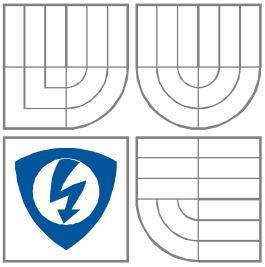
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

VYUŽITÍ PROCESORŮ ARM PRO ZPRACOVÁNÍ SIGNÁLŮ

USING ARM PROCESSORS FOR DIGITAL SIGNAL PROCESSING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE Miloš Vonička
AUTHOR

VEDOUCÍ PRÁCE Ing. Roman Mego
SUPERVISOR

BRNO, 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Elektronika a sdělovací technika**

Ústav radioelektroniky

Student: Miloš Vonička ***ID:*** 168284

Ročník: 3 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Využití procesorů ARM pro zpracování signálů

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s procesory z rodiny ARM, rozdělení architektur v jednotlivých řadách a jejich inštrukční sadou. Na vhodně vybraném ARM procesoru realizujte v jazyce C některé nejčastější algoritmy číslicového zpracování signálů (DSP). Analyzujte výpočetní výkon jednotlivých algoritmů při použití floating-point instrukcí a bez nich. Také analyzujte použití optimalizací kompilátoru.

Vybrané DSP algoritmy implementujte v jazyce symbolických adres (JSA). Implementace v jazyce C a JSA porovnejte z hlediska rychlosti výpočtu, využití pamětí a pracovních registrů. Diskutujte výhody a nevýhody obou programovacích přístupů jednak z hlediska technického (přínos pro aplikaci), tak i z hlediska ekonomického (prostředky vynaložené na vývoj).

DOPORUČENÁ LITERATURA:

[1] JAN, Jiří. Číslicová filtrace, analýza a restaurace signálů. Vyd. 2. Brno: VUTIUM, 2002, 427 s. ISBN 80-214-1558-4.

[2] ARM HOLDINGS. Cortex-M4 Devices: Generic User Guide [online]. 2010 [cit. 11. 5. 2014]. Dostupné z: http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf.

Termín zadání: 6. 2. 2017 ***Termín odevzdání:*** 30.5.2017

Vedoucí práce: Ing. Roman Mego

Konzultant:

prof. Ing. Tomáš Kratochvíl, Ph.D.
předseda oborové rady

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ Fakulta elektrotechniky a komunikačních technologií Ústav radioelektroniky

UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

ABSTRAKT

Cílem bakalářské práce je seznámit čtenáře s problematikou ARM procesorů na vybraném vzorku od společnosti STMicroelectronics. Práce má dále čtenáři připomenout základní principy analýzy signálů a snaží se o jejich demonstraci na zvolené platformě. Zájemcům o programování vlastního zařízení v jazyce C a JSA nabídne stručný návod na nastavení desky, vývojového studia a kompilátoru. Práce srovnává oba programátorské přístupy z hlediska technického a ekonomického. Důležitým bodem práce je porovnání rychlosti zpracování signálů při různých nastaveních kompilátoru a při použití operací v pohyblivé řádové čárce.

KLÍČOVÁ SLOVA

ARM, STM32F4DISCOVERY, DSP, optimalizace, kompilátor, Fourierova transformace

ABSTRACT

Aim of this bachelor's thesis is to enlighten its reader on issue of ARM processors on device manufactured by STMicroelectronics. Thesis should also remind the reader the basis of signal processing (DSP) and also tries to demonstrate it on chosen platform. This document also provides a brief tutorial, how to program owned device in C language, how to set up the IDE and compiler. Main goal of this thesis is to compare speeds of digital signal processing in C and ASM, compare speeds of DSP with or without usage of FPU and under different compiler settings.

KEYWORDS

ARM, STM32F4DISCOVERY, DSP, optimization, compiler, Fourier transform

VONIČKA, M. Využití procesorů ARM pro zpracování signálů. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, 2017. 37 s. Bakalářská práce. Vedoucí práce: Ing. Roman Mego

PROHLÁŠENÍ

Prohlašuji, že svoji bakalářskou práci na téma Využití procesorů ARM pro zpracování signálů jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Tato práce má za cíl uvést čtenáře do dnes již obsáhlého světa mikroprocesorů a mikrořadičů ARM, připomenout základní principy analýzy signálů. Signál bude analyzován pomocí ARM procesoru vybaveného podporou DSP operací a FPU jednotkou, programovaného v jazyce C a JSA. Výstupem práce je porovnání rychlosti zpracování digitálního signálu při použití různých kompilátorů a jejich nastavení a porovnání programovacích přístupů.

Bakalářská práce je členěna do šesti základních částí. Kapitola 1 představuje seznámení s procesory rodiny ARM, dělení na jednotlivé poddruhy a rozdíly mezi nimi. Teorií analýzy digitálních signálů, principy Fourierovy transformace a možnostmi jejího zefektivnění se zabývá druhá kapitola. Třetí kapitola nabízí návod na nastavení vývojových prostředí a přípravu projektů pro vývoj. Čtvrtá kapitola se věnuje problematice implementace zpracování digitálních signálů na desce STM32F4 Discovery, rozborem jejích periferií a knihoven. Kapitola pět hodnotí porovnání efektivity výpočtů s/bez využití floating-point jednotky, a s různými kombinacemi nastavení kompilátoru, porovnání rychlosti výpočtu FFT v jazyce C a JSA. Závěrem je shrnutí práce a porovnání výsledků.

1 SEZNÁMENÍ S PROCESORY Z RODINY ARM

V oblasti DSP, digitálního zpracování signálů, se v současnosti využívá výkonných 32bitových mikrořadičů. Tyto řadiče využívají několik typů architektur, stále ve větší míře se ale prosazují čipy s jádrem ARM. ARM, původně Acorn RISC Machine, je rodina procesorů s redukovanou instrukční sadou vyvíjená britskou společností ARM Holdings.

1.1 RISC

Architektura RISC (Reduced Instruction Set Computing) označuje procesory s návrhem zaměřeným na jednoduchou, vysoce specializovanou sadu strojových instrukcí [4]. Oproti architektuře CISC (Complex Instruction Set Computing) [5], používanou například x86 procesory, mohou obsahovat větší množství úžeji zaměřených instrukcí. Dalším znakem RISC je použití architektury zvané load-store, kde se přistupuje do paměti pomocí instrukcí učených právě k tomuto účelu namísto využití částí komplexnějších instrukcí. Kromě rodiny ARM lze k RISC procesorům řadit také například procesory od Sun Microsystems (řada Sparc), Atmel AVR nebo procesory Power (IBM PowerPC). Za zmínku stojí také od RISC odvozená architektura VLIW (Very Long Instruction Word). VLIW je architektura umožňující instrukční paralelismus aniž by CPU musel vyhodnocovat návaznost a možné kolize operací.

1.1.1 Historie RISC

V raných dobách počítačového průmyslu se programovalo převážně v jazyce symbolických adres nebo ve strojovém kódu. Tehdejší procesory (zpětně označované CISC) měly proto velice komplexní instrukce, které byly navrženy záměrně tak, aby zvládly co nejvíc úkolů. Aby návrháři čipů ulehčili tehdy nepříliš pokročilým kompilátorům, snažili se přizpůsobit instrukce tak, aby každá z nich mohla přistupovat ke všem dostupným druhům adresování (přímé, nepřímé, sekvenční, atd.). Taková instrukční sada se nazývá ortogonální. Instrukce ortogonální sady nejsou duplicitní, pro každou operaci tedy existuje jen jediná strojová instrukce. V půli 70. let bylo demonstrováno, že velká část programů vytvořená pomocí tehdejších kompilátorů většinu kombinací těchto ortogonálních instrukcí vůbec nevyužívala a že mnoho programů běžící na tehdejších procesorech strávilo většinu času prováděním jednoduchých instrukcí. Také bylo objeveno, že obsáhlejší instrukce nezřídka běžely pomaleji než posloupnost instrukcí jednodušších. Tato skutečnost byla zčásti způsobená uspěchaným návrhem tehdejších procesorů, optimalizovány byly primárně nejpoužívanější instrukce.

Frekvence CPU je limitována nejpomalejší pod-operací libovolné instrukce. Bylo zjištěno, že z hlediska výkonu je výhodnější instrukce rozdělit do drobnějších, a to i za cenu zvýšeného počtu registrů. Tehdejší počítače obsahovaly relativně málo registrů hned z několika důvodů. Zaprvé víc registrů znamená delší čas strávený čtením a ukládáním informací. Zadruhé vyšší počet registrů vyžaduje vyšší počet instrukčních bitů, což mimo jiné snižuje hustotu kódu (zvyšuje spotřebu paměti). Zatřetí výroba

registrů v CPU je náročnější na prostředky než výroba externí paměti.

Počátky RISC sahají ke konci 70. a počátku 80. let, kdy pokroky ve výrobě polovodičových součástek dovolily navýšit počet registrů za rozumnou cenu. Pod vedením IBM, Stanfordovy univerzity a Kalifornské univerzity v Berkeley vznikly procesory IBM 801, Stanford MIPS a Berkeley RISC 1 a 2. Tyto procesory sdílely podobnou filozofii návrhu s cílem co nejvíc zrychlit provádění nejjednodušších operací. Projekty těchto procesorů se dále vyvíjely a daly za vznik IBM PowerPC, MIPS architektuře a procesory z Berkeley se vyvinuly ve SPARC. Ke konci 80. let a v raných 90. letech se objevilo větší množství podobně koncipovaných procesorů, které se uchytily především v embedded sektoru jako součásti tiskáren, síťových prvků apod.

Přes mnohé úspěchy a uplatnění se procesorům založeným na RISC stále nedaří proniknout do sféry stolních počítačů a korporátních serverů, kde stále dominuje architektura Intel x86. Důvodem je rozsáhlá aplikační základna (včetně operačních systémů) psaná, nebo kompilovaná pro x86. Zadruhé Intel využil výhodu svého postavení na trhu a investoval obrovské částky do rozvoje vývoje i výroby. Od dob Pentia Pro (1995) využívá Intel překlad CISC instrukcí na RISC, čímž z velké části smazává nevýhody CISC při zachování zpětné kompatibility instrukcí.

1.1.2 Základní rysy RISC, porovnání s CISC

RISC procesor bude mít pro jakoukoli výkonnostní třídu méně tranzistorů dedikovaných logickým jednotkám. To v minulosti umožnilo návrhářům zvýšit počet registrů a procesory učinit vhodnými pro paralelizaci instrukcí. Hlavním rysem RISC architektury je tedy instrukční sada přizpůsobená vysoce pravidelnému běhu instrukcí v tzv. pipeline. Dalšími typickými rysy architektury jsou:

- Operace s externí pamětí pouze pomocí instrukcí load nebo store. Ostatní instrukce jsou omezeny pouze na práci s interními registry.
- Jednotný formát instrukcí - informace o vykonávané operaci (příznakové bity, opcode) je vždy na stejné bitové pozici u všech instrukcí.
- Zpracování jedné instrukce za cyklus.
- Existence obecně účelných registrů (General-purpose registers) zjednodušující návrh kompilátoru a umožňující ukládání jak dat, tak adres.
- Nízký počet hardwarově definovaných datových typů v protikladu k procesorům CISC, které v několika případech podporují komplexní čísla nebo obsahují instrukce na obsluhu řetězců.
- Jednoduché adresní režimy implementující složitější adresování pomocí sekvence aritmetických nebo load-store operací.

Tyto rysy mohou působit jako náhodný shluk nesouvisejících vlastností, ale každá z nich pomáhá udržet pravidelný průběh paralelizmu úloh (pipelining) provádějící instrukci každý takt.

