Soustředění mladých matematiků a fyziků

Plasnice 2024

Programování s registry

Autoři: Richard Dobíšek, Vojtěch Procházka

Vedoucí projektu: Mgr. Jan Sixta

Prohlašujeme, že jsme práci s názvem „Programování s registry“ vypracovali samostatně a čerpali pouze z uvedené literatury a dalších zdrojů.

Anotace

Cílem projektu bylo seznámit se s programováním mikrokontroléru ARM STMF411xC/E v C a za minimálního užití externích knihoven (pomocí přímého přístupu do registrů) zpracovávat a filtrovat audio signál.

Poděkování

Chtěli bychom poděkovat našemu vedoucímu Mgr. Janu Sixtovi za cenné rady, a pomoc při jak vymýšlení, psaní kódu, debugování, poskytnutí desky a za všechno, čím nám v průběhu projektu pomáhal. Dále děkujeme všem, kteří se podílí na realizaci a organizaci Soustředění mladých fyziků a matematiků.

1 Úvod

V rámci projektu programování s registry jsme pracovali s vývojovými kity STM32, konkrétně STM32F401RC a STM32 NUCLEO-F411RE. Začínali jsme s F401 bez žádných připojených periferií, na kterém jsme si vyzkoušeli základní funkce jako ovládání LED pomocí tlačítka a komunikaci mikrokontroleru s počítačem pomocí periferie USART. Na jiném typu (F411RE) s rozšířeními pro příjímání a odesílání analogového audio signálu jsme poté zpracovávali tento signál v reálném čase. Implementovali jsme efekt echo a highshelf filtr.

2 Vývojové kity STM32 NUCLEO-F411RE

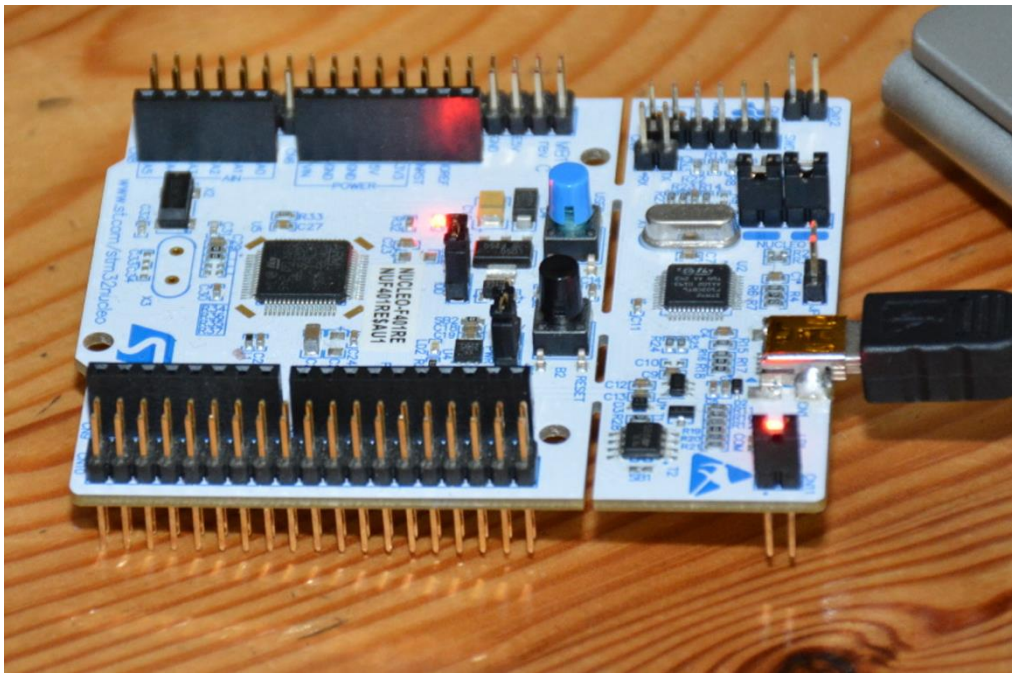
Mikrokontroler použit ve finálním projektu:

Založeno na vysokovýkonnostním Arm Cortex –M4 32-bit RISC procesoru s frekvencí až 100 MHz.

