POLITECHNIKA
BYDGOSKA
Wydział Telekomunikacji,
Informatyki i Elektrotechniki

Projekt z przedmiotu : Mikroprocesory			
Autor :	Jakub Kawalerski (116894)	Prowadzący:	dr inż. Sławomir Bujnowski
Temat projektu :	Zegar z alarmami		
Kierunek :	Informatyka stosowana	Grupa :	II

Spis treści

1.Specyfikacja projektu

1. Komunikacja z komputerem używając interfejs USART z buforem kołowym i obsługą przerwań,
2. Zaprojektowanie i wdrożenie protokołu komunikacyjnego,
3. Podłączenie fizyczne i logiczne wyświetlacza NOKIA5110 do STM32 oraz konfiguracja wyświetlacza w programie STM32CubeIDE,
4. Obsługa RTC z możliwością ustawienia 3 alarmów na każdy dzień tygodnia,
5. Generowanie sygnału sinusoidalnego o żądanej częstotliwości i amplitudzie z wykorzystaniem DAC przy wsparciu DMA,
6. Ustawienie częstotliwości próbkowania na 10 kHz

Elementy zastosowane przy tworzeniu projektu :

- MIKROPROCESOR: NUCLEO STM32-F401RE
- WYŚWIETLACZ: NOKIA 5110

2. Protokół komunikacyjny

Komunikacja pomiędzy urządzeniem, a użytkownikiem będzie odbywała się poprzez

program terminal np. PuTTY, Realterm.

Dzięki temu użytkownik będzie wstanie wysyłać dane komunikaty do urządzenia STM32.

2.1 Omówienie ramki

START RAMKI	DANE	SUMA KONTROLNA %256	KONIEC RAMKI
#	znaki ASCII	znaki Hex	:
0x23	wszystkie znaki oprócz 0x23 oraz 0x3B	od 0x30 do 0x39 | 0x41 do 0x46	0x3A
1 znak	zakres 0-128 znaków	2 znaki	1 znak

W wewnątrz ramki dopuszczalne są tylko i wyłącznie znaki ASCII,

Wiadomości zawarte wewnątrz ramki są wysyłane bajt po bajcie, poniżej przedstawiono przykład poprawnego wprowadzenia polecenia :

#	S	H	O	W	(	T	I	M	E	)	*C1*	:

Dane są pojedynczo wpisywanymi znakami kodowanymi w siedmiobitowym systemie kodowania znaków ASCII wraz z wartościami liczbowymi interpretowanymi w systemie dziesiętnym.

- maksymalna długość komendy wynosi 128 znaków,
- użytkownik rozpoczyna wprowadzanie ramki używając **„#”**,
- użytkownik kończy wprowadzanie ramki poprzez **„:”**,

Pole oznaczone kolorem zielonym jest to suma kontrolna modulo 256, która jest obliczana z sumy wszystkich znaków będących zawartością w ramce. Proces ten polega na sumowaniu każdego zawartego znaku w ramce, a następnie wykonanie operacji modulo 256.

Warto wspomnieć iż suma kontrolna podzielona jest na dwa bajty, oznacza to, że liczba nie przechodzi w całości , a znak po znaku.

Reprezentowana jest w systemie szesnastkowym, oznacza to że w przypadku otrzymania wartości trzy cyfrowej także zostanie odpowiednio obsłużona.

Cała ramka posiada wielkość 132 bajtów czyli 132 znaków.

2.2 Tabela rozkazów, komunikatów . Omówienie parametrów.

ROZKAZ	WYKONYWANA CZYNNOŚĆ	KOMUNIKAT ZWROTNY
SHOW(ALARMS)	Pokazuje alarmy	"Wyświetlam alarmy"
SHOW(DATE)	Pokazuje aktualną datę	"Pokazuje date"
SHOW(TIME)	Pokazuje aktualny czas	"Pokazuje czas"
SET_DATE[%d.%d.%d,%d]	Ustawienie daty	"Ustawiam date: %d.%d.%d"
SET_TIME[%d.%d.%d]	Ustawianie czasu	"Ustawiam czas: %d.%d.%d"
SET_ALARM[%d,%d.%d.%d]	Ustawienie alarmu	"Alarm został ustawiony"
REMOVE_ALARM[%d,%d]	Usuwa alarm na wybraną godzinę	"Alarm zostal usuniety."
EDIT_ALARM[%d,%d,%d.%d.%d]	Umożliwia edycję alarmu na wybranągodzinę	"Alarm zostal zmodyfikowany."
Inna komenda	BŁĄD	"Nieprawidłowa komenda"

Każdy z parametrów posiada odpowiednio zoptymalizowany zakres wartości

PARAMETR	ZAKRES	OPIS PARAMETRU
%hh	0-23	zakres dla godzin
%mm	0-59	zakres dla minut
%ss	0-59	zakres dla sekund
%DD	1-31	zakres dla dni
%MM	1-12	zakres dla miesięcy
%YY	0-99	zakres dla roku

2.3 Obsługa błędów

- Jeśli kilka znaków rozpoczęcia ramki zostanie wysłanych jeden po drugim ramka rozpocznie się od ostatniego znaku w powtórzeniu, a pozostałe zostaną zignorowane.
- Jeśli kilka znaków końca ramki zostanie wysłanych jeden po drugim, ramka zakończy się na pierwszym takim znaku w powtórzeniu, a pozostałe zostaną zignorowane.
- Jeśli ramka zostanie wprowadzona poprawnie, ale komenda nie zostanie rozpoznana, zostanie zwrócona ramka informująca o błędnej komendzie.
- Jeśli ramka zostanie wprowadzona poprawnie, ale program wykryje różnicę między sumą kontrolną wprowadzoną przez użytkownika a sumą kontrolną wyliczoną przez program, zostanie zwrócona ramka informująca o różnicy między sumami i dalszy kod nie zostanie wykonany.

- Jeśli komenda zostanie wprowadzona poprawnie, ale parametr zostanie podany nieprawidłowo, zostanie zwrócona ramka informująca o nieprawidłowym parametrze dla komendy.

- Jeśli komenda zostanie wprowadzona poprawnie, ale podany parametr przekroczy swój określony zakres, zostanie zwrócona ramka informująca o przekroczonym zakresie parametru dla danej komendy.

