České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra měření
Laboratoř videometrie

Možnosti použití balíku STM32duino

Vypracoval: Stanislav Novák, novaks11@fel.cvut.cz
Vedoucí práce: doc. Ing. Jan Fischer, CSc
Verze: 26. března 2024

0.1 Úvod, vhodné přečíst před čímkoli dalším

STM32duino je:

- softwarový balík rozšiřující nástroje Arduina o možnost programování mikrokontrolérů STM32
- komunitní projekt (byť v současnosti velkou měrou udržovaný STMicroelectronics)

Balík jako takový se nachází na githubu (github.com/stm32duino). Zde se nachází i částečná dokumentace ve které je možné dohledat některé klíčové informace.

Tato kapitola si neklade za cíl tuto dokumentaci plně nahradit/doplnit, cílem je vytvořit komplexní materiál který nebude nutně založen na ucelené koncepci a bude dokumentovat alespoň část problematiky. Tento dokument v žádném případě neaspiruje na referenční dokument, kapitola odráží stav STM32duina v době kdy byl sepisován (leden 2024).

Mimo githubu existuje diskuzní fórum www.stm32duino.com/, případně je příležitostně možné nalézt informace i jinde (např. na community.st.com). Na webech nadšenců lze také nalézt základní návody, obvykle však nejdou výrazněji do hloubky (i přesto existuje vcelku použitelný od Petra Šrámka na chiptron.cz[1]).

1 Základy STM32duina, instalace

1.1 "Pohádka"o Arduinu

Pro efektivní užití STM32duina je dobré mít určitou představu o Arduinu (resp. jeho softwarových nástrojích). Arduino začínalo (jako mnoho jiných užitečných otevřených záležitostí) jako školní projekt, konkrétně kdesi v Itálii. Cílem bylo naučit studenty efektivně základy programování; "hmatatelně"na vcelku jednoduchých deskách (pouze mikrokontrolér s minimem dalších součástek) založených zprvu na mikrokontrolérech ATmega. Samotné Arduino používá jazyk C (resp. C++) rozšířený (původně pro potřebu výuky) o knihovny abstraktních funkcí umožňující uživatelsky přístupně ovládat hardware (např. nastavit obdélníkový signál jisté frekvence na pinu funkcí s pouze dvěma argumenty). Tyto knihovny se často označují souhrnně jako jazyk Wiring.

Pro STM32duino je použita pouze softwarová část, konkrétně IDE a jazyk Wiring. Samotné IDE je možné stáhnout z webu www.arduino.cc/en/software; v tuto chvíli jsou dostupné dvě verze (1.y.z a 2.y.z) pro účely STM32duina nezáleží, kterou si vyberete. Arduino IDE je multiplatformní, existují nativní verze pro Linux i macOS. Samotná instalace je přímočará; Arduino je superpopulární a v případě problému lze dohledat téměř nekonečné množství rad/návodů na internetu.

Dokumentaci k jazyku Wiring lze nalézt na webu arduino.cc[2]. Podle této specifikace se mají chovat funkce i v STM32duinu (rozumějte ve významu "emulace"chování ATmegy).

1.2 Instalace STM32duina

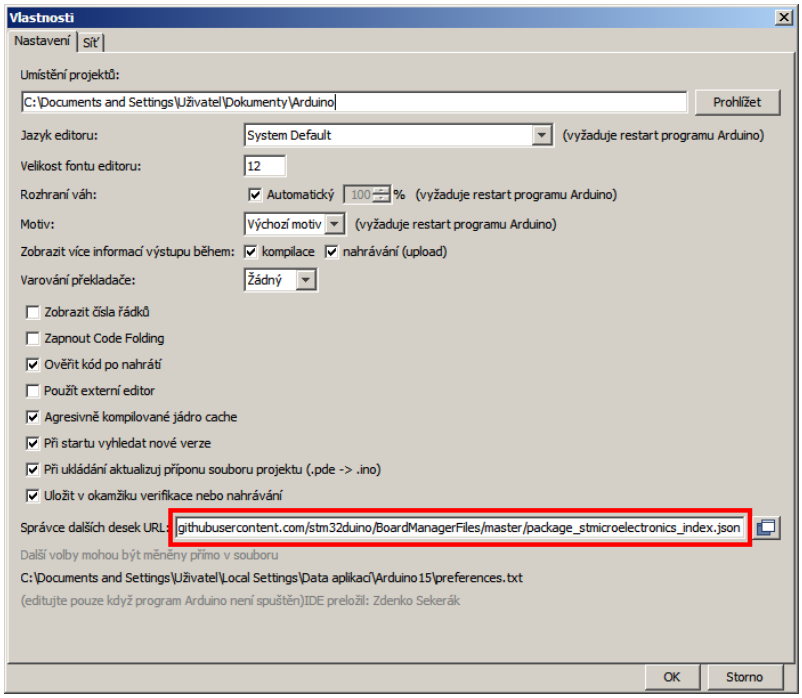
Do samotného Arduina je třeba nainstalovat STM32duino. Konkrétní postup lze nalézt na githubu[3], je dobré vycházet z tohoto, jelikož se adresa `.jsonu` již několikrát změnila. Alternativou (českou) může být článek P. Šrámka[4], případně je postup uveden v bakalářské práci autora tohoto dokumentu (dále pouze jako bp; webové tam odkazy již zastaralé).[5] Po samotné instalaci balíku je třeba zvolit desku a metodu nahrávání programu.

