Mikrokontrolery ARM

Laboratorium 2

Krzysztof Pierczyk
17 kwietnia 2020

WSTĘP

Na kolejnych zajęciach laboratoryjnych zajęliśmy się układami czasowo-licznikowymi obecnymi w mikrokontrolerach STM32. Przygotowane zadania wymagały wykorzystania większości funkcjonalności przez nie oferowanych począwszy od możliwości cyklicznego zgłaszania przerwań, przez generowanie przebiegów prostokątnych, na pomiarze czasu trwania impulsu kończąc. Nie wymuszały one skorzystania z mechanizmów synchronizacji liczników czy użycia zewnętrznych sygnałów taktujących, ale i te moduły można było przy odrobinie wysiłku zastosować.

ZADANIE 1

Pierwsze z zadań polegało na generowaniu przez układ czasowo-licznikowy TIM1 przerwań, które zmieniałyby stany diod LED3 – LED6 w taki sposób, aby uzyskać efekt „biegającego punktu". Przerwania te miały być generowane z okresem 200m. Główną część programu stanowiła sama konfiguracja układu TIM1. Jako, że pozostałe elementy, wykorzystywane przez program, stanowiły część poprzednich zajęć laboratoryjnych, poniżej umieszczono jedynie ten fragment kodu.

Procedura obsługi przerwania (*ang. Interrupt Service Routine, ISR*) umieszczona w wektorze `tim1_up_tim10_isr_vector` sprowadza się do sprawdzenia wartości licznika (inkrementowanego co przerwanie mod(3)), zgaszenia diody poprzedniej i zapalenia następnej. Obsługa portów GPIO odbywa się poprzez API systemu ISIX.

W ramach ciekawostki warto zwrócić uwagę na sytuację, która przydarzyła się w czasie pisania programu do tego zadania. Przy pierwszym podejściu do pisania udało się w poprawny sposób skonfigurować układ licznikowy, lecz program zachowywał się tak, jak gdyby wyzwalane były zawsze dwa przerwania z rzędu, tj. aktywowane były tylko dwie wybrane diody, a okres ich świecenia równy był przewidywanym 200ms. Po długich zmaganiach przyczyną błędu okazało się **zbyt późne gaszenie flagi przerwania generowanej wewnątrz modułu TIM1**.

```
// Timer1 configuration for periodic interrupts
void TIM1_config(void){

    // Enable clock from APB2 for the TIM1
    LL_APB2_GRP1_EnableClock(LL_APB2_GRP1_PERIPH_TIM1);

    // TIM1 configuration
    LL_TIM_InitTypeDef TIM1_struct{
        .Prescaler         = __LL_TIM_CALC_PSC(100000000 , 10000),
        .CounterMode       = LL_TIM_COUNTERMODE_UP,
        .Autoreload        = __LL_TIM_CALC_ARR(
                                 100000000 ,
                                 __LL_TIM_CALC_PSC(100000000 , 10000),
                                 LED_FREQ
                             ),
        .RepetitionCounter = 0
    };
    LL_TIM_Init(TIM1, &TIM1_struct);

    // Enable overflow (update) interrupts
    LL_TIM_EnableIT_UPDATE(TIM1);

    // Active TIM1 interrupt in NVIC module
    isix::set_irq_priority(
        TIM1_UP_TIM10_IRQn,
        isix_irq_prio_t{
            .prio = 1,
            .subp = 7
        }
    );
    isix::request_irq(TIM1_UP_TIM10_IRQn);

    // Enable TIM1
    LL_TIM_EnableCounter(TIM1);

    // Trigger the first update event by hand
    LL_TIM_GenerateEvent_UPDATE(TIM1);
}
```

Jak wiadomo, flagi przerwań w kontrolerze NVIC gaszone są automatycznie w ramach obsługi procedury przerwania. Jednak flagi generowane wewnątrz układów peryferyjnych mikrokontrolera gaszone automatycznie nie są. Okazuje się, że jeśli instrukcja czyszczenia flagi zostanie wykonana zbyt późno, wyjście z procedury obsługi przerwania może nastąpić przed następnym cyklem odpowiedniej szyny danych co spowoduje, że NVIC potraktuje starą wartość flagi jako wystąpienie kolejnego przerwania i rozpocznie ponowne wykonanie ISR.