Pipelining, standartní funkce procesorů s architekturou RISC, pracuje na principu využití různých částí procesoru na různé části instrukcí, čímž je umožněno vykonání

většího počtu funkcí za stejný čas, než bez využití řetězení. Pipeline v klasickém pojetí má 5 fází.

- Načtení instrukce z paměti (fetch). Čítač instrukcí (program counter, PC), registr obsahující adresu aktuálně vykonávané instrukce, dostává adresy instrukcí od prediktoru, který zvětšuje adresu o 4 byty (32 bitů). Problém nastává v případě větvení, skoku, nebo výjimky v programu, kdy je adresa následující instrukce jiná. V tomto případě je potřeba pipeline vyprázdnit a začít od správné adresy znovu, což znamená zdržení. Modernější procesory obsahují jednotku předpovídání skoku.
- Dekódování instrukce a načtení registrů (decode)
- Provedení instrukce (execute). Provedení některých instrukcí může trvat déle, než jeden cyklus. Příkladem může být celočíselné (integer) násobení a dělení, nebo libovolné operace ve floating-point. Takové instrukce jsou předány jednotce na ně specializované a zbytek pipeline pokračuje dál. Pro prevenci obtíží při zápisu dat zapisují takové instrukce výsledky operací do speciálně přidělených registrů.
- Čtení z paměti (Access)
- Zápis výsledků do paměti (Writeback)

Model sekvenčního zpracování instrukcí předpokládá dokončení každé instrukce před voláním další, což není při řetězení splněno a mohou vzniknout hazardy. Kompilátory proto musí být navrženy tak, aby se takovým situacím vyhnuly.

Vztah 1.1 [5] ukazuje, jak je doba běhu programu ovlivněna vlastnostmi procesoru:

$$\frac{Doba\ běhu}{Program} = \frac{Doba\ běhu}{Cyklus} + \frac{\text{Počet cyklů}}{Instrukce} + \frac{\text{Počet instrukcí}}{Program}. \tag{1.1}$$

Zatímco CISC procesor volí cestu co nejmenšího počtu instrukcí na program za cenu vyššího počtu cyklů na instrukci, RISC jde opačnou cestou. V případě poměru jednoduchých instrukcí (3 cykly) ku složitějším (16 cyklů u CISC, sekvence 12 instrukcí o 2 cyklech u RISC) 75 ku 25 bude trvat běh programu na CISC procesoru 625 cyklů, na RISC to bude již 750 cyklů, tedy o 125 cyklů víc. V případě poměru 95/5 je již situace opačná, tedy ve prospěch RISC procesoru. Vhodnost použití konkrétní architektury je tedy závislá na aplikaci. Zatímco v aplikacích jako ovládání strojů nebo směrování síťových paketů bude RISC vhodnější, při použití například rozličných CAD programů již bude výhoda na straně CISC.

1.2 Architektura ARM

Architektura ARM, základ každého ARM procesoru, se za roky vyvinula tak, aby vyhovovala rostoucí poptávce po nové funkcionalitě a novým trhům. ARM odpovídá architektuře RISC s několika vylepšeními umožňujícími kompromis mezi vysokým výkonem, hustotou kódu a nízkou spotřebou a velikostí jádra.

Firma ARM Holdings vyvíjí instrukční sady a architekturu veškerých produktů založených na procesorech ARM, ale nevyrábí je. Návrhy čipů licencuje výrobcům třetích stran, kteří si poté sami navrhují své produkty na základě nakoupených práv. Mezi takové výrobce patří Apple, Freescale Semiconductor, Qualcomm, STMicroelectronics nebo Samsung Electronics. ARM je celosvětově nejpoužívanější architekturou (na počty kusů) se zastoupením převážně v embedded sektoru a mobilních zařízeních. Aby mohl ARM pokrývat takto široké portfolio, nabízí několik výkonnostních tříd, které se liší instrukční sadou, délkou paměťových adres, periferiemi atd. [6].

1.2.1 Rozdělení procesorů ARM

Firma ARM Holdings dělí nabízená jádra podle o zaměření na profily A (aplikační procesory), R (procesory optimalizované pro operace v reálném čase) a M (procesory se zaměřením na mikrokontroléry). Přes čtvrtinu prodaných jader tvoří starší procesory ARM7, ARM9 a ARM11.

Aplikační procesory, aktuálně ARMv8-A a ARMv7-A nabízí podporu 64bitových ALU při zachování zpětné kompatibility. Z pohledu instrukční sady tedy nabízí jak starší A32, tak novější A64 a T32 (Thumb) – 16bitová podkategorie 32bitové sady obsahující nejčastěji používané instrukce. Procesory z této kategorie, označované jako Cortex-A, nacházejí uplatnění především ve spotřební elektronice, jako jsou komunikátory nebo tablety. Porovnávání jednotlivých řad není náplní této práce z důvodu volby procesoru z jiné skupiny jader.

ARMv8-R, tedy procesory pro nasazení v reálném čase nabízejí pouze 32bitovou architekturu a jsou zaměřeny na zpracování dat v reálném čase díky podpoře rychlých přerušení (low latency interrupt). Tato jádra také nabízí pokročilé možnosti debugování, obvody pro digitální zpracování signálů a zvuku, podporu ECC (kontrola a oprava chyb pomocí kontroly integrity dat). Tyto vlastnosti předurčují Cortexy-R pro použití v průmyslu jako řídící jednotky v automobilech, síťových prvcích nebo jako řadiče disků.

Cortex-M procesory jsou nejjednodušší skupinou jader vyvíjenou ARM Holdings. Podobě jako Cortex-R nabízí rozšíření v podobě FPU jednotky, rychlou odezvu přerušení, pokročilé debugování a dobrý výpočetní výkon na watt. Omezení v podobě pouze jednojádrových procesorů a podpory pouze 32bitové instrukční sady Thumb jsou pro použití v mikrokontrolérech spíše výhodou. Na jádrech Cortex-M3 a Cortex-M0 jsou postaveny procesory SecurCore používané v čipových kartách (platební karty, SIM).

1.2.2 Instrukční sady Thumb a ARM

Instrukční sada ARM je 32bitová, instrukce Thumb měly zpočátku (ARMv4) 16 bitů. Instrukční sada Thumb nabízela díky kratším instrukcím větší hustotu kódu, avšak na některé operace bylo potřeba víc instrukcí, což mělo za následek horší výkon.

S architekturou ARMv6 byla představena instrukční sada Thumb 2, která kombinovala 16 a 32bitové instrukce, čímž bylo dosaženo výkonu instrukcí ARM spolu s výhodami spojenými s menší velikostí kódu.

Tabulka 1.1 Skupiny instrukcí ARM a Thumb podle jejich funkce.

Skupina instrukcí	Popis
Větvení a podmínky	Tyto instrukce provádí následující: • Větvení do podprogramu • Větvení k vytváření cyklů • Větvení ve strukturách podmínek • Přepínání mezi instrukční sadou ARM a Thumb
Zpracování dat	Tyto instrukce pracují s general-purpose registry. Provádí operace jako sčítání, odčítání, bitové operace s registry a výsledek ukládají do třetího registru. Mohou pracovat s hodnotou v registru nebo hodnotou přímo dodanou v instrukci. Výsledky delší než 32 bitů se ukládají do 2 registrů.
Load a Store	Tyto instrukce načítají nebo ukládají hodnoty z/do paměti.
Vícenásobné load a store	Tyto instrukce načítají a ukládají obsah několika registrů za sebou.
Přístup ke status registru	Tyto instrukce přesouvají hodnoty ze status registrů do general-purpose registrů a naopak
Koprocesor	Tyto instrukce se starají o rozšíření ARM architektury.

1.2.3 Porovnání jader Cortex-M0 až Cortex-M7

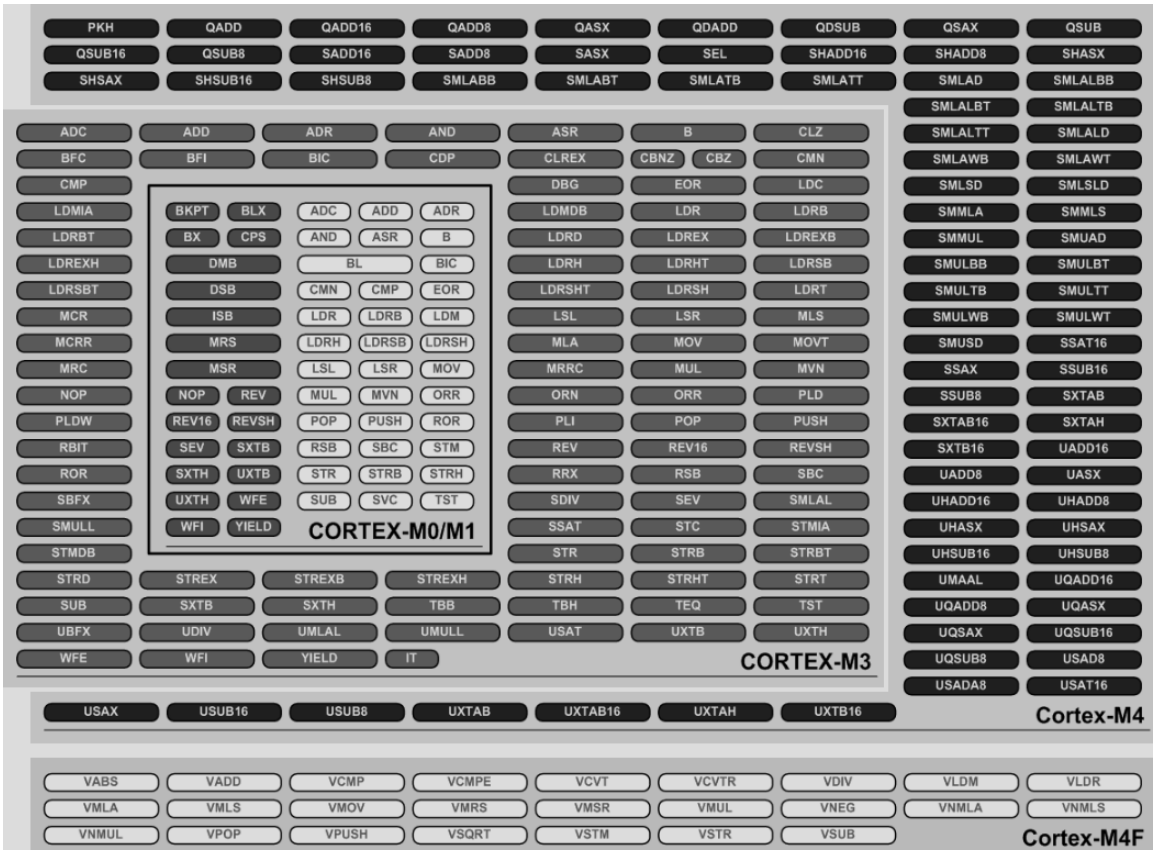
Veškeré procesory Cortex-M nabízejí stejnou funkcionalitu jako jejich nižší příbuzní a přidávají nějakou funkci navíc. Silnější jádra jsou tedy plně kompatibilní s kódem určeným pro slabší procesor typu Cortex-M.

Nejmenším zástupcem rodiny M je Cortex-M0. V minimální konfiguraci se rozkládá na pouhých 0,007 mm2 a obsahuje 12 tisíc hradel s celkovou spotřebou v řádu jednotek až desítek µW/MHz. Díky pouhým 56 instrukcím se není obtížné seznámit s dokumentací, což jej předurčuje pro využití v nenáročných aplikacích. Jeho nástupce, Cortex-M0+, se dále zaměřuje na úsporu energie (spotřeba nižší o 25% než u M0).

