

VKLÁDÁNÍ ASSEMBLEROVSKÝCH PROGRAMŮ DO JAZYKA C

V assembleru (JSA) se v současnosti realizují knihovní funkce např. pro jazyk C, ovladače periferií a též škodlivý software. Pro užitečné realizace se můžeme dostat do situace, kdy JSA nám umožní realizaci programu s potřebnými časovými parametry, umožní obsluhu nové periferie, kterou překladač ještě neumí obsloužit, případě umožní nahradit neefektivní řešení problému překladačem.

Existuje několik způsobů, jak vložit assemblerovskou rutinu do jazyka C.

- Vytvoření samostatného podprogramu v JSA volaného z jazyka C a připojeného linkerem při překladu.
- Vložení programu v JSA přímo do jazyka C tzv. inline assembler
- Vložení JSA do programu startup – není příliš šikovné

UKÁZKA PODPROGRAMU FILTRU FIR V JSA

Na začátku programu v jazyce symbolických adres (assembleru) musíme nadefinovat oblasti a proměnné, které se budou v podprogramu používat nebo jsou definovány v jazyce C

```
        AREA MOJEDATA, DATA, NOINIT, READWRITE
xn      SPACE 24                ; 24 Byte pro vzorky (16bitů) vstupního signálu
BAF     EQU 0x20000040          ; Adresa proměnné definované v JSA

        AREA asm_func, CODE, READONLY
GLOBAL BAF                    ; Proměnná definovaná v JSA na konkrétní adresu
EXTERN Yn                     ; Proměnná definovaná v jazyce C
EXPORT my_FIR                 ; my_FIR bude přilinkován k překladu jazyka C
```

UKÁZKA PODPROGRAMU FILTRU FIR V JSA

my_FIR

```
        LDR R3,=11                ; FIR stupně 11,  
; POSUN ZPOŽDĚNÝCH VZORKŮ, odpovídá realizaci operace  $z^{-1}$   
        LDR R1,=xn-20            ; Adresa proměnné xn-20 do registru R1  
opakzpet LDRH R2,[R1],#2          ; xn-10 v R2, R1=R1+2  
        STRH R2,[R1],# -4        ; Zápis na adresu xn-11, R1=R1-4  
        SUB R3,#1                ; Zmenšení počítadla  
        CBZ R3, opak             ; Skok pouze dopředu nebo IT NE  
        B opakzpet  
; ULOŽENÍ NOVÉHO VZORKU xn      ; R1 adresa xn-2, hodnota z jazyka C přichází v R0  
opak     STRH R0,[R1,#2]!        ; Preinkrementace adresy xn v registru R1  
; ZAHÁJENÍ VÝPOČTU SOUČTU SOUČINŮ  
        LDR R6, =11  
        LDR R2,=konstanty        ; Adresa koeficientu do registru R2  
opaknas  LDRH R4,[R1],#2          ; Vzorek xn do R4, + posun adresy  
        LDR R5,[R2],#4           ; Koeficient do R5, + posun adresy  
        MLA R3, R4, R5, R3        ; Součin a přičtení k R3  
        SUB R6, #1                ; Zmenšení počítadla  
        CBZ R6, opakvys  
        B opaknas
```

UKÁZKA PODPROGRAMU FILTRU FIR V JSA

```
SUB R6, #1           ; Zmenšení počítadla
CBZ R6, opakvys
B opaknas
Opakvys             ; Možnosti uložení výsledku
LDR R2,=BAF         ; Adresa proměnné BAF do R2
STR R3,[R2]         ; Uložení výsledku do proměnné BAF
LDR R2,=Yn          ; Adresa proměnné Yn do R2
STR R3,[R2]         ; Uložení výsledku do proměnné Yn
BX LR               ; Návrat do programu C, LR=návratová adresa

NEBO
Opakvys             ; Možnosti uložení výsledku
MOV R0, R3          ; Vrácení se hodnota z podprogramu je uložena v
                   ; R0 (viz. Systém ukládání hodnot do
                   ; podprogramu a z něj)
BX LR               ; Návrat do programu C, LR=návratová adresa

konstanty
DCD 1,2,3,4,5,6,5,4,3,2,1 ; Tabulka koeficientů filtru
END                 ; Cokoliv napsaného za názvem END bude překladač ignorovat
```

UKÁZKA JAZYKA C VOLAJÍCÍHO PODPROGRAM V JSA

```
#include <stdio.h>
#include "stm32F4xx.h"           // Device header
#include "Nastaveni_GPIO.c"
#include "SystemCoreClockSetHSI.c,,

extern int my_FIR(unsigned int x);

    unsigned int z, i=0, y, Yn;           // Definice globálních proměnných
extern unsigned int BAF, Yn;           // yn proměnná definovaná v jazyce C
// BAF proměnná definovaná v JSA

int main (void)
{
    SystemCoreClockSetHSI();
    SystemCoreClockUpdate();           // Get Core Clock Frequency
    Yn=z=0x12345;
    while(1) {                           // Nekonecna smycka
navest:    i=i+2;
            y=my_FIR(i);                 // Hodnota přenesená přes R0
            y=0x12345;
            z=BAF;                       // Hodnota uložená v JSA stejně jako Yn
    };
}
```

