Mikrokontrolery ARM

Laboratorium 3

Krzysztof Pierczyk
24 kwietnia 2020

WSTĘP

Na kolejnych zajęciach laboratoryjnych zajęliśmy się obsługą portu szeregowego RS232 i związanym z nim układem peryferyjnym USART (*ang. Universal Synchronous Asynchronous Receiver-Transmitter*). Używana przez nas płytka ewaluacyjna *STM32F411E-DISCO* wyposażona jest w trzy takie układu – USART1, USART2 oraz USART 6. Układy 1 i 6 dołączone są do magistrali APB2, natomiast układ 2 do magistrali APB1. Do tej pory, pisząc aplikacji bazujące na systemie ISIX, układ USART2 był inicjalizowany poprzez makro systemowe wywołujące procedurę inicjalizacji modułu oraz rejestrującą układ jako standardowe wyjście dla informacji debugowych jak logi o powodzie zatrzymania systemu, czy komunikaty wypisywane przez makro `dbprintf(...)`.

Tym razem naszym zadaniem było samodzielne skonfigurowanie modułu komunikacyjnego oraz wykorzystanie go stworzeni prostej aplikacji **interaktywnego interfejsu szeregowego**, który umożliwiałby wydawanie mikrokontrolerowi poleceń z poziomu konsoli komputera stacjonarnego.

ZADANIE 1

Pierwsze z zadań wymagało od nas przeprowadzenia podstawowej konfiguracji jednego z dostępnych modułów USART oraz zaimplementowania dwóch funkcji, które udostępniałyby blokujące operacje zapisu i odczytu z portu szeregowego. Funkcje te posłużyć miały na późniejszym etapie do budowy interaktywnej konsoli.

Jako moduł roboczy wybrano USART1 jako, że jego wyprowadzenia połączone są z największą ilością pinów mikrokontrolera. Zgodnie z treścią zadania konfiguracja miała zapewniać komunikację w formacie:

- Prędkość 115200 bitów / sekundę
- 8 bitów danych
- 1 bit stopu
- Brak kontroli parzystości

Pełna konfiguracja układu została przeprowadzona z wykorzystaniem funkcji przedstawionej poniżej.

```
void USART1_config(){

    LL_APB2_GRP1_EnableClock(
        LL_APB2_GRP1_PERIPH_USART1
    );

    LL_USART_InitTypeDef usart_struct{
        .BaudRate = 115200,
        .DataWidth = LL_USART_DATAWIDTH_8B,
        .StopBits = LL_USART_STOPBITS_1,
        .Parity = LL_USART_PARITY_NONE,
        .TransferDirection = LL_USART_DIRECTION_TX_RX,
        .HardwareFlowControl = LL_USART_HWCONTROL_NONE,
        .OverSampling = LL_USART_OVERSAMPLING_16
    };

    LL_USART_Init(USART1, &usart_struct);

    LL_USART_Enable(USART1);

}
```

Rysunek 1. Funkcja konfigurująca moduł USART w trybie komunikacji z odpytywaniem

Do zapewnienia podstawowych operacji komunikacyjnych zaimplementowano cztery podstawowe funkcje występujące w bibliotece standardowej języka C. Są to kolejno:

- **`void putc(char c)`** – umożliwia przesłanie pojedynczego znaku poprzez port szeregowy. Wykorzystuje aktywne oczekiwanie na zapalenie się flagi TXE (*ang. TX Empty*)

- **`void puts(const char *str)`** – przesyła napis przekazany w przez argument na wyjście portu szeregowego. Koniec napisu definiowany jest przez znak NULL

- **`char getc(void)`** – pobiera z portu szeregowego pojedynczy znak. Jeżeli nie znajduje się on w buforze, funkcja oczekuje na jakiego przyjście

- **`int getline(char *str)`** – pobiera linię znaków z portu szeregowego. Linia kończy się znakiem nowej linii lub znakiem powrotu karetki