Zatem jedną z dodatkowych nauk płynących z tych zajęć laboratoryjnych jest to, że flagi przerwań należy gasi jak najwcześniej (przynajmniej w przypadku mikrokontrolerów z rodziny STM32).

ZADANIE 2

Zadanie drugie wymagało nie tylko skonfigurowaniu układu podstawy czasu licznika TIM4, ale również modułu *Capture/Compare* w taki sposób, aby generował on na pinach podłączonych do diod LED sygnał prostokątny o częstotliwości 1Hz oraz wypełnieniu równym 50%. Sygnał taki miał pojawić się na diodach LED4, LED3 oraz LED5 i w każdym przypadku miał być przesunięty w fazie o kolejno 0°, 180° oraz 270°.

Aby urozmaicić zadanie i poprawić efekty wizualne będące wynikiem działania programu do tego zestawu dołączono diodę LED6 oraz zmieniono konfigurację przesunięć fazowych. W efekcie otrzymano:

- Przesunięcie 0° na diodzie LED3
- Przesunięcie 90° na diodzie LED5
- Przesunięcie 180° na diodzie LED6
- Przesunięcie 270° na diodzie LED4

Konfiguracja taka skutkuje powstanie efekt „biegającego punktu" z „zakładkami", tj. z momentami, w których dwie sąsiednie diody świecą się jednocześnie. Okresy te równe są połowie czasu świecenia dioda, a zatem ¼ okresu sygnału prostokątnego.

Ponownie fragment kodu odpowiedzialnego za konfigurację układu czasowo licznikowego został przedstawiony na poniższym listingu. Zarówno do inicjalizacji układu podstawy czasu jak i modułu Capture/Compare wykorzystane zostały struktury inicjalizacyjne z biblioteki *Low Level Library* (LL).

Podstawa czasu została skonfigurowana tak, aby **przepełnienie następowało z częstotliwością 2Hz**, co w połączeniu z trybem **trybem *Toggle on Match*** modułu C/C pozwoliło uzyskać generowanie fal prostokątnych o częstotliwości 1Hz. Kanały podpięte do diod LED3 i LED5 zostały ustawione na **polaryzację prostą**, a te podłączone na diod LED 6 i LED4 na **polaryzację odwróconą**. Po ustawieniu wartości rejestrów CCRx dla diod LED3 i LED5 na 0, a dla diod LED6 i LED4 na połowę wartości rejestru ARR zagwarantowało to pożądane przesunięci fazowe.

