KURS STM32

Wojciech Olech

CZĘŚĆ VII: TIMERY

Timery w mikrokontrolerach mają dwie podstawowe funkcje: stworzenie podstawy czasowej (czyli liczenie czasu), oraz generowanie sygnałów PWM.

Tworzenie podstawy czasowej jest użyteczne kiedy chcemy wykonywać pewną czynność z określonym interwałem, chcemy zmierzyć czas, lub chcemy taktować peryferia/*system operacyjny* (RTOS) z daną częstotliwością.

Sygnał PWM jest bardzo użyteczny do sterowania elementami z określoną mocą - najczęściej stosowany jest do sterowania silnikami elektrycznymi.

RODZAJE TIMERÓW

W mikrokontrolerach STM32 istnieje kilka rodzajów timerów:

- **Prosty timer** (***Basic timer***): zazwyczaj 16-bitowy, posiada jedynie funkcjonalność tworzenia podstawy czasu (co umożliwia taktowanie innych peryferiów lub triggerowanie przerwań z określoną częstotliwością). Nie posiada żadnych pinów I/O.
- **Timer ogólnego przeznaczenia** (***General purpose timer***): 16 lub 32-bitowy timer który posiada funkcjonalność prostego timera oraz posiada wyprowadzenia I/O które umożliwiają mu m. in. generowanie sygnału PWM, pomiar częstotliwości zewnętrznego sygnału, obsługę enkoderów i czujników halla.
- **Zaawansowany timer** (***Advanced timer***): Posiada funkcjonalność timerów ogólnego przeznaczenia ale dodatkowo zaawansowane feature'y, takie jak generowanie trzech komplementarnych sygnałów z wstawianiem czasu przestoju (*dead time*).
- **Timer wysokiej rozdzielczości** (***High resolution timer***): Specjalny timer stosowany w serii STM32F3 który służy stricte do sterowania silnikami.

- **Timer niskiej mocy (*Low-power timer*)**: Timer stworzony do zastosowań low-power, może pracować w prawie każdym trybie pracy/snu procesora, a nawet bez wewnętrznego źródła zegarowego (co pozwala mu pracować na przykład jako licznik impulsów). Tego rodzaju timery mają możliwość wybudzania mikrokontrolera ze snu.
- **Zegar czasu rzeczywistego (*Real-Time Clock*)**: Nie jest to stricte timer, ale znajduje się w tej samej kategorii. Taktowany specjalnym oscylatorem (optymalnie o częstotliwości 32.768kHz) pozwala na dokładny długotrwały pomiar czasu rzeczywistego, bez dryftu czasu który występuje w zegarach taktowanych innymi oscylatorami.

KONFIGURACJA TIMERA - LICZNIK CZASU

Żeby uruchomić timer, należy w Device Configuration Toolu wybrać interesujący nas timer, a następnie go skonfiugurować w interesującym nas trybie. Na start, skonfigurujemy timer tak żeby wywoływał co kilka sekund przerwanie które będzie migało diodą i wysyłało wiadomość po serialu.

Dla naszego zastosowania możemy wybrać dowolny timer. Wybierzmy TIM10 ponieważ jest to prosty timer, nie potrzebujemy niczego lepszego.

Posiada on kilka pól które musimy skonfigurować.

Dwa podstawowe pola jakie nas interesują to

- **Prescaler** - preskaler dzieli nam zegar którym taktujemy timer przez jego wartość. Pozwala on uzyskać dowolną częstotliwość taktowania zegara.
- **Counter Period** - Określa on maksymalną wartość licznika przy której zaczyna on liczyć od zera i przy której wywoła się *update event*.

Żeby poprawnie obliczyć częstotliwość wywołania *update eventu*, czyli przerwania licznika, musimy dodatkowo wiedzieć z jaką częstotliwością jest taktowany nasz timer. Zegary są taktowane za pomocą zegara APB do którego są podłączone, w widoku konfiguracji zegarów są to częstotliwości z pól "APB1 timer clocks" i "APB2 timer clocks". Domyślnie powinny być one taktowane częstotliwością rdzenia, czyli 84MHz dla STM32F401.

Żeby policzyć częstotliwość wywoływania *update eventu*, należy skorzystać z tego wzoru:

$$f_{\mathrm{UpdateEvent}}[\mathrm{Hz}] = \frac{\mathrm{TimerClock}}{(\mathrm{Prescaler}+1)(\mathrm{Period}+1)}$$

Załóżmy że chcemy migać diodą co sekundę. Nasz zegar timera wynosi 84 000 000Hz, musimy więc dobrać odpowiedni preskaler i okres zegara.