Cortex-M3 nabízí kompromis mezi výkonem a spotřebou, což z něj činí nejpoužívanější mikroprocesor v průmyslových kontrolérech. Oproti menšímu M0 nabízí vyšší výkon na takt (DMIPS/MHz), víc rozhraní, pokročilejší architekturu přerušení.

Novějším zástupcem rodiny M je Cortex-M7 s dvojnásobným výkonem na takt proti

M3, výkonnější floating-point jednotkou (než nabízí M4) při zachování podobné spotřeby. Mezi další vlastnosti patří superskalární architektura (zpracování 2 instrukcí v 1 taktu), přítomnost mezipaměti (cache) pro kompenzaci pomalého úložiště.



Obrázek 1.1 Porovnání jader Cortex-M z pohledu instrukční sady a zpětné kompatibility [12].

Nejnovějšími přírůstky v řadě Cortex-M jsou mikrořadiče Cortex-M23 a Cortex-M33 postavená okolo jádra ARMv8-M a technologie TrustZone zajišťující integritu a ochranu dat. Tyto čipy obsahují volitelně podporu DSP operací a/nebo FPU jednotky. V ostatních parametrech se podobají čipům Cortex-M4.

1.2.4 Vlastnosti jádra Cortex-M4

Procesor Cortex-M4 je výkonný procesor navržen pro použití v mikrokontrolérech. Výkonostně spadá mezi jádra M3 a M7. Jádra Cortex-M4 používají architekturu ARMv7E-M, která nabízí podporu Thumb 2 instrukcí, ale již nepodporuje starší ARM instrukce. Pro vývojáře nabízí M4 několik užitečných výhod, jako rychlou obsluhu přerušení, pokročilé možnosti debugování a možností obsahovat použitelné periferie jako FPU, MPU (Memory Protection Unit) aj. [6]

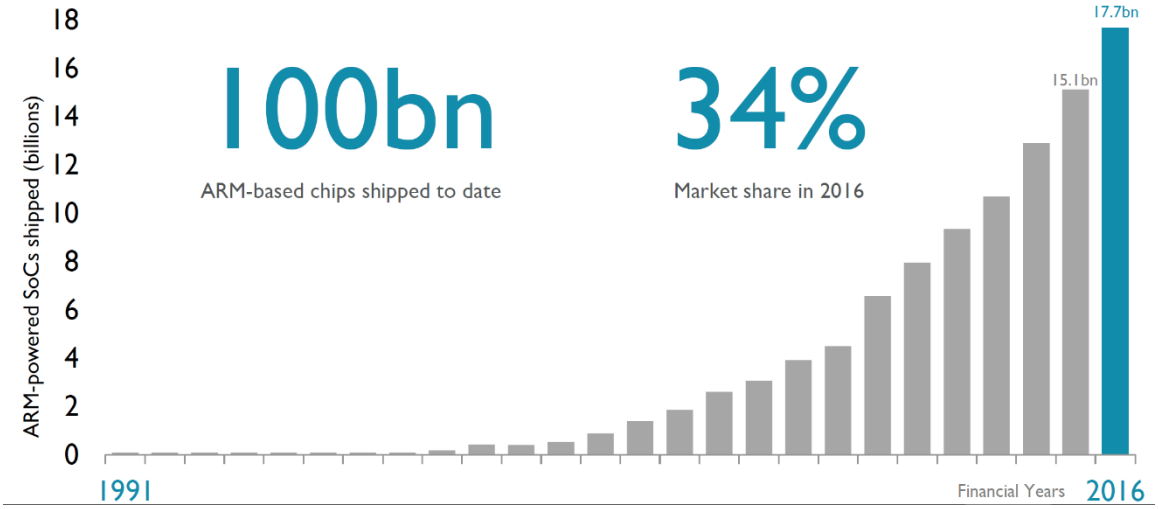
Procesor Cortex-M4 je postaven na Harvardské architektuře s třístupňovou pipeline. Odděluje tedy paměť programu a dat a jejich sběrnice, což procesoru umožnuje navýšit rychlost zpracování pomocí paralelního přístupu k pamětem. Mikroprocesory s jádry Cortex-M4 jsou nejslabší řadou ARM mikrořadičů vhodnou pro operace v plovoucí řádové čárce a pro aplikace v oblasti zpracování digitálního signálu.

Podobně jako u většiny DSP, i u procesorů Cortex-M4 je základním prvkem násobička. Ta podporuje instrukce pro násobení až 32bitových operandů, kdy výsledek je uložen jako 32 nebo 64bitová hodnota. Většinu instrukcí provádí násobička během 1 cyklu.

1.2.5 Možnosti architektury ARM

Za posledních pět let vyrostl trh s polovodiči o 4%, 70% prodaných čipů v roce 2016 obsahovala procesor, v roce 2011 to bylo méně než 40%. Za tuto dobu byl trh s mobilními zařízeními ovládnut ARM procesory, které zdvojnásobily svoje prodeje. Nabízí skvělý poměr výkon/spotřeba, jejich návrh, produkce a nasazení v produktech je levnější než u konkurenčních platforem. To vedlo velké výrobce mobilního hardwaru (Qualcomm, Apple) k návrhu vlastních SoC (System-on-Chip) založených na licenčních jádrech od ARM Holdings.

Překotný vývoj ARM jader zvedl jejich výkon natolik, že se otvírají nové možnosti jejich využití tam, kde donedávna dominovala zavedená x86 architektura. Ve sféře stolních PC nelze čekat v dohledné době výrazné změny, avšak na poli serverů se jeví využití ARM jader smysluplněji. ARM nabízí výrobně levnější variantu proti x86 (dominovanou v posledních letech Intelem), s možností nákupu hardwaru od více výrobců. Na současné úrovni výrobních technologií (10-14nm) je obtížné dosáhnout vysoké výtěžnosti složitých x86 jader, proto je jak z technologického, tak finančního hlediska, výhodná výroba jednodušších, mnoho-jádrových ARM čipů. Častějšímu nasazení do serverové oblasti zatím brání nepříliš rozvinutý ekosystém, podpora softwaru, případně dostupnost hardwaru a potenciální problémy při nasazení výrazně odlišného systému. Obrázek 1.2 ukazuje graf počtu prodaných SoC (system on chip) obsahujících ARM procesor za poslední roky. [13]



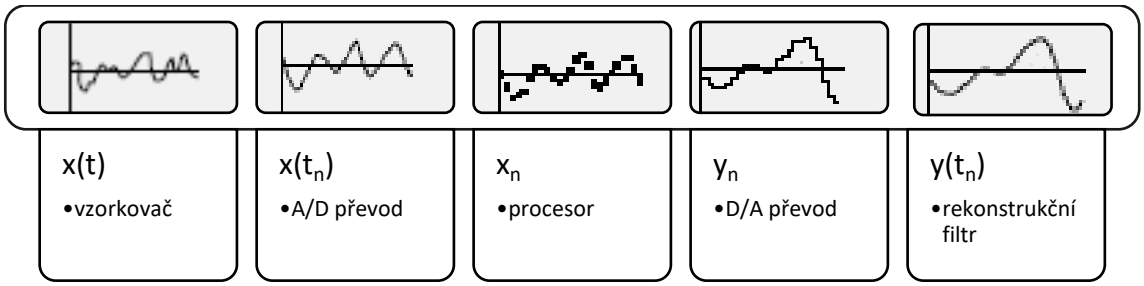
Obrázek 1.2 Porovnání prodejů čipů založených na ARM procesorech v posledních letech [13].

2 ZPRACOVÁNÍ DIGITÁLNÍCH SIGNÁLŮ (DSP)

Signály figurují jako prostředky pro komunikaci mezi lidmi a mezi lidmi a počítači. Jsou používány ke zkoumání okolního prostředí pro poodhalení detailů jinak špatně zaznamenatelných. Zpracování signálů se zabývá reprezentací, transformací a manipulací se signály a informacemi, které obsahují. Překotný vývoj číslicové techniky umožnil nabídnout alternativu k dříve čistě analogovému zpracování signálů.

2.1 Analogové a číslicové zpracování signálů

Digitální zpracování signálů přináší proti analogovému přístupu několik výhod. Nabízí pružnost zpracování, kdy snadnou změnou konstant programu, f_{vz} lze program přizpůsobit. Dále nabízí časovou stálost (odpadá ladění), snadnou slučitelnost s informačními systémy a realizaci komplikovaných způsobů zpracování, z nichž některé ani analogově realizovat nelze (například rozpoznávání obličejů). Omezením je nutnost A/D převodu, zaokrouhlovací chyby a frekvenční rozsah. Následující diagram (obrázek 2.1) znázorňuje postup digitálního zpracování analogového signálu.



Obrázek 2.1 Postup digitálního zpracování analogového signálu [2].

2.2 Fourierova transformace

Fourierova transformace převádí signál mezi časovým a frekvenčním znázorněním pomocí funkce komplexní exponenciály. Fourierova transformace S(ω) je definována následujícím vztahem [2]:

$$S(\omega) = \int_{-\infty}^{\infty} s(t) e^{-j\omega t} dt, \tag{2.1}$$

Kde s(t) je vstupní posloupnost vzorků, $e^{-j\omega t}$ představuje rotaci v komplexních souřadnicích a *S(ω)* je výstupní sekvence.

Pro transformaci signálu z oblasti spektra do časové oblasti slouží zpětná Fourierova transformace [2]:

$$s(t) = FT^{-1}\{S(\omega)\} = \frac{1}{2\pi}\int_{-\infty}^{\infty} S(\omega)e^{j\omega t}d\omega\,. \quad (2.2)$$

2.2.1 DFT, FFT

Pro účely zpracování číslicových signálů slouží modifikace Fourierovy transformace, numerická metoda zvaná diskrétní Fourierova transformace (DFT) schopná vyjádření spektra ze vzorků konečného intervalu signálu. Vstupem DFT je diskrétní navzorkovaný signál a výstupem jeho diskrétní spektrum. Pro svou výpočetní náročnost je její aplikace v teorii zpracování signálů omezená na malé počty vzorků. Z důvodu důležitosti DFT v oblasti spektrální analýzy a jiných oblastí digitálního zpracování signálů se DFT dostalo pozornosti vědců a matematiků. Matematicky je diskrétní Fourierova transformace vstupní sekvence x(n) o délce N je definována vztahem [1]

$$\mathrm{X(k)} = \sum_{\mathrm{n=0}}^{\mathrm{N-1}} \mathrm{x(n)} \cdot \mathrm{W^{nk}} =$$

$$\sum_{\mathrm{n=0}}^{\mathrm{N-1}} \mathrm{x(n)} \cdot \cos\left(\frac{2\pi \mathrm{nk}}{\mathrm{N}}\right) - j * \sum_{\mathrm{n=0}}^{\mathrm{N-1}} \mathrm{x(n)} \cdot \sin\left(\frac{2\pi \mathrm{nk}}{\mathrm{N}}\right), \quad (2.3)$$

kde W^{nk} je tradičně označována jako twiddle konstanta definovaná vztahem [1]

$$\mathrm{W^{nk}} = \mathrm{e^{-j2\pi nk/N}}\,, \quad (2.4)$$

k/N představuje úhel twiddle konstanty.

