Bakalářská práce

Rychlá akvizice signálu na platformě STM32

Ondřej Šmíd

Vedoucí: Ing. Stanislav Vítek, Ph.D.

Fakulta Elektrotechnická

Katedra radioelektroniky

23. května 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Šmíd** Jméno: **Ondřej** Osobní číslo: **469848**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra radioelektroniky**

Studijní program: **Elektronika a komunikace**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Rychlá akvizice signálu na platformě STM32

Název bakalářské práce anglicky:

Fast Signal Acquisition on the STM32 Platform

Pokyny pro vypracování:

1) Prostudujte možnosti rychlé akvizice signálu na platformě STM32. Vezměte v úvahu i připojení externí periferie.
2) Navrhněte strukturu akvizičního systému.
3) Implementujte obslužný program v prostředi FreeRTOS.
4) Na základě předchozích bodů připravte podklady pro demonstrační úlohu v předmětu zaměřeném na výuku mikroprocesorové techniky.

Seznam doporučené literatury:

[1] Developing Applications on STM32Cube with RTOS, ST Microelectronic User Manual UM1722, 2014
[2] Warren Gay, Beginning STM32. Developing with FreeRTOS, libopencm3 and GCC. Apress, 2018
[3] STM32s ADC modes and their applications, ST Microelectronic Application Note AN3116, 2010

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Stanislav Vítek, Ph.D., katedra radioelektroniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **12.02.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Stanislav Vítek, Ph.D.	prof. Mgr. Petr Páta, Ph.D.	prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, ...

Poděkování

Velmi rád bych chtěl poděkovat vedoucímu práce. Jeho trpělivost a cenné rady byly velkou oporou během přípravy programu a psaní práce. Dále bych rád poděkoval Ondřeji Šeredovi za pomoc s grafickými úpravami, své matce za pomoc s jazykovou úpravou a oběma rodičům za podporu během studia.

Anotace

Práce se zaměřuje na návrh akvizičního systému na platformě STM32, uvažuje externí analogově-digitální převodník a zpracování signálu na externím zařízení. Signál je v systému uchován pomocí cache paměti. Systém je testován, zda pracuje správně. Práce popisuje systém s maximální vzorkovací frekvencí $f_s = 1$ MHz. Součástí práce je i úloha přednášející problematiku pro předmět zaměřený na výuku mikroprocesorové techniky.

Klíčová slova: RTOS, STM32, Akvizice signálu

Annotation

The thesis focuses on the design of the acquisition system on the STM32 platform, considering an external analog-to-digital converter and signal processing on an external device. The signal is stored in the system by cache memory. The system is tested for proper operation. The thesis describes the system with the maximum sampling frequency of $f_s = 1$ MHz. Part of the thesis is also the role of lecturer for the subject focused on microprocessor teaching.

Keywords: RTOS, STM32, Signal acquisition

Obsah

I Seznam obrázků **II**

II Seznam kódů **III**

III Seznam zkratek **IV**

1 Úvod **1**
1.1 Motivace . . . 1
1.2 Cíl práce . . . 1
1.3 Rozbor jednotlivých částí práce . . . 1

2 Teoretická část **3**
2.1 Platforma STM32 . . . 3
2.1.1 Série STM32F2 . . . 3
2.1.2 Série STM32F4 . . . 4
2.1.3 Série STM32F7 . . . 4
2.1.4 Série STM32H7 . . . 5
2.1.5 Vývojová deska STM32F446RE Nucleo-64 . . . 6
2.2 Analogově-digitální převodníky . . . 6
2.2.1 Převodník s postupnou aproximací . . . 7
2.2.2 Komparační převodník . . . 8
2.3 Digitálně-analogový převodník . . . 8
2.4 GPIO . . . 9
2.5 Časovače a zdroje hodinového signálu . . . 10
2.6 Přerušení . . . 11
2.7 RTOS . . . 11
2.7.1 FreeRTOS . . . 13

3 Praktická část **14**
3.1 Výběr prostředí pro implementaci . . . 15
3.2 Analýza systému . . . 16
3.3 Testování akvizičního systému . . . 17
3.3.1 Rychlost systému . . . 18

4 Demonstrační úloha **21**

5 Závěr **22**

A Příloha 1 **26**
A.1 Schéma pro nastavení systémových hodin . . . 26

B Příloha 2 **27**
B.1 Demonstrační úloha . . . 27
B.1.1 Úvod . . . 27
B.1.2 Úkol . . . 27
B.1.3 Teoretický úvod . . . 27
B.1.4 Demonstrace . . . 28

Seznam obrázků

1 Periferie některých řad rodiny STM32 [1] . . . 3
2 Rozdělení série STM32F2 [2] . . . 3
3 Rozdělení série STM32F4 [3] . . . 4
4 Rozdělení série STM32F7 [4] . . . 5
5 Rozdělení série STM32H7 [5] . . . 5
6 Připojení periferií na piny levé části desky [6] . . . 6
7 Připojení periferií na piny pravé části desky [6] . . . 7
8 Princip aproximace v jednotlivých cyklech aproximačního ADC [7] . . . 7
9 Blokové schéma komparačního ADC [7] . . . 8
10 Schéma zapojení DAC [8] . . . 9
11 Stavový diagram tasku [9] . . . 12
12 Snímek zapojení aparatury pro měření rychlosti akvizičního systému, zleva: logický analyzátor, nucleo použité pro akviziční sytém a generátor signálu vytvořený z nuclea . . . 15
13 Schéma akvizičního systému – sdílení paměti částmi kódu . . . 16
14 Schéma soustavy pro testovaní komunikace . . . 17
15 Schéma funkce generátoru signálu . . . 18
16 Časový průběh logických hodnot jednotlivých částí sběrnice s vyznačeným časovým odstupem změny hodnoty hodinového signálu a datové části sběrnice 19
17 Snímek průběhu signálu výstupu akvizičního systému na osciloskopu . . . 20
18 Schéma zapojení zdrojů hodinového signálu zařízení série STM32F4 [10] . . . 26

Listings

1 Časově podmíněný blok pro spuštění změny hodnoty hodin sběrnice a vyčtení hodnoty sběrnice 17
2 Kód časově podmíněného bloku pro vyčtení hodnoty 19

Seznam zkratek

ADC Analog-digital converter - analogově-digitální převodník.

CPU Control processing unit - procesor.

DAC Digital-analog converter - digitálně-analogový převodník.

FIFO First in first out - typ zásobníku (fronty), ze kterého výčet prvků je v pořadí v jakém byly zapsány.

FPU Floating-point unit - matematický koprocesor.

GUI Graphical user interface - grafické rozhraní pro uživatele.

HAL Hardware abstraction layer - abstraktní vrstva hardwareu.

IDE Integrated development enviroment - vývojové prostředí.

LSB Least significant bit - nejméně významný bit.

MAC Media control access address - fyzická adresa.

RAM Random access memory - paměť s náhodným přístupem.

RC Resistor-capacitor oscillator - oscilátor složený z rezistoru a kapacitoru.

SRAM Static random access memory - statická paměť s náhodným přístupem.

USART Universal synchronous/asynchronous receiver and transmitter - synchronní/asynchronní sériové rozhraní.

1 Úvod

1.1 Motivace

Motivací pro vytvoření této práce byla úvaha, jak přijmout informace z externí periferie. Příkladem takové periferie je kamera JAI CM-040GE s rozhraním GigE Vision a rychlostí 61,15 fps a maximální velikostí snímku 4040 B [11] (přenosová rychlost činí přibližně 2 Gbit/s). Pro tuto přenosovou rychlost není na trhu levné řešení, jak z periferie zachytávat data. Cílem práce mělo původně být řešení tohoto problému.

Návrhem řešení tedy bylo prozkoumat možné použití mikrokontroleru pro komunikaci s periferií. Z množství mikrokontrolerů, které jsou dnes na trhu, byla vybrána platforma STM32, díky předešlé zkušenosti s programováním na této platformě a také díky širokému výběru desek a podpoře výrobce. Výrobce také poskytuje vývojové nástroje jako jsou IDE TrueSTUDIO nebo CubeMX – GUI, kterým je možné předkonfigurovat nastavení jednotlivých periferií – jejich registrů.

Díky složitosti problému a maximálních taktů mikroprocesorů této platformy bylo od návrhu funkčního řešení upuštěno. Úvaha se stala hypotetickou. Maximální takt mikrokontrolerů platformy STM32 je 480 MHz (viz podkapitola 2.1.4).

1.2 Cíl práce

Cílem práce je tedy prozkoumat možnosti vytvoření komunikačního prvku s periferií za použití levného řešení a platformy STM32 a s nejvyšší možnou přenosovou rychlostí komunikace. Jinými slovy jaká je náročnost režie, která je nutná pro akvizici signálu na platformě a o jakou míru klesne frekvence, kterou lze komunikovat s periferií, oproti maximálnímu systémovému taktu.

Dalším cílem je z načerpaných poznatků vytvořit úlohu do výuky mikroprocesorové techniky, která bude demonstrovat použití některých integrovaných periferií mikrokontrolerů platformy k získání představy, jak dané periferie pracují a jaké jsou možnosti využití mikrokontroleru platformy STM32 v praktickém úkolu.

1.3 Rozbor jednotlivých částí práce

Práce stručně popíše vlastností platformy STM32, zaměří se na rodinu 32bitových mikrokontrolerů. Na jednom ze zástupců této rodiny bude navrhnut obslužný program pro akvizici signálu.

Mikrokontrolery mají velké množství rozdílných komunikačních rozhraní. Jedním z nich je ADC převodník. Platforma používá ADC s postupnou aproximací. Rychlost akvizice signálu je ovlivněná touto součástkou, která díky principu provedení není nejlepší možná. Proto bude v textu navrhnuto řešení bez využití vestavěného převodníku. Místo něho bude uvažováno předřazení rychlejšího převodníku, který mikrokontroleru poskytuje vzorky signálu pro zpracování (viz podkapitola 2.2.2).

Mikrokontroler je možné používat k výpočtům a zpracování signálu, není však vhodné přijatý signál zpracovávat přímo mikrokontrolerem, pokud je dat velké množství nebo je zpracování složité, či samotné výpočty by byly pro vyhodnocení signálu moc náročné. Práce se proto nebude ve svém návrhu řešení věnovat zpracování signálu.

Bude navrženo řešení, které uvažuje mikrokontroler jako součástku, jež propojí počítač, či jiné zařízení pro vyhodnocení signálu, s rychlým převodníkem. Dále budou uvažovány, rozdílné přenosové rychlosti, se kterými musí mikrokontroler v této soustavě pracovat. Pokud přenosová rychlost převodníku bude vyšší než rychlost přijímání vyhodnocovacího prvku, bude pracovat jako cache pamět.

Řešení bude podrobeno testování pro zjištění rychlosti a správnosti funkce. Výsledky rychlosti zachycení budou zhodnoceny v závěru (viz kapitola 5).

Budou popsány některé principy ADC, zběžně nastíněná platforma STM32 a zástupce, u něhož bude navržen akviziční program. Práce objasní, co je to RTOS, jeho použití a verzi FreeRTOS. Vysvětlí, proč RTOS v práci není použit. Na závěr bude předložena demonstrační úloha, představující některé periferie a principy embedded programování, do předmětu pro výuku mikroprocesorové techniky.

Demonstrační úloha se zaměří na představení rozdílů implementací zadaného úkolu v různých programovacích jazycích s použitím middlewarů. Jednotlivé implementace budou obdobou stejného programu pro vytvoření jednokanálového osciloskopu.

