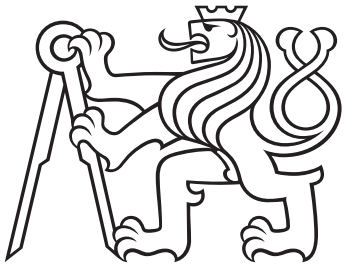
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra měření
Laboratoř videometrie



Možnosti použití balíku STM32duino

Vypracoval: Stanislav Novák, novaks11@cvut.cz
Vedoucí práce: doc. Ing. Jan Fischer, CSc
Verze: 21. listopadu 2024

0.1 Úvod

STM32duino je softwarový balík rozšiřující softwarové nástroje Arduina o možnost programování mikrokontrolérů STM32. Arduino je otevřená platforma jednodeskových počítačů založených původně na mikrokontrolérech ATmega. STM32duino začínalo jako komunitní projekt, v současnosti je velkou měrou udržovaný firmou ST.

Balík jako takový se nachází na githubu (github.com/STM32duino). Odtud je možné stáhnout zdrojové kódy a nalézt zde dokumentaci. Jako další zdroj informací existuje diskuzní fórum www.STM32duino.com, případně je možné nalézt informace i jinde (např. na community.st.com). Na webech nadšenců lze také nalézt základní návody, obvykle však nejdou výrazněji do hloubky (např. od Petra Šrámka na chiptron.cz).

Cílem této práce je lépe balík zdokumentovat z uživatelského pohledu, protože dostupná dokumentace není dostatečná a obsahuje v mnoha případech neaktuální informace.

1 Základy STM32duina, instalace

1.1 Arduino

Arduino vzniklo jako školní projekt pro výuku programování. Mikrokontroléry byly využity jako příklad praktického a pochopitelného užití programu ("hmatatelnost"pomocí LED, tlačítek a dalších prvků). Arduino bylo zprvu založeno na mikrokontrolérech ATmega osazených na deskách pouze s minimálním zapojením. Zapojení takovýchto desek se ukázalo jako velmi užitečné v praxi při vývoji elektroniky; výrobci komponent tento koncept často převzali – např. desky Nucleo od ST. Takovéto zapojení rovněž oceňují kutilové při svých projektech.

Arduino používá pro programování jazyk C (přesněji C++) rozšířené o knihovny pro jednoduché ovládání hardware (např. pro nastavení obdélníkový signál jisté frekvence na pinu funkcí s pouze dvěma argumenty). Tyto knihovny jsou často označovány jako jazyk Wiring. Ke psaní programů, kompilaci a nahrávání slouží nástroj Arduino IDE, při jeho vývoji byla upřednostňována jednoduchost ovládání.

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 je výhodnější použití verze 2 ve které je možnost ladění pomocí SWD. Zároveň, v nových verzích STM32duina byla již ukončena podpora IDE verze 1. Arduino IDE je multiplatformní, existují nativní verze pro Linux i macOS. Samotná instalace je přímočará (pro úplnost uvádíme odkaz na postup na arduino.cc). Samotné Arduino je velmi populární a v případě problému lze obvykle dohledat obrovské množství rad a návodů volně na internetu.

Dobře zpracovanou dokumentaci jazyka Wiring lze nalézt na webu arduino.cc. Podle této specifikace se mají chovat dokumentované funkce i v STM32duinu (rozumějte ve významu "emulace"chování ATmegy).

1.2 Instalace STM32duina, popis nastavení

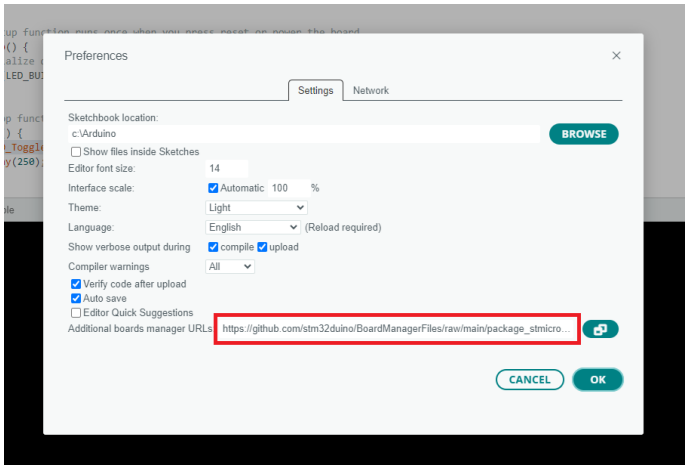
1.2.1 Instalace Arduina

Před instalací STM32duina je nutné provést instalaci Arduino IDE. Samotné IDE lze 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 je výhodnější použití verze 2, ve které je možnost ladění pomocí SWD. Zároveň, od STM32duina verze 2.8.0 je ukončena podpora IDE verze 1. Arduino IDE je multiplatformní, existují nativní verze pro Linux i macOS. Samotná instalace je přímočará (pro úplnost uvádíme odkaz na postup na arduino.cc). Samotné Arduino je velmi populární a v případě potíží lze obvykle dohledat značné množství relevantních rad a návodů volně na internetu.

1.2.2 Instalace STM32duina

Po instalaci Arduino IDE je třeba nainstalovat samotné STM32duino. Samotný postup lze nalézt na githubu (v angličtině). Pro účel této práce je v této kapitole postup taktéž uveden.

1. Nejdříve je třeba v nabídce **File** → **Preferences** nastavit adresu `https://github.com/stm32duino/BoardManagerFiles/raw/main/package_stmicroelectronics_index.json` do kolonky **Additional Boards Managers URLs:** (zvýrazněno na obrázku 1)



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

2. Dále je třeba v nabídce **Tools** → **Board:** → **Boards Manager...** (obrázek 2, vlevo) nainstalovat balík **STM32 MCU based boards:** (obrázek 2, vpravo; vhodné je využít filtr) tlačítkem **INSTALL**.

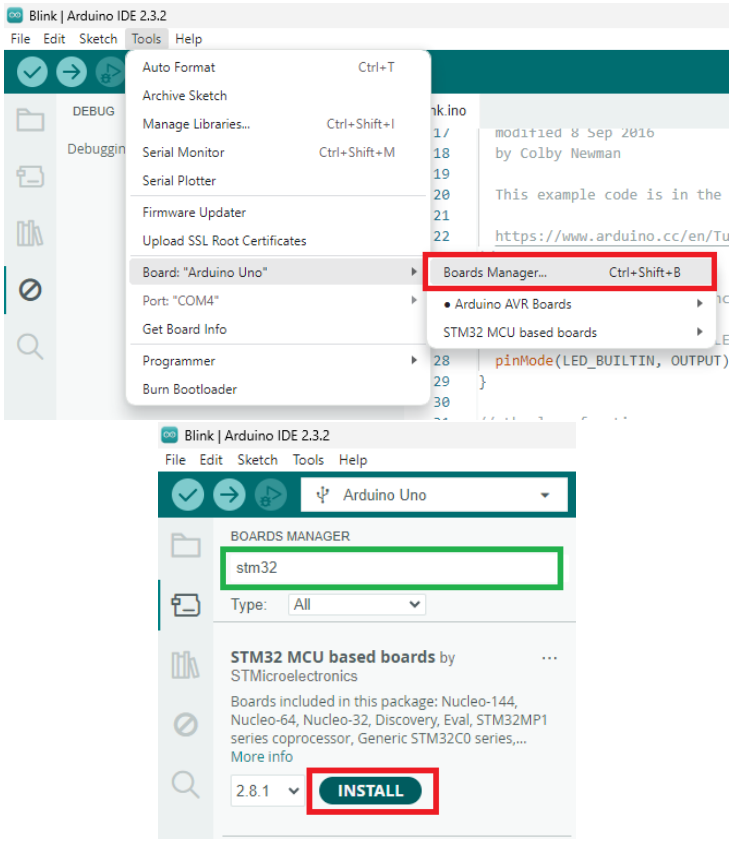
3. Po instalaci je třeba nastavit použitou vývojovou desku, a to ve dvou krocích. V prvním kroku se nastavuje skupina desek, např. Nucleo, nebo generická deska skrze **Tools** → **Board:** → **STM32 MCU based boards** (obrázek 3, vlevo; **Generic STM32G0 series** jako příklad). V druhém kroku se ze skupiny volí konkrétní deska skrze **Tools** → **Board part number:** (obrázek 3, vpravo; **STM32G030J6** jako příklad).

4. Nakonec je vhodné nastavit metodu nahrávání skrze **Tools** → **Upload Method** (závisí na konkrétní desce, která možnost je dostupná/ideální). Všechny metody nahrávání mimo **Mass storage** potřebují doinstalovat program STM32CubeProg (postup instalace tohoto zde není uveden, je přímočarý; zároveň ST poskytuje dostatečnou dokumentaci).

1.2.3 Možnosti nastavení STM32duina v Arduino IDE

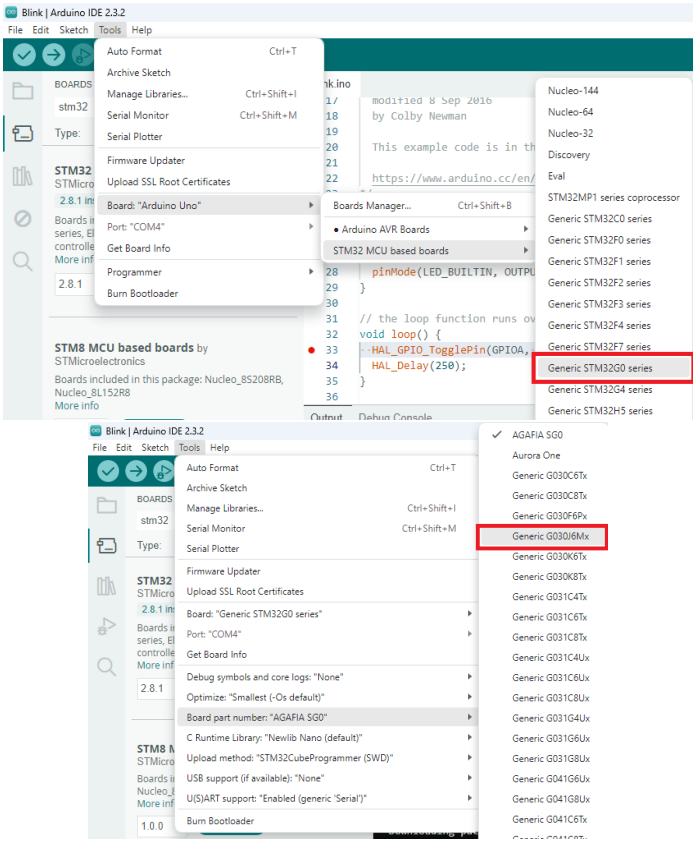
Po instalaci je vhodné představit nastavení, která STM32duino do Arduino IDE přináší. Všechna nastavení jsou součástí nabídky **Tools**. Na obrázku 4 je zobrazena nabídka s jednotlivými nastaveními. Pro úplnost je vhodné jednotlivá nastavení popsat v následujícím seznamu:

- **Board:** Zde se vybírá skupina dle cílového zařízení; takovou skupinou může být deska Nucleo, generická deska (jakákoli uživatelem navržená deska obsahující mikrokontrolér dané řady), případně jiná skupina desek. Toto nastavení již bylo ukázáno v předchozí kapitole.
- **Port:** Zde je vybírán port COM port pro komunikaci (používá se pro uživatelské výpisy, případně pro nahrávání programu). Tento je třeba typicky nastavit v případě použití UARTU, USB třídy CDC (Virtual COM port), nebo při nahrávání firmware bootloaderem periferií USART.



Obrázek 2: Instalace balíku

- **Get Board Info** Vypíše systémové informace o vybraném COM portu; v případně virtuálního COM portu ST-Linku se zobrazí jeho sériové číslo (může být v některých případech užitečné).
- **Debug symbols and core logs:** Toto nastavení je důležité při ladění ST-Linkem; určuje, zda symboly jsou součástí výstupu kompilátoru. Při ladění je vhodné nastavit na **Symbols Enabled (-g)**
- **Optimize:** Nastavení optimalizace při kompilaci (standardní nastavení optimalizace kompilátoru GCC, vhodné nastavit na **Smallest -Os**).
- **Board part number:** Zde se nastavuje konkrétní cílové zařízení pro které se má sestavit program. Možná nastavení se řídí nastavením **Board:**. Toto nastavení již bylo ukázáno v předchozí kapitole.
- **C Runtime Library:** Pokročilé nastavení knihoven C převážně relevantní pro výpisy; typicky není třeba měnit.
- **Upload method:** Určuje jakým způsobem se nahrává program. K tomuto existuje dokumentace dostupná na githubu. Metoda **Mass Storage** používá možnosti ST-Linku V2.1 pro nahrávání programu bez externích nástrojů. Ostatní metody používají externí nástroj (označené **STM32CubeProgrammer**), jednotlivá nastavení ukazují, které periferie jsou k nahrávání použity.