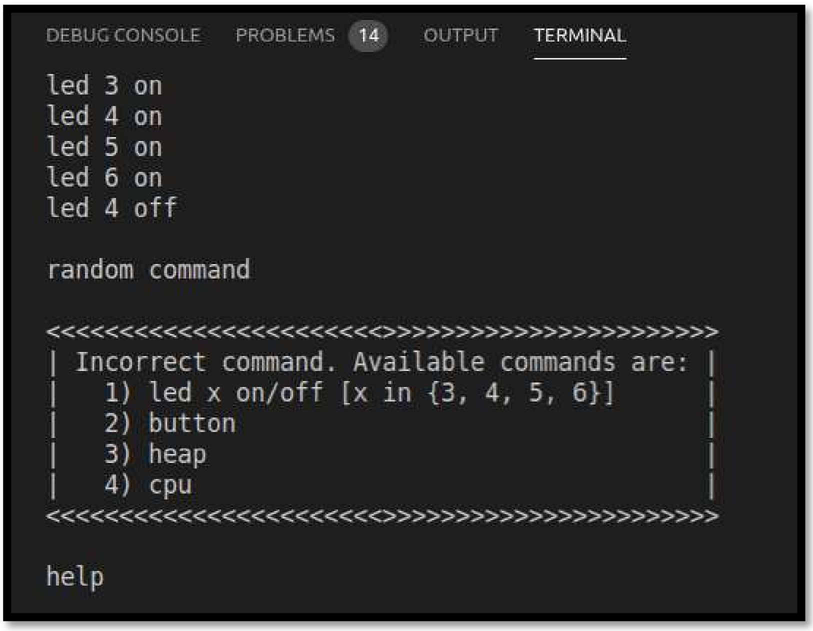
Poprawność działania funkcji została przeprowadzona poprzez implementację prostego programu typu *echo*, który wypisuje na wyjście taki sam ciąg znaków, jaki otrzymał na wejściu.

ZADANIE 2

Drugie zadanie to implementacja wspomnianej już interaktywnej konsoli. Po podłączeniu płytki do komputera poprzez konwerter USB↔RS232 oraz uruchomieniu dowolnego programu konsolowego (na etapie programowania użyto aplikacji **minicom**) możliwe jest wprowadzenie jednej z poniższych komend:

- *led X on / led X off* – włączenie lub wyłączenie jednej z dioda dostępnych na płytce. Wartość X należy do przedziału $\{3, 4, 5, 6\}$.
- *buton* – wyświetla stan przycisku User
- *heap* – wyświetla informacje o stercie systemu w tym ilość wolnego miejsca, ilość używanego miejsca oraz stopień fragmentacji
- *cpu* – wyświetla użycie jednostki obliczeniowej w (%) (z dokładnością do 0.1%)
- *help* – wyświetla listę dostępnych poleceń

Program ignoruje krótkie komendy, poniżej 10 znaków, które nie zostały zaimplementowane. Gdy błędna komenda jest dłuższa zostaje wyświetlony komunikat o braku wpisanego polecenia i lista dostępnych komend analogiczna do tej wyświetlanej przez polecenie *help*.



Rysunek 2. Polecenia dostępne w zaimplementowanej konsoli (I)

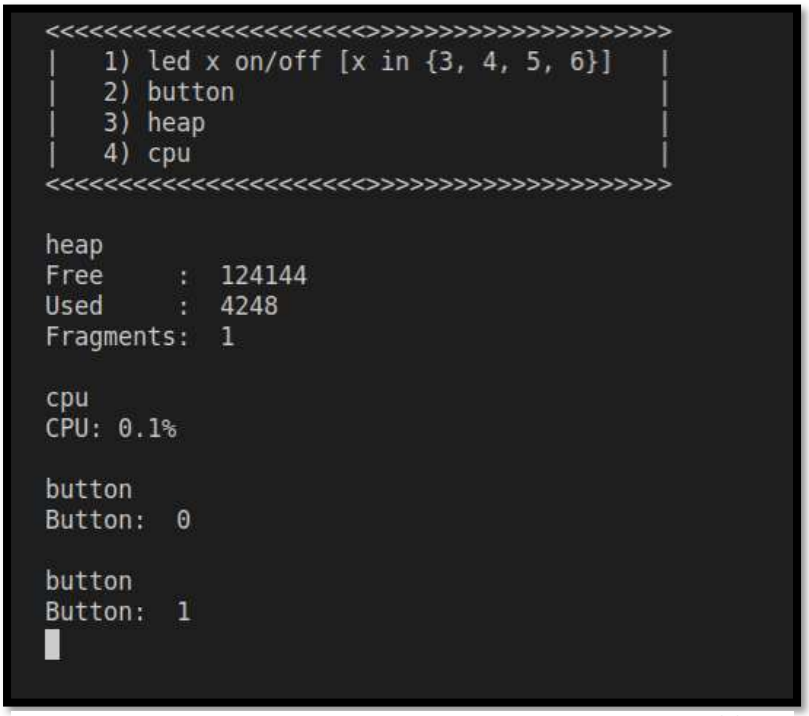
Dzięki dostępności podstawowych funkcji biblioteki standardowej języka C pokroju `sprintf()`, `strcmp()` czy `strcpy()` możliwe jest proste rozszerzenia napisanego programu o kolejne funkcjonalności. Można by w ten sposób nie tylko monitorować stan systemu działającego pod kontrolą mikrokontrolera ale także przeprowadzać akwizycję danych z czujników podłączonych do układu czy też zorganizować pracę projektowanego urządzenia tak, aby bardziej kosztowne obliczenia wykonywane były przez jednostkę klasy PC, a wyniki tych obliczeń trafiałyby do mikrokontrolera właśnie za pośrednictwem portu szeregowego.