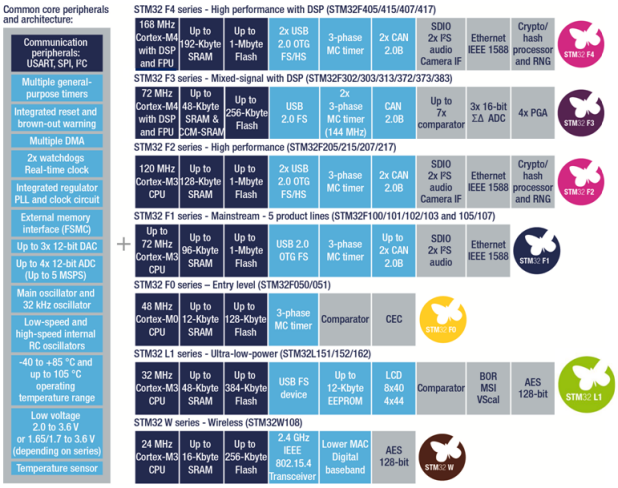
2 Teoretická část

2.1 Platforma STM32

STMicroelectronics je firma zabývající se výrobou polovodičových součástek [12]. Vyvíjí a vyrábí mimo jiné i mikrokontrolery. Platforma má několik verzí: 8bitovou a 32bitovou, u které existuje ještě varianta s duálním jádrem. Pro účely práce byla vybrána pouze 32bitová verze.

Rodina mikrokontrolerů STM32 32-bit využívá mikroprocesory Arm Cortex-M. Desky jsou dále děleny podle jádra procesoru, výkonnostních parametrů a podle počtu a variací různých periferií (viz obrázek 1). Dělení podle výkonu rozlišuje série F2, F4, F7 a H7 společně označované jako „high-performance“ [13].

Níže jsou zběžně představeny jednotlivé řady platformy.



1: Periferie některých řad rodiny STM32 [1]

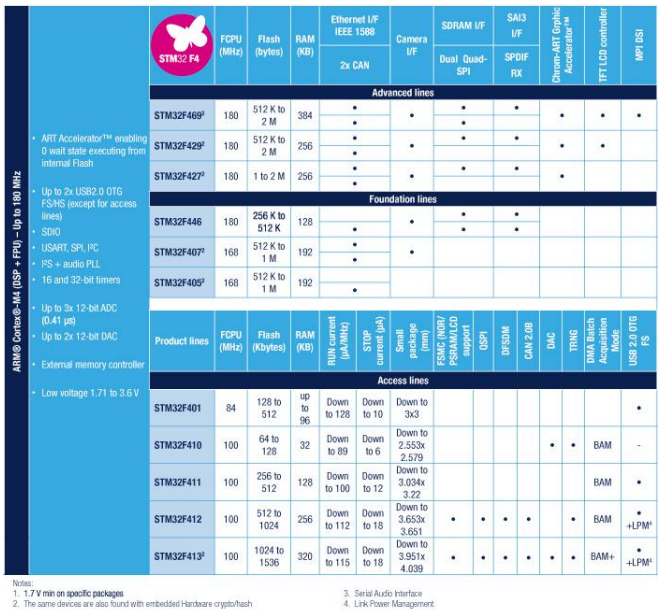
2.1.1 Série STM32F2

Série STM32F2 používá jádro procesoru ARM Cortex-M3 s maximálním taktem 120 MHz při energetické náročnosti 175 μA/MHz. Série má pouze dva zástupce (viz obrázek 2). Velikosti pamětí jsou 1 MB flash a 128 kB SRAM. Mikrokontrolerům je přidělená i MAC adresa.

2: Rozdělení série STM32F2 [2]

2.1.2 Série STM32F4

Tato série využívá procesor ARM Cortex-M4. Maximální takt je 180 MHz s energetickými nároky 89 - 260 μA/MHz při standardních podmínkách[1]. Je rozdělená do tří řad podle výkonu (viz obrázek 3) [3]. Obrázek ukazuje rozdělení jednotlivých řad a produktů s přidělenými periferiemi, ale i velikosti taktů, kde se hlavně liší základní řada (první mikrokontrolery řady), a velikosti pamětí.



ARM® Cortex®-M4 (DSP + FPU) – Up to 180 MHz

- ART Accelerator™ enabling 0 wait state executing from internal Flash
- Up to 2x USB2.0 OTG FS/HS (except for access lines)
- SDIO
- USART, SPI, I²C
- I²S + audio PLL
- 16 and 32-bit timers
- Up to 3x 12-bit ADC (0.41 µs)
- Up to 2x 12-bit DAC
- External memory controller
- Low voltage 1.71 to 3.6 V

STM32 F4	FCPU (MHz)	Flash (bytes)	RAM (KB)	Ethernet I/F IEEE 1588	2x CAN	Camera I/F	SDRAM I/F Dual Quad-SPI	SAI3 I/F SPDIF RX	Chrom-ART Graphic Accelerator™	TFT LCD controller	MPI DSI
Advanced lines											
STM32F469[2]	180	512 K to 2 M	384	•	•	•	•	•	•	•	•
STM32F429[2]	180	512 K to 2 M	256	•	•	•	•	•	•	•	
STM32F427[2]	180	1 to 2 M	256	•	•	•	•	•	•		
Foundation lines											
STM32F446	180	256 K to 512 K	128	•	•	•	•	•			
STM32F407[2]	168	512 K to 1 M	192	•	•	•					
STM32F405[2]	168	512 K to 1 M	192	•							

Product lines	FCPU (MHz)	Flash (Kbytes)	RAM (KB)	RUN current (µA/MHz)	STOP current (µA)	Small package (mm)	FSMC (NOR/ PSRAM/LCD support	QSPI	DFSDM	CAN 2.0B	DAC	TRNG	DMA Batch Acquisition Mode	USB 2.0 OTG FS
Access lines														
STM32F401	84	128 to 512	up to 96	Down to 128	Down to 10	Down to 3x3								•
STM32F410	100	64 to 128	32	Down to 89	Down to 6	Down to 2.553x 2.579					•	•	BAM	-
STM32F411	100	256 to 512	128	Down to 100	Down to 12	Down to 3.034x 3.22							BAM	•
STM32F412	100	512 to 1024	256	Down to 112	Down to 18	Down to 3.653x 3.651	•	•	•	•		•	BAM	• +LPM[4]
STM32F413[2]	100	1024 to 1536	320	Down to 115	Down to 18	Down to 3.951x 4.039	•	•	•	•	•	•	BAM+	• +LPM[4]

Notes:
1. 1.7 V min on specific packages
2. The same devices are also found with embedded Hardware crypto/hash
3. Serial Audio Interface
4. Link Power Management

3: Rozdělení série STM32F4 [3]

2.1.3 Série STM32F7

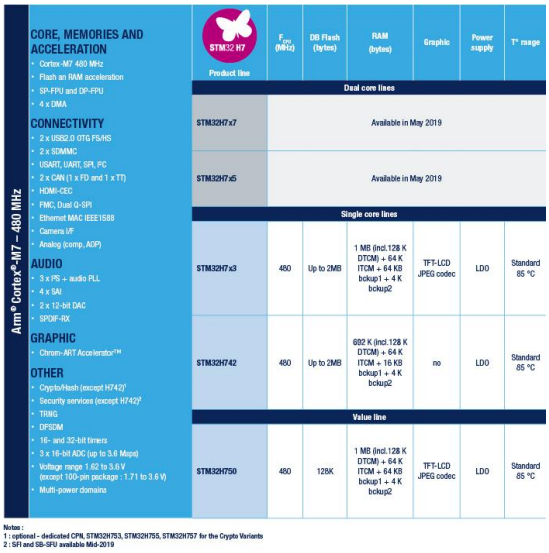
STM32F7 využívá jádro čipu ARM Cortex-M7. Jeho maximální možná rychlost je garantovaná nezávisle na tom, jestli program běží z vestavěné flash paměti nebo z externí. Celá série využívá pouze jeden takt procesoru, a to 216 MHz. Série poskytuje možné přepnutí jednotlivých částí desky do STOP módu, což snižuje energetické nároky. Velikou výhodou je přenositelnost kódu mezi sérií STM32F7 a STM32F4. Ta je zaručená díky zpětné kompatibilitě instrukčního setu jádra Cortex-M7 s jádrem Cortex-M4 a díky stejnému zapojení pinů. Jedinou překážkou je provedení ve dvou pouzdrech, používající 64, nebo 100 pinů (pokud oba mikrokontrolery obsahují stejné používané periferie). Další velkou výhodou oproti předešlé sérii je maximální velikost flash paměti až 2 MB s možností „čtení během zápisu“ (není implementováno pro všechny zástupce série, viz obrázek 4) [4].

[1] Napájecí napětí 3,3 V při teplotě 25 °C

4: Rozdělení série STM32F7 [4]

2.1.4 Série STM32H7

Série STM32H7, stejně jako předešlá řada, využívá jádro ARM Cortex-M7 s maximální rychlostí nezávisle na tom, odkud je program spouštěn. Takt má ale dvakrát větší, až 480 MHz (viz obrázek 5). Energetické nároky dosahují 275 μA/MHz. Je možné snížit energetickou náročnost při STOP módu. STM32H7 se částečně shoduje se zapojením pinů STM32F7, proto je i částečně kompatibilní s předešlými sériemi. Oproti předešlým sériím jsou implementovány bezpečnostní protokoly. Objevují se pouze u nejnovějších zástupců série, kteří mají integrovaný crypto/hash procesor [5].



5: Rozdělení série STM32H7 [5]

2.1.5 Vývojová deska STM32F446RE Nucleo-64

Pro návrh akvizičního signálu byla vybrána vývojová deska STM32F446RE Nucleo-64 (dále referované jen jako „nucleo“) [14].

Získávání nebo zachytávání signálu na platformě SMT32 je možné několika způsoby, podle typu mikrokontroleru. Práce se zabývá pouze výše zmíněnou vývojovou deskou, ostatní příslušníci rodiny 32-bit jsou pouze její variací.

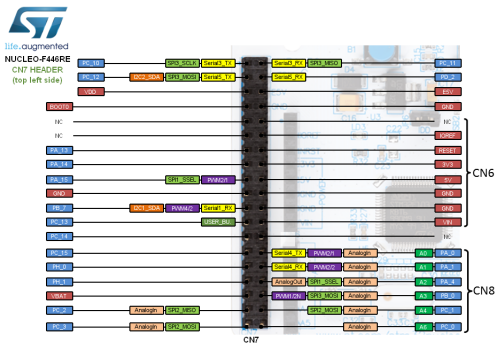
Mirokontroler je opatřen vícero standardizovanými komunikačními rozhraními (sběrnice SPI, I^2C atd.), ADC a GPIO.

Pro externí periferii je podstatné vzít v úvahu komunikační schopnosti dané periferie. Pokud je možná komunikace se zařízením pomocí jednoho z vestavěných rozhraní, pak je vhodné ho použít.