Obrázek 3: Nastavení desky

- **USB support (if available):** Pokud se používá periferie USB, je třeba zde nastavit, jaká třída (CDC, nebo HID) má být použita pro vložení odpovídající knihovny.
- **U(S)ART support:** Nastavení určující, zda se budou vkládat knihovny pro USART, pokud USART nebude v programu použit (např. pokud je namásto toho použito USB CDC), je vhodné vypnout (možností **Disabled (no Serial support)**) pro ušetření velikosti programu.
- **USB speed (if available):** Pokročilé nastavení pro STM32 s rychlejšími USB periferiemi než Full-speed.

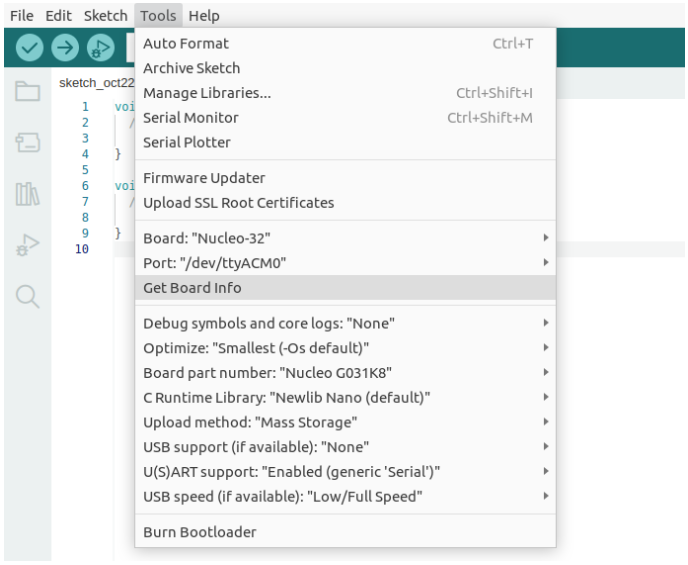
1.2.4 Prostředí Arduino IDE, příklady programů

Samotné prostředí je vcelku dobře dokumentované v internetové dokumentaci na (`https://docs.arduino.cc/software/ide/#ide-v2`), základní popis a první kroky jsou popsány na `https://docs.arduino.cc/learn/starting-guide/the-arduino-software-ide/`.

Arduino IDE má vcelku elegantně vyřešen přístup k příkladům programů, konkrétně skrze nabídku **File** → **Examples**. V nabídce jsou příklady pro lepší orientaci seskupeny, pro STM32duino jsou důležité jeho příklady ze skupiny příslušné skupině desek (např. **Examples for Generic STM32G0 series**).

1.2.5 Nahrávání programu

Před nahráváním je třeba správně nastavit metodu nahrávání (skrze **Tools** → **Upload Method**, viz předcházející kapitoly). Samotné nahrávání lze provést pomocí skrze nabídku **Sketch** →



Obrázek 4: Nastavení STM32duina

Upload, nebo intuitivněji tlačítkem šipky (→) na liště. Jako vhodné se jeví uvést možné chyby při kompilaci s krátkým vysvětlením:

- **Compilation error: ...** (typicky) Chyba v programu – program nedává z pohledu jazyka C smysl (typicky překlep, např. chybějící středník).
- **Failed uploading: no upload port provided**, nebo **Failed uploading: Cannot open port...** Nesprávně nastavený port. Port lze nastavit v **Tools** → **Port** Specifické problémy mohou nastat při použití UART-USB převodníků CH340, tyto potřebují doinstalovat ovladače (`https://www.wch-ic.com/downloads/category/30.html`). Další specifický problém nastává na Linuxu, kdy uživatel nemá přístup k portům (typicky je třeba uživatele přidat do skupiny `dialout`).
- **„...“ not found** (např. **NOD_G031K8 not found**). Pravděpodobně je zvolena metoda **Mass storage** a není připojeno tomuto odpovídající deska.
- **STM32CubeProg not found...**, STM32CubeProg není nainstalován, nebo prostředí neví ve které cestě. Řešení je popsáno na githubu.

Další uvedené chyby jsou spojeny se STM32CubeProg (jedná se přímo o výpis STM32CubeProg):

- **Error: Activating device: KO...**, tato chyba indikuje, že se podařilo otevřít COM port, ale mikrokontrolér STM32 neodpovídá. V tuto chvíli je dobré zkontrolovat, který COM port je nastaven, dále zapojení UARTU (Rx-Tx), případně ověřit, zda je zařízení v bootloaderu (ověřit stav pinu/bitu BOOT0; kompletní specifikace pro bootloader je aplikační nóta AN2606).
- **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 odpovídající zařízení nebylo nalezeno.

Pro konkrétní ověření nastavení metody nahrávání je dobré použít jednoduchý příklad, např. blikání LED z následující kapitoly.

1.3 Základní funkce a příklady

1.3.1 Digitální vstupy a výstupy

Jako tradiční první příklad se uvádí blikání LED. Blikání LED typicky ověřuje, že je možné do zařízení nahrát vlastní program. V Arduino IDE lze otevřít příklad **Basics** → **Blink** (příklady se nacházejí ve **File** → **Examples**), případně přepsat kód z této sekce (použitý pin odpovídá typické desce Nucleo-64, např. NUCLEO-G071RB). Pro použití na generické (uživatelské) desce je třeba změnit v kódu jméno pinu (nebo makro `LED_BUILTIN`) na jméno odpovídající připojené LED (v sérii s rezistorem připojeným k zemi).

```
1  void setup() {
2    pinMode(PA5, OUTPUT);
3  }
4
5  void loop() {
6    digitalWrite(PA5, HIGH);
7    delay(100);
8    digitalWrite(PA5, LOW);
9    delay(100);
10 }
```

Samotné Arduino skrývá standardní funkci `main()`, za funkce `setup()` (jednou se spustí na začátku) a `loop()` (po skončení funkce `setup()` se spouští donekonečna). Důležitým rozdílem oproti Arduinu je nutnost vždy iniciovat pin (funkcí `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()`. Jako příklad uvádíme „blikání při stisku tlačítka“ (pro NUCLEO-G071RB, ale snadno přenositelné).

```
1  void setup() {
2    pinMode(PC13, INPUT);
3    pinMode(LED_BUILTIN, OUTPUT);
4  }
5
6  void loop() {
7    if(digitalRead(PC13)==LOW){
8      digitalWrite(PA5, HIGH);
9      delay(100);
10     digitalWrite(PA5, LOW);
11     delay(100);
12   }
13 }
```

Tento příklad je validní, pokud je k tlačítku připojen externí pull-up rezistor. Pokud tlačítko při stisknutí pouze „zkratovává“ pin k zemi, je třeba zapnout pull-up rezistor na čipu (je třeba upravit druhý řádek na `pinMode(PC13, INPUT_PULLUP);`).

1.3.2 UART (sériová linka)

UART je rozšířeným rozhraním pro komunikaci mezi počítači, pro Arduino jako takové má typicky UART dvě využití :

1. Nahrávání programu
2. Uživatelské výpisy/ovládání programu

Na STM32 je typicky realizován periferií USART podporující jak asynchronní tak synchronní přenos, pro tuto chvíli se však omezme na asynchronní variantu. Pro ověření funkce výpisů je možné použít příklad **Communication** → **ASCIITable** obsažený v samotném Arduinu. Po nahrání programu a otevření sériového terminálu (v **Tools** → **Serial Monitor**) by se měl objevit vidět výpis části ASCII tabulky. Při nastavování UARTU je nutné dodržet nastavení baud rate 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 výchozí UART (viz kapitola „Definice desek“ **??**), případně příklad upravit. Při úpravě je typicky vhodné nahradit výchozí objekt `Serial` vlastním objektem `mySerial` (je možné pojmenovat téměř libovolně), kterému v inicializaci nastavíme, které piny má používat. Konkrétně příklad **ASCIITable** může vypadat následovně (porovnejte s výchozím příkladem):

```
1  HardwareSerial mySerial(PA10, PA9);//Rx,Tx
2
3  void setup() {
4    //Initialize serial and wait for port to open:
5    mySerial.begin(115200);
6    while (!mySerial) {
7      ;  // wait for the serial port to connect. Needed for native
       USB port only
8    }
9
10   // prints title with ending line break
11   mySerial.println("ASCII Table ~ Character Map");
12 }
13
14 // first visible ASCIIcharacter '!' is number 33:
15 int thisByte = 33;
16 // you can also write ASCII characters in single quotes.
17 // for example, '!' is the same as 33, so you could also use this:
18 // int thisByte = '!';
19
20 void loop() {
21   // prints value unaltered, i.e. the raw binary version of the
       byte.
22   // The mySerial Monitor interprets all bytes as ASCII, so 33, the
       first number,
23   // will show up as '!'
24   mySerial.write(thisByte);
25   // if printed last visible character '~' or 126, stop:
26   if (thisByte == 126) {  // you could also use if (thisByte ==
       '~') {
27     mySerial.write(10);
28     thisByte = 32;
29   }
30   while(mySerial.available()) mySerial.write(mySerial.read());//
       echo
31   // go on to the next character
32   thisByte++;
33   delay(100);
34 }
```

Při posílání dat do desky je typicky použita funkce `Serial.available()` (vrací kolik znaků se je

nabufferováno v paměti) a `Serial.read()` (vrací znak vyčtený z bufferu). Konkrétní použití lze pochopit z následujícího příkladu echa (přijatá data se posílají zpět):

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

Užitečné může být přečtení dokumentace k jednotlivým funkcím na `https://www.arduino.cc/reference/en/language/functions/communication/serial/`.

Úprava, která byla provedena (aby bylo možné použít pro UART jiné piny, než výchozí) využívala knihovnu **HardwareSerial** STM32duina. Dokumentace k této knihovně je dostupná na `https://github.com/STM32duino/Arduino_Core_STM32/wiki/API#hardwareserial`).

1.3.3 Analogové čtení

Měření napětí je realizováno funkcí `analogRead()`, fyzicky je k této funkci použita periferie ADC. Zde je možné se setkat s jednou zvláštností, ATmega disponovala 10-bitovým ADC, STM32 disponují obvykle 12-bitovým ADC (méně často 16- a 14-bitovým). Přesto, návratová hodnota je i pro STM32 0...1023 (realizováno odběrem 12-bitové hodnoty a bitovým posunem, (pro zpětnou kompatibilitu)). Pro nastavení 12-bitového rozlišení je třeba zavolat funkci `analogReadResolution(12)` (dle dokumentace `https://www.arduino.cc/reference/en/language/functions/analog-io/analogreadresolution/`).

Jednoduchý příklad vypisující na UART hodnoty naměřené na `PA0` může vypadat následovně:

```
1  HardwareSerial mySerial(PA3, PA2);//Rx,Tx
2  void setup() {
3    mySerial.begin(115200);
4    analogReadResolution(12);
5    pinMode(PA0, INPUT);
6  }
7
8  void loop() {
9    mySerial.print("PA0 value: ");
10   mySerial.println(analogRead(PA0));
11   delay(200);
12 }
```

1.3.4 Analogový zápis, DAC

Na pinech, které disponují DAC převodníkem (je vhodné ověřit v datasheetu), je možné zapisovat hodnotu napětí funkcí `analogWrite()`. Vstupními argumenty funkce `analogWrite()` jsou jméno pinu a hodnota (s rozsahem 0-255). Jednoduchý příklad může vypadat následovně:

```
1  void setup() {
2    pinMode(PA5, OUTPUT);
3    analogWrite(PA5, 127);
4  }
5  void loop() {}
```

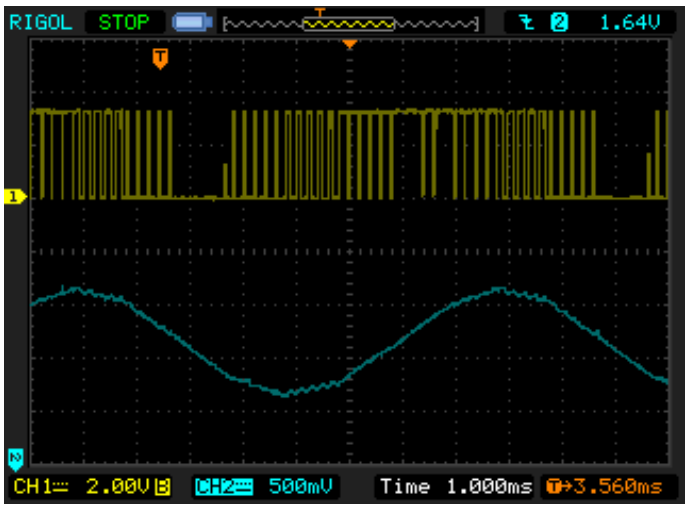
I zde se ukazuje podobná zvláštnost jako v případě ADC, DAC na ATmeze byl 8-bitový DAC, na STM32 je však typicky 12-bitový. Analogicky jako u ADC můžeme toto rozlišení upravit funkcí `analogWriteResolution(12)`.