[1] https://chiptron.cz/articles.php?article_id=155
[2] https://www.arduino.cc/reference/en/
[3] https://github.com/stm32duino/Arduino_Core_STM32/wiki/Getting-Started
[4] https://chiptron.cz/articles.php?article_id=155
[5] Novák, Stanislav *Jednoduché laboratorní přístroje pro výuku realizované mikrořadiči ATmega 328 a STM32*

1.2.1 Instalace STM32duina do prostředí Arduino IDE (převzato z bp, odkazy aktualizovány)

(celý postup vychází z dokumentace[6])

1. Spustíme aplikace Arduino IDE, v nabídce **Soubor** vybereme **Vlastnosti**.
2. V okně **Vlastnosti** vložíme do kolonky **Správce desek** (obrázek 1) `https://github.com/stm32duino/BoardManagerFiles/raw/main/package_stmicroelectronics_index.json`.



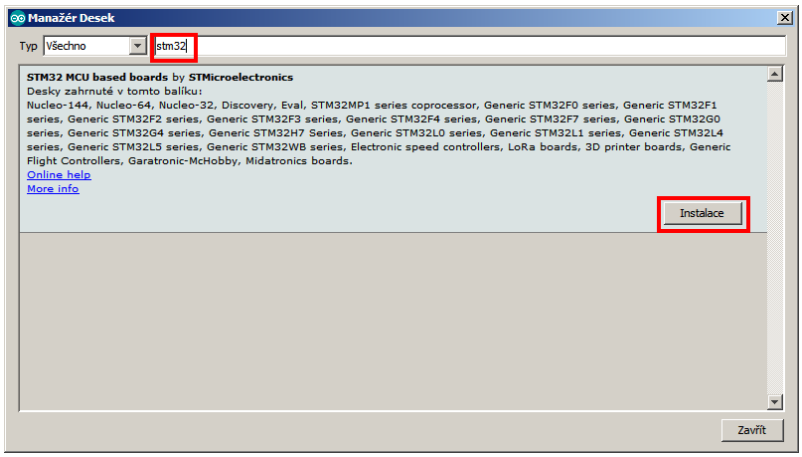
Obrázek 1: Okno **Vlastnosti** aplikace Arduino IDE

   Okno **Vlastnosti** zavřeme pomocí tlačítka **OK**.

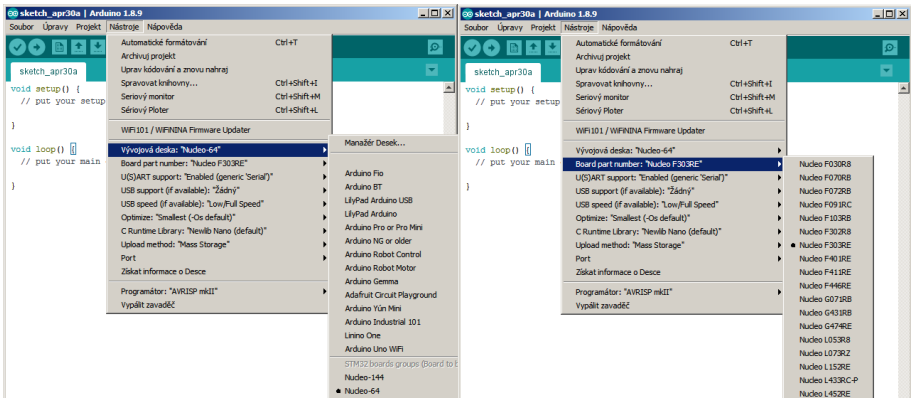
3. Otevřeme nabídku **Nástroje** → **Vývojová deska** → **Manažer Desek**. Zde zadáme do vyhledávání STM32 (obrázek 2) a nainstalujeme balík **STM32 MCU based boards**. Po instalaci okno **Manažér desek** zavřeme.
4. V nabídce **Nástroje** → **Vývojová deska** → **STM32 Boards** na horní liště už pouze zvolíme nejdříve sérii mikrokontroléru (obrázek 3, vlevo), a poté v nabídce **Nástroje** → **Board part number** konkrétní mikrokontrolér (obrázek 3, vpravo).

V nabídce **Nástroje** → **Upload Method:** zvolíme metodu nahrávání firmware do desky (obrázek 4). Uživatelsky nejpřívětivější je **Mass Storage**. Možnost **Mass Storage** je dostupná jen pro moduly Nucleo, případně další desky od ST (respektive desky obsahující ST-Link V2.1/V3). I přesto se u některých desek stává, že STM32duino nemá správně definován název zařízení – tento problém lze dočasně vyřešit přejmenováním zařízení **Mass Storage** v **Průzkumníku Windows** (např. testovaný modul G031-DISCO byla dočasně přejmenována z DIS_G031J6 na NOD_G031J6). Pokud programujeme modul, který možnost **Mass Storage** neposkytuje, musíme použít jinou metodu a musíme instalovat a nastavit aplikaci STM32CubeProgrammer.

[6] `https://github.com/stm32duino/Arduino_Core_STM32/wiki/Getting-Started`



Obrázek 2: Okno **Manažér desek**



Obrázek 3: Výběr modulu rodiny STM32

1.2.2 Instalace a nastavení STM32CubeProgrammer (převzato z bp)

Aplikaci STM32CubeProgrammer potřebujeme, pokud nemáme pro náš mikrokontrolér dostupné nahrávání pomocí **Mass Storage**.