512 kB Flash a 128 kB SRAM.

Mnoho pinů včetně GPI/O pinů a USART s přenosovou rychlostí až 12.5 Mbit/s.

Velikou výhodou je právě vysoký výpočetní výkon, velká paměť a vysoká modifikovatelnost díky možnosti přímo psát do registrů periferií.



Obrázek 1: STM32F401RC mikrokontroler

3 Postup

3.1 Příprava

Pro práci na projektu s STM32 je potřeba si nainstalovat STM32 Cube Programmer a STM32 Cube IDE, kde se bude psát kód v C. Pro zobrazování komunikace mezi počítačem a mikrokontrolerem jsme používali aplikaci CoolTerm ačkoli se dají použít i jiné alternativy.

3.2 GPIO

Začali jsme jednoduchým prvním pokusem rozsvítit LED diodu, která je součástí desky, na které se procesor nachází a poté jí spojit s tlačítkem rovněž na této desce. Pro tento pokus a pár dalších kroků jsme ještě založili projekt pomocí knihoven a nechali si některé základní

věci, hlavně konfiguraci systému vygenerovat knihovnami, abychom se mohli věnovat jednodušším věcem před konfigurací samotného systému. Pro toto bylo potřeba naučit se několik základních operací. Hlavní z nich byl zápis do samotných registrů, pomocí nichž se dá využívat všech funkcí, které tato deska obsahuje. Hlavní funkcí, kterou jsme pro tento jednoduchý úkol potřebovali bylo GPIO (general purpose input/output). GPIO má několik základních registrů, které jsme využívali. Nejdůležitější je MODER, kam se zapisuje funkce, kterou má daný pin zastávat (vstup/výstup) a poté IDR, kde se zobrazují vstupy a ODR/BSRR, kterými se vypisují výstupy. V této fázi náš kód vypadal jen tak, že nakonfiguroval správné hodnoty do registrů a poté v nekonečné smyčce, pokud přečetl vstup z tlačítka, tak zapsal 1 na výstup k LED diodě a pokud výstup z tlačítka nepřečetl, tak zapsal 0.

3.3 USART

Dále jsme se snažili zprovoznit USART pro komunikaci s počítačem. USART je po konfiguraci schopný posílat do jiného zařízení data z mikrokontroleru, nebo je naopak přijímat. Nejjednodušší způsob, jak se dá ověřit, že toto funguje je posílat data z mikrokontroleru do počítače, který je umí zobrazit v nějaké konzoli. Pro tento účel jsme používali terminál CoolTerm, který nám poslaná data zobrazoval a pro naše účely fungoval skvěle, ale jakákoliv aplikace pro komunikaci se sériovými porty by toto měla zvládnout. Pro nastavení této funkcionality bylo potřeba naprogramovat několik registrů, mimo jiné také BRR (Baud rate register), kde se nastavuje rychlost komunikace mikrokontroleru s počítačem. Na první pohled se může zdát, že se sem pouze zapíše rychlost, kterou by měla komunikace probíhat, ale bohužel to tak není jednoduché. Musí se sem napsat koeficient, kterým se dělí kmitočet hodin v této periferii, aby se dosáhlo cílené rychlosti komunikace (baudrate). Náš kód, který této funkcionality využíval byl kromě původního nastavení jedna funkce pro výpis textu do konzole, která ho postupně po znaku ukládala do datového registru USARTU a počkala, dokud se úspěšně neodeslal.