1.3.5 Funkce `analogWrite()`, PWM

Pokud na pinu není dostupný DAC, ale nějaký čítač je (dle datasheetu), funkce `analogWrite()` na cílovém pinu použije čítač ke generování obdélníkového signálu se střední hodnotou úměrnou hodnotě vstupního argumentu (PWM, po vyfiltrování odpovídá hodnotě, kterou by generoval DAC). Jako příklad, je zde uvedeno generování průběhu podobného sinu:

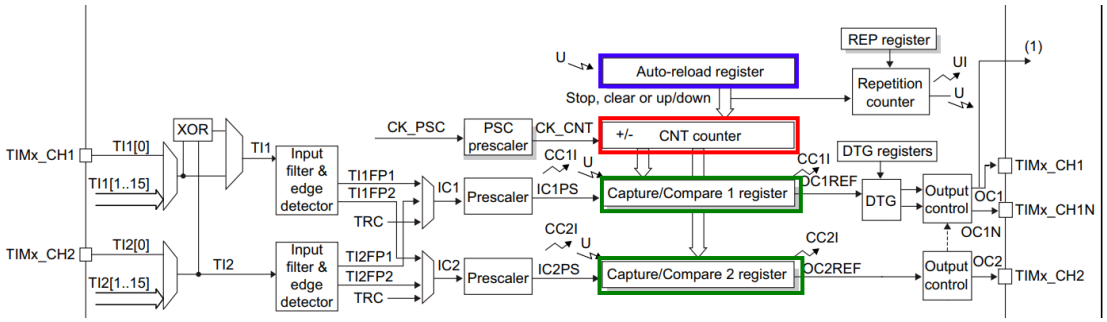
```
1  unsigned char c=0;
2  unsigned char sine[256];
3
4  void setup() {
5    for(int i=0; i<256; ++i) sine[i]=(sin(i*2*PI/255.)+1.0)*255/2;
6    //predpocitani sinu, 256 hodnot
7  }
8
9  void loop() {//perioda modulovaneho signalu 0.256 s
10   analogWrite(PA6, sine[c++]);// PWM out, given duty
11   delay(1);
12 }
```

Průběh podobný sinu lze zobrazit na osciloskopu, na obrázku 5 je vidět výstup čítače (kanál 1, žlutý) a výstup čítače vyfiltrovaný integračním RC článkem (kanál 2, modrý).



Obrázek 5: Průběh generovaného obdélníkového signálu a vyfiltrovaného

Frekvence výstupu funkce `analogWrite()` je v STM32duinu typicky 1000 Hz. Frekvenci lze změnit funkcí `analogWriteFrequency()`. Při změně frekvence na více pinech je důležité v datasheetu ověřit, které čítače jsou používány (funkce TIMx_CHy sekce 1.5.1 jak číst datasheet, zároveň je třeba brát v úvahu i negované (označené jako _N) výstupy (kanál a negovaný sdílí



Obrázek 6: Zjednodušený funkční nákres čítače, převzato z RM0444 (upraveno)

hodnotu Capture/Compare registru)). Z principu jeden čítač s více kanály může generovat na každém kanálu generovat obdélníkové signály různé střídy, ale všechny kanály sdílí jednu hodnotu frekvence (viz obrázek 6). Frekvence je odvozena od hodnoty zapsané Auto-reload registru (na obrázku 6 zvýrazněno modře), zároveň hodnoty střídy kanálů jsou odvozeny od hodnot v Capture/Compare registrech (na obrázku 6 zvýrazněno zeleně). Preferovanou alternativou k funkci `analogWriteFrequency()` může být použití knihovny **HardwareTimer** (viz dále).

Pokud na pinu není DAC, ani čítač, zapíše se hodnota digitálně (prahována oproti polovině číselného rozsahu vstupního argumentu).

1.3.6 Funkce `tone()`

Další funkcí Arduina generující obdélníkový signál je `tone()`. Tato funkce je uváděna pouze pro úplnost, jedná se o generování obdélníkového signálu se střídou 1:1, zadané frekvence. Funkce `tone()` má jedno zásadní omezení (vyplývající z Arduina), je možné generovat v jednu chvíli signál pouze na jednom pinu, což vždy nepostačuje. Zároveň funkce tone není založená na kanálu čítače daného pinu, ale na jiném čítači, který realizuje danou funkcionalitu pomocí přerušení, což je méně přesné typicky při vysokých frekvencích. Přestože je tato funkce často používána v příkladech Arduina, v STM32duinu je velmi doporučené použít jiné funkce, konkrétně knihovnu **HardwareTimer**.

1.3.7 Knihovna HardwareTimer

Knihovna **HardwareTimer** STM32duina nabízí pokročilejší možnosti nastavení čítačů STM32, než Arduino. Na githubu lze nalézt základní dokumentaci, ve které jsou vypsány funkce třídy a některé příklady. Jednoduchý příklad s PWM s knihovnou může vypadat takto:

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

Čítač je však periferie, kterou lze využít i jinak, např. ke generování periodické události (přerušení). Toho, dosáhneme tak, že k čítači „připneme“ funkcí `attachInterrupt()` funkci, tzv. callback (konkrétní realizace mechanizmu převzata z Arduina). Příklad, který tlakovým mechanismem bliká LED lze vidět níže:

```
1 HardwareTimer htim2 = HardwareTimer(TIM2);
```

```
2
3  callback_function_t callback(){
4    if(digitalRead(PA5)==LOW) digitalWrite(PA5,HIGH);
5    else digitalWrite(PA5,LOW);
6    return 0;
7  }
8
9  void setup() {
10   pinMode(PA5, OUTPUT);
11   htim2.setOverflow(10, HERTZ_FORMAT);//10 Hz frekvence volani
       callbacku
12   htim2.attachInterrupt(callback);
13   htim2.resume();//spustit citac
14 }
15 void loop(){}
```

Jelikož STM32 disponuje čítači jinými, než původní ATmega, typicky nelze použít knihovny Arduina (např. oblíbená čítačová knihovna **TimerOne** napsána přímo nad registry čítače ATmegy).

1.3.8 SPI, I2C, knihovny

SPI, I2C jsou rozšířená rozhraní pro komunikaci mezi mikropočítači. Arduino obě tato rozhraní podporuje stejně tak i STM32duino. Zmíněná rozhraní jsou velmi často používaná pro připojení rozličných modulů a zařízení, např. externích pamětí, lcd, chytrých senzorů.... Pro většinu takovýchto modulů a zařízení již existují knihovny pro Arduino a obrovským benefitem STM32duina je, že většina těchto knihoven je použitelná i s ním.

Rozhraní SPI

V typické formě používá čtyři signály, MOSI, MISO, SCK, CS. Jedná se o synchronní rozhraní, hodiny generuje uzel master signálem SCK, signálem MOSI posílá data master a ve stejnou chvíli signálem MISO posílá data slave. Signálem CS (volitelný) určuje uzel master, který slave je aktivní.

Použití SPI v Arduinu není v zásadě složité, v první řadě je třeba vložit knihovnu `<SPI.h>`. Konkrétní „výměna bytů“ je realizována funkcí `transfer()`, kdy vstupním argument je odeslaný byte, zároveň přijatý byte je vrácen (jako návratová hodnota). Obdobně jako pro UART je vhodné si pro SPI vytvořit vlastní objekt `mySPI` odpovídající třídy, kdy jsou definovány použité piny (nutné je definovat validní hodnoty MOSI, MISO, SCK pro periferii (viz sekce 1.5.1 jak číst datasheet); pro CS je možné použít makro `PNUM_NOT_DEFINED`). Následující příklad ukazuje komunikaci po SPI pro posuvný registr 74x164 (MOSI pin (připojen k pinu DSA; na pinu DSB trvale logická 1), a SCK (připojen k pinu CP)). Na výstupy posuvného registru (piny Q1-7) můžeme pro demonstraci připojit LED:

```
1  #include <SPI.h>
2  SPIClass mySPI(PA4, PA3, PA0, PNUM_NOT_DEFINED);//MOSI, MISO, SCK,
     CS
3  void setup() {
4    mySPI.begin();//inicializace
5    mySPI.beginTransaction(SPISettings(250000, MSBFIRST, SPI_MODE3));
       ///freq 250k, poradi, CPOL=1 CPHA=1
6  }
7
8  void loop() {
9    mySPI.transfer(0x55,SPI_TRANSMITONLY);//odesilani 0x55,bez prijmu
```

```
10    delay(1);
11  }
```

V předchozím příkladě funkcí beginTransaction je nastavena frekvence přenosu (250 kHz, nejpomalejší možná při 64MHz hodinách periferie (předděličkou 64M/256)), pořadí bitů (první nejvíce významný) a mód (mód 3 – CPOL=1, CPHA=1).

Arduino založené na ATmeze častokráte „trpělo“ na nedostatek periferií pro komunikaci s posuvnými registry, proto existuje softwarová emulace, funkce `shiftOut()`, kterou lze použít i v STM32duinu. Následující příklad demonstruje binární čítač (softwarový), jehož hodnota je postupně posílána do posuvného registru:

```
1  void setup() {
2    pinMode(PA0,OUTPUT);
3    pinMode(PA4,OUTPUT); //inicializace
4  }
5  uint8_t cnt = 0;//promenna realizujici "citac"
6  void loop() {
7    shiftOut(PA4, PA0, MSBFIRST, cnt++); //poslat cnt a inkrementovat
8    delay(100);
9  }
```

Implementace by měla respektovat původní možnosti Arduina, odpovídající dokumentaci lze nalézt na docs.arduino.cc.

I2C

I2C je rozhraní používající dva signály SDA (data), SCK (hodiny). Rozhraní obsluhuje jeden master uzel, který komunikuje s ostatními zařízeními na základě jejich adres. Podmínkou pro správné fungování je tzv. „drátový součin“ na obou signálech, typicky realizovaný pull-up rezistory.

V Arduinu je při použití třeba vložit knihovnu `Wire.h` a vytvořit objekt pro rozhraní (`myI2C`, zde specifikujeme použité piny). Objekt je třeba iniciovat funkcí `begin()`, následně je možné nastavit frekvenci přenosu funkcí `setClock()` (funkce je schopna nastavit pouze hodnoty frekvencí dle standardu (omezení Arduina)). Zasílání na rozhraní I2C začíná funkcí `beginTransmission()`, samotné vysílání je realizováno funkcí `write()`, zasílání je zakončeno funkcí `endTransmission()` (funkce vrací 0 přenos proběhl v pořádku (bit ACK 0)). Příjem je realizovaný bufferováním – nejdříve je funkcí `requestFrom()` „zažádáno“ o zadaný počet bytů, následně funkcí `read()` jsou data vyčtena z bufferu.