Pro výpočet DFT je potřeba provést N^2 komplexních součinů, což je neefektivní. DFT nevyužívá symetrie a periodicity spektra. Proces výpočtu diskrétní Fourierovy transformace DFT je optimalizován pomocí efektivních algoritmů souhrnně označovaných jako FFT (Fast Fourier Transform), které využívají zmíněné periodicity a symetrie. Principem FFT je rozdělení DFT o velikosti $N=N_1N_2$ na mnoho menších DFT o velikostech N_1 a N_2. Vstup i výstup transformace je v komplexním tvaru. Základní dělení FFT je podle faktorizace na takzvané radixy (číselné základy). Pro radix 2 platí, že $N=2^n$, u radixu 4 pak $N=4^n$ a u radixu 8 platí, že $N=8^n$. Podle pozice, kdy dochází k rozdělení na menší DFT, se FFT dělí na decimaci (rozklad) v časové (DIT) a decimaci ve frekvenční (DIF) oblasti. Zatímco DIT FFT je založena na rozkladu výpočtů DFT pomocí tvorby menších a menších posloupností vstupních dat x(n), DIF FFT se zaměřuje na rozklad výstupní posloupnosti X(k).

2.2.2 Radix-2

Pokud má vstupní posloupnost x(n) délku $N=2^n$, lze ji dělit 2 a získat tak z vyjádření DFT ve tvaru:

$\mathrm{X(k)} = \sum_{\mathrm{n=0}}^{\mathrm{N-1}} \mathrm{x(n)} \cdot \mathrm{W^{nk}}$ vyjádření ve tvaru [1]: (2.3)

$$\mathrm{X(k)} = \sum_{\mathrm{m=0}}^{\left(\frac{\mathrm{N}}{2}\right)-1} \mathrm{x(2m)} \cdot \mathrm{W^{2mk}} + \sum_{\mathrm{m=0}}^{\left(\frac{\mathrm{N}}{2}\right)-1} \mathrm{x(2m+1)} \cdot \mathrm{W^{k(2m+1)}} \quad (2.5)$$

obsahující v jedné sumě sudé a ve druhé liché vzorky. Po úpravě dostáváme [1]:

$$X(k) = \sum_{m=0}^{\left(\frac{N}{2}\right)-1} x(2m) \cdot W^{km} + W^{k} \sum_{m=0}^{\left(\frac{N}{2}\right)-1} x(2m+1) \cdot W^{km}. \quad (2.6)$$

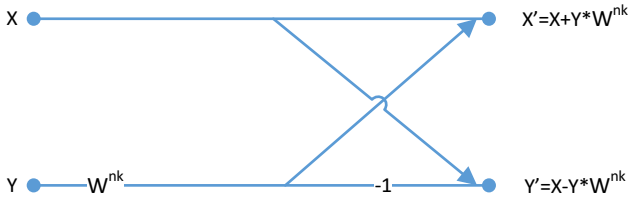
Substitucí $\sum_{m=0}^{\left(\frac{N}{2}\right)-1} x(2m) \cdot W^{km} = F_1$ a $\sum_{m=0}^{\left(\frac{N}{2}\right)-1} x(2m+1) \cdot W^{km} = F_2$ a využitím periodicity F_1 a F_2 s periodou $N/2$ získáváme:

$$X(k) = F_1(k) + W^{k} F_2(k) \text{ a} \quad (2.7)$$

$$X(k + N/2) = F_1(k) - W^{k} F_2(k). \quad (2.8)$$

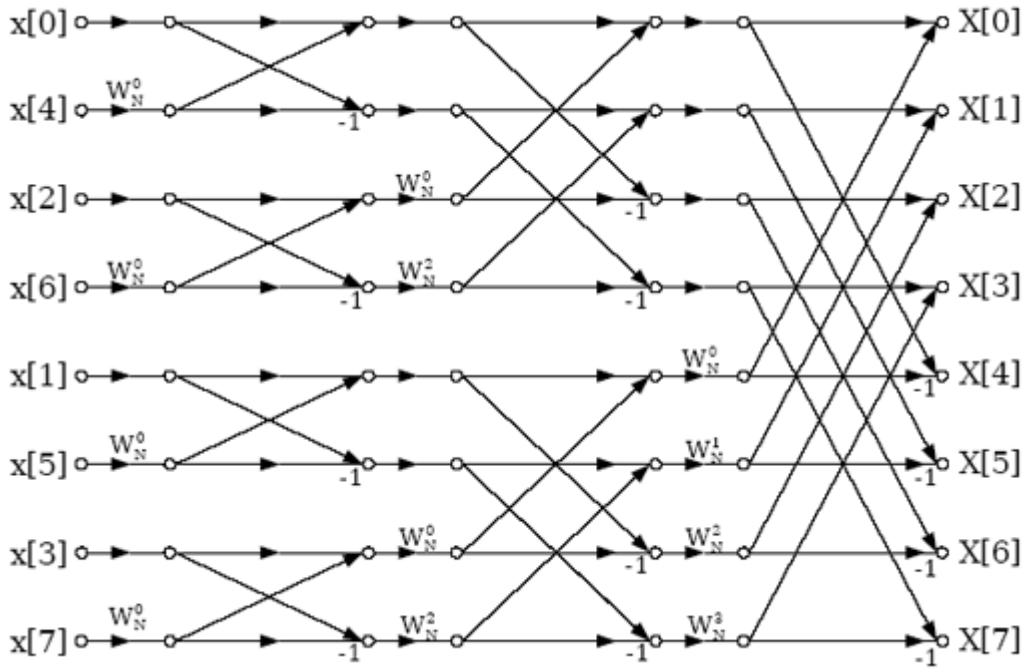
Tento postup se nazývá decimace v časové oblasti (DIT – Decimation In Time). Proces decimace může být opakován, až je dosaženo $N/2$ jednoduchých DFT prvního řádu (velikosti 2) nazývaných motýlek (butterfly). Pro $N=2^n$ může být proces proveden $\log_2 N$ krát, čímž se počet operací komplexního násobení sníží na $(N/2)\log_2 N$. Počet komplexních součtů bude $N\log_2 N$.

Na Obrázku 2.2 je znázorněn motýlek pro DIT – Decimation In Time, kde X a Y jsou vstupní hodnoty a X'' a Y'' jsou výstupní hodnoty DFT a W^{nk} je twiddle konstanta.



Obrázek 2.2 Algoritmus výpočtu 2bodé DFT v případě DIT pro radix-2. [1].

V případě $N=2^3$ bude FFT radix-2 rozdělen na 3 kroky. Názornější ukázku poskytuje Obrázek 2.3.



Obrázek 2.3 Algoritmus výpočtu FFT pro *N*=8 s decimací v časové oblasti pro radix2 [1].

Jak je z obrázku 2.3 patrné, výpočet frekvenčních koeficientů pro *N=8* se provádí ve 3 postupných fázích. V první fázi jsou vypočteny 4 2bodé DFT, ve druhé 2 4bodé a nakonec jedna 8bodá DFT.

Početní náročnost DFT se z původní *O(n2),* kvadratické, sníží na *O(n log n),* lineárnělogarimickou. Použitý zápis popisuje limitní chování funkce, tedy když se hodnota n blíží k nekonečnu [11].

Důležitý poznatek se týká pořadí vstupních dat po tom, co jsou n-krát decimována. Zamíchání s pořadím dat se nazývá bitově reverzní pořadí a pro správný výsledek FFT je důležité vzorky buď před, nebo po provedení FFT přetřídit.

Druhým důležitým algoritmem pro radix-2 je decimace ve frekvenční oblasti. K jejímu dosažení je potřeba rozdělit vzorec pro DFT do dvou sum, z nichž jedna obsahuje první polovinu vzorků a druhá suma druhou polovinu vzorků [25]:

$$\mathrm{X(k)} = \sum_{\mathrm{n=0}}^{\left(\frac{\mathrm{N}}{2}\right)-1} \mathrm{x(n)} \cdot \mathrm{W^{nk}} + \sum_{\mathrm{n=0}}^{\left(\frac{\mathrm{N}}{2}\right)-1} \mathrm{x(n)} \cdot \mathrm{W^{nk}} \qquad (2.9)$$

$$= \sum_{\mathrm{n=0}}^{(\mathrm{N}/2)-1} \mathrm{x(n)} \cdot \mathrm{W^{nk}} + \mathrm{W^{kN/2}} \sum_{\mathrm{n=0}}^{(\mathrm{N}/2)-1} \mathrm{x}\left(\mathrm{n} + \frac{N}{2}\right) \cdot \mathrm{W^{nk}}. \qquad (2.10)$$

Jelikož $\mathrm{W^{kN/2}}$= (-1)k, můžeme napsat [1]:

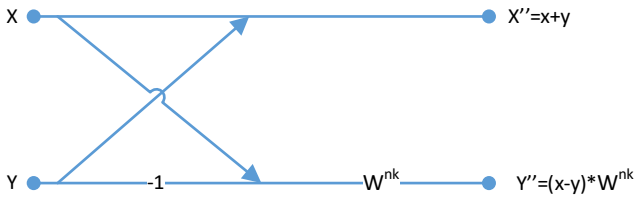
$$X(k) = \sum_{\mathrm{n=0}}^{(\mathrm{N}/2)-1} \mathrm{x(n)} \cdot \mathrm{W^{nk}} + (-1)^k x \sum_{\mathrm{n=0}}^{(\mathrm{N}/2)-1} \left(\mathrm{n} + \frac{N}{2}\right) \cdot \mathrm{W^{nk}}. \qquad (2.11)$$

Nyní je možné rozdělit, decimovat, X(k) na sudé a liché vzorky. S využitím vztahu $\mathrm{W_N^2} = W_{N/2}$ [25] získáme:

$$X(2k) = \sum_{\mathrm{n=0}}^{(\mathrm{N}/2)-1} [\mathrm{x(n)} + \mathrm{x}\left(\mathrm{n} + \frac{N}{2}\right)], \mathrm{k=0, 1, \ldots, N/2 - 1,} \qquad (2.12)$$

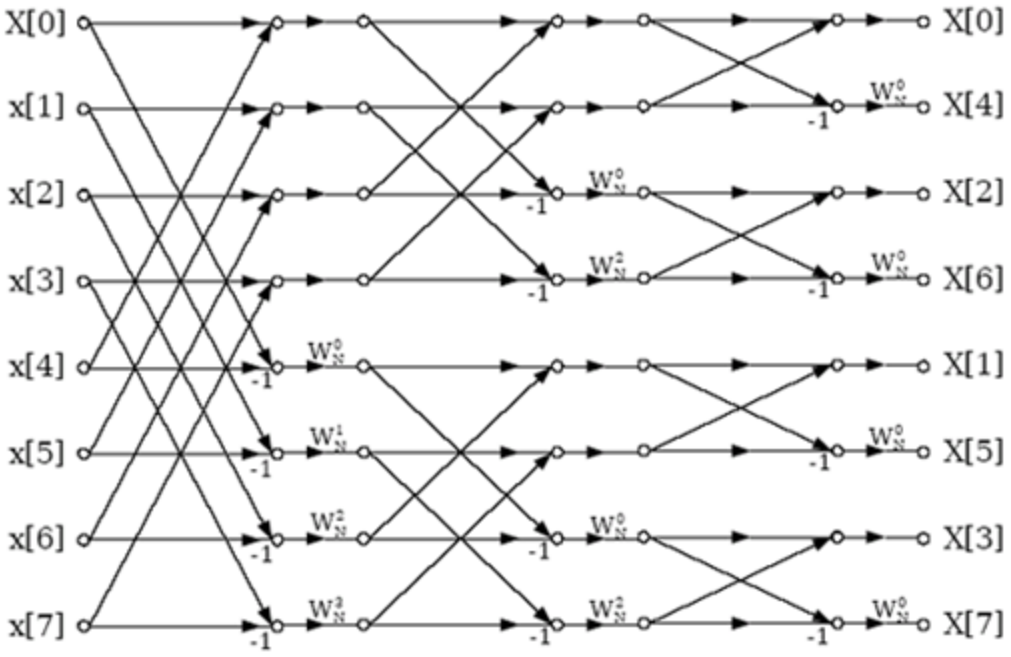
$$X(2k+1) = \sum_{n=0}^{(N/2)-1}[x(n) - x\left(n + \frac{N}{2}\right)], \text{k=0, 1, …, } N/2 \text{ -1.} \quad (2.13)$$

Tuto proceduru je možné opakovat až k dosažení 2bodých DFT. Stejně jako u algoritmu DIT zahrnuje proces $\log_2 N$ decimací, kde každá etapa zahrnuje jednoduché DFT (motýlky). Na Obrázku 2.3 je znázorněn motýlek pro DIF – Decimation In Frequency, kde X a Y jsou vstupní hodnoty a X'' a Y'' jsou výstupní hodnoty DFT a W^{nk} je twiddle konstanta.



