Laboratorium 3 MARM
Obsługa portów szeregowych

Lucjan Bryndza *Politechnika Warszawska*

20 grudnia 2019

Celem laboratorium jest zapoznanie się z układami sprzętowych portów szeregowych USART w mikrokontrolerach **STM32**. Uczestnik laboratorium pozna sposób konfiguracji portu szeregowego, oraz używanie portu szeregowego w trybie odpytywania oraz z wykorzystaniem systemu przerwań.

1. Zadania do wykonania na ćwiczeniach

Wszystkie ćwiczenia związane laboratorium realizować będziemy z wykorzystaniem płytki ewaluacyjnej **STM32F411E-DISCOVERY**.Po wykonaniu ćwiczenia należy opracować sprawozdanie, którego sposób przygotowania opisano w rozdziale 3.

1.1. Obsługa portu szeregowego RS232 w trybie odpytywania

Napisz zestaw funkcji który:

- Skonfiguruje port szeregowy **USART1** w trybie asynchronicznej transmisji szeregowej z następującymi parametrami transmisji:
  - Prędkość: 115200 bitów / sekundę.
  - 8 bitów danych
  - 1 bit stopu
  - Brak kontroli parzystości
- Napisz funkcję umożliwiającą wysyłanie danych poprzez port szeregowy w trybie odpytywania.
- Napisz funkcję umożliwiającą odbieranie danych poprzez port szeregowy w trybie odpytywania
- Przetestuj prawidłowość działania powyższych funkcji korzystając z terminala szeregowego oraz debuggera

1.2. Interaktywny terminal tekstowy

Korzystając z wcześniej opracowanych funkcji napisz program interaktywnego terminala (*shell*), który za pomocą zestawu prostych poleceń tekstowych wydawanych w programie terminalowym Putty, umożliwi:

- Włączenie lub wyłączenie wybranej diody **LD3..LD6**
- Odczytanie stanu klawisza **USER**
- Odczytanie ilości wolnej pamięci na stercie systemu ISIX
- Bieżące użycie procesora w procentach

1.3. Obsługa portu szeregowego RS232 z wykorzystaniem przerwań

Ciągłe sprawdzanie rejestru statusów, w oczekiwaniu na nowe dane, lub oczekiwanie na wysłanie danych powoduje marnowanie cykli jednostki centralnej, która w tym samym czasie może zająć się innymi obliczeniami, albo może być uśpiona w celu oszczędzania energii. Aby zapobiec temu zjawisku lepszym sposobem jest wykorzystanie systemu przerwań.

- Zmodyfikuj program opracowany w poprzednich podpunktach, tak aby procedury obsługi portu szeregowego korzystały z systemu przerwań.
- Zweryfikuj poprawność działania interaktywnego terminala tekstowego z tak zmodyfikowanymi funkcjami wykorzystującymi przerwania.

1.4. Pomiary

Podłączyć oscyloskop do linii nadajnika portu szeregowego, oraz dokonać pomiaru kilku nadawanych znaków i zmierzyć:

- Błąd procentowy o ile prędkość transmisji odbiega od zadeklarowanej prędkości 115200 bitów na sekundę.
- Jaki jest średni czas odstępu pomiędzy znakami przy nadawaniu w trybie z odpytywaniem oraz w trybie z wykorzystaniem przerwania.

