Mikrokontrolery ARM

Laboratorium 1

Krzysztof Pierczyk
3 kwietnia 2020

Kolejne zajęcia laboratoryjne przywidywały zapoznanie uczestników z podstawowymi elementami mikrokontrolerów z rodziny STM32 w postaci systemu dystrybucji zegarów, modułu portów ogólnego przeznaczenia (*ang. General Purpose Input Output, GPIO)* oraz kontrolera przerwań zewnętrznych EXTI. W ramach zadania uczestnicy mieli za zadanie skonfigurować podstawowe elementu systemu oraz napisać krótką aplikację, w trzech wersjach, która wykorzystywałaby porty GPIO.

Konfiguracja sygnałów zegarowych i pętli PLL

Konfiguracja sygnałów taktujących, zarówno rdzeń, jak i peryferia, została zaimplementowana na bazie funkcji `uc_periph_setup()`, udostępnionej w materiałach wykładowych. Wprowadzone w niej modyfikacje były w większości kosmetyczne. Sama konfiguracja sprowadzała się do ustalenia wartości dzielników i mnożnika pętli pętli PLL z użyciem oprogramowania **STM32CubeMX** oraz wyznaczenia nowej wartości opóźnień dla kontrolera pamięci Flash. Zgodnie z treścią zadania sygnały taktujące zostały ustalone na:

$$HCLK = 100MHz$$

$$APB1 = 50MHz$$

$$APB2 = 100MHz$$

Wartości te osiągnięto poprzez ustawienie, jak źródła sygnału taktującego SYSCLK, pętli PLL, która generowała sygnał na bazie zewnętrznego rezonatora kwarcowego HSE o częstotliwości 8MHz. Kolejne dzielniki taktowania zostały skonfigurowane do wartości:

$$Dzielnik\ SYSCLK = 1$$

$$Dzielnik\ magistrali\ APB1 = 2$$

$$Dzielnik\ magistrali\ APB2 = 1$$

$$Dzielnik\ M\ pętli\ PLL = 4$$

$$Dzielnik\ P\ pętli\ PLL = 2$$

Mnożnik pętli PLL ustawiono na 100. W ramach oszczędzania energii wyłączono także moduł wewnętrznego oscylatora RC (HSI). Zgodnie z wartościami podanymi w tabeli omawianej na wykładzie dla $90MHz < HCLK < 120MHz$ i zasilania układu na poziomie 3V, opóźnienia dla kontrolera pamięci Flash ustawiono na 3 WS (wait states).

Zgodnie z treścią zadania, zwrócono także uwagę na różnicę w częstotliwości migania diody (kontrolowanej przez program *blink*) przed i po przetaktowaniu układu. Różnic tych jednak nie zaobserwowano. Prowadzi to do wniosku, że sama konfiguracja systemu zegarowego nie zmienia taktowań systemowych, a jedynie, co najwyżej, dobiera inną

konfigurację dzielników. Wniosek ten może być szybko zweryfikowany po zajrzeniu do pliku *platform_startup.cpp* obecnego w katalogu płytki STM32F411E-DISCO.

Obsługa portów GPIO i kontrolera EXTI

Drugie zadanie polegało na napisaniu krótkiego programu, który zliczałby naciśnięcia przycisku USER, dostępnego na płytce ewaluacyjnej (modulo 16) i wartość tą wyświetlałby w naturalnym kodzie binarnym na diodach *LD3*, *LD4*, *LD5*, *LD6.* Program taki, zgodnie z treścią zadania, został sporządzony w trzech wersjach:

1. Z wykorzystaniem API systemu ISIX i aktywnego odpytywania (*ang. Polling*)
2. Z wykorzystaniem (do sterowania portów GPIO) niskopoziomowego API LL (*ang. Low Level Library*), dostarczanego przez STM, oraz aktywnego odpytywania
3. Z wykorzystaniem API LL oraz kontrolera przerwań zewnętrznych EXTI, do ograniczenia częstotliwości odczytywania stanu wejścia podłączonego do przycisku

