



**KAE/MINA**

# **Cvičení 2016/17**

Ing. Petr Weissar, Ph.D.

Ing. Petr Krist, Ph.D.

Ing. Kamil Kosturik, Ph.D.

# Program cvičení

- 1. Úvod, rozdělení kitů, prostředí Keil, nahrání programu**
2. Debug, bitové operace, GPIO
3. UART, připojení k PC, využití stdio knihovny
4. Časování, hodiny, SysTick
5. Časovač, přerušení, NVIC
6. SPI a připojení LCD
7. A/D převodník, připojení přes IIC
8. DMA
9. RTOS
10. Samostatná práce
11. Dtto
12. Dtto
13. Dokončení a odevzdání samostatné práce

# Organizace výuky, práce v laboratoři

- V laboratoři pracujeme samostatně/ve dvojicích
  - Společný sdílený síťový adresář – disk T:
  - Domácí adresář studentů H:
    - Přilogování pomocí Orion-přístupu
  - Další síťové disky (R/O) - Z: (SW, podklady), X: (bat)
- HW společný – zapůjčen studentům
  - Předpokládá se část práce (příprava) doma
  - Uzavřena „smlouva o výpůjčce“
- Možno vlastní projekt založený na Cortex-Mx procesoru
  - Nutno na počátku semestru schválit vyučujícím
- Bezpečnost práce
  - Studenti musí mít 50-tku v laboratořích
- Bezpečnost elektroniky a zařízení
  - Kromě dodaných kitů veškeré další připojování HW **až po souhlasu cvičícího**



# Podmínky zápočtu

- **Funkční aplikace** - semestrální práce
- Znalost probírané problematiky na cvičeních
- Splnění kroky domácí přípravy
  - Na konci slajdů „domácí příprava“ seznam vyzkoušeného
  - Na počátku slajdů „cvičení“ seznam, co se má umět
  - Nesplnění jednotlivých etap přípravy ovlivní známku ze cvičení, se kterou se pak „jde“ ke zkoušce
- Vracený HW v pořádku
- V případě využívání zapůjčených kitů nutná účast na cvičeních
- U vlastního HW vhodné konzultace o samostatné práci
  - Doporučeno po 3-4 týdnech

# Nezbytné kroky domácí přípravy

- I. Instalace Keil MDK, příp. Atollic IDE, ST-Link, analýza demo-programu, použití debuggeru
- II. Bitové operace, maskování, GPIO
- III. Funkce z knihovny stdio – procvičení
- IV. Časování, nastavení RCC bloku, rozdělení hodinových signálů perifériím
- V. Procvičení práce s LCD, doplnění functionality
- VI. Zpracování dat ze senzorů (A/D, externí)
- VII. DMA
- VIII. Instalace a procvičení RTOS
- IX. Samostatná práce
- X. ...
- XI. ...
- XII. ...

# Administrativa zapůjčení kitů

- Připraven formulář o "Zápůjčce"
  - Jedna kopie zůstává, druhou má student
- Kit obsahuje v krabičce
  - Nucleo STM32F103RB
  - MBED Application shield
  - Mini USB kabel
- Vracení HW na konci semestru, ideálně ve 13. týdnu po předvedení funkční semestrální práce

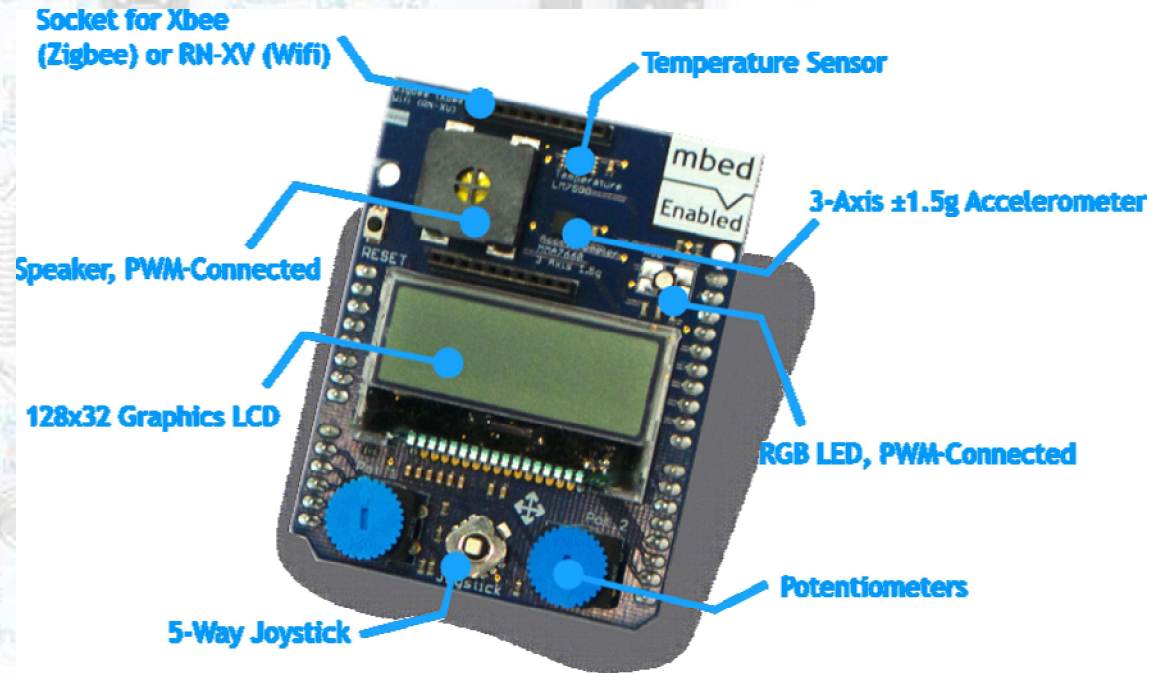
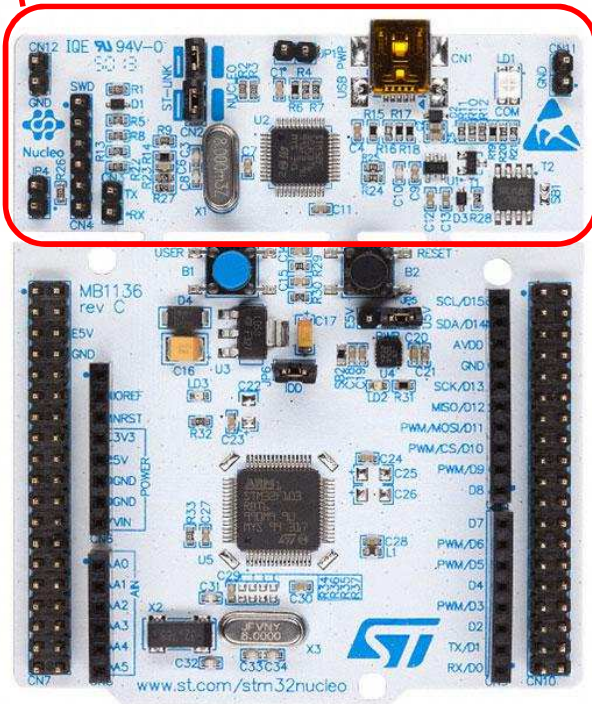


# Zdroje informací

- Dostupné v elektronické podobě – typicky PDF:
  - Internet – [www.st.com](http://www.st.com), [mbed.org](http://mbed.org), ...
  - Vybrané na CourseWare, zde na Z:\podklady\MINA\...
- Procesor:
  - **Reference manual** - STM32F101\_2\_3\_5\_107xx advanced ARM-based 32-bit MCUs (CD00171190)
  - **Datasheet** - STM32F103xB CD00161566.pdf
  - **Programming manual** - STM32F10x Programming manual (CD00228163)
- Nucleo kit
  - <https://developer.mbed.org/platforms/ST-Nucleo-F103RB/>
  - User manual - Nucleo UserManual (DM00105823)
  - Schematic - Nucleo Schematic (MB1136)
- MBED Shield
  - <https://developer.mbed.org/components/mbed-Application-Shield/>
  - Schematic - ApplicationShield.pdf
  - Komponenty
    - Akcelerometr - MMA7660FC.pdf
    - Temperature - LM75B.pdf
    - LCD - NHD-C12832A1Z-FSW-FBW-3V3.pdf

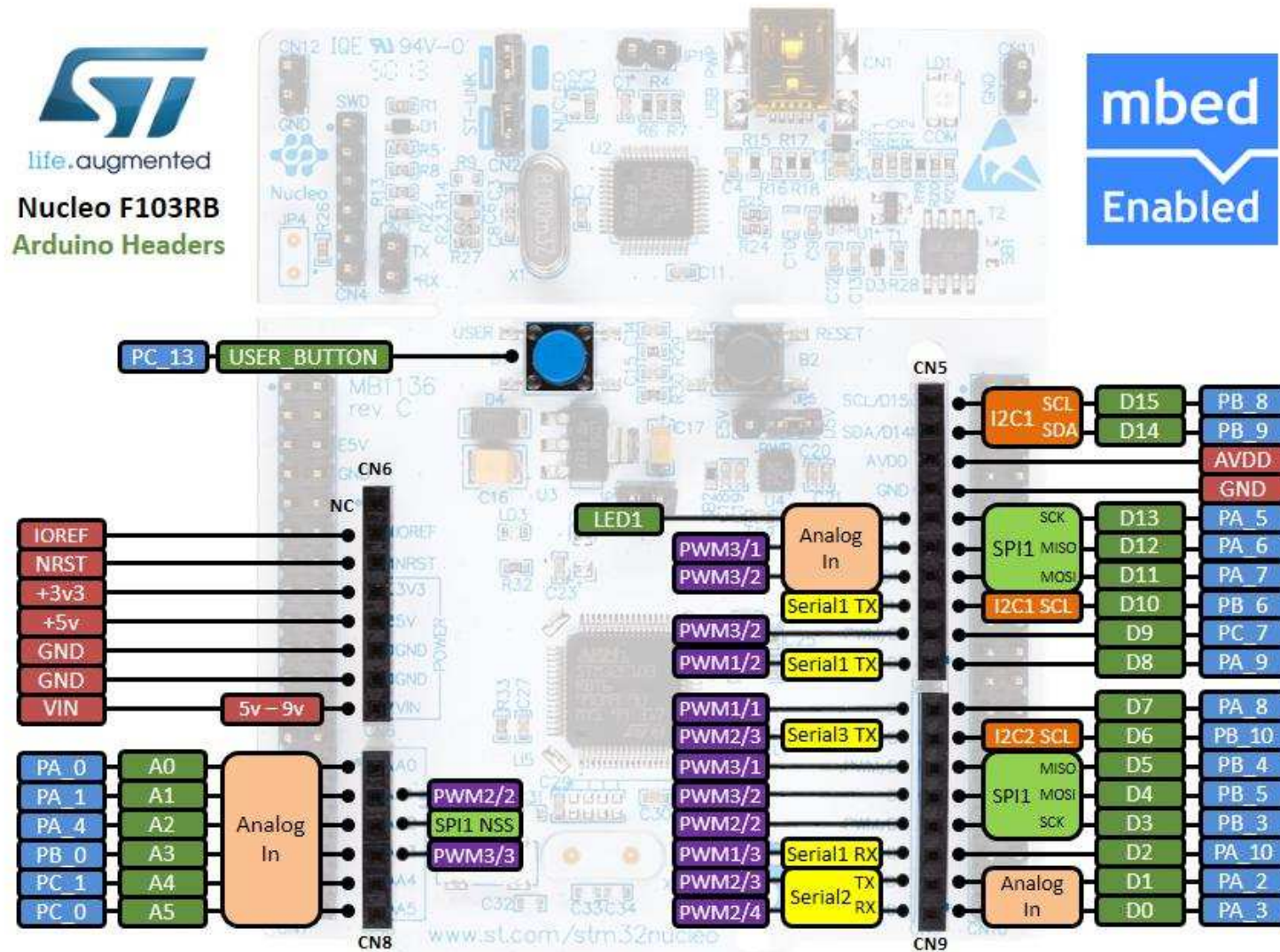
# Představení vývojového kitu

- STM32 Nucleo – osazené STM32F103RB
  - Vybrané signály na "arduino" konektorech
  - Všechny signály na "morpho" konektorech
  - Připojení USB - obsahuje ST-link
    - Implementuje 3 "device" – Debug-SWD, USB/UART, MassStorage (USB)
    - Ovladače přímo od ST
      - ST-Link, ST-Link/V2, ST-Link/V2-1 USB driver signed for XP, Windows7, Windows8
      - ke stažení např. <http://www.st.com/web/en/catalog/tools/PF260219#>
- MBED application shield – pro Arduino (3v3)

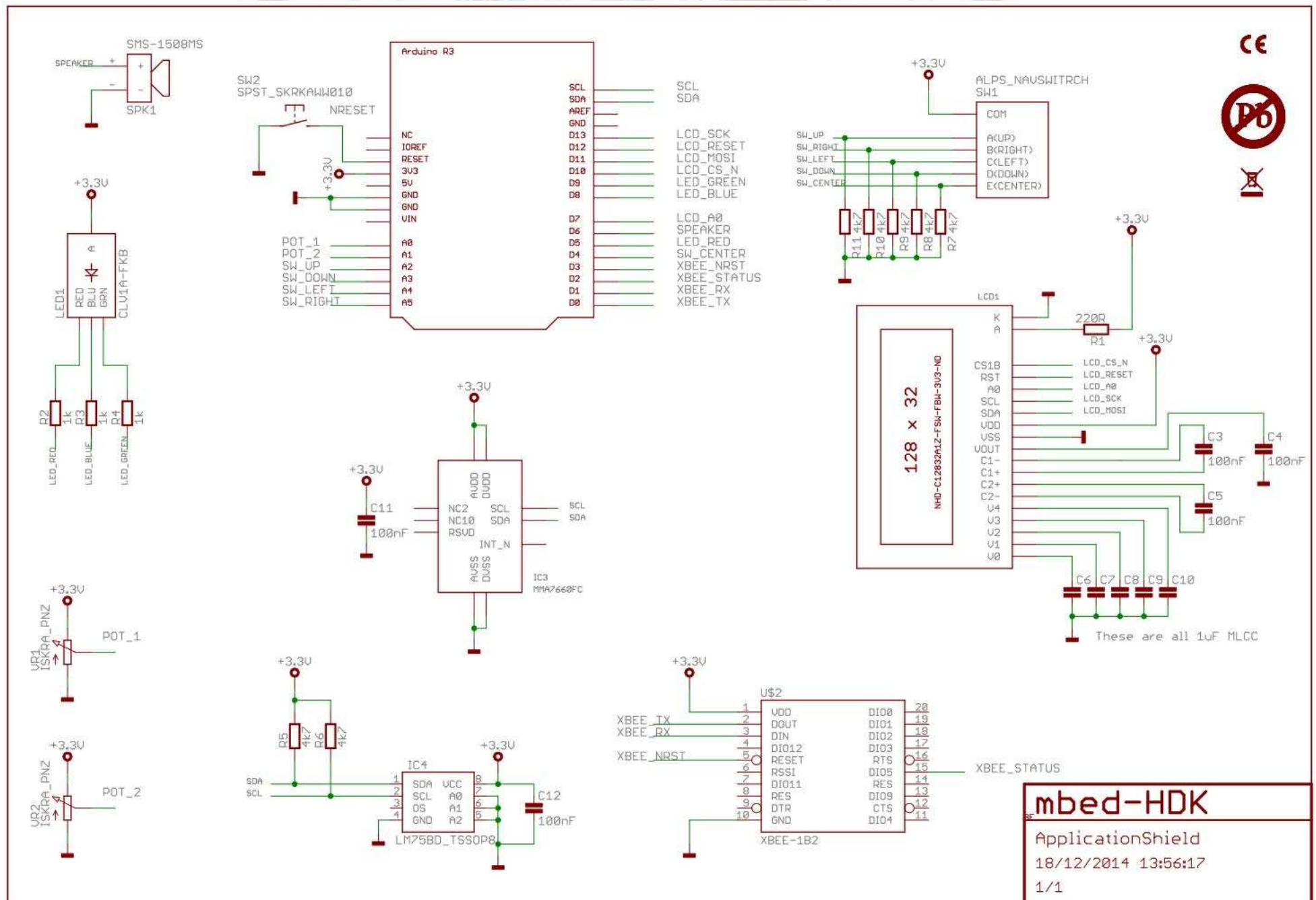




# Popis signálů Nucleo F103RB

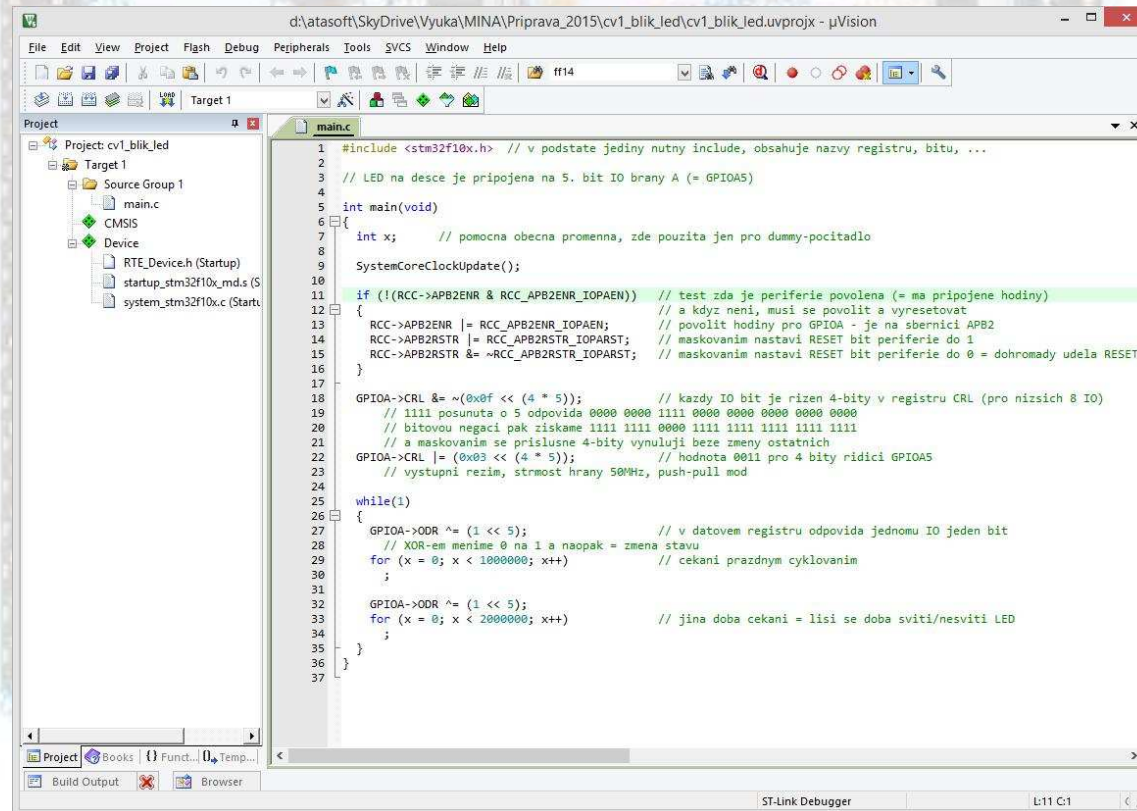


# Schéma zapojení MBED ApplicationShield



# Spuštění prostředí Keil MDK 5

- Na ploše ikona Keil Microvision 5 (zelená) 
- Předpokladem nainstalované "balíčky" (moduly, knihovny)
  - V lab. – splněno
  - Doma – viz. Slajdy pro přípravu, část "Instalace"



```
1 #include <stm32f10x.h> // v podstate jediný nutný include, obsahuje názvy registru, bitu, ...
2
3 // LED na desce je připojena na 5. bit IO brány A (= GPIOA5)
4
5 int main(void)
6 {
7     int x; // pomocná obecná proměnná, zde použita jen pro dummy-počítadlo
8
9     SystemCoreClockUpdate();
10
11     if (!(RCC->APB2ENR & RCC_APB2ENR_IOPAEN)) // test zda je periferie povolena (= má připojené hodiny)
12     { // a když není, musí se povolit a vyresetovat
13         RCC->APB2ENR |= RCC_APB2ENR_IOPAEN; // povolit hodiny pro GPIOA - je na sběrnici APB2
14         RCC->APB2RSTR |= RCC_APB2RSTR_IOPARST; // maskováním nastaví RESET bit periferie do 1
15         RCC->APB2RSTR &= ~RCC_APB2RSTR_IOPARST; // maskováním nastaví RESET bit periferie do 0 = dohromady udělá RESET
16     }
17
18     GPIOA->CRL &= ~(0x0f << (4 * 5)); // každý IO bit je řízen 4-bity v registru CRL (pro nižších 8 IO)
19     // 1111 posunuta o 5 odpovídá 0000 0000 1111 0000 0000 0000 0000 0000
20     // bitovou negací pak získáme 1111 1111 0000 1111 1111 1111 1111 1111
21     // a maskováním se příslušné 4-bity vynulují beze změny ostatních
22     GPIOA->CRL |= (0x03 << (4 * 5)); // hodnota 0011 pro 4 bity řídící GPIOA5
23     // výstupní režim, strmost hrany 50MHz, push-pull mod
24
25     while(1)
26     {
27         GPIOA->ODR ^= (1 << 5); // v datovém registru odpovídá jednomu IO jeden bit
28         // XOR-em meníme 0 na 1 a naopak = změna stavu
29         for (x = 0; x < 1000000; x++) // čekání prázdným cyklováním
30             ;
31
32         GPIOA->ODR ^= (1 << 5);
33         for (x = 0; x < 2000000; x++) // jiná doba čekání = liší se doba svítí/nesvítí LED
34             ;
35     }
36 }
37
```