První příklad (I2C scan) demonstruje způsob, jak ověřit, že na rozhraní odpovídá nějaké zařízení na nějaké adrese (pokud nastavuje bit ACK do 0):

```
1  #include <Wire.h>
2  TwoWire myI2C(PB7, PB6);  //SDA,SCL
3  HardwareSerial mySerial(PA3, PA2);//Rx,Tx
4
5  void setup() {
6    mySerial.begin(9600);
7    myI2C.begin();
8    myI2C.setClock(100000);
9    for (uint8_t adresa = 1; adresa < 127; ++adresa) {// vsechny
       adresy
10     myI2C.beginTransmission(adresa);
11     if (!myI2C.endTransmission()) {//ACK LOW
12       mySerial.print("Nalezeno: 0x");
13       mySerial.println(adresa, HEX);
```

```
14      }
15    }
16  }
17  void loop() {}
```

Druhý příklad ukazuje zápis a čtení, konkrétně je použit MEMS senzor MPU6050. MPU6050 má mít dle datasheetu v registru na adrese 0x75 zapsánu hodnotu 0x68. Příklad ukazuje, jak tuto hodnotu vyčíst a potvrdit (posláním textu na UART).

```
1  #include <Wire.h>
2  TwoWire myI2C(PB7, PB6);  //SDA,SCL
3  HardwareSerial mySerial(PA3, PA2);//Rx,Tx
4
5  void setup() {
6    mySerial.begin(9600);
7    myI2C.begin();
8    myI2C.setClock(100000);//nastaveni frekvence
9    uint8_t address=0x68;
10
11   myI2C.beginTransmission(address);
12   myI2C.write(0x75);//posilani 1 bytu (adresy-registru)
13   myI2C.endTransmission();
14
15   myI2C.requestFrom(address,1);//"zadost" o cteni 1 bytu (z adresy
      drive zaslane)
16   int devID = myI2C.read();//cteni 1 bytu z bufferu
17   if(devID == 0x68) mySerial.println("devID OK");// vypis na UART
      pokud vyctena hodnota odpovida datasheetu
18 }
19 void loop() {}
```

Jiným příkladem může být použití IO expandéru PCF8574T (zápis hodnoty 0xf, spodní 4 bity 1):

```
1  #include <Wire.h>
2  TwoWire myI2C(PB7, PB6);  //SDA,SCL
3
4  void setup() {
5    myI2C.begin();
6    myI2C.setClock(100000);//nastaveni frekvence
7    uint8_t address=0x20;//treba overit, lisi se dle nastaveni a
      vyrobce
8    myI2C.beginTransmission(address);
9    myI2C.write(0xf);//posilani 1 bytu honoty IO
10   myI2C.endTransmission();
11 }
12 void loop() {}
```

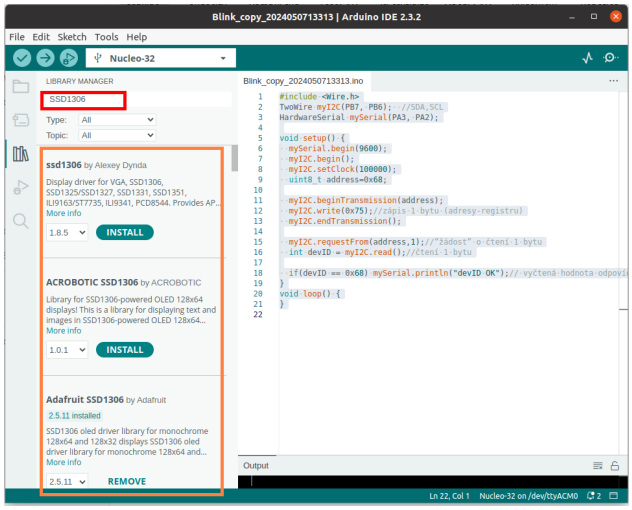
Na `https://github.com/STM32duino/Arduino_Core_STM32/wiki/API#i2C`githubu lze nalézt krátkou dokumentaci. Původní možnosti zůstávají dle specifikace Arduina, dokumentace Arduina se nachází na docs.arduino.cc.

Knihovny SPI, I2C

Dnes však není obvykle potřeba SPI a I2C komunikaci programovat na nízké úrovni, jak ukazovaly předchozí podkapitoly. Pro většinu dostupných zařízení již existují knihovny pro Arduino,

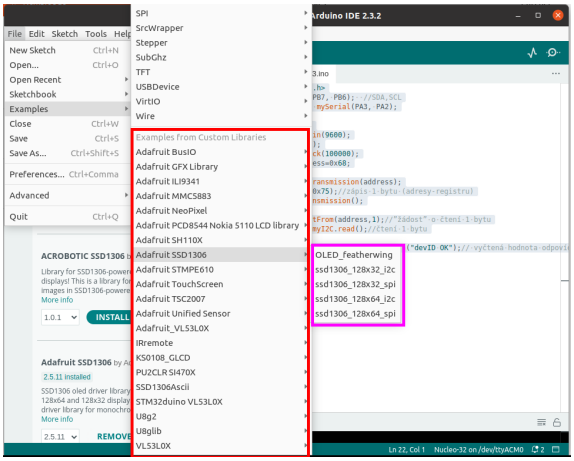
které je velmi často možné použít jak jsou i s STM32duinem. Tyto knihovny jsou dostupné buď přímo z Arduino IDE, nebo jako .zip archivy (informace o knihovnách je možné nalézt na `https://www.arduino.cc/reference/en/libraries/`).

Instalace knihoven z Arduino IDE je přímočará v nabídce **Tools** → **Manage Libraries...** otevřeme nabídku a pomocí filtru vyhledáme knihovnu a tlačítkem **INSTALL** nainstalujeme (obrázek 7).



Obrázek 7: Výběr knihoven

Pokud pomocí Arduino IDE nenalezneme vhodnou knihovnu, je možné nainstalovat knihovnu i jako archiv pomocí **Sketch** → **Include Library** → **Add .ZIP Library...**. Po instalaci je možné otevřít příklady knihoven v nabídce **File**>**Examples** (obrázek 8), ty mohou být užitečné, když integrujeme nějaký modul do našeho projektu (např. pro prvotní ověření funkčnosti).



Obrázek 8: Výběr příkladu příslušného knihovně

1.3.9 USB

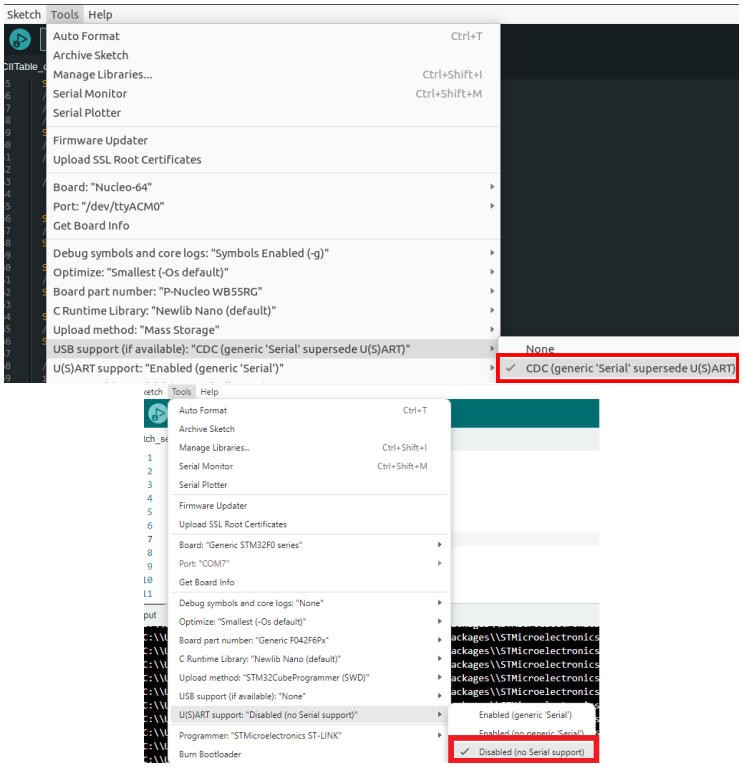
Většina mikrokontrolérů rodiny STM32 disponuje periferií USB (z řady STM32G0 pouze zařízení STM32G0B/C). STM32duino nabízí možnost využít knihovny Arduina pro USB, konkrétně

pro dvě třídy – CDC a HID.

Virtual COM port

V některých případech může být vhodným řešením použít periferii USB jako COM port místo periferie USART jako tzv. „Virtual COM Port“.

Funkcionalitu lze nastavit skrze **Tools → USB Support → CDC generic supersede...**, zobrazeno na obrázku 9, vlevo. Po tomto nastavení jsou data výchozí třídy `Serial` předávána USB.



Obrázek 9: Nastavení Virtual COM Port

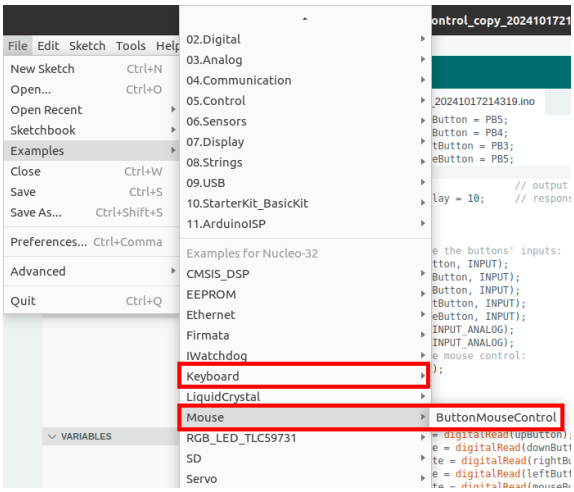
Po nahrání firmware by se deska při připojení k USB měla enumerovat jako COM port. Toto řešení má v sobě jistou eleganci (není třeba UART-USB převodník), přesto je zde jistý nedostatek, USB knihovna pro své fungování potřebuje určité množství paměti (okolo 10 kB Flash a 3 kB RAM), které je zejména u mikrokontrolérů s menším množstvím paměti citelné.

Pokud je použít pouze VCP a žádný USART, dává smysl nevkládat knihovny pro tuto periferii (skrze **Tools → U(S)ART support: → Disabled...**) a tím ušetřit na velikosti programu, na obrázku 9, vpravo je ukázáno, jak.

Třída HID, myš, klávesnice

Periferii USB lze pomocí knihoven Arduina použít jako zařízení třídy HID (Human Interface Device), to může být konkrétně klávesnice nebo myš. Chování těchto knihoven má být totožně s Arduinem, při použití této knihovny je dobré vycházet z příkladů **ButtonMouseControl** a **KeyboardMessage** (skrze **File → Examples → Examples for Nucleo-32**, pro rychlou orientaci může pomoci obrázek 10).

Použití příkladu pro myš je přímočaré, pro základní ověření stačí nastavit, které piny jsou použity ve funkci tlačítek. Místo tlačítek lze samozřejmě použít jiné senzory, např. potenciometry pro osy x a y.



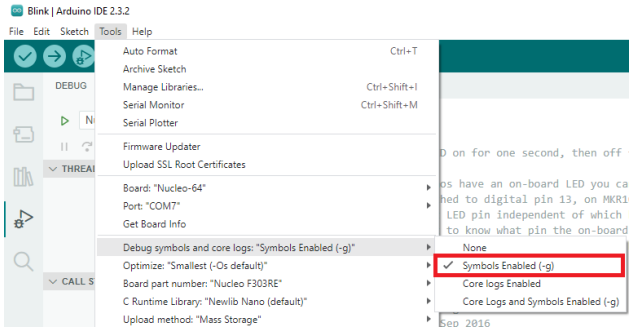
Obrázek 10: Příklady USB

Knihovna pro klávesnici je komplexnější než pro myš, její popis je dobré si před použitím nastudovat `https://docs.arduino.cc/language-reference/en/functions/usb/Keyboard/`. Konkrétně se rozlišuje `write()` (pošle se symbol) a `push()`/`release()`, kdy je klávesa stisknuta/uvolněna.

Pro zadávání textů se knihovna používá vcelku přímočaře (funkci `print()` se předávají texty). Při pokročilejších operacích (např. použití klávesy PrintScreen), lze použít některé ID symbolů dle specifikace (HID Usage Tables). V tom případě je však k ID symbolu přičíst offset 136 a poslat funkcí `write()` (příklad PrintScreen – kód 0x46+136=206). Omezující je však, že vstupní argument je `uint8_t` a tedy některé znaky tak není možné použít (pro ID vyšší než 0x73).

1.3.10 Možnosti debugu ve STM32duinu

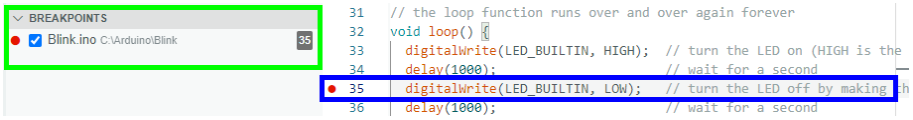
V návaznosti na možnosti Arduina, STM32duino nabízí možnosti debugu po SWD využívající ST-Link. To nabízí možnost program zastavovat, krokovat a vyčítat hodnoty registrů. Pro připojení ST-Linkem je třeba nejdříve nahrát zkompilovaný program do desky (např. ikonou **Upload** (→)), zároveň tento program musí obsahovat symboly debugu (nastavení viz obrázek 11). Následné připojení lze provést pomocí ikony **Start Debugging** na řádku ikon.