API systemu ISIX i aktywne odpytywanie

Pierwszy wariant był jednocześnie najprostszym do zrealizowania. ISIX-owy interfejs umożliwia szybką i intuicyjną konfigurację portów GPIO w dowolnym układzie. W przeciwieństwie do API LL pozwala on na konfigurację wielu pinów na raz.

```
while(true){

    // Display counter on LEDs
    periph::gpio::set(led_4, counter & (1U << 3));
    periph::gpio::set(led_3, counter & (1U << 2));
    periph::gpio::set(led_5, counter & (1U << 1));
    periph::gpio::set(led_6, counter & (1U << 0));

    // Debouncing
    if(!periph::gpio::get(button)){
        debounce_begin = isix::get_jiffies();
        button_pressed = false;
    }

    if(!button_pressed && isix::timer_elapsed(debounce_begin ,debounce_ms)){
        button_pressed = true;
        ++counter;
        if(counter > 15)
            counter = 0;
    }

}
```

Rysunek 1. Główna pętla programu stworzonego z wykorzystaniem API systemu ISIX i aktywnego odpytywania

Na początku pętli stany kolejnych diod ustawiane są zgodnie z wartościami bitów licznika `counter`. Następnie odczytywany jest aktualny stan wejścia podpiętego do przycisku. Jeżeli przycisk nie jest wciśnięty, pobierana jest wartość aktualnego czasu. Dzięki temu, kolejna instrukcja warunkowa, wykonywana w momencie, gdy od chwili zapisanej w zmiennej `debounce_begin` upłynął czas przewidziany na debouncing, nie wykona się tak długo, jak długo przycisk pozostanie niewciśnięty. Gdy jednak ten stan rzeczy się zmieni, pomiary czasu przestaną być aktualizowane i różnica między czasem zapisanym, a aktualnym zacznie rosnąć. Dzięki temu, po pewnym czasie, końcowe instrukcje warunkowe również się wykonają, inkremetując tym samym licznik `counter`.

