

        Serial.println("Disconnecting");
        connect = false;
    }
}

```

Po udanym połączeniu serwer rozłącza połączenie wracając do obsługi reszty systemu.

```
pClient->disconnect();
```

4. EKSPERYMENTY I POMIARY

4.1 Dobór częstotliwości zapalania lamp

W celu zmniejszenia poboru energii przez lampy, oraz zachowania pozornego stałego świecenia lamp zapalone są one z częstotliwością 50 Hz. W tym celu konieczne było odpowiednie dopasowanie okresu zliczania licznika sterującego zapalaniem, oraz wartością preskalera wewnętrznego źródła zegara taktującego, którego częstotliwość pracy wynosi 8 MHz. W celu obliczenia tych wartości zostały zastosowane wzory:

$$TIM_{clock} = \frac{Main_{clock}}{Prescaler} [Hz], \quad f = \frac{TIM_{clock}}{ARR} [Hz],$$

Main_{clock} - częstotliwość głównego zegara taktującego

TIM_{clock} - częstotliwość zegara taktującego licznik

Prescaler - wartość preskalera

ARR - (Auto Reload Register) wartość okresu zliczania licznika

W celu uzyskania częstotliwości pracy licznika (200 Hz z koniecznością obsługi czterech lamp) wartość preskalera została ustalona na wartość 800, oraz wartość okresu zliczania licznika na 50.

$$TIM_{clock} = \frac{8\text{ MHz}}{800} = 10\text{ kHz}, \quad f = \frac{10\text{ kHz}}{50} = 200\text{ [Hz]}$$

W celu płynnej kontroli nad współczynnikiem wypełnienia sygnału PWM sterującego poszczególną lampą wartość preskalera, oraz okresu zliczania zostały dobrane metodą eksperymentalną w celu uniknięcia efektu niepełnego wygaszania się lamp. Wartość preskalera wyniosła 80, natomiast wartość okresu zliczania 500.

Chcąc uniknąć sytuacji w której lampy zaczną widocznie migotać, lub całkowicie zgasną wartości współczynnika wypełnienia zostały dobrane z przedziału od 12,8 do 100 procent, można ją obliczyć z wzoru:

$$Duty = \frac{CCR}{ARR} * 100 [\%]$$

Duty - wartość współczynnika wypełnienia sygnału PWM

CCR - wartość Capture Compare Register

ARR - (Auto Reload Register) wartość okresu zliczania licznika

4.2 Okres synchronizacji czasu

Do zliczania czasu wyświetlanego przez zegar został wykorzystany wewnętrzny zegar taktujący. W wyniku rozregulowanej częstotliwości generowanej przez rezonator na każdej sekundzie powstaje różnica 35 µs w liczonym czasie. W wyniku tego konieczne było zastosowanie 12 godzinnych interwałów synchronizujących czas z serwerem czasu rzeczywistego w celu eliminacji niepoprawnie wyświetlanego czasu.

5. UWAGI KOŃCOWE

Wykonany projekt spełnił wymagania projektowe zakładające poprawne działanie podstawowych funkcjonalności wskaźnika czasu, jak i te związane z komunikacją z serwerem. Zegar oprócz wskazywania i konfiguracji czasu umożliwia ustawienie budzika na określoną godzinę, oraz jego włączanie oraz wyłączanie. Może on pracować w dwóch trybach wyświetlania, z ciągłym wyświetlaniem czasu, oraz chwilowym. Chwilowy tryb pracy bazuje na czujniku ruchu, który w przypadku wykrycia ruchu w promieniu 3 m zapala lampy na okres 8 sekund. Możliwość zmiany trybu pracy, jak i ustawiania czasu zegara lub budzika, zostały zrealizowane za pomocą wielofunkcyjnych przycisków. Dany stan pracy wskazywany jest przez określony kolor diody RGB. Jasność lamp można zmieniać poprzez manipulowanie potencjometrem.

Komunikacja z serwerem opiera się o krótkie i szybkie transmisję pojedynczych danych synchronizujących za pomocą protokołu Bluetooth 4.0.

Projekt testowego serwera został zaimplementowany na mikrokontrolerze ESP32 na poziomie niezbędnym do przetestowania działania zegara. Jedyna możliwość monitorowania jego pracy odbywa się przez port szeregowy w postaci przesyłanych informacji dotyczących działań podejmowanych przez serwer.

W celu rozwijania, oraz poprawienia aktualnego działania projektu należałoby się skupić na następujących aspektach:

- wykonanie interfejsu graficznego serwera umożliwiającego dynamiczne zarządzanie serwerem, oraz zegarem przez sieć bezprzewodową,
- integracja zegara z innymi urządzeniami w sieci, oraz rozszerzenie jego funkcjonalności np. o możliwość wyświetlania wartości temperatury, oraz wilgotności

pobieranej z czujników w sieci, czy zintegrowanie go z sprzętem do odtwarzania dźwięku,

- dodanie osobnego dokładniejszego modułu czasu rzeczywistego podtrzymywanej osobną baterią, dzięki której zegar zachowa pobraną konfigurację.

BIBLIOGRAFIA

- [1] Jarosław Sieracki, *Elektronika Praktyczna 2/2008*, <https://ep.com.pl/files/2928.pdf>
- [2] *Dokumentacja techniczna LM2596* <https://www.alldatasheet.com/view.jsp?Searchword=LM2596&sField=4>
- [3] *LM2596 3-A Step-Down Voltage Regulator*, <https://www.elcircuit.com/p/lm2596-3-step-down-voltage-regulator.html>
lampy
- [4] *Dokumentacja techniczna ESP32*, <https://docs.espressif.com/projects/esp-idf/en/latest/esp32s2/hw-reference/esp32s2/user-guide-devkitm-1-v1.html>
- [5] http://mirley.firlej.org/zegar_nixie/