Obrázek 11: Nastavení symbolů debugu do výstupu kompilátoru

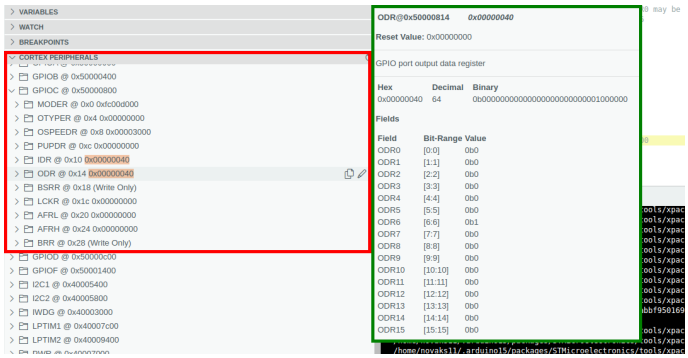
Debug je řešen pomocí OpenOCD (v kombinaci se standardním GDB) a nabízí obvyklé možnosti. Obvyklými možnostmi rozumíme breakpointy (program se zastaví v místě umístěného červeného bodu (na obrázku 12 modře zvýrazněno); všechny aktivní breakpointy zobrazeny v seznamu

(na obrázku 12 zvýrazněno zeleně)), krokování (pomocí obvyklých ovládacích prvků (na obrázku 13)) a watch výrazy. Navíc je zde využito standardních (dle CMSIS) .svd souborů, díky kterým je možné přehledně zobrazit obsah registrů zařízení (sekce „Cortex peripherals“ v okně, zobrazeno na obrázku 14).



Obrázek 12: Ilustrace použití breakpointu

Obrázek 13: Ovládací prvky debugu, zleva **Continue**/**Pause** (spustí/pozastaví běh programu), **Step Over** (vykoná aktuální řádek a zastaví na dalším), **Step Into** (zastaví uvnitř funkce volané na řádce), **Step Out** (zastaví při návratu z aktuálně vykonávané funkce), **Restart** (vyvolá reset), **Stop** (ukončí debug)

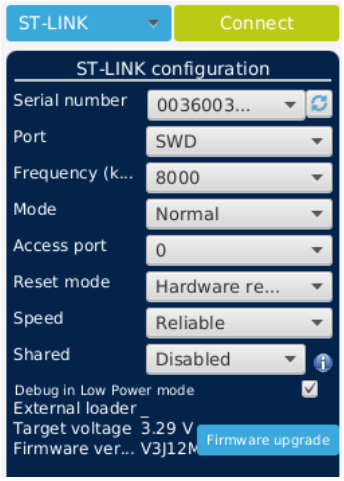


Obrázek 14: Výpis registrů periferií

1.4 Pokročilé možnosti a příklady

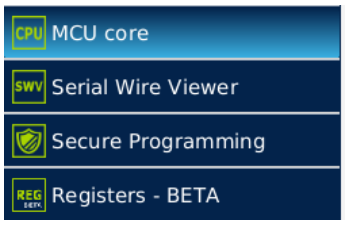
1.4.1 Pokročilé možnosti debugu

Debug v nástroji STM32Programmer V návaznosti na debug v STM32duino, se jeví jako vhodné na tomto místě zmínit možnosti debugu v programu STM32CubeProg. Zde probíhá debug bez znalosti struktury programu, kdy jsou vyčítány registry mikrokontroléru, zároveň jsou zde možnosti, jak do registrů zapisovat, jádro zastavovat a další. V první řadě je dobré se zaměřit na nastavení připojení debuggeru, konkrétně na položku **Mode**.



Mode **Normal** je vhodný pro většinu operací. **Normal** má jeden nedostatek, při připojování typicky vyvolává reset, což v některých případech nemusí být žádoucí; v takovém případě je lepší použít **Mode Hot Plug**, který reset nevyvolává. Posledním modem je **Under reset**, který reset nechává aktivní (to může být výhodné), tento mode však není vhodný, pokud je třeba program z nástroje debuggovat (program se nedostane z reset handleru).

Samotný debug se ovládá ze tří karet (**MCU core**, **Serial Wire Viewer**, **Registers**):

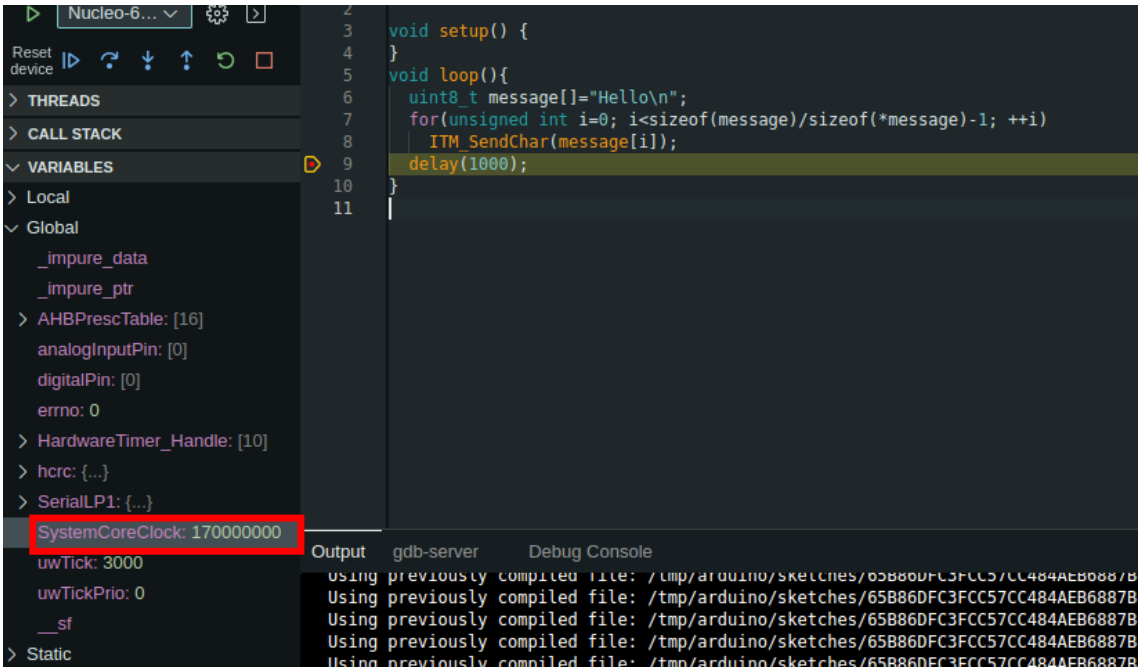


V kartě **MCU core** je možné sledovat a upravovat obsah registrů jádra (případně jádro zastavovat, krokovat, resetovat), v kartě **Serial Wire Viewer** je možné sledovat výpisy po SWO (Cortex-M0/M0+ nepodporuje), v kartě **Registers** lze vyčítat a měnit obsah registrů periferií.

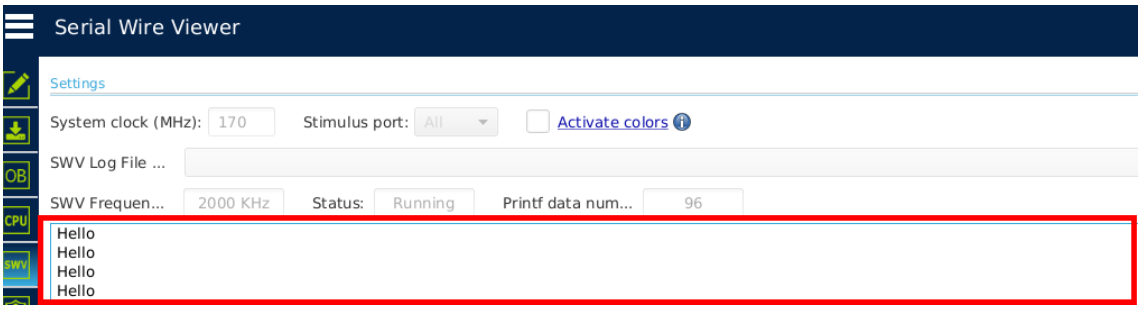
Použití SWO SWO je možností, jako pomocí rozhraní SWD (rozhraní jádra pro debug) realizovat výpisy do konzoli (typicky při ladění). Výhodou SWO je nezávislost na periferiích (typicky UART) mikrokontroléru. Použití SWO demonstruje následující příklad napsaný pomocí STM32duina:

```
1  void setup() {
2  }
3  void loop(){
4    uint8_t message[]="Hello\n";
5    for(unsigned int i=0; i<sizeof(message)/sizeof(*message)-1; ++i)
6      ITM_SendChar(message[i]);
7    delay(1000);
8  }
```

Program nahrajeme a připojíme se ST-linkem z Arduino IDE. Pro STM32CubeProg budeme potřebovat znát frekvenci hodin, ty vyčteme jako proměnnou `SystemCoreClock`, zde 170 MHz.



Odpojíme (pomocí tlačítka **Stop**) ST-link od Arduino IDE a připojíme se z STM32CubeProg, zadáme frekvenci hodin a spustíme zobrazení dat; v textovém poli bychom měli vidět výpisy:



Podrobnější popis všech funkcionalit STM32CubeProg lze nalézt v UM2237. Další pokročilé možnosti debugu jsou popsány v aplikační nótě AN4989 (pospáni pro standardní vývojové nastroje).

Možnosti GDB pomocí okna Debug Console

GDB je standardní debugovací nástroj. Při otevření okna **Debug Console** (ikona >) je možné zadávat přímo příkazy. Pomocí příkazů lze dosáhnout dále, než pouze grafickým rozhraní z Arudino IDE, např. je možné krokovat kód po instrukcích příkazem `ni` (ne po řádcích v C). Použití příkazu `ni` je výhodné u příkladů s inline assemblerem, kdy z pohledu C je celý kód v inline assembleru „jedním řádkem“.

ST-link

Pro úplnost dává smysl se dodatečně se zmínit o ST-linku a jeho možnostech. ST-link je proprietární debugger společnosti ST, lze použít výlučně na ladění mikrokontrolérů STM32, případně STM8. ST-Link jako takový je dostupný v několika verzích založených na různých mikrokontrolérech STM32, zde uvádíme některé:

- **ST-Link V1**: Stará verze debuggeru (současné softwarové nástroje již nepodporují) založená na STM32F103C8/STM32F103CB. Lze ji nalézt na nejstarších vývojových deskách STM32 (např. STM32VLDISCOVERY). S verzemi V2 sdílí mikrokontrolér, ale liší se zapojením. Teoreticky lze drátkování a přehráním firmware „upgradovat“ na V2 (po internetu kolují návody, od kterých se ST distancuje).
- **ST-Link V2.0** – aktuální verze debuggeru vhodná pro většinu aplikací:
  - **ST-Link V2.0**: Buď jako samostatný debugger (obsahuje ESD ochrany vstupů), případně na některých převážně starších vývojových deskách (namísto verze V2.1). Samostatný debugger, typicky podporuje debug STM32 po SWD a JTAG, dále STM8 po SWIM. Zařízení je založeno na mikrokontrolérech STM32F103C8/STM32F103CB.
  - **ST-Link V2.0 ISOL**: Zařízení je založeno na mikrokontrolérech STM32F103C8/STM32F103CB, oproti V2 obsahuje izolaci signálů pomocí optočlenů (typicky vhodné pro některé výkonové aplikace). Dobré je přečíst si před zapojením návod UM1075, konkrétně dbát na zapojení zemního pinu (oproti „neizolované“ variantě). Měl by mít stejné možnosti jako „neizolovaná“ V2.0.
  - **ST-Link V2.1**: Zařízení je založeno na mikrokontrolérech STM32F103CB, typicky se nachází na vývojových deskách. Oproti V2.0 nepodporuje JTAG ani SWIM. Oproti poskytuje V2.0 navíc „Virtual COM port“ (funkce USB/UART převoníku, pro uživatelské výpisy) a MCO (8MHz hodinový výstup).
- **ST-Link V3** – oproti V2 poskytující více možností (rychlejší SWD rozhraní, Access porty pro složitější struktury jader):
  - **STLINK-V3SET**: Zařízení poskytující velké množství rozhraní (debug STM32 SWD/JTAG, STM8 SWIM, VCP, CAN, „bridge“...). Debugger je rozšiřitelný o připojitelné („sandwitchovatelné“) desky, např. o desku s optočleny pro izolaci. Debugger je založen na mikrokontroléru STM32F723IE.
  - **STLINK-V3MINI/STLINK-V3MINIE/STLINK-V3MODS**: Debuggery je založen na mikrokontroléru STM32F723IE, jedná se o minimalistické verze pro uživatelské aplikace (pomocí konektoru/jako modul při-pájitelný do uživatelské aplikace).
  - **STLINK-V3PWR**: Zařízení založený na STM32H7 poskytující mimo debugu možnost měření spotřeb (i malých proudů). Měření spotřeb je analogické zařízení X-NUCLEO-LPM01A s kterým sdílí software pro měření.