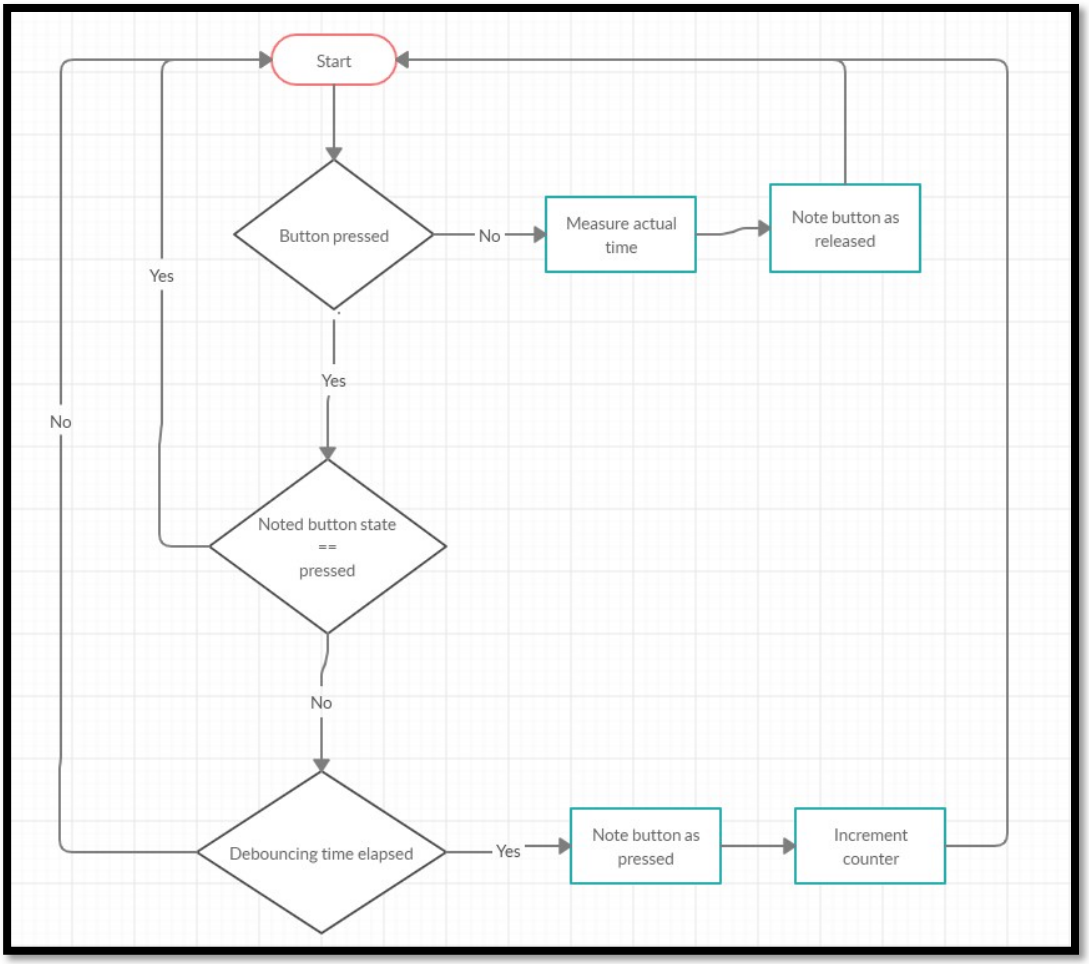
Biblioteka LL i aktywne odpytywanie

W wersji drugiej programu, zmiany zaszły jedynie od strony technicznej. Logika programu pozostała niezmieniona, a jedynie wywołania funkcji systemu ISIX zostały zastąpione tymi z biblioteki LL. Jak widać na listingu zamieszonym poniżej porty GPIO zostały zainicjalizowane z wykorzystaniem dedykowanych struktur definiowanych w plikach biblioteki. Warto zwrócić uwagę na fakt, że stosunek objętości kodu, do realizowanej funkcjonalności jest o wiele większy niż w przypadku interfejsu systemu ISIX.

```
// Configure LEDs
LL_GPIO_InitTypeDef led_init_struct{
    .Mode = LL_GPIO_MODE_OUTPUT,
    .Speed = LL_GPIO_SPEED_FREQ_LOW,
    .OutputType = LL_GPIO_OUTPUT_PUSHPULL
};
led_init_struct.Pin = LL_GPIO_PIN_12;
LL_GPIO_Init(GPIOD, &led_init_struct);
led_init_struct.Pin = LL_GPIO_PIN_13;
LL_GPIO_Init(GPIOD, &led_init_struct);
led_init_struct.Pin = LL_GPIO_PIN_14;
LL_GPIO_Init(GPIOD, &led_init_struct);
led_init_struct.Pin = LL_GPIO_PIN_15;
LL_GPIO_Init(GPIOD, &led_init_struct);

// Configure button
LL_GPIO_InitTypeDef button_init_struct{
    .Pin = LL_GPIO_PIN_0,
    .Mode = LL_GPIO_MODE_INPUT,
    .Pull = LL_GPIO_PULL_NO
};
LL_GPIO_Init(GPIOA, &button_init_struct);
```

Rysunek 2. Konfiguracja portów GPIO i kontrolera EXTI z wykorzystaniem biblioteki LL

Na poniższym diagramie przedstawiono algorytm programowej realizacji niwelacji drgań styków wykorzystany w pierwszych dwóch wariantach programu.