UKÁZKA JSA VLOŽENÉHO DO FUNKCE JAZYKA C

Jinou možností je vytvoření funkce s vloženým assemblerem. Aby překladač nehlásil chyby, je potřeba před vlastním asm souborem definovat použité registry.

```
int f(int x)
{
    int r0,r1;
    __asm
    {
        ADD r0, x, 1
        MOV r1,0x55
        EOR x, r0, r1
    }
    return x;
}
```

UKÁZKA VLOŽENÉHO INLINE ASSEMBLERU DO JAZYKA C

Další možností je zápis v tzv. inline assembleru.

```
#include <stdio.h>
#include "stm32F4xx.h"           // Device header

char b[20], x, y;
int main(void)
{
    x=1; y=3;
    __asm("ADD y, y, #1\n,,
          "MOV b[2], y\n");
    b[1]=y+1;
    return 0;
}

int main(void)
{
    x=1; y=3;
    int r0;
    __asm("ADD r0, y, #1\n"
          "MOV b[2], y\n");
    b[1]=y+1;
    return 0;
}
```

Řešení vlevo je o instrukci delší, než zápis `b[1]=y+1;`. Řešení vpravo je stejné se zápisem `b[1]=y+1;`

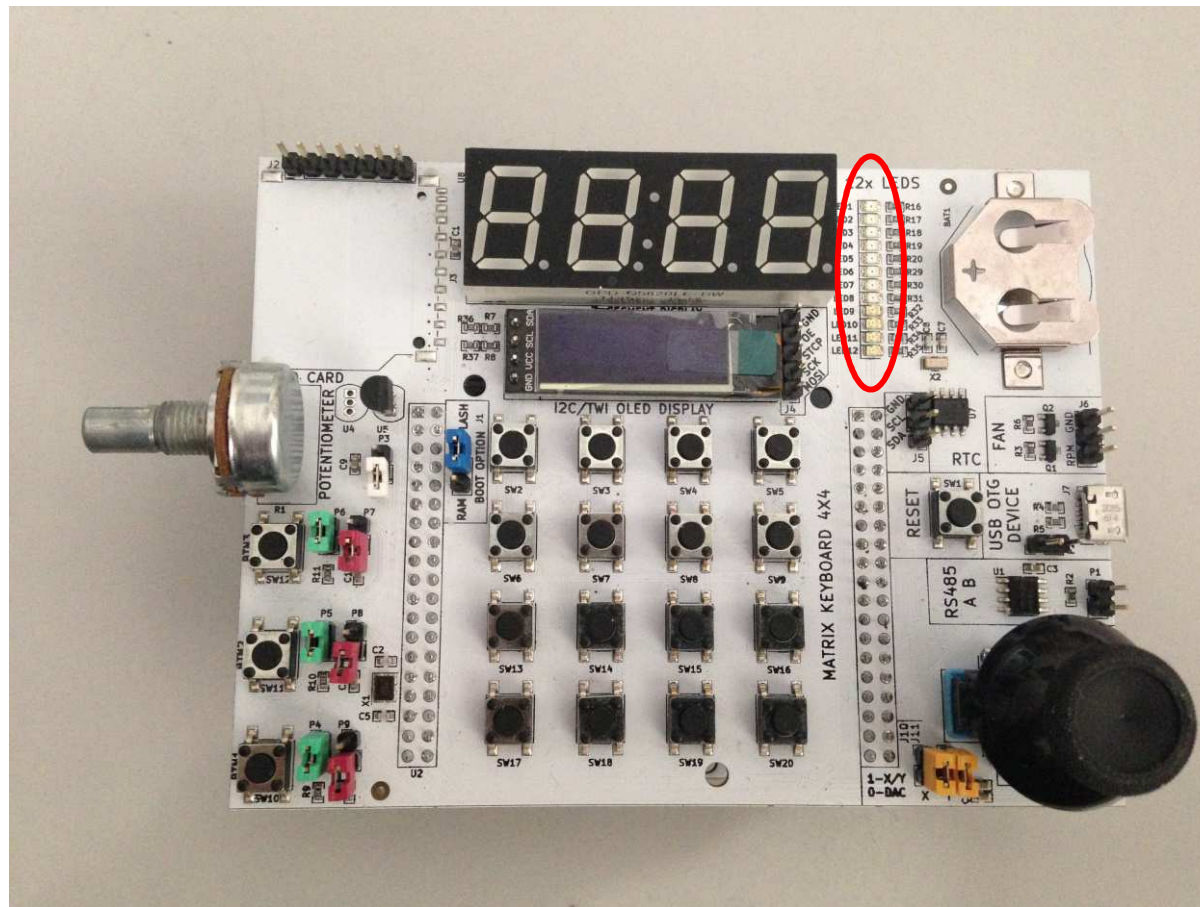
V jazyce C jsou všechny proměnné umístěny v datové paměti procesoru. Proto odkazy na ně budou vždy manipulovat s adresou. Operace mezi registry je vždy výhodnější.

⇒ Zápis v jazyce C je vhodné dělat pokud možno co nejdelší, a teprve potom hodnotu ukládat. Zpracování příkazu pak probíhá s mezi registry.