3. Konfiguracja projektu

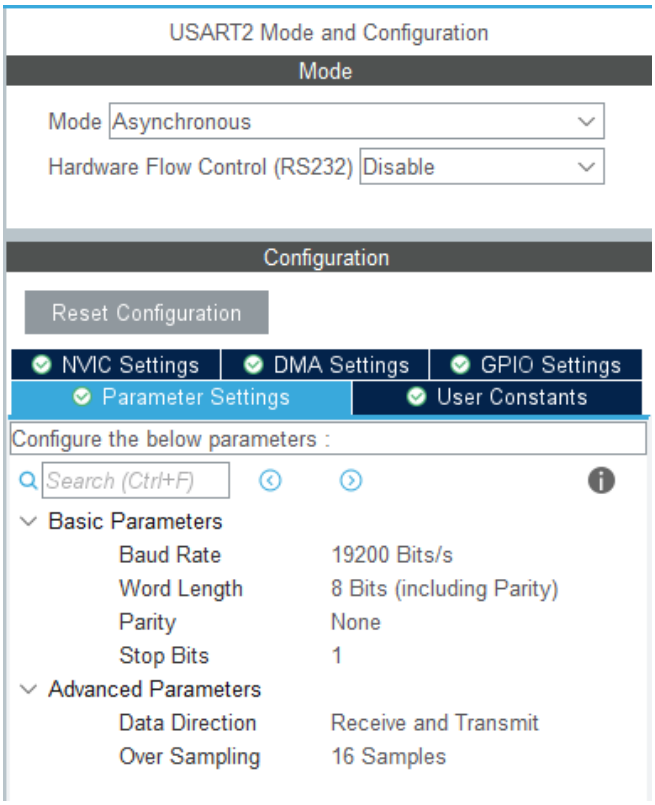
3.1 USART2

- **Baud Rate (19200 Bits/s)** – jest to ilość bitów, przesyłanych na sekundę,

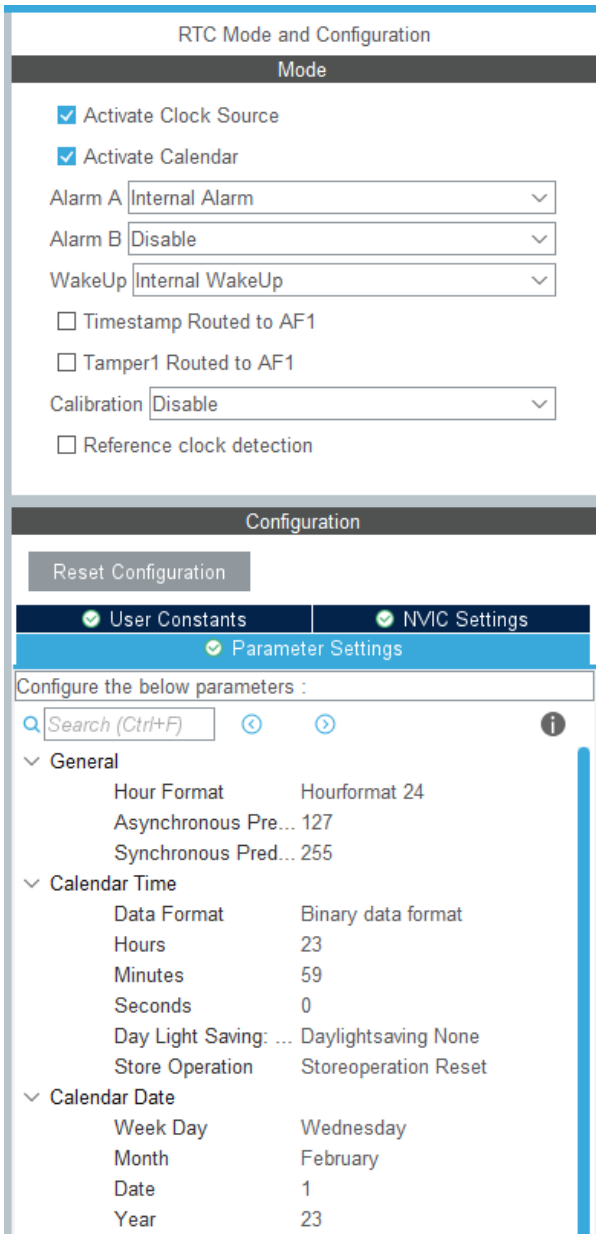
- **Word Lenght** - określa liczbę bitów na jedną ramkę,

- **Parity** – nazywane są bitami parzystości, są to bity które pozwalają na wykrywanie błedów oraz ich późniejsze usuwanie,

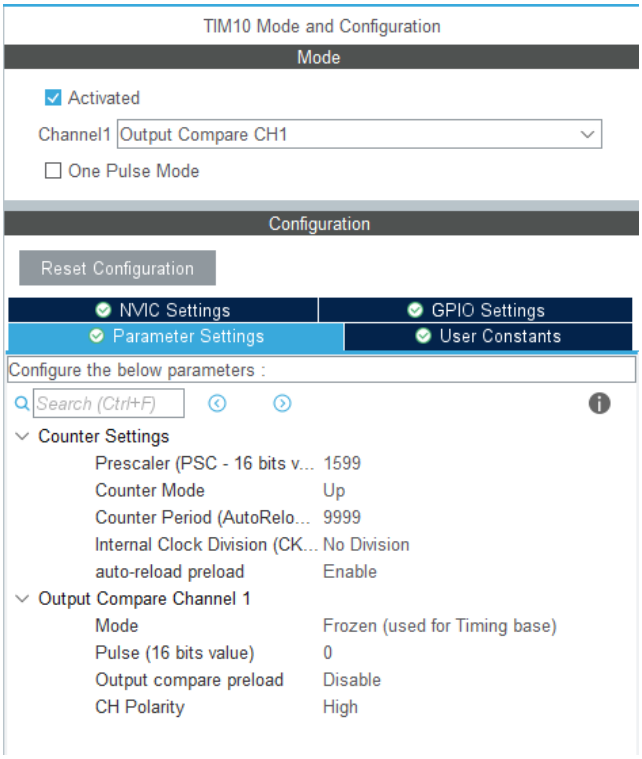
- **Stop Bits** – bit stopu przekazujący informacje o końcu transmisji.