3.4 DMA

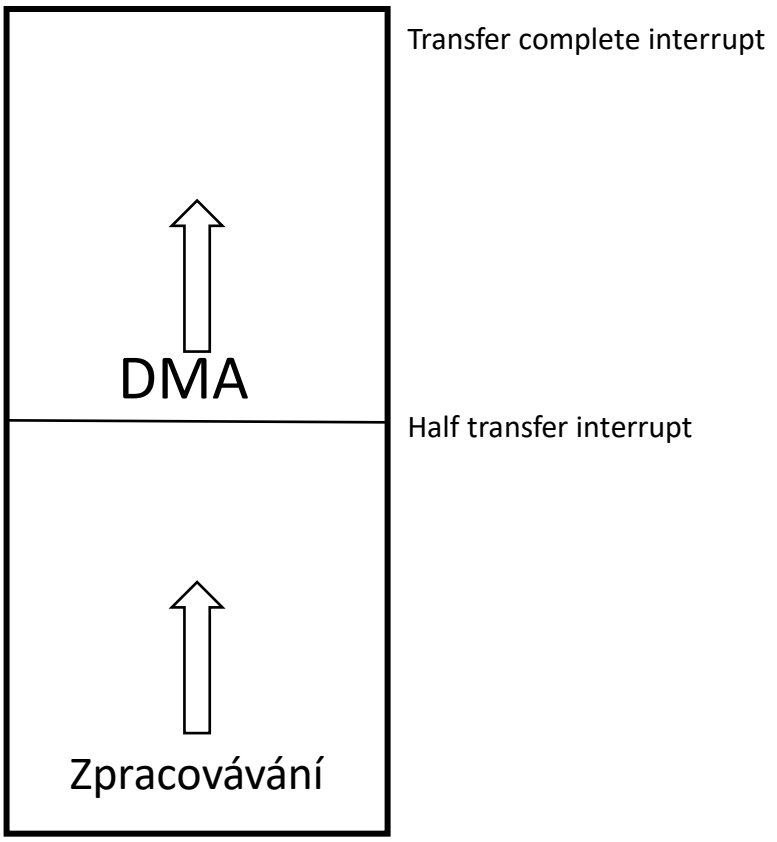
Náš poslední menší experiment před konfigurací celého systému bylo využití DMA. DMA je periferie procesoru, která slouží k přesouvání dat mezi periferiemi a pamětí, nebo v rámci paměti, čímž usnadňuje práci procesoru. Chtěli jsme jí využít k zefektivnění výpisu do konzole v počítači, kterou jsme zprovoznili pomocí USARTu. Při tomto výpisu se procesor musí zabývat postupně každým znakem ve zprávě, kterou chceme do konzole poslat a počkat, dokud se úspěšně nezpracuje, což je neefektivní využití hlavního procesoru, když přesně k tomuto účelu existuje DMA, které stačí zadat, aby přesunula blok paměti z místa, kde se nachází zpráva, kterou chci poslat do počítače na místo, kde se nachází vstupní registr USARTu. Pro první jednodušší seznámení jsme začali s jednodušším úkolem překopírovat pole z paměti do jiného. Poté jsme zrychlovali výpis do konzole. Mimojiné je třeba nastavit správný kanál DMA podle tabulkz v manuálu a povolit využití DMA i na druhé straně. Náš kód v tuto chvíli vypadal velmi podobně, jako funkce pro tisk pomocí USART, ale místo toho, aby text po znaku posílala přes USART, tak nastavila DMA, aby přesunula data z textu do datového registru USART a poté zapnutí DMA. Ještě bylo potřeba po opakovaném použití vynulovat označení, že přenos už proběhl

3.5 Nastavení Systému

Dále už přišla nová výzva v podobě naprogramování systémové konfigurace, kterou v současné době dělaly knihovny generované nástrojem STM32 Cube Mx. Toto zahrnovalo několik základních kroků. Tyto kroky byly povolení ko-processoru pro výpočty v plovoucí řádové čárce (floating point IEEE754), nastavení systémových hodin a jejich správné přepnutí a poté nastavení děliček kmitočtu jdoucího do periferií. Nakonec ještě vytvoření přerušení pro počítání času (SysTick, systémový časovač), který každou milisekundu (84 000 cyklů hodin) zvýší číslo znázorňující počet milisekund od zapnutí programu. Toto můžeme využít pro vlastní funkci zpoždění (Delay), která počká, dokud neuběhne nějaký čas a do té doby zaměstná procesor, nebo pro omezení částí kódu tak, aby se spouštěly „jen“ každých 100 ms, protože častěji to není potřeba. Hlavní část této konfigurace bylo zjišťování, co všechno je potřeba zapnout a jak nastavit některé hodnoty k hodinám, aby běželi podle našich přestav (a omezení systému). Jako nejjednodušší řešení se ukázalo použít hodnoty, kterými se před tím inicializovali knihovny, protože ty už je mají správně spočítané.