Nucleo používá procesor s jádrem ARM® 32-bit Cortex®-M4 CPU with FPU a je opatřeno paměťmi flash a SRAM o velikostech 512 kB a 128 kB. Frekvence procesoru dosahuje 180 MHz. Na stejné frekvenci běží i dva časovače, které jsou dále použity v návrhu systému. Mikroprocesor má další časovače (dohromady 17) s maximální rychlostí 90 MHz. GPIO využívají maximální rychlosti 90 MHz pro změnu logické hodnoty. Kromě dvou pinů jsou všechny s tolerancí 5 V, jinak se využívá napětí 3,3 V. Jak už bylo zmíněno, interní ADC využívá aproximačního principu a je tvořen kapacitní sítí. Na desce jsou tři 12bitové převodníky, každý s rychlostí 2,4 MSPS, dohromady pracující až na 7,2 MSPS na 24 kanálech. Převod ADC používá vlastní oddělené hodiny nezávislé na systémových. Pro účely testu systému bude využit interní DAC. Nucleo má dva 12bitové [14].

Na desce jsou i sběrnice. SPI sběrnice má maximální rychlost 45 Mbit/s. V návrhu nebude nepoužita. Cílem implementace GPIO je univerzálnost, a uvažované ADC i přístroj pro zpracování signálu ji nemusejí podporovat.

Obrázky 6 a 7 ukazují vyvedení jednotlivých periferií na piny nuclea. Je zde vidět, že jednotlivé periferie nelze používat společně, pro překrytí na daném pinu.

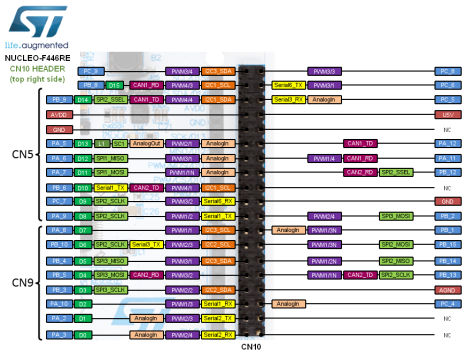


6: Připojení periferií na piny levé části desky [6]

2.2 Analogově-digitální převodníky

Tato kapitola je věnována principům ADC převodníků. Popíše princip převodníku používané–ho platformou STM32 a dalšího, který by mohl sloužit jako substituce vestavěného převodníku pro signály s vyšší frekvencí.

Analogově-digitální převodník je součástka, která převádí analogový signál do digitální (číslicové) reprezentace [15]. Přesnost převodu je závislá na počtu kvantovacích úrovní a převod je zatížen řadou chyb (například chybou kvantování) [7].



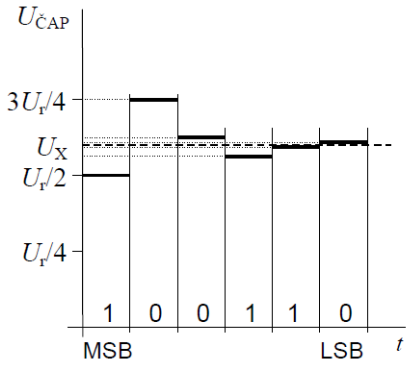
7: Připojení periferií na piny pravé části desky [6]

2.2.1 Převodník s postupnou aproximací

Vestavěný převodník na platformě STM32 je s postupnou aproximací [16]. Převod uskutečňuje napěťový komparátor. Převodník má v sobě napěťovou referenci U_r, se kterou komparátor porovnává vstupní, měřené napětí U_x. Pokud je vstupní napětí větší než polovina napěťové reference, pak se do aproximačního registru zapíše jednička, v opačném případě nula.

Pro vyšší přesnost se využívá více bitů aproximačního registru. Každý další bit se vyhodnocuje v následujícím cyklu. Převod tedy trvá stejný počet cyklů, jako je počet bitů registru. Jinými slovy, pokud se požadujeme vyšší přesnost, děje se tak na úkor rychlosti převodu.

V každém cyklu se porovnává hodnota z minulého porovnání, zvětšená (pokud byla v minulém cyklu zapsána do registru jednička), nebo zmenšená (pokud byla do registru zapsána nula) o polovinu referenčního napětí. Takže v prvním cyklu vždy porovnáváme napětí vstupní s polovinou referenčního napětí. Pokud byla zapsána hodnota jedna do registru, pak to znamená, že vstupní napětí je z obou vyšší, a proto pro další porovnání přidáme k původní polovině napěťové reference ještě čtvrtku napěťové reference. Hodnota, se kterou v tomto cyklu porovnáváme vstupní napětí je tři čtvrtě hodnoty referenčního napětí. Všechny další cykly se chovají stejně. (viz obrázek 8, $U_{\text{ČAP}}$ je napětí digitálně-analogového převodníku) [7]. Po celou dobu převodu musí být vstupní napětí neměnné, což vyžaduje předřazení obvodu sample and hold[2].

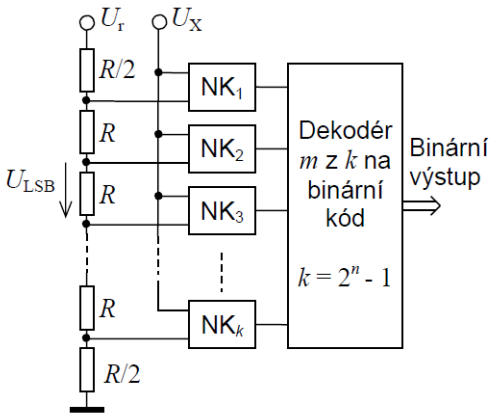


8: Princip aproximace v jednotlivých cyklech aproximačního ADC [7]

[2]Blíže popsáno v [7]

2.2.2 Komparační převodník

Komparační analogově-digitální převodník pracuje stejně jako převodník s postupnou aproximací na bázi porovnání vstupního napětí U_x s napětím referenčním U_r. Rozdíl je v tom, že více bitová přesnost je docílena už v jednom cyklu. Vstupní napětí je přivedeno na kaskádu rezistorů. Všechny rezistory R mají stejnou hodnotu, kromě prvního a posledního, které mají hodnotu poloviční oproti ostatním $R/2$. Napětí v uzlech mezi rezistory se přivede na vstupy napěťových komparátorů (NK). Všechny porovnávají dělená napětí s napěťovou referencí. Výsledky komparátory zapisují do registru (viz obrázek 9: Blokové schéma komparačního ADC [7]). Rychlost převodu může být až desetina nanosekundy [7]. Díky rychlosti, a hlavně díky převodu v jednom taktu je tento převodník nejlepším řešením pro použití jako náhrada pomalého vestavěného aproximačního převodníku na desce. Nevýhodou je vysoká cena [7].



9: Blokové schéma komparačního ADC [7]

2.3 Digitálně-analogový převodník

Pro výstup signálu z nuclea v analogové podobě slouží integrovaný digitálně analogový napěťově výstupní 12bitový převodník. Výstup může být 12 nebo 8bitový s možností levého nebo pravého zarovnání. Výstupy jsou na mikrokontroleru dva a každý má vlastní převodník. Výstup může být rozdělen nezávisle mezi oba převodníky nebo sdružený, při nastavení synchronizační operace. Převod může být spuštěn externím zdrojem.

Spouštění každého převodníku je možné nastavením příslušného bitu (ENx bit) registru DAC_CR. Převodník je spuštěn následně po příkazu zapnutí. Oba převodníky mají předřazený buffer, který slouží pro postupné nastavení hodnot (BUFFx bit registru DAC_CR).

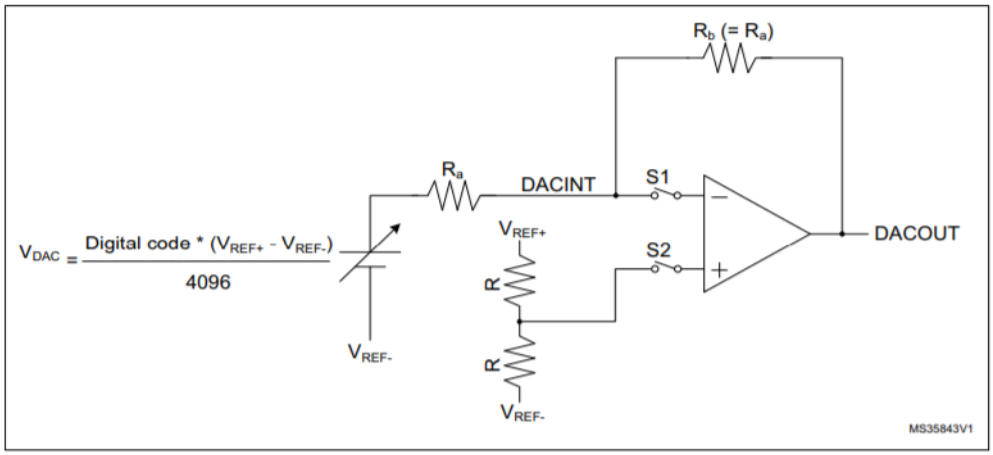
Pro jednotlivé použití převodníků má každý převodník k dispozici tři registry pro zápis digitálních hodnot. Pro 8bitový mód je možnost pouze pravého zarovnání a příslušný registr je DAC_DHR8Rx. 12bitové módy mají registry DAC_DHL12Lx (levé zarovnání) a DAC_DHR12Rx (pravé zarovnání). Po zapsání hodnot uživatelem do těchto registrů jsou data přesunuta do registru DHRx[3] a následně automaticky, programem, nebo externím příkazem nahrána do registru DORx. Data není možné zapsat přímo do DORx registru, ale musejí být předána přes DHRx registry. Po předání dat je výstup spuštěn v dalším taktu.

Výstupem je napětí o velikosti přepočtu dané hodnoty na poměrnou část referenčního napětí $\mathrm{DAC}_{\mathrm{OUTPUT}} = U_{\mathrm{ref}} \frac{\mathrm{DORx}}{4096}$ [10].

Po celou dobu jsou data uchována v paměti RAM a procesor přenáší data na DAC (DHRx→DORx).

[3]Data holding registr - registr pro skladování dat

Rychlost převodníku je ovlivněna použitím zásobníku (viz obrázek 10). Pokud není použit, pak je rychlost závislá na konstantě RC[4] [8].



10: Schéma zapojení DAC [8]

2.4 GPIO

Nucleo má 114 I/O pinů, z nichž 112 je s tolerancí 5 V[5]. Každý z pinů je možné nastavit jako výstupní, vstupní nebo jako vstupně výstupní. Pro nastavení výstupu je možná kombinace push-pull, nebo open-drain s, nebo bez možností pull-up, nebo pull-down. Vstup je plovoucí s, nebo bez možností pull-up, nebo pull-down. Maximální rychlost změny logické hodnoty pinů dosahuje 90 MHz.

GPIO se skládá z několika registrů. Všechny registry jsou 32bitové. Čtyři jsou konfigurační, jeden registr pro set a reset, locking registr, dva pro alternující funkce a následně dva registry datové (ODR a IDR).

Každý pin má možnost vyžití vždy jen jedné periferie v danou dobu. To zajišťuje multiplexer, přes který jsou všechny piny zapojené. Ten připojuje funkci periferie na pin a zabraňuje tak ostatním v přístupu na pin.

Čtyři konfigurační registry jsou GPIOx_MODER, kterým je vybírána periferie, GPIOx_OTYPER (push-pull/open-drain), GPIOx_PUPDR (pull-up/down) a GPIOx_OSPEEDER, jímž je nastavena rychlost periferie. Propojení alternující funkce periferie s pinem zajišťují registry GPIOx_AFRL a GPIOx_AFRH.