Stažení a Instalace: Z webu[7] stáhneme aplikaci a nainstalujeme (přímočaré).

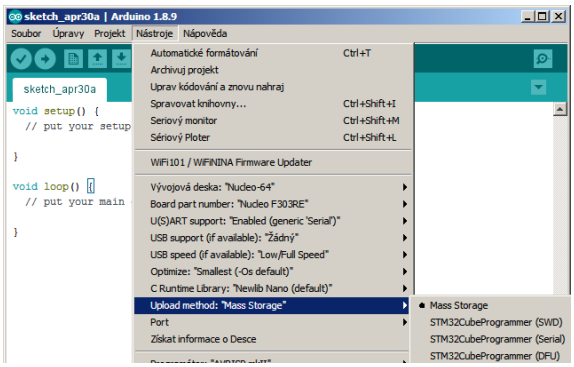
1.3 Prostředí Arduino IDE (převzato z bp, upraveno)

Arduino IDE je multiplatformní nástroj, který bude v této práci primárně používán pro nahrávání programů do mikrokontroléru.

1.3.1 Použití:

Sama aplikace má pro přehlednost jen několik málo prvků, rozdělme je na horní lištu, řádek s ikonami, bílou kartu sloužící k zápisu kódu, černé pole s výpisem informací a stavový řádek,viz obrázek 5. Pro nahrání již hotového programu je nutné program otevřít (pomocí horní lišty **Soubor**

[7] `https://www.st.com/en/development-tools/stm32cubeprog.html`



Obrázek 4: Výběr metody nahrávání firmware do modulu

→ **Otevřít**). Po nahrání programu do mikrokontroléru slouží ikona se symbolem "→"na řádku s ikonami. Pokud proběhlo nahrání do modulu správně, vypíše se nám tato skutečnost do černého pole, v opačném případě se vypíše chyba. K nejčastějším chybám patří:

- **Compilation error: ...** (typicky) Chyba v kódu – přečteme si, co nám prostředí hlásí a chybu opravíme
- **Failed uploading: ...** Chyba při nahrávání; má několik možných příčin:
  - **Failed uploading: no upload port provided** nebo
    Failed uploading: Cannot open port... Chybně nastavený COM-port – ten nastavíme pomocí horní lišty **Nástroje** → **Port**. Pokud nevíme, který port používáme, zkoušíme nejdříve vyšší čísla portů. Pozn.: Pro čínské klony Arduino (využívající UART/USB převodník CH340) je obvykle nutné doinstalovat ovladače. Pod Linuxem je také možná nepřístupnost tty portů – je nutné přidat uživatele do skupiny dialout.
  - **Error: Activating device: KO...** Tato chyba indikuje ze se podařilo otevřít COM port, ale mikrokontrolér STM32 neodpovídá. V tuto chvíli je třeba zkontrolovat který COM port je nastaven, dále zapojení UARTU (Rx-Tx), případně ověřit, zda je zařízení v bootloaderu (v tom případě si BOOT0; kompletní specifikace AN2606[8])
  - **STM32CubeProgrammer not found...**, viz sekce 3.2
  - **Error: No debug probe detected** Je nastavena nahrávací metoda SWD (ST-Link) a žádný ST-Link není připojen
  - **Error: Target device not found** Je nastavena nahrávací metoda USB-DFU a žádné zařízení nebylo nalezeno.
  - **"..."not found** Pravděpodobně je zvolena metoda Mass storage a není připojeno tomuto odpovídající zařízení (odpovídající Nucleo).

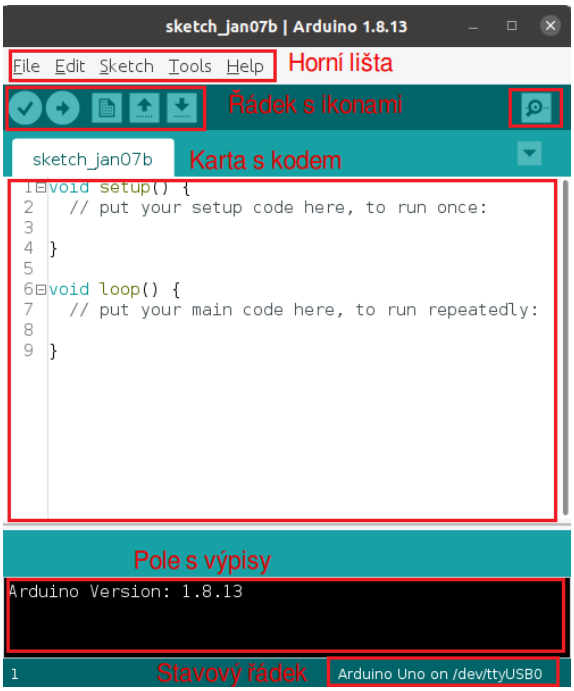
2 Základní funkce a příklady

2.1 Digitální vstupy a výstupy

Alternativně je možné vycházet z článku na chiptron.cz[9].