3.6 I2S

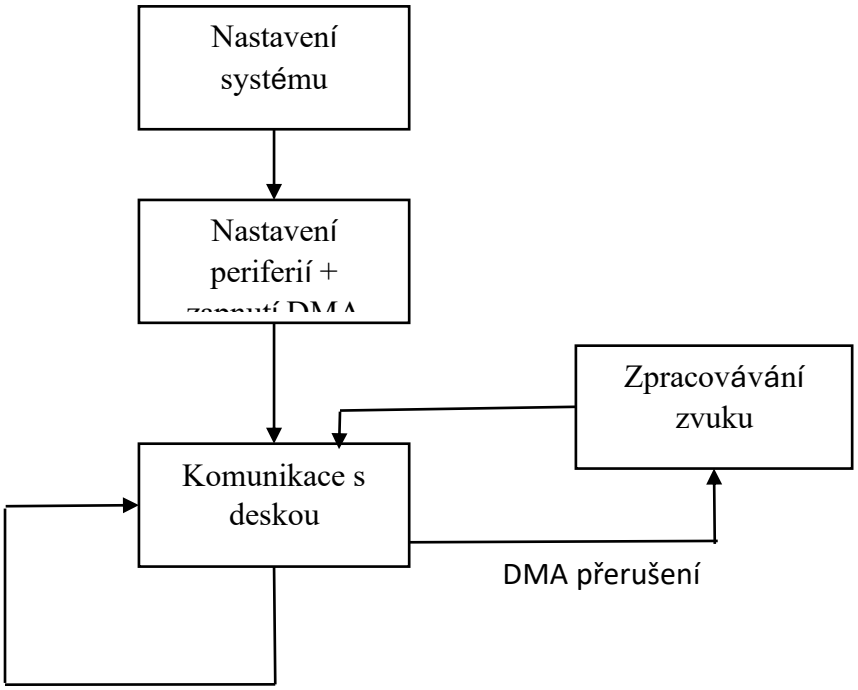
Jelikož jsme se po konfiguraci procesoru rozhodli pustit se do zpracování zvukového signálu, tak jsme začali používat desku se zvukovým kodekem (integrovaný převodní A/D a D/A) a potřebovali jsme s ní nějakým způsobem komunikovat. K tomuto jsme používali periferii SPI/I2S, kterou F411RE disponuje. Jako každou jinou periferii této desky je i I2S potřeba nastavit zápisem do příslušných registrů. Využívá se 1 kanál na odesílání dat a jeden kanál na jejich příjem. Tyto data ukládáme do 2 zásobníků (buffer) o velikosti 1024 vzorků z nichž jeden slouží k odesílání a druhý k přijmu. Pro práci s těmito zásobníky a rychlý přenos k I2S používáme znovu DMA. Používáme 2 kruhové instance DMA, které postupně přesouvají data ze zásobníku do I2S/z I2S do zásobníku a sami tuto operaci do nekonečna opakují. V průběhu své činnosti průběžně v polovině a na konci bufferu generují přerušení, které umožňuje zpracování dat, než je DMA přepíše.



Obrázek 2: Náčrtek funkce zásobníků

Ve chvíli, kdy procesor dostane od DMA přerušení, že byla určitá část dat přenesená, tak jí může začít zpracovávat. Jelikož ví, že DMA se právě zabírá přesouváním dat v jedné polovině zásobníku (podle toho, jaké přišlo přerušení) a snaží se zpracovat data tak, aby se o ně s DMA nepřetahovali, tak začne zpracovávat data od druhé poloviny zásobníku, než od které dostal přerušení. Tento čas má na naplnění odesílajícího zásobníku vzorky dat, které poputují na výstup. Tedy může za tu dobu buď nějaké generovat, nebo nějak zpracovat data, která mu přišla na vstupu. Důležité na tomto zpracování je, že se musí stihnout velmi rychle, protože ve chvíli, kdy přestane stíhat rychlost, kterou DMA přenáší informace, tak zaprvé audio signál bude nekompletní a za druhé nebude stíhat nic jiného, což je potřeba například pro změnu různých hodnot pomocí potenciometru na desce, nebo pro reagování na nastavení spínačů které se na desce rovněž nacházejí.