Konfigurační registry a registry alternujících funkcí AFRL/H je možné zamknout proti modifikaci. To umožňuje registr GPIOx_LCKR, který je nastavován sedmnáctým bitem tohoto registru (GPIOx_LCKR[16]), prvních šestnáct bitů je pro konfiguraci zámku. Tento registr je možné nastavovat pouze zápisem celého slova (word – 32bitový integer). Změna konfigurace pinu (konfigurační registry) je možná pouze pro resetování tohoto registru. Do tohoto registru je možné, jak zapisovat, tak z něj i vyčítat.

„Input/Output“ data registry jsou 32bitové registry, které zajišťují zápis a čtení logické hodnoty pinu. Oba registry jsou sice 32bitové, ale uživatel přistupuje pouze k prvním 16 bitům. Zatím co do registru ODR je možné zapisovat a je možné ho i číst, registr IDR slouží pouze pro čtení. Každý z registrů obsluhuje vlastní piny, které jsou přiřazené jednotlivým bitům v registru. Pokud tedy zapíšu do registru GPIOA_ODR hodnotu 1, všechny piny

[4]Zde se jedná o násobek velikosti odporu rezistoru a výstupní kapacity pinu DACOUT
[5]Je možné na ně připojit pěti voltový vstup signál.

k němu přiřazené budou mít hodnotu logická nula kromě pinu PA_0, na kterém bude logická jedna.

Bity registru ODR je možné ovlivňovat set/reset registrem GPIOx_BSRR [10].

2.5 Časovače a zdroje hodinového signálu

Nucleo má čtyři zdroje hodin, dva oscilátory HSI[6] a HSE[7] a dva hlavní PLL[8]. Dále je možné připojit dva další oscilátory které jsou součástí desky - (LSI[9] RC, 32 kHz) oscilátor a (LSE[10] krystal, 32,768 kHz). Každý s nich je možné nezávisle zapnout, pokud nejsou používané. Dále se zdroj hodin násobí a dělí podle potřeby jednotlivých periferií. Jednotlivé domény mají rozdílné frekvenční nároky a maximální rychlosti. AHB má obecně maximální frekvenci 180 MHz, APB2 90 MHz a APB1 pouze 45 MHz. Tyto zdroje hodin je možné použít jako systémové hodiny. Všechny hodiny periferií jsou řízeny systémovými hodinami, kromě některých (jejich seznam je v [10]).

HSE je generován externím krystalovým, nebo keramickým oscilátorem, nebo uživatelem nastaveným signálem. Zdroj hodinového signálu musí být v tomto případě umístěn, co možná nejblíže pinu oscilátoru pro minimalizaci chyb (například stabilita). Tato možnost je zařazena na desku pro připojení vysoce stabilního zdroje signálu, pokud je potřeba. To je možné využít právě pro real-timeové[11] aplikace (viz 2.7). HSE je možné použít pro generování přerušení běhu programu.

HSI hodinový signál generuje RC oscilátor s frekvencí 16 MHz. Lze ho použít přímo jako hodinový signál systémových hodin nebo může být připojen jako zdroj PLL. Není tak stabilní jako mohou být externí zdroje hodin, oproti tomu má ale výhodu, že je integrován, a navíc má rychlejší startovací čas. Stabilita a přesnost závisí i na výrobním procesu, díky tomu se může hodinový signál na dvou mikrokontrolerech stejného typu odchylovat. Výrobce kalibruje všechny své HSI s jednoprocentní odchylkou[12].

PLL funguje tak, že jeden ze zdrojů HSI nebo HSE je násoben a dále dělen na tři výstupy. Prvním výstupem je systémový hodinový signál (max. 180 MHz), druhý je používaný USB OTG FS nebo SDIO (max. 48 MHz, periferie neřízená systémovými hodinami) a třetí je používán na další periferie neřízené systémovými hodinami nebo může být použit také jako zdroj systémových hodin. PLL není možné nastavovat během běhu programu, nebo po jeho spuštění.

Systémové hodiny nemohou být zastaveny po spuštění z HSI nebo PLL. Změna systémového signálu je možná pouze, pokud je nový generátor systémového hodinového signálu je ve stavu „ready“ [10] (schéma pro nastavení systémových hodin si je možné prohlédnout na obrázku 18).

Časovače TIM1 a TIM8 jsou 16bitové čítače s funkcí automatického znovu spuštění. Jsou nezávislé, ale mohou být spolu synchronizovány. Čítače inkrementují, dekrementují nebo mohou pracovat v kombinaci těchto modů. Prescaler[13] těchto časovačů je možné měnit v průběhu programu (mohou nabývat hodnot z intervalu 1 - 65536).

Registry zaručující běh časovače jsou čítací registr TIMx_CNT, prescaler registr TIMx_PSC, znovu spouštěcí registr TIMx_ARR a opakovací registr TIMx_RCR. Všechny registry

[6] High speed internal clock signal - vysoko rychlostní interní hodinový signál
[7] High speed external clock signal - vysoko rychlostní externí hodinový signál
[8] Phase-locked loop - fázový závěs
[9] Low speed internal clock signal - nízko rychlostní interní hodinový signál
[10] Low speed external clock signal - nízko rychlostní externí hodinový signál
[11] Reálný čas
[12] Při teplotě 25 °C
[13] Elektrický obvod jehož částí je PLL a je používán pro změnu (zvýšení) frekvence vstupu

mohou být i za běhu programu a časovače čteny, ale je možné do nich i zapisovat[14].

Funkce jednotlivých registrů je celkem jednoduchá, ale pro přehled čítací registr po spuštění časovače začne počítat se zpožděním jednoho taktu. Pokud časovač inkrementuje (v ostatních modech je funkce analogická), pak s každým taktem (určuje prescaler) se jeho hodnota zvýší o jedna. Při dosažení hodnoty ARR registru je vygenerována událost (může být použito pro přerušení) a registr je vynulován. Proces se pak opakuje. Opakování probíhá v nekonečné smyčce, pokud není nastavena hodnota opakovacího registru, který určuje počet opakování (plus jedna, viz [10]).

Při změně hodnot registrů běží všechny registry s novou hodnotou do přetečení kromě opakovacího registru, který si svoji hodnotu změní až po přetečení.

Jednotlivé časovače mohou mít čtyři různé zdroje hodin, interní a externí, obě varianty je možné mít ve dvou provedeních se zdrojem hodinového signálu nebo jako trigger. Pokud je zvolen například interní trigger[15], je možné připojit jeden časovač k druhému a brát ho jako prescaler pro druhý. Tedy pokaždé, když první časovač vygeneruje událost, je spuštěn „tik“ druhého.

2.6 Přerušení

Přerušení je odchýlení od normálního běhu programu. Přerušení tasku[16] je závislé na externí (nebo interní) události. Pokud daná událost nastane, pak se běh programu přeruší a procesor začne vykonávat rutinu po přerušení. Před během programu uživatel musí stanovit, co se má stát, pokud daná událost nastane. K nastavení přerušení je použit NVIC[17].

Nucleo obsahuje 96 kanálů přerušení. Jednotlivé přerušení je se mohou časově překrývat (než se ukončí proces spuštěný přerušením, nastane další událost), a proto je k dispozici 16 priorit. Přerušení se stejnou prioritou jsou spouštěná záhy po sobě, jak nastaly události, jež vyvolaly spuštění.

Pro vyhodnocení externích událostí je k dispozici 23 detektorů hran signálu. Trigger detektoru, takzvaný EXTI[18], může být nastaven na detekci náběžných, sestupných hran nebo jejich kombinace.

Pro generování přerušení je nutné nastavit příslušný bit maskovacího registru, nakonfigurovat a povolit přerušení. Nastavení se provede nastavením maskovacího registru EXTI_IMR s 23 bity pro možnou volbu přerušení, výběr triggeru je v registrech EXTI_RTSR (obsluha náběžné hrany) a EXTI_FTSR (obsluha sestupné hrany). Je nutné nastavit NVIC IRQ[19] pro detekci přerušení softwarem.

GPIO je propojené s externím přerušením. Jednotlivé piny jsou sdruženy pod jednotlivé registry. Například pro všechny GPIOx_PIN_0 (piny připojené k nultému bitu registru GPIOx) mají společné vyhodnocení EXTI0 [10].

2.7 RTOS

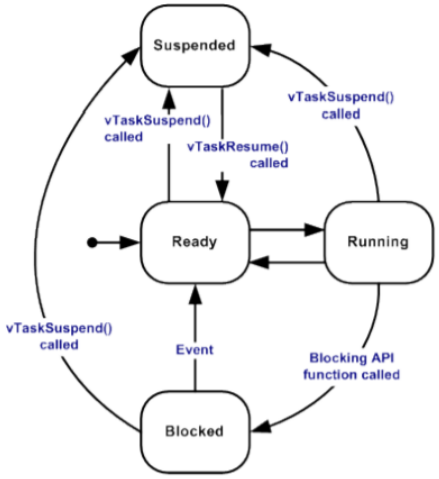
RTOS je zkratka pro „Real-Time Operating System“[20]. V normálním běhu programu jsou jednotlivé tasky vykonávány sekvenčně podle toho, jak jsou za sebe řazeny v kódu [17]. RTOS je obslužný program - operační systém, který je schopen podle svého vnitřního nastavení

[14] Pro další než základní funkce časovače je možné nahlédnout do [10] a seznámit se s dalšími registry
[15] Trigger - spouštěč
[16] Task - úkol
[17] Nested vector interrup controller - vnořený řídící vektor přerušení
[18] External interupt/event controller - kontroler externích přerušení
[19] Interrupt request - žádost přerušení
[20] Operační systém v reálném čase

přepínat vykonávání jednotlivých úkolů mezi thready[21]. Pokud tedy několik procesů v programu potřebuje souběžné spuštění, je nutné pro tuto funkci použít operační systém, který bude přepínat mezi vlákny. Operační systém využívá scheduler[22], který přiřazuje procesoru task podle určitého algoritmu. Přepínání je dvojí, takzvané preemptivní a nepreemptivní. První z nich přiděluje tasky v závislosti na jejich prioritě. Příkladem takového algoritmu je „Round-Robin scheduling“, který přiděluje jednotlivé úkoly procesoru na předem stanovenou dobu (doby vykonávání tasků nemusí být pro všechny tasky stejné). Pokud task nedokončí své provedení v čase jemu schedulerem přiděleném, jeho provádění se pozastaví [18]. Druhý se řídí podle jednotlivých úloh. Každý task je za celou dobu svého „života“ v jednom ze čtyř stavů [9]:

- Ready - task je připraven na spuštění po svém vytvoření.
- Running - task je vykonáván procesorem.
- Blocked - v tomto stavu task není vykonáván, čeká na určitou událost, případně na uplynutí doby, po kterou nemůže být vykonáván.
- Suspended - tento stav musí být vynucen (například událostí nebo jiným taskem) a task v něm nemůže přejít do aktivního stavu do té doby, než je mu znovu povoleno jedno spuštění (událostí nebo taskem).

Na obrázku 11 je možné vidět stavový diagram jednoho úkolu a jednotlivé situace, díky kterým může úkol změnit stav. Task je funkce typu void nebo void pointer. Funkce, která musí obsahovat nekonečnou smyčku. Ukončení tasku je řešeno jiným taskem nebo ukončením programu. Stejně tak může task vytvořit jiný task.



11: Stavový diagram tasku [9]

RTOS je operační systém, který zajišťuje deterministické vykonávání jednotlivých úkolů [19]. Pokud je od výsledného programu vyžadováno, aby byl řídícím programem v reálném čase (například použití pro vyhodnocování zpětné vazby motoru), pak je nutné, aby byl použit operační systém, který skutečně bude přepínat mezi jednotlivými procesy s přesnou periodou.