# Připravený projekt, kompilace, spuštění

- Stáhnout ze Z:\podklady\MINA\cv1\_blik\_led na svůj H:
- Otevřít: Project – Open Project – najít příslušný **.projx**
- Build (kompiluje změněné soubory a linkuje výsledek)
  - Ikona na liště
  - Menu: Project – Build Target
  - Klávesa F7
- Download (to HW)
  - Ikona na liště
  - Klávesa F8
- Bliká zelená LED na Nucleo desce ...
- Další možností spuštění programu je využití Debuggeru



# Program cvičení

1. Úvod, rozdělení kitů, prostředí Keil, nahrání programu
- 2. Debug, bitové operace, GPIO**
3. UART, připojení k PC, využití stdio knihovny
4. Časování, hodiny, SysTick
5. Časovač, přerušení, NVIC
6. SPI a připojení LCD
7. A/D převodník, připojení přes IIC
8. DMA
9. RTOS
10. Samostatná práce
11. Dtto
12. Dtto
13. Dokončení a odevzdání samostatné práce

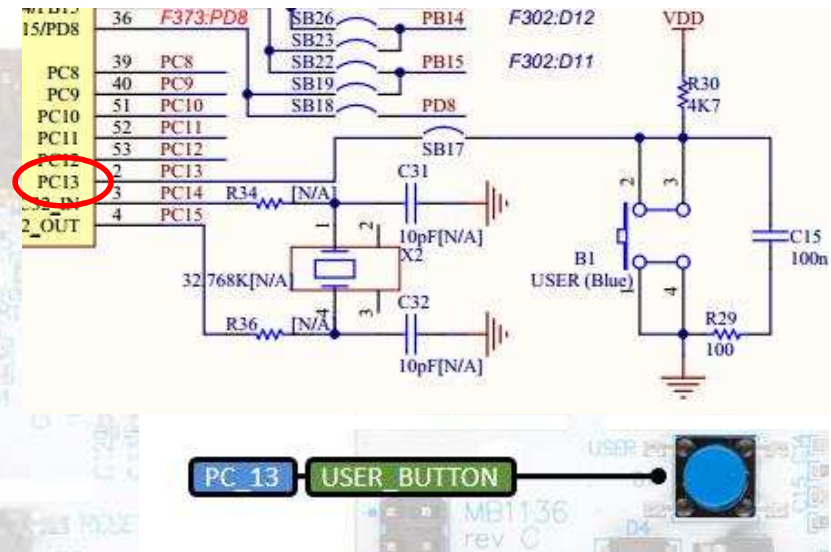
# Z domácí přípravy I

- Umíme vytvořit nový projekt a nakonfigurovat jej
- Umíme přeložit projekt
- Umíme projekt nahrát do Nucleo desky
- Umíme spustit debugger
  - Krokovat program
  - Používat breakpointy
  - Sledovat registry procesoru
  - Sledovat obsah proměnných
- Máme základní znalosti pro práci s GPIO
  - Periférie musí mít povolené hodiny
  - Periférie by měla projít RST cyklem po připojení hodin
  - GPIO mají různé režimy, nastavení ve 4-bitech v GPIOx->CRx
  - Hodnoty bitů možno nastavit v registru GPIOx->ODR nebo BSRR



# Čtení stavu tlačítka

- Nový projekt
- Zkopírovat blikání z cv1
- Inicializace GPIO C13
  - Režim signálu zvolit 1000 (=0x08)
    - "Input with pullup/down"



```
if (!(RCC->APB2ENR & RCC_APB2ENR_IOPCEN))
{
    RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;
    RCC->APB2RSTR |= RCC_APB2RSTR_IOPCRST;
    RCC->APB2RSTR &= ~RCC_APB2RSTR_IOPCRST;
}

GPIOC->CRH &= ~(0x0f << (4 * (13 - 8))); // vyvod 13 je rizen v CRH, kde jsou vyvody 8-15
GPIOC->CRH |= (0x08 << (4 * (13 - 8))); // 1000 - Input with pull up/down, input mode
```

- Stav vývodu testován v GPIOC->IDR
  - Vyčíst konkrétní bit pomocí maskování
  - Zůstává "zobrazen" poslední stav

```
if ((GPIOC->IDR & (1 << 13)) != 0) // nestisknuto (stisknutím se připojí na zem, tj. log 0)
{
    ... Blikání ...
}
```

- **Samostatně** – místo "posledního" stavu nastavit určitý stav
  - log. 1 (svítí) nebo log. 0 (nesvítí)
  - Maskováním stavu registru ODR
  - Zapsání do BSRR

# Makro pro nastavení 4-bitů

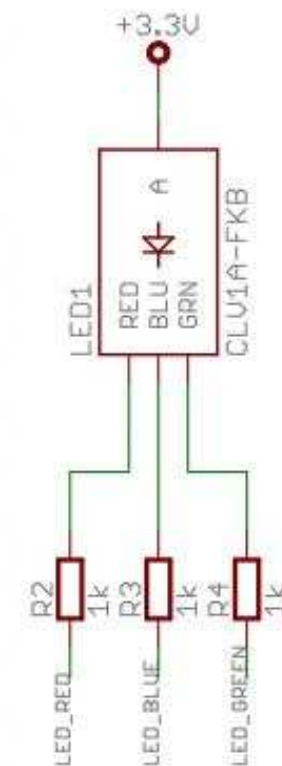
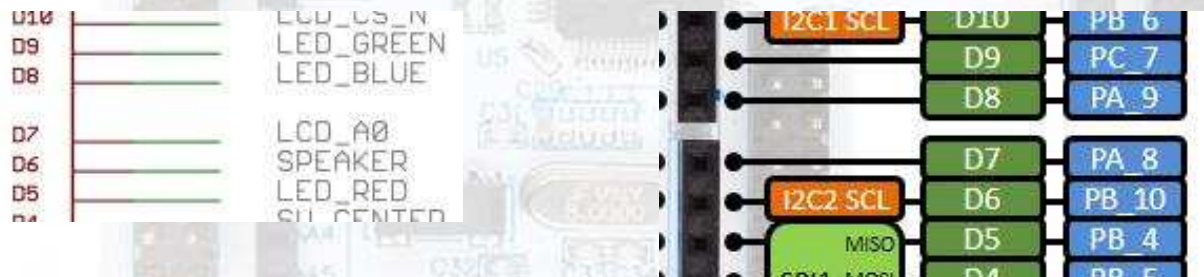
- Nastavování/maskování čtveřic bitů v konfiguračním registru portu (CRL/H) snadno vede na chybu překlepem
- Je výhodné si připravit makro, které celou operaci "zapouzdří"

```
#define SHIFT4(val, nibble) ((val) << (4 * (nibble)))  
  
...  
  
GPIOA->CRL &= ~SHIFT4(0x0f, 5);  
GPIOA->CRL |= SHIFT4(0x03, 5);
```

- Pozor na závorkování
  - V makrech se parametry při překlada nekontrolují na typy a mohlo by dojít na "záludné" chyby
- Vícebitová (zde 4-bitová) konfigurace bitů se v registrech procesoru využívá často, makro se bude "hodit"

# Využití RGB LED - mbed shield

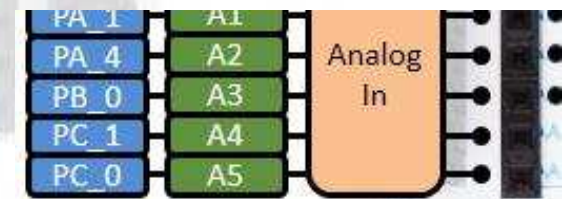
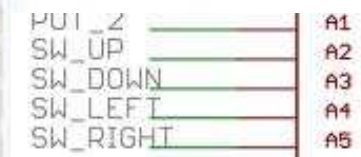
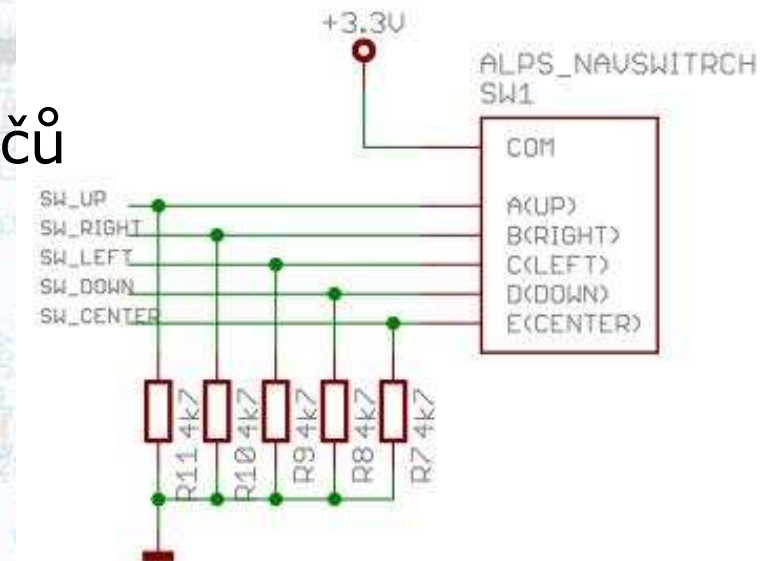
- Na mbed-shield je RGB LED připojena z 3v3 na vývody procesoru
  - R = D5 = PB4
  - G = D9 = PC7
  - **B = D8 = PA9**
- Pro ověření funkčnosti využijeme modrou
  - GPIOA 9 – změníme pouze z PA5 (on-board LED)
  - Viz. video <https://youtu.be/4OL-jZWbrh0>





# Využití křížového ovladače/tlačítek

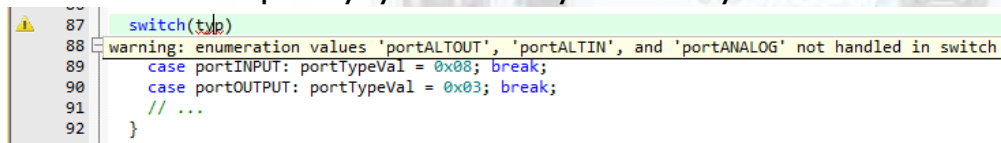
- Křížový ovladač má interně 5 spínačů
  - Sepnutý kontakt je Vcc (log. 1), neseprnutý 0
  - Down            A3 = PB0
  - Left            A4 = PC1
  - **Center**        **D4 = PB5**
  - Up            A2 = PA4
  - Right          A5 = PC0
- Pro ověření funkce využijeme středové tlačítko
  - Připojeno na GPIOB 5
  - Nutno přidat inicializaci GPIOB brány a pinu 5 jako INPUT





# GPIO inicializace – jako funkce – II

- Inicializace "portu" pokud ještě nebyla provedena
- Hodnota pro konfiguraci je 4-bitová
  - Viz. popis registrů CRx
  - Keil MDK upozorní, pokud nejsou "pokryty" všechny hodnoty enum



- Hodnota "konfigurace" se zapíše do příslušného CRx registru
  - Pro IO 0-7 je to CRL
  - Pro IO 8-15 je to CRH, nutno odečíst 8, viz. popis CRx v RM

```
...
if (!(RCC->APB2ENR & bitPortEn))
{
    RCC->APB2ENR |= bitPortEn;
    RCC->APB2RSTR |= bitPortRst;
    RCC->APB2RSTR &= ~bitPortRst;
}
...
```

```
...
switch(typ)
{
    case portINPUT: portTypeVal = 0x08; break;
    case portOUTPUT: portTypeVal = 0x03; break;
    // ... A další kombinace
}

if (portTypeVal >= 0x100) // zadny shodny case ?
    return 0;           // konec s chybou
...
```

```
...
if (portNum < 8) // pro spodnich 8 je v CRL
{
    baseGPIO->CRL &= ~SHIFT4(0x0f, portNum);
    baseGPIO->CRL |= SHIFT4(portTypeVal, portNum);
}
else // pro hornich 8 je v CRH
{
    portNum -= 8; // CRH je pro IO 8-15
    baseGPIO->CRH &= ~SHIFT4(0x0f, portNum);
    baseGPIO->CRH |= SHIFT4(portTypeVal, portNum);
}

return 1;
}
```



# Využití funkce pro inicializaci IO

- Úvodní inicializace IO signálů se zjednoduší

```
...
int main(void)
{
    int x;        // dummy-pocitadlo

    // varianta pro on-board
    InitIOPort(GPIOA, 5, portOUTPUT);
    InitIOPort(GPIOC, 13, portINPUT);

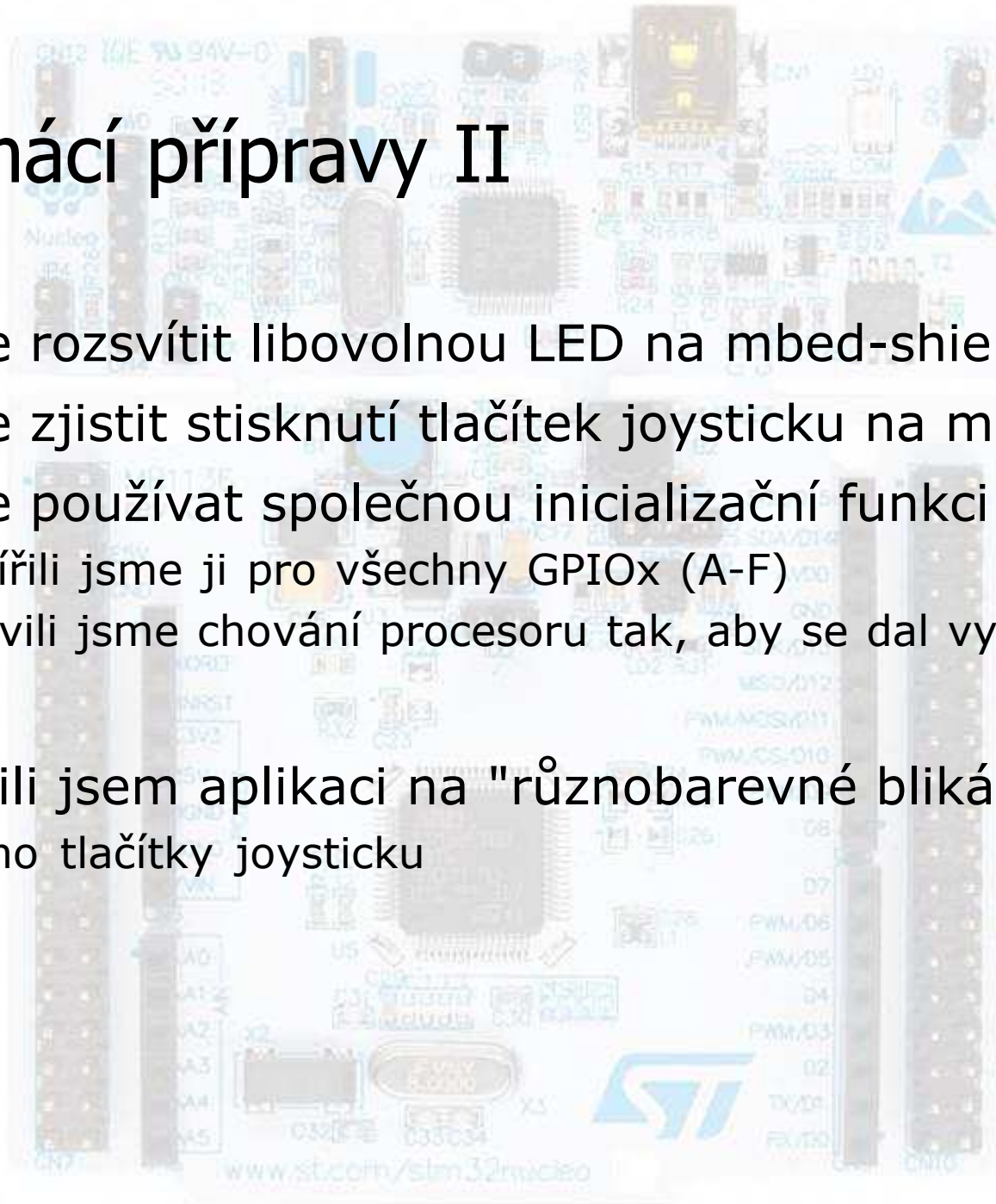
    while(1)
    {
        ...
    }
}
```

# Program cvičení

1. Úvod, rozdělení kitů, prostředí Keil, nahrání programu
2. Debug, bitové operace, GPIO
- 3. UART, připojení k PC, využití stdio knihovny**
4. Časování, hodiny, SysTick
5. Časovač, přerušení, NVIC
6. SPI a připojení LCD
7. A/D převodník, připojení přes IIC
8. DMA
9. RTOS
10. Samostatná práce
11. Dtto
12. Dtto
13. Dokončení a odevzdání samostatné práce

# Z domácí přípravy II

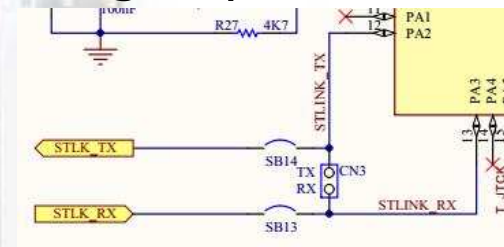
- Umíme rozsvítit libovolnou LED na mbed-shield
- Umíme zjistit stisknutí tlačítek joysticku na mbed-shield
- Umíme používat společnou inicializační funkci IO portů
  - Rozšířili jsme ji pro všechny GPIOx (A-F)
  - Upravili jsme chování procesoru tak, aby se dal využívat i port PB4
- Vytvořili jsem aplikaci na "různobarevné blikání"
  - Řízeno tlačítky joysticku



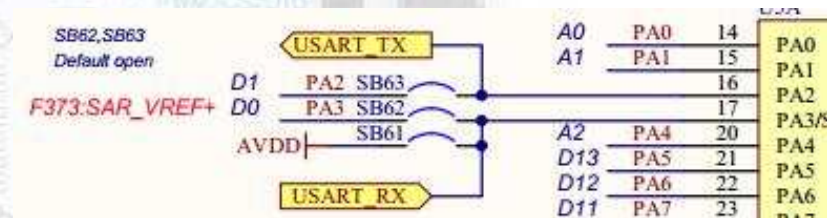


# Propojení uP a PC pomocí UART

- ST-Link vytváří přes USB také virtuální COM port
  - Na PC pracujeme s "klasickým" sériovým portem COMx
- Na straně Nucleo desky jsou vyvedeny Rx a TX signály
  - Dle schématu STLK\_TX a STLK\_RX z ST-Link
    - Viz. str.3



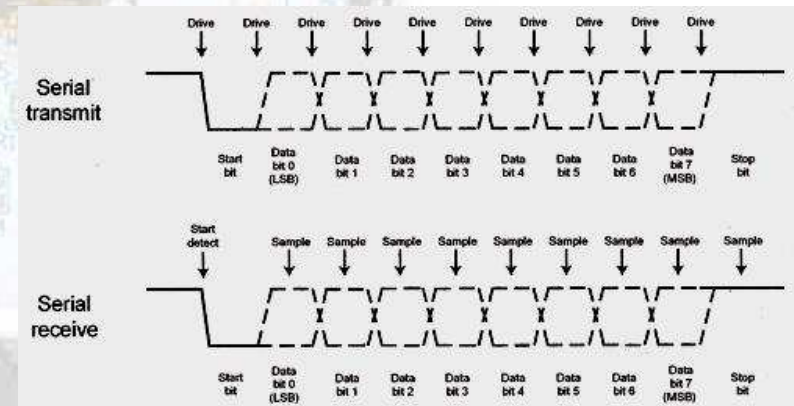
- Připojeno na USART\_TX a USART\_RX "našeho" uP (viz. str.1)
- Na uP připojeny na PA2 a PA3



- Alternativně možno komunikovat přes UART na "Arduino" konektoru – signály D0 a D1
  - Nutno spojit propojky SB62 a SB63 a naopak rozpojit SB14 a SB13
    - Pokud SB13 zůstane, bude možné na PC "odposlouchávat" vysílaná data z uP

# Vlastnosti U(S)ART

- Sériový přenos bajtově orientovaný
  - Nastavitelná volba 8- nebo 9-bit
  - LS-bit se vysílá první
  - 1 startbit, 1/1,5/2 stop bity
  - V PC používán asynchronní přenos – GND, TxD, RxD
- Hodiny generované z na APB1 (max. 36MHz)
  - Platí pro USART2,3 a UART4,5
  - Pozor, USART1 je na APB2 (max. 72MHz)
  - Rychlost sběrnice nastavena v PPRE1 (resp. PPRE2)
    - Registr RCC\_CFGR – viz. kap. 7.3.2
- Prostřednictvím bloku AFIO mohou být USARTx přivedeny na "různé" dvojice vývodů
  - Registr REMAP – viz. RM, kap. 9.4.2



## Bit 3 USART2\_REMAP: USART2 remapping

This bit is set and cleared by software. It controls the mapping of USART2 CTS, RTS, CK, TX and RX alternate functions on the GPIO ports.