SAMOSTATNÉ ÚLOHY – ZADÁNÍ ÚLOHY 2

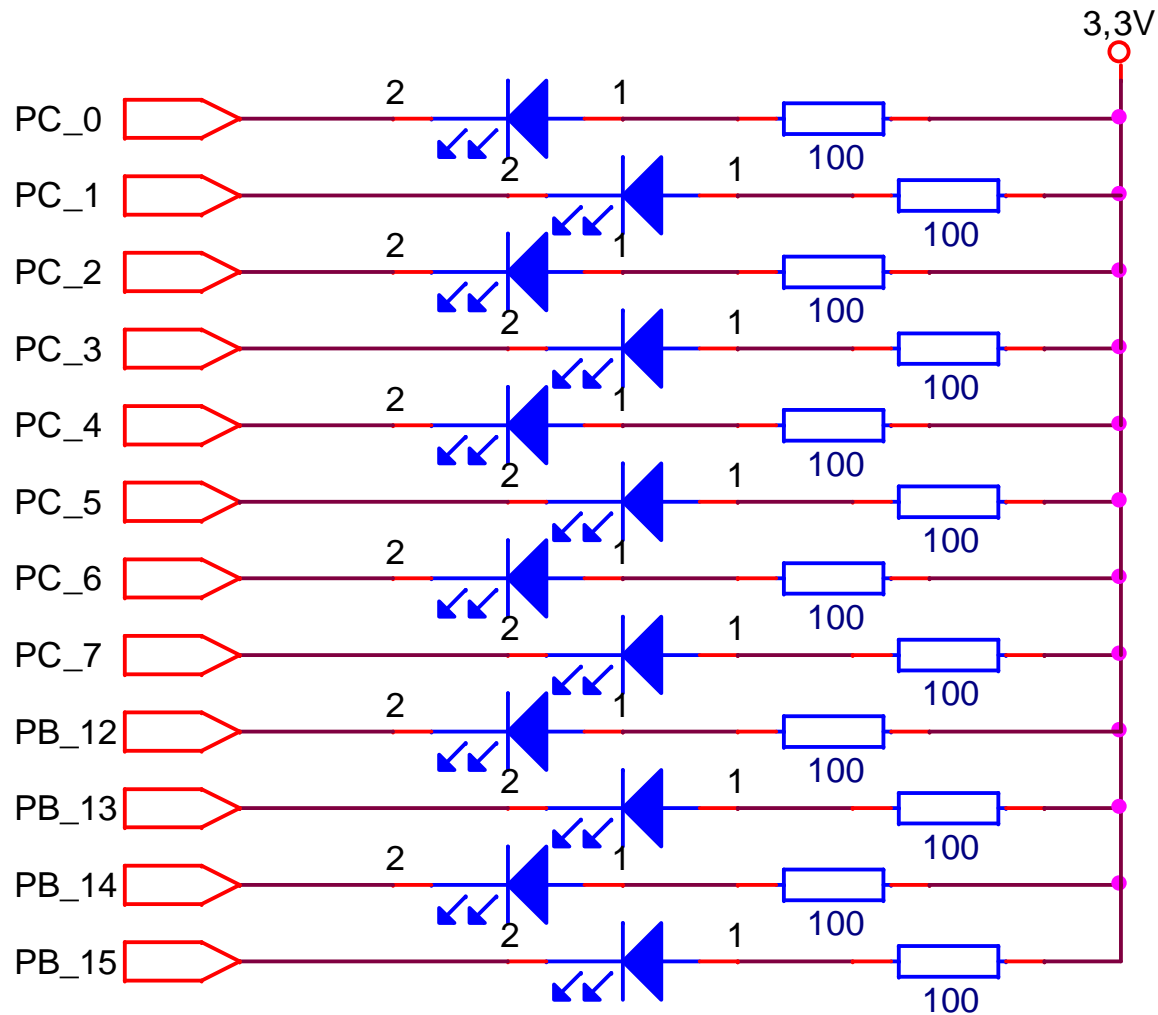
Realizace světelného HADA

Navrhněte v jazyce C program s vloženým programem v assembleru realizující funkci světelného hada tvořeného 12 diodami LED na modulu z obrázku. Rychlost pohybu HADA bude realizována přerušovací rutinou časovače TIM2 a Váš ASM program bude posouvat HADA od PB15 k PC0.



