µC			
Dominik Wróbel	**08 I 2018**	**Pn 9:30**	

Spis Treści

1. Zajęcia nr 1

1.1. Zapalanie diody ‘ Hello World ‘

Opis działania programu

Zapalenie na płytce przykładowej konfiguracji świecących LED.

Kod źródłowy (najważniejsze fragmenty)

```
ORG 0

LJMP START ; Skok do etykiety start aby pominąć zajmowane adresy

ORG 100h
START:
mov P0, #0x00 ; 00000000 Wpisanie do rejestrów wartości odpowiadających stanom LED w HEX
mov P1, #0x55 ; 01010101
mov P2, #0xF0 ; 11110000
mov P3, #0xC3 ; 11000011
```

```
PETLA_MAIN: ; Pętla główna programu - nic nie robi
JMP PETLA_MAIN

end ; koniec programu
```

Wnioski

Zapalenie LED można realizować jedną instrukcją wpisując bajt danych do odpowiedniego rejestru.

1.2. Zmiana stanu LED z wykorzystaniem pętli opóźniającej i instrukcji CALL

Opis działania programu

Program miga LED wykorzystując pętle opóźniającą i instrukcję CALL bez konkretnie dobranej częstotliwości.

Kod źródłowy (najważniejsze fragmenty)

```
ORG 0x00

JMP MAIN

ORG 0x30
PETLA_MAIN:    ; Pętla główna
MOV P1, #0x55 ; 01010101
CALL WAIT ; Skok do etykiety WAIT znajdującej sie dalej w programie
MOV P1, #0xAA ; 10101010
CALL WAIT
JMP PETLA_MAIN

WAIT: ;  Początek podprogramu WAIT
MOV R0, #200 ; Wpisanie do rejestru R0 wartości 200
LOOP: ; Decrement Jump Not Zero
DJNZ R0, LOOP  ; Dekrementuj wartosc podanego rejestru R0 az będzie zerem, gdy nie jest skacz do LOOP
RET ; Powrót do miejsce gdzie została użyta instrukcja CALL
```

Wnioski

Program prawidłowo realizuje migotanie diodą, jednak odbywa się ono bez konkretnej częstotliwości oraz z dużym obciążeniem mikrokontrolera.

1.3. Miganie diodą z wykorzystaniem pętli opóźniających 6 Hz, CALL

Opis działania programu

Celem programu było migotanie diodami z określoną częstotliwością równą numerowi stanowiska przy wykorzystaniu pętli opóźniających i instrukcji CALL. Obliczenia są zawarte w komentarzach.

Kod źródłowy (najważniejsze fragmenty)

```
; Cykl maszynowy = 1,2 us
```

```
; MOV 1 cykl
; CALL 2 cykle
; JMP 2 cykle
; RET 2 cykle
; DJNZ 2 cykle

MAIN_PETLA:
MOV P1, #0x55 ; 01010101 - diody  1,2 us

CALL WAIT ; Wywołanie opóźnienia
; 2 * 2,4 us + ... - RET + CALL
; ... + 84690 us  - WAIT
; CALL WAIT =  84695 us

MOV P1, #0xAA ; 10101010 - zmiana stanu LED 1,2 us
; Tej instrukcji nie biorę do obliczeń, bo ma wartość
; nieznacząca w stosunku do czasu WAIT

CALL WAIT ; Wywołanie opóźnienia

JMP MAIN_PETLA ; Skok do MAIN - pętla główna programu 2 cykle
; Tej instrukcji nie biorę do obliczeń, bo na wartość
; nieznacząca w stosunku do czasu WAIT

WAIT:

THREE: ; Pętla najbardziej zewnętrzna
MOV R3, #25 ; 1 cykl * R3

        TWO: ; Pętla środkowa
        MOV R2, #20 ; 1 cykl * R2 * R3

                ONE: ; Pętla najbardziej wewnętrzna
                MOV R1, #46 ; 1 cykl * R1 * R2 * R3
                DJNZ R1, $ ; 2 cykle * R1 * R2 * R3

        DJNZ R2, ONE  ; 2 cykle * R2 * R3

DJNZ R3, TWO ; 2 cykle * R3

RET ; 2 cykle

; SUMARYCZNIE ( bez call i ret ) :
; Cykl * ( 3*R3 + 3*R2*R3 + 3*R1*R2*R3 )
; f = 6 Hz => T = 1/f = 1/6 = 0,17 s
; Aby działało musi być WAIT = T/2 = 0,085 s
; Wiec biorę R1 = 46, R2 = 20, R3 = 25
; Co daje czas całkowity = 84690 us = 0,084690 s = 0,085 s


end
```

