Laboratorium 2 MARM Układy czasowo licznikowe, liczniki czuwające

Lucjan Bryndza *Politechnika Warszawska*

11 grudnia 2019

Celem laboratorium jest zapoznanie się z układami czasowo-licznikowymi w mikrokontrolerach **STM32**. Generacja sygnału prostokątnego, sygnału **PWM**, oraz pomiarami częstotliwości oraz wypełnienia przebiegów impulsów z wykorzystaniem układów czasowo licznikowych. Uczestnik również nabędzie wiedzę dotyczącą układów liczników czuwających ang. *watchdog* oraz sposobem ich wykorzystania do poniesienia niezawodności działania systemów wbudowanych.

1. Zadania do wykonania na ćwiczeniach

Wszystkie ćwiczenia związane laboratorium realizować będziemy z wykorzystaniem płytki ewaluacyjnej **STM32F411E-DISCOVERY**. Laboratorium składać się będzie z czterech niezależnych ćwiczeń opisanych w podrozdziałach. Po wykonaniu ćwiczenia należy opracować sprawozdanie, którego sposób przygotowania opisano w rozdziale 3.

1.1. Generowanie przerwań w określonych przedziałach czasowych

Układ *Time Base* w licznikach podstawowych może służyć do generowania przerwań w określonych przedziałach czasowych, które mogą być wykorzystane do dowolnych celów. Napisz program który korzystając z układu czasowo licznikowego **T1** który na diodach: **LD4, LD3, LD5, LD6** wygeneruje efekt biegającego punktu. W tym celu:

- Zaprogramuj układ licznika T1 tak aby zgłaszał przerwanie co 200ms.
- Znajdź odpowiedni wektor przerwania i dodaj go w pliku źródłowym.
- Skonfiguruj potrzebne porty GPIO w kierunku wyjścia.
- Uzupełnij procedurę obsługi przerwania aby wygenerować efekt biegającego punktu.

1.2. Sprzętowe generowanie sygnału prostokątnego o wypełnieniu 50%

Napisz program który z wykorzystaniem układu licznikowego **T4** w sposób sprzętowy wygeneruje sygnał o następujących właściwościach:

- Kształt prostokątny o wypełnieniu 50%
- Częstotliwość sygnału 1Hz
- Sygnał o przesunięciu fazowym 0° na diodzie **LD4**
- Sygnał o przesunięciu fazowym 180° na diodzie **LD3**
- Sygnał o przesunięciu fazowym 270° na diodzie **LD5**

1.3. Generowanie sygnałów PWM

Napisz program który z wykorzystaniem układu czasowo licznikowego **T4** wygeneruje sygnał PWM o częstotliwości 200Hz i następujących parametrach:

- Wypełnienie *20%* na diodzie **LD4**
- Wypełnienie *40%* na diodzie **LD3**
- Wypełnienie *60%* na diodzie **LD5**
- Wypełnienie o wartości zmiennej od 0 do 100% na diodzie **LD6**.

Wypełnienie ostatniego przebiegu powinno się zmieniać z krokiem o 10% po każdorazowym wciśnięciu przycisku **USER** (*PA0*). Zbadaj oscyloskopem przebieg na diodzie LD5, sprawdzając tryb pracy PWM1 oraz PWM2 sprawdź jakich przebiegów nie są w stanie wygenerować poszczególne tryby.

1.4. Pomiar częstotliwości i wypełnienia impulsów

Uwaga! Ćwiczenie jest opcjonalne jeśli zostanie czasu na laboratorium.

Napisz program umożliwiający pomiar częstotliwości sygnału prostokątnego z wykorzystaniem układu **CC**. Program powinien co sekundę wypisywać aktualnie zmierzoną częstotliwość przebiegu prostokątnego.