Vývojové desky často nabízejí možnost použití ST-Linku V2.1 pro cílovou aplikaci (zmíněno např. v návodu UM1724). Mimo těchto existují i neoriginální verze, od které se ST distancuje:

- **ST-link V2 „dongle“** Jedná se o minimální zapojení ST-Linku V2 (obrázek) používajících typicky klon mikrokontroléru STM32F103CB. V současnosti (2024) lze tyto upgradovat na aktuální firmware a dokonce používat se softwarovými nástroji od ST (v některých případech mohou nastávat problémy, typicky založené na problémech použitých klonů mikrokontrolérů). Zajímavé, že existuje více verzí DPS s různým rozložením vývodů (často se lze setkat s kabelovými vývody na uživatelských deskách, které díky tomu nemusí být vzájemně kompatibilní).

Další možnosti debugu

Mimo nastíněných možností je možné používat i tradičnější metody k odlaďování programu (uvedeno pro úplnost). První jsou výpisy pomocí periferie UART (viz kapitola 1.3.2), nebo pomocí SWO (kapitola 1.4.1). Obsah výpisu závisí na uživateli, typické je vypisovat hodnoty proměnných, případně registrů mikrokontroléru.

Druhou tradiční metodou může být použití LED a tlačítka, velmi dobře zpracovaná je problematika pro mbed (mechanismus lze aplikovat samozřejmě i jinde) v bakalářské práci L. Bielesch, kapitola 6.1. Základní myšlenkou je „zastavování-se“ v programu na určitých místech pro lokalizaci problému (mechanismus velmi podobný breakpointu). Toho lze dosáhnout tak, že na určitých místech v programu jsou umístěna místa kam je vložena část kódu, která rozbliká LED. Takto část kódu (blikání LED) je vykonávána dokud není stisknuto tlačítko, příklad takovéto části kódu je uveden níže.

```
1   while(digitalRead(PC13)==HIGH){//cekani na stisk tlacitka
2     digitalWrite(PA5, HIGH);
3     delay(100);
4     digitalWrite(PA5, LOW);
5     delay(100);
6   }
7   while(digitalRead(PC13)==LOW);//cekani na uvolneni tlacitka
```

1.4.2 Struktura balíku STM32duino

Adresářová struktura souborů balíku

Pro hlubší pochopení funkcí STM32duina a pro možnosti přizpůsobení programů je vhodné prozkoumat strukturu balíku. Konkrétně se jedná o popis adresářové struktury a rolí jednotlivých souborů. Soubory balíku lze nalézt v adresáři **%loclappdata%/Arduino15/packages/STMicroelectronics/**

- **hardware/stm32/2.8.1** – základní adresář balíku (obsah odpovídá gitu Arduino_Core_STM32)
  - **CI**, **cmake**, **debugger**, **tools** – složky obsahují skripty zajišťující funkce balíku, např. build, debug ...
  - **cores** – složka obsahuje implementaci „jazyka Wiring“ pro STM32
  - **libraries** – složka obsahuje knihovny portované z Arduina (např. **USBDevice**, **Wire**)

  - **system** – složka obsahuje drivery (CMSIS a HAL) a middleware STM32 (knihovny pro USB). Dále jsou zde pro každou řadu STM32 nastavení driverů.
  - **variants** – dle řad uspořádané definice desek (důležité, popsáno dále)
  - **boards.txt** – soubor obsahující parametry definovaných desek. Desky obsažené v tomto souboru se zobrazí v UI Arduina.
  - **keywords.txt** – seznam rezervovaných názvů (pinu, periferii...) pro kontrolu při buildu
  - **package.json** – data pro zobrazeni při instalaci z **Manažera desek**
  - **platform.txt** – nastavení compileru, buildu a podobně
- **tools** složka obsahující převážně podpůrné soubory
  - **CMSIS** – CMSIS knihovny
  - **STM32_SVD** – SVD definice pro debug
  - **STM32Tools** – nástroje převážně pro nahrávání programu
  - **xpack-arm-none-eabi-gcc** – GCC pro kompilaci
  - **xpack-openocd** – OpenOCD pro debug

Pro konkrétní úpravy může být výhodné upravovat především obsah složky **variants**. Při specifických případech může dávat smysl opravovat obsah složek nastavení ve složce **system**, knihoven **libraries** a implementace „jazyka Wiring“ **cores**. .

Definice desek ve složce variants, soubor boards.txt, a soubory jednotlivých variant

V základním adresáři (**hardware/stm32/2.8.1**) se typicky nachází soubor **boards.txt** obsahující základní definici desek. Desky obsažené v souboru se zobrazí v nabídce v Arduino IDE jako možná cílová deska. Obdobně jako v Arduino IDE jsou definovány skupiny desek (např. „Nucleo_32“, nebo „GenG0“ a dále jednotlivé desky (např. „Nucleo G031K8“). Součástí definice desky je jméno, max. velikosti flash a RAM (`maximum_size`, `maximum_data_size`) pro build, cestu k „**variantě**“ (rozvinuto dále) a odpovídající `.svd` soubor. Dále zde lze nalézt flagy kompilátoru specifické pro danou desku.

V podadresáři `variants` lze nalézt jednotlivé **varianty**. „Varianta“ obsahuje několik souborů, příkladem takového může být `/variants/STM32G4xx/ G431R(6-8-B)(I-T)_G441RB(I-T)` (analogicky pro jinou desku). Ve „variantě“ lze nalézt několik souborů, jejich účel je následující:

- **boards_entry.txt** – Soubor s definicí, ze které lze vycházet při definici vlastní desky v souboru **boards.txt**
- **CMakeLists.txt** – Soubor sloužící při buildu k automatickému vkládání zdrojových kódů (CMake).
- **generic_clock.c** – Soubor typicky obsahuje funkci inicializace hodin. Pokud je třeba upravit lze použít program STM32CubeMX (nastavit hodny zde a odpovídající vygenerovanou část kódu pouze zkopírovat).
- **ldscript.ld** — Linkerscript, standardní GCC (`https://ftp.gnu.org/old-gnu/Manuals/ld-2.9.1/html_chapter/ld_3.html`). Lze použít linkerscript generovaný programem STM32CubeMX s úpravami (je vhodné nastavit velkosti pamětí jako makra (`maximum_size`, `maximum_data_size`) ze souboru **boards.txt**).
- **PeripheralPins.c**: Soubor obsahuje definice (formou polí) s „alternativními funkce“ jednotlivých pinů. V tomto souboru je třeba ověřit, jaké jméno pinu využít při využití určité funkce dle datasheetu (viz dále následující kapitola).

- **PinNamesVar.h** – Soubor obsahuje definice zajišťující alternativní funkce pinů.
- **variant_NUCLEO_G431RB.cpp**, **variant_generic.cpp**: Soubor obsahuje definice obsahující přiřazení původních pinů Arduina (`D0-Dx` a `A0-Ay`) pinům mikrokontroléru (formou pole). Dále se zde může nacházet funkce nastavující hodiny (alternativně k souboru **generic_clock.c**).
- **variant_NUCLEO_G431RB.h**, **variant_generic.h**: Soubor obsahuje definice přiřazení původních pinů Arduina (`D0-Dx` a `A0-Ay`) pinům mikrokontroléru (formou maker). Dále jsou zde uvedeny definice pro výchozí `Serial, I2C, SPI`, timer pro `tone` a další (`BUILTIN_LED` na desce).

Soubor PeripheralPins.c

Soubor **PeripheralPins.c** definice alternativních funkcí pinů. Definice jsou realizovány formou pole, např. pro piny ADC je definováno pole `PinMap_ADC[]`. Z těchto polí lze odvozovat, které periferie jsou pro danou funkcionalitu použity. Pro některé piny může nastat, že je možné danou funkcionalitu realizovat více periferiemi (např. pokud je na mikrokontroléru více ADC, na pinu PA0 lze číst kanálem 1 všech), je pin uveden vícekrát (např. PA_0 a PA_0_ALT1). V tomto případě je třeba si dát pozor, které piny jsou v kódu využívány (zkontrolovat v souboru).

Mimo alternativních funkcí existují tzv. remapované piny. Remapované piny jsou hardwarovou funkcionalitou některých STM32, registrů lze nastavit, který typicky ze dvou pinů je (např. `PA9`, nebo `PA11`) bude připojen na fyzický vývod. V definicích STM32duina jsou takto použité piny označeny `_R` , např. `PA_9_R`; balík při kompilaci by měl být schopen tyto piny správně nastavit.

build_opt.h

Ve Wiki STM32duina se často naráží na soubor `build_opt.h`. Tento soubor v předchozích verzích (v1) Arduina IDE nabízel přijatelnou možnost, jak upravovat vlastnosti knihoven STM32duina (pomocí maker) tato makra byla použita se stejnou syntaxí jako flagy kompilátoru (např. pro zapnutí UARTU `-DHAL_UART_MODULE_ENABLED`).

Problémem tohoto řešení je, že v aktuální verzi se cachují object fily a změny `build_opt.h` se projeví jen při kompilaci knihoven (nekontrolují se a nepropagují se změny). Soubor by bylo možné používat, ale jen při vymazání těchto object filů při každém buildu. Jako možná cesta k „čistému buildu“ je přenastavení optimalizace v nabídce STM32duina (to typicky donutí balík zkompilovat knihovny znovu). Druhou alternativou je vytvoření definice vlastní desky a přidat flagy tam (následující sekce).

Pro úplnost je vhodné zde zmínit, že pomocí maker jsou konfigurovatelné i samotné HAL drivery (`https://github.com/STM32duino/Arduino_Core_STM32/wiki/HAL-configuration`).

Definice vlastní desky

V případě, že je vhodné definovat vlastní desku, je dobré si přečíst odpovídající stránku na githubu (`https://github.com/STM32duino/Arduino_Core_STM32/wiki/Add-a-new-variant-%28board%29`). Důvodem pro vytvoření definice vlastní definice může být např. problematické nastavení knihoven pomocí maker (jako náhrada použití souboru `build_opt.h`). V tom případě (nastavení knihoven makry) je vhodné si přečíst tuto stránku (`https://github.com/STM32duino/Arduino_Core_STM32/wiki/Custom-definitions`).

Úprava sekcí paměti (linkerscript)

Pokud je třeba změnit rozvržení RAM, nebo Flash (např. požadujeme uživatelské sekce), správným krokem je upravit linkerscript. To může být výhodné, když uživatel chce rezervovat určité místo a nestojí o to, aby to zajistil kompilátor. Linkerscript je standardizovaný (dokumentace na webu `https://ftp.gnu.org/old-gnu/Manuals/ld-2.9.1/html_chapter/ld_3.html`). Při úpravě

uspořádání paměti, je vhodné do pole MEMORY definovat vlastní sekci (např. MY_RAM), té přiřadit výchozí adresu a délku. Zároveň je třeba náležitě upravit existující sekce. Výsledek může vypadat takto (posledních 16 bytů RAM si uživatel rezervuje):

```
/* Memories definition */
MEMORY
{
  RAM    (xrw)    : ORIGIN = 0x20000000,   LENGTH =
   LD_MAX_DATA_SIZE-16
  FLASH    (rx)    : ORIGIN = 0x8000000 + LD_FLASH_OFFSET, LENGTH =
    LD_MAX_SIZE - LD_FLASH_OFFSET
  MY_RAM    (xrw)    : ORIGIN = 0x20000000+LD_MAX_DATA_SIZE-16,
   LENGTH = 16
}
```

1.4.3 Programování s STM32duino s využitím HAL a CMSIS definic

V tuto chvíli dává smysl se podívat na komponenty, na kterých je STM32duino založeno, motivací může být použití těchto komponent v Arduino IDE k rozšíření možností. Konkrétně se jedná o knihovny HAL od ST a Arm GNU toolchain. S knihovnami HAL je možné vytvořit efektivnější program, který bude do velké míry reflektovat hardwarové uspořádání STM32. Součástí HAL jsou i CMSIS definice, pomocí nichž je možné pracovat přímo nad registry. Další možností je využít možností GCC a vkládat do programu napsaném v C sekce napsané v assembleru.