Wnioski

Program działał prawidłowo z odpowiednią częstotliwością, w celu osiągnięcia 6 Hz należało użyć 3 pętli opóźniających aby można było otrzymać odpowiednio duże opóźnienie. Oczywiście uzyskana częstotliwość nie jest równa dokładnie 6 Hz, ponieważ nie wszystkie instrukcje zostały wzięte pod uwagę przy obliczaniu, również dokładne ustawienie wartości rejestrów jest bardzo trudne.

2. Zajęcia nr 2

2.1. Miganie diodą bez konkretnej częstotliwości, wykorzystanie przerwań

Opis działania programu

Program miga diodami bez określonej częstotliwości, ale tym razem robi to z wykorzystaniem przerwań zegarowych.

Kod źródłowy (najważniejsze fragmenty)

```
ORG 0x0B ; Adres pod które skacze przerwanie od przepełnienia Timera 0
        JMP INTERRUPT_T0 ; Skocz do obsługi przerwania

ORG 0x30

INITIALIZE:
        MOV TMOD, #0x01 ; Konfiguracja timera 0, na 16 bit
                        ; Są dwa Timery, mogą być 8 lub 16 bitowe
                        ; Timer 0 ma bity 0-3 w rejestrze, a Timer 1 bity 4-7
                        ; DLA TIMER 0 : 1 = 0x01 -> Timer 16 bit , 2 -> 0x02 Timer 8 bit
                        ; DLA TIMER 1 : 1 = 0x10 -> Timer 16 bit , 2 -> 0x20 Timer 8 bit

                        ; Ustawienie wartości początkowych dla Timera 0
        MOV TL0, #0xFF ; Do rejestrów TL0 i TH0 wpisujemy wartość od której ma zacząć liczyć
licznik
        MOV TH0, #0xEF ; Licznik liczy do 2^16 = 65 536 , licznik zlicza cykle maszynowe = 1,2
us


        SETB EA;      ; Włączenie przerwań globalnie

        SETB ET0;     ; Włączenie przerwań on Timera 0

        SETB TR0;     ; Rozpoczyna działania Timera 0

MAIN_PETLA:
        JMP MAIN_PETLA; ; Pętla główna - nic nie robi

IRTERRUPT_T0:
        CPL P2.3      ; Zmiana stanu LED
        RETI          ; Wyjście z obsługi przerwania, gasi flagę przerwania
```

Wnioski

Program działa poprawnie, jest to lepsze rozwiązanie niż pętle opóźniające, ponieważ jest mniej obciążające dla mikrokontrolera.

2.2. Miganie diodą z częstotliwością 6 Hz, wykorzystanie przerwań

Opis działania programu

Program miga diodami z częstotliwością numeru stanowiska przy wykorzystaniu przerwań zegarowych. Obliczenia w komentarzach.

Kod źródłowy (najważniejsze fragmenty)

```
ORG 0x0B ; Adres pod który skacze program przy przepełnieniu Timer 0
        JMP INTERRUPT_T0 ; Skocz do obsługi przerwania Timera 0

ORG 0x30 ;

INITIALIZE:

        MOV TMOD, #0x01 ; Timer 0 - 16 Bit
        ; Chce migać 6 Hz => T = 1/f = 0,17 s
        ; T / 2 = 83500 us
        ; Przez 83500 us stan wysoki i przez 83500 us stan niski
        ; Ale licznik liczy cykle i ma 65 536 stanów, co daje
        ; max. opóźnienie = 65 536 * 1,2 = 78643,2 us

        MOV TL0, #00011000b ; 65 536 - 1000 = 64536
        MOV TH0, #11111100b ; 1000 * 1,2 us = 1200 us

        MOV R0, #70;            ; 83500 us / 1200 us = 70
                        ; Zmiana stanu diody po 70 przerwaniach
                        ; bo 70 * 1200 = 83500

        SETB TR0;       ; Włącz Timer 0

        SETB ET0;       ; Włącz przerwania od Timer 0

        SETB EA;        ; Włącz przerwania globalnie

MAIN:
        JMP MAIN;       ; Pętla główna programu - nic nie robi

INTERRUPT_T0:
        MOV TL0, #00011000b ; Ponownie załadowanie do licznika wartości od której
        MOV TH0, #11111100b ; ma liczyć = 64536, co daje 1200 us na przerwanie
        DJNZ R0, BACK ; Jeśli nie minęło 70 przerwań, to wróć z przerwania
        CPL P2.3 ; Jeśli minęło 70 przerwań, to zmień stan diody
```

```
        MOV R0, #70 ; Odczekaj kolejne 70 przerwań
        RETI ; Wróć z przerwania i wyzeruj flagę przepełnienia
BACK:
        RETI
```