[8] https://www.st.com/resource/en/application_note/an2606-stm32-microcontroller-system-memory-boot-mode-stmicroelectronics.pdf

[9] https://chiptron.cz/articles.php?article_id=157



Obrázek 5: Prostředí Arduino IDE

Jako tradiční první příklad je uvedeno blikání LED, obvykle ověřující, že je možné do zařízení nahrát vlastní program. V Arduino IDE stačí otevřít příklad (příklady se nacházejí ve "File"→"Examples") "Basics"→"Blink", případně přepsat kód z této sekce.

```
1  void setup() {
2    pinMode(LED_BUILTIN, OUTPUT);
3  }
4
5  void loop() {
6    digitalWrite(LED_BUILTIN, HIGH);
7    delay(100);
8    digitalWrite(LED_BUILTIN, LOW);
9    delay(100);
10 }
```

Samotné Arduino skrývá klasickou `main()` funkci za funkce `setup()` (jednou se spustí na začátku) a `loop()` (po skončení `setup()` se spouští donekonečna). Důležitým rozdílem oproti Arduinu je nutnost vždy iniciovat pin (`pinMode()`).

Pokud bychom chtěli číst digitální hodnotu pinu (např. tlačítka), je třeba iniciovat pin jako digitální vstup (`INPUT`) a poté lze číst hodnotu funkcí `digitalRead()`.

2.2 UART (sériová linka)

Alternativně je možné vycházet z článku na chiptron.cz[10].

Arduino jako takové typicky počítá se sériovou linkou pro dva účely:

[10] https://chiptron.cz/articles.php?article_id=163

1. Nahrávání programu
2. Uživatelské výpisy/ovládání

Pro uvěření funkce výpisů je možné použít příklad "Communication"→"ASCII". Po nahrání programu a otevření sériového terminálu (v "Tools"→"Serial Monitor") bychom měli vidět výpis ASCII abecedy. Při nastavování UARTU je nutné dodržet nastavení baudrate (rychlost) a dále mít nastavený správný port počítače (v "Tools"→"Port"). Při použití jiných desek, než Nucleo je také třeba dohledat na kterých pinech se nachází používaný UART (sekce 3.1), nebo přemapovat piny pomocí funkcí `setRx()` a `setTx()` (podle dokumentace[11]), nebo využití instancí `"Serialx"`(podle dokumentace[12]).

Při posílaní dat do desky je možno použít funkce `Serial.available()` (kolik znaků se je nabufferováno) a `Serial.read()` (vrací znak z bufferu), případně jiné; konkretní použití lze vidět v konkrétních příkladech Arduina; obvykle však stačí si přečíst dokumentaci[13] k funkcím. Jako triviální příklad může sloužit i takovéto echo:

```
1 void setup() {
2   Serial.begin(115200);
3 }
4
5 void loop() {
6   if(Serial.available()) Serial.write(Serial.read());
7 }
```

2.3 Analogové čtení, analogový zápis, čítače

Analogové čtení je realizováno funkcí `analogRead()`. Zde je možné se setkat s jednou zvláštností, ATmega disponovala 10-bitovým ADC a STM32 disponují typicky 12-bitovým ADC, přesto návratová hodnota je i pro STM32 0...1023 (odběr 12 bitů, ale bitový posun pro zpětnou kompatibilitu). Pro nastavení 12-bitového rozlišení je třeba zavolat funkci `analogReadResolution(12)` (dle dokumentace[14]).

Funkce `analogWrite()` má vcelku specifické chování; pokud je z na cílovém pinu čítač začne generovat obdélníkový signál (v STM32duinu typicky 1000Hz) se střední hodnotou úměrnou analogové hodnotě. Pokud na pinu není DAC, ani čítač digitálně se zapíše (proti polovině číselného rozsahu). Dále je možné v případě, že je na pinu DAC využít jeho, chování funkce lze poupravit pomocí maker, vcelku hezky zpracováno na githubu,[15] jak. Jako příklad, je zde uvedeno generování "sinusového průběhu"pomocí PWM (převzto z bp):

```
1 unsigned char c=0;
2 unsigned char sine[256];
3
4 void setup() {
5   for(int i=0; i<256; ++i) sine[i]=(sin(i*2*PI/255.)+1)*255/2;//
      sine counting
6 }
7
8 void loop() {//loop genarating sine with period 1.024 s
9   analogWrite(D6, sine[c]);// PWM out, given duty
```

[11] https://github.com/stm32duino/Arduino_Core_STM32/wiki/API#hardwareserial
[12] https://github.com/stm32duino/Arduino_Core_STM32/wiki/API#hardwareserial
[13] https://www.arduino.cc/reference/en/language/functions/communication/serial/
[14] https://www.arduino.cc/reference/en/language/functions/analog-io/analogreadresolution/
[15] https://github.com/stm32duino/Arduino_Core_STM32/wiki/API#analog

```
10    delay(4);//4 ms delay
11    c++;//
12  }
```

Poslední funkcí je `tone()`. Jedná se o generování obdélníkového signálu se střídou 1:1, dané frekvence. Arduino má pro generování obdélníku jednu zásadní limitaci, je možné definovat frekvenci, nebo střídu (=ale ne obojí najednou). V stm32duinu však došlo k do-iplementování takovýchto funkcí, včetně dokumetace (knihovna HardwareTimer)[16]. Jednoduchý příklad s PWM s knihovnou může vypadat takto:

```
1  HardwareTimer htim2 = HardwareTimer(TIM2);
2
3  void setup() {
4    htim2.setPWM(1, PA0, 100, 50); //Channel 1, pin PA0, 100Hz, 50%
       duty
5  }
6
7  void loop(){}
```

2.4 I2C, SPI, knihovny

Částečně je možné vycházet z článku na chiptron.cz[17].

Úvodem, v Arduinu se periferie I2C nejmenuje I2C, ale TWI (z historických důvodů), proto se ani knihovna, kterou je třeba vložit nejmenuje I2C, ale `Wire.h`.