0: No remap (CTS/PA0, RTS/PA1, TX/PA2, RX/PA3, CK/PA4)

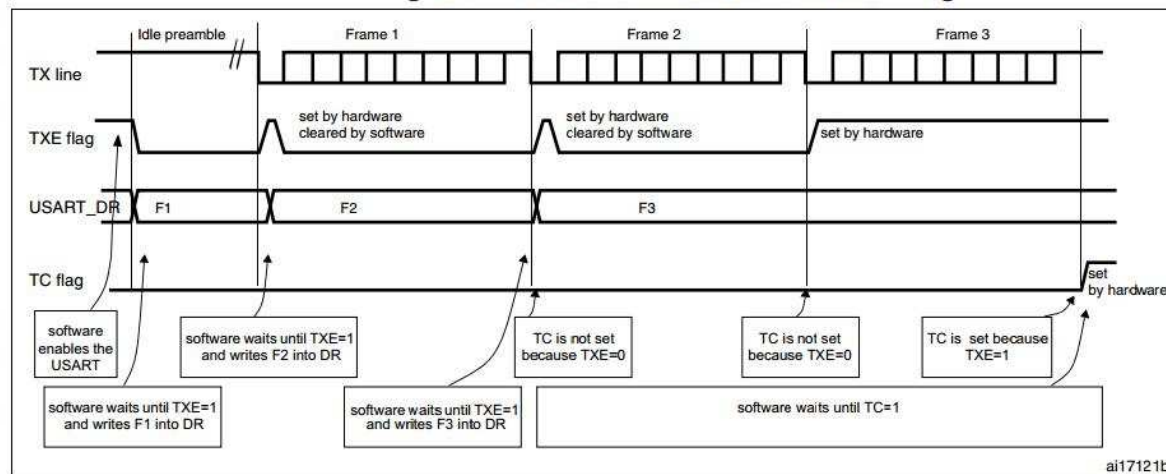
1: Remap (CTS/PD3, RTS/PD4, TX/PD5, RX/PD6, CK/PD7)

- Pokud není nutno "remapovat", blok AFIO se nemusí použít

# Detaily příjmu a vysílání

- Kap. 27.2 v RM – popis vlastností USARTu
  - Samostatně najít
- Kap. 27.3.2 – Transmitter
  - Figure 281 - TC/TXE behavior when transmitting

Figure 281. TC/TXE behavior when transmitting



- Samostatně prostudovat podmínky pro hodnoty příznaků TXE a TC
- Kap. 27.3.3 – Receiver
  - Figure 282. – Detekce START bitu
  - Figure 283. – Vzorkování vstupu pro detekci šumu



# Baud Rate Register

## 27.6.3 Baud rate register (USART\_BRR)

Note: The baud counters stop counting if the TE or RE bits are disabled respectively.

Address offset: 0x08

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

- Určuje dělicí poměr pro generování přenosové rychlosti
  - Podíl může být necelý
    - 12 bitů mantissa = celá část
    - 4 bity fraction = "desetinná" část jako poměr v šestnáctinách, tj. x/16

$$\text{Tx/ Rx baud} = \frac{f_{CK}}{(16 * \text{USARTDIV})}$$

legend:  $f_{CK}$  - Input clock to the peripheral (PCLK1 for USART2, 3, 4, 5 or PCLK2 for USART1)

- Dle RM kap. 27.3.4 platí:
  - Tj. pro určení **BRR** vzorec upravíme

$$\text{UsartDiv} = \frac{\text{APB1Clock}}{16 * \text{BaudSpeed}}$$

- Konkrétní výpočet:
  - APB1Clock = 36MHz (72MHz Core / 2) = reálně max. rychlost APB1
  - BaudSpeed = 38400
  - Výsledek = 58,59375
- Celá část bude 58 (=0x3A)
- Desetinná část odhadem
  - 8/16 = 0,5
  - 9/16 = 0,5625 (rozdíl k vypočtené = - 0,03125)
  - 10/16 = 0,625 (rozdíl k vypočtené = 0,03125)
  - Je možno volit 9 (=0x9) nebo 10 (=0xA) bez ztráty přesnosti

$$\frac{\text{Fraction}}{16} = \frac{\text{Des\_cast}}{100000}$$

- Desetinná část výpočtem vychází z přímé úměry
  - Výsledek 9,5 vede také na 9 nebo 10

$$\text{Fraction} = \frac{16 * \text{Des\_cast}}{100000} = \frac{16 * 59375}{100000} = 9,5$$

- Výsledná hodnota do **BRR** = **0x03AA** (nebo 0x03A9)
  - Tj. "(celá\_část << 4) + (desetinná\_část & 0x0F)"

# USART registry Status Reg., Data Reg.

## 27.6.1 Status register (USART\_SR)

Address offset: 0x00

Reset value: 0x00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
Reserved						rc_wd	rc_wd	f	rc_wd	rc_wd	f	f	f	f	f

- **TXE** = Transmit data register empty
  - Nastaven (do log. 1), když je přesunut obsah datového registru do vysílacího shift-registru
  - Nulován zápisem do datového registru
- **TC** = Transmission complete
  - Nastaven při dokončení rámce (parita a stop-bity)
  - Vynulovat nutno "ručně"
- **RXNE** = Read data register not empty
  - Nastaven při přenesení dat z přijímacího shift-registru do DR
  - Vynulován čtením datového registru
- **ORE** = Overrun error
  - Nastaven v případě, že přišla nová data a "stará" nebyla vyzvednuta z DR
    - Nová data jsou ztracena, obsah DR zůstává
  - Nulování "ručně" po vyřešení problému komunikace
- **NE** = Noise error flag
  - Nastaven pokud je detekován šum při příjmu některého bitu ve znaku
- **FE** = Framing error
  - Nastaven při detekování chyby rámce, typicky špatné stop-bity
- **PE** = Parity error
  - Nastaven když nevyšla požadovaná parita

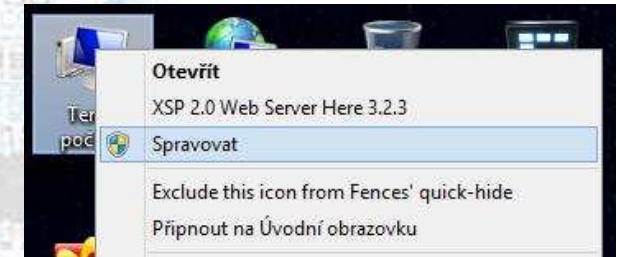
- Config Reg. 1 - využívané
  - **UE** = USART Enable
    - Povolení funkce USARTu
  - **PCE** = Parity control enable
  - **PS** = Parity selection
    - 0 = Even (sudá), 1 = Odd (lichá)
  - **PEIE, TXEIE, TCIE, RXNEIE**
    - Povolení generování přerušení při splnění podmínek ve Status R
  - **TE, RE**
    - Povolení Transmitter a Receiver funkcionality
- Config Reg. 2 – především nastavení pro **USART** režim
- Config Reg. 3 – především DMA
  - **EIE** = Error interrupt enable
    - Nutný pro generování chybových přerušení v DMA režimu

[illegible]

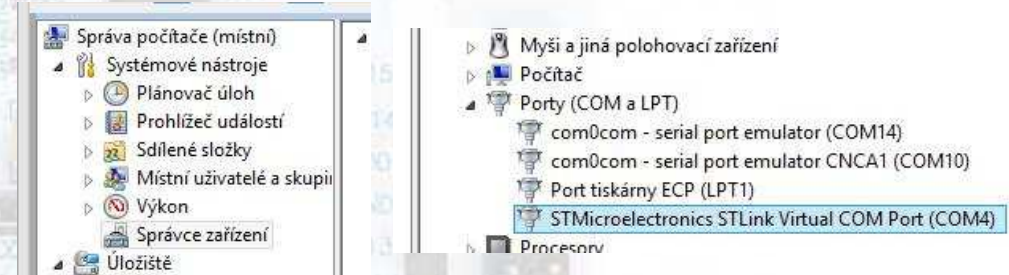


# Terminálové programy

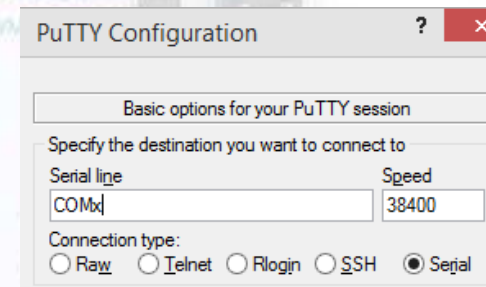
- Zjištění připojeného COM portu
  - Tento počítač - (pravým tlačítkem) – Správa počítače



- Správce zařízení
- STM... Virtual COM Port



- Putty
  - Volba Serial, nastavit rychlost
  - Uložit pro příště, možná již uloženo ?



- Hercules Terminal

# Vlastní funkce pro vysílání/příjem

- Vysílání bajtu
  - Před začátkem odesílání bajtu počkat na uvolnění vysílacího datového registru
    - *Transmit Data Register Empty*
  - Zapsat data k odeslání do datového registru
- Příjem bajtu
  - Vyčkat než nějaká data přijdou do přijímacího datového registru
    - *Read data register not empty*
  - Přečíst datový registr a vrátit z funkce
- Před použitím nezapomenout inicializovat UART
  - Potřebuje mít "známé" hodiny, tj. jak je nastaven SystemCoreClock – používá k výpočtu hodnoty do BRR registru

```
int sendUart(int ch)
{
    ...
}

int recvUart(void)
{
    ...
}

int main(void)
{
    char c = 'A';

    SystemCoreClockUpdate();
    Uart2Init(38400);

    while(1)
    {
        sendUart(c);
        c++;
        if (c > 'Z')
            c = 'A';

        cekej(1000000); // cekani for
    }
}
```

# Inicializace U(S)ART2

- Povolit hodiny z APB1
- CR registry
  - Stačí jen RE a TE bit
- Spočítat a nastavit BRR
- Povolit globálně CR1\_UE

```
void Uart2Init(int baudrate)
{
    if ((RCC->APB1ENR & RCC_APB1ENR_USART2EN) == 0)
    {
        // not running yet ?
        RCC->APB1ENR |= RCC_APB1ENR_USART2EN;
        RCC->APB1RSTR |= RCC_APB1RSTR_USART2RST;
        RCC->APB1RSTR &= ~RCC_APB1RSTR_USART2RST;
    }

    USART2->CR1 = USART_CR1_RE | USART_CR1_TE;
    USART2->CR2 = 0;
    USART2->CR3 = 0;

    {
        ... Vypocet BRR
    }

    USART2->CR1 |= USART_CR1_UE;

    InitIOPort(GPIOA, 2, portALTOUT);
    InitIOPort(GPIOA, 3, portALTIN);

    // AFIO->REMAP not required, default "mode"
}
```

- Nezapomenout nastavit využití vývody (GPIO porty)
  - Rx na vstup (float)
  - Tx na Alt. Output s Push/pull
- Případně nastavit příslušný REMAP bit v AFIO



# Výpočet hodnoty pro rychlost – reg. BRR

```
{
uint32_t tmpreg = 0x00, apbclock = 0x00;
uint32_t integerdivider = 0x00;
uint32_t fractionaldivider = 0x00;

apbclock = RCC->CFGR & RCC_CFGR_PPRE1;    // 0x00000700 = 0..0111 0000 0000 = zachovej bity 10:8, zbytek zahodit
apbclock >>= 8;

if ((apbclock & 0x04) == 0)    // test higher bit from selected 3 bits
    apbclock = SystemCoreClock; // 0xx = AHB not divided, use CoreClock (typically equal to AHB))
else
    // compute from AHB resp. Core: 100 = AHB/2, 101 = AHB/4, 110 = AHB/8, 111 = AHB/16
    apbclock = SystemCoreClock >> ((apbclock & 0x03) + 1); // lower 2 bits

integerdivider = ((25 * apbclock) / (4 * baudrate));    // 100 * APBClock / (16 * BaudRate)
tmpreg = (integerdivider / 100) << 4;    // integer division, shift to "mantissa" part

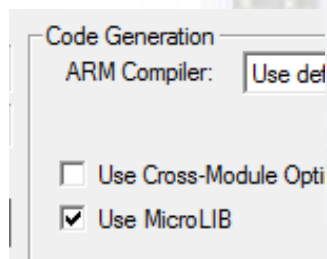
fractionaldivider = integerdivider - (100 * (tmpreg >> 4)); // rest = fractional part
tmpreg |= (((fractionaldivider * 16) + 50) / 100) & ((uint8_t)0x0F); // convert hundredth to 1/16 and cut lower 4bits

USART2->BRR = (uint16_t)tmpreg;    // Write to USART BRR register
}
```

1. Zjištění aktuální frekvence APB1 sběrnice
  - Vybrat PPRE1 bity z CFGR registru
  - Rozhodnout jaký je dělicí poměr (1, 2, 4, 8, 16)
  - Předpokládáme, že výchozí AHB jede na SystemCoreClock frekvenci
    - A že SystemCoreClock obsahuje platnou hodnotu
2. Chceme počítat celočíselně = vše počítáme 100x větší a uvažujeme, že se pohybujeme v setinách
  - Místo  $100 * \dots / (16 * \dots)$  počítáme  $25 * \dots / (4 * \dots)$ , protože násobíme hodinovou freq, která je v řádu  $10^6$  a už bychom narazili na limit rozsahu uint32 ( $2^{32}-1 = 4.29 * 10^9$ , tedy v 100x jsou max. hodiny 42MHz)
  - Celočíselnou část odřízneme dělením 100, posuneme vlevo o 4b pro BRR
3. Desetinnou část vezmeme jako zbytek po celé části (100x)
  - Přepočítáme setiny na šestnáctiny, kvůli celočíselnému „odříznutí“ nahradíme zaokrouhlení přičtením  $\frac{1}{2}$ , tedy 50 pro setiny
  - Pro jistotu zamaskujeme pouze spodní 4b hodnoty

# MicroLIB jako náhrada knihovny stdio, použití std. funkcí

- Při použití funkcí z knihovny STDIO se interně volají low-level funkce
  - Pro odeslání bajtu: **int fputc(int ch, FILE \*f)**
  - Pro příjem bajtu: **int fgetc(FILE \*f)**
  - Tělo funkcí musí dodat aplikace
  - Pro ARM procesory vyžadováno použití "omezené" verze knihovny, tzv. **MicroLIB**
    - Viz. např. <http://www.keil.com/arm/microlib.asp> – je úspornější
    - Nebo [http://www.keil.com/support/man/docs/armlib/armlib\\_chr1358938940288.htm](http://www.keil.com/support/man/docs/armlib/armlib_chr1358938940288.htm)
  - V kódu komunikace je velmi vhodné otestovat zapnutí volby této knihovny v projektu
  - Pak je možno používat běžné funkce typu getchar, putchar, puts, printf, ...
- Inicializace, indikace přijatých znaků apod. řeší aplikace sama



```
#ifndef __MICROLIB
#error Musi byt zapnuta MicroLIB v projektu !!
#endif

int fputc(int ch, FILE *f)
{ ... Stejně jako sendUart
}

int fgetc(FILE *f)
{ ... Stejně jako recvUart
}

unsigned char isCharRecv(void)
{
    return (USART2->SR & USART_SR_RXNE); // log 1 = something in recv buffer
}
```

# Program cvičení

1. Úvod, rozdělení kitů, prostředí Keil, nahrání programu
2. Debug, bitové operace, GPIO
3. UART, připojení k PC, využití stdio knihovny
- 4. Časování, hodiny, SysTick**
5. Časovač, přerušení, NVIC
6. SPI a připojení LCD
7. A/D převodník, připojení přes IIC
8. DMA
9. RTOS
10. Samostatná práce
11. Dtto
12. Dtto
13. Dokončení a odevzdání samostatné práce



# Z domácí přípravy III

- Umíme nastavit sériovou komunikaci
- Umíme přijímat a vysílat data pomocí stdio funkcí
  - Program testuje, zda jsou k dispozici data k příjmu
  - Funkce související s UART-komunikací jsou ve zvláštním souboru
    - Logicky včetně příslušného hlavičkového souboru
- Vytvořili jsme aplikaci pro RGB a Joystick řízenou z PC
  - Lze volit barvu svícení/blikání
  - Do PC se hlásí zvolená barva
  - Do PC se hlásí stisknuté tlačítko joysticku

# Struktura programu s „knihovnou“

- Vytvořit adresář "mimo projekt"
  - Bude obsahovat \*.c a \*.h zdroje knihovny
  - Zatím UART funkce a MBED (InitIOPort, ...)
- Nová „Source group“ v projektu
  - Přidat uart.c a mbed.c z adresáře knihovny
- Při překladu nemůže najít mbed.h a uart.h
  - Přidat adresář s knihovnou do vlastností projektu – Záložka C/C++ – volba Include Paths
- Ověřit funkčnost na příkladu blikání

```
#include "mbed.h"
#include "uart.h"

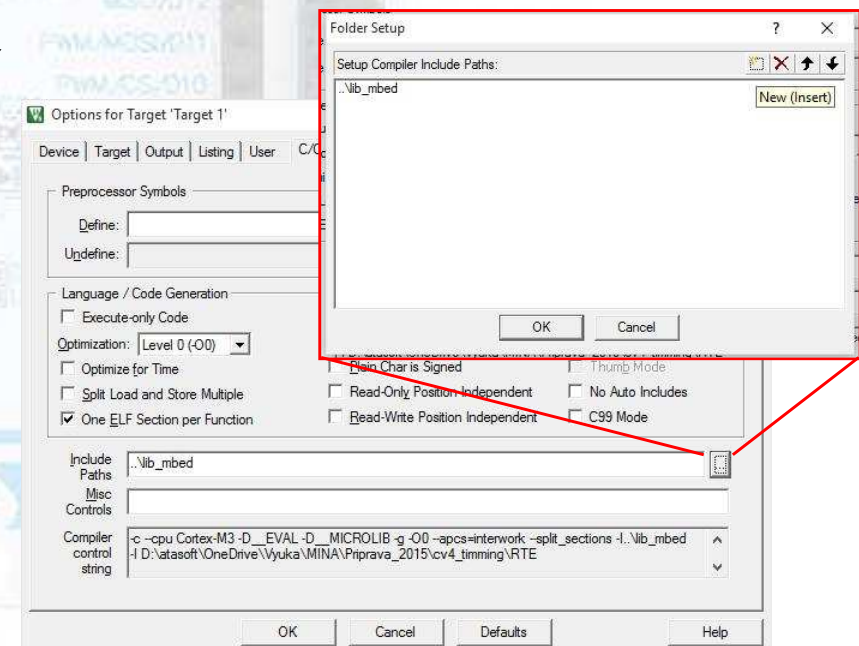
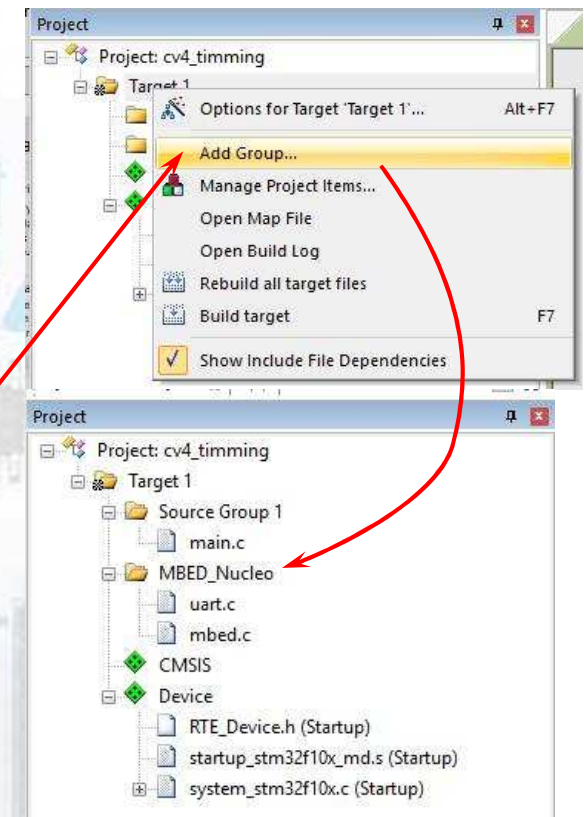
int main(void)
{
    char c = 'X';

    SystemCoreClockUpdate();
    Uart2Init(38400);

    InitIOPort(GPIOA, 5, portOUTPUT);    // ON-BOARD led

    printf("Core: %d\r\n", SystemCoreClock);

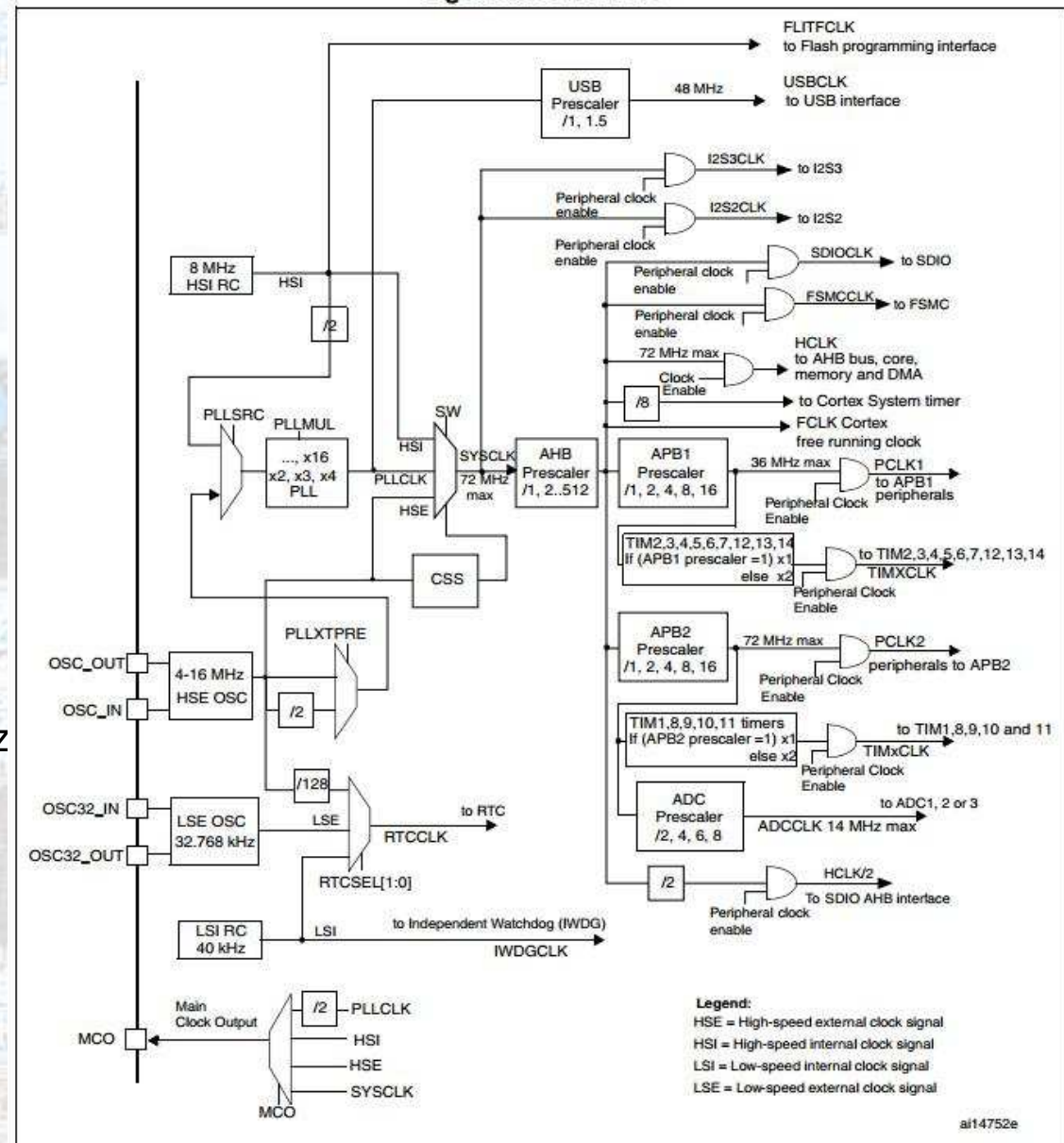
    ... blikání LED
```