[21] Thread - vlákno
[22] Scheduler - přepínač

2.7.1 FreeRTOS

FreeRTOS je operační systém dostatečně malý, aby mohl běžet na mikrokontrolerech [19]. Kernel je poskytovaný pod licencí MIT[23] [21]. Kernel využívá jen některé funkcionality RTOS, například , komunikaci mezi vlákny, synchronizaci úkolů a časování [19].

Použití kernelu, který zajišťuje komunikaci mezi thready, může vést k problémům s pamětí. Pokud by vlákna nebo jednotlivé tasky přistupovaly na stejnou adresu paměti, mohl by jeden druhému přepsat potřebná data. Pro bezpečnou práci s pamětí samotný kernel využívá paměť pouze jedním threadem a nemožností přístupu ostatních, ale taky předávání informací mezi nimi.

Způsoby komunikace mezi thready jsou fronty, semafory a mutexy. Fronta kernelu FreeRTOS je zásobník typu FIFO, který může zprostředkovávat komunikaci mezi tasky nebo mezi taskem a přerušením. Paměť využitá frontou je alokovaná přímo kernelem. Frontu při vytvoření definuje počet prvků (statická délka) a typ prvku. Data jsou kopírována do fronty a z fronty (funkce put/pop), což je paměťově náročné. Proto je doporučené používat frontu ke komunikaci pouze v nezbytných případech.

Existuje dvojí typ semaforu. Používá se pro signalizaci některému z dalších tasků. Binární semafor je proměnná používaná přerušením nebo taskem k signalizaci určité události tasku. Čítací semafor se používá pro záznam počtu událostí.

Mutex je zvláštní binární semafor přiřazený k určitému zdroji (například DAC) signalizující tasku, zda je zdroj využíván jiným taskem, nebo zda je možné se zdrojem pracovat a přistupovat k němu [9].

Kernel alokuje paměť za běhu programu dynamicky. Jsou podporovány i klasické funkce alokace ze standardní knihovny malloc/free. Jelikož použití těchto funkcí nemusí být bezpečné, jsou implementovány obdobné funkce v kernelu pvPortMalloc a vPortFree.

Časovače kernel poskytuje jako softwarové řešení. Jsou dvojího typu, jednorázové a opakované. Po uplynutí doby časovače spustí obslužnou funkci čítače. Jednorázové časovače spustí obslužnou funkci pouze jednou. Pro opakované spuštění je nutné je resetovat. Vytvoření časovače je možné kdykoli, ale spuštění časovače nastane až se spuštěním scheduleru. Periodu je možné změnit kdykoli v průběhu spuštění časovače, ale změna nastane až v další periodě [17].

[23] Licence softwaru umožňující i volné komerční využití části, celého i změněného softwaru [20]

3 Praktická část

Jak bylo výše zmíněno, pro demonstraci implementace akvizičního systému byla vybrána vývojová deska STM32F446RE NUCLEO-64.

Pro vytvoření systému na zpracování signálu, který má ještě pracovat rychle, je nutné nejprve diskutovat možnosti příjmu signálu, které má deska k dispozici. Pokud budeme uvažovat analogový signál, je přirozeným výběrem periferie ADC, jak bylo výše popsáno (viz podkapitola 2.2.1). Tento převodník ale není ze své podstaty nejrychlejší. Lze tedy uvažovat možnost připojení externího ADC, například komparačního převodníku, který zajišťuje převod v jednom taktu (viz podkapitola 2.2.2).

V tomto případě už nemluvíme o zpracování analogového, ale digitálního signálu. Ten je možné přijímat integrovanými komunikačními rozhraními, jako jsou různé sběrnice SPI, I^2C, CAN a jiné, DCMI, USART, nebo je možné přijímat digitální signál na GPIO rozhraní (viz podkapitola 2.4). GPIO je dvakrát rychlejší, než sběrnice SPI. Také má tu výhodu, že bere v potaz obecnou periferii, nikoli pouze periferii, která podporuje nejrychlejší komunikační rozhraní.

Pro akviziční systém bylo tedy zvoleno rozhraní GPIO.

Signál nemá cenu zachytávat, pokud s ním nejsou další úmysly – zpracování, zobrazení, uložení. Z tohoto důvodu bylo přistoupeno k podmínce, že nucleo bude signál poskytovat dále. Není důležité, co se se signálem stane po průchodu nucleem. Uvažuje se zde pouze, že předávání hodnoty bude dále obecné (v testu systému je možné nahlédnout na konkrétní případ, viz podkapitola 3.3).

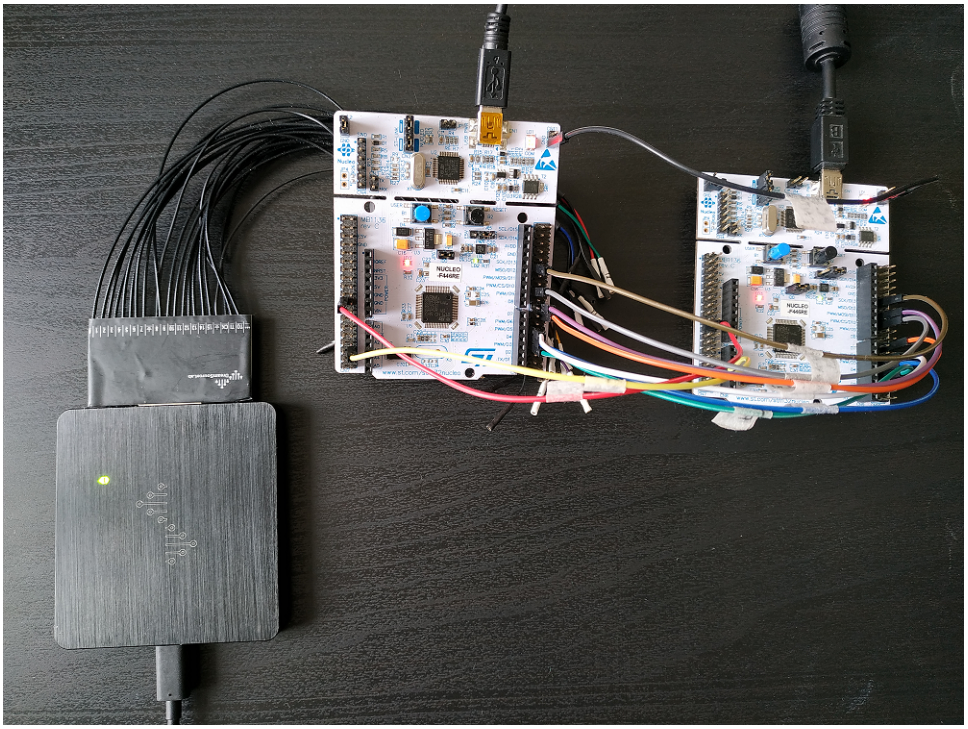
Systém je tedy navržen na zachytávání a poslání signálu pro zobrazení, uschování nebo zpracování. Systém je rozdělen na dvě části, akviziční a vysílací (viz níže).

Struktura akvizičního programu se dvěma větvemi, které obě budou periodicky pracovat, vyhovuje přesně základnímu použití RTOS, což bylo i původním záměrem. Během realizace od něj ale bylo upuštěno. Při testování rychlosti systému bylo zjištěno, že RTOS běh programu pouze zpomaluje. Z důvodu rychlosti nebyl tedy nerealizován program s použití FreeRTOS kernelu.

Nepoužití operačního systému je dokonce nutné. Pokud by byl použit operační systém v režimu stejných priorit obou procesů, pak by oba měly přesně nastavené časy, po které se mohou vykonávat, a následně by byl prováděn druhý z nich. Není možné přerušit oba procesy v jejich průběhu (jednotlivé tasky je možné přerušit, ale nikoli pokud by to rušilo jejich běh, viz kapitola 2.7). A proto není možné ani použití rozdílných priorit procesů, protože by jeden z nich mohl být vždy přerušen. Například by jeden task vyčítal hodnotu z fronty během jejího zápisu do fronty, nepřečetl by žádnou hodnotu, protože by na dané adrese žádná hodnota nebyla. Jiným asi vhodnějším příkladem je převod ADC, pokud by se měnila perioda dotázání na vzorek, díky přerušení taskem pro vyčtení, vzorkovací perioda by nebyla stálá.

Navíc scheduler sám o sobě je soubor instrukcí, které je nutné vykonat. Pokud je cílem dosáhnout rychlejších výsledků a dovoluje to nenáročnost interakce jednotlivých tasků, pak je nutné RTOS vyřadit.

V na obrázku 12 je zobrazené praktické zapojení aparatury.



12: Snímek zapojení aparatury pro měření rychlosti akvizičního systému, zleva: logický analyzátor, nucleo použité pro akviziční sytém a generátor signálu vytvořený z nuclea

3.1 Výběr prostředí pro implementaci

Kernel FreeRTOS je napsaný v jazyku C, stejně jako HAL poskytovaný k nucleu výrobcem. V dnešní době je možné využít možností middlewaru, který usnadňuje použití. Zpravidla bývá napsaný v nějakém z vyšších jazyků. Byla vyzkoušena implementace akvizičního systému na dvou middlewarech - MBED[24] napsaný v jazyce C++ a Zerynth[25], který je v jazyce Python.

Zerynth má možnost práce s FreeRTOSem a díky výhodám jazyka Python, jako je jednoduchost a rychlost psaní programu, se zdál nejlepší volbou. Bohužel, při pokusu o zvýšení rychlosti programu, se objevil problém s tím, že middleware implementuje správu času pouze v milisekundách. Další nevýhodou byla podpora pouze pro desku STM32F401RE, která má výrazně nižší frekvenci, než se kterou pracuje nucleo. Pro tyto skutečnosti bylo od implementace v tomto middlewaru upuštěno.

Pro práci v Pythonu je nejvýznamnějším middlewarem MicroPython[26]. Chybí mu ale podpora FreeRTOS (je možné implementovat, ale přesahuje to náročnost této práce). Proto další volbou bylo použití MBED.

Jazyk C++, v kterém je MBED napsaný, přináší stejně, jako Python výhodu vyššího a modernějšího jazyka oproti C. MBED podporuje ovládání v microsekundách. Po zjištění, že FreeRTOS není vhodný pro takovou úlohu, byl akviziční systém vytvořen pomocí tohoto middlewaru.

I od toho řešení bylo nutno upustit. MBED je poskytován jako opensource projek, nicméně standardní knihovna není zveřejněna. Uživatel nemá možnost výběru určité periferie. Nucleo má sedmnáct časovačů, ale pouze dva z nich pracují s frekvencí 180 MHz (viz podkapitola 2.5). Není ale jasné, který z nich middleware použil, a nedá se to specifikovat.