4 Jak Program funguje?



Obrázek 3: Flowchart popisující základní průběh programu

Program začíná nastavením systému, kde se seřídí systémové hodiny. Poté se postupně nastaví všechny periferie. Hlavně GPIO, které je potřeba pro veškerou komunikaci procesoru s venkem. Poté také I2S, které komunikuje se zvukovým zařízením a DMA, které se stará o rychlý přenos těchto dat do paměti, kde se poté dají zpracovat. Dále se zapíná ještě ADC, které se stará o převod analogových dat z potenciometru na digitální, která se dají použít dále v programu pro. upravování některých konstant (zejména koeficientu alfa u ozvěny, nebo zisku u bikvadratického filtru). ADC zajišťuje fungování dvou modrých potenciometrů na desce.

Následuje nekonečná smyčka, ve které počítač každých 100 ms komunikuje s deskou s ovládacími prvky a výstupovými zařízeními. Zejména tedy snímá natočení potenciometrů, které se vyhodnotí a převedou buď na signál 0-1 v případě potenciometru ovládajícího koeficient alfa, nebo na signál -15 ... 15 dB v případě potenciometru ovládajícím zisk filtru. Také zde čte spínače, kde spínač nejvíce vlevo zapíná/vypíná přeměnu zvuku a druhá zleva určuje, zda zvuk bude podléhat efektu ozvěny (echo), nebo bikvadratického filtru. Tyto spínače jsou také spojené s 2 LED diodami v pravém spodním rohu desky, kde ta nejvíce vpravo svítí v případě, že je změna zvuku zapnutá a druhá zprava svítí, pokud je zapnutý bikvadratický filtr a pokud je zapnutá ozvěna, tak nesvítí. Nakonec je zde ještě 8 LED diod, které sloupcem znázorňují maximální amplitudu, kterou od minulého zobrazení zpracoval procesor.

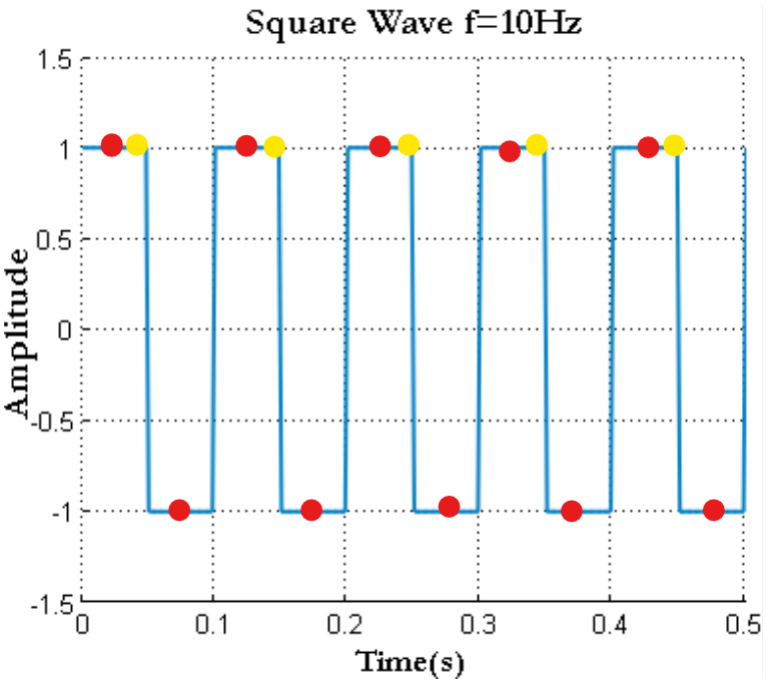
Nakonec je přerušení, který řídí zpracování zvuku. Toto přerušení se spustí vždy, když DMA zahlásí, že je na konci, nebo v polovině zásobníku. V tomto přerušení si počítač nejdříve