# STM32F1 clock

- **SYSCLK** – základní takt (max. 72)
  - **HSI**
    - High Speed Internal
    - Default po RESETu
    - 8MHz
  - **HSE**
    - High Speed External
    - 4-16MHz, zde 8MHz
  - **PLL násobička**
    - x2-x16 z HSI nebo HSE
    - Mezi HSI a PLL je /2
- **AHB** – základní sběrnice (max. 72MHz)
  - Dělena z SYSCLK
  - Zdroj pro všechny další "takty"
- **Low Speed External**
  - Pro RTC
  - Na Nucleo C3 je 32,768kHz
- **Low Speed Internal**
  - 40kHz
  - Pro WDG, příp. RTC

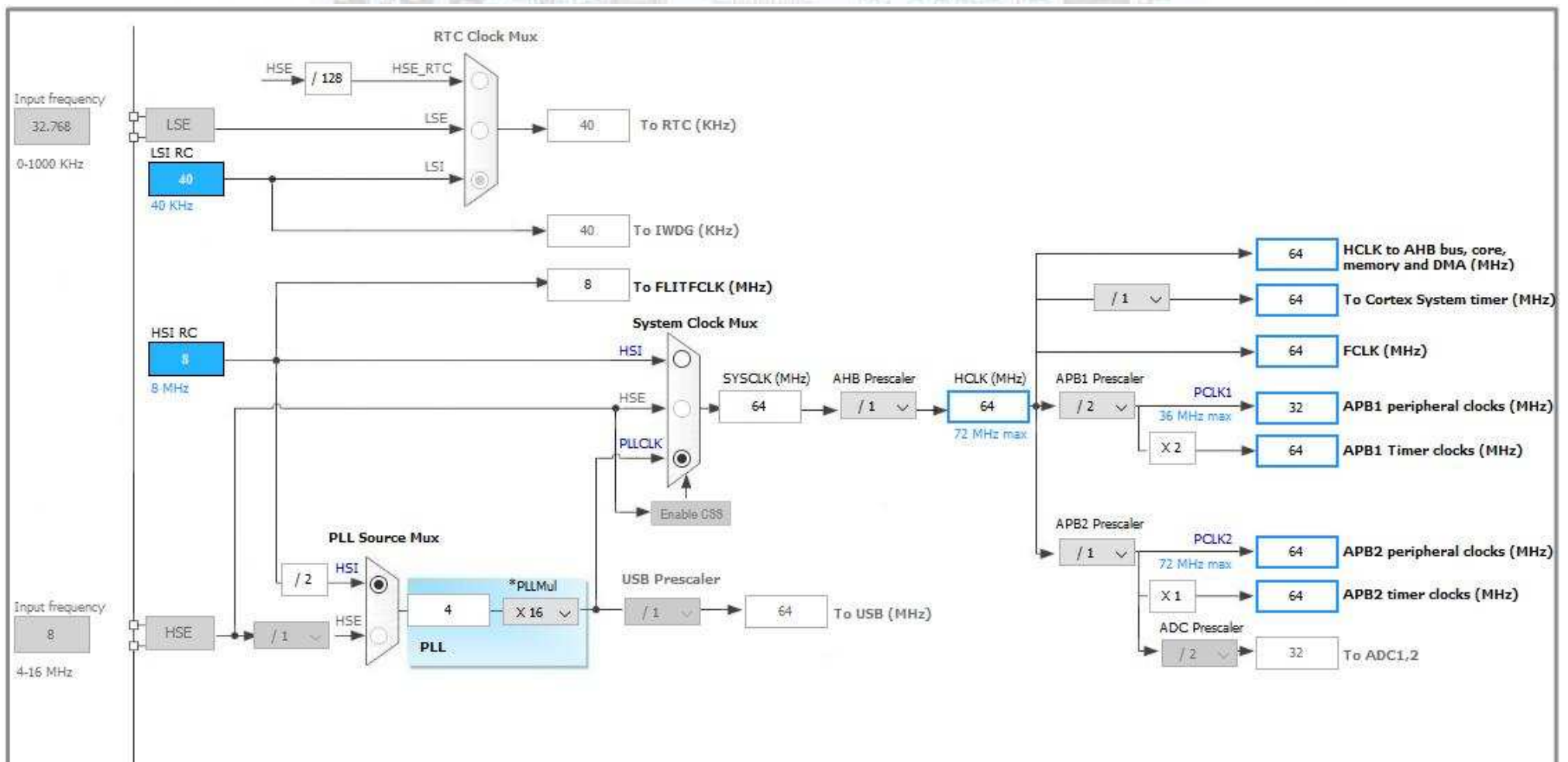
Figure 8. Clock tree



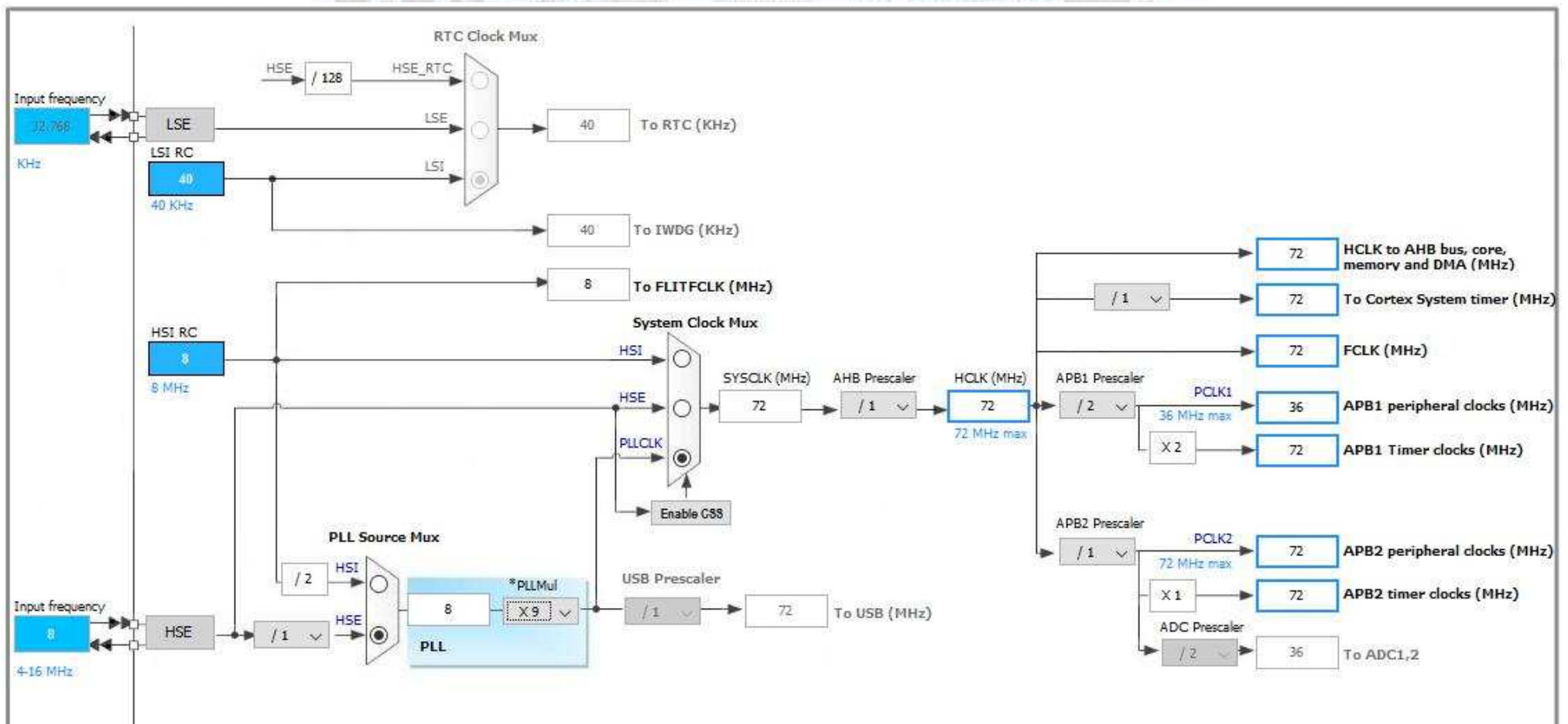
1. When the HSI is used as a PLL clock input, the maximum system clock frequency that can be achieved is 64 MHz.
2. For full details about the internal and external clock source characteristics, please refer to the "Electrical characteristics" section in your device datasheet.



# Konfigurace pro HSI a PLL pro 64MHz

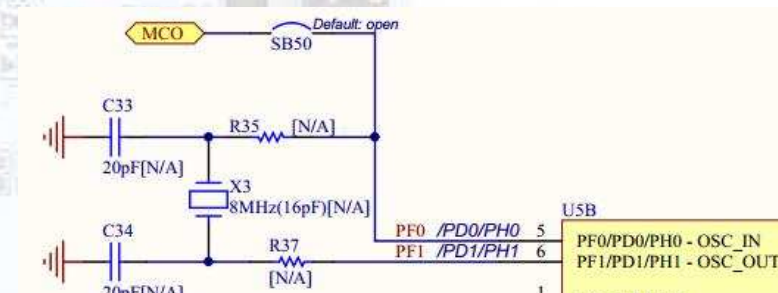
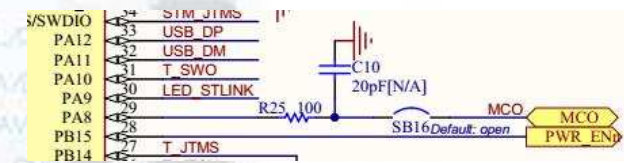
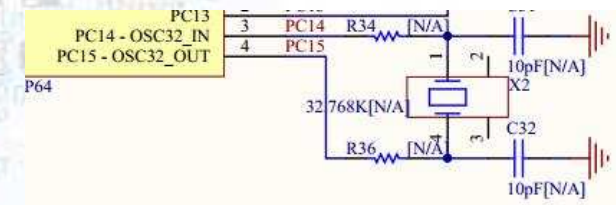


# Konfigurace HSE 8MHz a PLL pro 72MHz



# Nucleo deska a zdroje hodin

- Nucleo Rev C1
  - Žádný krystal
  - Není propojení s ST-Link MHz
  - Nutno využít HSI
- Nucleo Rev C3
  - Osazen krystal 32,768kHz pro LSE
  - Nastavené propojky pro využití HSE z ST-Link (MCO)
  - HSE má 8MHz





# RCC – Reset and Clock Control pro MD

- RM 7.2 Clocks
  - HSE, HSI, PLL, LSI, LSE
  - RM 7.2.6 SysClock selection
    - Po RESETu zvolen HSI jako zdroj hodin
    - Při změně zdroje hodin se kontroluje, zda je "ready", jinak se nepřepne
- RCC\_CR – Control reg.
  - Jednotlivé zdroje hodin mají xxxRDY příznak, že je možné je využít
  - Zdroje hodin možno "vypnout"
    - Blokováno, pokud je daný zdroj právě používán
- RCC\_CFGR – Clock Config.
- RCC\_CIR – Interrupt register
  - Řídí možnosti přerušení – generované např. při "ztrátě" HSE
- RCC\_APBxRSTR, RCC\_APBxEN, ... – řízení hodin pro periférie

### 7.3.1 Clock control register (RCC\_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved						PLL RDY	PLLON	Reserved				CSS ON	HSE BYP	HSE RDY	HSE ON
						r	rw					rw	rw	r	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]					Res.	HSI RDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

- PLLON – zapnutí PLL
- PLLRDY – PLL stav ready = fázový závěs zavěšen
- HSEON – zapnutí HSE
- HSERDY – HSE stav ready
  - Oscilátor stabilní po dobu 6 cyklů
- HSICAL – interní kalibrace HSI, R/O
- HSITRIM – doladění HSI
  - Vhodné např. pro kompenzaci teploty
  - Defaultní hodnota 16 (0x10), tj. polovina 5-bit rozsahu
  - Při zápisu celého registru nenechávat 0 !!
- HSION – zapnutí HSI
  - Defaultní po RESETu
- HSIRDY – HSI stav ready
  - Oscilátor stabilní po dobu 6 taktů

### 7.3.2 Clock configuration register (RCC\_CFGR)

Address offset: 0x04

Reset value: 0x0000 0000

Access:  $0 \leq \text{wait state} \leq 2$ , word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during clock source switch.

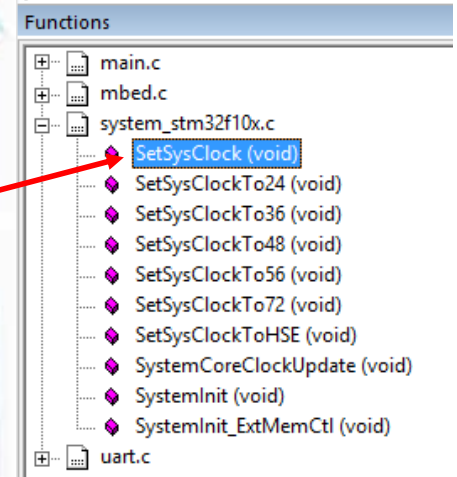
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					MCO[2:0]			Res.	USB PRE	PLLMUL[3:0]				PLL XTPRE	PLL SRC
					rw	rw	rw		rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADCPRE[1:0]		PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

- PLLMUL – násobící poměr PLL
  - V rozsahu x2 – x16
- PLLXTPRE – předdělení HSE /2
- PLLSRC – PLL source
  - 0 = HSI
  - 1 = HSE
- ADCPRE – dělení pro ADC /2, /4, /6, /8
- PPRE2 – dělení pro APB2 sběrnici (max. 72MHz)
  - /1, /2, /4, /8, /16
- PPRE1 – dělení pro APB1 sběrnici (max. 36MHz)
  - /1, /2, /4, /8, /16
- HPRE – dělení pro AHB sběrnici (max. 72MHz)
  - /1, /2, /4, /8, ... /512
  - V případě dělení více jako 1 nutno zapnout "prefetch buffer" ve FLASH\_ACR
- SWS – Systém Clock Switch Status
  - Aktuálně vybrané zdroj hodin, vhodné kontrolovat po přepnutí, využívá SystemCoreClockUpdate()
- SW – System Clock Switch – volba zdroje hodin
  - 00: HSI selected as system clock
  - 01: HSE selected as system clock
  - 10: PLL selected as system clock
  - 11: not allowed

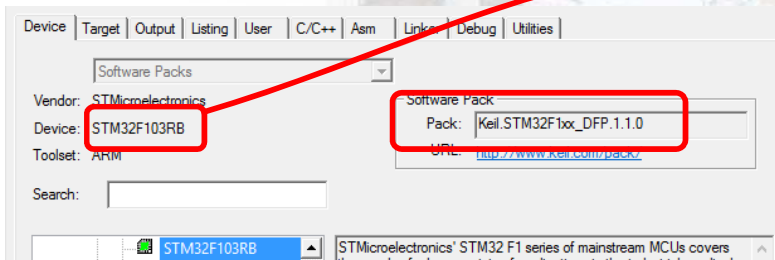


# Knihovny funkce pro nastavení CLK

- Po RESETu se jede z HSI (viz. RCC)
- Startup volá SystemInit()
  - Na konci se volá SetSysClock()
- Podle nastavení maker pro SYSCLK se zavolá funkce pro přednastavené "hodiny"
  - Zkusí zapnout HSE
  - Pokud se nenastaví HSERDY, použije HSI
- Defaultně pro STM32F103RB je 72MHz
  - Viz. Keil.STM32F1xx\_DFP.pdsc
  - Odkazováno z vlastností "Target"



```
419 static void SetSysClock(void)
420 {
421     #ifdef SYSCLK_FREQ_HSE
422         SetSysClockToHSE();
423     #elif defined SYSCLK_FREQ_24MHz
424         SetSysClockTo24();
425     #elif defined SYSCLK_FREQ_36MHz
426         SetSysClockTo36();
427     #elif defined SYSCLK_FREQ_48MHz
428         SetSysClockTo48();
429     #elif defined SYSCLK_FREQ_56MHz
430         SetSysClockTo56();
431     #elif defined SYSCLK_FREQ_72MHz
432         SetSysClockTo72();
433     #endif
434
435     /* If none of the define above is enabled, the HSI is us
436        source (default after reset) */
437 }
```



```
<!-- ***** Device 'STM32F103RB' ***** -->
<device Dname="STM32F103RB">
  <processor Dfpu="0" Dmpu="0" Dendian="Little-endian" DcLock="72000000"/>
  <compile header="Device\Include\stm32f10x.h" define="STM32F10X_MD"/>
```

# Využití knihovných funkcí pro SYSCLK

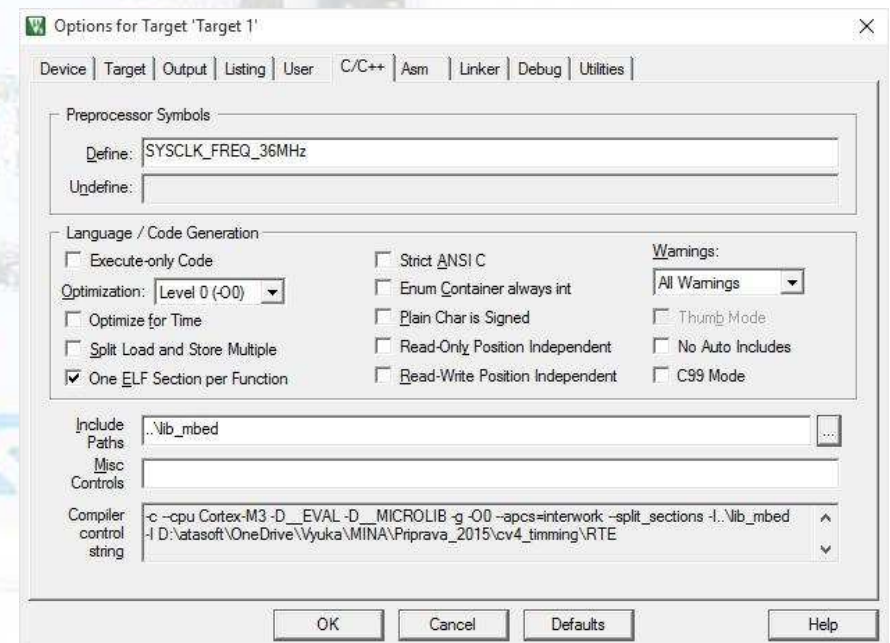
- Ověření aktuálně nastavené SystemCoreClock
  - Mělo by vypsát na terminál 72000000 (72M)
  - Nebo breakpoint za CoreClockUpdate() a vypsát proměnnou
    - Např. v okně Watch
- Změna symbolu preprocesoru na SYSCLK\_FREQ\_36MHz
  - Použije se "jiná" funkce v SetSystemClock
  - Ověřit výpisem na terminál

```
#include "mbed.h"
#include "uart.h"

int main(void)
{
    SystemCoreClockUpdate();
    Uart2Init(38400);

    printf("CoreClock: %d\r\n", SystemCoreClock);

    ...
}
```