Přiložený příklad ukazuje generování pruhů na I2C variantě SSD1306 (vcelku neprůhledným způsobem, přesto důležitá je zde komunikace a ta by měla být jasná).

```
1  #include <Wire.h>
2
3  uint8_t initSTR[] = {//string for initialization of lcd
4   0xAE,0x20,0x00,0xB0,0xC8,0x00,0x10,0x40,0x81,0xFF,0xA1,0xA6,0xA8,0
      x3F,
5   0xA4,0xD3,0x00,0xD5,0xF0,0xD9,0x22,0xDA,0x12,0xDB,0x20,0x8D,0x14,0
      xAF
6  };
7
8  void setup() {
9    Wire.begin();
10   Wire.beginTransmission(0x3C);//beginning of transmission
11   Wire.write(initSTR, sizeof(initSTR));//"buffering" of the data
12   Wire.endTransmission();//end of transmission
13 }
14
15 void loop() {
16   delay(1000);
17   for (uint8_t i = 0; i < 64 / 8; i++) {
18     Wire.beginTransmission(0x3C);
19     Wire.write(0x40);//write data
20     for (uint8_t j = 0; j < 128; j++) Wire.write(j & 4 ? 0x0 : 0xff
       );
```

[16] https://github.com/stm32duino/Arduino_Core_STM32/wiki/HardwareTimer-library
[17] https://chiptron.cz/articles.php?article_id=166

```
21      Wire.endTransmission();
22    }
23
24    delay(1000);
25    for (uint8_t i = 0; i < 64 / 8; i++) {
26      Wire.beginTransmission(0x3C);
27      Wire.write(0x40);//write data
28      for (uint8_t j = 0; j < 128; j++) Wire.write(j & 8 ? 0x0 : 0xff
        );
29      Wire.endTransmission();
30    }
31  }
```

Jelikož oproti ATmeze je na STM32 typicky větší množství možností přemapování, byly přidány přemapovávací funkce pinů I2C (`Wire.setSDA()` a `Wire.setSCL()`), opět na github[18]u lze nalézt krátkou dokumentaci[19], původní možnosti zůstávají dle specifikace (např. nastavení hodin) na arduino.cc[20].

Pro použití většiny zařízení, již v současnosti existují knihovny. Tyto knihovny jsou dostupné buď přímo z Arduino IDE, nebo jako `.zip` archivy (základní informace o knihovnách je možné nalézt na arduino.cc[21]). V předmětu LPE katedry měření FEL ČVUT je využíván tříosý magnetometr MMC5883. V současnosti existuje knihovna od firmy Adafruit dostupná jako `.zip` archiv z github[22], je možné ho přidat pomocí "Sketch"→"Include Library"→"Add .ZIP Library..."(ke knihovně je nutné doinstalovat závilosti (knihovny); ty jsou již dostupné z Arduino IDE, alespoň "Adafruit Unified Sensor"). Pro začátek je dobré si otevřít příklad dodávaný s knihovnou (v příkladech "Adafruit MMC5883"→"magsensor") a rozšířit ho o mapování pinů SDA a SCL. Základní příklad (alespoň dle chování) nepoužívá "flipování", proto je zde uveden následující příklad (modifikovaný příklad "magsensor") (výstup tří os je možné zobrazit pomocí "Tools"→"Serial Plotter").[23]

```
1
2   #define INTERNAL_CONTROL_0    0x08
3   #define FLIP_SET 0x08
4   #define FLIP_RESET 0x10
5
6   #include <Adafruit_MMC5883.h>
7
8   Adafruit_MMC5883 mag = Adafruit_MMC5883(12345);
9
10  void setup(void)
11  {
12    Serial.begin(115200);
13    Wire.setSDA(PB_9);
14    Wire.setSCL(PB_8);//Warning: PB8 may be also BOOT0 pin on G4
15    if(!mag.begin())
16    {
17      Serial.println("ERROR: magnetometer not found");
18      while(1);//break
19    }
```

[18] https://github.com/stm32duino/Arduino_Core_STM32/wiki/API#i2C
[19] https://github.com/stm32duino/Arduino_Core_STM32/wiki/API#i2C
[20] https://www.arduino.cc/reference/en/language/functions/communication/wire/
[21] https://www.arduino.cc/reference/en/libraries/
[22] https://github.com/adafruit/Adafruit_MMC5883
[23] Před opsáním upraveného příkladu **upozornění:** PB8 může být na STM32G431 použit jako **BOOT0 pin**.

```
  mag.setContinuousFreq(MMC5883_CMFREQ_5HZ);//set frequency
}

void loop(void)
{
  /* Get a new sensor event */
  sensors_event_t event1,event2;
  mag.getEvent(&event1);

  Wire.beginTransmission(0x30);//set flipping pulse
  uint8_t cmd[2] = {INTERNAL_CONTROL_0,FLIP_SET};
  Wire.write(cmd,2);
  Wire.endTransmission();
  delay(400);

  mag.getEvent(&event2);
  Wire.beginTransmission(0x30);//reset flipping pulse
  cmd[1]=FLIP_RESET;
  Wire.write(cmd,2);
  Wire.endTransmission();

  /* Display the results can be used with arduino Serial plotter*/
  Serial.print((event1.magnetic.x-event2.magnetic.x)/2);
  Serial.print(" ");
  Serial.print((event1.magnetic.y-event2.magnetic.y)/2);
  Serial.print(" ");
  Serial.println((event1.magnetic.z-event2.magnetic.z)/2);
  /* "Compass" by atan2 function in degrees for xy
  Serial.println(180*atan2(event1.magnetic.y-event2.magnetic.y,
   event1.magnetic.x-event2.magnetic.x)/PI);*/
  delay(400);
}
```