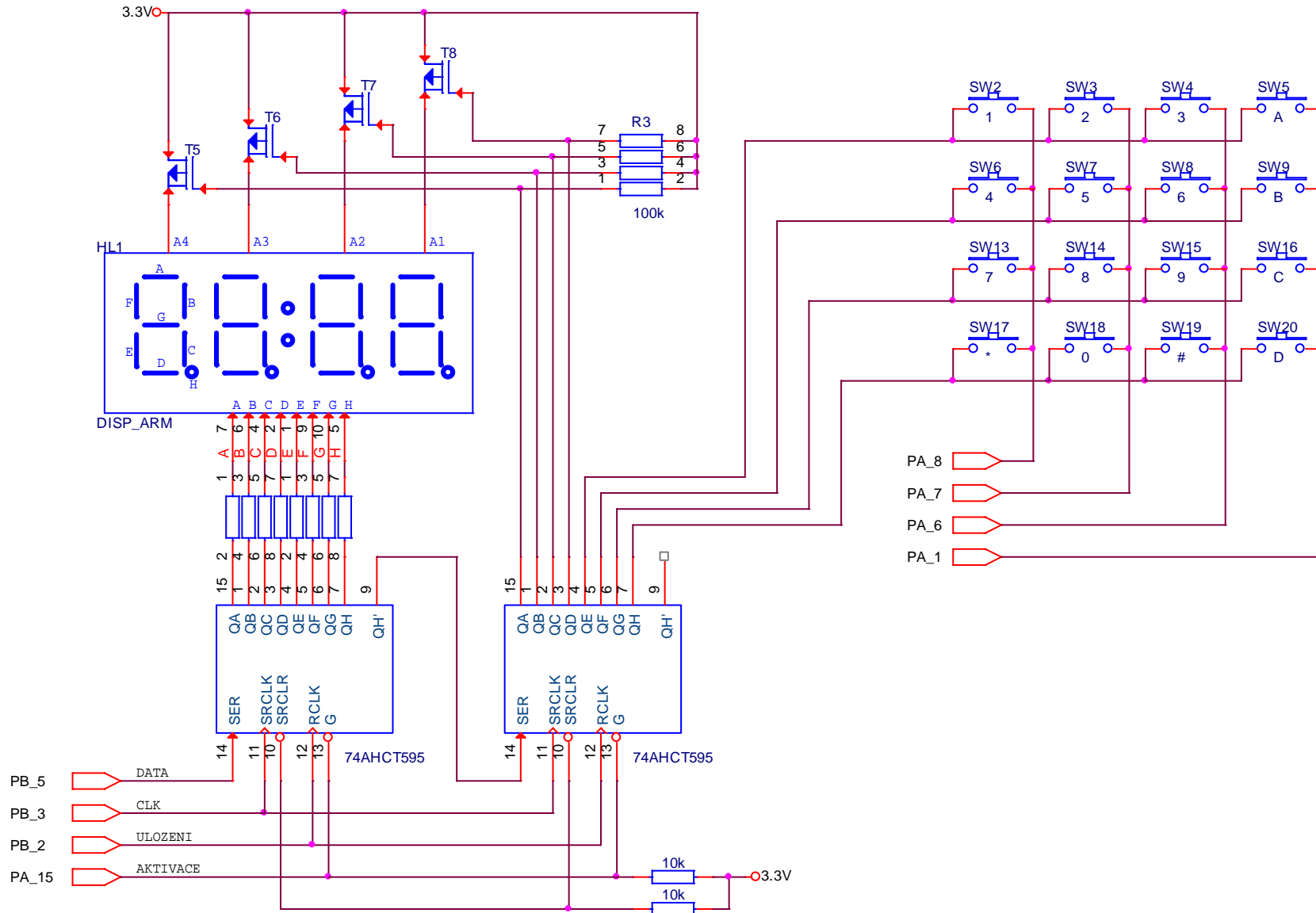
ZAPOJENÍ ÚLOHY 2

Světelný HAD je tvořen 12 diodami LED připojenými k spodní polovině brány PC0 až PC7 a horní čtvrtině brány PB12 až PB15.



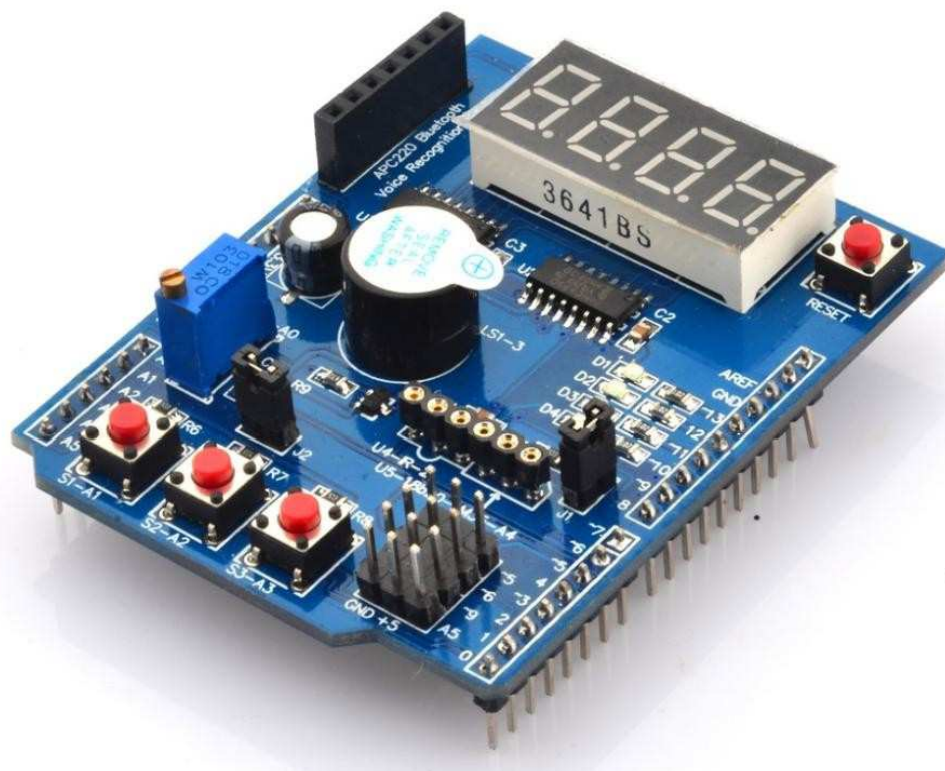
ZAPOJENÍ ÚLOHY 2

Zapojení displeje na desce ze strany 1.