[24] viz [22]
[25] viz [23]
[26] viz [24]

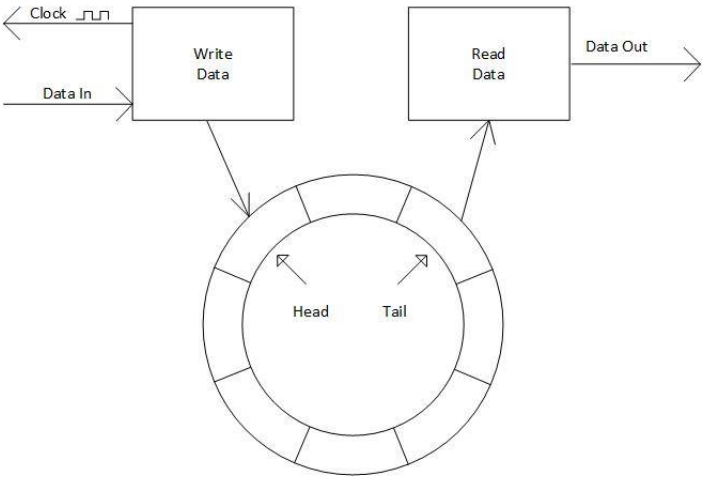
Ve výsledku byl použit pro vytvoření akvizičního systému jazyk C, ve kterém výrobce poskytuje standardní knihovnu pro nucleo. Pro nastavení periferií byl zvolen poskytovaný HAL. V řídící smyčce nebyly používány funkce pro ovládání periferií HAL. STMicroelektronics poskytuje standardní knihovnu, ve které je možné nahlédnout do kódu, kde jsou jednotlivé funkce ošetřené pro bezpečné volání a mají v sobě příkazy pro zjišťování, zda je daná periferie zapnutá apod. To jsou procesy, které program zpomalovaly, a proto jsou výsledné příkazy použité v řídící smyčce akvizičního systému jsou omezené pouze na ovládání jednotlivých nezbytných registrů periferií.

3.2 Analýza systému

Pro realizaci byla zvolena možnost implementace cache paměti. Při popisu systému jako „black box“ lze hovořit o systému, který na jedné straně zachytí signál s určitou přenosovou rychlostí $R_{b,in}$ a na druhé straně produkuje upravený signál s přenosovou rychlostí $R_{b,out}$. Motivací pro výběr tohoto systému, který nezpracovává, ale pouze zachycuje a před zpracovává signál a upravuje rychlost pro další systém, který s produkovanými daty bude dále pracovat, bylo zachování vysoké vstupní rychlosti. Ta je závislá na časové složitosti obslužného programu (viz podkapitola 3.3.1).

Pro návrh akvizičního systému bylo vyřazeno vzorkování signálu. Práce je zaměřená tedy až na vzorkovaný signál.

Obslužný program je rozdělen na dvě části (viz obrázek 13). Jedna část programu se stará o akvizici signálu a druhá o „interpretaci“ – vyčítání vzorků. Obě rutiny jsou na sobě nezávislé. Data jsou v programu uchovávána v kruhové frontě v poli, které je staticky alokované před během programu.



13: Schéma akvizičního systému – sdílení paměti částmi kódu

Embedded programování se vyznačuje odporem k alokování paměti za běhu programu z důvodů možných chyb a ohrožení bezpečnosti, program by se nemusel chovat deterministicky, stejně dvakrát za sebou ve stejných situacích. Proto se programátor uchyluje k jiným řešením, v tomto případě je to alokace největší možné paměti, kterou má nucleo k dispozici. Je možné si to dovolit, nejen proto, že celé nucleo je používané jen k akvizici signálu, ale také protože program ve svém běhu nikde dále alokaci nepoužívá.

Pokud je dokonce známá velikost alokované paměti, používané ke skladování přijatých vzorků, lze také podle přenosových rychlostí $R_{b,in}$ a $R_{b,out}$ určit, jak dlouho bude moci program běžet v režimu větší vstupní rychlosti, než dojde k vyčerpání paměti. Po vyčerpání

paměti program přestane zapisovat vzorky do fronty. Není to řešení vhodné, protože program nedává žádné upozornění, že přestane zapisovat některé vzorky (při každém čtení z fronty se zápis zase zprovozní). Nicméně pokud se program bude používat podle časové závislosti, která říká, po jak omezenou dobu lze program používat bez problému při daném poměru přenosových rychlostí, bude program pracovat správně.

První blok programu – akviziční větev, je tvořená podmínkou přetečení čítače (viz Výpis kódu 1). Tedy pokud časovač dosáhne určité hodnoty, blok programu se spustí (viz podkapitola 2.5). Prvním příkazem v tomto bloku musí být resetování čítače. Pokud by se čítač resetoval až po zbytku kódu v bloku, prodlužovala by se doba dalšího spuštění bloku. Následně je změněná hodnota hodin sběrnice[27] (realizováno bitovou disjunkcí). Dalším příkazem je přečtení aktuální hodnoty na komunikačním rozhraní periferie a zapsání do kruhové fronty.

```
1 if (TIMx->SR & TIM_SR_UIF)
2 {
3   TIMx->SR = 0x0000;           // timer reset
4   GPIOx->ODR ^= GPIO_PIN_x;    // change clock ouput value
5   set_que(&que, GPIOx->IDR);   // function for read input value
6   // and store it in queue
7 }
```

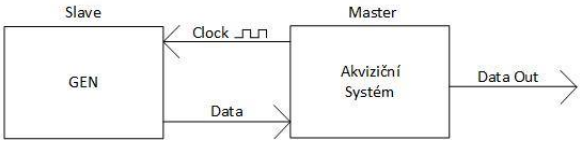
Výpis kódu 1: Časově podmíněný blok pro spuštění změny hodnoty hodin sběrnice a vyčtení hodnoty sběrnice

Druhý blok – vyčtení hodnoty, se spustí stejným mechanizmem jako akviziční část. Časovače jsou ale pro oba bloky rozdílné (jsou použity časovače TIM1 a TIM8, které pracují s taktem 180 MHz). Časovač je následně po spuštění bloku resetován a následuje vyčtení hodnoty z fronty.

3.3 Testování akvizičního systému

Cílem testování akvizičního systému bylo zjistit, zda byl vytvořen systém, který dokáže přijmout vzorky signálu, uskladnit je a následně je poslat dále s největší možnou rychlostí. Zároveň musí být signál bezchybně přenesen.

Byl vytvořen generátor signálu, který bude předávat vzorky určitého signálu akvizičnímu systému v potřebném formátu z druhého nuclea. Obě zařízení byla propojena 8bitovou sběrnicí, tvořenou propojenými piny, doplněnou signalizačním hodinovým signálem. Dohromady pracují jako SPI sběrnice, kde akviziční systém je označován jako Master a generuje hodinový signál pro generátor signálu, který je v režimu Slave. Sběrnice je realizovaná přes digitální rozhraní GPIO (viz obrázek 14).



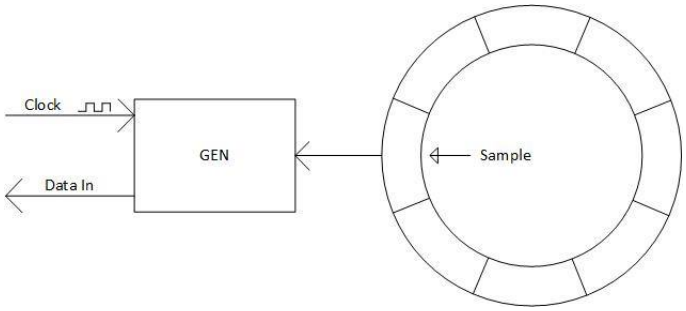
14: Schéma soustavy pro testovaní komunikace

Jako generovaný signál byla zvolena pro testování funkce sinus a to z důvodu periodicity a známému průběhu. Než se program dostane do hlavní smyčky, vygeneruje jednu periodu funkce sinus pro daný počet vzorků. Tyto hodnoty jsou zapsány do globálního pole a následně při každém dotázání Mastera (režim detekce náběžné i sestupné hrany hodinového signálu, viz 2.6) se na sběrnici vynese hodnota funkce, která je další v pořadí. Při změně hodnoty

[27] Hodiny sběrnice slouží k taktování komunikace s periferií poskytující vzorky signálu.

hodinového signálu se vyvolá přerušení běhu programu a je volána funkce pro zapsání aktuální hodnoty sinu na sběrnici (viz obrázek 15).

Pro zjištění kvality funkce systému bylo nutné zjistit maximální přenosovou rychlost, se kterou je systém schopen pracovat správně, a maximální možný rozdíl přenosových rychlostí $R_{b,in}$ a $R_{b,out}$ při správné funkci paměti.



15: Schéma funkce generátoru signálu

3.3.1 Rychlost systému

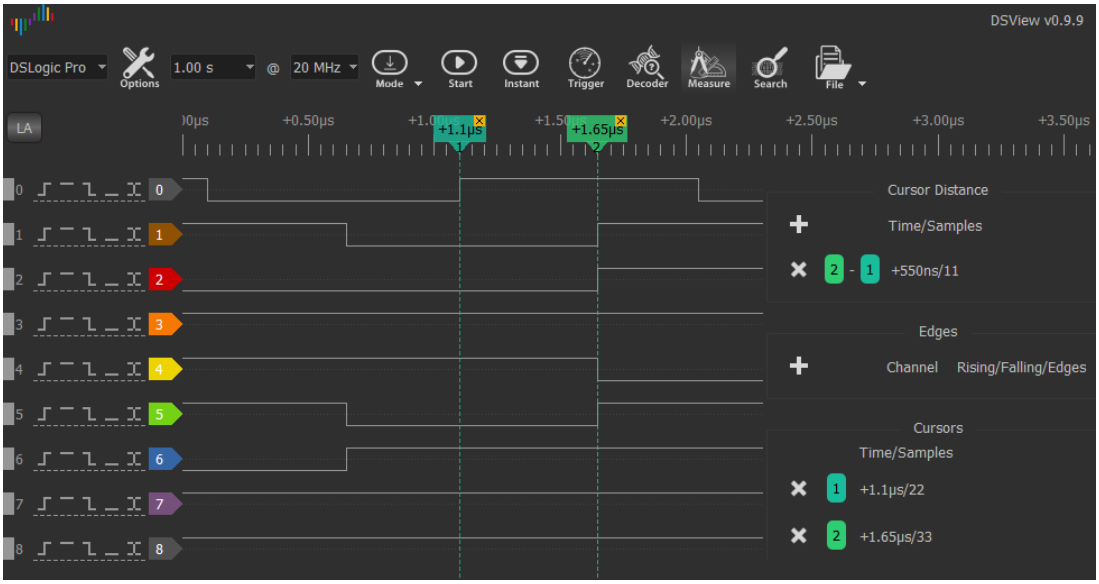
Jak bylo zmíněno výše, přenosová rychlost zápisu je závislá na uživatelem nastavené hodnotě časovače, po jejímž překročení se spustí blok programu pro získání vzorku. Pokud je doba, po kterou časovač plyne, vyšší než doba běhu instrukcí bloku, pak je vše v pořádku. V opačném případě se bude doba běhu bloku prodlužovat a program se zpomalí.

Pro testování hraniční rychlosti běhu celého programu byly nastaveny hodnoty obou časovačů akvizičního systému stejně. Díky tomu se spouští oba bloky programu přímo za sebou v cyklu (přenosové rychlosti se tedy rovnají: $R_b = R_{b,in} = R_{b,out}$). Celkový počet strojových instrukcí tedy určuje rychlost systému.

Nucleo je schopné pracovat s taktem 180 MHz [14]. Počet instrukcí je možné omezit, pokud si pro sběrnici vybereme piny na jednom z registrů (bude se číst a nastavovat pouze registr jedné domény).

Test pro zjištění rychlosti byl proveden dvěma způsoby. V obou případech byla zjišťována maximální možná rychlost zápisu a výčtu hodnoty z fronty. V prvním, na rozdíl od druhého, nebyl použit výčet hodnoty z nuclea (nucleo nepředávalo z paměti vyčtenou hodnotu žádnému dalšímu systému). Měření bylo provedeno bez použití periferie pro vyčtení hodnot z mikrokontroleru kvůli zjištění časové složitosti obsluhy zápisu hodnoty do paměti. Protože nucleo v tuto chvíli nedávalo žádný výstup, bylo možné určit rychlost běhu programu pouze z logického analyzátoru, který detekoval přítomnost nenulového napětí na pinech sběrnice.