Použití SPI není v zásadě složitější než I2C, pro přemapování pinů existují obdobné funkce pro STM32duino jako u I2C; dokumentace se nachází na githubu[24]; dále se sluší zmínit i specifikaci knihovny dle Arduina na arduino.cc[25]. Přiložený příklad používá vykresluje dva vzory na LCD pcd8544; opět komunikace vcelku přímočará, čtení by bylo analogické.

```
/* zdroj (minimal configurace pcd8544), upraveno dle hardware:
 * https://www.youtube.com/watch?v=RAlZ1DHw03g, Julian Ilett*/
#include <SPI.h>

#define RST PC7
#define CE  PA9
#define DC  PB6
#define DIN  PA7
#define CLK  PA5

void LcdWriteData(uint8_t dat)
```

[24] https://github.com/stm32duino/Arduino_Core_STM32/blob/main/libraries/SPI/README.md
[25] https://www.arduino.cc/reference/en/language/functions/communication/spi/

```
12 {
13   digitalWrite(DC, HIGH); //DC pin is high for data
14   digitalWrite(CE, LOW);  //CS set low
15   SPI.transfer(dat); //transfer the data
16   digitalWrite(CE, HIGH); // get CS back high
17 }
18
19 void LcdWriteCmd(uint8_t cmd)
20 {
21   digitalWrite(DC, LOW); //DC pin is low for commands
22   digitalWrite(CE, LOW);
23   SPI.transfer(cmd);
24   digitalWrite(CE, HIGH);
25 }
26
27 void setup()
28 {
29   pinMode(RST, OUTPUT);
30   pinMode(CE, OUTPUT);
31   pinMode(DC, OUTPUT);
32   digitalWrite(RST, LOW);
33   delay(1);/
34   digitalWrite(RST, HIGH);
35
36   SPI.setMOSI(DIN);
37   SPI.setSCLK(CLK);
38   SPI.begin();
39
40   LcdWriteCmd(0x21);  // LCD extended commands
41   LcdWriteCmd(0x80|0x20);  // set LCD Vop (contrast)
42   LcdWriteCmd(0x04);  // set temp coefficent
43   LcdWriteCmd(0x14);  // LCD bias mode 1:40
44   LcdWriteCmd(0x20);  // LCD basic commands
45   LcdWriteCmd(0x0C);  // LCD normal video
46 }
47 void loop()
48 {
49
50   for(int i=0; i<504; i++) LcdWriteData(i&4 ? 0xff :0x00); // reset
       LCD
51   delay(2000);
52   for(int i=0; i<504; i++) LcdWriteData(i&2 ? 0xff :0x00); // set
      LCD
53   delay(2000);
54
55 }
```

Pro porovnaní s I2C je zde uveden následující příklad s SSD1306.

```
1 #include <SPI.h>
2
3 uint8_t initSTR[] = {0xAE, 0xD5, 0x80, 0xA8, 63, 0xD3, 0x0, 0x40, 0
    x8D, 0x14, 0x20, 0x00, 0xA1, 0xC8, 0xDA, 0x24, 0x81, 0xCF, 0xD9,
```

```
      0xF1, 0xDB, 0x40, 0xA4, 0xA6, 0xAF);

#define RST PC7
#define CS PA9
#define DC PB6
#define DIN PA7
#define CLK PA5

void command(uint8_t c) {
  digitalWrite(CS, HIGH);
  digitalWrite(DC, LOW);
  digitalWrite(CS, LOW);
  SPI.transfer(c);
  digitalWrite(CS, HIGH);
};

void data(uint8_t c) {
  digitalWrite(CS, HIGH);
  digitalWrite(DC, HIGH);
  digitalWrite(CS, LOW);
  SPI.transfer(c);
  digitalWrite(CS, HIGH);
};

void setup() {
  Serial.begin(9600);
  pinMode(RST, OUTPUT);
  pinMode(CS, OUTPUT);
  pinMode(DC, OUTPUT);

  digitalWrite(RST, LOW);
  delay(1);  //ms delay for being sure
  digitalWrite(RST, HIGH);

  SPI.setMOSI(DIN);
  SPI.setSCLK(CLK);
  SPI.begin();

 //initialization commands, same sequence as by i2c
 for(int i=0;i<sizeof(initSTR)/sizeof(*initSTR);++i) command(
    initSTR[i]);
}

void loop() {
  command(0x0 | 0x0);  // low col = 0
  command(0x10 | 0x0);  // hi col = 0
  command(0x40 | 0x0); // line #0
  digitalWrite(CS, HIGH);
  digitalWrite(DC, HIGH);
  digitalWrite(CS, LOW);
  for(int i=0; i<1024; i++) SPI.transfer(i&4 ? 0xff :0x00); //
    reset LCD
```

```
53    digitalWrite(CS, HIGH);
54
55    delay(2000);
56
57    command(0x0 | 0x0);  // low col = 0
58    command(0x10 | 0x0);  // hi col = 0
59    command(0x40 | 0x0); // line #0
60    digitalWrite(CS, HIGH);
61    digitalWrite(DC, HIGH);
62    digitalWrite(CS, LOW);
63    for(int i=0; i<1024; i++) SPI.transfer(i&2 ? 0xff :0x00); // set
       LCD
64    digitalWrite(CS, HIGH);
65
66    delay(2000);
67  }
```