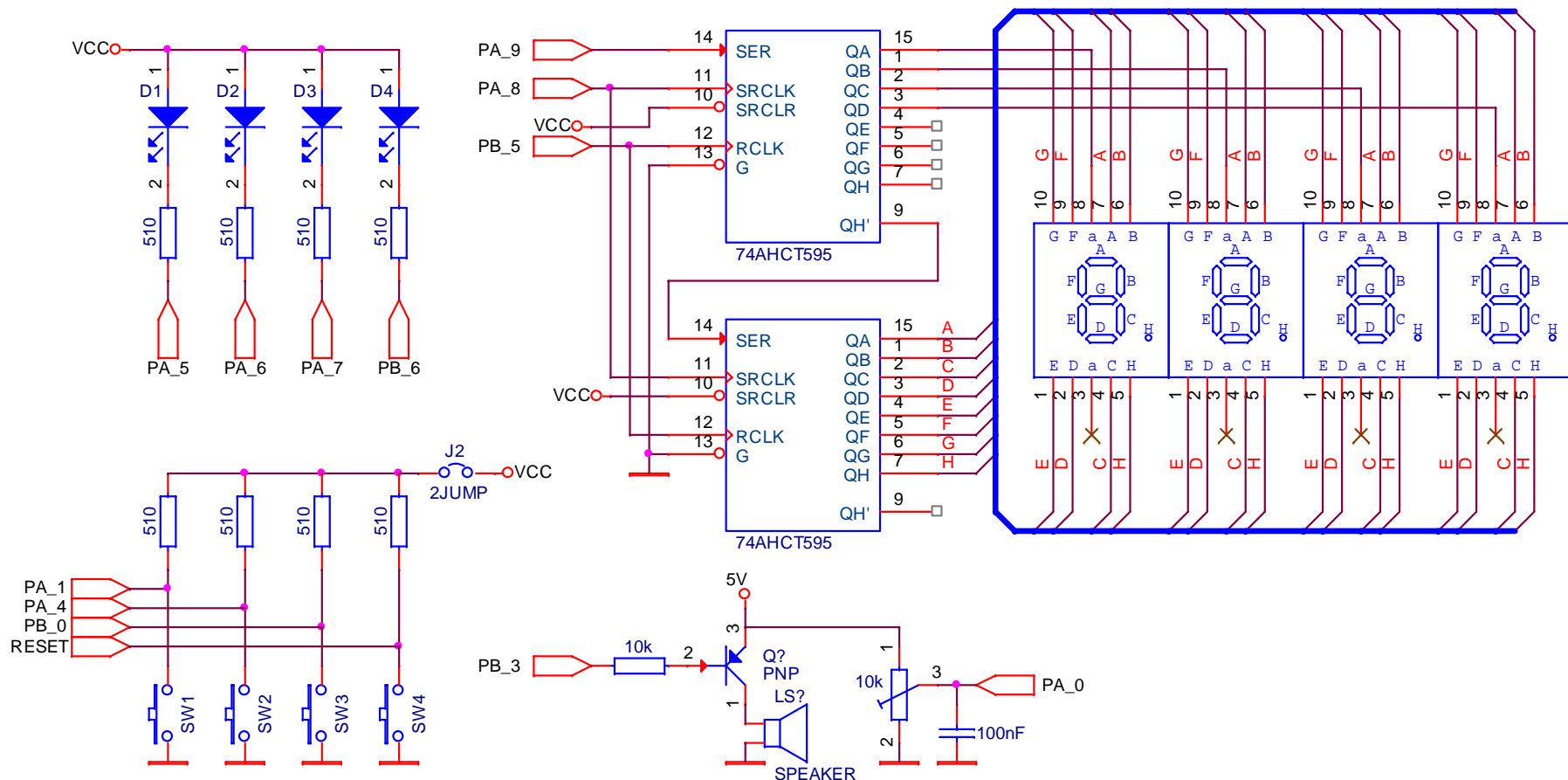
ČTYŘMÍSTNÝ DYNAMICKY OVLÁDANÝ DISPLEJ

Alternativně lze úlohu 2 realizovat na 4 diodách nebo displeji uvedeného nastavného modulu z obrázku. Nastavný modul se zasune do konektorů ARDULINO. **Při nasazení je potřeba dát pozor**, aby destička trvale nezmáčkla černé tlačítko.