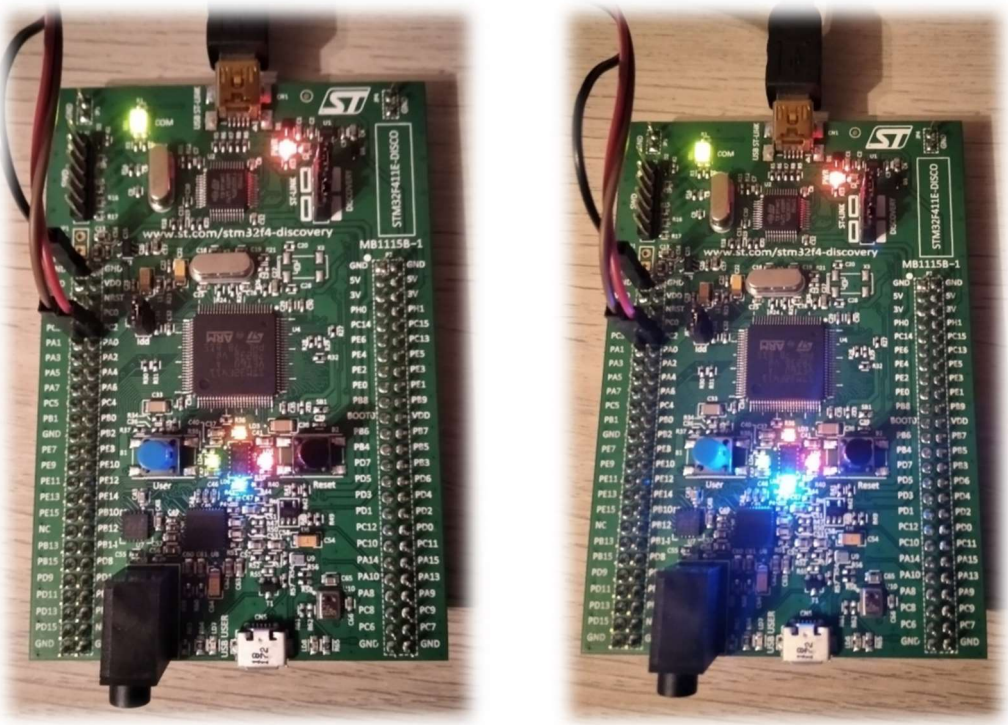
```c
void TIM4_config(void){

    // Enable clock from APB1 for the TIM4 periph
    LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_TIM4);

    // Enable ARR preloading
    LL_TIM_EnableARRPreload(TIM4);

    // TIM4 configuration
    LL_TIM_InitTypeDef TIM4_struct{
        .Prescaler         = __LL_TIM_CALC_PSC(100000000 , 10000),
        .Autoreload        = __LL_TIM_CALC_ARR(
                                100000000,
                                __LL_TIM_CALC_PSC(100000000 , 10000),
                                2
                             )
    };
    LL_TIM_Init(TIM4, &TIM4_struct);

    // TIM4 compare/capture configuration
    LL_TIM_OC_InitTypeDef TIM4_CC_struct{
        .OCMode       = LL_TIM_OCMODE_TOGGLE,
        .OCState      = LL_TIM_OCSTATE_ENABLE
    };
    // Channel_2 = LED3
    TIM4_CC_struct.OCPolarity   = LL_TIM_OCPOLARITY_LOW,
    TIM4_CC_struct.CompareValue = 0;
    LL_TIM_OC_EnablePreload(TIM4, LL_TIM_CHANNEL_CH2);
    LL_TIM_OC_Init(TIM4, LL_TIM_CHANNEL_CH2, &TIM4_CC_struct);
    // Channel_2 = LED5
    TIM4_CC_struct.OCPolarity   = LL_TIM_OCPOLARITY_HIGH,
    TIM4_CC_struct.CompareValue = (uint32_t)(TIM4_struct.Autoreload / 2);
    LL_TIM_OC_EnablePreload(TIM4, LL_TIM_CHANNEL_CH3);
    LL_TIM_OC_Init(TIM4, LL_TIM_CHANNEL_CH3, &TIM4_CC_struct);
    // Channel_2 = LED6
    TIM4_CC_struct.OCPolarity   = LL_TIM_OCPOLARITY_HIGH,
    TIM4_CC_struct.CompareValue = 0;
    LL_TIM_OC_EnablePreload(TIM4, LL_TIM_CHANNEL_CH4);
    LL_TIM_OC_Init(TIM4, LL_TIM_CHANNEL_CH4, &TIM4_CC_struct);
    // Channel_2 = LED4
    TIM4_CC_struct.OCPolarity   = LL_TIM_OCPOLARITY_LOW,
    TIM4_CC_struct.CompareValue = (uint32_t)(TIM4_struct.Autoreload / 2);
    LL_TIM_OC_EnablePreload(TIM4, LL_TIM_CHANNEL_CH1);
    LL_TIM_OC_Init(TIM4, LL_TIM_CHANNEL_CH1, &TIM4_CC_struct);

    // Enable TIM4
    LL_TIM_EnableCounter(TIM4);

    // Initialize shadow registers
    LL_TIM_GenerateEvent_UPDATE(TIM4);
}
```

ZADANIE 3

Zadanie trzecie również poruszało kwestię generacji fali prostokątnej z użyciem układu TIM4, jednak tym razem nacisk położony został na manipulację stopniem wypełnienia fali, a nie jej przesunięciem fazowym. Zgodnie z treścią zadania 3 z 4 diod LED zostały zasilone falą prostokątną o wypełnieniu kolejno 20%, 40% oraz 60%. Ostatni z diod została zainicjalizowana sygnałem prostokątnym o wypełnieniu 0 (tj. stałą wartością napięcia równą 0).