Obrázek 2.4 Algoritmus výpočtu 2bodé DFT v případě DIF pro radix-2. [1].

V případě $N=2^3$ bude FFT radix-2 rozdělen na 3 kroky. Názornější ukázku poskytuje Obrázek 2.5, který znázorňuje 8bodou DFT s decimací ve frekvenční oblasti.



Obrázek 2.5 Algoritmus výpočtu FFT pro N=8 s decimací ve frekvenční oblasti pro radix2. [1].

Spoužitím DIT nebo DIF se početní náročnost DFT se z původní O(n2), kvadratické, sníží na O(n log n), lineárnělogarimickou. Použitý zápis popisuje limitní chování funkce, tedy když se hodnota n blíží k nekonečnu [11].

2.2.3 Radix-4

Pokud je počet vzorků $N=4^n$, je samozřejmě stále možné použít radix-2, ale efektivnější je využít radix-4. Stejné pravidlo platí i u vyšších číselných základů (8, 16, …).

Radix-4 s decimací v čase dělí vstupní sekvenci x(n) na 4 části: x(4n), x(4n+1), x(4n+2), x(4n+3), kdy *n* nabývá hodnot od 0 do N/4-1 a získat tak vyjádření DFT ve tvaru [26]:

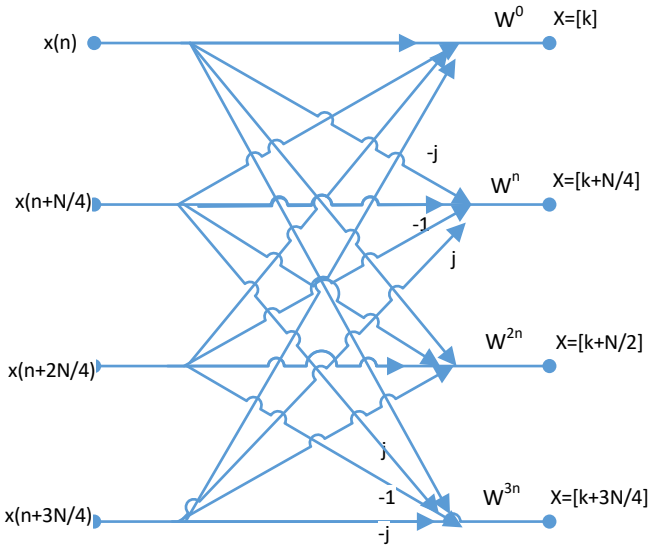
$$X(p,q) = \sum_{l=0}^{3}[W^{lq}F(l,q)]W^{lp}, \tag{2.14}$$

kde $F(l,q) = \sum_{l=0}^{\frac{N}{4}-1} x(4m+l)W^{nq}$, *p=0, 1, 2, 3; l=0, 1, 2, 3; q=0, 1, …, N/4-1* a *X(p, q) = X(N/4p + q).*

Soustava rovnic definující motýlek pro radix-4 DIT může být vyjádřena (pro přehlednost v maticovém tvaru) takto [26]:

$$\begin{bmatrix} X(0,q) \\ X(1,q) \\ X(2,q) \\ X(3,q) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} * \begin{bmatrix} W^0F(0,q) \\ W^qF(1,q) \\ W^{2q}F(2,q) \\ W^{3q}F(3,q) \end{bmatrix} \tag{2.15}$$

Obrázek 2.6 znázorňuje základ výpočtu algoritmu radix-4.



Obrázek 2.6 Algoritmus výpočtu 4bodé DFT pro radix-4. [1].

Algoritmus DIF spočívá v rozdělení N-bodé DFT na 4 menší DFT:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W^{nk} \Rightarrow$$

$$X(4k) = \sum_{n=0}^{\frac{N}{4}-1} \left[x(n) + \left(n + \frac{N}{4}\right) + (-1)^k x\left(n + \frac{N}{2}\right) + j^k x\left(n + \frac{3N}{4}\right)\right] W^0 W_{N/4}^{kn} \quad (2.15)$$

$$X(4k+1) = \sum_{n=0}^{N/4-1} \left[x(n) + \left(n + \frac{N}{4}\right) + (-1)^k x\left(n + \frac{N}{2}\right) + j^k x\left(n + \frac{3N}{4}\right)\right] W^n W_{N/4}^{kn} \quad (2.16)$$

$$X(4k+2) = \sum_{n=0}^{N/4-1} \left[x(n) + \left(n + \frac{N}{4}\right) + (-1)^k x\left(n + \frac{N}{2}\right) + j^k x\left(n + \frac{3N}{4}\right)\right] W^{2n} W_{N/4}^{kn} \quad (2.17)$$

$$X(4k+3) = \sum_{n=0}^{N/4-1} \left[x(n) + \left(n + \frac{N}{4}\right) + (-1)^k x\left(n + \frac{N}{2}\right) + j^k x\left(n + \frac{3N}{4}\right)\right] W^{3n} W_{N/4}^{kn},$$

pro k=0, 1, …, N/4-1. (2.18)

Proces decimace lze opakovat $\log_4 N$-krát, až je dosaženo 4bodých DFT. Za předpokladu (založeno na obrázku 2.6):

$x(n) = Ra + j\,Ia$; $x(k) = Ra' + j\,Ia'$;

$x(n + N/4) = Rb + j\,Ib$; $x(k + N/4) = Rb' + j\,Ib'$;

$x(n + N/2) = Rc + j\,Ic$; $x(k + N/2) = Rc' + j\,Ic'$;

$x(n + 3N/4) = Rd + j\,Id$; $x(k + 3N/4) = Rd' + j\,Id'$;

$W^n = RW^b + jIW^b$; $W^{2n} = RW^b + jIW^b$; $W^{3n} = RW^b + jIW^b$,

Lze podle rovnic 2.15-2.18 sestavit rovnice pro reálnou a imaginární část výstupních hodnot takto:

$$Ra' = Ra + Rb + Rc + Rd \; ; \quad (2.19)$$

$$Ia' = Ia + Ib + Ic + Id \; ; \quad (2.20)$$

$$Rb' = (Ra + Ib - Ic - Id)RW^b - (Ia - Rb - Ic + Rd)IW^b \; ; \quad (2.21)$$

$$Ib' = (Ia - Rb - Ic + Rd)RW^b + (Ra + Ib - Rc - Id)IW^b \; ; \quad (2.22)$$

$$Rc' = (Ra - Rb + Rc - Rd)RW^b - (Ia - Ib + Ic - Id)IW^b \; ; \quad (2.23)$$

$$Ic' = (Ia - Ib + Ic - Id)RW^b + (Ra - Rb + Rc - Rd)IW^b \; ; \quad (2.24)$$

$$Rd' = (Ra - Ib - Rc + Id)RW^b - (Ia + Rb - Ic - Rd)IW^b \; ; \quad (2.25)$$

$$Id' = (Ia + Rb - Ic - Rd)RW^b + (Ra - Ib - Rc + Id)IW^b \; . \quad (2.26)$$

2.3 Filtrace, porovnání FIR a IIR

Filtrem se rozumí zařízení (nebo program), který ze signálu odejme nechtěnou část. V oboru digitálního zpracování signálů se rozlišují dva hlavní typy filtrací podle tvaru jejich impulsní charakteristiky, filtry s konečnou impulsní charakteristikou (FIR) a filtry s nekonečnou impulsní charakteristikou [2]. Výstupem filtru je konvoluce vstupního signálu s impulzní charakteristikou filtru (odezvou na jednotkový impuls). Vlastnosti filtru jsou popsány přenosovou funkcí, její úpravou lze získat frekvenční charakteristiku filtru. Její zpětnou Z transformací je potom možné získat diferenční rovnici filtru.

Filtry FIR jsou vždy stabilní, jejich návrh je jednodušší a výpočet probíhá rychleji, než u IIR. Pro kvalitní filtraci je ale nutné použít vyšší řád filtru.

Filtry IIR mohou být nestabilní a v podstatě se jedná o doplnění FIR rekurzivní částí. Filtry IIR mají vždy nelineární fázovou charakteristiku, k lineární se mohou pouze přiblížit. Obecně nabízí IIR filtry oproti FIR větší možnosti filtrace, již s filtrem nižších řádů lze dosáhnout kvalitního výstupu. S jejich pomocí lze vytvořit i fázovací články, tedy prvky s konstantním přenosem a definovanou fázovou charakteristikou.

2.4 Korelace

U korelační analýzy jde o určení vztahů mezi 2 a více signály. V praxi má význam analyzovat pouze náhodné signály. Jsou-li vyšetřovány 2 časové okamžiky stejného signálu, jedná se o autokorelační funkci [2]:

$$\mathrm{R}(n_1, n_2) = \lim_{\Omega \to \infty} \frac{1}{\Omega} \sum_{\omega=1}^{\Omega} x_\omega(n_1) \cdot x_\omega(n_2). \quad (2.2)$$

Pro případ bílého šumu a volbě časových okamžiků n_1 a n_2 stejných, korelace bude nenulová, pro všechny ostatní případy volby časových okamžiků bude nulová. V případě šumu generovaného pseudonáhodným generátorem korelace odhalí nežádoucí souvislost vzorků.

Korelační analýzu lze aplikovat i jinými způsoby. Příkladem může být detekce známého signálu v dlouhém signálu, navíc různě rušeném (Sonar, EKG, detekce hlasu apod.). Dalším využitím může být vyloučení šumu o známém charakteru z analýzy signálu, nebo detfekce přítomnosti periodického signálu v šumu.

3 ANALÝZA SIGNÁLŮ NA DESCE STM32F4 DISCOVERY

3.1 Platforma

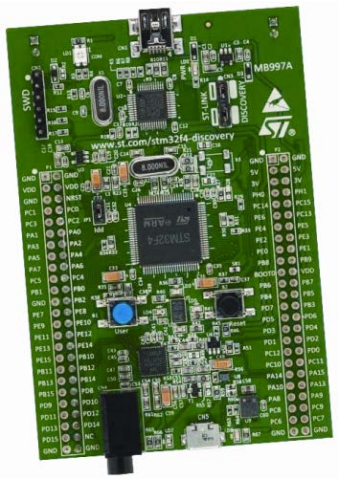
Po dohodě s vedoucím práce a doporučení Ing. Aleše Povalače byla pro účely analýzy signálů zvolena deska STM32F4 Discovery od firmy STMicroelectronics [18]. Deska obsahuje licencované jádro Cortex-M4 od společnosti ARM Holdings vybavené floating-point jednotkou (FPU).

3.1.1 Deska STM32F4 Discovery

Tato vývojářská deska nabízí za danou cenu (10$ bez daní) bohaté příslušenství jako integrovaný zvukový DAC s výstupem na sluchátka a předzesilovačem třídy D, 4 barevné LED, 2 tlačítka, akcelerometr a mikrofon. K desce je pomocí micro-AB konektoru možné připojit USB OTG kabel a přes něj například číst a nahrávat data na externí úložiště.