Poslední příklad ukazuje nastavení SPI pro posuvný registr (CPOL=0, CPHA=0, detaily je možné najít v dokumentaci[26]):

```
1  #include <SPI.h>
2  #define DIN PA7
3  #define CLK PA5
4
5  void setup() {
6    Serial.begin(9600);
7    SPI.setMOSI(DIN);
8    SPI.setSCLK(CLK);
9    SPI.begin();
10   SPI.setDataMode(SPI_MODE0);
11 }
12
13 void loop() {
14   SPI.transfer(0x55);
15   delay(1);
16 }
```

Arduino založené na ATmeze častokráte "trpělo"na nedostatek periferií pro komunikaci s posuvnými registry (SPI), proto existuje softwarová emulace kterou lze použít i v STM32duinu (funkce `shiftOut()`).

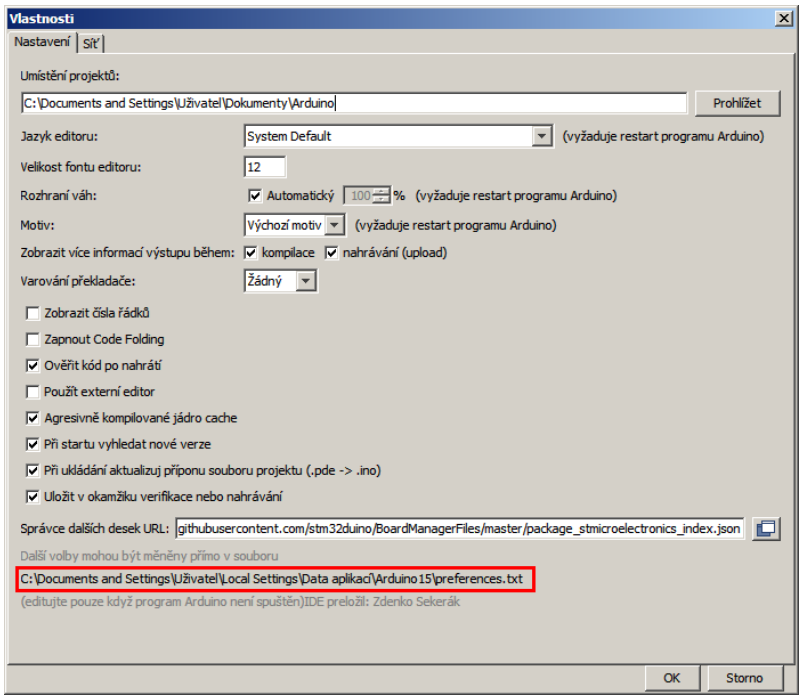
Pro použití SSD1306 existuje řada knihoven často dostupných z Arduino IDE, za zmínku stojí SSD1306Ascii – vcelku minimalistická a disponující příklady pro velké množství konfigurací (rozměrů displaye). Arduinovské knihovny pro zařízení připojená po I2C nebo SPI obvykle bez problémů na STM32duinu (používají se typicky stejné knihovní funkce). Problematičtější je použití knihoven využívajících jiné periferie (konkrétně skvělá a populární čítačová knihovna TimerOne fungovat nebude, je napsána přímo nad registry čítače ATmegy).

[26] https://docs.arduino.cc/learn/communication/spi/

3 Pokročilé možnosti a příklady

3.1 Definice desek

Balík definice jednotlivých desek, zde jsou definované výchozí piny pro UART a jiné. Chiptron[27] se ve článku o UARTu zmiňuje o jednotlivých souborech, neuvádí však kde se nachází. Tuto cestu lze získat z nastavení (File->Prefernces) Arduina, obrázek 6, typicky `%localappdata%/Arduino15`:



Obrázek 6: Okno **Vlastnosti** aplikace Arduino IDE

Zde v podadresáři `packages/STMicroelectronics/hardware/stm32/2.7.1/variants/STM32G4xx/G431R(6-8-B)(I-T)_G441RB(I-T)` (intuitivně upravte pokud třeba pro jinou desku/verzi) lze nalézt jednotlivé definice.

3.2 Přidání STM32CubeProg do proměnné PATH systému: (převzato z bp)

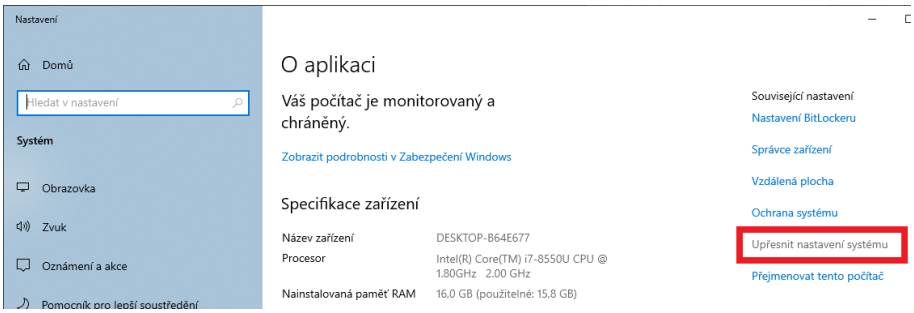
Ve starých verzích bylo toto nutné, v současných se uvádí jen pro úplnost/troubleshooting.

