Podstawy Systemów Mikroprocesorowych

Lab. 5. Magistrala szeregowa RS232. v.0.1

Dariusz Tefelski

2020-11-01

Spis treści

Laboratorium 5.

Magistrala szeregowa RS232.

Ćwiczenie ma na celu zapoznanie użytkownika z magistralą szeregową **RS232** oraz metodami jej obsługi w mikrokontrolerach **AVR**. Zadanie polega na napisaniu programów wykorzystujących wbudowane urządzenie **UART** do odbioru i transmisji komend sterujących i sprawdzenie ich działania wykorzystując komunikację poprzez **RS232** z komputerem PC.

Część 1:

Napisać program, który będzie odbierał pojedyncze znaki wysyłane za pomocą magistrali **RS232** z komputera PC. Jeśli odebranym znakiem będzie '**x**' należy odesłać ciąg znaków "**ok\n**".

UWAGA! Prędkość wysyłania znaków (baud rate) należy ustawić na niestandardowe 1 Mbit/s (**1000000 baud**). Zapewni to szybką i bezbłędną transmisję do konwertera USB/RS232. Niestety z "okrągłą" częstotliwością zegara nie można ustawić bez błędu standardowych prędkości transmisji takich jak np. 115200, 9600 itp.
1 Mbit/s jest maksymalną prędkością dla zegara **16 MHz** w przypadku **ATmega32** bez włączenia trybu **X2**. Wartość rejestru **UBRR** wyniesie wtedy **0**!
Pamiętaj! W programie komunikacyjnym na PC też trzeba będzie ustawić tą niestandardową prędkość transmisji.

UWAGA!: Porty szeregowe w systemie **GNU/Linux** dostępne są jako pliki specjalne. W przypadku standardowych portów wbudowanych w płytę główną komputera (lub kartę rozszerzeń) są to pliki **/dev/ttyS0**, **/dev/ttyS1**, **/dev/ttyS2** (pod systemem **MS Windows** są to odpowiednio **COM1**, **COM2**, **COM3**, ...). W przypadku portów szeregowych podłączonych poprzez USB (za pomocą konwerterów) są to pliki **/dev/ttyUSB0**, **/dev/ttyUSB1**, itd. ewentualnie pliki **/dev/ttyACM0**, **/dev/ttyACM1** ... W systemie **MS Windows** tworzone są wirtualne urządzenia **COM** o kolejnych, wyższych numerach. W systemie **Apple OS X** dostępny port wirtualny będzie w postaci pliku np. **/dev/cu/usbserial-AC00KFWT**

W przypadku podłączenia płytki EvB 5.1 do komputera PC poprzez kabel USB A-B, wirtualny port szeregowy dostępny będzie standardowo jako plik **/dev/ttyUSB0** dla systemu **GNU/Linux** (pod warunkiem, że nie mamy innych podobnych urządzeń podłączonych wcześniej). Pod systemem **MS Windows** pojawi się nowy port **COM**. W systemie **Apple OS X** dostępny port wirtualny będzie w postaci pliku np. **/dev/cu/usbserial-AC00KFWT** Na płytce EvB 5.1 wyprowadzenia portu szeregowego znajdują się na złączu podpisanym **FT232** w lewym górnym rogu, obok złącza **USB B**. Normalna praktyka podłączenia to nadajnik – odbiornik (**TXD–RXD**). Należy zatem podłączyć wyprowadzenie **TXD** ze złącza **FT232** do wyprowadzenia **RXD** w mikrokontrolerze, oraz **RXD** ze złącza **FT232** do wyprowadzenia **TXD** w mikrokontrolerze. Odpowiednie piny należy znaleźć w nocie katalogowej mikrokontrolera **ATmega32**.

