Laboratorium 1 MARM System zegarów, porty GPIO, przerwania EXTI

Lucjan Bryndza *Politechnika Warszawska*

28 listopada 2019

Celem laboratorium jest zapoznanie się z systemem dystrubucji, oraz konfiguracji zegarów w mikrokontrolerach STM32. Zapoznanie się z obsługą portów GPIO mikrokontrolera z użyciem niskopoziomowych bibliotek *Low Level API* dostarczanych przez firmę ST, oraz bibliotekami niskopoziomowymi systemu ISIX. Zapoznanie się ze sposobem zgłaszania przerwań zewnętrznych z wykorzystaniem kontrolera **EXTI**.

1. Zadania do wykonania na ćwiczeniach

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

1.1. Konfiguracja sygnałów zegarowych oraz pętli PLL

W przykładzie **lab1_clk** w pliku *clocks_and_pll.cpp* znajduje się funkcja *uc_periph_setup()* której zadaniem jest konfiguracja sygnałów zegarowych oraz pętli *PLL* mikrokontrolera. W obecnej konfiguracji procesor taktowany jest z domyślnego generatora *HSI*, oraz został włączony generator *HSE*. Uzupełnij kod aby:

- Częstotliwość taktowania jednostki centralnej **HCLK** wynosiła 100MHz
- Częstotliwość taktowania magistrali **APB1** wynosiła 50MHz
- Częstotliwość taktowania magistrali **APB2** wynosiła 100MHz

Sprawdź działanie przykładu zarówno przed jak i po uzupełnieniu kodu. Zaobserwuj częstotliwość mrugania diody LED. Aby w wygodny sposób wyznaczyć potrzebne ustawienia pętli PLL można skorzystać z oprogramowania *STM32CubeMX*.

1.2. Zapoznanie się z obsługą portów GPIO i kontrolerem EXTI

- Do portu *PA0*, dołącz oscyloskop pracujący w trybie jednokrotnego wyzwalania oraz zapisz oscylogram powstały po wciśnięciu klawisza **USER**.
- Korzystając z funkcji GPIO systemu ISIX napisz program, który po każdym wciśnięciu klawisza **USER** *(PA0)*, będzie zwiększał stan zmiennej lokalnej o jeden, oraz wyświetlał stan najmłodszych 4 bitów tej zmiennej na diodach: **LD4, LD3, LD5, LD6**.
- Zmodyfikuj program tak aby zamiast funkcji systemu ISIX do obsługi portów GPIO wykorzystana została biblioteka LL.
- Zmodyfikuj program, tak aby zamiast cyklicznego odczytywania stanu klawisza **PA0**, wykorzystać przerwania oraz kontroler przerwań zewnętrznych **EXTI**.

Program powinien w sposób programowy uwzględniać eliminację drgań zestyków przełącznika.

1.3. Badanie maksymalnej częstotliwości pracy portów GPIO

- Napisz program zmieniający stan linii PB0 na przeciwny z maksymalną możliwą częstotliwością.
- Zbadaj za pomocą oscyloskopu jak będzie wyglądał przebieg wyjściowy dla wszystkich możliwych konfiguracji szybkości pracy portu GPIO w kierunku wyjścia. Wnioski umieść w sprawozdaniu.

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:

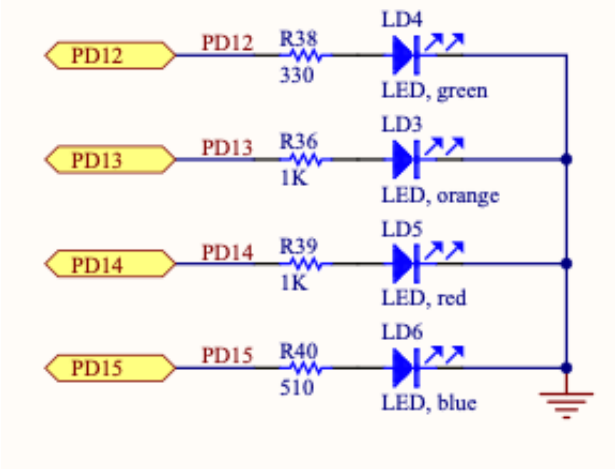
Konwerter USB	**STM32F411E-DISCO**
GND (niebieski)	GND
RXD (zielony)	PA2 (*USART2_TXD*)
TXD (czerwony)	PA3 (*USART2_RXD*)