W sposób analogiczny do przedstawionego w sprawozdaniu z laboratorium pierwszego skonfigurowano przerwania od przycisku USER poprzez moduł przerwań zewnętrznych *EXTI*. Procedura obsługi przerwania została wykorzystana do realizacji programowej eliminacji drgań styków. Wykrycie faktycznego wciśnięcia przycisku powoduje zwiększenie wartości rejestru CCR4 o 50, co stanowi 1/10 wartości rejestru ARR powiększonej o 1. W praktyce skutkuje to **zwiększeniem wypełnienia sygnału prostokątnego o 10 punktów procentowych**. Gdy liczba w rejestrze osiągnie wartość (ARR + 1), kolejne wciśnięcia przycisku skutkują obniżeniem wartości o 50. Kod wykorzystany w czasie realizacji tego zadania jest analogiczny do kodu z zadania drugiego poza dwiema różnicami:

- Polaryzacje wszystkich kanałów *CO* zostały ustawione na proste
- Tryb pracy wyjść kanałów zmieniono z *Toggle on Match* na *PWM1*/*PWM2*



Rysunek 1. Porównanie intensywności świecenia diody LED przy kolejno 10% i 100% wypełnienia fali prostokątnej

Wnioski dotyczące trybów PWM

Badania przeprowadzono przy ustawieniu modułu czasowo-licznikowego w trybie zliczania do góry. W takim przypadku oba tryby PWM wykazują identyczne możliwości w kontekście sterowania średnią wartością napięcia generowaną na obciążeniu.

- Z użyciem trybu PWM1 można osiągnąć wartości wypełniania z zakresu [0%; 100%] manipulując wartością rejestru CCRx pomiędzy wartościami 0 i (ARR+1) zgodnie ze wzorem

$$Duty\ Cycle = \frac{CCRx}{ARR + 1}$$

- W trybie PWM2 powyższa zależność jest odwrócona – wypełnienie 0% uzyskujemy przy CCRx ≥ ARR + 1, a 100% przy CCRx = 0. Poziom wypełnienia określa wzór

$$Duty\ Cycle = \ 1 - \frac{CCRx}{ARR + 1}$$

Oczywiście w obu przypadkach, gdy wartość ARR będzie wynosiła 2^{16} (lub 2^{32}dla układów TIM2 i TIM5) wpisanie do CCRx wartości (ARR + 1) nie będzie możliwe, zatem w trybie PWM1 nie będzie możliwe osiągnięcie wypełnieni 100%, a w trybie PWM2 wypełnienia 0%. Rozdzielczość wypełnienia jest niezależna od trybu i wynosi

$$\frac{1}{ARR + 1}$$

W trybie zliczania *Center-aligned* rozdzielczość ta jest dwukrotnie zmniejszona, natomiast niezależnie od wartości ARR możliwe jest osiągnięcie dowolnego wypełnienia fali. Wzory to opisujące kolejno dla trybu PWM1 i PWM2 prezentują się następująco:

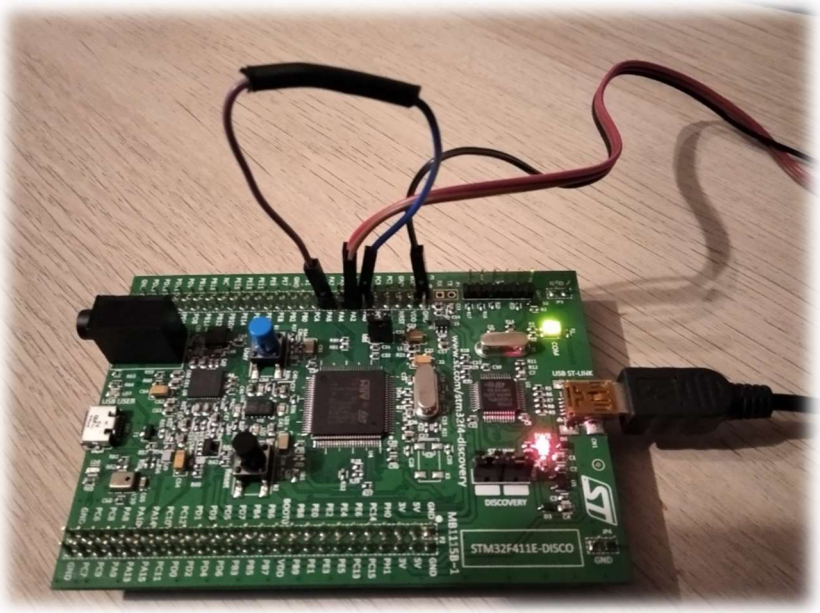
$$Duty\ Cycle_{PWM1} = \frac{CCRx}{ARR}$$