Jako oprogramowanie komunikacyjne można wykorzystać odpowiednio dla:

- GNU/Linux:
  - **cutecom** (wygodny, dobry do testowania portów, graficzny)
  - **hterm** (wygodny, dobry do testowania portów, graficzny)
  - **moserial** (wygodny, dobry do testowania portów, graficzny, ale brak dowolnego ustawienia prędkości portu szeregowego!)
  - **picocom** (terminalowy, tekstowy, poprzez komendę **picocom /dev/ttyUSB0 –b 1000000**, gdzie parametr liczbowy to prędkość transmisji, wyjście przez **Ctrl+a Ctrl+x**)
  - **minicom** (terminalowy, tekstowy, w terminalu uruchomić **minicom -D /dev/ttyUSB0 -b 1000000**, ale należy w ustawieniach wyłączyć także sprzętową kontrolę przepływu (flow control). Można skonfigurować poprzez **minicom -s**)
  - **screen** (terminalowy, tekstowy, poprzez komendę: **screen /dev/ttyUSB0 1000000** , gdzie parametr liczbowy, to prędkość transmisji, wyjście przez **Ctrl+a k**)
  - **putty** (terminalowy, graficzny)
- MS Windows:
  - **bray terminal** (wygodny, dobry do testowania portów, graficzny)
  - **realterm** (wygodny, dobry do testowania portów, graficzny)
  - **hterm** (wygodny, dobry do testowania portów, graficzny)
  - **yat** (wygodny, dobry do testowania portów, graficzny)
  - **termie** (wygodny, dobry do testowania portów, graficzny)
  - **termite** (wygodny, dobry do testowania portów, graficzny)
  - **putty** (terminalowy, graficzny)

- Apple OS X:
  - picocom (terminalowy, tekstowy, instalacja przez **brew install picocom** (instalacja **brew** podana na stronie https://brew.sh), wywołanie poprzez komendę **picocom –b 1000000 /dev/cu/usbserial-AC00KFWT**, gdzie parametr liczbowy to prędkość transmisji, wyjście przez **control+a control+x**, włączenie lokalnego echa przez **control+a control+c**.)

Część 2:

Rozbudować program o obsługę komend wielo-znakowych ’‘**on**” - włącz i ’‘**off**” - wyłącz i za pomocą tych komend sterować włączaniem i wyłączaniem diody na płycie uruchomieniowej. W tym celu należy przygotować bufor odbieranych znaków. Dodatkowo jednocześnie z obsługą **RS232** powinna zostać wykorzystana obsługa przycisku – podobnie jak w poprzednim ćwiczeniu, odczyt stanu naciśnięcia przycisku w przerwaniu licznika co **100 ms**. Przycisk **S1** należy podłączyć do linii portu **PC1**, a diodę **LEDS 1** do linii portu **PC0**.

Zapalanie diody powinno odbywać się w pętli głównej w funkcji **main()** na podstawie stanu zmiennej globalnej ‘**led_status**’.

W rozwiązaniu zadania należy uwzględnić obsługę sytuacji błędnych typu niepełna przesłana instrukcja i kolejne znaki następnej. Można to zrealizować za pomocą czyszczenia bufora nadesłanych znaków po określonym czasie (w celu sprawdzenia wstępnie można posłużyć się dłuższym czasem np. **1 s**, a następnie zejść do bezpiecznego minimum). Dekodowanie komendy może być także zrealizowane poprzez odebranie określonej ilości znaków albo na podstawie wystąpienia znaku kończącego komendę np. **‘\n’** lub innego.

Część 3:

Przygotować kalkulator **RPN** (Reverse Polish Notation) w ramach interfejsu **RS232**. Użytkownik z podłączonego terminala wpisuje liczbę i zatwierdza znakiem nowej linii (**‘\n’**). Mikrokontroler odczytuje liczbę i zapisuje do zmiennej **y** (przed tym zmienną **y** przepisuje do zmiennej **x**). Użytkownik wpisuje drugą liczbę i zatwierdza znakiem nowej linii (**‘\n’**). Mikrokontroler przepisuje zmienną **y** do zmiennej **x** i zapisuje odczytaną z **RS232** liczbę do zmiennej **y**. Użytkownik wpisuje działanie (**‘+’**, **‘-’**, **‘*’**, **‘/’**) i zatwierdza znakiem nowej linii (**‘\n’**). Mikrokontroler oblicza wartość: **x y** i wynik zapisuje do zmiennej **y** oraz przesyła w postaci tekstowej poprzez **RS232**.

Należy obsłużyć sytuacje wyjątkowe (np. przez wysłanie komunikatu o błędzie) typu **dzielenie przez 0**, **niewłaściwa komenda** itp.