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

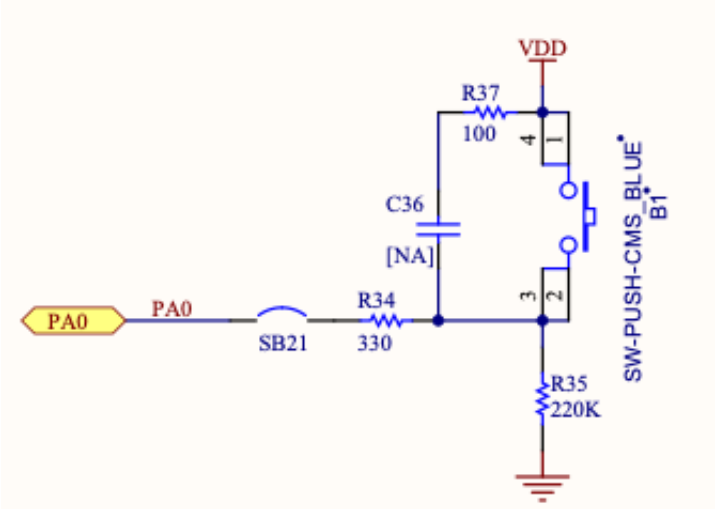
Zestaw STM32F469I-DISCO wyposażony jest w zintegrowany programator STLINK-V2.1, który za pomocą interfejsu USB udostępnia dodatkowy UART diagnostyczny, zatem nie ma potrzeby dołączania dodatkowej przejściówki. Szeregowy port diagnostyczny połączony wewnętrznie z interfejsem **USART3**: PB11(RX), PB10(TX)

2.2. Diody LED oraz przyciski w zestawie STM32F411E-DISCO

Zestaw ewaluacyjny posiada 4 diody LED podłączone do portu **PD** do pinów 12...15.



Jeśli chodzi o przyciski wejściowe do dyspozycji mamy tylko jeden przycisk o nazwie USER dołączony do portu PA0.



2.3. 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/lab1 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].

2.4. 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. W przypadku sortowania powinien wystarczyć stos o wielkości 1kB. 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

Do konfiguracji portów GPIO z wykorzystaniem API systemu ISIX możemy wykorzystać funkcje:

```
template <typename MODE,typename T>
        void setup( int pin, MODE&& tag)

template <typename MODE,typename T>
        void setup( std::initializer_list<T> pins, MODE&& tag)
```

Jako pierwszy należy przekazać numer portu GPIO lub grupę numerów portów. W systemie ISIX dla STM32 mamy odpowiednie definicje:

- **periph::gpio::num::PA0** - Pin0 z portu PA
- ...
- **periph::gpio::num::PA15** - Pin15 z portu PA
- **periph::gpio::num::PB0** - Pin0 z portu PB
- ...

Jako parametr *tag* należy przekazać parametr konfiguracyjny opisujący konfigurację portu szeregowego przy czym do dyspozycji mamy następujące klasy:

- **in** - Konfiguruje port w kierunku wejścia
- **out** - Konfiguruje port w kierunku wyjścia
- **analog** - Konfiguruje port w trybie analogowym
- **alt** - Konfiguruej port w trybie funkcji alternatywnej

Do niektórych klas możemy przekazać dodatkowe parametry konfiguracyjne:

```
enum class pulltype {
        floating,
        down,
        up
};
enum class outtype {
        pushpull,  //! Push pullmode
        opendrain_pu, //! Open drain pullup
        opendrain_pd, //! Open drain pulldown
};
enum class speed {
        low,
        medium,
        high
};
namespace mode {
        struct in {
                pulltype pu;
        };
        struct an {
                pulltype pu;
        };
        struct out {
```

```
                outtype out;
                speed spd;
        };
        struct alt {
                outtype out;
                int altno; //Alternate function identifier
                speed spd; // Port speed
        };
}
```

Więcej szczegółów znajduje się w prezentacji związanej z portami GPIO.

3. Instrukcja opracowania sprawozdania

- Umieścić oscylogramy prezentujące sygnał wyjściowy w zależności od ustawienia trybu szybkości pracy portu. Wyciągnąć wnioski płynące z oscylogramów.
- Umieścić oscylogram powstały po wciśnięciu klawisza **USER**. Wyciągnij wnioski jakie płyną z tego oscylogramu?
- Uzasadnić dlaczego konieczna jest eliminacja drgań zestyków w przypadku używania przełączników mechanicznych?
- Czy drgania zestyków można wyeliminować w sposób sprzętowy zaproponuj odpowiedni schemat, wraz ze stosownymi obliczeniami. Proszę również uzasadnić i opisać wybrany schemat.
- Narysuj diagramy aktywności *UML* zastosowanego algorytmu programowej eliminacji drgań zestyków w trybie obsługi z odpytywaniem, oraz w trybie z przerwaniem zewnętrznym **EXTI**.

Literatura

[1] *STM32F411E-DISCOVERY* kit user manual

[2] *STM32F469I-DISCOVERY* kit user manual

[3] *ISIX-RTOS* API examples

[4] *Visual Studio Code* keyboards shortcuts