Rysunek 3. Algorytm programowej realizacji niwelacji drgań styków z wykorzystaniem aktywnego odpytywania

Biblioteka LL i przerwania z kontrolera EXTI

Ostatni wariant programu różnił się nie tylko użytymi narzędziami do obsługi układów peryferyjnych, ale wprowadził także zmiany w logice działania układu. Tym razem, główna pętla programu nie odczytuje za każdym razem stanu przycisku, a jedynie sprawdza stan flagi ustawianej przez przerwanie w momencie wykrycia narastającego zbocza. Detekcja ta jest jedynym zadaniem procedury obsługi przerwania. Warto w tym miejscu zauważyć, że zmienną globalną dzieloną przez główny wątek i przerwanie **należy zadeklarować z kwalifikatorem volatile**. W przeciwnym wypadku, rdzeń, w procedurze obsługi przerwania, usilnie ładuje ją do rejestru R3, który, po zakończeniu procedury, jest ładowany ze stosu bez wcześniejszego zapisania do pamięci. Fakt ten mimo, iż może się wydawać oczywisty, potrafił napsuć sporo krwi i zapewnić niezapomniane wrażenia związane z kilkugodzinnym szukaniem błędu na poziomie kodu źródłowego.

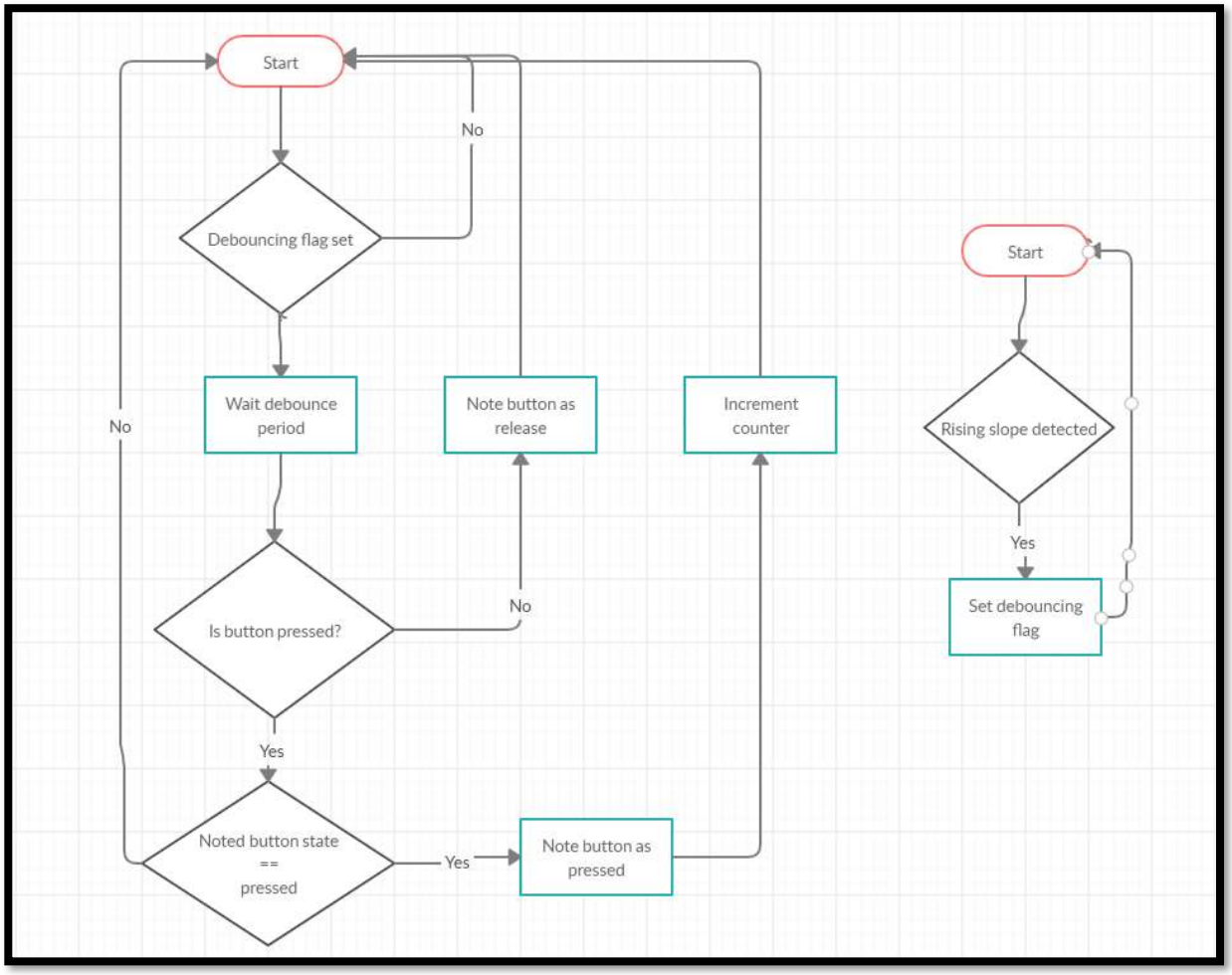
```
        ...

        // Check if debounce is active
        if(debounce_active){

            // Wait for input to stabilize
            isix::wait_ms(debounce_ms);

            // Check button state
            if(LL_GPIO_IsInputPinSet(GPIOA, (1U << 0))){
                if(!button_pressed){
                    button_pressed = true;
                    ++counter;
                    if(counter > 15)
                        counter = 0;
                }
            }
            else
                button_pressed = false;
        }
    }

    extern "C" {

        void exti0_isr_vector() {

            // Set debounce indicator
            debounce_active = true;

            // Clear interrupt flag
            LL_EXTI_ClearFlag_0_31(LL_EXTI_LINE_0);
        }

    }
```