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*)

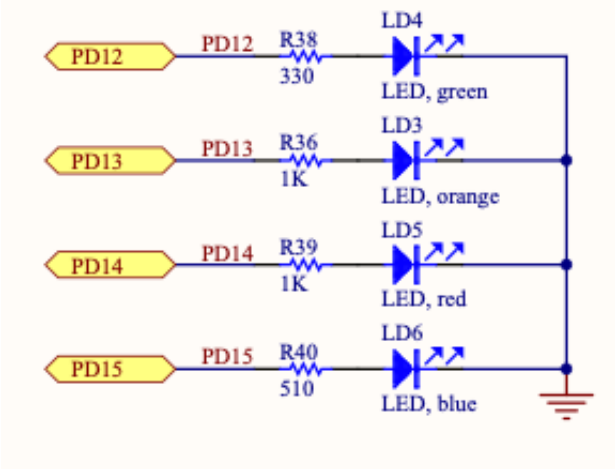
Tabela 1: Połączenie konwertera RS232-USB z zestawem STM32F411E-DISCO

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)

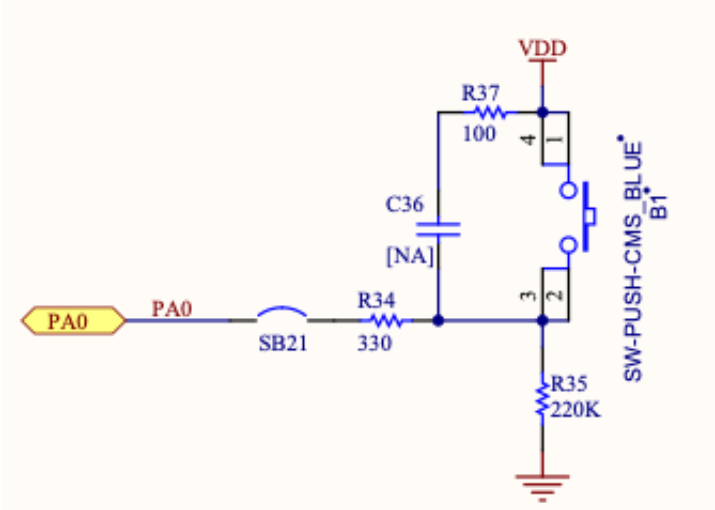
Port szeregowy **UART1** dostępny jest na pinach: **PB6** → TX, oraz **PB7** → RX przy wybraniu funkcji alternatywnej AF07. Na czas testu przejściówkę RS232 ↔ USB dołączyć do odpowiednich portów.

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/lab3 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. 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.

Aby uzyskać informację na temat aktualnie dostępnej ilości pamięci należy użyć funkcji:

```
      struct memory_stat {
            size_t free;
            size_t used;
            size_t fragments;
      };

      void heap_stats( memory_stat& meminfo );
```

Funkcja wypełnia strukturę informacjami na temat użytej pamięci, czyli: Ilość wolnej pamięci na stercie, ilość używanej pamięci, oraz informację o tym na jak bardzo sterta jest pofragmentowana.

Uzyskanie informacji o aktualnym wykorzystaniu czasu procesora możliwe jest za pomocą funkcji:

```
            int isix::cpuload();
```

Funkcja zwraca aktualne użycie czasu procesora w promilach. Funkcja domyślnie jest wyłączona, aby ją włączyć należy w pliku konfiguracyjnym *isix_config.h* dopisać następującą makrodefinicję włączającą to API w systemie ISIX:

```
      #define CONFIG_ISIX_CPU_USAGE_API 1
```

2.5. Przydatne funkcje z biblioteki LL do obsługi portu szeregowego

Biblioteka LL dostępna jest po załączeniu pliku nagłówkowego *stm32_ll_usart.h*. Poniżej podano zestaw przydatnych funkcji podczas realizacji zadania.

Konfiguracja trybu pracy portu szeregowego możliwa jest z wykorzystaniem funkcji:

```
ErrorStatus LL_USART_Init(USART_TypeDef *USARTx,
  ↪ LL_USART_InitTypeDef *USART_InitStruct);
```

