Zadanie 1 PWR Racing Team Maksymilian Chudzik 273149

Program jest napisany na rejestrach.

```
#include<main.h>
```

W folderze Inc zrobiłem plik main.h gdzie wszystko includuje, pomaga to w uporządkowaniu kodu.

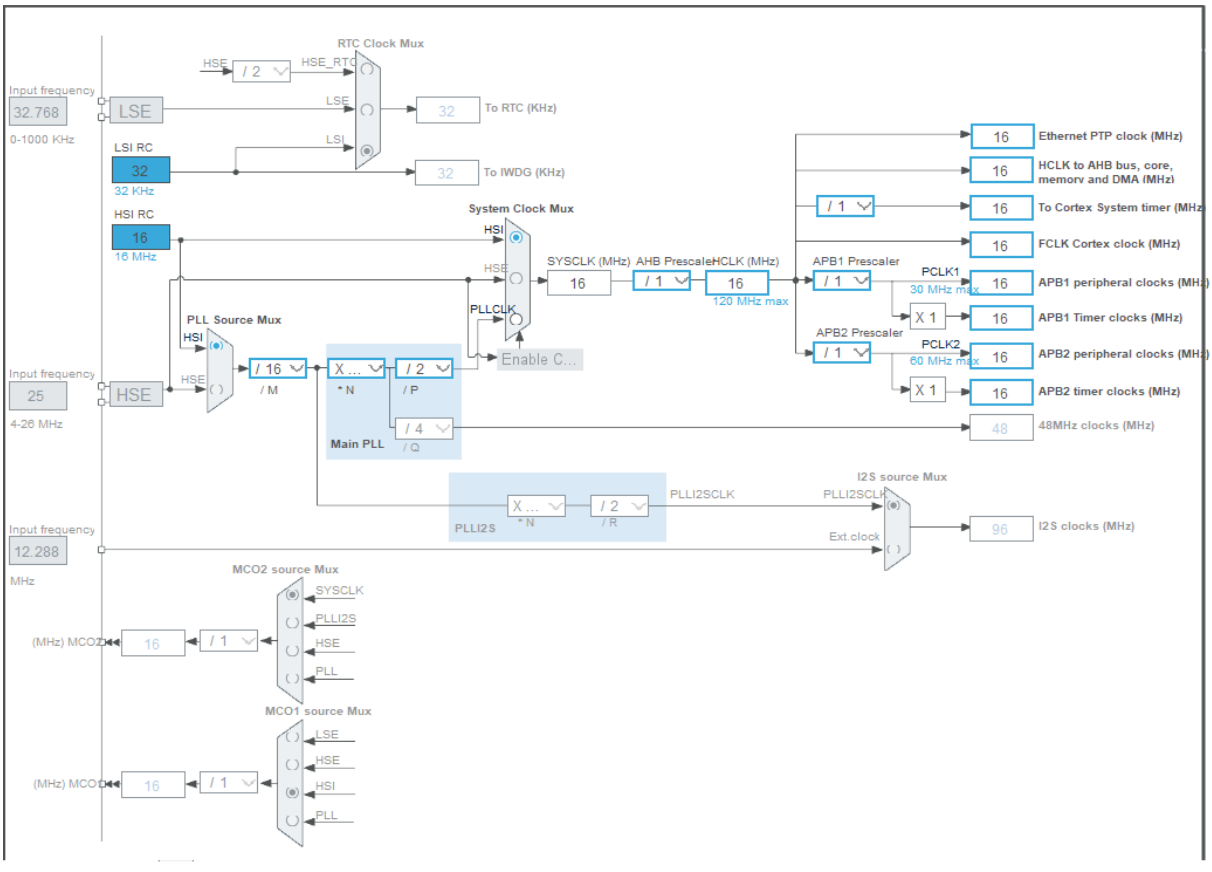
```
// Tworzenie zmiennej Tick do zliczania czasu
volatile uint32_t Tick;

uint32_t Timer_PA5;

// Zmiana stanu na przeciwny
#define TOGGLEPA5 GPIOA -> ODR ^= (GPIO_ODR_OD5)

// Zmienna do zmiany okresu czasu w ms
#define TimePA5 100  // T=1/5=0,2s=200ms   f=1/T=1/0,2=5Hz
// Jeden cykl zawiera wł i wył dlatego 200ms/2=100ms na czynność

uint32_t GetSystemTick(void);

void ConfigurePA5 (void);
void IWDG_Init(void);
void IWDG_Refresh(void);
```

Tworzenie zmiennych oraz funkcji.