1. Otevřeme **Průzkumník Windows**, pravým tlačítkem myši klikneme na položku **Tento Počítač**, v nabídce vybereme položku **Vlastnosti**.

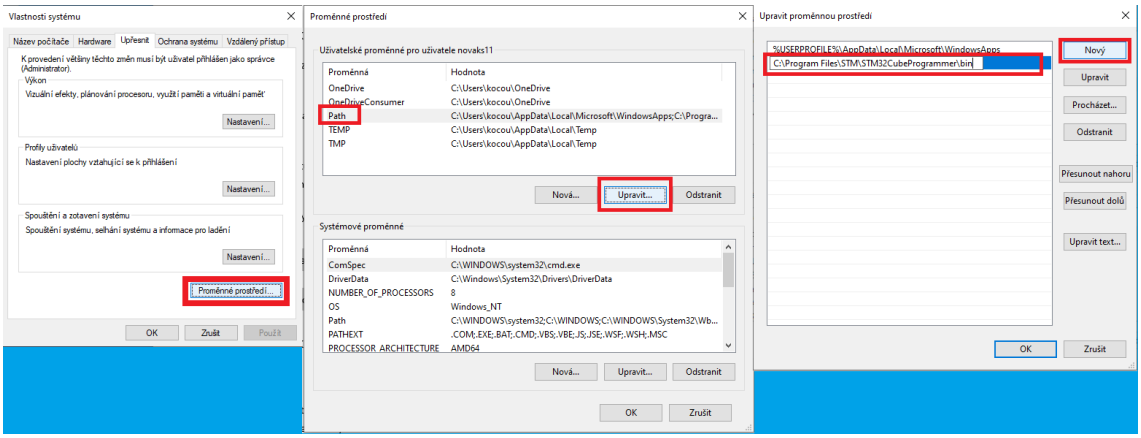
2. V okně **Vlastnosti Systému** klikneme na položku **Upřesnit nastavení systému** (obrázek 7). V okně **Vlastnosti systému** v kartě **Upřesnit** klikneme na tlačítko **Proměnné prostředí...** (obrázek 8, vlevo).

3. V tabulce okna s názvem **Uživatelské prostředí** (nahoře) vybereme proměnou **Path** a klikneme na tlačítko **Upravit...** (obrázek 8, uprostřed). V tabulce uprostřed okna vybereme volné místo a klikneme na tlačítko **Nový**. Do volného místa vložíme cestu k STM32CubeProgrammer

[27] https://chiptron.cz/articles.php?article_id=163



Obrázek 7: Okno **Vlastnosti systému**



Obrázek 8: Přidání aplikace STM32CubeProgrammer do proměnného prostředí

za kterou připíšeme `\bin` (typicky **C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin**).

4. Zavřeme všechna okna pomocí **OK** a restartujeme systém.

3.3 Programování s STM32duinem s využitím HAL a CMSIS definic (převzato z bp)

Základem balíku STM32duino je toolchain shodný s balíkem STM32Cube. Je tedy možné používat standardní definice jmen pinů (např. PA1) a periferií (např. GPIOA->ODR, dle CMSIS standardu) s možnostmi jazyka C. Díky tomu je možné přistupovat přímo k registrům periferií, případně vložit instrukce assembleru.

Možnost přístupu k registrům periferií mikrokontrolérů: Uvedený zdrojový kód bliká LED bliká LED a po každé změně stavu LED vypíše hodnotu GPIO brány s LED na UART.

```
1  uint32_t value;
2
3  void setup() {
4    // initialize serial communications at 9600 bps:
5    Serial.begin(9600);
```

```
6   pinMode(LED_BUILTIN, OUTPUT);
7 }
8
9 void loop() {
10  digitalWrite(LED_BUILTIN, LOW);
11  value = GPIOA->ODR;//GPIOA
12  Serial.println(value);
13  delay(1000);
14  digitalWrite(LED_BUILTIN, HIGH);
15  value = GPIOA->ODR;//GPIOA
16  Serial.println(value);
17  delay(1000);
18 }
```

Možnost použití kódu napsaného v assembleru: Uvedený zdrojový kód bliká LED, ale každou změnu stavu LED realizuje pomocí kódu napsaného v assembleru.

```
1  //HOWTO: https://gcc.gnu.org/onlinedocs/gcc/extensions-to-the-c-
       language-family/how-to-use-inline-assembly-language-in-c-code.
       html
2
3  void turn_on_pa5(void){
4    __asm__ volatile (//instrukce ASM
5      "MOV R0, #0x20\n\t"
6      "LDR R1, =0x48000000\n\t"
7      "LDR R2, [R1,#0x14]\n\t"
8      "ORR R2, R0\t\n"
9      "STR R2,[R1,#0x14]\n\t"
10
11   );
12 }
13 void turn_off_pa5(void){
14   __asm__ volatile (
15     "MOV R0, #0x20\n\t"
16     "EOR R0, 0xFFFFFFFF\n\t"
17     "LDR R1, =0x48000000\n\t"
18     "LDR R2, [R1,#0x14]\n\t"
19     "AND R2, R0\n\t"
20     "STR R2,[R1,#0x14]\n\t"
21   );
22 }
23
24 void setup() {
25   pinMode(LED_BUILTIN, OUTPUT);
26 }
27 void loop() {
28   turn_on_pa5();
29   delay(200);
30   turn_off_pa5();
31   delay(200);
32 }
```