3.2 RTC



3.3 Timer



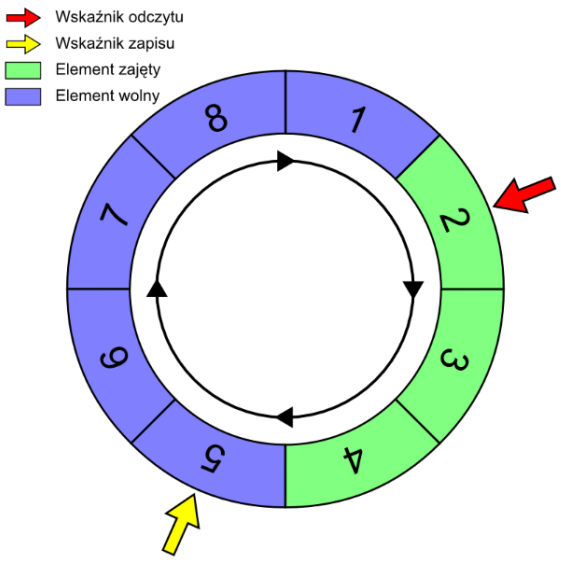
4. Protokół komunikacyjny – kod

4.1 Deklaracja zmiennych

```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define MAX_SIZE_BUFFER 1024
#define MAX_SIZE_CMD 132
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
RTC_HandleTypeDef hrtc;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
//RX BUFFER
uint8_t USART_RX_BUFFER[MAX_SIZE_BUFFER];
__IO int USART_RX_Empty = 0;
__IO int USART_RX_Busy = 0;

//TX BUFFER
uint8_t USART_TX_BUFFER[MAX_SIZE_BUFFER];
__IO int USART_TX_Empty = 0;
__IO int USART_TX_Busy = 0;

//FRAME
char FRAME[MAX_SIZE_CMD];
int frame_idx;
int frame status = 0;
```

4.2 Bufor kołowy

Jest specjalna forma przechowywania informacji, która korzysta z pamięci o stałej wielkości i położeniu do zapisu nieskończonej liczby danych. Składający się z elementów takich jak obszar pamięci do przechowywania informacji oraz dwóch wskaźników, które określają pozycję ostatnio zapisanej oraz najstarszej informacji. W przypadku dodawania nowych danych, wskaźnik ostatniego zapisu jest inkrementowany, natomiast przy odczytywaniu danych, wskaźnik najstarszej informacji jest inkrementowany.



4.3 USART

HAL_UART_RxCpltCallback - jest to funkcja wywoływana jako callback w momencie zakończenia odbierania danych przez moduł UART. Sprawdza, czy podany w argumentach uchwyt "huart" odpowiada uchwytowi "huart2" i w takim przypadku inkrementuje zmienną "USART_RX_Empty". Jeśli ta zmienna osiągnie wartość "MAX_SIZE_BUFFER", zostanie zresetowana do zera. Na koniec wywoływana jest funkcja "HAL_UART_Receive_IT" w celu ponownego odebrania danych z bufora o długości 1 bajtu.

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    if (huart == &huart2) {
        USART_RX_Empty++;
        if (USART_RX_Empty >= MAX_SIZE_BUFFER) {
            USART_RX_Empty = 0;
        }
        HAL_UART_Receive_IT(&huart2, &USART_RX_BUFFER[USART_RX_Empty], 1);
    }
}
```

HAL_UART_TxCpltCallback - jest wywoływana jako callback w momencie zakończenia transmisji danych przez moduł UART. Sprawdza, czy podany w argumentach uchwyt "huart" odpowiada uchwytowi "huart2" i w takim przypadku sprawdza czy zmienna "USART_TX_Empty" jest różna od zmiennej "USART_TX_Busy". Jeśli tak, pobierana jest wartość z bufora "USART_TX_BUFFER" i inkrementowana zmienna "USART_TX_Busy". Jeśli ta zmienna osiągnie wartość "MAX_SIZE_BUFFER", zostanie zresetowana do zera. Na koniec wywoływana jest funkcja "HAL_UART_Transmit_IT" w celu wysłania przez moduł UART danych o długości 1 bajtu.

```
    void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
        if (huart == &huart2) {
            if (USART_TX_Empty != USART_TX_Busy) {
                uint8_t tmp = USART_TX_BUFFER[USART_TX_Busy];
                USART_TX_Busy++;
                if (USART_TX_Busy >= MAX_SIZE_BUFFER) {
                    USART_TX_Busy = 0;
                }
                HAL_UART_Transmit_IT(&huart2, &tmp, 1);
            }
        }
    }
```

Funkcja **USART_fsend** jest używana do wysyłania wiadomości przez moduł UART. Funkcja przyjmuje jako argumenty format wiadomości i wartości, które mają być wstawione w miejsca specjalne znaki formatu (takie jak "%d" lub "%s"). Najpierw tworzona jest tablica znaków "tmp_rs", w której zapisywana jest wiadomość z formatowaniem. Następnie funkcja "vsprintf" jest wywoływana w celu wstawienia wartości w miejsca specjalne znaki formatu. Tablica "frame_message" jest używana do przechowywania kodowanej wiadomości, która jest tworzona przez wywołanie funkcji "USART_fsend_encode". Następnie kodowana wiadomość jest zapisywana w buforze wysyłania UART. W końcu sprawdzany jest stan bufora wysyłania i, jeśli jest pusty, następny bajt wiadomości jest natychmiast wysyłany przez funkcję "HAL_UART_Transmit_IT". W przeciwnym razie, wiadomość jest wysyłana w momencie, gdy bufor wysyłania jest gotowy do wysłania kolejnego bajtu.