Uwagi do ćwiczenia:

- Wykorzystaj układ licznika **TIM3** oraz układ capture **CC1** do pomiaru częstotliwości
- Wykorzystaj tryb pracy układu CC *Input capture*
- Jako wejście sygnału wykorzystaj port **PA6** funkcja alternatywna **AF02**. (*Układ CC1*)
- Podłącz generator sygnałowy do wybranego pinu i zbadaj działanie programu dla sygnałów o częstotliwości *10kHz*, *100kHz*, *1MHz*. Zapisz wyniki 10 pomiarów dla każdej częstotliwości

2. Materiały pomocnicze

2.1. Podłączenie zestawu STM32F411-DISCOVERY do portu szeregowego

Ponieważ zestaw **STM32f411E-DISCO** nie posiada wyprowadzonego portu szeregowego, umożliwiającego bezpośrednie dołączenie zestawu do komputera,

musimy skorzystać dodatkowej przejściówki RS232(TTL) na USB. Możemy wykorzystać dowolną przejściówkę realizującą to zadanie. W opisie zaprezentowano konwerter z układem FT232RL o oznaczeniu WSR-04502, ale w przypadku innego konwertera sposób postępowania będzie podobny. Układ należy połączyć z płytką ewaluacyjną za pomocą przewodów według konfiguracji z tabeli 1

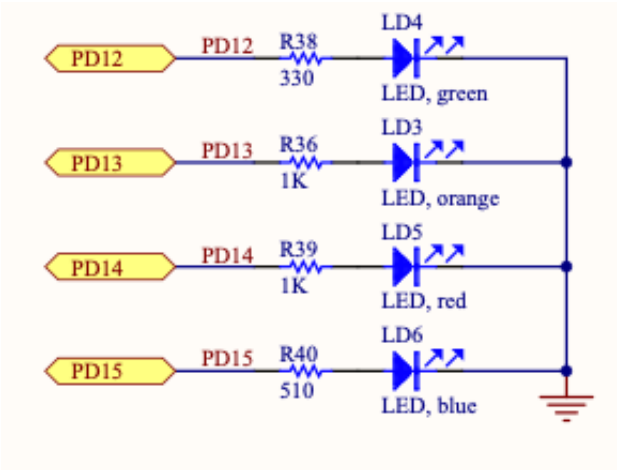
Konwerter USB	**STM32F411E-DISCO**
GND (niebieski)	GND
RXD (zielony)	PA2 (*USART2_TXD*)
TXD (czerwony)	PA3 (*USART2_RXD*)

Tabela 1: Połączenie konwertera RS232-USB z zestawem STM32F411E-DISCO

Zestaw STM32F469I-DISCO wyposażony jest w zintegrowany programator STLINK-V2.1, który za pomocą interfejsu USB udostępnia dodatkowy UART diagnostyczny, zatem nie ma potrzeby dołączania dodatkowej przejściówki. Szeregowy port diagnostyczny połączony wewnętrznie z interfejsem **USART3**: PB11(RX), PB10(TX)

2.2. Diody LED oraz przyciski w zestawie STM32F411E-DISCO

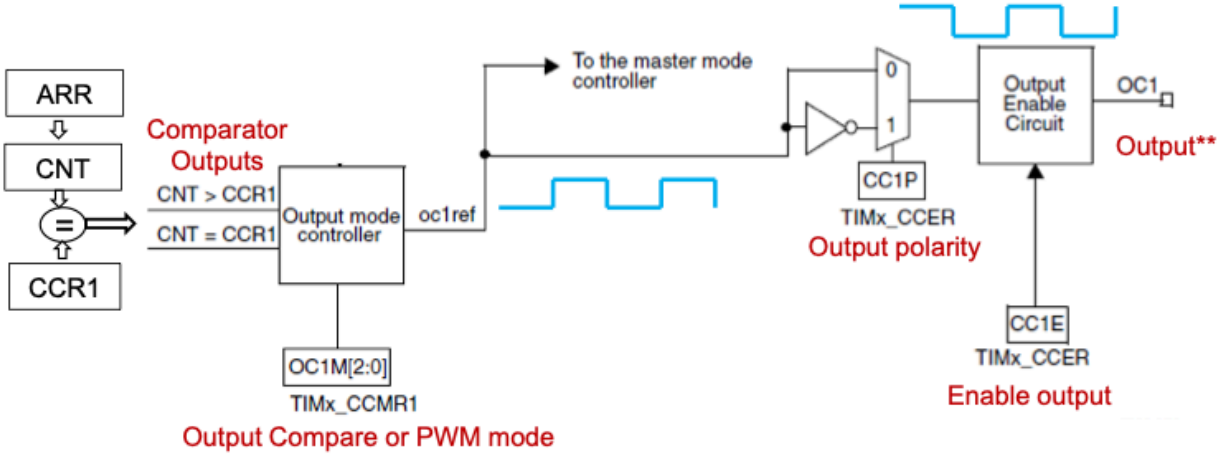
Zestaw ewaluacyjny posiada 4 diody LED podłączone do portu **PD** do pinów *12...15*. Porty te jednocześnie są dołączone jako funkcja alternatywna **AF02** do wyjść układu czasowo licznikowego **TIM4** kanały **CH1, CH2, CH3, CH4**.