Wnioski

Program działał poprawnie, wykorzystanie przerwań zamiast pętli opóźniających jest lepszym rozwiązaniem biorąc pod uwagę nakład pracy w pisaniu programu oraz pracę mikrokontrolera.

2.3. Program 3 – Sterowanie diodą PWM

Opis działania programu

Program steruje diodą przy pomocy sterowania PWM, czyli ustawia stan wysoki oraz stan niski na okres w danym stosunku, co powoduje, że dioda świeci jaśniej lub ciemniej. Częstotliwość powinna być większa niż 50 Hz aby nie było widać migania.

Kod źródłowy (najważniejsze fragmenty)

```
ORG 0x0B ; Pod ten adres skacze program gdy powstaje przepełnienie od Timera 0
        JMP INTER_TIM0    ; Skocz do obsługi przerwania

ORG 0x30 ; Tu zaczyna się inicjalizacja - omijam ważne adresy

INITIALIZE:

        MOV TMOD, #0x01 ; Timer 0 , tryb 16 - bit


        MOV TL0, #0xC4 ; Chcę aby mieć przerwanie co 1000 us
        MOV TH0, #0xFC ; Cykl maszynowy = 1,2 us
        ; 1000 / 1,2 = 833
        ; 65 536 - 833 = 64 708 = 0x FC C4
        ; Więc początkowa wartość licznika to FC C4, co daje przerwania
        ; co 1000 us

        MOV R0, #10 ; Rejestr R0 będzie odpowiadał za czas trwania stanu niskiego

        MOV R1, #5 ; Rejestr R1 będzie odpowiadał za czas trwania stanu wysokiego

        MOV P1, #0x04; Zapalona dioda świecąca stale aby móc porównać ze sterowaną PWM
                ; Dioda testowa jest początkowo w stanie niskim

        SETB TR0 ; Włączenie Timera 0

        SETB ET0 ; Włączenie przerwań od Timera 0

        SETB EA ; Włączenie przerwań globalnie

MAIN_PETLA:
```

```
        JMP MAIN_PETLA ; Pętla główna programu - nic nie robi

INTER_TIM0:

        MOV TL0, #0xC4 ; Ponowne ustawienie wartości początkowej licznika 16 bit
        MOV TH0, #0xFC ;

        DJNZ R0, GO_BACK_R0 ;  minie 255 przerwań, zanim zaświeci się dioda P1.1
        MOV P1, #00000101b ; Dioda zaświeca się

        DJNZ R1, GO_BACK_R1 ; Minie 50 przerwań, zanim dioda zgaśnie
        MOV P1, #00000100b ; Dioda gaśnie

        ; Okres T = 1000 us * ( R0 + R1 )
        ; f = 1/T oraz f > 50 Hz aby oko nie widzało przerwań świecenia
        ; czyli 1 / 1000 us ( R0 + R1 ) > 50
        ; Co jest spełnione dla R0 = 10 oraz R1 = 5
        MOV R0, #10
        MOV R1, #5

GO_BACK_R0:
        RETI;
GO_BACK_R1:
        MOV R0, #1 ; Aby zapobiec przepełnieniu z 0 na 255 rejestru R0 ustawiamy go na 1
        RETI
```

Wnioski

Program działał poprawnie dopiero po licznych modyfikacjach, ilość zmiennych wpływających na świecenie diodą wprowadza liczne komplikacje w obliczeniach. Trudnością w tym programie okazał się odpowiedni dobór wartości rejestrów oraz przepełnianie rejestru R0 przy instrukcji DJNZ.

