KIV/OS - cvičení č. 3

Martin Úbl

14. října 2022

1 Obsah cvičení

- pomocné (auxiliary) periferie
- miniUART - představení
- Aux driver
- miniUART driver

2 Auxiliary periferie

Mezi pomocné periferie mikrokontroléru BCM2835 se řadí:

- miniUART
- SPI1
- SPI2

SPI1 a SPI2 jsou dva kanály pro komunikaci pomocí protokolu SPI. Tento protokol a tyto kanály pro teď vynecháme, jelikož je nepotřebujeme a budeme se soustředit na obecnou funkci auxiliary periferií a miniUART rozhraní.

Auxiliary subsystém řídí své jednotlivé periferie – dovede je zapínat a vypínat, povolovat či zakazovat jim vyvolávat přerušení, číst, zda k nějakému přerušení došlo, a tak dále. Ovladač pro tento subsystém bude veskrze velmi jednoduchý. Momentálně přerušení nepotřebujeme, a tak budeme jen zapínat a vypínat periferie.

3 miniUART

Rozhraní miniUART je „redukovanou“ verzí rozhraní UART (Universal Asynchronnous Receiver Transmitter). To je pravděpodobně nejjednodušší rozhraní pro komunikaci mezi dvěma uzly. Jedná se o tzv. arytmický přenos, tedy takový přenos, který není přesně synchronizován po celou dobu existence propojení, ale

k synchronizaci dochází bezprostředně před vysíláním každého jednoho znaku. Oba konce komunikace mají předem nastavené parametry přenosu:

- délka znaku (standardně 7 či 8 bitů)
- počet start a stop bitů (standardně 1 start a 1 stop bit)
- modulační (přenosová) rychlost
- zabezpečení dat (standardně není žádné, ale může být paritní bit)
- řízení toku dat
- a další...

V té úplně nejjednodušší (a stále standardní) podobě UART vyžaduje pouze dva vodiče (resp. 3, když počítáme i propojení zemí) - takzvaný TX (vysílací) a RX (přijímací) vodič. Tyto vodiče jsou logicky propojeny křížem s protějším účastníkem komunikace.

V podobě, ve které je obsažen miniUART v RPi Zero lze nastavovat pouze vybrané parametry:

- délku znaku 7 nebo 8 bitů
- modulační rychlost

Vše ostatní je zafixováno:

- bez paritního bitu
- 1 start a 1 stop bit
- bez řízení toku dat

Modulační rychlost je vždy odvozena od taktovací frekvence jádra procesoru. Dle referenční příručky BCM2835 je vypočtena takto:

$$V_{mod} = \frac{f}{8 \cdot (B + 1)}, \tag{1}$$

kde V_{mod} je modulační rychlost v Baudech, f je taktovací frekvence jádra v Hertzech a B je celočíselný obsah registru pro nastavení děličky. Vhod nám přijde ale spíše vzorec s vyjádřeným B:

$$B = \frac{f}{8 \cdot V_{mod}} - 1 \tag{2}$$

Principielně je často přijímací mechanismus řešen jako posuvný registr kombinovaný se znakovou FIFO. V tomto případě při příjmu přijímač plní posuvný registr přijímaným znakem, ten vloží do FIFO a signalizuje vnější kód, a to buď přerušením nebo pouhým nastavením příznaku. Vnější kód má k dispozici vždy přední prvek z fronty prostřednictvím vyhrazeného registru.

Vysílací mechanismus funguje velmi podobně, ovšem v případě miniUART a RPi Zero zvládne pouze jeden znak ve výstupní frontě. Pro vysílání dalšího znaku je třeba počkat na přerušení nebo na odebrání příznaku plné výstupní fronty.

4 Aux driver

Jak bylo avizováno, aux driver bude velmi jednoduchý. Nejprve doplníme konstanty s offsety pro memory-mapped IO do `peripherals.h`:

```
constexpr unsigned long AUX_Base = Peripheral_Base + 0x00215000UL
    ;
```

Následně doplníme i výčet registrů:

```
enum class AUX_Reg
{
  // registr pro priznaky cekajicich preruseni
  IRQ = 0,
  // registr pro povolovani AUX periferie
  ENABLES = 1,

  // mini UART registry
  MU_IO = 16,
  MU_IER = 17,
  MU_IIR = 18,
  MU_LCR = 19,
  MU_MCR = 20,
  MU_LSR = 21,
  MU_MSR = 22,
  MU_SCRATCH = 23,
  MU_CNTL = 24,
  MU_STAT = 25,
  MU_BAUD = 26,

  // ... pro ted ignorujme SPI1 a 2
};
```

Poté vytvoříme soubory `bcm_aux.cpp` a `bcm_aux.h` v příslušných adresářích ovladačů.