2.3. Wykorzystanie układu licznika do generowania sygnałów prostokątnych i PWM

Układy czasowo licznikowe mogą być wykorzystywane do sprzętowego generowania sygnałów prostokątnych bez udziału jednostki centralnej. Każdy z kanałów

przechwytująco - porównujących możemy wykorzystać do generowania niezależnej częstotliwości na wyjściach układu. Przebieg prostokątny będzie dostępny na sprzętowych wyprowadzeniach układu licznika **TIMx_CHy**, gdzie *(x=numer kanału licznika, y=numer kanału porównującego)*. Aby z poszczególnych układów porównujących uzyskać sygnały o różnych częstotliwościach należy użyć trybu: *Output Compare* a wyjście skonfigurować w trybie *Toggle on match*. Poniższy rysunek przedstawia pracę licznika w trybie compare lub PWM.



2.4. Konfiguracja pobranie oraz uruchomienie projektu

Aby pobrać repozytorium możemy się posłużyć komendą, którą należy uruchomić w wierszu polecenia *GIT Bash*:

```
git clone --recursive -b pub/pw/lab2 https://boff.pl/cgit/
  ↪ public/isixsamples/
```

W kolejnym kroku należy zmienić katalog na *isixsamples* oraz pobrać konfigurację dla środowiska *VSCode*:

```
cd isixsamples
curl http://bryndza.boff.pl/downloads/prv/vscodecfg.zip --
  ↪ output vscodecfg.zip
```

Pobrany plik należy rozpakować bezpośrednio do katalogu *isixsamples*.

Konfiguracja projektu dla zestawu STM32F411 discovery odbywa się za pomocą polecenia:

```
python waf configure --debug --board=stm32f411e_disco
```

Kompilacje projektu możemy wykonać za pomocą polecenia:

```
python waf
```

Natomiast zaprogramowanie zestawu odbywa się za pomocą polecenia:

```
python waf program
```

Projekt możemy również konfigurować oraz uruchamiać bezpośrednio z *Visual Studio Code*. Najpierw należy w pliku *.vscode/task.json* zmienić argumenty dla polecenia *waf configure*. Następnie za pomocą polecenia **CTRL+P** otwieramy wiersz polecenia i wpisujemy *task waf configure* w kolejny kroku należy zbudować projekt za pomocą polecenia *task waf*, a w kolejnym kroku zaprogramować poleceniem *task waf program*.

Debugowanie programu następuje po wciśnięciu klawisza **F5** wcześniej jednak należy w pliku *.vscode/launch.json* ustawić odpowiednio ścieżkę do uruchamianego programu.

Opis skrótów klawiaturowych dla środowiska **VSCode** możemy znaleźć tutaj: [4].