Desku je možné napájet z 5V USB, jímž se také spojuje s PC za asistence ST-LINK utility. Desku je možné programovat také externím SWD konektorem. Díky přítomnosti 192KB RAM, 1MB flash paměti a zmíněného DAC je deska velice vhodná pro analýzu signálů [9].

Floating-point jednotka podporuje výpočty s plovoucí řádovou čárkou s přesností single (tedy 32 bitů). Výpočet s podporou double je sice možný, ale pomalý – kompilátor bude softwarově emulovat double pomocí celočíselných operací (ne pomocí floatu). Pro snížení spotřeby energie je možné FPU jednotku vypnout, a to i za běhu programu. Obrázek 3.1 nabízí náhled na desku STM32F4 Discovery.



Obrázek 3.1 Vývojářská deska STM F4 Discovery [18].

3.1.2 Kompilátor

V oblasti ARM procesorů je možné volit ze dvou kompilátorů pro jazyk C – GCC nebo kompilátoru od ARM Ltd. Alternativní kompilátory vycházejí z obou zmíněných variant.

Kompilátor GCC (GNU Compiler Collection) [24] je soubor kompilátorů pod projektem GNU, vydáván pod licencí GNU GPL (General Public License) a je považován za standart ve světě unixových operačních systémů. Za 20 let své existence podporuje mnoho jazyků (C, C++, Java, Objective-C, Go, atd.) a platforem (x86, ARM, Freescale).

Kompilátor ARM v5 [22] je výsledkem 20letého vývoje spolu s architekturou ARM. Při jeho návrhu byl kladen důraz na optimalizace, velikost kódu a zaměření na embedded sektor. Nevýhodou je cena za licenci a uzavřenost.

3.1.3 Vývojové prostředí

Pro účely této práce byla zvolena dvě vývojová prostředí – EmBitz v1.11 v kombinaci s kompilátorem GCC a Keil µVision [22] s kompilátorem ARM v5. Lze říci, že pro vývoj na STM Discovery je možné použít prakticky libovolné vývojové prostředí, mezi nabízené alternativy ke zvoleným IDE patří CooCox CoIDE, Eclipse, MS Visual Studio. Obě zvolená prostředí umožňují vývoj v jazyce C a debugging s využitím ST-link GDB serveru.

EmBitz nahrazuje starší Em::Blocks, přidává podporu rozšíření a nadále je šířen pod GNU GPL v3, tedy je možné ho používat zdarma (i pro komerční účely) a bez omezení. EmBitz obsahuje kompilátor GCC GNU v5.4, čímž oproti Em::Blocks částečně zjednodušuje přípravu vývoje.

Keil µVision v5.23 úzce spolupracuje s produkty od STM a ARM Ltd., což umožňuje pohodlnou přípravu desky pro vývoj. Součástí IDE je ARM CC (C compiler) v5.06 build 422 a stejně jako EmBitz umožňuje pokročilý debugging a rozsáhlá nastavení. Licence zdarma je pro nekomerční použití a do velikosti kódu 32kb, což je částečně limitující již pro účely tohoto projektu.

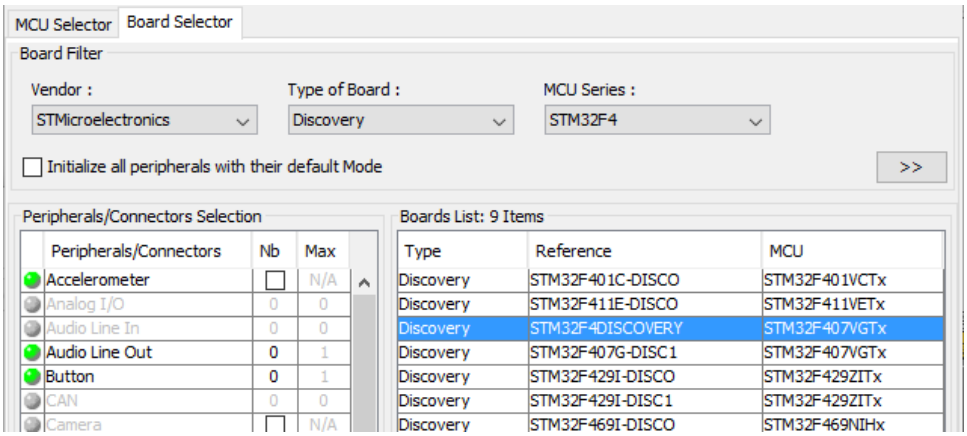
3.2 Příprava prostředí

3.2.1 Knihovny HAL

STMicroelectronics v současné době nabízí 2 balíky knihoven. Starší SPD (standard peripheral drivers) nabízí rozsáhlou dokumentaci a příklady kódů, pohybuje se kolem nich široká komunita vývojářů a podporují všechny starší STM produkty až po desku STM32F4 Discovery zvolenou pro tuto práci. Novější HAL knihovny je možné použít v celém portfoliu nabízených produktů STM, ale dokumentace není na tak vysoké úrovni jako u staršího typu. Veškeré ukázkové kódy jsou navíc dodávány se staršími knihovnami, takže je nelze s novými použít. Z důvodu budoucí kompatibility kódu na novějších mikrokontrolérech (jako STM32F7 a další) byl pro tuto práci zvolen soubor knihoven HAL [10].

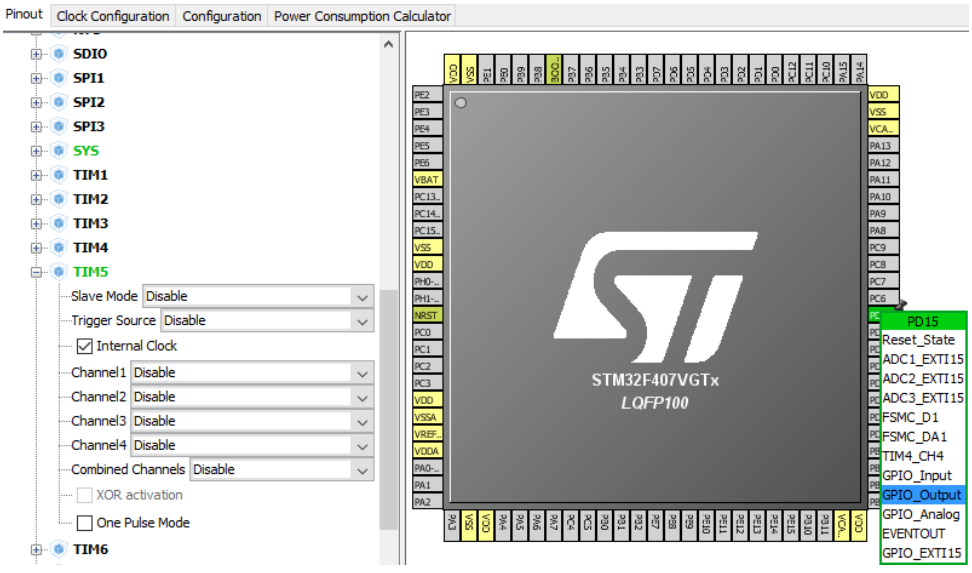
3.2.2 Příprava projektu v STM32CubeMX

V STM32Cube [21] lze nakonfigurovat pomocí GUI veškeré periferie a připravit projekt pro IDE včetně HAL knihoven. Po stisknutí **New Project** a zvolení **MCU Series: STM32F4** v záložce **Board Selector** je potřeba vybrat desku **STM32F4DISCOVERY.** Obrázek 3.2 ukazuje proces výběru desky v prostředí STM32Cube.



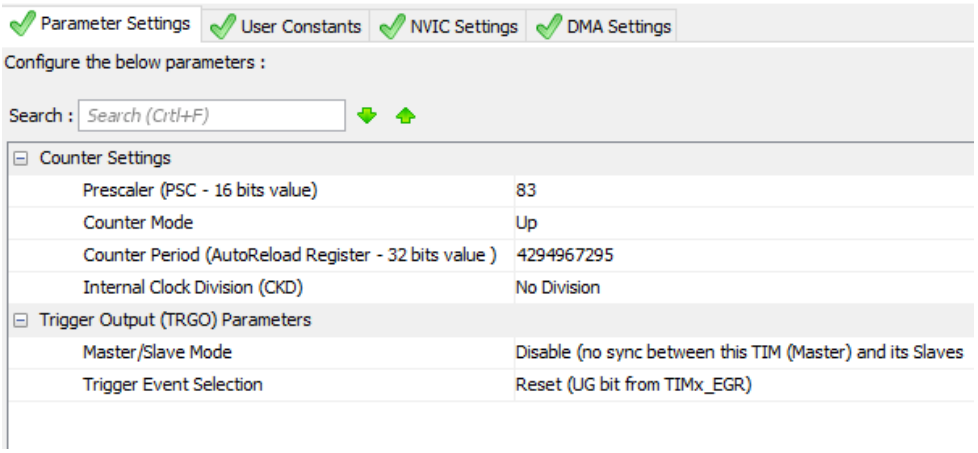
Obrázek 3.2 Proces výběru desky v prostředí STM32CubeMX.

Z periferií je pro účely tohoto projektu potřeba pouze **Timer5**, případně **GPIO15**, timer pro měření doby zpracování signálu a GPIO pro ovládání kontrolní LED (vizuální kontrola). Ve výchozím stavu je v záložce **Pinout** zapnuto hned několik modulů, z nichž je ke splnění zadání tohoto projektu potřeba pouze **Timer**, ideálně **TIM2** nebo **TIM5** pro jejich 32bitovou délku. Klávesovou zkratkou **CTRL+P** lze vymazat veškeré rozmístění pinů. Časovač je nutné zapnout zaškrtnutím položky **Internal Clock** v příslušném menu. Pro vizuální zpětnou vazbu je vhodné povolit jeden z pinů portu D (**PD12-15**) jako **GPIO output**, kterým jsou ovládány LED diody na desce Discovery. Obrázek 3.3 ukazuje nastavení **Pinout**. Obrázek 3.3 ukazuje nastavení periférií na desce Discovery.



Obrázek 3.3 Proces nastavení periferií v prostředí STM32CubeMX.

Na záložce **Configuration** v položce **Control** probíhá nastavování periferií spuštěných na záložce **Pinout**. Poklepáním na **TIMx** lze v nově otevřeném okně **TIMx Configuration** měnit nastavení čítače. Vhodným nastavením hodnoty **PSC** (předličky hodin) a **Coutner Period** (32bitová hodnota - 4 294 967 295) lze dosáhnout takové frekvence čítání, aby při běžných vstupních datech program běžel dostatečně krátkou dobu na to, aby se nezavolalo přerušení způsobené přetečením čítače, a zároveň byla zachována dostatečná rychlost čítání. Časovač je řízen kmitočtem 84MHz (**APB1 Timer Clocks** na záložce **Clock Configuration**), požadovaná doba jedno tiku je 1µs, předdělička **PSC** tedy bude 84-1=83. Obrázek 3.4 ukazuje detail nastavení parametrů čítače **TIM5**.