Použití funkcí STM32 HAL

ST, jak výrobce mikrokontroléru dodává knihovny HAL, které by měly pokrývat vše, co periferie STM32 nabízí. Jelikož je STM32duino na těchto knihovnách založeno, je možné jejich funkce volat. Knihovny HAL jsou dostupné pro každou řadu STM32 jako součást STM32Cube (např. STM32CubeG0) balíčků dostupných od výrobce (z st.com), součástí je nápověda ve formátu .chm. Je dobré zmínit, že se nejedná o podporovanou funkcionalitu, ale spíše o využití známých skutečností. Jako příklad je zde uvedeno blikání LED:

```
void setup() {
  __HAL_RCC_GPIOC_CLK_ENABLE();//zapnuti hodin GPIOA
  GPIO_InitTypeDef GPIO_InitStruct ={GPIO_PIN_6,GPIO_MODE_OUTPUT_PP
   ,
          GPIO_NOPULL,GPIO_SPEED_FREQ_VERY_HIGH,0};//nastaveni
   pinu
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);//samontna inicializace
}

void loop() {
  HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_6);//zmena logicke urovne PC6
  delay(100);
}
```

Znalost knihoven HAL může být velkou výhodou při přechodu na pokročilejší IDE, které knihovny HAL může využívat; jedná se např. o STM32CubeIDE (zdarma od výrobce ST), případně komerční IAR EWARM, nebo Keil µVision v kombinaci s STM32CubeMX.

Použití funkcí STM32 LL

LL knihovny jsou součástí HAL, jako méně abstraktní alternativa k hlavní části HAL knihoven. Použití v STM32duino je velmi omezené, tyto knihovny nejsou typicky zahrnuty při sestavování

programu; jednou z výjimek může být funkce `LL_GPIO_TogglePin()` kterou lze použít (přesto `HAL_GPIO_TogglePin()`, je také dostupná).

Možnost přístupu k registrům periferií mikrokontrolérů podle CMSIS:

CMSIS je standard definovaný firmou ARM (tvůrcem jader Cortex-M, které STM32 používá) obsahující knihovny. Jeho účelem bylo standardizovat knihovny pro mikrokontroléry s jádry ARM Cortex-M pro snadnější portování mezi mikrokontroléry různých výrobců. Jeho součásti jsou definice registrů, knihovny pro DSP, RTOS a další. Zmíněné definice registrů jsou použity i v STM32duinu a můžeme je využít pro nízkoúrovňové programování, příkladem je následující blikání LED:

```
1  void setup() {
2    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;//zapnuti hodin GPIOA
3    GPIOA->MODER &= ~GPIO_MODER_MODE5;
4    GPIOA->MODER |= GPIO_MODER_MODE5_0;//nastaveni pinu
5  }
6
7  void loop() {
8    GPIOA->ODR |= GPIO_ODR_OD5;  // zapis log 1
9    delay(100);
10   GPIOA->ODR &= ~GPIO_ODR_OD5; // zapis log 0
11   delay(100);
12 }
```

Pro úplnost a další pochopení je dobré si uvést, jak by vypadal identický program (ekvivalentní řádek po řádku) bez použití definic:

```
1  void setup() {
2    *(uint32_t*)(0X40021000+0x4C) |= (0b1<<0);//zapnuti hodin GPIOA
3    *(uint32_t*)(0x48000000+0x00) &= ~(0b11<<10);
4    *(uint32_t*)(0x48000000+0x00) |= (0b1<<10);//nastaveni pinu
5  }
6
7  void loop() {
8    *(uint32_t*)(0x48000000+0x14) |= (0b1<<5);  // zapis log 1
9    delay(100);
10   *(uint32_t*)(0x48000000+0x14) &= ~(0b1<<5); // zapis log 0
11   delay(100);
12 }
```

Možnost použití kódu napsaného v assembleru:

Vkládání kódu napsaného v assembleru je již krajní možností, v tu chvíli opravdu je program psán „instrukci po instrukci“. Díky tomu můžeme mít opravdu naprostou kontrolu nad jádrem (možná efektivita), však na úkor abstrakce. STM32duino používá ke kompilaci GCC, které nabízí možnost vkládání assembleru do kódu napsaného v C, jedná se o tzv. inline assembler, příkladem může být následující blikání LED:

```
1  //for G03 Nucleo LD3; tailorable to any other initialized GPIO PIN
2  void write_p(uint32_t state, uint32_t gpioAdd, uint32_t pin){
3   // when LSB 1 of "state" write gpio_pin HIGH, else LOW
4   // state output state
5   // gpioAdd as GPIOx base address
6   // pin number of pin e.g. 6 for PA6
```

```
7
8   __asm__ volatile (
9     "LDR R4, [R1,#0x14]\n"//load ODR to R4
10    "LDR R3, =1\n" //set R3 1
11    "LSL R3, R3, R2\n"//1<<pin; R2 may be overwritten
12    "BIC R4, R4, R3\n" // clear ODx
13
14    "LDR R2, =1\n"//set R2 1
15    "TST R0, R2\n"//compare 1 with state value; R0 may be
      overwritten
16    "BEQ WRITE\n"//if state 1 skip setting of OD6
17    "ORR R4, R4, R3\n"//set ODx 1
18
19    "WRITE: STR R4,[R1,#0x14]\n"
20  );
21 }
22
23 void setup() {
24  pinMode(PC6, OUTPUT);
25 }
26 uint8_t a=0;
27 void loop() {
28  write_p(++a, 0x50000800, 6);//used GPIOC as 0x50000800
29  a &=1;
30  delay(200);
31 }
```

Předchozí příklad je napsaný pro NUCLEO-G031K8 – jeho problém je přenositelnost, konkrétně adresa brány GPIO není na všech mikrokontrolérech STM32 stejná. Pro dosažení přenositelnosti dává smysl při kombinaci C a assembleru použít CMSIS drivery a předávat adresu struktury, která odpovídá základní adrese periferie (následující příklad).

```
1  //for G03 Nucleo LD3; tailorable to any other initialized GPIO PIN
2  void write_p(uint32_t state, GPIO_TypeDef* gpio, uint32_t pin){
3   // when LSB 1 of "state" write gpio_pin HIGH, else LOW
4   // state output state
5   // gpio type GPIO_TypeDef* as GPIOx (to COMPLY CMSIS def)
6   // pin number of pin e.g. 6 for PA6
7
8   __asm__ volatile (
9     "LDR R4, [R1,#0x14]\n"//load ODR to R4
10    "LDR R3, =1\n" //set R3 1
11    "LSL R3, R3, R2\n"//1<<pin; R2 may be overwritten
12    "BIC R4, R4, R3\n" // clear ODx
13
14    "LDR R2, =1\n"//set R2 1
15    "TST R0, R2\n"//compare 1 with state value; R0 may be
      overwritten
16    "BEQ WRITE\n"//if state 1 skip setting of OD6
17    "ORR R4, R4, R3\n"//set ODx 1
18
19    "WRITE: STR R4,[R1,#0x14]\n"
20  );
```

```
}

void setup() {
 pinMode(PC6, OUTPUT);
}
uint8_t a=0;
void loop() {
 write_p(++a, GPIOC, 6);//used GPIOC as x50000800
 a &=1;
 delay(200);
}
```

Dokumentaci k inline assembly v GCC lze nalézt na stránce `https://GCC.gnu.org/onlinedocs/GCC/extensions-to-the-c-language-family/how-to-use-inline-assembly-language-in-c-code.html`, případně v knize[1] a programming manuálech k používanému jádru, např. PM0223. Předávání parametrů pro jádra Cortex-M je definováno v dokumentu AAPCS Procedure Call Standard for the ARM Architecture, dostupné online `https://github.com/ARM-software/abi-aa`.

Následující příklad ukazuje, jak lze použít inline assembly pro práci s polem a jak předat návratovou hodnotu zpět do C. Zároveň je uvedena intristic funkce NOP (může být vhodné k ní přiřadit break point pro demonstraci).

```
uint32_t sumArray(uint32_t * array, uint32_t len){
// array uint32_t array to be summed
// len size of array
// return sum of array content

 __asm__ volatile (
   "MOVS R2,#0\n"
   "START: LDR R3, [R0]\n"//load ODR to R4
   "ADD R0,#4\n"//increment pointer
   "ADD R2,R2,R3\n"
   "SUB R1,R1,#1 \n"
   "BNE START\n"//if state 1 skip setting of OD6
   "MOV R0,R2\n"//use R2 as return value
 );
 return (uint32_t) array;//return first argument (R0) that is
   actually "array" pointer
}

void setup() {
uint32_t pole[3]={1,2,3};
  pole[0] = sumArray(pole,sizeof(pole)/sizeof(*pole));
  __NOP();
}

void loop() {
}
```

Následující příklad ilustruje rychlé načítání hodnot registru (adresy) do pole:

[1] J. Yiu, The Definitive guide to ARM Cortex-M0 and Cortex-M0+ Processors. Oxford, Uk; Singapore: Elsevier, 2015.

```
1  void sampleArray(uint32_t * array, uint32_t len, uint32_t* address)
       {
2  // array uint32_t array to be filled
3  // len size of array (elements, not bytes)
4  // address to be sampled to array
5   __asm__ volatile (
6     "START: LDR R3, [R2]\n"
7     "STMIA R0!, {R3}\n"
8     "SUB R1,R1,#1 \n"
9     "BNE START\n"
10  );
11 }
12
13 void setup() {
14 uint32_t pole[30]={1,2,3};
15   sampleArray(pole,sizeof(pole)/sizeof(*pole),(uint32_t*)&(GPIOA->
       IDR));
16   __NOP();
17 }
18
19 void loop() {}
```

Při debugování programů napsaných v inline assembleru se celá inline sekce jeví jako jeden řádek; pro krokování výhodné je výhodné použít příkaz `ni` GDB (viz sekce „Možnosti GDB pomocí okna **Debug Console**“).

1.5 Specifické vlastnosti balíku, pokyny pro kurzy (LPE, KPE)

V této kapitole jsou popsány některé specifické záležitosti související nasazením STM32duina ve výuce, které však přímo nesouvisejí s balíkem jako takovým (sekce Jak číst datasheet, Boot a Option byty). Dále jsou zde uvedené konkrétní příklady pro předměty a kurzy kde již byl balík STM32duino využit (Předmět LPE, Kurz KPE, Kurz ETC)

1.5.1 Jak číst datasheet

Datasheet (korektně česky katalogový list) je dokument obsahující klíčové parametry součástky (v tomto případě mikrokontroléru STM32). Mimo elektrických parametrů, lze zde nalézt popisy funkcí jednotlivých pinů mikrokontroléru. Znalost těchto funkcí je klíčová pro psaní téměř jakéhokoli programů pokud např. na pinu není funkce kanálu ADC, není možné tam měřit napětí. Jednotlivé funkce daných pinů jsou uvedeny v tabulce, typicky v sekci nazvané **Pinouts, pin description and alternate functions**. Na obrázku 15 je vidět část takovéto tabulky; funkce pinu jsou uvedeny ve sloupci **Alternate functions** a **Additional Functions**. Rozdíl mezi **Alternate** a **Additional** funkcemi je, že je může být zvolena vždy jedna **Alternate** funkce (, nebo GPIO) a k tomu povoleno libovolné množství **Additional** funkcí. V STM32duinu nastává jedna nekompatibilita – jazyk Wiring neřeší případ kdy na stejném pinu je možné použít stejnou funkcionalitu různými prostředky (např. pokud je třeba generovat PWM a na pinu jsou dva kanály čítače). Tato skutečnost je vyřešena definováním pinu vícekrát (např. PA_0 a PA_0_ALT1), viz kapitola 1.4.2. Zároveň může nastat opačný problém; funkcionalita je dosažitelná více piny (např. stejný kanál čítače na dvou různých pinech), přesto mikrokontrolér toto nepodporuje – je proto třeba ověřovat v datasheetu a souboru **PeripheralPins.c**, které piny jsou kdy použity k dosažení dané funkcionality.