2.5. Podstawowe API systemu ISIX do ćwiczenia

Podstawową funkcją diagnostyczną służącą do wypisywania danych na domyślny port diagnostyczny (terminal) jest funkcja:

```
#define dbprintf(fmt, ...) fnd::tiny_printf("%s:%d|" fmt "\r\n",
    ↪ _isix_dbglog_extract_basename(__FILE__),__LINE__,##
    ↪ __VA_ARGS__)
```

Funkcja przyjmuje identyczny zestaw argumentów jak standardowa funkcja **printf** pozbawiona jest jedynie formatowania i wyświetlania liczb zmiennoprzecinkowych.

Do utworzenia nowego wątku systemu operacyjnego możemy wykorzystać funkcję:

```
ostask_t isix::task_create(task_func_ptr_t task_func, void *
    ↪ func_param, unsigned long stack_depth, osprio_t priority,
    ↪ unsigned long flags=0);
```

Jako pierwszy argument funkcja przyjmuje funkcję, która będzie wykorzystana do realizacji wątku. Jako argument *func_param* możemy przekazać argument przekazany funkcji realizującej wątek. Parametr *stack_depth* określa wielkość stosu dla danego wątku. Do realizacji zadania powinien wystarczyć stos o wielkości 2kB. Jako parametr *priority* należy przekazać priorytet wątku, przy czym 0 oznacza priorytet najwyższy.

Obsługa przerwań w systemie ISIX dostępna jest po dołączeniu następujących plików nagłówkowych:

```
#include <isix/arch/irq_platform.h>
#include <isix/arch/irq.h>
```

Aby włączyć oraz wyłączyć wybraną linię przerwania IRQ od urządzenia w kontrolerze NVIC należy użyć nastepujących funkcji:

```
void isix::request_irq( int irqno );
void isix::free_irq( int irqno );
```

Natomiast priorytet przerwania możemy ustawić za pomocą następującej funkcji:

```
/** Set irqnumber to the requested priority level
 * @param[in] irqno IRQ input number
 * @param[in] new interrupt priority
 */
 //! Isix IRQ splited priority
typedef struct isix_irq_prio_s {
      uint8_t prio;
      uint8_t subp;
} isix_irq_prio_t;

void isix_set_irq_priority( int irqno, isix_irq_prio_t priority );
```

Jako pierwszy argument należy przekazać numer przerwania, natomiast jako drugi argument należy przekazać priorytet oraz podpriorytet przerwania. Domyślnie w systemie ISIX przerwania używają jednego bitu wywłaszczenia, a więc priorytet określa liczba z zakresu 0-1, natomiast podpriorytet liczba z zakresu 0-7.

2.6. Przydatne funkcje z biblioteki LL do obsługi liczników

Biblioteka LL dostępna jest po załączeniu pliku nagłówkowego *stm32_ll_tim.h*. Poniżej podano zestaw przydatnych funkcji podczas realizacji zadania.

Funkcja umożliwiająca konfigurująca tryb pracy wejścia układu przechwytującego.

```
void LL_TIM_IC_SetActiveInput(TIM_TypeDef *TIMx, uint32_t
  ↪ Channel, uint32_t ICActiveInput);
```

Konfiguracja filtru dla wejścia przechwytującego

```
void LL_TIM_IC_SetFilter(TIM_TypeDef *TIMx, uint32_t
  ↪ Channel, uint32_t ICFilter);
```

Konfiguracja preskalera układu przechwytującego

```
void LL_TIM_IC_SetPrescaler(TIM_TypeDef *TIMx, uint32_t
  ↪ Channel, uint32_t ICPrescaler);
```

Ustawienie polaryzacji zbocza układu przechwytującego

```
void LL_TIM_IC_SetPolarity(TIM_TypeDef *TIMx, uint32_t
  ↪ Channel, uint32_t ICPolarity);
```

Włączenie przerwania wybranego układu przechwytującego

```
void LL_TIM_EnableIT_CC1(TIM_TypeDef *TIMx);
void LL_TIM_EnableIT_CC2(TIM_TypeDef *TIMx);
void LL_TIM_EnableIT_CC3(TIM_TypeDef *TIMx);
void LL_TIM_EnableIT_CC4(TIM_TypeDef *TIMx);
```