Rysunek 4. Realizacja programowej eliminacji drgań styków z wykorzystaniem przerwania

Po wykryciu narastającego zbocza wyzwalane jest przerwanie, które ustawia flagę `debounce_active`. Flaga ta jest sprawdzana w pętli głównej. Po jej ustawieniu, program odczekuje czas równy okresowi debouncingu i sprawdza stan przycisku. Jeżeli jest on wysoki, zostaje on uznany za wciśnięty, a licznik jest inkrementowany. W przeciwnym wypadku przerwanie uznawane jest za wynik drgania styków.

Przedstawiony algorytm nie jest oczywiście optymalny, a nawet można by powiedzieć, żebezsensowny. Eliminuje on jeden blok warunkowy z pętli głównej, ale wprowadza zmiany kontekstu procesora związane z wywołaniem procedury obsługi przerwania. Program ten miał jednak za zadanie, w praktyczny sposób, zapoznać uczestnika laboratorium z konfiguracją podstawowych układów peryferyjnych. Na tej płaszczyźnie w

pełni spełnił on swoją rolę. Poniżej przedstawiono kolejny diagram obrazujący zasadę działania algorytmu.

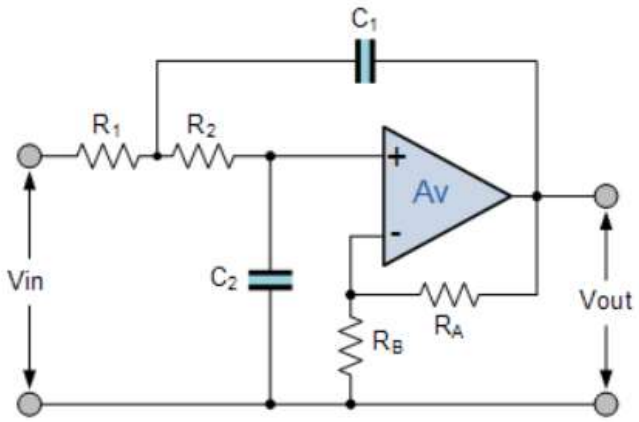


Rysunek 5. Algorytm programowej eliminacji styków z wykorzystaniem przerwania

Eliminacja drgań styków

Niestabilność sygnału w czasie zmiany stanu przełączników mechanicznych wołana jest mechanicznymi drganiami metalowych okładek zapewniających zwarcie dwóch części obwodu. Drgania te, na płaszczyźnie mechanicznej, są bardzo trudne do wyeliminowania, gdyż ich częstotliwość jest bardzo dużo, a same elementy stosunkowo niewielkie. Eliminacja niestabilności sygnału jest **konieczna w przypadku pracy z szybkimi układami** (np. większością układów cyfrowych), gdyż ich czas reakcji jest mniejszy niż okres niestabilny przełączania i może dochodzić do omyłkowego zliczania impulsów, które nie były zamierzone. W przypadku znacznej części układów analogowych, np. przełączników obwodów zasilania stałego, nie jest to tak krytyczne, gdyż z założenia obwody te mają dużą bezwładność i skutki drgań styków są dla nich w praktyce niewidoczne.