V těch budou v základu pouze dvě funkce – povolení a zakázání příslušné periferie. Periferie (a odpovídající bity v registrech) jsou číslovány dle klíče, který lze najít v dokumentaci BCM2835. Vytvořme proto výčtový typ:

```
enum class AUX_Peripherals
{
  MiniUART    = 0,
  SPI1        = 1,
  SPI2        = 2,
};
```

Pak definujme funkci pro zapínání periferie:

```
void AUX_Enable(hal::AUX_Peripherals aux_peripheral)
{
  volatile unsigned long* aux =
    reinterpret_cast<unsigned long*>(AUX_Base);

  int reg = static_cast<int>(AUX_Reg::ENABLES);

  aux[reg] = aux[reg] | (1 << static_cast<int>(aux_peripheral));
}
```

Analogicky můžeme definovat i funkci pro vypnutí odmaskováním příslušného bitu (ponecháno jako cvičení čtenáři).

5 miniUART driver

V této sekci (ne)napíšeme minimalistický driver pro miniUART tak, abychom byli schopni přes něj odeslat data. Bude tak sloužit jako diagnostický výstup, až budeme ladit nějaké další vlastnosti systému.

miniUART jako takový má možností spousty – pro jejich výčet si můžete opět prohlédnout příslušné sekce BCM2835 referenční příručky.

Do té se ale budete stejně muset podívat – psaní driveru pro miniUART bude obsahem následujícího úkolu za body. Co je třeba pro rozběhání miniUARTu udělat?

- nastavit GPIO piny pro UART na výstupní/vstupní
- povolit aux periferii UARTu
- nastavit délku znaku a modulační rychlost
- vynulovat všechny možné příznaky přerušení, atd.

Pak by mělo být možné odeslat znak přes miniUART do hostitelského počítače. Rozhraní miniUART definuje registr `MU_LSR`, který na konkrétních bitech signalizuje, zda je odchozí FIFO schopen přijmout další znaky. Na ten budeme aktivně čekat, dokud nebude FIFO opět volný. Pak stačí zapsat odchozí znak do registru `MU_IO` a měl by být odeslán.

V hostitelském počítači pak můžeme vyzvednout znak nějakým klientem nad sériovým rozhraní (Windows – např. PuTTy, Linux/maxOS – klidně `cat`, `screen`, PuTTy, ...). Stačí jen portu nastavit příslušné vlastnosti (Windows – klientem, v nastavení nebo `mode`, Linux/macOS – klientem nebo `stty`, ...).

Pokud používáte nějaký ze standardních USB-TTL převodníků, port se objeví v systému jako klasický sériový port. Na Windows se pak port maskuje jako port s prefixem COM (např. `COM4`), na Linux/macOS jako zařízení konzole/teletype se suffixem dle řadiče (např. `/dev/ttyS0`, `/dev/ttyUSB1`, ...).

6 Úkol za body

Jak bylo psáno výše - Vaším úkolem bude implementovat alespoň základ ovladače pro miniUART tak, abyste byli schopni přenést řetězec a číslo do hostitelského počítače, a také přijmout znak z něj.

Bude tedy pak možné zavolat něco jako:

```
UART_Write("Cislo je: ");

int a = 15;
UART_Write(a);
UART_Write("\r\n");
UART_Write("Zadej znak: ");

char c;
UART_Read(&c);

c += 1;
UART_Write(c);
```

Kód otestujte v qemu. Driver implementujte tak, abyste ho ideálně mohli použít v semestrální práci – tak či tak se objevuje ve všech zadáních.

Pozn.: tento úkol se dá splnit jen vykucháním kódu ze cv. 3 dostupného na home a SREC bootloaderu, který jsme kompilovali na 1. cvičení.