# Vlastní nastavení hodin – HSI pro 64MHz

- Zapnout pro jistotu HSI a čekat na ready
- Uvést CFGR registr do stavu jako po RESETu
  - Nastaven HSI jako zdroj hodin
    - Aby se příp. dále mohly jiné zdroje hodin "vypnout"
  - Pozor na nutnost vypnout PLL, aby se mohlo vypnout HSE, pokud jej používá
- Uvést CR do RESET-state
  - Běží pouze HSI
  - Defaultní HSITRIM hodnota
- Nastavit děličky pro sběrnice
  - A nastavit násobení PLL \*16
  - Pozor, HSI má před PLL ještě /2
- Povolit PLL násobičku a čekat na funkčnost
- Nutno nastavit FLASH latency wait states
  - Viz. RM 3.3.3 - Flash access control register
  - 000 Zero WS,  $0 < \text{SYSCLK} \leq 24 \text{ MHz}$
  - 001 One WS,  $24 \text{ MHz} < \text{SYSCLK} \leq 48 \text{ MHz}$
  - 010 Two WS,  $48 \text{ MHz} < \text{SYSCLK} \leq 72 \text{ MHz}$
- Možno nastavit PLL jako zdroj hodin
- Aktualizovat SystemCoreClock

```
void SetClockHSI64MHz(void)
{
    RCC->CR |= RCC_CR_HSION;           // enable HSI for safe
    while(!(RCC->CR & RCC_CR_HSIRDY)) // wait for ready
        ;

    RCC->CFGR &= ~RCC_CFGR_SW;          // HSI as clock (value 00)
    RCC->CR &= ~RCC_CR_PLLON;           // PLL Off (if from HSE)
    RCC->CR = 0x83;                     // reset state = HSI On, HSITRIM = 16

    RCC->CFGR = 0x00;                   // reset state = HSI as clock
                                        // PLLSRC from HSI

    RCC->CFGR |= RCC_CFGR_PLLMULL16;    // (8MHz / 2) * 16 = 64

    RCC->CFGR |= RCC_CFGR_HPRE_DIV1;    // max. 72MHz, here 64
    RCC->CFGR |= RCC_CFGR_PPRE1_DIV2;   // max. 36MHz, here 32
    RCC->CFGR |= RCC_CFGR_PPRE2_DIV1;   // max. 72MHz, here 64

    RCC->CFGR |= RCC_CFGR_ADCPRE_DIV8; // slowest, ie. /8

    RCC->CR |= RCC_CR_PLLON;            // enable/start PLL
    while(!(RCC->CR & RCC_CR_PLLRDY)) // wait for ready
        ;

    FLASH->ACR &= ~FLASH_ACR_LATENCY; // RM 3.3.3, Table 8
    FLASH->ACR |= FLASH_ACR_LATENCY_2; // 48MHz < SYSCLK ≤ 72MHz

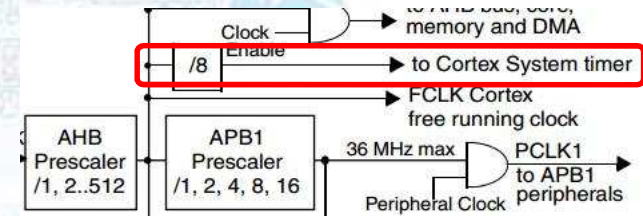
    RCC->CFGR |= RCC_CFGR_SW_PLL;       // use PLL (with HSI)

    SystemCoreClockUpdate();           // recalculate
}
```



# Systick a přerušení

- Systick je časovač v Cortex jádře
  - Existuje na všech procesorech Cortex-M
  - Na C-M3 je připojen k přímo AHB dělený 8
  - Samotný časovač je realizován jako 24-bitový count-down (viz. PM, kap. 4.5)
  - Má automatický reload z registru STK\_LOAD (PM 4.5.2)
  - Řízení registrem STK\_CTRL (PM 4.5.1)
    - Povolení běhu a povolení přerušení
- Generované přerušení řízení NVIC řadičem
  - Stejně jako ostatní přerušení, viz. CV5
  - Pro inicializaci a nastavení vhodné volat Core-CMSIS knihovní funkci SysTick\_Config
    - Nastaví LOAD registr, nastaví NVIC a povolí přerušení
    - Viz. tělo funkce v **core\_cm3.h**
- Jméno funkce obsluhy přerušení je uvedeno v kódu startup s modifikátory EXPORT a [WEAK]
  - Soubor startup\_stf32f10x\_md.s
  - Takže je možno napsat vlastní funkci se stejným jménem, která překryje "WEAK"
  - Viz. 12.28 EXPORT or GLOBAL - [http://www.keil.com/support/man/docs/armasm/armasm\\_dom1361290009343.htm](http://www.keil.com/support/man/docs/armasm/armasm_dom1361290009343.htm)



**WEAK**

`symbol` is only imported into other sources if no other source exports an alternative `symbol`. If `[WEAK]` is used without `symbol`, all exported symbols are weak.

# Systick a časování čekání

- Obsluha přerušení musí být **void SysTick\_Handler(void)**
- Globální proměnná pro počítání uplynulých **ms** musí být **volatile**
- Inicializace řadiče přerušení a nastavení periody opakování dělá knihovní funkce **SysTick\_Config**
  - POZOR, nekontroluje se, zda funkce obsluhy přerušení existuje !!
- Počítání ms možno např. využít k přesnému čekání

```
...
volatile uint32_t _ticks = 0;

void SysTick_Handler(void)
{
    _ticks++;
}

void cekejMs(uint32_t ms)
{
    ms += _ticks;
    while(_ticks < ms)
    ;
}

int main(void)
{
    ...
    SystemCoreClockUpdate();
    SysTick_Config(SystemCoreClock / 1000); // Every 1 ms

    ...
    while(1)
    {
        ...
        //cekejFor(1000000);
        cekejMs(100);
        ...
    }
}
```

```
54 ; Vector Table Mapped to Address 0 at Reset
55 ;
56 AREA RESET, DATA, READONLY
57 EXPORT __Vectors
58 EXPORT __Vectors_End
59 EXPORT __Vectors_Size
60
61 __Vectors DCD __initial_sp ; Top of Stack
62 DCD Reset_Handler ; Reset Handler
63 DCD NMI_Handler ; NMI Handler
```

```
73 DCD DebugMon_Handler ; Debug Monitor Handler
74 DCD 0 ; Reserved
75 DCD PendSV_Handler ; PendSV Handler
76 DCD SysTick_Handler ; SysTick Handler
77
78 ; External Interrupts
79 DCD WWDG_IRQHandler ; Window Watchdog
80 DCD PVD_IRQHandler ; PVD through EXTI Line detect
```

```
173 PendSV_Handler ENDP
174 PROC
175 EXPORT PendSV_Handler [WEAK]
176 B .
177 ENDP
178 SysTick_Handler PROC
179 EXPORT SysTick_Handler [WEAK]
180 B .
181 ENDP
182 Default_Handler PROC
183
184
185 EXPORT WWDG_IRQHandler [WEAK]
```

# Program cvičení

1. Úvod, rozdělení kitů, prostředí Keil, nahrání programu
2. Debug, bitové operace, GPIO
3. UART, připojení k PC, využití stdio knihovny
4. Časování, hodiny, SysTick
- 5. Časovač, přerušení, NVIC**
6. SPI a připojení LCD
7. A/D převodník, připojení přes IIC
8. DMA
9. RTOS
10. Samostatná práce
11. Dtto
12. Dtto
13. Dokončení a odevzdání samostatné práce



# Z domácí přípravy IV

- Umíme vytvořit obsluhu přerušení pro SysTick a nastavit ho
- Máme rozdělený kód mezi funkci main a knihovnu
  - V knihovně je kód pro sériovou komunikaci po UART2 pomocí funkcí z stdio
  - V knihovně jsou funkce pro využití HW mbed-shield
    - Tlačítka joysticku
    - RGB LED ve statickém režimu (ON, OFF)
- Umíme vytvořit aplikaci využívající "naši" knihovnou
- Máme aplikaci pro RGB a Joystick přes UART využívající časování pomocí SysTick

# NVIC – Nested Vector Interrupt Controller

- Registry a funkce v PM (!), detaily na přednášce
  - Kap. 2.3.2 – typy vyjímek, "klasické" IRQ je jen jednou z nich, celkem 68 možných
    - Pending se nuluje automaticky
      - Při vyvolání obsluhy INTu
- Kap. 4.3 – přehled registrů
  - Bitově orientované – vždy trojice registrů po 32b, přístup jako pole
    - Set enable, Clear enable, ...
  - Priority – 8b hodnota pro každý zdroj IRQ, tj. 21 registrů IPR
- Z praktického hlediska – řízení konkrétního IRQ zajistí funkce z CMSIS Core
  - Viz. PM kap. 4.3.10
  - Příslušné číslo bitu:
    - Nejlépe najít v IRQn\_Type
      - Viz. stm32f10x.h
    - Procesor je STM32F10X\_MD
      - (mid. density)

Table 41. Mapping of interrupts to the interrupt variables<sup>(1)</sup>

Interrupts	CMSIS array elements <sup>(1)</sup>				
	Set-enable	Clear-enable	Set-pending	Clear-pending	Active Bit
0-31	ISER[0]	ICER[0]	ISPR[0]	ICPR[0]	IABR[0]
32-63	ISER[1]	ICER[1]	ISPR[1]	ICPR[1]	IABR[1]
64-67	ISER[2]	ICER[2]	ISPR[2]	ICPR[2]	IABR[2]

1. Each array element corresponds to a single NVIC register, for example the element ICER[1] corresponds to the ICER1 register.

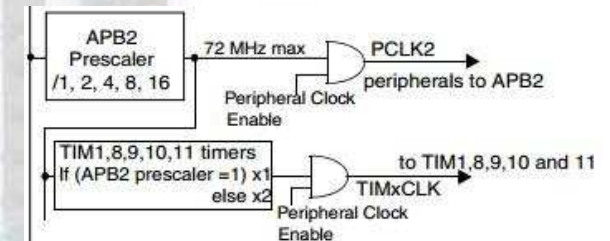
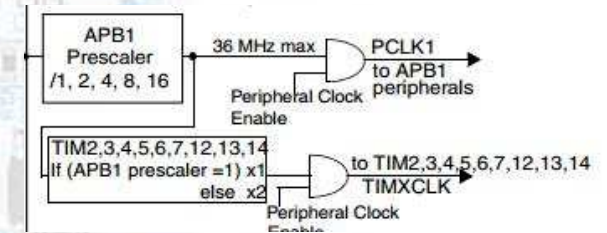
Table 43. CMSIS functions for NVIC control

CMSIS interrupt control function	Description
void NVIC_SetPriorityGrouping(uint32_t priority_grouping)	Set the priority grouping
void NVIC_EnableIRQ(IRQn_t IRQn)	Enable IRQn
void NVIC_DisableIRQ(IRQn_t IRQn)	Disable IRQn
uint32_t NVIC_GetPendingIRQ(IRQn_t IRQn)	Return true (IRQ-Number) if IRQn is pending
void NVIC_SetPendingIRQ(IRQn_t IRQn)	Set IRQn pending
void NVIC_ClearPendingIRQ(IRQn_t IRQn)	Clear IRQn pending status
uint32_t NVIC_GetActive(IRQn_t IRQn)	Return the IRQ number of the active interrupt
void NVIC_SetPriority(IRQn_t IRQn, uint32_t priority)	Set priority for IRQn
uint32_t NVIC_GetPriority(IRQn_t IRQn)	Read priority of IRQn
void NVIC_SystemReset(void)	Reset the system



# Časovače - Timers

- "Obyčejné" časovače – 2 – 14 (kap. 15 a 16 RM)
  - Všechny časovače jsou nezávislé, mají až 4 "kanály"
  - Zdrojem hodin je APB1, pokud PPRE1 == 1, pak x1, jinak x2
  - Umožňuje čítání vnitřního taktu nebo externích pulsů
  - 16b obousměrný čítač s možností 16b předděličky
  - Umožňuje funkce pomocí "kanálů"
    - Input Capture
    - Output Compare
    - Generování PWM (zarovnané na hranu i na střed pulsu)
    - Výstup typu One-pulse
    - Přerušení
      - Přetečení/podtečení
      - Input nebo output událost
  - Časovače se mohou "řetězit" = slave nebo master
- "Advanced" časovače – 1 a 8 (kap. 14 RM)
  - Zdrojem hodin je APB2, pokud PPRE2 == 1, pak x1, jinak x2
  - Navíc mají proti "obyčejným":
    - Complementary outputs with programmable dead-time
    - Repetition counter to update the timer registers only after a given number of cycles of the counter.
- Procesor 103RB má pouze
  - TIM1 a TIM2, TIM3 a TIM4 (každý se 4 kanály)
  - Viz. Kap 2.1 DS





# Režimy čítače

- Čítání dle směru - mez v ARR
  - Viz. RM 15.3.2
  - Upcounting mode
  - Downcounting mode
  - Center-aligned mode
- POZOR, čítá 0 .. N (ARR)
- Capture/compare block
  - RM 15.3.4

Figure 126. Capture/compare channel 1 main circuit

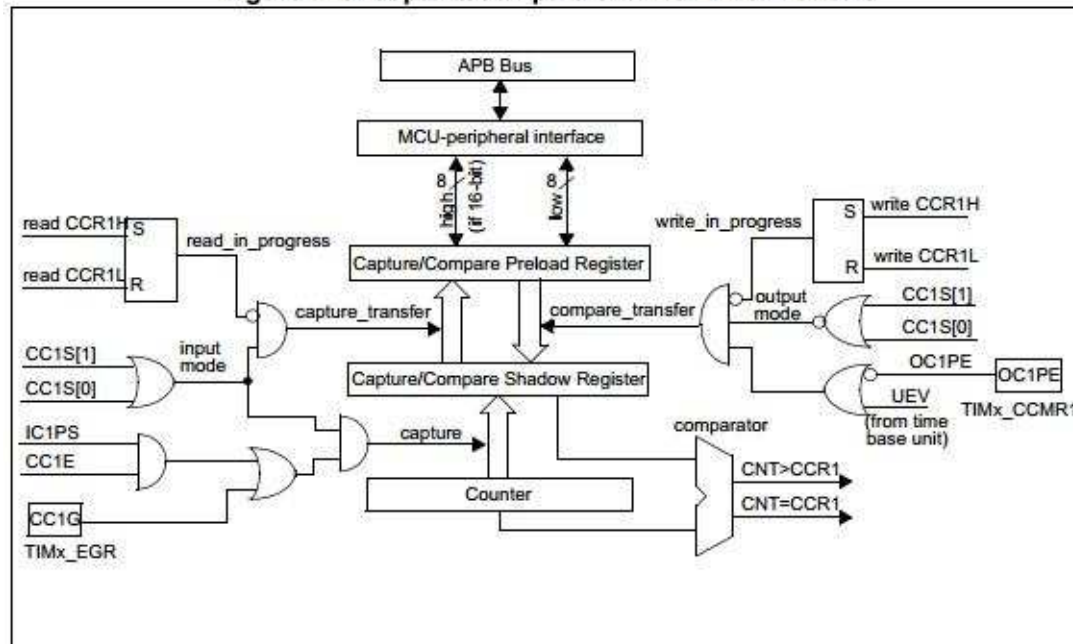


Figure 103. Counter timing diagram, internal clock divided by 1

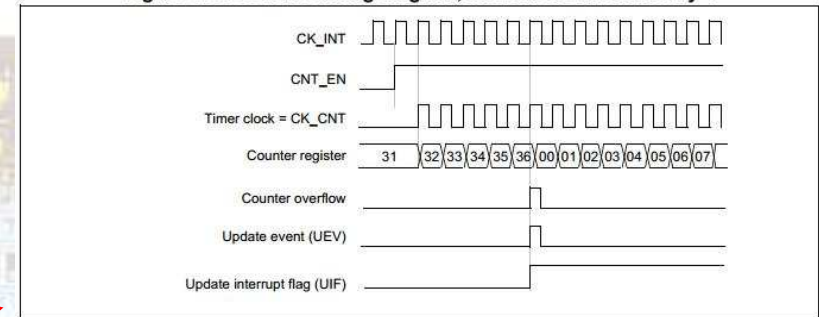


Figure 109. Counter timing diagram, internal clock divided by 1

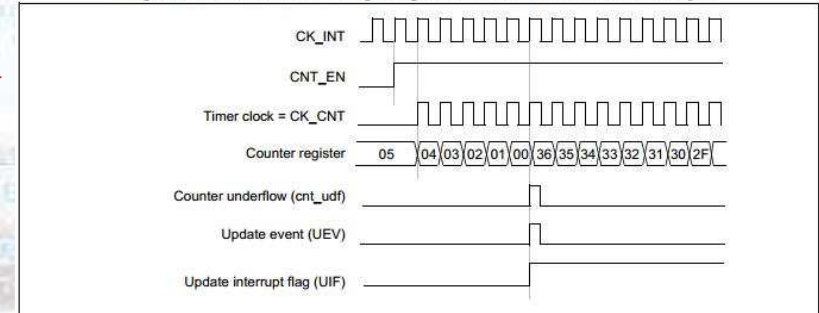
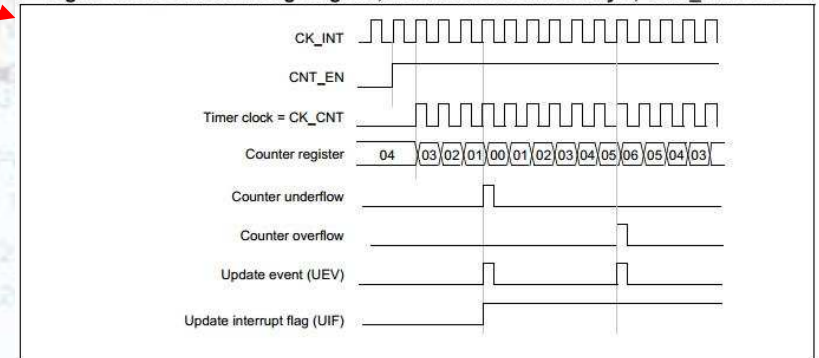


Figure 114. Counter timing diagram, internal clock divided by 1, TIMx\_ARR=0x6



# PWM – edge/center aligned

- Při změně více PWM výstupů z jednoho časovače lze nastavit synchronizaci
  - Edge = pulsy začínají ve stejný okamžik
  - Center = středy pulsů stejné
    - Pro omezení rušení
    - Poloviční perioda PWM

Figure 130. Edge-aligned PWM waveforms (ARR=8)

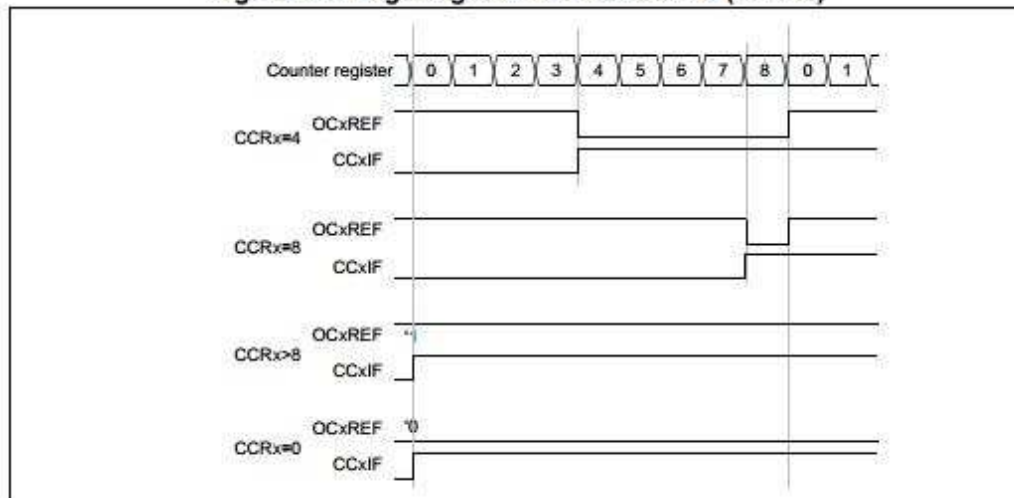
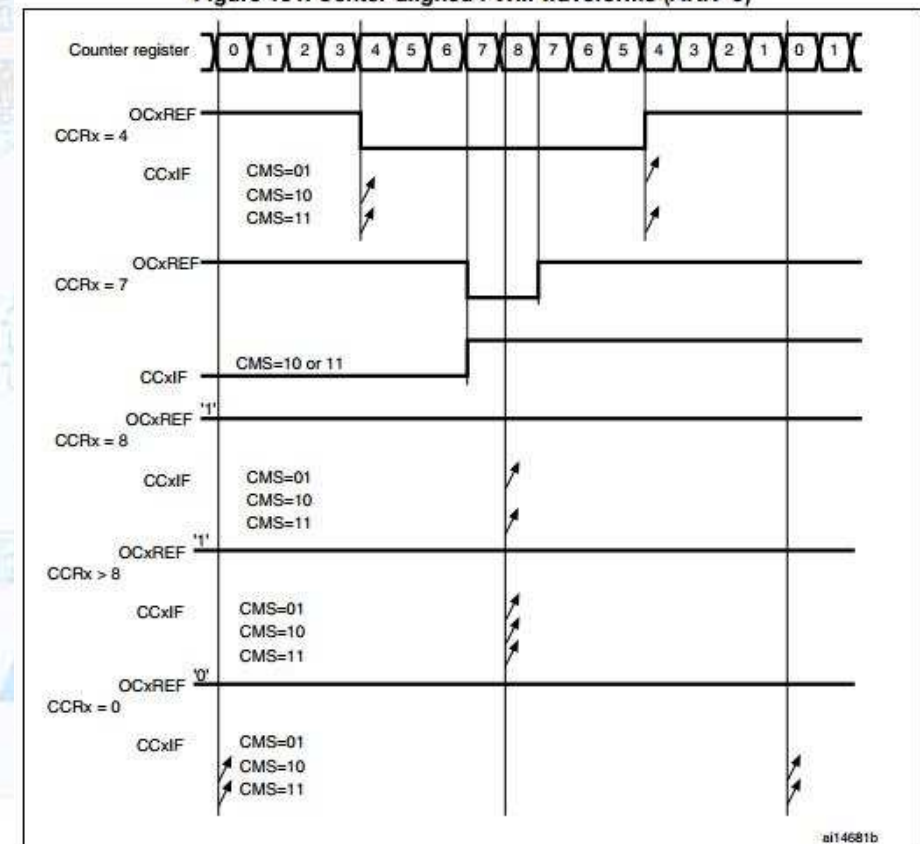