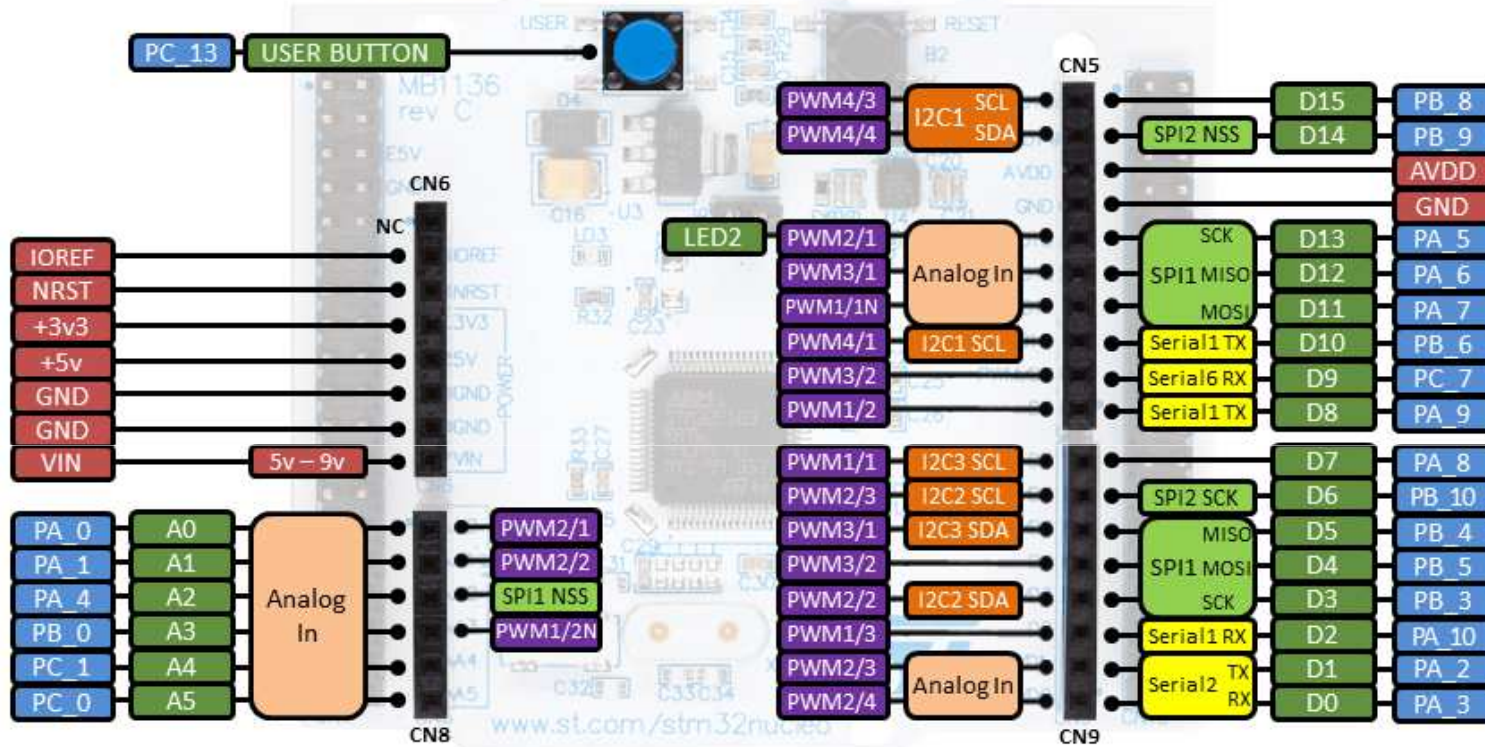
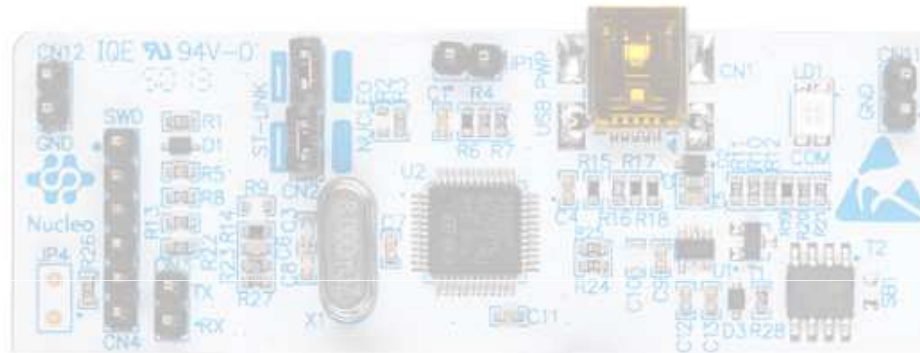
ZAPOJENÍ ČTYŘMÍSTNÉHO DYNAMICKY OVLÁDANÉHO DISPLEJE

Displej na modulu se společnými anodami je ovládán dvěma obvody 74595 kaskádně zapojenými. Do obvodů se informace ukládá bit po bitu přes sériový vstup (IO vývod 14, PB8) následovaný náběžnou hranou hodin SRCLK (IO-11, PA8). Po zapsání všech 16 bitů se obsah registrů překopíruje náběžnou hranou RCLK (IO-12, PA5). Po zapsání všech 16 bitů se obsah registrů překopíruje náběžnou hranou RCLK (IO-12, PB5).