```
void USART_fsend(char *format, ...) {
    char tmp_rs[132];
    int i;
    __IO int idx;
    va_list arglist;
    va_start(arglist, format);
    vsprintf(tmp_rs, format, arglist);
    va_end(arglist);
    char frame_message[132] = { 0 };
    USART_fsend_encode(tmp_rs, frame_message);
    idx = USART_TX_Empty;
    for (i = 0; i < strlen(frame_message); i++) {
        USART_TX_BUFFER[idx] = frame_message[i];
        idx++;
        if (idx >= MAX_SIZE_BUFFER) {
            idx = 0;
        }
    }
    __disable_irq();

    if ((USART_TX_Empty == USART_TX_Busy)
            && (__HAL_UART_GET_FLAG(&huart2,UART_FLAG_TXE) == SET))
        USART_TX_Empty = idx;
        uint8_t tmp = USART_TX_BUFFER[USART_TX_Busy];
        USART_TX_Busy++;
        if (USART_TX_Busy >= MAX_SIZE_BUFFER) {
            USART_TX_Busy = 0;
        }
        HAL_UART_Transmit_IT(&huart2, &tmp, 1);
    } else {
        USART_TX_Empty = idx;
    }
    __enable_irq();
}
```

USART_fsend_encode - jest używana do kodowania wiadomości wysyłanej przez moduł UART. Przyjmuje jako argumenty tablicę znaków "message" oraz tablicę znaków "output", do której zostanie zapisana kodowana wiadomość. Najpierw wywoływana jest funkcja "check_sum_command", która oblicza sumę kontrolną dla danej wiadomości i zapisuje ją w zmiennej "checkSumMessage". Następnie funkcja "sprintf" jest wywoływana w celu utworzenia kodowanej wiadomości w formacie "#%s\r\n%02X:", gdzie "%s" jest zastąpione przez oryginalną wiadomość, a "%02X" jest zastąpione przez sumę kontrolną w formacie dwóch szesnastkowych znaków z zerami na początku, jeśli to konieczne. Ostatecznie kodowana wiadomość jest zapisywana w tablicy "output".

```
void USART_fsend_encode(char message[], char output[]) {
    int checkSumMessage = check_sum_command(message);
    sprintf(output, "#%s\r\n%02X:", message, checkSumMessage);
}
```

4.4 Obsługa ramki

USART_kbhit - sprawdza, czy w buforze odbiorczym są jakieś dane. Jeśli wskaźnik USART_RX_Empty jest równy wskaźnikowi USART_RX_Busy, to znaczy, że bufor jest pusty i funkcja zwraca wartość 0. W przeciwnym razie bufor nie jest pusty i funkcja zwraca wartość 1.

```
uint8_t USART_kbhit() {
    if (USART_RX_Empty == USART_RX_Busy) {
        return 0;
    } else {
        return 1;
    }
}
```

Ramka jest wywoływana w nieskończonej pętli

```
/* USER CODE BEGIN 2 */
  LCD_init();
  HAL_UART_Receive_IT(&huart2, &USART_RX_BUFFER[0], 1);
  USART_fsend("STM32 Start");
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
  while (1) {
      if (USART_kbhit()) {
          USART_getline();
      }
```

Funkcja **check_sum_command()** jest odpowiedzialna za wyliczanie sumy kontrolnej dla przesłanej wiadomości. Zmienna mod jest zmienną pomocniczą, która przechowuje sumę kodów ASCII dla każdego znaku w przesłanej wiadomości. W pętli for każdy znak w przesłanej wiadomości jest dodawany do mod. Na końcu funkcja zwraca wartość modulo 256, co jest używane jako suma kontrolna.

```
int check_sum_command(char *msg) {
    int mod = 0;
    for (int i = 0; i < strlen(msg); i++) {
        mod = mod + msg[i];
    }
    return mod % 256;
}
```

Kod funkcji **convertHexToDecimal** konwertuje liczbę zapisaną w formacie szesnastkowym na system dziesiętny. Funkcja przyjmuje jako argument łańcuch znaków "num" reprezentujący liczbę w formacie szesnastkowym.

Następnie, długość tego łańcucha jest określona przy pomocy funkcji "strlen". Zmienna "base" jest ustawiona na 1, a zmienna "temp" jest ustawiona na 0.

Pętla "for" jest wykonywana od końca łańcucha do początku. W każdej iteracji, wartość kolejnego znaku jest dodawana do zmiennej "temp". W zależności od tego, czy znak jest cyfrą od 0 do 9, czy też literą od A do F, wartość jest obliczana i dodawana do "temp". Zmienna "base" jest mnożona przez 16 po każdej iteracji.

Na koniec, funkcja zwraca wartość zmiennej "temp", która jest reprezentacją liczby w formacie dziesiętnym.

```
int convertHexToDecimal(char num[]) {
    int len = strlen(num);
    int base = 1;
    int temp = 0;
    for (int i = len - 1; i >= 0; i--) {
        if (num[i] >= '0' && num[i] <= '9') {
            temp += (num[i] - 48) * base;
            base = base * 16;
        } else if (num[i] >= 'A' && num[i] <= 'F') {
            temp += (num[i] - 55) * base;
            base = base * 16;
        }
    }
    return temp;
}
```

USART_getchar - służy do odebrania jednego bajtu danych z bufora odbiorczego UART. Zmienna tmp jest zmienną tymczasową, do której przypisywany jest odebrany bajt danych. Następnie, zmienna USART_RX_Busy jest zwiększana, co oznacza, że kolejny bajt danych będzie dostępny na następnej pozycji w buforze odbiorczym. Jeśli wartość USART_RX_Busy przekroczy MAX_SIZE_BUFFER, jest ona ustawiana na 0, co oznacza, że bufor jest ponownie w pełni wypełniony danymi. Funkcja zwraca wartość odebranego bajtu danych.

```
uint8_t USART_getchar() {
    uint8_t tmp;
    if (USART_RX_Empty != USART_RX_Busy) {
        tmp = USART_RX_BUFFER[USART_RX_Busy];
        USART_RX_Busy++;
        if (USART_RX_Busy >= MAX_SIZE_BUFFER) {
            USART_RX_Busy = 0;
        }
        return tmp;
    } else {
        return 0;
    }
}
```

USART_getline():

1. Funkcja jest odpowiedzialna za odbieranie danych przesłanych przez użytkownika i przetwarzanie ich na polecenia.
2. Funkcja USART_getline() używa funkcji USART_getchar() w celu pobrania pojedynczego znaku, który jest następnie sprawdzany.
3. Znak „#” oznacza początek ramki danych. W takim przypadku zmienna frame_status jest ustawiana na 1, indeks ramki jest ustawiany na 0, a bufor FRAME jest czyszczony.
4. Znak „:” oznacza koniec ramki danych. W takim przypadku zmienna frame_status jest ustawiana na 0, a bufor FRAME jest sprawdzany pod kątem sumy kontrolnej.
5. Jeśli suma kontrolna jest zgodna, funkcja startCommand(FRAME) jest wywoływana. W przeciwnym razie użytkownik otrzymuje komunikat o błędzie sumy kontrolnej.
6. Każdy inny znak jest dodawany do bufora FRAME. Funkcja sprawdza również, czy długość bufora nie została przekroczona. W takim przypadku użytkownik otrzymuje komunikat o błędzie.