Na obrázku 16 je zachycen časový průběh změn logických hodnot na sběrnici, kde „0“ ozna-čuje hodinový signál a následně jsou číslovány jednotlivé bity sběrnice podle pořadí („1“ označuje LSB). Na obrázku jsou vidět i dva markery. První označuje čas změny hodinového signálu a druhý změnu hodnoty na sběrnici. Následně je vidět i jejich časový rozdíl, který čítá 550 ns, což je doba běhu cyklu generátoru. Vzorkovací rychlost je možné určit z časových rozdílů náběžných hran hodinového signálu $R_s = 1$ MSPS, což odpovídá přenosové rychlosti $R_b = 8$ Mbit/s (pro 8bitovou sběrnici).



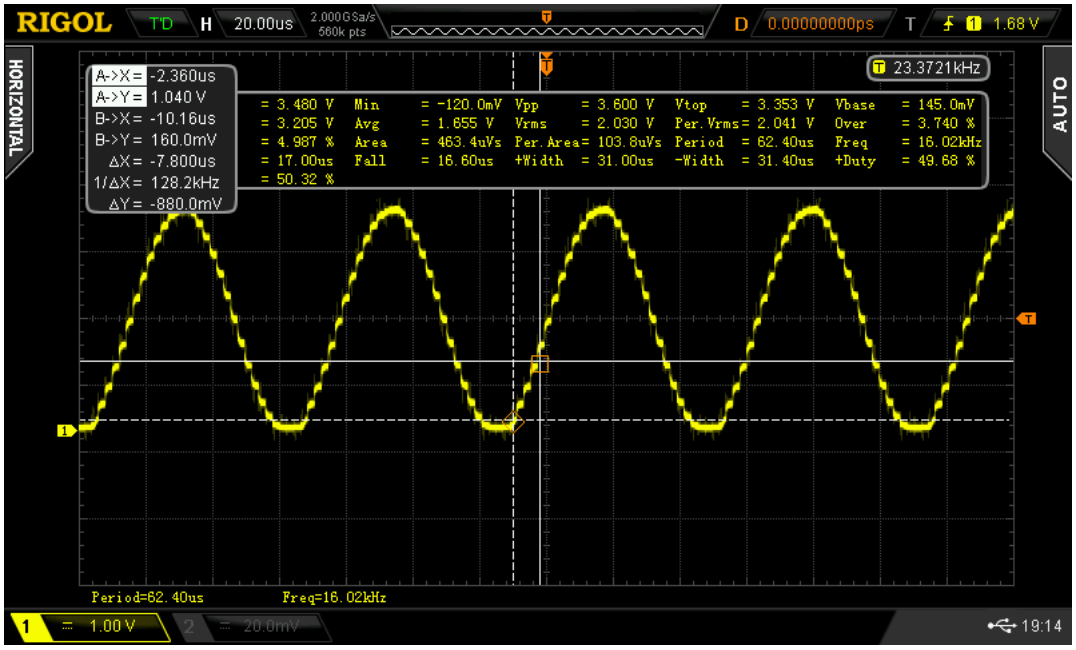
16: Časový průběh logických hodnot jednotlivých částí sběrnice s vyznačeným časovým odstupem změny hodnoty hodinového signálu a datové části sběrnice

Kód pro vyčtení hodnoty z fronty, jehož průběh byl zachycen na obrázku 16, je vidět níže (viz Výpis kódu 2). V kódu je sekce pro vložení samotné funkce pro vyčtení hodnoty. Kam v druhém případě testování rychlosti bylo zařazeno volání funkce, s jejíž pomocí se nastavovala hodnota napětí analogového výstupu na jednom z pinů. Pro tyto účely byla použita funkce HAL, která by mohla být optimalizována.

```
if (TIMx->SR & TIM_SR_UIF)
{
  TIMx->SR = 0x0000;
  int tmp = get_que(&que);
  if (tmp >= 0)
  {
    // place your code here
  }
}
```

Výpis kódu 2: Kód časově podmíněného bloku pro vyčtení hodnoty

Pro účely testu generátor produkuje funkci sinus rozdělenou na třicet vzorků. Velikost napětí byla snímána osciloskopem, který určil frekvenci signálu jako $f = 16,02$ kHz, ta odpovídá přenosové rychlosti $R_b = 3,8$ Mbit/s. Z obrázku 17 je vidět, že signál je správně reprezentován a je vidět průběh funkce sinus.



17: Snímek průběhu signálu výstupu akvizičního systému na osciloskopu

4 Demonstrační úloha

Demonstrační úloha byla vytvořena pro výuku mikroprocesorové techniky (viz příloha Appendix B). Zaměřuje se na představení základních periferií. Činí tak úkolem úlohy, který vyžaduje po studentech vytvoření programu, který bude fungovat jako snímací část osciloskopu. Nastavený mikrokontroler spolu s přiloženým programem vytvoří plně funkční jednokanálový osciloskop.

Úloha stručně představuje problematiku programování pro embedded zařízení.

Pro úlohu student může použít jakýkoli mikrokontroler, který má možnost přerušení tlačítkem, LED diodu, USART a analogově-digitální převodník. Vstupem do programu je signál přivedený do převodníku. Výstupem programu jsou vzorky signálu posílané po sériovém portu.

V teoretické části úlohy je zmiňován mikrokontroler STM32F401RE. Ten byl zvolen pro demonstraci implementace programu. Součástí práce jsou implementace programu v jazycích C, C++ a Python. Jednotlivé implementace ukazují rozdílné přístupy k embedded programování. Na jedné straně je program psaný v jazyce C s použitím HAL výrobce mikrokontroleru a na druhé jsou middlewary Zerynth a MBED. Když srovnáme implementace obou middlewarů jsou skoro totožné. Nedávají možnost výběru použité periferie, ale jejich vytvoření je časově nenáročné.

Studenti díky oběma přístupům mají možnost srovnat, jak velké úsilí je nutné vynaložit pro daný, program v jazyce, který dává možnost plného ovládání.

Studenti jsou nuceni se seznámit s jednotlivými nástroji a použitými periferiemi.

5 Závěr

V úvodu byl vytyčen cíl – sestavit akviziční systém zprostředkovávající zpomalení komunikace mezi ADC a případným prvkem pro vyhodnocení signálu. Tento systém byl vytvořen a testován.

Akviziční systém, který je v této práci představován, je jen jednou z možných implementací. Ta byla podrobena testu pro zjištění správné funkce a hlavně zjištění maximální frekvence signálu, který může být vzorkován převodníkem, jemuž nastavuje vzorkovací periodu tento systém.

Pro testování byl vytvořen generátor signálu. Zjištění správné funkce bylo provedeno připojením generátoru signálu a systém nastavoval napětí na výstupu DAC, které bylo snímáno osciloskopem (viz obrázek 17). Test potvrdil správnou funkci.

Dalším předmětem testu bylo zjištění maximální vzorkovací frekvence $f_s = 1$ MHz. Podle vzorkovacího teorému je možné zachytávat signál s frekvencí menší než $f = 0,5$ MHz.

Je jasné, že není možné říct, že tento výsledek je nejlepší možný, protože nebylo zkoumáno, kolik instrukcí je nutné pro obsluhu systému a kolik se skutečně používá v asembleru. Úpravou přeloženého kódu v asembleru by mohlo přinést určité zlepšení. Bohužel je jasné, že signál s frekvencí $f_s = 10$ MHz není možné tímto způsobem zachytávat.

Systém byl vytvořen pomocí softwarových nástrojů, poskytovaných STMicroelektronics v jazyce C na mikrokontroleru 32bitovém mikrokontroleru STM32F446RE Nucleo-64. Tento mikrokontroler byl vybrán díky nízké ceně (208,78 korun[28]) a svému taktu. Pro zjištění, zda existují lepší varianty výběru mikrokontroleru z této platformy, byly představeny jednotlivé série platformy STM32.

Pokud by nebyla jedním z kritérií výběru cena, po zjištění možností mikrokontroleru v závislosti na jeho taktu v podkapitole 3.3.1, pak by byl lepší volbou jakýkoli mikrokontroler ze série STM32H7, které pracují s taktem 480 MHz. Díky stejné rychlosti, velikosti pamětí a obsahu všech potřebných periferií potřebných pro program je jedno, který ze zástupců série by byl vybrán pro rychlejší řešení programu. Pro srovnání ceny mikrokontroler STM32H743AII6 stojí 382,72 korun[29].

Výše byly popsány periferie a jejich části, které byly použity k vytvoření akvizičního signálu. A jednou z nich byl převodník (viz podkapitola 2.2). Bohužel, ani na sérii STM32H7 platformy STM32 není převodník jiného principu, který by byl výrazně rychlejší.

Akviziční systém byl vytvořen také v middlewaru MBED. Tento middleware se pro daný typ programu na mikrokontroleru ve své stávající podobě (MBED OS 5) nehodí díky chybějícímu výběru časovačů. Stejně tak ani další dva middlewary napsané v jazyce Python. Přes ujišťování vývojářů těchto nástrojů jazyk Python nebo lépe, v něm napsané nástroje neulehčují programování firmwaru mikrcokontrolerů. Nejdou totiž použít přesné aplikace, kde je důležité, která z možných stejných periferií je použita.

Systém měl být podle zadání práce vytvořen také s použitím kernelu FreeRTOS. V kapitole 3 bylo vysvětleno, proč takový program nebyl vytvořen. Cílem práce bylo vytvořit nejrychlejší možnou implementaci akvizičního systému, a tomuto cíli kernel nepřidával žádnou hodnotu navíc, dokonce svojí režií program zpomaloval.

Práce splnila cíl, program pracuje přibližně se stokrát nižším taktem, než je nejvyšší možný takt systémových hodin. Pro zlepšení výsledku, tak jak bylo navrženo výše, je nutná výměna mikrokontroleru, pak lze odhadnout, že program by mohl vzorkovat signál s frekvencí přibližně $f_s = 2$ MHz.

Pro další vývoj v případě připojení kamery nebo jiné rychlé periferie by bylo nutné nalézt

[28] Cena jednoho kusu mikrokontroleru STM32F446RET7, viz [25]

[29] Cena jednoho kusu tohoto mikrokontroleru, viz [26]

mikrokontroler, který komunikuje dostatečně rychle po sběrnici, nebo pracuje s výrazně vyšší frekvencí.

V demonstrační úloze je představen rozdílný přístup programování v C a middlewarech MBED a Zerynth. Cílem úlohy je vytvořit jednokanálový oscilátor a studenti mají možnost nahlédnout do řešení v jednotlivých implementacích.

Reference

[1] STMicroelectronics. Stm32 32-bit mcu family. http://www.emcu.it/STM32/port1.png. [13.05.2019].

[2] STMicroelectronics. Stm32f2 series. https://www.st.com/en/microcontrollers-microprocessors/stm32f2-series.html. [17.05.2019].

[3] STMicroelectronics. Stm32f4 series. https://www.st.com/en/microcontrollers-microprocessors/stm32f4-series.html. [16.05.2019].

[4] STMicroelectronics. Stm32f7 series. https://www.st.com/en/microcontrollers-microprocessors/stm32f7-series.html. [16.05.2019].