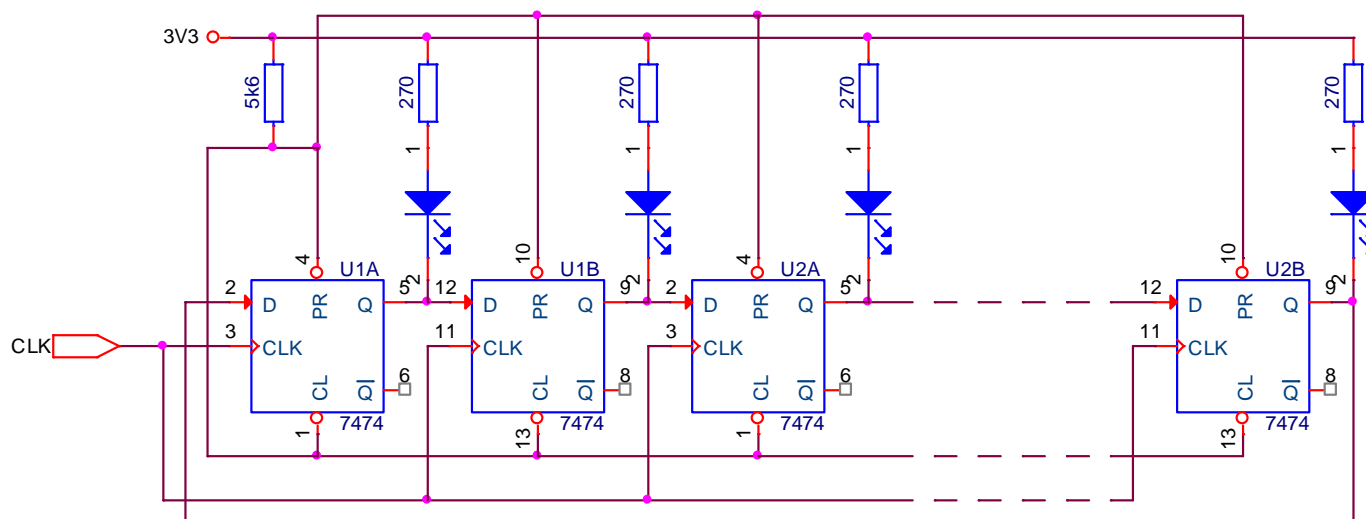


POPIS KONEKTORŮ NA DESCE NUCLEO ST32F401RE


 life.augmented
Nucleo F401RE
 Arduino Headers



OBVODOVÉ ŘEŠENÍ ÚLOHY 2



Každý výstup PČ může představovat **jeden bit proměnné** realizovaného programu posunu. Změna hodnoty v posuvném registru odpovídá **posunu doleva/doprava** ve zvolené proměnné programu. Pro cyklické opakování posunu **HADA** odpovídajícímu schématu je potřeba v programovém řešení uchovat 12 bit před posunem nebo 13 bit po posunu. Uchovaný bit bude následně přemístěn na nejnižší bit posunuté proměnné (posun doleva). Pro posun doprava je třeba algoritmus modifikovat.

VÝVODY OBSAZENÉ DESTIČKOU DISPLEJE A JEJICH POUŽITELNOST

| Indikační diody | | | | |
|-----------------|---------------------|-------|-------------|---------------------|
| Vývod | Funkce | Vstup | Výstup | Alternativní Funkce |
| PA-5 | Led D1 | ANO | ANO | ANO |
| PA-6 | Led D2 | ANO | ANO | ANO |
| PA-7 | Led D3 | ANO | ANO | ANO |
| PB-6 | Led D4 | ANO | ANO | ANO |
| Tlačítka | | | | |
| PA-1 | SW-1 | ANO | NE | ANO (vstupní) |
| PA-4 | SW-2 | ANO | NE | ANO (vstupní) |
| PB-0 | SW-3 | ANO | NE | ANO (vstupní) |
| Reset | SW-4 | ANO | NE | NE |
| Displej | | | | |
| PA-9 | Sériová data | ANO | ANO | ANO |
| PA-6 | Hodinový signál | ANO | ANO | ANO |
| PA-7 | Zápis posloupnosti | ANO | ANO | ANO |
| Zvuk | | | | |
| PB-3 | Spínání repro | NE | ANO | ANO |
| Analogový vstup | | | | |
| PA-0 | Vstup děliče napětí | ANO | ANO (pozor) | ANO |

Záchytný systém AF – PA-6, PB-4, PC-6 (AFIO2)

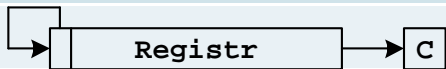
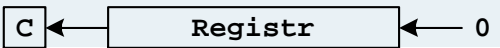
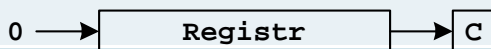
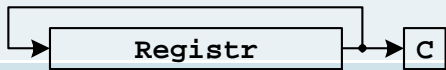

IC1IOS (AFIO14) – PA-0, PA-4, PA-8, PC-0, PC-4, atd. dle tabulky IC1IOS

INSTRUKCE ROTACE - OBECNĚ

Operace **ROTACE** můžeme rozdělit na **logické** a **aritmetické**. Procesor může disponovat pouze instrukcemi pro logickou rotaci, z kterých lze snadno vytvořit i rotace aritmetické. **Logická rotace** se využívá k **střádání sériově získávaných bitů, vysouvání jednotlivých bitů do sériové posloupnosti** nebo **při sériové implementaci násobení**.