```
void USART_getline() {
    uint8_t sign = USART_getchar();

    if (sign == 0x23) {
        frame_status = 1;
        frame_idx = 0;
        memset(&FRAME[0], 0, sizeof(FRAME));
        FRAME[frame_idx] = sign;
        frame_idx++;
    } else if (sign == 0x3A) {
        if (frame_status == 1) {
            frame_status = 0;
            FRAME[frame_idx] = sign;
            frame_idx++;
            char checksum_frame[3] = { FRAME[frame_idx - 3],
                    FRAME[frame_idx - 2], '\0' };
            int checksum_frame_dec = convertHexToDecimal(checksum_frame);
            FRAME[frame_idx - 3] = '\0';
            memmove(&FRAME[0], &FRAME[1], MAX_SIZE_CMD);
            int checksum_program = check_sum_command(FRAME);
            if (checksum_program == checksum_frame_dec) {
                startCommand(FRAME);
            } else {
                USART_fsend(
                        "Suma kontrolna nie jest zgodna z suma kontrolna podawana przez uzytkownika.");
                USART_fsend(
                        "Suma kontrolna podana przez uzytkownika wynosi: %02X.",
                        checksum_frame_dec);
                USART_fsend(
                        "Suma kontrolna wyliczona przez program wynosi: %02X.",
                        checksum_program);
            }
        }
    } else {
        FRAME[frame_idx] = sign;
        frame_idx++;
        if (frame_idx > MAX_SIZE_CMD) {
            USART_fsend("Przekroczono dlugosc ramki.");
            frame_status = 0;
            return;
        }
    }
}
```

4.5 Zaimplementowane komendy

- Poniżej przedstawiono wszystkie komendy dostępne do wywołania przez użytkownika

```
void startCommand(char *command) {
    int hour, minute, second;
    int weekDay;
    int day, month, year;
    int slot;
    if (strcmp("SHOW(ALARMS)", command) == 0) {
        USART_fsend("Wyświetlam alarmy");
        showAlarms();
    } else if (strcmp("SHOW(DATE)", command) == 0) {
        USART_fsend("Pokazuje date");
        showDate();
    } else if (strcmp("SHOW(TIME)", command) == 0) {
        USART_fsend("Pokazuje czas");
        showTime();
    } else if (sscanf(command, "SET_DATE[%d.%d.%d,%d];", &day, &month, &year,
            &weekDay) == 4) {
        if ((day >= 1 && day <= 31) && (month >= 1 && month <= 12)
                && (year >= 1 && year <= 99)
                && (weekDay >= 0 && weekDay <= 6)) {
            USART_fsend("Ustawiam date: %d.%d.%d", day, month, year);
            setDate(year, month, day, weekDay);
        } else {
            USART_fsend("Niepoprawny format daty");
        }

    } else if (sscanf(command, "SET_TIME[%d.%d.%d];", &hour, &minute, &second)
            == 3) {
        if ((hour >= 0 && hour <= 23) && (minute >= 0 && minute <= 59)
                && (second >= 0 && second <= 59)) {
            USART_fsend("Ustawiam czas: %d.%d.%d", hour, minute, second);
            setTime(hour, minute, second);
        } else {
            USART_fsend("Wprowadzono błędny format czasu.");
        }

    } else if (sscanf(command, "SET_ALARM[%d,%d.%d.%d];", &weekDay, &hour,
            &minute, &second) == 4) {
        if (weekDay >= 0 && weekDay <= 6) {
            setAlarmValue(hour, minute, second, weekDay);
        } else {
            USART_fsend("Wprowadzono błędną wartość dnia.");
        }
    } else if (sscanf(command, "REMOVE_ALARM[%d,%d];", &weekDay, &slot) == 2) {
        removeAlarm(weekDay, slot);
    } else if (sscanf(command, "EDIT_ALARM[%d,%d,%d.%d.%d];", &weekDay, &slot,
            &hour, &minute, &second) == 5) {
        editAlarm(weekDay, slot, hour, minute, second);
    } else {
        USART_fsend("Nieprawidłowa komenda");
    }
}
```

5. Alarmy

5.1 Ustawianie 3 alarmów na każdy dzień tygodnia

W pierwszej kolejności należy sprawdzić, czy alarm o danych parametrach już istnieje. W tym celu wywoływana jest metoda "checkIfSameAlarmExist". Jeśli alarm jest unikalny, to przypisujemy wartości nowego alarmu, wówczas możemy przejść do sortowania poprzez metodę "sortAlarmValue". Jeśli warunki są spełnione, wywoływana jest metoda "setAlarm", która ustawia dzień tygodnia do ustawienia alarmu jako przerwanie.

W zależności od zaistniałej sytuacji, do użytkownika zwrócony zostanie odpowiedni komunikat informujący o statusie wykonanej operacji.

```
void setAlarmValue(int hours, int minutes, int seconds, int weekDay) {
    for (int i = 0; i <= 2; i++) {
        if (AlarmActive[weekDay][i] == 0) {
            if (checkIfSameAlarmExist(hours, minutes, seconds, weekDay) != 1)
                AlarmActive[weekDay][i] = 1;
                Alarm[weekDay][i].AlarmTime.Hours = hours;
                Alarm[weekDay][i].AlarmTime.Minutes = minutes;
                Alarm[weekDay][i].AlarmTime.Seconds = seconds;
                sortAlarmValue(weekDay);
                setAlarm();
                USART_fsend("Alarm został ustawiony");
                return;
            } else {
                USART_fsend("Już istnieje taki alarm.");
                return;
            }
        }
    }
    USART_fsend("Wszystkie alarmy dla tego dnia są zajete.");
}
```

Metoda ma na celu sprawdzenie powtórzenia alarmu. Każdy wprowadzony alarm powinien posiadać unikalne wartości.