[5] STMicroelectronics. Stm32h7 series. https://www.st.com/en/microcontrollers-microprocessors/stm32h7-series.html. [16.05.2019].

[6] STMicroelectronics. Nucleo-f446re. https://os.mbed.com/platforms/ST-Nucleo-F446RE/. [13.05.2019].

[7] Vladimír Haasz, Jan Holub, Michal Janošek, Petr Kašpar, and Vojtěch Petrucha. *Elektrická měření: přístroje a metody*. Česká technika - nakladatelství ČVUT, Praha, 3. přepracované vydání edition, 2018.

[8] STMicroelectronics. *AN4566*, 2 edition, 8 2015. DocID026799.

[9] Markéta Pecoldová. *Firmware pro základnovou stanici domácího asistivního systému*. České vysoké učení technické v Praze, Praha, 2014.

[10] STMicroelectronics. *STM32F446xx advanced Arm®-based 32-bit MCUs*, 4 edition, 2 2018. RM0390.

[11] JAI. *CM-040GE*, 2 2013. 31000000.

[12] STMicroelectronics. Who we are. https://www.st.com/content/st_com/en/about/st_company_information/who-we-are.html#. [30.04.2019].

[13] STMicroelectronics. Stm32 high performance mcus. https://www.st.com/en/microcontrollers-microprocessors/stm32-high-performance-mcus.html. [06.05.2019].

[14] STMicroelectronics. *STM32F446xC/E*, 6 edition, 9 2016. DocID027107.

[15] STMicroelectronics. *AN1636*.

[16] STMicroelectronics. *AN2834*, 3 edition, 2 2017. DocID15067.

[17] Václav Král. *Justovací modul kmitočtového filtru*. České vysoké učení technické v Praze, Praha, 2018.

[18] GeeksforGeeks. Program for round robin scheduling. https://www.geeksforgeeks.org/program-round-robin-scheduling-set-1/. [22.05.2019].

[19] FreeRTOS. What is an rtos/freertos. https://freertos.org/about-RTOS.html. [01.05.2019].

[20] Open Source Initiative. The mit license. https://opensource.org/licenses/MIT. [15.05.2019].

[21] FreeRTOS. Why choose freertos? https://freertos.org/index.html. [15.05.2019].

[22] ARM MBED. Developing with mbed. https://www.mbed.com/en/. [17.05.2019].

[23] Zerynth. What is zerynth. https://www.zerynth.com/get-started/#what-is-zerynth. [07.05.2019].

[24] MicroPython. Micropython. http://micropython.org/. [07.05.2019].

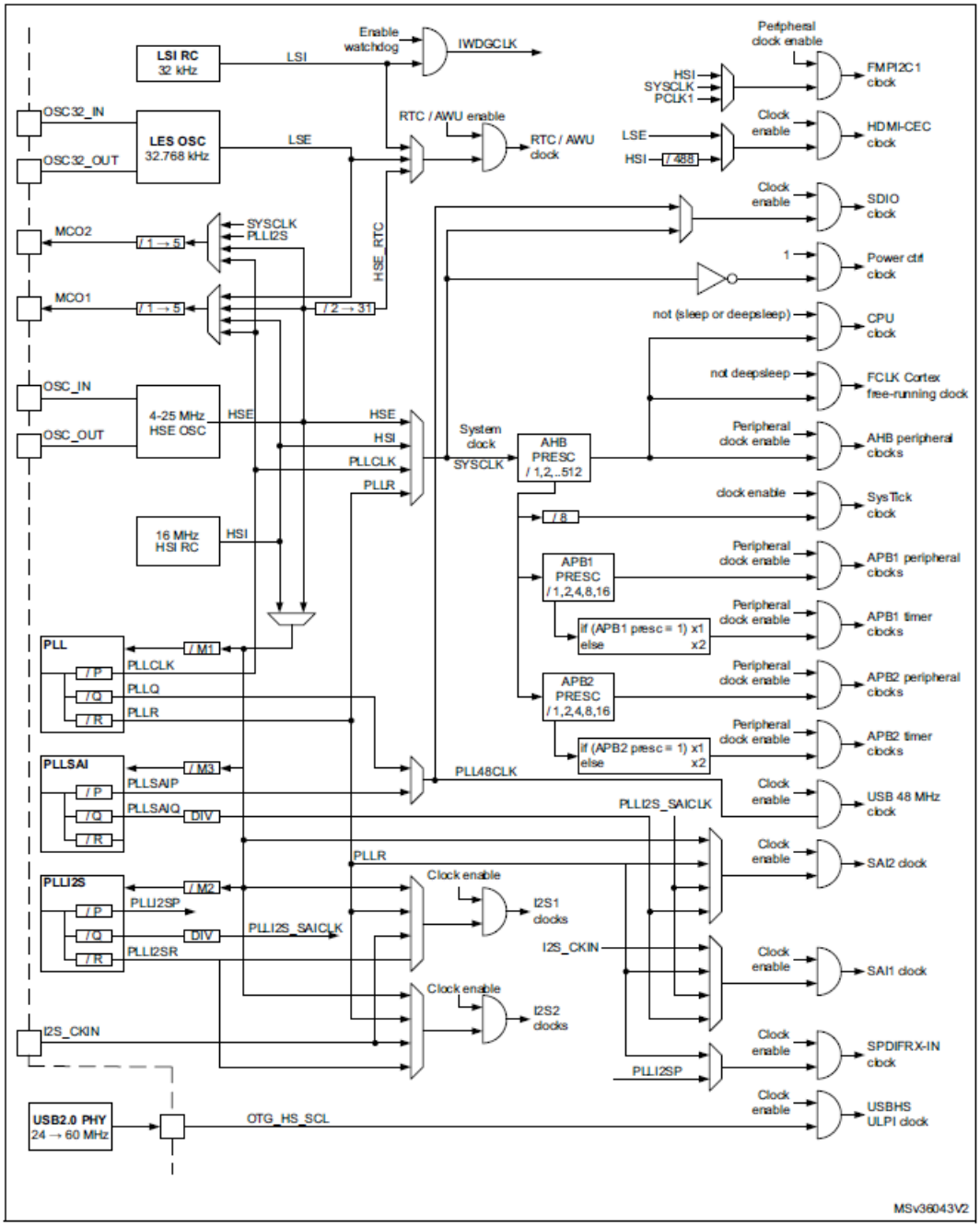
[25] Mouser Elektronics. Stm32f446ret7. https://cz.mouser.com/ProductDetail/STMicroelectronics/STM32F446RET7?qs=sGAEpiMZZMuoKKEcg8mMKHeGRAWWvG589Dl0ZOG8CroKWtWm5wy1wg%3D%3D. [18.05.2019].

[26] Mouser Elektronics. Stm32h743aii6. https://cz.mouser.com/ProductDetail/STMicroelectronics/STM32H743AII6?qs=%2Fha2pyFadujiWVHRlW6sBcoy8%2FfLIOlpIZXx1LaGBZKEQlAq6hSzEg==. [18.05.2019].

[27] STMicroelectronics. *STM32F401xD STM32F401xE*, 3 edition, 1 2015. DocID025644.

A Příloha 1

A.1 Schéma pro nastavení systémových hodin



Appendix 18: Schéma zapojení zdrojů hodinového signálu zařízení série STM32F4 [10]

B Příloha 2

B.1 Demonstrační úloha

B.1.1 Úvod

V následující úloze se seznámíme se zásadami programování mikroprocesorové techniky. Demonstrujeme použití některých periferií mikrokontroleru. Předně se budeme věnovat digitálním vstupům kontroleru. Využijeme časovač, komunikační rozhraní USART a seznámíme se s přerušením normálního běhu programu.

Cílem úlohy je naprogramovat zařízení schopné pracovat jako osciloskop po připojení do počítače. Program pro vykreslení výsledků na pc je součástí podkladů úlohy.

B.1.2 Úkol

- Pomocí mikrokontroleru a připojeného programu sestavte jednokanálový osciloskop.
- Vyberte si jeden z jazyků C nebo Python. Pokud si vyberete, práci v jednom z middlewarů Zerynth, MicroPython, MBED, pak se s ním blíže seznamte.
- Implementujte obslužný program, který po stisku tlačítka na microkontroleru začne vzorkovat signál na analogovém vstupu a bude je rovnou posílat přes komunikační rozhraní USART do počítače.
- Pomocí LED mikrokontroleru indikujte vzorkování signálu.
- Po dalším stisku tlačítka ukončete program.
- Snažte se vytvořit osciloskop s největší možnou vzorkovací frekvencí.

B.1.3 Teoretický úvod

Díky velké diverzitě návrhů a provedení mikroprocesorů nejsou programy psané pro specifický mikroprocesor přenositelné na jiný. Dokonce nemusí být možnost změnit mikrokontroler za jiný se stejným čipem, díky rozdílnému připojení periferií. Proto je nutné si před zahájením řešení problému nastudovat přesné potřeby budoucího programu (paměťová náročnost, rychlost rozhraní atd.) a podle nich zvolit nejvhodnější dostupný mikrokontroler. Mikrokontrolerů je dnes velké množství, a proto je většinou rozhodujícím faktorem cena.

Vedle výběru mikrokontroleru je stejně důležitý i výběr programovacího jazyka. Dlouhou dobu byl jediným jazykem pro programování embedded zařízení procedurální jazyk C, popřípadě embedded C/C++. Dnes se ale začal hojně používat i Python ve verzích middleware Micropython [24] nebo například Zerynth [23].

Pro demonstrační úlohu budeme používat vývojovou desku STM32F401RE Nucleo-64 (dále referovaný jen jako "nucleo"), což je mikrokontroler s mikroprocesorem ARM Cortex-M4 [27]. Pro programování zvolíme nejprve middleware Zerynth (Python/C), MBED (C/C++) a na závěr si demonstrujeme, na implementaci v C, že má programátor při použití tohoto nástroje velkou moc a může nastavovat svobodně věci, které například vývojář middleware neimplementoval pro danou vývojovou desku, ale i jakou náročnost daná svoboda má.

Každý pin nuclea je schopen zpracovávat logický signál, díky GPIO[30]. Následně jsou jednotlivé periferie vyvedeny na piny desky. Zapojení jednotlivých periferií je možné nalézt v dokumentaci [27]. Z té zjistíme, že není možné používat zároveň některé periferie mikrokontroleru.

[30] General Purpose Input/Output - digitální pin pro vstup a výstup

B.1.4 Demonstrace

V následujících ukázkách kódu je možné shlédnout řešení úkolu. Příklady jsou implementovány v jazycích C, C++ a Python. Už pouhým náhledem je naprosto jasné, který z programů je výhodnější z hlediska časové náročnosti vytvoření. Pokud bychom se ale blíže podívali na příklad s použitím middlewaru MBED a při náhledu do standardních knihoven MBED zjistíme, že vývojář nám neposkytl možnost vybírat některé periferie. Například programu psaným jazykem C jsme použili časovač TIM1, který je podle specifikace rychlejší (spolu s TIM8) než ostatní čítače mikrokontroleru. Takovou možno za použití MBED nebo Zerynth nemáme.

Příklady nesplňují poslední bod úkolu, rychlost. V implementaci v C je možné si prohlédnout a zkusit nalézt parametry, které je nutné změnit pro zvýšení vzorkovacího kmitočtu. Nastavení těchto parametrů je vhodné provést v programu CubeMX.

Součástí přiložených programů je soubor README, který obsahuje popis a návod jednotlivých programů potřebné k jejich spuštění.