Figure 131. Center-aligned PWM waveforms (ARR=8)



# Registry CR1 a CR2 - control register 1, 2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- CKD – předdělení čítače, umožňuje častější vzorkování
- ARPE – povoluje preload ARR (jen pro spec. případy)
- CMS – Center aligned mode selection
  - 00 – edge-aligned – nejčastější
- **DIR** – direction – 0 = UP, 1 = DOWN
- OPM – One-pulse mode
- **URS** – Update request source
  - 0 = "Update interrupt" (UEV) generován ze všech událostí
  - 1 = "Uint" generován jen při přetečení/podtečení
- UDIS – Update disable
  - 1 = zakázání generování UEV události
- **CEN** – Counter enable – 0 = disable, 1 = enable
  - Automaticky vynulován v režimu one-pulse
- Registr CR2 ovládá případné DMA a také Master režimy



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

• Registr **DIER** = DMA/Interrupt enable register

- CCxDE – DMA request enable od CapCom kanálů
- TDE, UDE – DMA req. enable od Triggeru a Události
- CCxIE – Interrupt enable od CapCom kanálů
- **UIE** – Update Interrupt enable
  - Vzniká při přetečení

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Reserved		TIF	Res	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

• Registr **SR** = Status Register

- CCxOF – v Capture režimu indikuje, že hodnota čítače byla zachycena do TIMx\_CCRx registru, aniž by předtím byl vynulován příslušný CCxIF příznak
  - Nastaven HW do 1, vynulovat nutno SW
- TIF – Trigger Interrupt – pro událost Trigger ve Slave módu
- CCxIF - nastaven HW, nulován SW
  - V Compare režimu indikuje, že došlo ke shodě hodnot v CNT a CCx registru
  - V Capture režimu indikuje, že hodnota čítače byla vložena do TIMx\_CCRx registru
- **UIF** – příznak přerušení, nastaven do 1
  - Podle nastavení UDIS jen přetečení/podtečení, nebo i další události

# Využití přetečení časovače pollingem

- Pro init. TIM3 je nutno:
  - Povolit v APBxENR a Reset
  - Nastavit v CR1 alespoň CEN, dále DIR=0 => UP
  - Nastavit předděličku PSC
  - Nastavit ARR
- Přetečení indikuje příznak UIF
  - Nezapomenout "shodit" na 0
- Pro zjištění "hodin" pro časovač nutno:
  - Pro TIM1 (a 8) zjistit PPRE2 z RCC\_CFGR
  - Pro ostatní TIM se bere PPRE1 (tj. takt APB1)
  - Izolovat ty 3 bity a posunout na "spodní"
    - 0xx: HCLK not divided
    - 100: HCLK divided by 2
    - 101: HCLK divided by 4
    - 110: HCLK divided by 8
    - 111: HCLK divided by 16
- Pokud je APBx == 1, pak pro TIMx je clock x1
- Jinak je clock x2 (viz. RCC clocks)

```
...
int main(void)
{
    SystemCoreClockUpdate();
    Uart2Init(38400); puts("Start APP\r\n");

    { // TIM3 init block
        uint32_t apb1 = GetTimerClock(3);

        if (!(RCC->APB1ENR & RCC_APB1ENR_TIM3EN))
        {
            RCC->APB1ENR |= RCC_APB1ENR_TIM3EN; // enable preiph.
            RCC->APB1RSTR |= RCC_APB1RSTR_TIM3RST;
            RCC->APB1RSTR &= ~RCC_APB1RSTR_TIM3RST;
        }

        TIM3->CR1 = TIM_CR1_URS; // request only from over/underflow
        TIM3->CR2 = 0;

        TIM3->PSC = apb1 / 100000 - 1; // 10us = 100kHz
        TIM3->ARR = 1000 - 1; // reload, here 10ms

        TIM3->CR1 |= TIM_CR1_CEN; // counter enable
    }

    while(1)
    {
        ...
        if (TIM3->SR & TIM_SR_UIF) // update flag set ?
        {
            TIM3->SR &= ~TIM_SR_UIF; // clear
            putchar('$'); // action
        }
    }
}
```

```
uint32_t GetTimerClock(int b) // get source Clock from APBx config
{
    uint32_t apbdiv = 0, timerClock = SystemCoreClock;

    switch(b)
    {
        case 1:
            apbdiv = RCC->CFGR & RCC_CFGR_PPRE2; // 0x00003800
            apbdiv >>= 11;
            break;
        case 2:
        case 3:
        case 4:
        case 5:
            apbdiv = RCC->CFGR & RCC_CFGR_PPRE1; // 0x00000700 = bits 10:8
            apbdiv >>= 8; // shift down
            break;
    }

    if ((apbdiv & 0x04) == 0) // highest bit from 3 == 0 ?
        timerClock = SystemCoreClock; // x1
    else
        timerClock = 2 * (SystemCoreClock >> ((apbdiv & 0x03) + 1));

    return timerClock;
}
```

# Přerušeni od časovače

- Obsluha přerušeni: **TIM3\_IRQHandler**
  - Viz. NVIC a startup
  - Nezapomenout "shodit" UIF příznak
    - Jinak se INT vyvolávají pořád !!!
  - Zde počítáme 100 přetečení po 10ms = 1s
- Povolení = nastavit bit UIE v registru DIER
- Nastavení NVIC
  - Funkce z CMSIS – core\_cm3.h

```
__STATIC_INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)
{
    NVIC->ISER[(((uint32_t)(int32_t)IRQn) >> 5UL)] =
        (uint32_t)(1UL << (((uint32_t)(int32_t)IRQn) & 0x1FUL));
}
```

- Ve enum-u IRQn\_Type najít "správný" název
  - Zde **TIM3\_IRQn = 29, /\*!< TIM3 global ... \*/**
- Po uplynutí 1s "akce"

```
volatile uint32_t sec1 = 0;
void TIM3_IRQHandler(void)
{
    static uint32_t cnt = 0;

    TIM3->SR &= ~TIM_SR_UIF; // clear req.

    cnt++;
    if (cnt >= 100)
    {
        cnt = 0;

        sec1 = 1;
    }
}

int main(void)
{
    ... Init UART, ...
    {
        ... Init TIM3 to polling-style

        TIM3->DIER |= TIM_DIER_UIE;
        NVIC_EnableIRQ(TIM3_IRQn);
    }

    while(1)
    {
        if (sec1) // 1s elapsed ?
        {
            sec1 = 0;
            putchar('#');
        }
    }
}
```



# Časovač jako zdroj PWM

- Speciálním režimem CapCom je generování PWM
  - Při shodě CCR registru s CNT dojde k jedné hraně PWM, při podtečení/přetečení k druhé hraně
- "Komparační" hodnota v registru CCRx

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
IC2F[3:0]				IC2PSC[1:0]				IC1F[3:0]			IC1PSC[1:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- **CCMR** registr řídí dvojici CapCom kanálů
  - **CCMR1** pro ch1 a ch2, **CCMR2** pro ch3 a ch4
  - PWM jsou výstupní režimy
  - CC2S – CapCom selection
    - 00 = konfigurace jako output
  - OC2M – Output Compare mode
    - 000 = blokováno
    - ...
    - 110 = PWM mode 1
      - Při čítání nahoru je výstup aktivní dokud  $CNT < CCRx$ , jinak neaktivní
      - Při čítání dolů je výstup neaktivní dokud  $CNT > CCRx$ , jinak aktivní
    - 111 = PWM mode 2 = inverzní k mode 1

# PWM řízení jasu RGB LED

- IO pro LED z RGB mají alternativní funkci z čítačů
  - LED R - D5 = PB4 – Timer 3, Channel 1
  - **LED G - D9 = PC7 – Timer 3, Channel 2**
  - LED B - D8 = PA9 – Timer 1, Channel 2
- Pozor, nastavit příslušné GPIO jako AlternativOUT

```
...
TIM3->CCMR1 &= ~TIM_CCMR1_OC2M;    // channel 2 mode
TIM3->CCMR1 |= TIM_CCMR1_OC2M_1 | TIM_CCMR1_OC2M_2;    // 110 = PWM mode 1

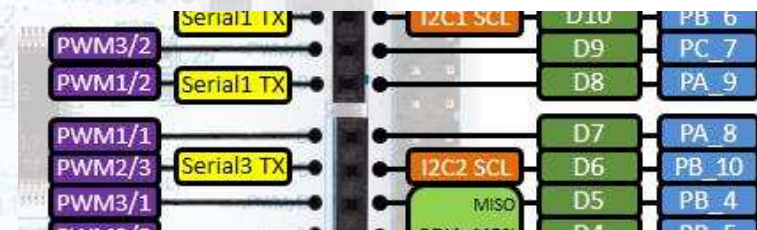
TIM3->CCER |= TIM_CCER_CC2E;

TIM3->CCR2 = TIM3->ARR / 2;    // 50% PWM

InitIOPort(GPIOC, 7, portALTOUT);    // PC7 = LED-G

if (!(RCC->APB2ENR & RCC_APB2ENR_AFIOEN))
{
    RCC->APB2ENR |= RCC_APB2ENR_AFIOEN;    // enable AFIO
    RCC->APB2RSTR |= RCC_APB2RSTR_AFIO_RST;
    RCC->APB2RSTR &= ~RCC_APB2RSTR_AFIO_RST;
}

AFIO->MAPR &= ~ AFIO_MAPR_TIM3_REMAP;
AFIO->MAPR |= AFIO_MAPR_TIM3_REMAP_FULLREMAP;    // 11: Full remap
...
```



# Program cvičení

1. Úvod, rozdělení kitů, prostředí Keil, nahrání programu
2. Debug, bitové operace, GPIO
3. UART, připojení k PC, využití stdio knihovny
4. Časování, hodiny, SysTick
5. Časovač, přerušení, NVIC
- 6. SPI a připojení LCD**
7. A/D převodník, připojení přes IIC
8. DMA
9. RTOS
10. Samostatná práce
11. Dtto
12. Dtto
13. Dokončení a odevzdání samostatné práce



# Z domácí přípravy V

- Umíme vytvořit obsluhu přerušení od časovače
- Využíváme funkce z knihovny
  - V knihovně je kód pro sériovou komunikaci po UART2 pomocí funkcí z stdio
  - V knihovně jsou funkce pro využití HW mbed-shield
    - Tlačítka joysticku
    - RGB LED ve statickém režimu (ON, OFF)
- Pomocí PWM umíme generovat různý jas na složkách RGB LED
  - Jedním z demo režimů je "dýchání" LED

# Zapojení LCD na kitu mbed

- LCD datasheet - NHD-C12832A1Z-FSW-FBW-3V3.pdf

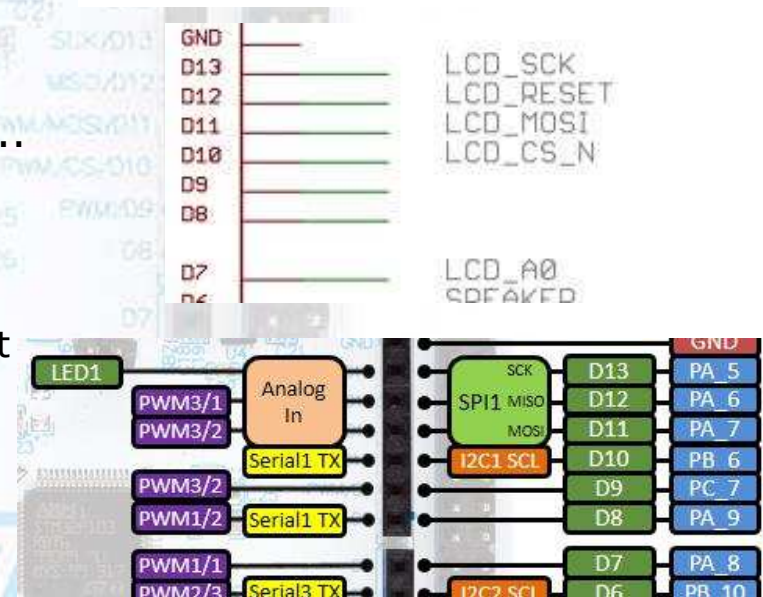
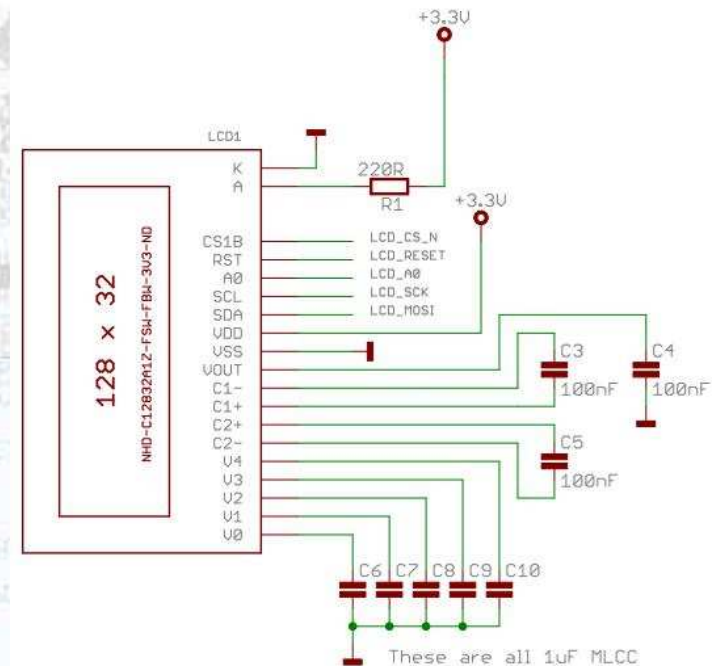
- Elektrické zapojení vývodů
- Základní zobrazení komunikace
- Základní tabulka povelů/funkcí

- Řadič – datasheet - st7565r.pdf

- Podrobný HW popis signálů, časování
  - Řadič "umí" paralelní i SPI komunikaci, na modulu je pouze SPI
- Podrobný popis funkcí, organizace paměti, ...

- Zapojení na modulu

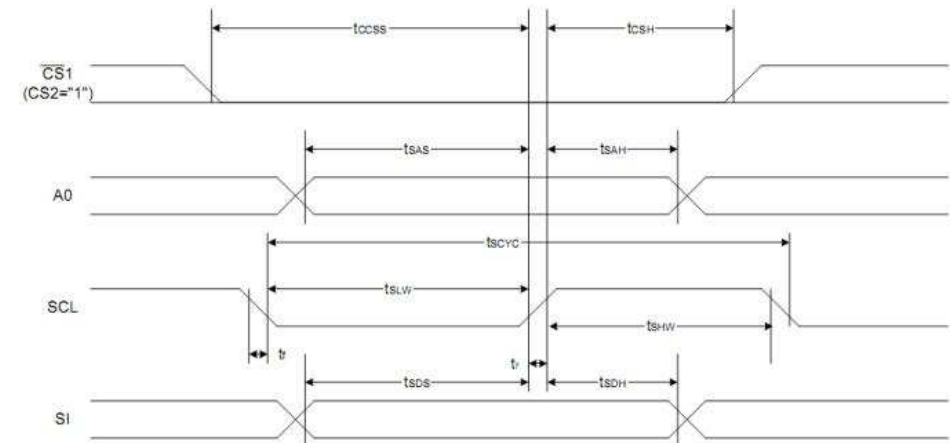
- Možno využít **SPI1 (PA5, PA7)** v režimu "output-only", tj. jen MOSI
  - Příp. možno signál "hodiny" a "data" generovat programově
- Signál CS (výběr) je na **PB6**
- Signál A0 (přepínač příkaz/data) na **PA8**
- Signál RESET na **PA6**



# Časování SPI komunikace

- SPI Clock period
  - min. 50ns => max. 20MHz
- Přesah dat SI okolo hrany SCL
  - Min. 20ns, resp. 10ns
- Nastavení CS
  - Min. 20ns před komunikací
  - Min. 40ns po
- Řízení hodin (signál CLK)
  - Data platná při vzestupné hraně
  - Na počátku log. 1
- Doba trvání RESETu
  - Min. 1us
- Vše viz. ST7565R datasheet
  - Str. 64-65, pro 3v3 VDD

The 4-line SPI Interface



Item	Signal	Symbol	Condition	Rating		Units
				Min.	Max.	
4-line SPI Clock Period		T <sub>scyc</sub>		50	—	ns
SCL "H" pulse width	SCL	T <sub>shw</sub>		25	—	
SCL "L" pulse width	SCL	T <sub>slw</sub>		25	—	
Address setup time	A0	T <sub>sas</sub>		20	—	
Address hold time	A0	T <sub>sah</sub>		10	—	
Data setup time	SI	T <sub>sds</sub>		20	—	
Data hold time	SI	T <sub>sdh</sub>		10	—	
CS-SCL time	CS	T <sub>css</sub>		20	—	
CS-SCL time	CS	T <sub>csh</sub>		40	—	

\*1 The input signal rise and fall time (tr, tf) are specified at 15 ns or less.

\*2 All timing is specified using 20% and 80% of VDD as the standard.

Reset Timing

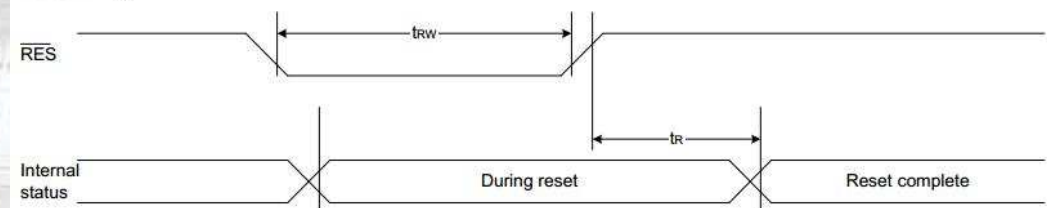


Figure 41

Table 36

(VDD = 3.3V, Ta = -30 to 85°C)

Item	Signal	Symbol	Condition	Rating			Units
				Min.	Typ.	Max.	
Reset time		tr		—	—	1.0	us
Reset "L" pulse width	/RES	trw		1.0	—	—	us



# SPI na STM32F103RB

- RM kap.25 – SPI Features
  - full duplex
  - 8-/16-bit transfer
  - master/slave
  - LSB-/MSB-first
  - programovatelná rychlost
  - podpora DMA
  - volitelná kompatibilita s I<sup>2</sup>S komunikací (audio aplikace)
- SPI v režimu "Master" – RM 25.3.3
  - Nutno nastavit všechny registry před povolením komunikace (SPE bit)
  - Příznak prázdného vysílacího bufferu TXE
- Přerušení – RM 25.3.11
  - TXIE, RXIE, ERRIE (některá chyba)

Figure 237. SPI block diagram

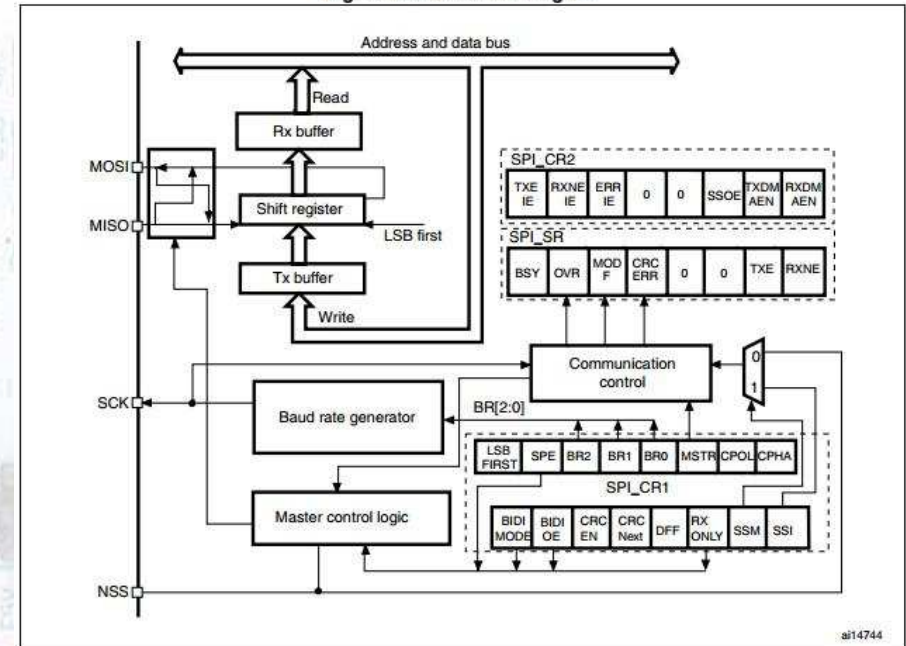
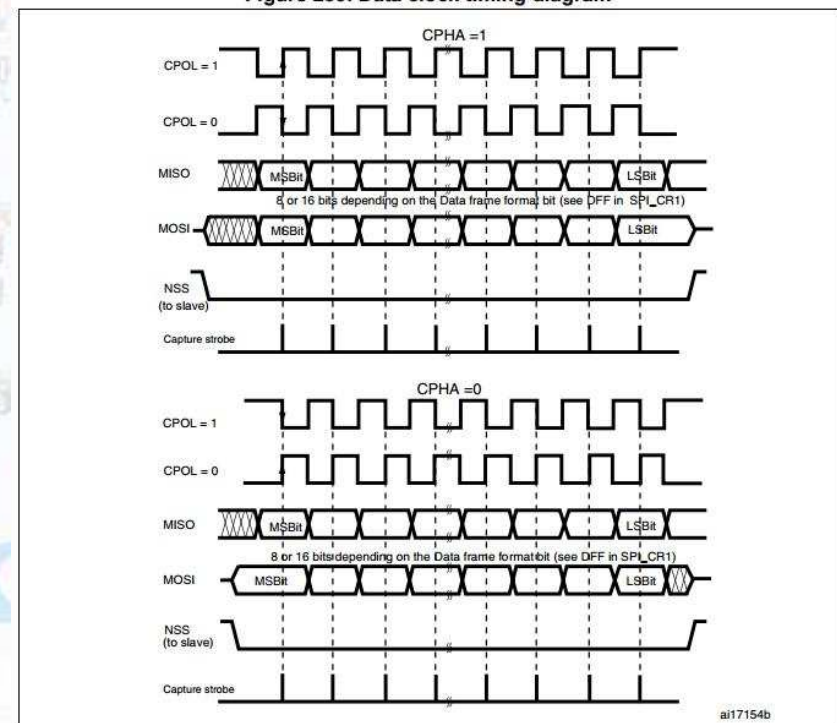