zjistí, jestli je na konci zásobníku, nebo v jeho polovině a podle toho začne zpracovávat příslušnou polovinu dat. Jelikož je zvukový výstup rozdělený do pravého a levého kanálu, každý z nichž jde do jiného sluchátka, tak zpracovává vždy dva signály najednou. Vždy se podívá, jestli datový záznam nemá vyšší amplitudu, než nejvyšší naměřenou a případně jí takto uloží. Poté si zjistí, jestli má zpracovat data pomocí bikvadratického filtru, nebo pomocí ozvěny, toto udělá a data zapíše do zásobníku určeného k odeslání dat. Bohužel neumíme oba efekty zároveň, protože náš kód není dost optimalizovaný, takže počítač nestíhá vyplnit zásobník vzorky než se k nim dostane DMA.

5 Zpracování zvuku

Po nastavení systému jsme byli schopni přijímat a vysílat signál z mikrokontroleru nezávisle pro levou a pravou stranu s frekvencí 48 kHz. Jako první jsme vygenerovali sinovou vlnu a nastavili modré potenciometry tak aby bylo jejich otáčením možno měnit frekvenci a amplitudu, kde amplituda mění hlasitost a frekvence výšku tónu.

Poprvé jsme použili čtení signálu jen k poslouchání zvuku z počítače a pak pomocí mikrokontroleru posílat nijak neupravené audio do levého sluchátka. Signál na vstupu se dá považovat za spojitou funkci, ale náš počítač nevidí vstup jako funkci, nýbrž si uloží amplitudu funkce Fs krát za sekundu, v našem případě 48 000x. Aby vzorkování fungovalo správně, tak Fs musí být alespoň dvakrát větší než nejvyšší frekvence signálu na vstupu. Díky tomu dokáže počítač zaznamenat situaci s vysokou amplitudou stejně tak nízkou amplitudou a díky tomu je reprezentace funkce pomocí diskrétních bodů poměrně přesné.



Obrázek 4: Obdélníkový průběh vlny

Na obrázku je signál s frekvencí 10 Hz. Pro Fs = 20 Hz (červená) počítač vidí jak body s amplitudou 1, tak body s amplitudou –1, takže spojením červených bodů přibližně vznikne vstupní funkce. Žluté body vznikly s Fs = 10 Hz, takže počítač zachytil pouze maxima a tím pádem by to vypadalo po načtení vstupu, že signál je konstantní, což není pravda. Naše Fs je 48 kHz, takže zvládneme zachytit signál s frekvencí až 24 kHz. Jelikož pro lidské ucho je práh slyšitelnosti maximálně 20 kHz, budeme schopni zachytit všechny relevantní složky audia.

5.1 Ozvěna (Echo)

Poté jsme se rozhodli naprogramovat echo, které bude do sluchátek přehrávat kombinaci aktuálního zvuku a zvuku který hrál před 500 ms, čímž se vytvoří ozvěna. Data se musí ukládat, což je možné díky velké RAM kterou F411 disponuje. Bylo potřeba uložit alespoň 24

000 vzorků výstupního signálu do pole Y, protože počítač čte 48 000 krát za sekundu a je potřeba si pamatovat data 0,5 s zpět. X[n] je aktuální vstup signálu z počítače. Nový výstup se spočítá jako vážený průměr současného vstupu a výstupu zpožděného o 24 000 vzorků, kde proměnný koeficient alfa rozděluje váhy.

$$y[n] = \alpha \cdot x[n] + (1 - \alpha) \cdot y[n - 24000]$$

Čím je alfa nižší tím je ozvěna silnější. Kdyby byla suma koeficientů před X a Y[n-24000] větší než jedna, tak by se amplituda výstupu zvyšovala do nekonečna, takže by došlo k přehlcení výstupu signálem o příliš vysoké amplitudě.