Pin						Pin name (function upon reset)	Pin type	I/O structure	Note	Alternate functions	Additional functions
SO8N	WLCSP18	TSSOP20	UFQFPN28	LQFP32 / UFQFPN32	LQFP48 / UFQFPN48						
-	D4	11	10	11	-	PA4	I/O	FT_a	-	SPI1_NSS/I2S1_WS, SPI2_MOSI, TIM14_CH1, LPTIM2_OUT, EVENTOUT	ADC_IN4, TAMP_IN1, RTC_TS, RTC_OUT1, WKUP2
-	C3	12	11	12	16	PA5	I/O	FT_ea	-	SPI1_SCK/I2S1_CK, TIM2_CH1_ETR, LPTIM2_ETR, EVENTOUT	ADC_IN5
-	E3	13	12	13	17	PA6	I/O	FT_ea	-	SPI1_MISO/I2S1_MCK, TIM3_CH1, TIM1_BK, TIM16_CH1, LPUART1_CTS	ADC_IN6

Obrázek 15: Tabulka s popisy pinu

1.5.2 LPE

MMC5883

V předmětu LPE katedry měření FEL ČVUT je využíván tříosý magnetometr MMC5883. V současnosti existuje knihovna od firmy Adafruit dostupná jako .zip archiv na githubu. Knihovnu je možné přidat pomocí "Sketch"→"Include Library"→"Add .ZIP Library..."(ke knihovně je nutné nainstalovat závislosti (další knihovny); ty jsou již dostupné z Arduino IDE). 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 uvedeno následující příklad (modifikovaný příklad "magsensor") (výstup tří os je možné zobrazit pomocí "Tools"→Serial Plotter").[2]

```
#define INTERNAL_CONTROL_0    0x08
#define FLIP_SET 0x08
#define FLIP_RESET 0x10

#include <Adafruit_MMC5883.h>

Adafruit_MMC5883 mag = Adafruit_MMC5883(12345);

void setup(void)
{
  Serial.begin(115200);
  Wire.setSDA(PB_9);
  Wire.setSCL(PB_8);//Warning: PB8 may be also BOOT0 pin on G4
  if(!mag.begin())
  {
    Serial.println("ERROR: magnetometer not found");
    while(1);//break
  }
  mag.setContinuousFreq(MMC5883_CMFREQ_5HZ);//set frequency
}
```

[2]Před přepsáním upraveného příkladu **upozornění:** PB8 může být na STM32G431 použit jako **BOOT0 pin**.

```
21 void loop(void)
22 {
23   /* Get a new sensor event */
24   sensors_event_t event1,event2;
25   mag.getEvent(&event1);
26
27   Wire.beginTransmission(0x30);//set flipping pulse
28   uint8_t cmd[2] = {INTERNAL_CONTROL_0,FLIP_SET};
29   Wire.write(cmd,2);
30   Wire.endTransmission();
31   delay(400);
32
33   mag.getEvent(&event2);
34   Wire.beginTransmission(0x30);//reset flipping pulse
35   cmd[1]=FLIP_RESET;
36   Wire.write(cmd,2);
37   Wire.endTransmission();
38
39   /* Display the results can be used with arduino Serial plotter*/
40   Serial.print((event1.magnetic.x-event2.magnetic.x)/2);
41   Serial.print(" ");
42   Serial.print((event1.magnetic.y-event2.magnetic.y)/2);
43   Serial.print(" ");
44   Serial.println((event1.magnetic.z-event2.magnetic.z)/2);
45   /* "Compass" by atan2 function in degrees for xy
46   Serial.println(180*atan2(event1.magnetic.y-event2.magnetic.y,
      event1.magnetic.x-event2.magnetic.x)/PI);*/
47   delay(400);
48 }
```

Příklad s PWM

V předmětu PWM byl dále prezentován vzorový příklad s PWM napsaný po mbedem. Jeho port pro STM32duino vypadá následovně:

```
1  #define SINE_STEPS 8        // Number of dutycycle steps for output
       wave
2  #define SINE_OUT_FREQ 2000  // Frequency of output sine in Hz
3  //#define PI 3.141592f        // Constants to compute the sine
       waveform
4  #define SINE_STEPS_RAD (2.0f * PI / (float)SINE_STEPS)
5  float sine_duty[SINE_STEPS];                    // Table to
       generate the sine waveform using dutycycles
6  #define PWM_FREQ 16000                          // Frequency of
       Pulse Width Modulated signal in Hz
7  HardwareSerial Serial2(PA3_ALT1, PA2_ALT1);                //
       UART2 TX RX
8  HardwareTimer pwmGen = HardwareTimer(TIM1);  //PA9
9  HardwareTimer pwm_ticker = HardwareTimer(TIM3);
10
11
12 // Ticker calls this fucntion to update the PWM dutycycle
13 callback_function_t pwm_duty_updater() {
```

```
14    static int idx = 0;
15    pwmGen.pause();
16    pwmGen.setPWM(2, PA9, PWM_FREQ, sine_duty[idx] * 100);  //channel
       ,pin Set the dutycycle % to next value in array
17    idx++;
                // Increment the idx
18    if (idx == SINE_STEPS)
19      idx = 0;  // Reset the idx when teh end has been reached
20    return NULL;
21  }
22  unsigned int aread0;
23  unsigned int aread1;
24  char buffer[20];
25  char comIN;
26  unsigned char status = 0;
27  float v1;
28  float v2;
29  float tim = 0;
30  unsigned char i;
31  void setup() {
32    pinMode(PA8, OUTPUT);
33    pinMode(PA0, INPUT_ANALOG);
34    pinMode(PA1, INPUT_ANALOG);
35    analogReadResolution(16);
36
37    //pwm_ticker.setMode(1, TIMER_OUTPUT_COMPARE, NC);//by now not
       use
38    pwmGen.setMode(2, TIMER_OUTPUT_COMPARE_PWM1, PA9);
39    Serial2.begin(9600);  //9600 baud rate
40    for (i = 0; i < SINE_STEPS; i++) {
41      sine_duty[i] =
42        ((sin(i * SINE_STEPS_RAD)) * 0.8f + 1.0f) / 2.0f;  // convert
       sine (-1.0 .. +1.0) into dutycycle (0.0 .. 1.0)
43    }
44    Serial2.printf("PWM test & measure!\r\n");
45  }
46
47  void loop() {
48    if (Serial2.available()) {  // je k dispozici novy prikaz?
49      comIN = Serial2.read();
50      switch (comIN) {  // a podle toho co prislo, proved danou akci
51        case '0':       // not sending data
52          status = 0;
53          Serial2.printf("Off\r\n");
54          break;
55        case '1':  // sending data
56          status = 1;
57          Serial2.printf("On\r\n");
58          break;
59        case 'a':  // LED off
60          digitalWrite(PA8, LOW);
61          Serial2.printf("LED Off\r\n");
```

```
62          break;
63        case 's':  // LED on
64          digitalWrite(PA8, HIGH);
65          Serial2.printf("LED On\r\n");
66          break;
67        case 'q':  // generate PWM 2kHz
68          Serial2.printf("2 kHz PWM On\r\n");
69          pwm_ticker.pause();
70          pwmGen.pause();
71          pwmGen.setPWM(2, PA9, 200, 50);
72          break;
73        case 'r':  // generate PWM 2kHz
74          Serial2.printf("16 kHz PWM On\r\n");
75          pwm_ticker.pause();
76          pwmGen.pause();
77          pwmGen.setPWM(2, PA9, 16000, 50);
78          break;
79        case 'w':  // stop generating PWM
80          pwm_ticker.pause();
81          pwmGen.pause();
82          pwmGen.setPWM(2, PA9, 16000, 0);
83          Serial2.printf("Off\r\n");
84          break;
85        case 'e':
86          pwm_ticker.pause();
87          pwmGen.pause();
88          pwmGen.setPWM(2, PA9, 12500, 50);
89          delay(1);
90          pwmGen.setPWM(2, PA9, 12500, 0);
91          Serial2.printf("Burst\r\n");
92          break;
93        case 'n':
94          pwmGen.pause();
95          pwmGen.setPWM(2, PA9, (float)PWM_FREQ, 50);
96          pwm_ticker.pause();
97          pwm_ticker.attachInterrupt(pwm_duty_updater);
98          pwm_ticker.setPWM(1, NC, (SINE_STEPS * SINE_OUT_FREQ)/2000,
      50);//freq divided by 2000 for real-time oscillosope check
99          Serial2.printf("2kHz PWM sine On\r\n");
100         break;
101       case 'm':
102         pwm_ticker.pause();
103         pwmGen.setPWM(2, PA9, 12500, 0);
104         Serial2.printf("Off\r\n");
105         break;
106       default:  // kdyz prijde neznamy znak, provede se "default"
     cast kodu
107         break;
108     };  // switch
109   };    // if
110   aread0 = analogRead(PA_0);
111   aread1 = analogRead(PA_1);
```

```
112   ;
113   aread0 /= 20;  // hruby prevod na mV...(65535/3300)
114   aread1 /= 20;
115   v1 = aread0 / 1000.0f;
116   v2 = aread1 / 1000.0f;
117   if (status) {
118     tim += 0.05;
119     Serial2.printf("%04u,%04u\r\n",aread0,aread1); //SerialChart
120     //Serial2.printf("$$P%.2f,%.3f,%.3f;", tim, v1, v2);  // Data
       Plotter by Jiri
121     // Maier
122   }
123   delay(50);
124 }
```

1.5.3 Příklady s LCD

SSD1306 po SPI

```
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);
4
5  #define RST PC7
6  #define CS PA9
7  #define DC PB6
8  #define DIN PA7
9  #define CLK PA5
10
11 void command(uint8_t c) {
12   digitalWrite(CS, HIGH);
13   digitalWrite(DC, LOW);
14   digitalWrite(CS, LOW);
15   SPI.transfer(c);
16   digitalWrite(CS, HIGH);
17 };
18
19 void data(uint8_t c) {
20   digitalWrite(CS, HIGH);
21   digitalWrite(DC, HIGH);
22   digitalWrite(CS, LOW);
23   SPI.transfer(c);
24   digitalWrite(CS, HIGH);
25 };
26
27 void setup() {
28   Serial.begin(9600);
29   pinMode(RST, OUTPUT);
30   pinMode(CS, OUTPUT);
31   pinMode(DC, OUTPUT);
```

```
32
33    digitalWrite(RST, LOW);
34    delay(1);  //ms delay for being sure
35    digitalWrite(RST, HIGH);
36
37    SPI.setMOSI(DIN);
38    SPI.setSCLK(CLK);
39    SPI.begin();
40
41   //initialization commands, same sequence as by i2c
42   for(int i=0;i<sizeof(initSTR)/sizeof(*initSTR);++i) command(
       initSTR[i]);
43  }
44
45  void loop() {
46    command(0x0 | 0x0);  // low col = 0
47    command(0x10 | 0x0);  // hi col = 0
48    command(0x40 | 0x0); // line #0
49    digitalWrite(CS, HIGH);
50    digitalWrite(DC, HIGH);
51    digitalWrite(CS, LOW);
52    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  }
```

PCD8544 po SPI

Další příklad používá vykresluje dva vzory na LCD pcd8544.

```
1  /* zdroj (minimal pcd8544), upraveno dle hardware:
2   * https://www.youtube.com/watch?v=RAlZ1DHw03g, Julian Ilett*/
3  #include <SPI.h>
4
5  #define RST PC7
6  #define CE  PA9
7  #define DC  PB6
8  #define DIN  PA7
9  #define CLK  PA5
```

```
10
11 void LcdWriteData(uint8_t dat)
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 }
```

SSD1306 po I2C

Následující příklad ukazuje generování pruhů na I2C variantě modulu s displejem SSD1306.

```
1 #include <Wire.h>
```

```
2  uint8_t initSTR[] = {//string for initialization of lcd
3   0xAE,0x20,0x00,0xB0,0xC8,0x00,0x10,0x40,0x81,0xFF,0xA1,0xA6,0xA8,0
      x3F,
4   0xA4,0xD3,0x00,0xD5,0xF0,0xD9,0x22,0xDA,0x12,0xDB,0x20,0x8D,0x14,0
      xAF
5  };
6
7  void setup() {
8    Wire.begin();
9    Wire.beginTransmission(0x3C);//zacatek prenosu
10   Wire.write(initSTR, sizeof(initSTR));//"bufferovani" zpravy
11   Wire.endTransmission();//konec prenosu, nasleduje odeslani
12 }
13
14 void loop() {
15   delay(1000);
16   for (uint8_t i = 0; i < 64 / 8; i++) {
17     Wire.beginTransmission(0x3C);
18     Wire.write(0x40);//write data
19     for (uint8_t j = 0; j < 128; j++) Wire.write(j & 4 ? 0x0 : 0xff
       );
20     Wire.endTransmission();
21   }
22
23   delay(1000);
24   for (uint8_t i = 0; i < 64 / 8; i++) {
25     Wire.beginTransmission(0x3C);
26     Wire.write(0x40);//write data
27     for (uint8_t j = 0; j < 128; j++) Wire.write(j & 8 ? 0x0 : 0xff
       );
28     Wire.endTransmission();
29   }
30 }
```

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 displeje). Tato konkrétní knihovna podporuje jak I2C, tak SPI. Arduinovské knihovny pro zařízení připojená po I2C nebo SPI obvykle bez problémů na STM32duinu (používají se typicky knihovní funkce).

1.5.4 KPE

V kurzu praktické elektroniky (pro studenty nastupující do prvního ročníku v běhu 2024 bylo používáno STM32duino. Příklady kopírovaly příklady z tohoto dokumentu (LED, tlačítko, PWM, sinus pomocí PWM).