$$Duty\ Cycle_{PWM1} = \ 1 - \frac{CCRx}{ARR}$$

ZADANIE 4

Ostatnie z zadań poruszało kwestię wykorzystania modułu *C/C* do pomiaru długości trwania impulsów, a konkretnie pomiaru częstotliwości fali prostokątnej podanej na jeden z pinów mikrokontrolera. Ze względu na brak dostępu do sprzętu laboratoryjnego, w tym generatora sygnałów, **kanał wejściowy CH1 układu TIM3 został podłączony do wyjścia CH2 układu TIM5** skonfigurowanego w trybie PWM1.

Preskaler licznika TIM5 ostawiony został tak, aby generować sygnał taktujący o częstotliwości 10MHz. Pozwala to na regulację częstotliwości sygnału prostokątnego w zakresie [152.59Hz; 5MHz]. Aby kontrolować częstotliwość sygnału PWM skonfigurowano również moduł przerwań *EXTI* do obsługi przycisku USER (z uwzględnioną programową procedura eliminacji drgań styku). Wykryte wciśnięcia powodują zmianę wartości w rejestrze ARR na jedną z trzech predefiniowanych. Wartości te określają **częstotliwości wyjściowej fali prostokątnej na poziomie 10kHz, 100kHz oraz 1MHz**. Przy każdej zmianie częstotliwości modyfikowany jest również rejestr CCR1 tak, aby utrzymać wypełnienie na poziomi 50%.



Rysunek 2. Połączenie kanału wyjściowego układu TIM5 (generatora) z kanałem wejściowym układu TIM3

Konfigurację układu TIM3 rozpoczęto od ustawienia podstawy czasu na maksymalną możliwą częstotliwość, tj. 100MHz przy jednoczesnym ustawieniu wartości rejestru ARR na maksymalną (2^{16}). Taki dobór parametrów pozwala osiągnąć największą dostępną rozdzielczość pomiaru na poziomie 10ns przy pomiarze okresu sygnału.

```
void TIM3_config(){

    // Enable clock from APB1 for the TIM1 periph
    LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_TIM3);

    // Enable ARR preloading
    LL_TIM_EnableARRPreload(TIM3);

    // TIM3 configuration
    LL_TIM_InitTypeDef TIM3_struct{
        .Prescaler  = 0,
        .Autoreload = 0xffff
    };
    LL_TIM_Init(TIM3, &TIM3_struct);

    // TIM3 compare/capture configuration
    LL_TIM_IC_InitTypeDef TIM3_IC_struct{
        .ICPolarity    = LL_TIM_IC_POLARITY_RISING,
        .ICActiveInput = LL_TIM_ACTIVEINPUT_DIRECTTI,
        .ICPrescaler   = LL_TIM_ICPSC_DIV1,
        .ICFilter      = LL_TIM_IC_FILTER_FDIV1
    };
    LL_TIM_IC_Init(TIM3, LL_TIM_CHANNEL_CH1, &TIM3_IC_struct);
    LL_TIM_SetSlaveMode(TIM3, LL_TIM_SLAVEMODE_RESET);
    LL_TIM_SetTriggerInput(TIM3, LL_TIM_TS_TI1FP1);

    // Enable C/C interrupts
    LL_TIM_EnableIT_CC1(TIM3);
    // Active TIM1 interrupt in NVIC module
    isix::set_irq_priority(TIM3_IRQn, {0, 0});
    isix::request_irq(TIM3_IRQn);

    // Enable channel 1
    LL_TIM_CC_EnableChannel(TIM3, LL_TIM_CHANNEL_CH1);

    // Enable TIM3
    LL_TIM_EnableCounter(TIM3);

    // Initialize shadow registers
    LL_TIM_GenerateEvent_UPDATE(TIM3);
}
```