Włączenie wybranego układu przechwytującego

```
void LL_TIM_CC_EnableChannel(TIM_TypeDef *TIMx, uint32_t
  ↪ Channels);
```

Włączenie wybranego licznika

```
void LL_TIM_EnableCounter(TIM_TypeDef *TIMx);
```

Ustawienie trybu pracy wybranego układu porównującego

```
void LL_TIM_OC_SetMode(TIM_TypeDef *TIMx, uint32_t Channel,
  ↪  uint32_t Mode);
```

Ustawienie wartości licznika porównującego

```
void LL_TIM_OC_SetCompareCH1(TIM_TypeDef *TIMx, uint32_t
  ↪ CompareValue);
```

Włączenie trybu preload na wybranym układzie porównującym

```
void LL_TIM_OC_EnablePreload(TIM_TypeDef *TIMx, uint32_t
  ↪ Channel);
```

Włączenie generowania zdarzenia update

```
void LL_TIM_GenerateEvent_UPDATE(TIM_TypeDef *TIMx);
```

Ustawienie wartości licznika *auto reload* (**ARR**)

```
void LL_TIM_SetAutoReload(TIM_TypeDef *TIMx, uint32_t
  ↪ AutoReload);
```

Sprawdzenie aktywności wybranej flagi układu CC

```
uint32_t LL_TIM_IsActiveFlag_CC1(TIM_TypeDef *TIMx);
```

Skasowanie aktywności wybranej flagi układu CC

```
 void LL_TIM_ClearFlag_CC1(TIM_TypeDef *TIMx);
```

W systemie ISIX wektory przerwań mają inne nazwy niż w przypadku przykładów z biblioteki **LL**. W przypadku układów czasowo-licznikowych nazwy wektorów przerwań są następujące:

```
ISR_VECTOR(tim1_brk_tim9_isr_vector);
ISR_VECTOR(tim1_up_tim10_isr_vector);
ISR_VECTOR(tim1_trg_com_tim11_isr_vector);
ISR_VECTOR(tim1_cc_isr_vector);
ISR_VECTOR(tim2_isr_vector);
```

```
        ISR_VECTOR(tim3_isr_vector);
        ISR_VECTOR(tim4_isr_vector);
        ISR_VECTOR(tim8_brk_tim12_isr_vector);
        ISR_VECTOR(tim8_up_tim13_isr_vector);
        ISR_VECTOR(tim8_trg_com_tim14_isr_vector);
        ISR_VECTOR(tim8_cc_isr_vector);
        ISR_VECTOR(tim5_isr_vector);
        ISR_VECTOR(tim6_dac_isr_vector);
        ISR_VECTOR(tim7_isr_vector);
```

Należy pamiętać, że w przypadku deklaracji wektorów w języku C++ należy użyć konstrukcji **extern "C"**, a funkcje wektorów przerwań powinny być zadeklarowane jako **void fun(void)**.

3. Instrukcja opracowania sprawozdania

- Umieścić oscylogramy sygnałów **PWM** wygenerowanych na diodach LED dla wypełnienia 0%, 30%, 100% w trybach **PWM1** oraz **PWM2**,
- Wyciągnąć wnioski odnośnie przebiegów wyjściowych w trybie **PWM**, w zależności od trybu pracy **PWM1** lub **PWM2**
- Opracować wyniki pomiarów częstotliwości generatora, wyznaczyć wartość średnią, niepewność pomiaru, błąd bezwzględny oraz względny dla wszystkich zmierzonych częstotliwości.
- Wyciągnąć wnioski na temat dokładności pomiaru częstotliwości w trybie capture.
- Wyznaczyć minimalne i maksymalne wartości częstotliwości jakie możemy zmierzyć w trybie capture.
- Co możemy zrobić aby zwiększyć dokładność pomiaru przebiegów o wysokiej częstotliwości? Proszę zaproponować rozwiązania.

Literatura

[1] *STM32F411E-DISCOVERY* kit user manual

[2] *STM32F411* reference manual

[3] *ISIX-RTOS* API examples

[4] *Visual Studio Code* keyboards shortcuts