3. Zajęcia nr 3

3.1. Konfiguracja portu szeregowego

Opis działania programu

Program ma na celu przygotowanie do pracy port szeregowy, tak aby mógł wymieniać dane z komputerem, ustawia się szybkość wymiany danych przy pomocy liczników oraz odpowiednich rejestrów. Obliczenia w komentarzach.

Kod źródłowy (najważniejsze fragmenty)

```
org 0x23   ; Pod ten adres skacze program gdy nastąpi przerwanie od
JMP SP_INTER ; portu szeregowego, czyli gdy ustawiony zostanie bit TI ( wysłanie znaku )
          ; lub RI ( odczytanie znaku )

org 0x30
```

```
KONF:
        ; Do wybrania trybu służy rejestr SCON, który zawiera bity SM0 oraz SM1
        ; służące do konfiguracji
        ; Jeśli wybierzemy tryb 1, to baud rate zależy od wartości wpisanej do
        ; górnego rejestru Timera 1 w trybie 8-bitowym, czyli od wartości jego przepełnienia

CLR SM0 ;
SETB SM1 ; Wybieramy tryb 1 działania portu szeregowego

MOV TMOD, #0x20 ; Timer 1 jako 8-bit Timer , aby można było ustawić baud rate

                ; Wartość TH1 dla danego baud rate, które chcemy uzyskać obliczamy ze wzoru :
                ; TH1 = 256 - (  [ fz / 12 * 32 ] / baud rate  )
                ; gdzie fz/12 to zegar systemowy
                ; a 32 to wartość stała
                ; po podstawieniu baud rate = 4800 otrzymano
                ; TH1 = 249
MOV TH1, #0xF9  ; Wpisuje obliczano wartosc do TH1

SETB TR1 ; Włączenie Timera 1

SETB EA ; Pozwolenie na globalne przerwania

SETB ES ; Pozwolenie na przerwania od portu szeregowego

SETB REN ; Pozwolenie na czytanie z portu szeregowego


SP_INTER: ; Tu będzie obsługa przerwania od portu szeregowego
```

Wnioski

Przy wymianie danych z innymi urządzeniami zawsze należy dokonać odpowiedniej konfiguracji portów, tak aby dane zostały wysyłane i odbierane poprawnie. Niektóre z rejestrów można adresować bitowo tak jak zostało to zrobione z wybraniem trybu portu szeregowego – rejestr SCON.

3.2. Wysyłanie znaku z przerwaniami po porcie szeregowym do komputera

Opis działania programu

Program cyklicznie wysyła jeden znak do komputera, który jest odbierany przy pomocy programu i wyświetlany na ekranie. Komunikacja tylko w jedną stronę, wykorzystanie przerwań od portu szeregowego.

Kod źródłowy (najważniejsze fragmenty)

```
org 0x23 ; Tu skacze program gdy zajdzie przerwanie od portu szeregowego
JMP INTER ; Skocz do obsługi przerwania od portu szregowego
```

```
org 0x30 ; Konfiguracja

INITIALIZE:

MOV SCON, #0x50 ; Tryb 1 działania portu szeregowego
                ; oraz ustawienie REN ( zezwolenie na czytanie )

MOV PCON, #0x80 ; Gdy ustawimy bit SMOD rejestru PCON ( najstarszy ), to we wzorze na baud rate
                ; mamy 16 zamiast 32, rejestr PCON nie jest bit-addreseable wiec trzeba ustawić cały

MOV TMOD, #0x20 ; Timer 1 w trybie 8-bit

MOV TH1, #243 ; Wartość TH1 dla danego baud rate, które chcemy uzyskać obliczamy ze wzoru :
                ; TH1 = 256 - (  [ fz / 12 * 16 ] / baud rate  )
                ; gdzie fz/12 to zegar systemowy
                ; a 32 to wartość stała
                ; po podstawieniu baud rate = 4800 otrzymano
                ; TH1 = 243

MOV TL1, #243 ; Aby za pierwszym razem licznik liczył od 243, a nie od 0

SETB TR1 ; Włącz Timer 1

SETB EA ; Pozwolenie na przerwania globalne
SETB ES; Przerwania od portu szeregowego włączone
CLR RI ; Wyzruj bit czytania z portu szeregowego
CLR TI ; oraz bit wysyłania
MOV SBUF, #0 ; Wyślij 0 po porcie szeregowym


MAIN: ; Pętla główna programu
MOV SBUF, #0 ; Wyślij 0 po porcie szeregowym
JMP MAIN;

INTER:
JB TI, CHECK_WRITE ; Jeżeli TI równe 1 ( wysłanie ), to skocz do CHECK
CLR RI ; Jeśli nic nie wysłał, a coś odczytał, to wyzeruj RI
RETI ; i wróć z przerwania

CHECK_WRITE:
CLR TI ; Wyzeruj TI
RETI ; Wróć z przerwania

END
```