```
int checkIfSameAlarmExist(int hours, int minutes, int seconds, int weekDay) {
    for (int i = 0; i <= 2; i++) {
        if (Alarm[weekDay][i].AlarmTime.Hours == hours
                && Alarm[weekDay][i].AlarmTime.Minutes == minutes
                && Alarm[weekDay][i].AlarmTime.Seconds == seconds) {
            return 1;
        }
    }
    return 0;
}
```

Sortowanie alarmu odbywa się poprzez pętle "for", gdzie pobierane są po 2 znaki na operację. Jest sprawdzenie, czy alarmy są aktywne i sortuje na początek alarmy aktywne, a na końcu przesuwa alarmy nieaktywne.

Jeśli dwa alarmy są aktywne, to odbywa się warunek, w którym dzięki funkcji "compareAlarmValue" ustawiane jest odpowiednie miejsce alarmu.

Cała operacja powtarza się w celu przejrzenia wszystkich połówek i ustawienia alarmów w kolejności, od najszybciej wywołanego do tego, który zostanie wywołany najpóźniej.

```
void sortAlarmValue(int weekDay) {
    RTC_AlarmTypeDef tempAlarm;
    int tempAlarmActive;
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            if (AlarmActive[weekDay][j] < AlarmActive[weekDay][j + 1]) {
                tempAlarm = Alarm[weekDay][j];
                tempAlarmActive = AlarmActive[weekDay][j];
                Alarm[weekDay][j] = Alarm[weekDay][j + 1];
                AlarmActive[weekDay][j] = AlarmActive[weekDay][j + 1];
                Alarm[weekDay][j + 1] = tempAlarm;
                AlarmActive[weekDay][j + 1] = tempAlarmActive;
            } else if (AlarmActive[weekDay][j] == 1
                    && AlarmActive[weekDay][j + 1] == 1) {
                if (compareAlarmValue(Alarm[weekDay][j], Alarm[weekDay][j + 1])
                        == 0) {
                    tempAlarm = Alarm[weekDay][j];
                    tempAlarmActive = AlarmActive[weekDay][j];
                    Alarm[weekDay][j] = Alarm[weekDay][j + 1];
                    AlarmActive[weekDay][j] = AlarmActive[weekDay][j + 1];
                    Alarm[weekDay][j + 1] = tempAlarm;
                    AlarmActive[weekDay][j + 1] = tempAlarmActive;
                }
            }
        }
    }
}
```

W tej funkcji odbywa się porównywanie wartości w celu ustawienia danych alarmów w odpowiedniej kolejności.

```
int compareAlarmValue(RTC_AlarmTypeDef Alarm1, RTC_AlarmTypeDef Alarm2) {
    if (Alarm1.AlarmTime.Hours < Alarm2.AlarmTime.Hours) {
        return 1;
    } else if ((Alarm1.AlarmTime.Hours == Alarm2.AlarmTime.Hours)
            && (Alarm1.AlarmTime.Minutes < Alarm2.AlarmTime.Minutes)) {
        return 1;
    } else if ((Alarm1.AlarmTime.Hours == Alarm2.AlarmTime.Hours)
            && (Alarm1.AlarmTime.Minutes == Alarm2.AlarmTime.Minutes)
            && (Alarm1.AlarmTime.Seconds < Alarm2.AlarmTime.Seconds)) {
        return 1;
    } else {
        return 0;
    }
}
```

Ustawianie alarmu na dzień tygodnia odbywa się w następujący sposób:

0 - oznacza niedzielę,

6 - oznacza sobotę.

Cały proces jest odpowiednio ustawiany i sprawdzany w metodzie setAlarmForDay.

```
void setAlarm() {
    HAL_RTC_GetTime(&hrtc, &sTime, RTC_FORMAT_BIN);
    HAL_RTC_GetDate(&hrtc, &sDate, RTC_FORMAT_BIN);

    switch (sDate.WeekDay) {
    case 0x01:
        setAlarmForDay(1);
        break;
    case 0x02:
        setAlarmForDay(2);
        break;
    case 0x03:
        setAlarmForDay(3);
        break;
    case 0x04:
        setAlarmForDay(4);
        break;
    case 0x05:
        setAlarmForDay(5);
        break;
    case 0x06:
        setAlarmForDay(6);
    case 0x07:
        setAlarmForDay(0);
        break;
    default:
        break;
    }

}
```

W metodzie setAlarmForDay, najpierw sprawdzana jest aktywność alarmu oraz to, czy już nie ma wybranego momentu wybicia. Jeśli alarm nie spełnia żadnego z tych warunków, to jest ustawiany na północ następnego dnia, aby mieć automatyczne ustawienie na kolejny dzień.

W bloku switch-case jest dodatkowa weryfikacja, aby uwzględnić wyjątki takie jak rok przestępny, miesiąc o długości 30 dni, czy też luty z 28 lub 29 dniami.

Po weryfikacji, alarm jest ustawiany jako przerwanie.