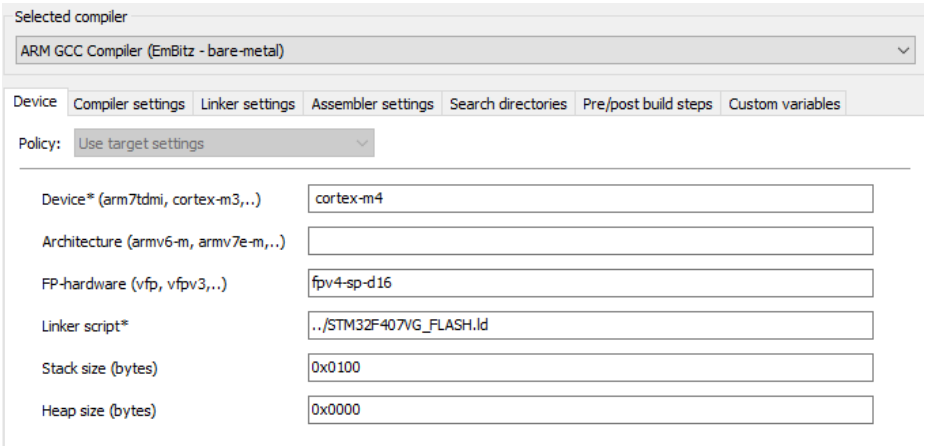
Obrázek 3.4 Proces nastavení parametrů čítače v prostředí STM32CubeMX.

V témž okně na záložce **NVIC Settings** je potřeba zapnout globální funkci starající se o přerušení přetečením čítače.

Klávesovou zkratkou **ALT+P** je možné otevřít nové okno s nastavením projektu, kde je pod položkou **Toolchain/IDE** potřeba vybrat **MDK-ARM V4** v případě použití IDE EmBitz nebo **MDK-ARM V5** v případě µVision IDE. Na vedlejší záložce je potom třeba vybrat položku nastavující kopírování všech použitých knihoven do složky projektu (**necessary library files**). Tato volba je nutná pro pozdější využití DSP knihoven a pro operace s FPU. Po potvrzení je možné pomocí **CTRL+SHIFT+G** vygenerovat šablonu kódu do vybrané složky.

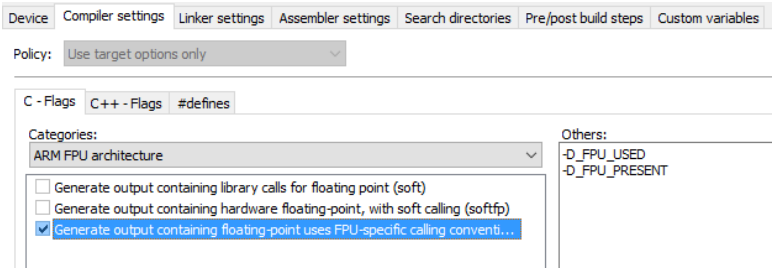
3.2.3 Tvorba projektu ve vývojovém prostředí EmBitz

IDE EmBitz umožnuje import projektů pro Keil µVision 4 a starší, ale pro jejich funkčnost je potřeba provést několik kroků. V **Project – Build options – Device** je třeba po zvolení kompilátoru **ARM GCC** vyplnit položky **device**: „*cortex-m4*“, **FP-hardware**: „*fpv4-sp-d16*“, **linker script**: „*../STM32F407VG_FLASH.ld*“, **stack size**: „*0x0100*“ a **heap size**: „*0x0000*“. Obrázek 3.5 ukazuje podrobnosti nastavení Device pro desku Discovery v prostředí EmBitz.



Obrázek 3.5 Proces nastavení zařízení ve vývojovém prostředí EmBitz.

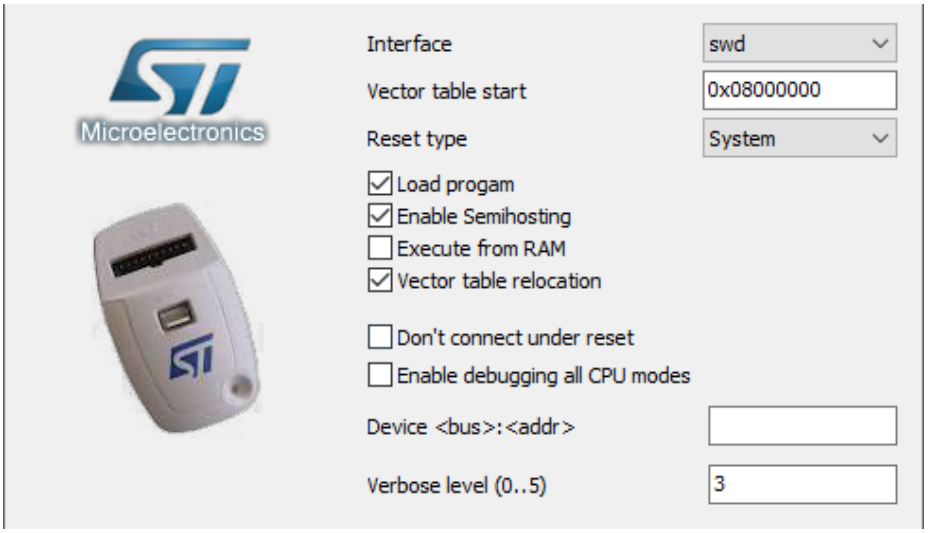
Nastavení kompilátoru na záložce **Compiler settings** vyžaduje přidání direktiv *„D_FPU_USED“* a *„D_FPU_PRESENT“* a zaškrtnutí volby pro hardwarové volání FPU jednotky. Obrázek 3.6 ukazuje nastavení kompilátoru GCC pro využití FPU jednotky.



Obrázek 3.6 Proces nastavení parametrů kompilátoru v prostředí EmBitz.

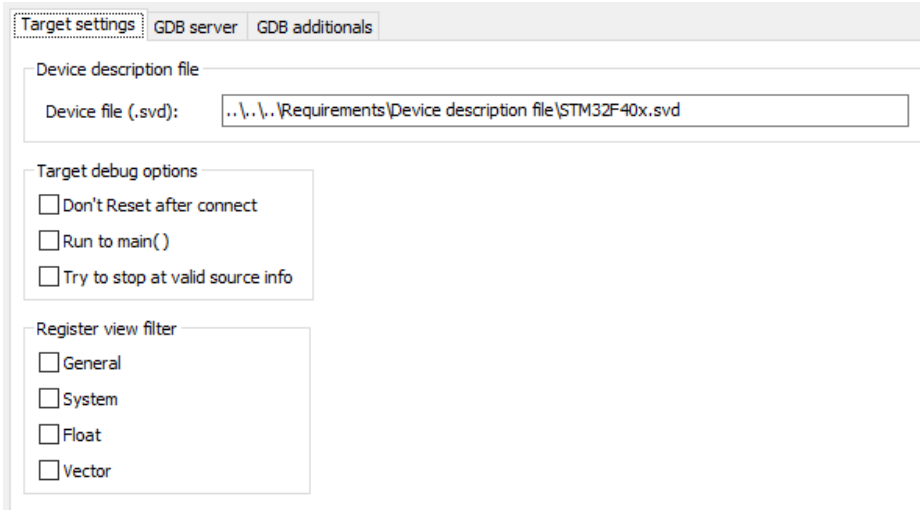
Pro možnost využití debugeru je nutné nainstalovat ovladače ST-Link [23]. Aby fungoval výstup funkce *printf()* je třeba v **Project – Build options – Linker settings – Categories:Library selection** zaškrtnout položku **Use ARM Semihosting**. Semihosting umožňuje přesměrování standardního výstupu **stdout** do debuggeru, který jej následně předává do vývojového prostředí. Pro jeho aktivaci je třeba jej povolit v nastavení projektu i debuggeru, tedy **Debug – Interfaces – GDB server – Settings –** zaškrtnout **Enable Semihosting**. Nakonec stačí už jen přidat standartní knihovnu mezi připojené soubory (includes).

```
#include <stdio.h>
```



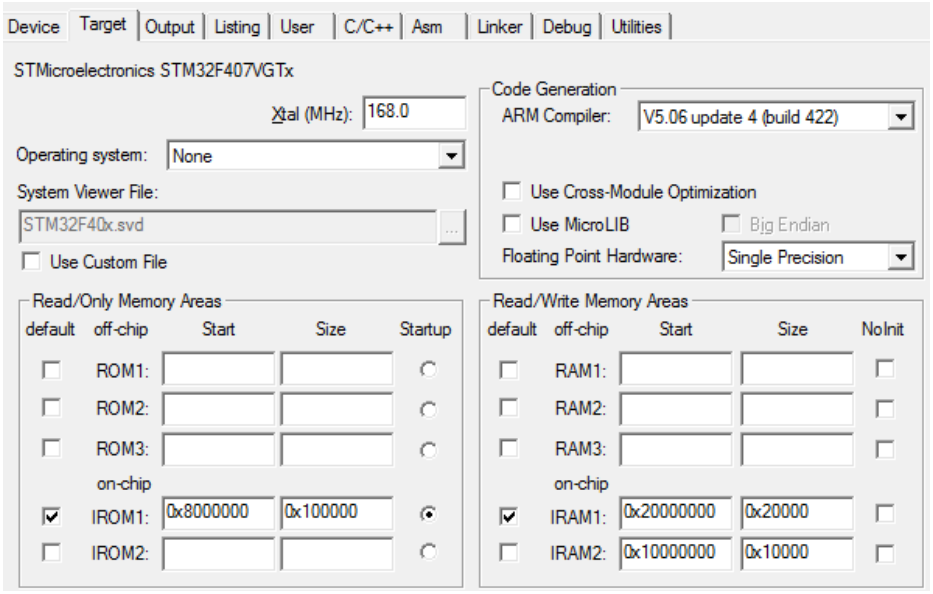
Obrázek 3.7 Proces nastavení debugování v prostředí EmBitz.

Z přílohy je třeba zkopírovat do složky projektu popisný soubor procesoru *„STM32F40x.svd“* a nastavit k němu cestu pod **Debug** – **Interfaces** v záložce **Target settings**. Obrázek 3.8 zobrazuje připojení popisného souboru v prostředí EmBitz.



Obrázek 3.8 Proces nastavení popisného souboru zařízení v prostředí EmBitz.

Po instalaci Keil MDK [22] a ovladačů ST-Link [23] je díky spolupráci STM a Keil projekt po vygenerování v STM CubeMX téměř připraven. Zbývá volba kompilátoru, FPU a nastavení frekvence procesoru. Tato nastavení se provádí v Target options (Alt+ F7).



Obrázek 3.9 Nastavení Target options v prostředí Keil µVision .

4 DSP S VYUŽITÍM FPU JEDNOTKY NA DESCE STM32F4DISCOVERY

Tato kapitola se věnuje rozboru použitých algoritmů a funkcí pro digitální zpracování signálů na desce STM32F4Discovery.

V případě FFT bylo u veškerých výpočtů předpokládáno správné (bitově reverzní v případě radix-2 nebo číselně reverzní v případě radix-4 algoritmů) uspořádání vstupních hodnot, a žádný z algoritmů tedy neimplementuje nebo nevyužívá implementaci bitově/číselně reverzního pořadí.

4.1 Inicializace a měření rychlosti zpracování kódu

Veškeré podpůrné funkce se volají z hlavní funkce `main(void)`. Před spuštěním analyzovaného kódu je třeba inicializovat obsluhu periferií, rozhraní Flash a Systick pomocí:

```
HAL_Init();
MX_GPIO_Init();
```

Po dokončení analyzovaného kódu skočí program do nekonečného while cyklu, ze kterého je pro kontrolu každých 1000ms volána funkce pro rozvěcení kontrolní LED.

```
HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);
```

Jelikož se práce zabývá hodnocením rychlosti zpracování, je potřeba přesně měřit jeho dobu. K tomuto účelu dobře poslouží 32bitový časovač TIM5. Jeho inicializace a spuštění se provádí následujícími příkazy:

```
MX_TIM5_Init();
HAL_TIM_Base_Start_IT(&htim5);
```

Pomocí funkce `__HAL_TIM_GetCounter(&htimx)` lze jednoduše (rychle) získat aktuální hodnotu čítače. Funkcí `printf()` je možné hodnotu vypsat do konzole například na konci programu. Tato funkce vyžaduje podporu semihostingu jak ze strany kompilátoru, tak ze strany debuggeru. Spolehlivě funguje pouze v kombinaci EmBitz a GCC, v prostředí µVision je nutné hodnotu čítače zjistit jinak, například pomocí watcheru v rámci debugu.