Wnioski

Program wykonuje swoje zadanie, należy pamiętać, że przerwanie od portu szeregowego wywołuje zarówno wysłanie jak i odebranie danych, dlatego w obsłudze przerwania należy uwzględnić te dwie sytuacje. Ponadto zapis to rejestru PCON mógłby być lepiej zrealizowany przy pomocy

instrukcji OR tak aby nie zerować innych bitów tego rejestru – niestety PCON nie jest adresowany bitowo.

3.3. Echo, odbieranie znaku z komputera i wysyłanie go do komputera

Opis działania programu

Program echo odbiera znak z komputera, a następnie odsyła go do komputera gdzie jest wyświetlany na ekranie, wykorzystanie przerwań od portu szeregowego.

Kod źródłowy (najważniejsze fragmenty)

```
org 0x23 ; Tu skacze program gdy nastąpi czytanie lub wysłanie przez port szeregowy
JMP INTER ; Skocz do obsługi przerwania od portu szeregowego

org 0x30 ; Konfiguracja - omijam ważne adresy
INITIALIZE: ; Etykieta konfiguracji

MOV SCON, #0x50 ; Tryb 1 działania portu szeregowego oraz ustawienie
                ; REN ( zezwolenie na czytanie )

MOV PCON, #0x80 ; Dzielimy przez 16 przy obliczaniu wartości do TH1

MOV TMOD, #0x20 ; Licznik 8-bit Timer 1

MOV TH1, #243 ; ; Wartość TH1 dla danego baud rate, które chcemy uzyskać obliczamy ze wzoru
:
                ; TH1 = 256 - (  [ fz / 12 * 16 ] / baud rate  )
                ; gdzie fz/12 to zegar systemowy
                ; a 32 to wartość stała
                ; po podstawieniu baud rate = 4800 otrzymano
                ; TH1 = 243


MOV TL1, #243 ; Pierwsze liczenie również od tej wartości

SETB TR1 ;
SETB EA ;
SETB ES
CLR RI ;
CLR TI ;
MOV A, SBUF ; Odbierz znak z portu szeregowego i wpisz go do akumulatora

MAIN: ; Pętla główna programu - nic nie robi
JMP MAIN;

INTER: ; Obsługa przerwania od portu szeregowego
JB RI, CHECK ; Jeśli RI jest jedynką to skocz do Check
CLR TI ; Jeśli RI nie jest jedynką, to przerwania spowodował TI
RETI ;
CHECK:
CLR RI ;
```

```
MOV SBUF, A ; Odczytaj znak z portu szeregowego i zapisz go do akumulatora
MOV A, SBUF ; Wyślij znak po porcie szeregowym
RETI ;

END
```

Wnioski

Odbieranie i wysyłanie danych odbywa się przy pomocy rejestrów SBUF oraz tzw. Akumulatora dzięki któremu dane są szybko wysyłane i odbierane. Należy pamiętać, że gdyby program główny również korzystał z akumulatora, to należy zastosować stos do przechowywania wartości akumulatora.

4. Zajęcia nr 4 – STM 1

4.1. Wyświetlanie dynamiczne na wyświetlaczach 7-segmentowych

Opis działania programu

Program działa tak jak stoper, odlicza sekundy oraz minuty i wyświetla je na czterech wyświetlaczach 7-egmentowych. Program wykorzystuje dwa przerwania, do zliczania oraz wyświetlania.