Figure 239. Data clock timing diagram



1. These timings are shown with the LSBFIRST bit reset in the SPI\_CR1 register.

# Časování SPI

- Příznak TXE
  - Data registr prázdný
- Příznak BSY
  - SPI přenos probíhá
- DMA

Figure 243. TXE/BSY in Slave transmit-only mode (BIDIMODE=0 and RXONLY=0) in the case of continuous transfers

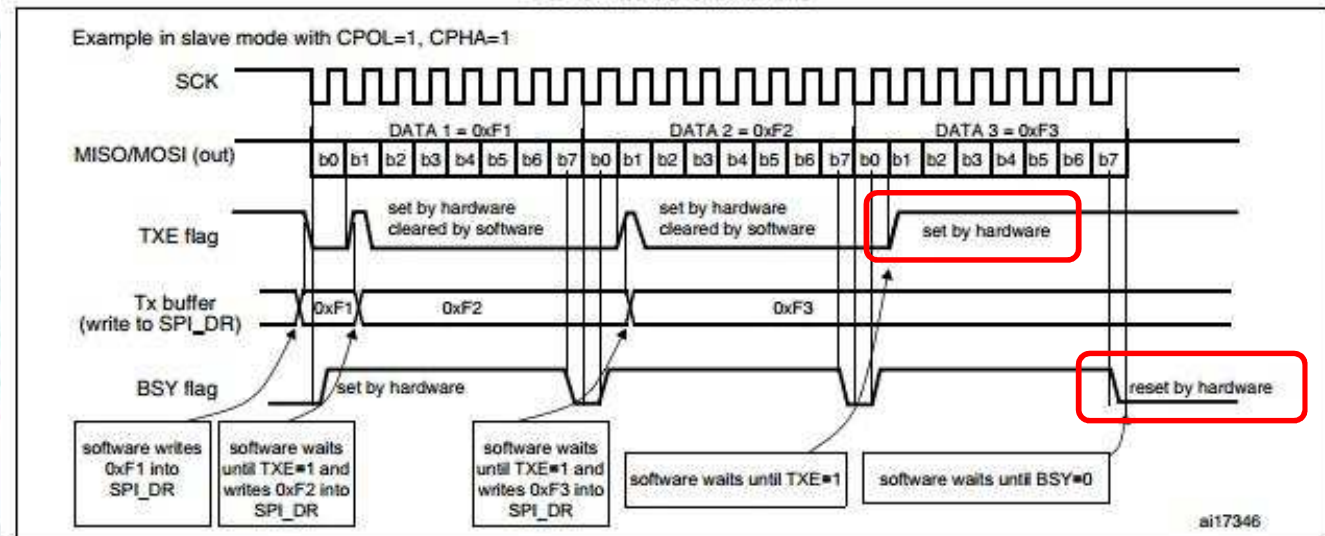
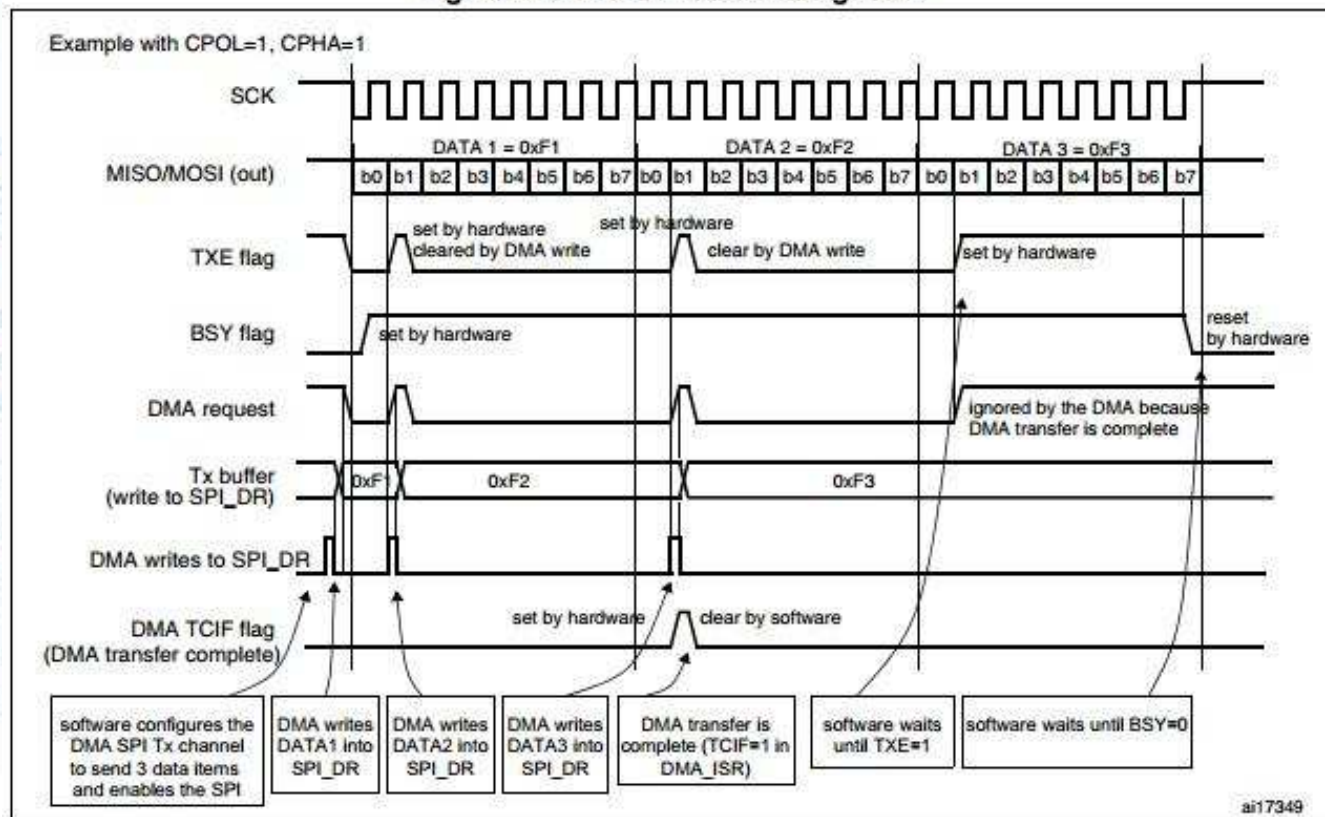


Figure 246. Transmission using DMA



# Registr SPI - CR1 (Control Reg.)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDI OE	CRC EN	CRC NEXT	DFF	RX ONLY	SSM	SSI	LSB FIRST	SPE	BR [2:0]			MSTR	CPOL	CPHA
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- **BIDIMODE** – 0 = 2-line unidirectional, 1 = 1-line bidirectional
- **BIDIOE** – jen pro BIDI režim
- **DFF** – Data Frame Format – 0 = 8-bit, 1 = 16-bit
- **SSM, SSI** – SW Slave Management
  - Experimentálně ověřeno, že musí být oba v log. 1
- **LSBFIRST** – Frame Format – 0 = MSB transmit first, 1 = LSB
- **SPE** – SPI enable
  - Veškeré změny nastavení provádět při SPE=0
- **BR** – Baud Rate – 000 =  $f_{clk}/2$ , 001 =  $f_{clk}/4$ , ...
  - SPI1 připojen na sběrnici APB2 (tj. typicky 72MHz)
- **MSTR** – Master selection – 1 = Master
- **CPOL** – Clock polarity
- **CPHA** – Clock phase
  - Viz. obrázek 239



# Další registry SPI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TXEIE	RXNEIE	ERRIE	Res.	Res.	SSOE	TXDMAEN	RXDMAEN
								rw	rw	rw			rw	rw	rw

- **CR2** – Control Register
  - **xxxIE** – povolení přerušení
  - **xxDMAEN** – povolení DMA pro daný směr přenosu

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								BSY	OVR	MODF	CRC ERR	UDR	CHSID E	TXE	RXNE
								r	r	r	rc_w0	r	r	r	r

- **SR** – Status Registr
  - **BSY** – Busy – SPI komunikuje nebo Tx bufer není prázdný
  - **OVR** – Overrun – přijata nová data před odebráním minulých
  - **MODF** – Mode Fault
  - **UDR** – Underrun Flag – kap. 25.4.7
  - **TXE** – Transmit Buffer Empty
    - Možno zapsat další data do registru **DR**
  - **RXNE** – Receive Buffer not Empty
    - Přijatá data v **DR** registru

# Komunikace s LCD (datasheet ST7565R)

- Signál **RESET** je aktivní v nule
  - Pro správnou inicializaci musí trvat min. 1us – viz. časování v DS, str. 65
  - Kompletní inicializační sekvence viz. DS, str. 51
- Po RESETu je nutno nastavit provozní registry
  - V knihovně použita sekvence ze zdrojových kódů MBED knihovny
    - [https://developer.mbed.org/users/dreschpe/code/C12832\\_lcd/docs/tip](https://developer.mbed.org/users/dreschpe/code/C12832_lcd/docs/tip)
  - Řadič "umí" 132x72, použitý LCD je 128x32 a jen SPI připojení
- Při komunikaci nutno nastavit signál **CS** na log. 0
- Signál **A0** vybírá, kam se data zapisují
  - 0 = registry
  - 1 = obrazová RAM
- Použitelné příkazy
  - Display address set – **0xBx**
    - Výběr řádku/stránky (osmice pixel-line)
    - V rozsahu 0-3
  - Column address set – **0x1x** následován **0x0x**
    - Výběr pozice v řádku, zápis 2 hodnot, nejprve vyšší 4 bity, pak nižší
    - V rozsahu 0-127
    - Vnitřní ukazatel aktivní pozice v paměti se inkrementuje při zápisu dat automaticky

**Table 16: Table of ST7565R Commands** (Note) \*: ignored data

Command	Command Code											Function
	A0	/RD	/WR	D7	D6	D5	D4	D3	D2	D1	D0	
(1) Display ON/OFF	0	1	0	1	0	1	0	1	1	1	0	LCD display ON/OFF 0: OFF, 1: ON
(2) Display start line set	0	1	0	0	1	Display start address						Sets the display RAM display start line address
(3) Page address set	0	1	0	1	0	1	1	Page address				Sets the display RAM page address
(4) Column address set upper bit Column address set lower bit	0	1	0	0	0	0	1	Most significant column address				Sets the most significant 4 bits of the display RAM column address.
				0	0	0	0	Least significant column address				Sets the least significant 4 bits of the display RAM column address.
(5) Status read	0	0	1	Status			0	0	0	0	0	Reads the status data
(6) Display data write	1	1	0	Write data								Writes to the display RAM
(7) Display data read	1	0	1	Read data								Reads from the display RAM
(8) ADC select	0	1	0	1	0	1	0	0	0	0	0	Sets the display RAM address SEG output correspondence 0: normal, 1: reverse
(9) Display normal/reverse	0	1	0	1	0	1	0	0	1	1	0	Sets the LCD display normal/ reverse 0: normal, 1: reverse
(10) Display all points ON/OFF	0	1	0	1	0	1	0	0	1	0	0	Display all points 0: normal display 1: all points ON
(11) LCD bias set	0	1	0	1	0	1	0	0	0	1	0	Sets the LCD drive voltage bias ratio 0: 1/9 bias, 1: 1/7 bias (ST7565R)
(12) Read-modify-write	0	1	0	1	1	1	0	0	0	0	0	Column address increment At write: +1 At read: 0
(13) End	0	1	0	1	1	1	0	1	1	1	0	Clear read/modify/write
(14) Reset	0	1	0	1	1	1	0	0	0	1	0	Internal reset
(15) Common output mode select	0	1	0	1	1	0	0	0	*	*	*	Select COM output scan direction 0: normal direction 1: reverse direction
(16) Power control set	0	1	0	0	0	1	0	1	Operating mode			Select internal power supply operating mode
(17) V <sub>0</sub> voltage regulator internal resistor ratio set	0	1	0	0	0	1	0	0	0	Resistor ratio		Select internal resistor ratio(Rb/Ra) mode
(18) Electronic volume mode set Electronic volume register set	0	1	0	1	0	0	0	0	0	0	1	Set the V <sub>0</sub> output voltage electronic volume register
				0	0	Electronic volume value						
(19) Static indicator ON/OFF Static indicator register set	0	1	0	1	0	1	0	1	1	0	0	0: OFF, 1: ON
				0	0	0	0	0	0	0	0	Set the flashing mode
(20) Booster ratio set	0	1	0	1	1	1	1	1	0	0	0	select booster ratio 00: 2x, 3x, 4x 01: 5x 11: 6x
(21) Power save	0	1	0									Display OFF and display all points ON compound command
(22) NOP	0	1	0	1	1	1	0	0	0	1	1	Command for non-operation
(23) Test	0	1	0	1	1	1	1	1	*	*	*	Command for IC test. Do not use this command



# Zdrojový kód knihovny pro LCD - části

- **MBED\_LCD.C** – obecné funkce pro práci s LCD

- Nezávislé na HW připojení, využití SPI, ...
- Obsahuje font pro znaky 8x8 – nutný **font\_8x8.h**
- Vnitřně "includeje" **MBED\_LCD\_HW.H**
- Pro aplikaci stačí využívat hlavičky v **MBED\_LCD.H**
- K dispozici funkce

```
void MBED_LCD_set_start_line(byte x); // x position on line (page) - 0-127
void MBED_LCD_set_page(byte p);      // select line (8px height) - 0-3
void MBED_LCD_init_all(void);        // HW init, LCD init-sequence
void MBED_LCD_write_data(byte val);  // write 1 byte of data
void MBED_LCD_write_bytes(byte *bptr, int count); // write many bytes
bool MBED_LCD_WriteCharXY(char c, byte col, byte row); // write single char
bool MBED_LCD_WriteStringXY(char *cp, byte col, byte row); // write string
void MBED_LCD_FillDisp(byte bFill);  // fill display area with "value"
```

- **MBED\_LCD\_HW.C** – HW závislé funkce

- Používají je pouze vnitřně v MBED\_LCD
- Podle přepínače při překladu může používat SPI nebo generovat signály "manuálně"
- Využívá HW funkce z knihovny - **MBED.H**

```
InitIOPort .... a nastavení portALTOUT, portOUTPUT
SET_IO_HIGH
SET_IO_LOW
```

# Zdrojový kód nastavení SPI a signálů

- Povolit a resetovat v RCC v APB2
- Nastavit SPI1->CR1
  - Dělení 72MHz / 4 = 18MHz
  - CPOL a CPHA sladit s průběhy periférie
- Nastavit výstupy SCK a MOSI
  - Režim Alternative output
- AFIO->MAPR
  - Nastavit SPI1 No remap
- Nastavit RESET, A0 a CS
  - Režim OUTPUT
  - RESET a CS jsou aktivní v 0
    - Tj. zapnout do log.1

## Bit 0 SPI1\_REMAP: SPI1 remapping

This bit is set and cleared by software. It controls the mapping of SPI1 NSS, SCK, MISO, MOSI alternate functions on the GPIO ports.

0: No remap (NSS/PA4, SCK/PA5, MISO/PA6, MOSI/PA7)

1: Remap (NSS/PA15, SCK/PB3, MISO/PB4, MOSI/PB5)

```
void MBED_LCD_init_hw(void)
{
#ifdef MBED_LCD_USE_SPI
if (!(RCC->APB2ENR & RCC_APB2ENR_SPI1EN)) // enable SPI
{
RCC->APB2ENR |= RCC_APB2ENR_SPI1EN;
RCC->APB2RSTR |= RCC_APB2RSTR_SPI1RST;
RCC->APB2RSTR &= ~RCC_APB2RSTR_SPI1RST;
}

SPI1->CR1 = SPI_CR1_BR_0; // 001 = clk/4 - from APB2 (72MHz)
SPI1->CR1 |= SPI_CR1_MSTR; // Master mode
SPI1->CR1 |= SPI_CR1_SSI | SPI_CR1_SSM; // SS control
SPI1->CR1 |= SPI_CR1_CPHA | SPI_CR1_CPOL; // see RM pg. 696/1128
SPI1->CR2 = 0; // nothing special

SPI1->CR1 |= SPI_CR1_SPE; // SPI enable

InitIOPort(GPIOA, 5, portALTOUT); // SPI1 SCK
InitIOPort(GPIOA, 7, portALTOUT); // SPI1 MOSI

if (!(RCC->APB2ENR & RCC_APB2ENR_AFIOEN))
{
RCC->APB2ENR |= RCC_APB2ENR_AFIOEN;
RCC->APB2RSTR |= RCC_APB2RSTR_AFIORST;
RCC->APB2RSTR &= ~RCC_APB2RSTR_AFIORST;
}

AFIO->MAPR &= ~ AFIO_MAPR_SPI1_REMAP;
#else
SET_IO_HIGH(GPIOA, 5);
InitIOPort(GPIOA, 5, portOUTPUT); // SCK

SET_IO_HIGH(GPIOA, 7);
InitIOPort(GPIOA, 7, portOUTPUT); // MOSI
#endif

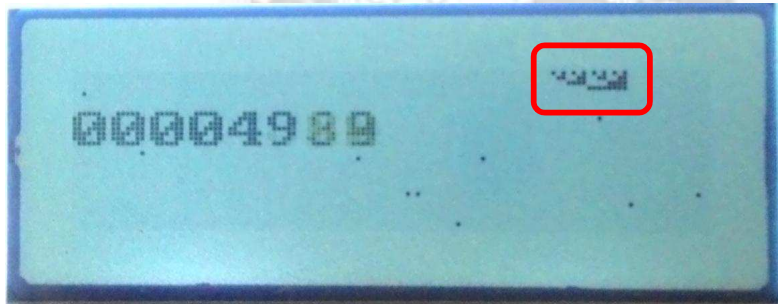
SET_IO_HIGH(GPIOA, 6);
InitIOPort(GPIOA, 6, portOUTPUT); // RESET

SET_IO_HIGH(GPIOA, 8);
InitIOPort(GPIOA, 8, portOUTPUT); // A0

SET_IO_HIGH(GPIOB, 6);
InitIOPort(GPIOB, 6, portOUTPUT); // CS
}
```

# Práce s knihovnou LCD

- V programu je třeba nejprve zavolat inicializaci LCD
- Zjistit, zda je bit D0 ve stránce horní nebo dolní možno např. vzorem "počítadlo"



- Pro výpis čísel použít stdio

```
#include "mbed_lcd.h"

...

int main(void)
{
    uint32_t last = 0;
    int i = 0;

    SystemCoreClockUpdate();
    SysTick_Config(SystemCoreClock / 1000); // interrupt every 1 ms

    MBED_LCD_init_all();

    MBED_LCD_set_page(0); // first 8-pix line
    MBED_LCD_set_start_line(96); // horizontal position

    for(i = 0; i < 16; i++) // generate "counter" pattern
        MBED_LCD_write_data(i);

    ...

    while(1)
    {
        char buff[20];

        sprintf(buff, "%08x", CUR_TICKS); // elapsed milliseconds
        MBED_LCD_WriteStringXY(buff, 0, 1);

        ...
    }
}
```



# Program cvičení

1. Úvod, rozdělení kitů, prostředí Keil, nahrání programu
2. Debug, bitové operace, GPIO
3. UART, připojení k PC, využití stdio knihovny
4. Časování, hodiny, SysTick
5. Časovač, přerušení, NVIC
6. SPI a připojení LCD
- 7. A/D převodník, připojení přes IIC**
8. DMA
9. RTOS
10. Samostatná práce
11. Dtto
12. Dtto
13. Dokončení a odevzdání samostatné práce

# Z domácí přípravy VI

- Umíme přidat do projektu "knihovnu" pro LCD displej
  - Umíme vypsát na LCD text
    - V knihovně je k dispozici funkce MBED\_LCD\_WriteCharXY
    - Využívá se základní font 8x8
- Umíme přijímat data (text) z UARTu (terminál v PC) a zobrazovat je na LCD
  - Speciální znakem možno "smazat displej" a psát o pozice 0,0
  - Zobrazování respektuje znaky CR a LF
  - Je možné přepnout na "negativní" zobrazení a zpět

# A/D převodník

- Viz. RM kap. 11
  - STM32F1xx má 2 A/D převodníky
  - Nastavitelná rychlost převodu
    - Dělení z AHB2
    - Vlastní dělička v konfigur. Registru
  - Převodníky mohou převádět střídavě pro rychlejší vzorkování
    - Podpořeno DMA apod.
  - Je k dispozici 16 externích kanálů multiplexovaně
    - Plus interní teplota a  $V_{REFINT}$  jako kanály 16 a 17
  - Převod se spouští událostí
    - Ručně nastavením bitu v registru
    - Automaticky podle časovače, ext. vstupu apod.
  - Procesor má vlastní referenci na VDD, u nás 3,3V