```
void setAlarmForDay(int weekDay) {
    HAL_RTC_GetTime(&hrtc, &sTime, RTC_FORMAT_BIN);
    HAL_RTC_GetDate(&hrtc, &sDate, RTC_FORMAT_BIN);
    for (int i = 0; i <= 2; i++) {
        if (AlarmActive[weekDay][i] == 1) {
            if (compareAlarmWithTime(Alarm[weekDay][i]) == 1) {
                sAlarm.AlarmTime.Hours = Alarm[weekDay][i].AlarmTime.Hours;
                sAlarm.AlarmTime.Minutes = Alarm[weekDay][i].AlarmTime.Minutes;
                sAlarm.AlarmTime.Seconds = Alarm[weekDay][i].AlarmTime.Seconds;
                sAlarm.AlarmTime.SubSeconds = 0;
                sAlarm.AlarmTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
                sAlarm.AlarmTime.StoreOperation = RTC_STOREOPERATION_RESET;
                sAlarm.AlarmMask = RTC_ALARMMASK_NONE;
                sAlarm.AlarmSubSecondMask = RTC_ALARMSUBSECONDMASK_ALL;
                sAlarm.AlarmDateWeekDaySel = RTC_ALARMDATEWEEKDAYSEL_DATE;
                sAlarm.AlarmDateWeekDay = sDate.Date;
                sAlarm.Alarm = RTC_ALARM_A;
                if (HAL_RTC_SetAlarm_IT(&hrtc, &sAlarm, RTC_FORMAT_BIN)
                        != HAL_OK) {
                    Error_Handler();
                }
                return;
            }
        }
    }
    sAlarm.AlarmTime.Hours = 0;
    sAlarm.AlarmTime.Minutes = 0;
    sAlarm.AlarmTime.Seconds = 0;
    sAlarm.AlarmTime.SubSeconds = 0;
    sAlarm.AlarmTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
    sAlarm.AlarmTime.StoreOperation = RTC_STOREOPERATION_RESET;
    sAlarm.AlarmMask = RTC_ALARMMASK_NONE;
    sAlarm.AlarmSubSecondMask = RTC_ALARMSUBSECONDMASK_ALL;
    sAlarm.AlarmDateWeekDaySel = RTC_ALARMDATEWEEKDAYSEL_DATE;
    switch (sDate.Month) {
    case 2:
        if (sDate.Year % 4 == 0) {
            if (sDate.Date == 29) {
                sAlarm.AlarmDateWeekDay = 1;
            } else {
                sAlarm.AlarmDateWeekDay = sDate.Date + 1;
            }
        } else {
            if (sDate.Date > 28) {
                sAlarm.AlarmDateWeekDay = 1;
            } else {
                sAlarm.AlarmDateWeekDay = sDate.Date + 1;
            }
        }
        break;
    case 4:
    case 6:
    case 9:
    case 11:
        if (sDate.Date == 30) {
            sAlarm.AlarmDateWeekDay = 1;
        } else {
            sAlarm.AlarmDateWeekDay = sDate.Date + 1;
        }
        break;
    default:
        if (sDate.Date == 31) {
            sAlarm.AlarmDateWeekDay = 1;
        } else {
            sAlarm.AlarmDateWeekDay = sDate.Date + 1;
        }
        break;
    }
    sAlarm.Alarm = RTC_ALARM_A;
    if (HAL_RTC_SetAlarm_IT(&hrtc, &sAlarm, RTC_FORMAT_BIN) != HAL_OK) {
        Error_Handler();
    }
}
```

Funkcja sprawdza czy wprowadzone dane czasowe przez użytkownika nie są danymi przestarzałymi, czyli godzina minuta sekunda które już minęły danego dnia .

```
int compareAlarmWithTime(RTC_AlarmTypeDef Alarm) {
    HAL_RTC_GetTime(&hrtc, &sTime, RTC_FORMAT_BIN);
    HAL_RTC_GetDate(&hrtc, &sDate, RTC_FORMAT_BIN);
    if (sTime.Hours < Alarm.AlarmTime.Hours) {
        return 1;
    } else if ((sTime.Hours == Alarm.AlarmTime.Hours)
            && (sTime.Minutes < Alarm.AlarmTime.Minutes)) {
        return 1;
    } else if ((sTime.Hours == Alarm.AlarmTime.Hours)
            && (sTime.Minutes == Alarm.AlarmTime.Minutes)
            && (sTime.Seconds < Alarm.AlarmTime.Seconds)) {
        return 1;
    } else {
        return 0;
    }
}
```

5.2 Pokazanie wszystkich alarmów

Funkcja odpowiada za pokazanie wszystkich dostępnych alarmów.

```
void showAlarms() {
    for (int i = 0; i <= 6; i++) {
        USART_fsend("%d", i);
        for (int j = 0; j <= 2; j++) {
            if (AlarmActive[i][j] == 1) {
                USART_fsend("%d. %02d'%02d'%02d", j,
                        Alarm[i][j].AlarmTime.Hours,
                        Alarm[i][j].AlarmTime.Minutes,
                        Alarm[i][j].AlarmTime.Seconds);
            } else {
                USART_fsend("%d. Alarm nieaktywny", j);
            }

        }
    }
}
```

5.3 Edycja istniejących alarmów

Metoda odpowiada za edytowanie istniejących już alarmów , w tym celu użytkownik musi wprowadzić numer dnia tygodnia oraz slotu ustalonego poprzez jego wcześniejsze sortowanie. Jak w każdym przypadku użytkownik zostanie poinfromowany o statusie operacji poprzez odpowiendi komunikat

```
void editAlarm(int weekDay, int slot, int hours, int minutes, int seconds) {
    if (AlarmActive[weekDay][slot] == 1) {
        if (sAlarm.AlarmTime.Hours == Alarm[weekDay][slot].AlarmTime.Hours
                && sAlarm.AlarmTime.Minutes
                        == Alarm[weekDay][slot].AlarmTime.Minutes
                && sAlarm.AlarmTime.Seconds
                        == Alarm[weekDay][slot].AlarmTime.Seconds) {
            HAL_RTC_DeactivateAlarm(&hrtc, sAlarm.Alarm);

        }
        Alarm[weekDay][slot].AlarmTime.Hours = hours;
        Alarm[weekDay][slot].AlarmTime.Minutes = minutes;
        Alarm[weekDay][slot].AlarmTime.Seconds = seconds;
        sortAlarmValue(weekDay);
        setAlarm();
        USART_fsend("Alarm zostal zmodyfikowany.");

    } else {
        USART_fsend("Wybrany alarm nie jest ustawiony.");
    }
}
```

5.4 Usunięcie danego alarmu

Funkcja odpowiada za usunięcie alarmu, lecz w tym przypadku użytkownik musi podać dzień tygodnia oraz wcześniej omawiany slot. Analogicznie zwracany jest odpowiedni komunikat.