Konfiguracja trybu pracy wybranego portu szeregowego odbywa się na podstawie parametrów konfiguracyjnych przekazanych za pomocą struktury konfiguracyjnej:

```
typedef struct
{
  uint32_t BaudRate; /*!< This field defines expected Usart
    ↪ communication baud rate. */

  uint32_t DataWidth; /*!< Specifies the number of data bits
    ↪ transmitted or received in a frame. This parameter can be
    ↪ a value of @ref USART_LL_EC_DATAWIDTH. */
  uint32_t StopBits; /*!< Specifies the number of stop bits
    ↪ transmitted. This parameter can be a value of @ref
    ↪ USART_LL_EC_STOPBITS. */
  uint32_t Parity; /*!< Specifies the parity mode. This parameter
    ↪ can be a value of @ref USART_LL_EC_PARITY.
  uint32_t TransferDirection; /*!< Specifies whether the Receive
    ↪ and/or Transmit mode is enabled or disabled. This
    ↪ parameter can be a value of @ref USART_LL_EC_DIRECTION. */
  uint32_t HardwareFlowControl; /*!< Specifies whether the
    ↪ hardware flow control mode is enabled or disabled. This
    ↪ parameter can be a value of @ref USART_LL_EC_HWCONTROL. */
  uint32_t OverSampling; /*!< Specifies whether USART oversampling
    ↪  mode is 16 or 8.bThis parameter can be a value of @ref
    ↪ USART_LL_EC_OVERSAMPLING. */

} LL_USART_InitTypeDef;
```

Włączenie lub wyłączenie portu szeregowego odbywa się za pomocą funkcji:

```
void LL_USART_Enable(USART_TypeDef *USARTx);
void LL_USART_Disable(USART_TypeDef *USARTx);
```

Funkcje przydatne do sprawdzania flag statusu portu szeregowego:

```
// Transmit register empty
uint32_t LL_USART_IsActiveFlag_TXE(USART_TypeDef *USARTx);
// Receive register not empty
uint32_t LL_USART_IsActiveFlag_RXNE(USART_TypeDef *USARTx);
```

Wysyłanie oraz odbieranie pojedynczego bajtu danych:

```
uint8_t LL_USART_ReceiveData8(USART_TypeDef *USARTx);
```

```
LL_USART_TransmitData8(USART_TypeDef *USARTx, uint8_t Value
  ↪ );
```

Włączenie lub wyłączenie przerwań od wybranych flag statusu:

```
// Enable TXE flag interrupt
void LL_USART_EnableIT_TXE(USART_TypeDef *USARTx);
// Enable RXNE flag interrupt
void LL_USART_EnableIT_RXNE(USART_TypeDef *USARTx);
// Disable TXE flag interrupt
void LL_USART_DisableIT_TXE(USART_TypeDef *USARTx);
// Disable RXNE flag interrupt
void LL_USART_DisableIT_RXNE(USART_TypeDef *USARTx);
```

W systemie ISIX wektory przerwań mają inne nazwy niż w przypadku przykładów z biblioteki **LL**. W przypadku portów szeregowych nazwy wektorów przerwań są następujące:

```
ISR_VECTOR(usart1_isr_vector);
ISR_VECTOR(usart2_isr_vector);
ISR_VECTOR(usart3_isr_vector);
ISR_VECTOR(usart4_isr_vector);
ISR_VECTOR(usart5_isr_vector);
ISR_VECTOR(usart6_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 zebrane dane pomiarowe oraz opracować wyniki pomiarów.
- Umieścić kilka oscylogramów podczas nadawania znaków poprzez port szeregowy.
- W sprawozdaniu umieścić opis na temat w jaki sposób zostało napisane oprogramowanie.
- Załączyć kilka zrzutów z terminala podczas pracy interaktywnego terminala.
- Opisać dlaczego transmisja szeregowa jest jednym z chętniej wykorzystywanych protokołów transmisji w najtańszych mikrokontrolerach oraz w celach diagnostycznych?

Literatura

[1] *STM32F411E-DISCOVERY* kit user manual

[2] *STM32F411* reference manual

[3] *ISIX-RTOS* API examples

[4] *Visual Studio Code* keyboards shortcuts