ZADANIE 3

W zadaniu trzecim należało zmodyfikować poprzedni program tak, aby funkcje realizujące komunikację szeregową działały w oparciu o system przerwań jednostki USART. Komunikacja taka mogłaby zostać zrealizowana na wiele sposobów. W tym przypadku przepływ danych zdecydowano się zorganizować w następującej formie:

- **Dane wejściowe** – przerwania modułu RX układu USART zostało na stałe włączone w procedurze konfiguracyjnej. Przerwania występują niezależnie od działania reszty systemu i niezależnie od niego zbierają dane, które pojawiają się na wejściu portu. ISR (procedura obsługi przerwania, *ang. Interrupt Service Routine, ISR*) wykorzystuje dwa bufory mieszące 100 liczb typu `char` każda. W danej chwili ISR operuje tylko na jednym z buforów, natomiast do drugiego dostęp ma pozostała część systemu. W tym czasie wywołana zostać może funkcja `getline(char*)`, która zwróci dane obecne buforze gasząc przy tym odpowiednią flagę (co ma informować ISR o wykorzystaniu wcześniej wpisanych danych). Gdy procedura obsługi przerwań odbierze znak nowej linii (lub znak powrotu karetki) zapala ona ponownie tę flagę informując tym samym, że dostępne są nowe dane (nowa komenda) Wskaźniki, z których korzysta ISR oraz system zostają w tym momencie zamienione.

  Zaletą takie modelu jest to, że akwizycja danych jest zupełnie niezależna od pozostałych procesów w systemie. Program samodzielnie decyduje o tym kiedy należy zebrać dane z bufora. Wadą tego podejścia jest fakt, że w przypadku nieregularnego odbierania danych pewna ich część może zostać nadpisana przez nowsze dane i utracona.
- **Dane wyjściowe** – domyślnie przerwania TX układu USART są wyłączone. Gdy system zażąda wysłania napisu przez port szeregowy poprzez wywołanie funkcji



Rysunek 3. Rysunek 2. Polecenia dostępne w zaimplementowanej konsoli (II)

- `puts(const char *)` jeden z specjalnie przeznaczonych do tego buforów zostaje zapełniony danymi, a przerwania modułu TX są aktywowane. Funkcja wraca natychmiast po przekopiowaniu danych wejściowych do bufora co sprawia, że system nie jest zmuszony czekać na zakończenie procesu transmisji. Po zakończeniu przesyłu ISR wyłączy ponownie przerwania i uśpi moduł nadawczy do czasu następnego zapytania. Jeżeli zdarzy się tak, że polecenia przesłania danych wystąpią w krótki odstępie czasu i układ USART nie zdążył przesłać jeszcze pierwszego zestawu danych, napis wejściowy zostanie umieszczony w drugim z buforów, dzięki czemu funkcja ponownie wróci natychmiast po przekopiowaniu danych. Przy trzech żądaniach zapisu w krótkim czasie funkcja wstrzyma się do momentu ukończenia jednego z transferów po to, aby nie nadpisać poprzednich danych. Ponownie zaletą tego typu podejścia jest niezależność przeprowadzanej transmisji od pozostałych zadań systemu.

PODSUMOWANIE

Komunikacja szeregowa cieszy się od wielu lat dużą popularnością. Moduły umożliwiające jej obsługę znajdują się praktycznie w każdym współczesnym mikrokontrolerze. Powodem tej popularności jest fakt, że ten typ komunikacji wymaga

stosunkowo małej liczby przewodów (w porównaniu do transmisji równoległej) co znacznie obniża koszty instalacji go wykorzystujących.

W skrajnych warunkach (przykład protokołu 1-Wire) do komunikacji mogą być potrzebne jedynie dwa przewody, przy czym można w ten sposób utworzyć nie tylko połączenia typu point-to-point ale także pełnoprawną magistralę zawierającą zarówno urządzenia typu Master jak i Slave.Tam gdzie potrzebna jest większa szybkość transmisji wprowadza się dodatkową linię taktującą (protokoły SPI, I2C), co zwiększa stabilność komunikacji i umożliwia rozwinięcie większych prędkości.

Dodatkową zaletą protokołów komunikacji szeregowej jest też fakt, że fizyczna implementacja wymaganych przez nie elementów jest prosta i tania. Na przykład realizacja na podłożu krzemowym modułu umożliwiającego komunikację przy pomocy protokołu UART sprowadza się tak naprawdę do stworzenia kilku rejestrów przesuwnych, dlatego też moduły tego typu są tak chętnie umieszczane przez producentów w mikrokontrolerach.