- Logická rotace doprava = 0 → nejvyšší bit, nejnižší obvykle do příznaku přenosu.
- Logická rotace doleva - situace přesně obrácená (nejnižší bit bude 0 a nejvyšší do příznaku přenosu).
- Aritmetická rotace doprava – bity se posunou a nejvyšší bit se kopíruje sám na sebe. Pro záporné číslo (dvojkový doplněk) je výsledná hodnota = číslo/2.
- U ARM existují další typy rotací jako je ROR a RRX.
- V jazyce C je rotace doprava $oper=oper>>1$ je **určena** vlastnostmi operandu *signed* =aritmetická, *unsigned* = logická.
- Má-li uP barell shifter – možnost posunu o několik bitů najednou.

INSTRUKCE ROTACE

| Instrukce | Operace | Funkce |
|---------------------|--------------------------|---|
| ASR {S} Rd, Rm, #kn | $Rd = Rm \gg kn$ | Aritmetická 32-bitová rotace vpravo |
| ASR Rd, Rn | $Rd = Rd \gg Rn$ |  |
| ASR.W Rd, Rm, Rs | $Rd = Rm \gg Rs$ | |
| LSL {S} Rd, Rm, #kn | $Rd = Rm \ll kn$ | Logická 32-bitová rotace vlevo |
| LSL Rd, Rn | $Rd = Rd \ll Rn$ |  |
| LSL.W Rd, Rm, Rs | $Rd = Rm \ll Rs$ | |
| LSR {S} Rd, Rm, #kn | $Rd = Rm \gg kn$ | Logická 32-bitová rotace vpravo |
| LSR Rd, Rn | $Rd = Rd \gg Rn$ |  |
| LSR.W Rd, Rm, Rs | $Rd = Rm \gg Rs$ | |
| ROR {S} Rd, Rm, #kn | Rotace Rm hodnotou kn | 32-bitová rotace vpravo |
| ROR Rd, Rn | Rotace Rd hodnotou Rn |  |
| ROR.W Rd, Rm, Rs | Rotace Rm hodnotou Rs | |
| RRX.W {S} Rd, Rm | Rotace {C, Rd} = {Rn, C} | 33-bitová rotace vpravo |
| | |  |

Řešení druhé úlohy se skládá z:

- ❖ Úlohy 1 s úpravou inicializace V/V za pomoci souboru `Nastaveni_GPIO.c`.
- ❖ Inicializace přerušování od časovače TIM2, které nyní bude realizovat časový interval posunu HADA \Rightarrow smažeme funkci `Delay()` z úlohy 1 stejně jako čtení stavu tlačítka.
- ❖ V přerušovací rutině TIM2 provedeme podprogram posunu HADA nebo nastavíme indikátor, který v hlavním programu zavolá podprogram posunu.
 - TIM2 bude určovat periodu posunu.
 - Periodu posunu bude určovat **N** zavolaných obsluh přerušovací rutiny (viz. Přednášky str.166).
- ❖ Program v jazyce symbolických adres (vložený ASM) realizující posun HADA.

Postup řešení druhé úlohy

- ❖ Vyjdeme z funkční úlohy 1 nebo 2
- ❖ Smažeme funkci Delay()
- ❖ Budeme inicializovat nastavení časovače TIM2 na funkci vzestupně nebo sestupně čítajícího čítače, který při dosažení maximální hodnoty uložené v registru ARR (nebo nuly) nastaví žádost o přerušení od UIF (Update interrupt flag)
- ❖ Konfiguraci bitů TIM2_CR1 a TIM2_CR2 ponecháme ve stavu po vynulování procesoru. Tj. všechny bity budou nulové = čítač čítá nahoru a je zastaven.
 - Povolíme hodinový signál pro časovač TIM2
 - Nastavíme hodnotu předděliče PSC
 - Nastavíme hodnotu maxima čítače v registru ARR
 - Povolíme přerušení od časovače TIM2
 - Povolíme přerušení od časovače TIM2 v kontroléru NVIC

ŘEŠENÍ ÚLOHY 2

- Spustíme časovač nastavením bitu CEN v registru TIM2_CR1.
- Pro jistotu smažeme příznak UIF (Update interrupt flag)

Obsluha přerušení od časovače TIM2 se bude skládat

- **Nulování příznaku UIF**, který je jedním z mnoha příznaků generovaných podle konfigurace časovače TIM2.
- Nastavení indikátoru posunu HADA nebo zavolání ASM podprogramu.

Vytvoření assemblerovského podprogramu posunu HADA.

Při obvodovém řešení by výstupy posuvného registru ovládaly jednotlivé diody LED. Rychlost posunu by určoval hodinový signál viz. strana 13. Přednastavení PC není ve schématu řešeno.

ŘEŠENÍ ÚLOHY 2

Před vytvořením ASM programu musíme rozhodnout, kterou část řešení programu svěříme.

- **Realizaci kruhového posunu**
- Realizaci posunu a přímý zápis hodnoty do brány PC a PB, nebo diod na nastavném modulu, nebo segmentů na jednom 7 segmentovém displeji

Program **posun_hada** bude do jazyka C zakomponován následovně. Jazyku C bude sděleno, že vně existuje podprogram

```
extern unsigned int posun_hada(unsigned int hodnota);
```

Podle pravidel přenosu hodnot do podprogramů, bude **hodnota** přesunuta **do registru R0**. Stejně jako hodnota, kterou podprogram **posun_hada** vrátí.