Kod źródłowy (najważniejsze fragmenty)

```
/* deklaracja zmiennych do zliczania */

int counter_1 = 0;
int SEK_1 = 0;
int counter_2 = 0;
int SEK_10 = 0;
int counter_3 = 0;
int MIN_1 = 0;
int counter_4 = 0;
int MIN_10 = 0;
int main_count = 0;
int slow_down = 0;

/* fragment funkcji konwertującej , mikrokonroler na każdy z pinów wystawia po kolei stany
wysokie iniskie */

void CONV_NUM(int number) // The function converts numbers into 7-SEG display
{
switch(number)
{
case 0 :
        HAL_GPIO_WritePin(SEG_A_GPIO_Port,SEG_A_Pin,GPIO_PIN_SET);
        HAL_GPIO_WritePin(SEG_B_GPIO_Port,SEG_B_Pin,GPIO_PIN_SET);
        HAL_GPIO_WritePin(SEG_C_GPIO_Port,SEG_C_Pin,GPIO_PIN_SET);
        HAL_GPIO_WritePin(SEG_D_GPIO_Port,SEG_D_Pin,GPIO_PIN_SET);
        HAL_GPIO_WritePin(SEG_E_GPIO_Port,SEG_E_Pin,GPIO_PIN_SET);
```

```
            HAL_GPIO_WritePin(SEG_F_GPIO_Port,SEG_F_Pin,GPIO_PIN_SET);
            HAL_GPIO_WritePin(SEG_G_GPIO_Port,SEG_G_Pin,GPIO_PIN_RESET);
            HAL_GPIO_WritePin(SEG_DP_GPIO_Port,SEG_DP_Pin,GPIO_PIN_RESET);
break;

/* Fragment przerwania od Timera systemowego 1 KHz

void HAL_SYSTICK_Callback()
{ // To przerwanie służy do wyświetlania odpowiednich wartości
        //{
                main_count++;
                if(main_count<4)
                { // Sterowanie anodami
                        HAL_GPIO_WritePin(COM1_GPIO_Port, COM1_Pin, GPIO_PIN_RESET);
                        HAL_GPIO_WritePin(COM2_GPIO_Port, COM2_Pin, GPIO_PIN_SET);
                        HAL_GPIO_WritePin(COM3_GPIO_Port, COM3_Pin, GPIO_PIN_SET);
                        HAL_GPIO_WritePin(COM4_GPIO_Port, COM4_Pin, GPIO_PIN_SET);
                        CONV_NUM(SEK_1); // Wyświetlanie
                }

/* Fragment przerwania od Timera skonfigurowanego przy pomocy środowiska CUBE na 1Hz

if  ( htim->Instance == TIM11) // TIM11 jest ustawiony na 1 Hz i generuje przerwanie co sekundę
{

        // Zliczanie sekund
        SEK_1++;
        if( SEK_1 == 10) // Przepełnienie
        {
                SEK_1=0;
        }

        counter_2++; // Zliczanie dziesiątek sekund
        if( counter_2 == 10)
        {
                counter_2=0;
                SEK_10++;
                if(SEK_10==6)
                {
                        SEK_10=0;
                }
        }
```

Wnioski

Program działał poprawnie na trzech z czterech wyświetlaczy, pierwszy wyświetlacz nie działał do końca poprawnie, wyświetlając nie zawsze odpowiednie liczby, powodem takiego zachowania mógł być fakt, że dobrana częstotliwość przy zmianie anod była zbyt duża – nie została ona obliczona, a była jednie dobierana eksperymentalnie.

5. Zajęcia 5 – STM 2

5.1. Wysyłanie i odbieranie danych przy pomocy DMA

Opis działania programu

Program odbiera dane z komputera , a następnie je odsyła korzystając z DMA (Direct Memory Access) , czyli modułu, który ma bezpośredni dostęp do pamięci RAM, dzięki czemu może wyręczyć procesor w przesyłaniu danych pomiędzy układami.

Kod źródłowy (najważniejsze fragmenty)

```
  while (1) // Pętla główna
  {
  /* USER CODE END WHILE */
          HAL_UART_Receive_DMA(&huart2, &received, 1); // DMA czeka na dane o rozmiarze
1
  /* USER CODE BEGIN 3 */

  }

// Przerwania odpowiedzialne za odesłanie znaku przy pomocy DMA

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
// Przerwania od odczytania danych
{ // Funkcja jest wywoływana po otrzymaniu 1 znaku
          size = sprintf( (char * ) data,  "1, %d", received);
          HAL_UART_Transmit_DMA(&huart2, data, size);
}
```

Wnioski

Użycie DMA pozwala na łatwe i szybkie wysyłanie danych pomiędzy układami peryferyjnymi jednocześnie zwalnia z tego obowiązku procesor.