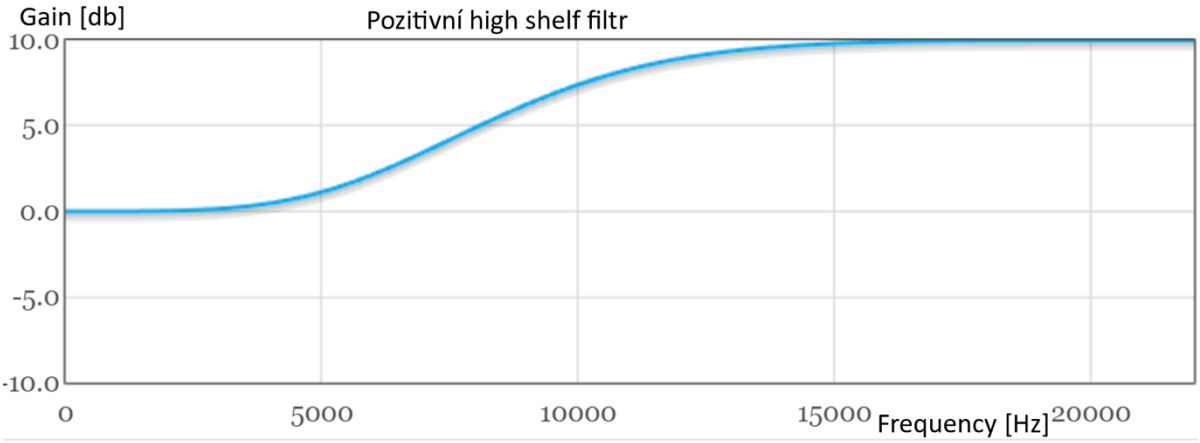
5.2 Filtry

Dalším cílem bylo filtrovat signál. Z teoretického hlediska jde o to, že lze každý signál rozložit na sumu vln (sinů) o různých frekvencích a filtry upravují signál tak, že mění amplitudy jednotlivých vln a tím mění celý signál. Například obdélníkový průběh vlny (podobná jako v kapitole rozšířené funkce) může být zapsána jako:

$$x(t) \approx \sin(t) + \frac{1}{3}\sin(3t) + \frac{1}{5}\sin(5t) + \dots$$

Tuto funkci můžeme reprezentovat jako amplitudu v závislosti na čase, ale také po rozložení na siny můžeme funkci zakreslit jako amplitudu jednotlivých vln v závislosti na jejich frekvenci. Filtry můžou s těmito frekvencemi pracovat.

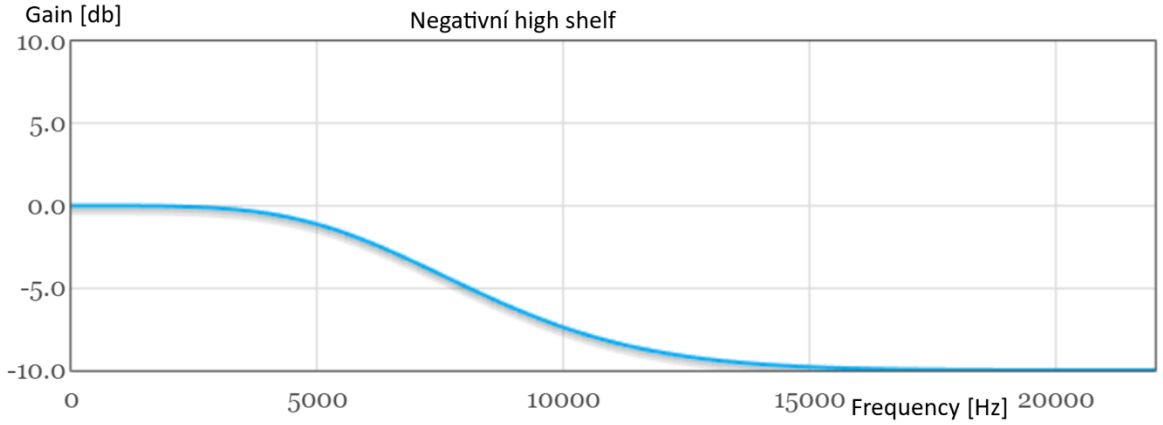
Nejčastěji užívané filtry jsou dolní propusť (lowpass), horní propusť(highpass), pásmová propusť (bandpass), zářezový filtr (notch), špičkový filtr (peak), low shelf, high shelf. Na obrázku je přenosová charakteristika funkce pozitivní high shelf.