TimePA5 ma wartość 100ms ponieważ, $T = \frac{1}{f}$ $T = \frac{1}{5} = 0{,}2s = 200ms$ jeden cykl zawiera się w 200ms a w jednym cyklu mamy dwa działania włączanie i wyłączanie diody na porcie PA5 $\frac{200ms}{2} = 100ms$

```c
int main(void)
{
    // Konfiguracja systemowego zegara SysTick na 1ms
    SysTick_Config(16000000/1000);

    ConfigurePA5();

    // Inicjalizacja Watchdoga
    IWDG_Init();

    // Podmiana czasu na aktualny czas
    Timer_PA5 = GetSystemTick();

    while(1)
    {
        if((GetSystemTick() - Timer_PA5) > TimePA5) // Sprawdzanie czy czas odpowiadający 5Hz został osiągnięty
                {
                    Timer_PA5 = GetSystemTick(); // Podmiana czasu na aktualny czas
                    TOGGLEPA5; // Zmiana stanu na PA5 / diody
                }

        // Odświeżanie Watchdoga
        IWDG_Refresh();
    }
}
```

Główna funkcja main w programie, na początku bierzemy systemowy zegar SysTick_Config



Ze schematu widzimy go jako To Cortex System timer (MHz) ma on wartość 16MHz ustawioną domyślnie, nie ma sensu używać skomplikowanego timera, dlatego używam najprostszego. Najpierw inicjalizuje pin, później watchdoga. Podmieniam czas zmiennej Timer_PA5 na aktualny, warunek if miał sens. Wchodzę w pętlę nieskończoną while. W if sprawdzam czy od

ostatniego sprawdzenia upłyną czas 100ms, jeżeli tak to wchodzę do if i z powrotem aktualizuję czas na naszej zmiennej oraz zmieniam stan na naszym pinie PA5 z 0 na 1 lub odwrotnie. Ostatnim zadaniem jest wywołanie funkcji IWDG_Refresh.

```
void ConfigurePA5 (void)
{
    // Włączenie portu A
    RCC -> AHB1ENR |= (RCC_AHB1ENR_GPIOAEN);

    // Ustawienie pinu PA5 na Output
    GPIOA -> MODER &= ~ (GPIO_MODER_MODE5_1);
    GPIOA -> MODER |=  (GPIO_MODER_MODE5_0);

    // Ustawienie prędkości na Very high speed ze względu na bardzo szybkie mruganie diodą
    GPIOA -> OSPEEDR |= (GPIO_OSPEEDER_OSPEEDR5);

    // Ustawienie pullUp/pullDown na no PU PD
    GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPD5);

    // Wyłączenie diody stan niski
    GPIOA -> BSRR &= ~(GPIO_BSRR_BS5);
```

Funkcja, która konfiguruje nam pin PA5 tak aby działał poprawnie. Na początku muszę aktywować pin PA5, zmieniam rejestr RCC_AH1ENR na 1, wiemy to z dokumentacji Reference Manual z rozdziału 5.3.5 . Następnie zmieniam wartość MODER na 01 aby wartość była zgodna z Output (jest to pin do którego podpięta jest dioda której wysyłamy dane a nie przyjmujemy od niej dane), wiemy to z dokumentacji Reference Manual z rozdziału 6.4.1 . Następnie zmieniam prędkość na Very high speed, nie jest to wymagane dla tak prostego działania, teoretycznie wystarczyło by Medium speed.

Bits 2y:2y+1 **OSPEEDRy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O output speed.
00: Low speed
01: Medium speed
10: High speed
11: Very high speed

Później zmieniam wartość PUPDR na NO PullUp PullDown czyli floating, wiemy to z dokumentacji Reference Manual z rozdziału 6.4.4 . Ostatnią czynnością w funkcji upewniam się i wyłączam diodę ustawiając stan niski.

```c
void SysTick_Handler(void)
{
    Tick++; // Zwiększanie Timera systemowego
}

uint32_t GetSystemTick(void)
{
    return Tick; // Zwracanie Timera systemowego
}
```

Kolejne funkcje SysTick_Handler zwiększa nam wartość Tick o 1, a funkcja GetSystemTick zwraca nam zmienną Tick czyli aktualny czas.

```c
void IWDG_Init(void)
{
    // Odblokowanie dostępu do rejestrów IWDG_KR
    IWDG->KR = 0x5555;

    // Ustawienie preskalera na 32
    IWDG->PR = 4;

    // Ustawienie wartości przeładowania na maksymalną wartość
    IWDG->RLR = 0xFFF;

    // Ponowne odblokowanie dostępu i rozpoczęcie pracy IWDG
    IWDG->KR = 0xCCCC; // Uruchomienie Watchdoga
}

void IWDG_Refresh(void)
{
    // Zapisanie klucza odświeżającego do rejestru IWDG_KR
    IWDG->KR = 0xAAAA; // Odświeżenie Watchdoga
}
```

W funkcjach obsługujących WatchDoga wszystko wiemy z dokumentacji Reference Manual z rozdziału 17 a zmianę rejestrów z rozdziału 17.4 . Na początku ustawiamy IWDG_KR na 0x5555, umożliwia nam to zmianę preskalera IWDG_PR na dzielnik 32 oraz wartość przeładowania IWDG_RLR na maksymalną wartość. Ostatnią zmianą w funkcji IWDG_Init rozpoczynamy pracę watchdoga. Ostatnia funkcja IWDG_Refresh resetuje nam wartość watchdoga, tak aby watchdog się nie przeładował.