Jedna sekunda to 1Hz. Nasz preskaler i okres są wartościami 16-bitowymi, więc maksymalna wartość to 65535.
Żeby otrzymać jakąś *ładną* wartość zegara dla której łatwo będzie ustawić okres, możemy podzielić główny zegar przez 42000, co da nam 2000Hz. Chcąc otrzymać 1Hz, musimy 2000Hz podzielić przez 2000. Te dwie wartości (minus jeden) to nasz preskaler i okres. Finalnie dostajemy wzór

$$f_{\mathrm{UpdateEvent}}[\mathrm{Hz}] = \frac{84000000}{(41999+1)(1999+1)} = 1\mathrm{Hz}$$

OBSŁUGA TIMERA W KODZIE

Po ustawieniu timera i wygenerowaniu kodu możemy przejść do jego obsługi w kodzie.

Przerwanie timera powinno zostac wygenerowane w pliku `stm32f4xx_it.c` Obsługuje ono z użyciem HALa generyczne przerwanie timera, wywołując odpowiedni callback handler. W tym przypadku, interesujący nas callback to `void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim)` który musimy zdefiniować w naszym kodzie.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim) {
    // Sprawdź czy przerwanie przyszło od naszego timera
    if (htim == &htim10) {
      // Zmień stan diody (dla Nucleo makro GPIO będzie inne)
      HAL_GPIO_TogglePin(BOARD_LED_GPIO_Port, BOARD_LED_Pin);
    }
}
```

Żeby uruchomić timer, musimy wywołać funkcję `HAL_TIM_Base_Start_IT`, na przykład po inicjalizacji timera.

```
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM10_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim10);
  /* USER CODE END 2 */
```

Możemy teraz uruchomić wrzucić nasz program na mikrokontroler i zobaczyć efekt - dioda powinna zmieniać swój stan co sekundę.

ZMIANA OKRESU LICZNIKA

HAL ułatwia inicjalizację, uruchamianie i podstawową obsługę peryferiów ale jest też abstrakcją na wiele rzeczy, o których niekoniecznie możemy wiedzieć, patrząc pobieżnie na to co oferuje. Żeby dostać się do niektórych, bardziej zaawansowanych możliwości peryferiów, trzeba spojrzeć na niskopoziomowe funkcje HALa oraz rejestry.

Timery mają trzy ważne rejestry które nas interesują i które sterują działaniem timera:

- **PSC** - Prescaler Register, w tym rejestrze trzymana jest wartość preskalera jaką ustawiliśmy
- **ARR** - Auto-Reload Register, w tym rejestrze trzymana jest wartość okresu jaką ustawiliśmy
- **CNT** - Counter Register, w tym rejestrze jest trzymana aktualna wartość licznika timera

CNT i ARR są ze sobą ściśle powiązane, bo w momencie w którym CNT osiąga wartość zachowaną w ARR następuje wywołanie przerwania licznika (update event, PeriodElapsedCallback).

Zmieniając ARR i PSC możemy w takim razie zmieniać częstotliwość licznika (oraz wypełnienie sygnału PWM, ale o tym później). Należy jednak pamiętać o jednej ważnej rzeczy - **overflow**. Nie możemy zmniejszać wartości ARR w dowolnym momencie, bo może się okazać że zmniejszyliśmy ją w chwili kiedy CNT jest większy od nowego ARR, co spowoduje overflow i błędne działanie programu. Optymalnym rozwiązaniem jest zmiana ARR przy wywołaniu update eventu, czyli naszego callbacka PeriodElapsedCallback, bo wtedy mamy pewność że CNT == 0

Użyjemy do tego HALowego makra `__HAL_TIM_SET_AUTORELOAD`. **Nie należy zmieniać rejestrów ręcznie dopóki nie mamy 100% pewności że musimy to zrobić, makra są zawsze lepszym rozwiązaniem ponieważ zawsze poprawnie obsłużą wszystkie wymagane rejestry i struktury - `__HAL_TIM_SET_AUTORELOAD` poza zmianą ARR zmienia też pole Period w strukturze timera!**

Możemy na przykład użyć tego makra żeby przyspieszać i spowalniać szybkość migania diody. Przetestujmy ten kod:

```
/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    // Sprawdź czy przerwanie przyszło od naszego timera
    if (htim == &htim10) {
        HAL_GPIO_TogglePin(BOARD_LED_GPIO_Port, BOARD_LED_Pin);

        uint16_t arr_value = __HAL_TIM_GET_AUTORELOAD(htim);
        if (arr_value >= 200) {
            __HAL_TIM_SET_AUTORELOAD(htim, arr_value - 100);
        } else if (arr_value >= 100) {
            __HAL_TIM_SET_AUTORELOAD(htim, arr_value - 10);
        } else {
            __HAL_TIM_SET_AUTORELOAD(htim, 2000);
        }
    }
}
/* USER CODE END 4 */
```

DODATKOWE MATERIAŁY:

- STM32 cross-series timer overview: https://www.st.com/content/ccc/resource/technical/document/application_note/54/0f/67/eb/47/34/45/40/DM00