Obrázek 5: Pozitivní high shelf bikvadratický filtr

Na ose x jsou frekvence složek signálu a na y-ose je velikost změny jejich amplitudy. Gain $g = 20 \cdot \log_{10}\left(\frac{A_1}{A_0}\right)$, kde A_0 je původní amplituda a A_1 je nová amplituda, takže $A_1 = A_0 \cdot 10^{\frac{g}{20}}$. High shelf upraví amplitudy podle své přenosové funkce, takže nízké frekvence pod 5 000 Hz zůstanou prakticky nezměněny. Pozitivní high shelf jako na obrázku násobí amplitudy vysokých frekvencí, takže budou až $\sqrt{10}$ krát větší. Celkově budou tedy vyšší frekvence budou více dominantní (tón bude vyšší) a zvuk vyšších kmitočtů hlasitější. Negativní high shelf funguje přesně opačně, takže potlačuje vysoké frekvence a nižší

frekvence budou lépe slyšet.



Obrázek 6: Negativní high shelf bikvadratický filtr

Toto je ale čistě teoretická záležitost, protože počítač nerozkládá signál na frekvence, ale jen provádí aritmetické operace mezi vstupy a výstupy. High shelf který jsme používali má bikvadratický přenos, což znamená, že používá pro výpočet výstupu aktuální, předchozí a před-předchozí vstup a také předchozí a před-předchozí výstup. Přenos lze zapsat jako podíl dvou kvadratických polynomů.

$$H(z) = \frac{a_2 \cdot z^{-2} + a_1 \cdot z^{-1} + a_0}{1 + b_1 \cdot z^{-1} + b_2 \cdot z^{-2}}$$

$$y[n] = a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] - b_1 y[n-1] - b_2 y[n-2]$$

Koeficienty ovlivní povahu filtru tedy bude-li dolní propusť, high shelf etc. Tyto vzorečky jsme použili pro před počítání koeficientů pozitivního highshelf, kde je peak gain > 1 takže dochází k posilování vyšších frekvencí.

$$K = \tan\left(\frac{\pi F_c}{F_s}\right)$$

$$V = 10^{\left(\frac{\text{PeakGain}}{20}\right)}$$

$$\text{norm} = \frac{1}{1 + \sqrt{2} \cdot K + K^2}$$

$$a_0 = \left(V + \sqrt{2V} \cdot K + K^2\right) \cdot \text{norm}$$

$$a_1 = 2 \cdot \left(K^2 - V\right) \cdot \text{norm}$$

$$a_2 = \left(V - \sqrt{2V} \cdot K + K^2\right) \cdot \text{norm}$$

$$b_1 = 2 \cdot \left(K^2 - 1\right) \cdot \text{norm}$$

$$b_2 = \left(1 - \sqrt{2} \cdot K + K^2\right) \cdot \text{norm}$$

6 Závěr

Závěrem považujeme náš projekt za úspěšný, protože se podařilo seznámit se s programováním mikrokontroléru ARM STM32 a povedlo se nám napsat vlastní aplikaci bez použití knihoven. Dále se nám povedlo registrů využít k rozsvícení LED diody. Ke komunikaci s počítačem prostřednictvím UART. Povedlo se nám přenášet data s DMA a nakonec i komunikovat se zvukovým signálem, a přitom ho v reálném čase zpracovávat. Toto všechno jsme zvládli bez využití knihoven.

Zdroje

https://www.earlevel.com/main/2011/01/02/biquad-formulas/

https://www.st.com/en/microcontrollers-microprocessors/stm32f411.html

https://www.st.com/resource/en/reference_manual/rm0383-stm32f411xce-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

https://www.st.com/resource/en/datasheet/stm32f205rb.pdf