Następnym krokiem było uruchomienie modułu *C/C* w trybie *Input Capture* tak, aby obserwował on zmiany poziomu logicznego na kanale CH1. Przy wykryciu zbocza narastającego zapisuje on aktualną wartość rejestru CNT w rejestrze CCR1. Filtr sygnału wejściowego został wyłączony, aby uzyskać maksymalną dostępną rozdzielczość.

Aby pomiar okresu sygnału wejściowego nie był obciążony opóźnieniami programowymi**, skonfigurowano również moduł synchronizujący w trybie *Reset***. Sygnał *Trigger*, skonfigurowany jako wejście kanału CH1, powoduje dzięki temu nie tylko zapisanie wartości CNT do rejestru CCR1, ale również zresetowanie licznika. Dzięki temu, pomiar okresu realizowany jest **całkowicie sprzętowo**, a w rejestrze CCR1 znajduje się zawsze ostatnia zmierzona wartość, która może następnie posłużyć do wyznaczenia częstotliwości sygnału **poza procedurą pomiaru**.

Układ TIM1 skonfigurowany został tak, aby wyzwalać przerwania z okresem 1s. Procedura obsługi tego przerwania dokonuje odczytu wartości rejestru CCR1 i obliczenia na jej podstawie częstotliwości sygnału zgodnie ze wzorem

$$\frac{100MHz}{CCR1}$$

a następnie wypisuje rzeczywistą wartość częstotliwości i wartość obliczoną na podstawie pomiaru. Poniższa tabela przedstawia wyniki pomiarów trzech częstotliwości testowych oraz związanych z nimi wartości błędów. Wartości pomiarów zostały uśrednione z 20 kolejnych próbek.

Częstotliwość sygnału	Średnia wartość mierzona	Niepewność pomiarowa	Błąd bezwzględny	Błąd względny
10 kHz	10 002 Hz	+/- 1 Hz	2Hz	0.02%
100 kHz	100 200 Hz	+/- 100 Hz	200Hz	0.2%
1 MHz	1 020 408 Hz	+/ 10 101 Hz	20 408Hz	2.048%

Rysunek 3. Wyniki pomiarów częstotliwości fali prostokątnej przy pomocy układu czasowo-licznikowego TIM3

Jak widać, w zakresie niskich częstotliwości na poziomie kilku – kilkudziesięciu kHz pomiar obarczony jest niewielką niepewnością oraz stosunkowo niewielkim błędem. Jednak przy pomiarze częstotliwości stanowiącej 1/10 częstotliwości taktującej układ TIM3 dokładność pomiarów znacznie spada. Aby zwiększyć dokładność pomiarów w tym obszarze **należałoby zwiększyć częstotliwość taktowania układu czasowo-licznikowego** , co przy aktualnie używanym mikrokontrolerze nie jest możliwe. Czas obsługi programowej pomiaru został skrócony do zera, więc na tej płaszczyźnie nie da się już uzyskać dodatkowej precyzji.

Mimo wszystko należy uznać, że niepewności i błędy względne pomiarów są stosunkowo niewielkie i układ taki mógłby być z powodzeniem stosowany do przeprowadzania niekrytycznych pomiarów częstotliwości sygnałów prostokątnych tym bardziej, że pozwala on na pomiar częstotliwości w zakresie od ok. 1.526 kHz do 5MHz (przy czym oczywiście górna granica pomiarów obciążana byłaby już znacznie większą niepewnością pomiarową!). Aby zwiększyć zakres częstotliwości pomiarowych w dolnym rejonie można by skorzystać z układu TIM2/TIM5, który oferuje licznik 32-bitowy. Taki zabieg pozwoliłby mierzyć częstotliwości od poziomu ok. 0.023Hz.