Walkę z tym problemem podejmuje się nie tylko w domenie programowej, jak zostało to zaprezentowane w niniejszym sprawozdaniu. Istnieją także dedykowane układy scalone, które zapewniają filtrację składowych wysokoczęstotliwościowych i tym samym poprawną interpretację zmiany stanu przełączników. Najprostsze układy, które mogą poprawić charakterystykę pracy typowego przycisku, to proste filtry RC. Poniżej przedstawiono propozycję aktywnego filtra RC drugiego rzędu z wykorzystaniem wzmacniacza operacyjnego ogólnego przeznaczenia.



Rysunek 6. Przykładowy dolnoprzepustowy filtr aktywny drugiego rzędu

Obliczenie wymaganych wartości elementów biernych jest bardzo proste i wymaga od projektanta zaledwie paru obliczeń. Po pierwsze, w przypadku układów typu micro-switch, z których korzystaliśmy w czasie laboratorium, zakres napięć wejściowych i wyjściowych filtra powinien być identyczny. Stąd wnioskujemy, że jego wzmocnienie musi być jak najbliższe jedności. Wzmocnienie powyższego układu otrzymujemy ze wzoru:

$$\mathrm{G} = 1 + \frac{\mathrm{R_A}}{\mathrm{R_B}}$$

Jeżeli zatem chcemy utrzymać tę wartość jak najbliżej jedynki, to stosunek R_A do R_B musi być jak najmniejszy. Możemy przyjąć wartości $R_A = 1k\Omega$, $R_B = 1M\Omega$.

Przyjmując założenie, że użytkownik nie jest w stanie wcisnąć przycisku więcej niż 15 razy na sekundę dochodzimy do wniosku, że sygnały o częstotliwości powyżej 15Hz są dla nas bezużyteczne. Dla bezpieczeństwa, warto przyjąć pewien margines błędu. Wynika to chociażby z faktu, że szerokość pasma przejściowego filtra dolnoprzepustowego tego typu jest stosunkowo duża. Stąd, częstotliwość odcięcia możemy przyjąć na poziomie 25Hz.

Przy założeniu, że wartości rezystancji R_1 i R_2 są identyczne i równe R oraz, że wartości pojemności C_1 i C_2 są identyczne i równe C, wzór na częstotliwość odcięcia filtra możemy zapisać jako:

$$f_C = \frac{1}{2\pi RC}$$

Przekształcając wzór otrzymujemy $RC = \frac{1}{2\pi f_C} \approx 6.3663\ [ms]$. Oczywiście dobranie odpowiedniej pary rezystancji i pojemności zależy przede wszystkim od wartości elementów biernych dostępnych na rynku, jednak w celach czysto teoretycznych, przyjmując dowolne wartości obu parametrów możemy ustalić

$$R = 1050\Omega$$

$$C = 6\mu F$$

Filtr taki posiad częstotliwość odcięcia równą w przybliżeniu 25.26Hz, co jest wynikiem zupełnie zadowalającym.