# Vlastní návrh kódu pro měření A/D

- Prozkoumejte možnosti a registry bloku A/D převodníku
- Výsledky měření vypisujte na LCD nebo terminálu
- Potenciometry připojeny na A0 a A1 vstupy
  - Tj. PA0 a PA1 na procesoru
  - Hodnoty potenciometrů – 10k $\Omega$ 
    - Výběr času vzorkování – viz. DS 5.3.18, equation 1
  - Potenciometry připojeny na 3,3V, tj. není třeba měřit referenci
- Uvažujte obě možnosti měření
  - Regular
    - Možno měřit "sekvenci jednoho kanálu" a ten měnit podle potřeby
    - Po každém měření nutno vyčíst data-registr
  - Injection
    - Lze až 4 kanály "najednou", výsledkem 4 data-registry
- Neřešte DMA, analog watchdog, nespojitý režim ani dual-mode
- Spouštějte převod manuálně = nastavením příslušného bitu
- Využijte pouze ADC1
- Přerušení využijte až tehdy, kdy bude fungovat manuální start převodu
- Výhledově zkuste vyčíst a spočítat teplotu
  - Konstanty do vzorečku viz. DS kap. 5.3.19
  - Není kalibrováno, takže reálná teplota může mít chybu až  $\pm 20^{\circ}\text{C}$

# Program cvičení

1. Úvod, rozdělení kitů, prostředí Keil, nahrání programu
2. Debug, bitové operace, GPIO
3. UART, připojení k PC, využití stdio knihovny
4. Časování, hodiny, SysTick
5. Časovač, přerušení, NVIC
6. SPI a připojení LCD
7. A/D převodník, připojení přes IIC
- 8. DMA**
9. RTOS
10. Samostatná práce
11. Dtto
12. Dtto
13. Dokončení a odevzdání samostatné práce

# Z domácí přípravy VII

- Umím používat A/D převodník
- Umím spojit data z A/D převodníku s LCD
  - Zobrazují se hodnoty z A/D jako číslo
  - Zobrazují se hodnoty z A/D ve formě bargrafu





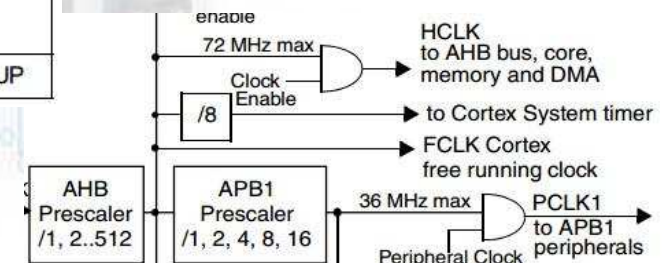
# DMA přenosy na STM32F1xx

- DMA = Direct Memory Access – RM kap. 13
  - Kromě přenosů dat z/do paměti umí i přenos z/do registru
    - Kombinace memory-to-memory, peripheral-to-memory, memory-to-peripheral
  - 7 kanálů (pro DMA1), (+ 5 pro DMA2 – to "naše" 103RB nemá)
  - Možno nastavovat prioritu přenosů na sběrnici
  - Přenosy byte (8b), half-word (16b) a word (32b)
  - Možno využít přerušení
    - Half-transfer (1/2 dat přenesena), Transfer Complete, Error
    - Viz. RM 13.3.6
  - Mapování periférií na DMA kanály – RM 13.3.7

Table 78. Summary of DMA1 requests for each channel

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC1	ADC1	-	-	-	-	-	-
SPI/I <sup>2</sup> S	-	SPI1_RX	SPI1_TX	SPI2/I2S2_RX	SPI2/I2S2_TX	-	-
USART	-	USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I <sup>2</sup> C	-	-	-	I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1		TIM1_CH1	-	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
TIM2	TIM2_CH3	TIM2_UP	-	-	TIM2_CH1	-	TIM2_CH2 TIM2_CH4
TIM3	-	TIM3_CH3	TIM3_CH4 TIM3_UP	-	-	TIM3_CH1 TIM3_TRIG	-
TIM4	TIM4_CH1	-	-	TIM4_CH2	TIM4_CH3	-	TIM4_UP

- Připojeno na AHB sběrnici
  - Nutno povolit bitem DMA1EN v AHBENR



## • **DMA\_ISR** – DMA interrupt status register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				TEIF7	HTIF7	TCIF7	GIF7	TEIF6	HTIF6	TCIF6	GIF6	TEIF5	HTIF5	TCIF5	GIF5
				r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TEIF4	HTIF4	TCIF4	GIF4	TEIF3	HTIF3	TCIF3	GIF3	TEIF2	HTIF2	TCIF2	GIF2	TEIF1	HTIF1	TCIF1	GIF1
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

- Bity pouze pro čtení
  - Všechny jsou HW nastavovány do log. 1
  - Nulování pomocí zápisu do příslušného bitu v DMA\_IFCR
- **TEIFx** – příznak chyby přenosu
- **HTIFx** – příznak přenesení ½ dat
- **TCIFx** – příznak dokončení přenosu
- **GIFx** – globální příznak přerušení

## • **DMA\_IFCR** – DMA interrupt flag clear register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				CTEIF7	CHTIF7	CTCIF7	CGIF7	CTEIF6	CHTIF6	CTCIF6	CGIF6	CTEIF5	CHTIF5	CTCIF5	CGIF5
				w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTEIF4	CHTIF4	CTCIF4	CGIF4	CTEIF3	CHTIF3	CTCIF3	CGIF3	CTEIF2	CHTIF2	CTCIF2	CGIF2	CTEIF1	CHTIF1	CTCIF1	CGIF1
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

- Bity pouze pro zápis
  - Zápisem log.1 se "shodí" příslušný příznak v DMA\_ISR
- **CTEIFx** – Clear TEIFx příznak
- **CHTIFx** – Clear HTIFx příznak
- **CTCIFx** – Clear TCIFx příznak
- **CGIFx** – Clear GIFx příznak

# DMA\_CCRx – DMA channel x configuration register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	MEM2 MEM	PL[1:0]		MSIZE[1:0]		PSIZE[1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- **MEM2MEM** – log. pro memory-to-memory přenos
- **PL** – priority level – 00 = Low, 01 = Medium, 10 = High, 11 = Very high
- **MSIZE** – Memory size – 00 = 8-bits, 01 = 16-bits, 10 = 32-bits
- **PSIZE** – Peripheral size – 00 = 8-bits, 01 = 16-bits, 10 = 32-bits
- **MINC** – Memory increment mode – povolení ++ adresového počítadla
- **PINC** – Peripheral increment mode – povolení ++ adresového počítadla
- **CIRC** – Circular mode – povolení
  - Po dokončení přenosu se znovu načte počítadlo přenosů a přenos běží dál
  - Vhodné pro kruhové buffery, zpracování z ADC apod.
- **DIR** – Data transfer direction
  - 0 = Read from peripheral
  - 1 = Read from memory
- **TEIE** – Transfer error interrupt enable
- **HTIE** – Half transfer interrupt enable
- **TCIE** – Transfer complete interrupt enable
- **EN** – Channel enable = spuštění DMA přenosu na "kanále"
  - Během přenosu jsou změny v registrech zakázány
  - Po skončení přenosu je nutno EN "vypnout" = nastavit log. 0



# DMA datové registry

- **DMA\_CNDTRx** - DMA channel x number of data register
  - 16b hodnota (max. 65535 bajtů)
  - Může být zapisován pouze při vypnutém "kanálu"
  - Během přenosu je R/O a indikuje zbývajících počet bajtů
    - Po skončení přenosu obsahuje 0
    - V cyklickém režimu se automaticky reloaduje (?)
- Adresové registry
  - Pokud je xSIZE != 00, nejsou dolní bity uvažované
    - xSIZE == 01 (half-word) – bit A0 ignorován
    - xSIZE == 10 (word) – bity A0, A1 ignorovány
  - **DMA\_CPARx** – DMA channel x peripheral address register
    - Ovlivněn PSIZE
  - **DMA\_CMARx** – DMA channel x memory address register
    - Ovlivněn MSIZE

# Přenos řetězce do TX registru UART2

- Základem kostra programu pro UART2 komunikaci
- Data k přenosu = řetězec
  - Const = obsah umístěn ve Flash
  - Možno vygenerovat "lorem-ipsium" generátorem
- Pro UART2\_TX musíme použít DMA kanál 7
- Povolit periférii v AHBENR
- Začít s vypnutým DMA
- Vynulovat všechny status-flagy
- Konfigurace (CCR)
  - 8-bitové přenosy (00)
  - Nejnižší priorita (00)
  - Memory-increment (vyčítání z paměti sekvenčně)
  - Read-from-memory
- Adresu textu přetypovat na číslo
- Pro adresu DR použít referenci
- Přenos spuštěn nastavením EN
- Test dokončení dle bitu TCIF7
- Po skončení nezapomenout
  - Zastavit přenos (EN = 0)
  - Vynulovat příznaky přenosu

```
...
int main(void)
{
    const char *textDemo = "Lorem ipsum ... \r\n"
        " ... \r\n";

    Uart2Init(38400);
    puts("Start APP\r");

    RCC->AHBENR |= RCC_AHBENR_DMA1EN;    // enable - see RCC clock schema

    DMA1_Channel7->CCR &= ~DMA_CCR7_EN;  // disable DMA7

    DMA1->IFCR |= (DMA_IFCR_CTEIF7 | DMA_IFCR_CHTIF7
        | DMA_IFCR_CTCIF7 | DMA_IFCR_CGIF7); // Clear status flags

    DMA1_Channel7->CCR = DMA_CCR7_MINC | DMA_CCR7_DIR;

    DMA1_Channel7->CNDTR = strlen(textDemo);
    DMA1_Channel7->CMAR = (uint32_t)textDemo;
    DMA1_Channel7->CPAR = (uint32_t)&(USART2->DR);

    USART2->CR3 |= USART_CR3_DMAT;

    DMA1_Channel7->CCR |= DMA_CCR7_EN; // start transfer
    while(!(DMA1->ISR & DMA_ISR_TCIF7)) // Wait for complete DMA transfer
    { // TODO: Test error flag DMA_ISR_TEIF7 in a real application!

    DMA1_Channel7->CCR &= ~DMA_CCR7_EN; // stop = finish transfer (!)

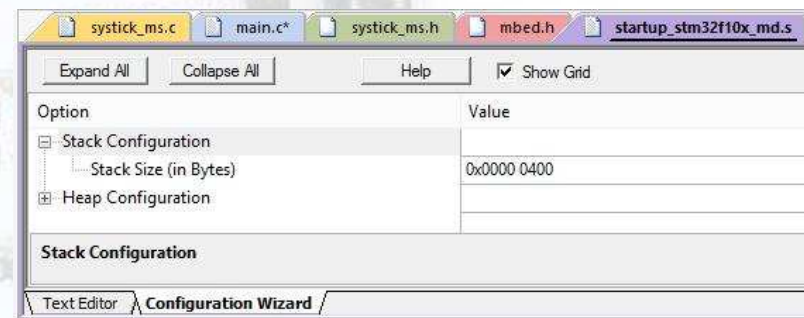
    DMA1->IFCR |= (DMA_IFCR_CTEIF7 | DMA_IFCR_CHTIF7
        | DMA_IFCR_CTCIF7 | DMA_IFCR_CGIF7); // Clear status flags

    while(1)
        ...
}
```

- Ověřte v debuggeru přenos dat během krokování

# Kopírování bloku paměti – "ručně"

- Procesor má k dispozici 20kB SRAM
  - Naalokovat 2x8kB blok (zdrojové a cílové pole) - např. `uint32_t`
- Změřit dobu přenosu pomocí for cyklu
  - Využít `systick` na 1ms a zjistit počet "tiků" pro 1000 kopírování
- Při `CoreClock=72MHz` očekávejte okolo 700ms
- POZOR – při alokaci polí ve funkci jsou tyto umístěny na zásobníku
  - Ve `startup_...s` nastaven na 0x400
    - `Stack_Size EQU 0x00000400`
- Při spuštění skončí ve smyčce v `HardFault_Handler`
  - = chyba při práci s pamětí (zásobníkem)
- Řešení
  - Pole globální – použije se `.bss` segment – viz `Listings\*.map` soubor
  - Zvětšit `stack` (nedoporučuji)

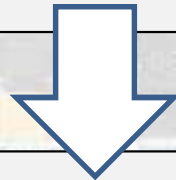




# Optimalizace kopírování bloku

- Místo použití přístupu pomocí pole lze využít přímo ukazatelů – s výhodou na 32b hodnoty

```
for(u = 0; u < DEMO_BUF_SIZE_32; u++)  
    dataDest[u] = dataSrc[u];
```



```
tickStart = CUR_TICKS;  
for(i = 0; i < 1000; i++)  
{  
    ptrSrc = dataSrc;  
    ptrDest = dataDest;  
    for(u = 0; u < DEMO_BUF_SIZE_32; u++)  
    {  
        *ptrDest++ = *ptrSrc++;  
    }  
}  
tickEnd = CUR_TICKS;  
  
printf("FOR PTR copy: %d ms\r\n", tickEnd - tickStart);
```

- Zrychlení na cca 485ms

# Kopírování bloku pomocí DMA

- Nastavit velikost přenášených dat na 32b slovo
- Režim MEM2MEM
  - Zdrojová adresa v CPAR, cílová v CMAR
- Očekávaný čas DMA
  - 172ms

```
RCC->AHBENR |= RCC_AHBENR_DMA1EN; // viz. clock schema u RCC
...
DMA1_Channel7->CCR &= ~DMA_CCR7_EN;

DMA1->IFCR |= (DMA_IFCR_CTEIF7 | DMA_IFCR_CHTIF7 | DMA_IFCR_CTCIF7 | DMA_IFCR_CGIF7);

DMA1_Channel7->CCR = DMA_CCR7_MEM2MEM
| DMA_CCR7_MSIZE_1 // 10 = 32b
| DMA_CCR7_PSIZE_1 // 10 = 32b
| DMA_CCR7_MINC     // dest increment
| DMA_CCR7_PINC     // src increment
| DMA_CCR7_DIR;     // PL = 00 - low priority

DMA1_Channel7->CMAR = (uint32_t)dataDest;
DMA1_Channel7->CPAR = (uint32_t)dataSrc;

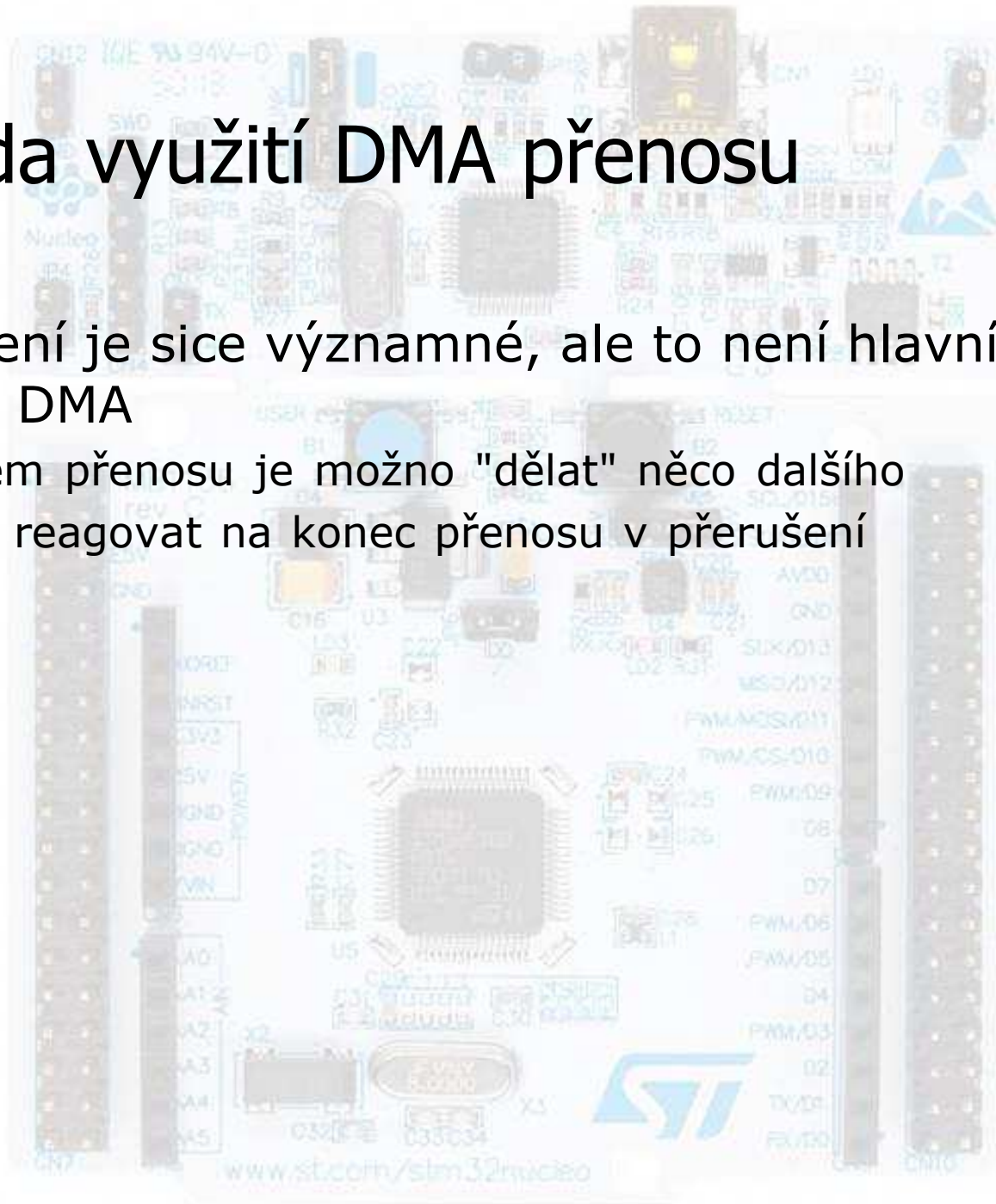
tickStart = CUR_TICKS;
for(i = 0; i < 1000; i++)
{
    DMA1_Channel7->CNDTR = DEMO_BUF_SIZE_32; // always set
    DMA1_Channel7->CCR |= DMA_CCR7_EN;       // start
    while(!(DMA1->ISR & DMA_ISR_TCIF7))      // Wait for complete
    ;

    DMA1_Channel7->CCR &= ~DMA_CCR7_EN;       // stop = unlock registers
    DMA1->IFCR |= (DMA_IFCR_CTEIF7 | DMA_IFCR_CHTIF7 | DMA_IFCR_CTCIF7 | DMA_IFCR_CGIF7);
}

tickEnd = CUR_TICKS;
printf("DMA copy: %d ms\r\n", tickEnd - tickStart);
```

# Výhoda využití DMA přenosu

- Zrychlení je sice významné, ale to není hlavní efekt využití DMA
  - Během přenosu je možno "dělat" něco dalšího
  - Příp. reagovat na konec přenosu v přerušení
  - ...





# Program cvičení

1. Úvod, rozdělení kitů, prostředí Keil, nahrání programu
2. Debug, bitové operace, GPIO
3. UART, připojení k PC, využití stdio knihovny
4. Časování, hodiny, SysTick
5. Časovač, přerušení, NVIC
6. SPI a připojení LCD
7. A/D převodník, připojení přes IIC
8. DMA
- 9. RTOS**
10. Samostatná práce
11. Dtto
12. Dtto
13. Dokončení a odevzdání samostatné práce

# Z domácí přípravy VIII

- Umím používat DMA přenosy
- Ověřil(a) jsem si efekty při využití DMA přenosů



# Real Time Operating System

- Program je vykonáván v procesech/vlákněch
- Proces typicky na něco čeká
  - Časový interval
  - Událost od HW nebo jiného vlákna (např. předaná data)
- Inicializace složitější
  - Nastavení HW komponent
  - Příprava "tasků" – priority, velikost zásobníku, ...
- Budeme používat FreeRTOS



# RTOS – příklad použití

1. Nový projekt v Keil pro Nucleo desku
2. Využít funkce pro mbed-desku a blikat LED
  - Připravit inicializaci pro všechny 3 LED
  - Připravit funkci typu "toggle-LED"
3. Do adresáře s projekty a knihovnou překopírovat
  - Z:\podklady\MINA\FreeRTOSV8.2.3
4. Další návod viz. "postup":
  - z:\podklady\MINA\freertos\_STM32F107\_kucharka.pdf
  - Pro naši F103 žádný rozdíl v základním postupu
  - Změny proti desce s F107
    - Nutno v "debug", kde nemáme TIM6, ale jen např. TIM4
    - LED nejsou na GPIOE, ale "různě" = upravit LEDFlashTask
  - Pokud nelze najít FreeRTOSConfig.h, je nutno přidat do "Include Paths" aktuální adresář (= ".")

# Program cvičení

1. Úvod, rozdělení kitů, prostředí Keil, nahrání programu
2. Debug, bitové operace, GPIO
3. UART, připojení k PC, využití stdio knihovny
4. Časování, hodiny, SysTick
5. Časovač, přerušení, NVIC
6. SPI a připojení LCD
7. A/D převodník, připojení přes IIC
8. DMA
9. RTOS
- 10. Samostatná práce**
11. Dtto
12. Dtto
13. Dokončení a odevzdání samostatné práce