```
void removeAlarm(int weekDay, int slot) {
    if (AlarmActive[weekDay][slot] == 1) {
        AlarmActive[weekDay][slot] = 0;
        if (sAlarm.AlarmTime.Hours == Alarm[weekDay][slot].AlarmTime.Hour
                && sAlarm.AlarmTime.Minutes
                        == Alarm[weekDay][slot].AlarmTime.Minutes
                && sAlarm.AlarmTime.Seconds
                        == Alarm[weekDay][slot].AlarmTime.Seconds) {
            HAL_RTC_DeactivateAlarm(&hrtc, sAlarm.Alarm);
        }
        sortAlarmValue(weekDay);
        setAlarm();
        USART_fsend("Alarm zostal usuniety.");

    } else {
        USART_fsend("Wybrany alarm nie jest ustawiony.");
    }
}
```

5.5 Przerwanie jako alarm

```
/* USER CODE BEGIN 4 */
void HAL_RTC_AlarmAEventCallback(RTC_HandleTypeDef *hrtc) {
    if (sAlarm.AlarmTime.Hours == 0 && sAlarm.AlarmTime.Minutes == 0
            && sAlarm.AlarmTime.Seconds == 0) {
        USART_fsend("Zmiana dnia");
    } else {
        USART_fsend("ALARM: %02d.%02d.%02d", sAlarm.AlarmTime.Hours,
                sAlarm.AlarmTime.Minutes, sAlarm.AlarmTime.Seconds);
    }

    setAlarm();
}
```

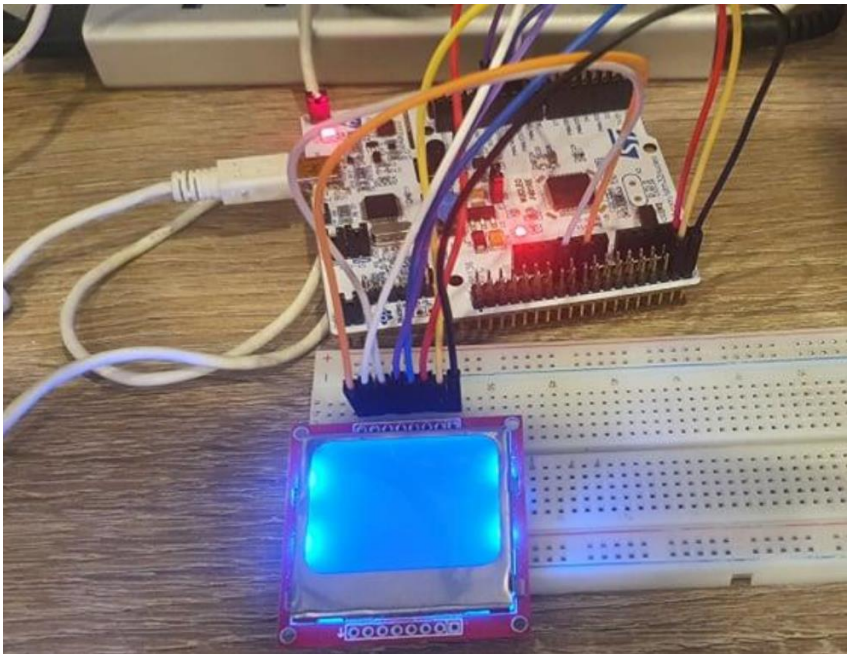
5.6 Obsługa timera

Timer służy do odświeżania ekranu co 1 sekundę.

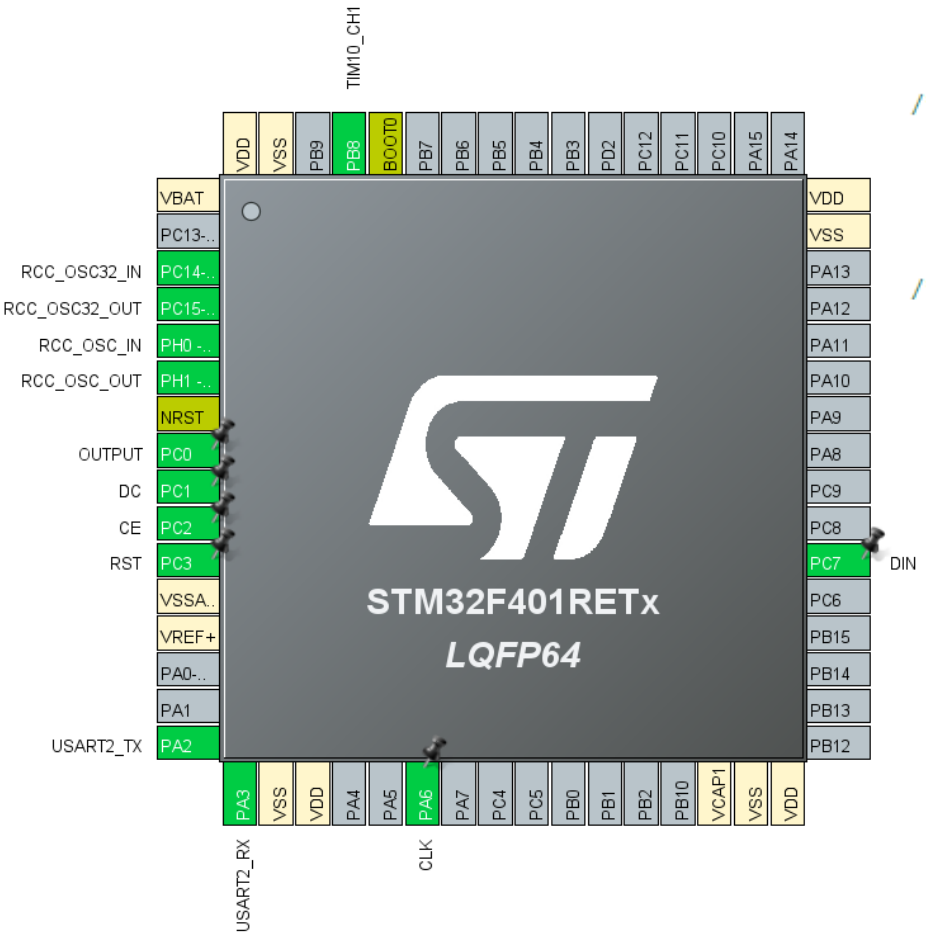
```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM10) {
        displayLcdTimeAndDate();
    }
}
```

6. Podłączenie fizyczne i logiczne wyświetlacza NOKIA 5110

- Podłączenie fizyczne :



- Konfiguracja logiczna w IDE :




```
/* USER CODE BEGIN SysInit */
  LCD_setRST(RST_GPIO_Port, RST_Pin);
  LCD_setCE(CE_GPIO_Port, CE_Pin);
  LCD_setDC(DC_GPIO_Port, DC_Pin);
  LCD_setDIN(DIN_GPIO_Port, DIN_Pin);
  LCD_setCLK(CLK_GPIO_Port, CLK_Pin);
/* USER CODE END SysInit */
```