Pro ověření funkčnosti poslouží funkce `HAL_Delay()` se vstupním celočíselným parametrem udávaným v milisekundách. Tato funkce nabízí zpoždění založené na časovači Systick, který z důvodu obsluhy jiných přerušení není pro účely měření vhodný. Při zadání zpoždění 1000ms se inkrementuje **TIM5** 999816krát. Volání **HAL_Delay(1000)** tedy ve skutečnosti zpozdí program o 999,816ms.

4.2 Analýza s využitím HAL knihoven

K vešekré analýze s využitím matematických knihoven bylo do paměti uloženo pole

vzorků audiosignálu se spektrem v rozsahu 0-22khz.

Pro potřeby číslicového zpracování signálu využívá projekt HAL MSP [10] (signal processing) knihovny, ty je potřeba přidat do projektu *(***Add files – hal_msp.c, DSP_lib/***)*. Číslicové zpracování souborů vyžaduje knihovnu matematických operaci, pro účely práce vyhovuje knihovna `arm_math.h` [20], tu je třeba přidat mezi **Includes.**

```
#include “arm_math.h”
```

Do nastavení kompilátoru je potřeba přidat předdefinovanou proměnou **ARM_MATH_CM4**.

4.2.1 Korelace, filtrace a FFT s využitím HAL knihoven

Pro filtraci bylo využito funkcí `arm_fir_init_f32` a `arm_fir_f32` [27]. Funkce `arm_fir_init_f3`, které jsou parametry předány koeficienty filtru a počet vstupních dat vytvoří instanci třídy `arm_fir_instance_f32` [27]. Tato instance je použita jako vstupní parametr (vedle parametrů obsahujících adresy vstupního a výstupního pole vzorků) funkce `arm_fir_f32` provádějící samotnou filtraci.

Pro Korelaci byla využita funkce `arm_correlate_f32` [28]. Funkci jsou parametry předány adresy a délky vstupních polí vzorků a adresa výstupního pole.

Pro rychlou Fourierovu transformaci bylo využito funkce `arm_cfft_f32` [15], která využívá algoritmu s různými radixy. Zpracovává několik fází FFT ve formě radix-8 spolu s jednou fází radix-2 nebo radix-4, podle potřeby. Algoritmus podporuje zpracování posloupností o délce 16, 32, 64, …, 4096 bitů a pro každou používá jinou tabulku předpočítaných twiddle konstant. Predpočítané twiddle konstanty jsou definovány v `arm_const_structs.h` [15]. Následující příkaz volá 64bodou FFT, kde `pointerSource` je ukazatel na pole komplexních hodnot vstupního signálu o velikosti 2 * délka FFT, kam budou uloženy i výsledky FFT. Argumenty `ifftFlag` a `bitrFlag` se starají o zpětnou FFT a bitově reverzní pořadí výsledku [15].

```
ifftFlag=0;
bitrFlag=0;
arm_cfft_f32(arm_cfft_sR_f32_len64, pointerSource, ifftFlag, bitrFlag);
```

4.3 Vlastní implementace FFT v jazyce C

V jazyce byly implementovány algoritmy radix-2 a radix-4. Pro oba byly v rámci optimalizací předkalkulovány hodnoty twiddle konstant (zde se nabízí možnost využít tabulky s hodnotami twiddle faktorů z `arm_common_tables.h`).

```
int i;
 int numsamp = 8; //počet vzorků
 struct complex *W;

      for (i=0; i<numsamp; i++)
      {
       W[i].r=cos(i*2.0f*PI/(float32_t)N);
```

```
       W[i].i=-sin(i*2.0f*PI/(float32_t)N);
      }
```

Samotné funkci pro rychlou Fourierovu transformaci jsou potom parametry předány ukazatel na pole vstupních (a výstupních) hodnot, délka FFT *N* a ukazatel na pole twiddle konstant. Pro oba algoritmy byla zvolena decimace ve frekvenční oblasti (DIF), ale výpočetní náročnost je u DIT stejná. Celý algoritmus je rekurzivní (na bázi vnoření).

Pro implementaci algoritmu DIF radix-2 byly využity rovnice 2.9 - 2.13 z kapitoly 2. Funkce `void radix2()` prochází v cyklu pole vstupních hodnot, provádí komplexní součty a součiny s twiddle konstantami podle podle obrázku 2.4 v kapitole 2 a pro provedení *N/2* cyklů volá 2x rekurzivně sebe na pole vstupů o poloviční délce viz obrázek 2.5 z kapitoly 2.

```
      void radix2(struct complex *data, int N, struct complex *W){
          int    n, N2;
          struct complex butterfly[2];
          N2=N/2;

          for (n=0; n<N2; n++)//2-point DFT
          {
            . . . //2bodá DFT s ukládáním výsledků
          }
                if (N2!=1)
            {
            radix2(&data[0], N2, W);
            radix2(&data[N2],N2, W);
            }
      }
```

Algoritmus DIF radix-4 využívá stejného principu jako radix-2 DIF, avšak provádí pouze N/4 cyklů a rekurzivně volá funkci na pole o čtvrtinové délce. Implementace motýlku proběhla podle rovnic 2.19 - 2.25 odvozených v kapitole 2 a podle obrázku 2.6 v téže kapitole.

```
void radix4(struct complex *data, int N, struct complex *W){
      int    n, N2;
      struct complex butterfly[4];
       N2=N/4;
      for (n=0; n<N2; n++) {
            //4bodá DFT s ukládáním výsledků in-place
      . . .
      }
      if (N2!=1){
            radix4(&data[0], N2, W);
            radix4(&data[N2], N2, W);
            radix4(&data[N2*2], N2, W);
            radix4(&data[N2*3], N2, W);
      }
}
```

4.3.1 Syntax v jazyce symbolických adres

Asembler, součást vývojového prostředí, parsuje kód v jazyce symbolických adres a vytváří objektový kód. Jedná se o soubor obsahující jak strojový kód, tak metadata. Každá sekce objektového kódu obsahuje informace o velikosti sekcí, jejich pozici v objektovém

souboru a v paměti. Tyto sekce je možné označovat direktivou `AREA`. Začátek, spuštění a konec programu označují direktivy `ENTRY`, `start` a `END`.

Každý řádek v JSA následuje základní formát.

```
{symbol {instrukce|directiva|pseudo-instrukce} {;komentář}
```

Všechny 3 sekce jsou nepovinné. Symbol označuje použitou instrukci nebo pseudo-instrukci, v případě direktivy zastupuje proměnnou nebo konstantu. Direktivy jsou použity pro předávání důležitých informací asembleru. Symbol musí začínat v prvním sloupci, nesmí obsahovat mezery nebo tabulátor, pokud není ohraničen znakem „|“. Instrukce, pseudo-instrukce ani direktivy nesmí být předcházeny mezerou nebo tabulátorem. Poslední částí je komentář začínající středníkem a končící koncem řádku. Asembler ignoruje prázdné řádky a komentáře, takže mohou být použity pro přehlednost kódu. Pro zvýšení čitelnosti kódu může být řádek zalomen použitím zpětného lomítka a pokračováním bez mezery za \ na dalším řádku. Limit znaků na jeden řádek je 4095.

Zdrojový kód v JSA může obsahovat čísla, řetězce (`“test“`) a jednoduché znaky (`'x'`). Čísla lze vyjadřovat v dekadické soustavě, šestnáctkové soustavě (s předponou `0x`) nebo libovolné číselné soustavě n od 2 do 9 (s předponou `n_`). Dále lze vyjádřit čísla s plovoucí řádovou čárkou (`25.1`), logické hodnoty `{true}` a `{false}`.

4.3.2 Použití „inline asembleru“

Kompilátor ARM poskytuje možnost použití inline asembleru [16], který umožňuje nahrazení části kódu v C nebo C++ jazykem symbolických adres. Inline asembler podporuje jak většinu ARM, tak Thumb instrukcí. Omezení se týká Thumb instrukcí na procesorech vybavených instrukční sadou Thumb 2, některých nových instrukcí koprocesoru pro práci s plovoucí řádovou čárkou. Pro použití inline asembleru platí několik pravidel, které jsou shrnuty v následujících odstavcích.

Inline asembler je volán příkazem `__asm` a následován seznamem instrukcí pro asembler ohraničeným složenými nebo jednoduchými závorkami.

```
__asm("instrukce[;instrukce]");
__asm{instrukce[;instrukce}
```

Instrukce mohou být odděleny koncem řádku, nemohou obsahovat komentáře ve formátu pro asembler, ale v bloku kódu pro inline asembler může být použit komentář v syntaxu pro C nebo C++ (vyjma řádků, kde je použito víc instrukcí).

```
__asm
    {
       STMFD sp!, {r0}    // ulož r0 - chyba: čtení před zápisem
       ADD r0, x, 1
    }
```

Po sobě jdoucí instrukce na jednom řádku musí být odděleny středníkem, pokud se příkaz nevejde na jeden řádek, musí být rozdělen zpětným lomítkem. Se jmény registrů je zacházeno jako s názvy proměnných v jazyce C, nemusí nutně odkazovat na stejnojmenné registry. Pokud není registr deklarován v předcházejícím kódu v C, hlásí kompilátor varování. V inline asembleru nesmí být prováděny operace ukládání a načítání

z registru, tyto operace doplní kompilátor. Inline asembler neumožňuje přímý přístup k fyzickým registrům, pouze k virtuálním registrům. Pokud je z virtuálního registru čteno dřív, než zapisováno, kompilátor vrátí chybu [16]. Při potřebě čtení z registrů `pc`, `lr` nebo `sp` je nutné použít intrinzitní (kompilátorem obstarávané) registry `__current_pc`, `__current_sp` a `__return_address`. ARM také zásadně nedoporučuje měnit režimy procesoru nebo koprocesorů přímo v inline asembleru, ale použít intrinzitní funkce.

4.4 Vlastní implementace FFT v jazyce symbolických adres

Implementace v JSA vychází z implementace v jazyce C. V jazyce C je ponechán kód funkce `main(void)`, veškerá obsluha periferií, inicializace rozhraní Flash pomocí `HAL_Init()`, stejně tak generování signálu a předpočet twiddle konstant. Pouze časově nejnáročnější operace byly přepsány v JSA – tedy výpočet motýlků. Zdrojové kódy funkcí jsou obsaženy v příloze.

4.4.1 Radix-2

Pro implementaci radix-2 algoritmu byly stejně jako v případě implementace v jazyce C využity rovnice 2.9 - 2.13 z kapitoly 2 odvozené pro decimaci ve frekvenční oblasti podle obrázku 2.4 ze stejné kapitoly.

Při použití inline asembleru je nutné před použitím virtuální registr deklarovat:

```
float32_t s0=data[n].r;
float32_t s1=data[n+N2].r;
float32_t s4
__asm
{.  .  .}
```

V inline asembly byly použity instrukce pro práci s vektorovou floating-point jednotkou (VFP) [16]:

```
VADD.F32 s4, s0, s1
VSUB.F32 s5, s2, s3
VMUL.F32 s0,s6, s8
```

Instrukce `VADD` slouží pro sčítání v plovoucí řádové čárce, v tomto případě v single precision. Použité registry S jsou u jádra Cortex-M4 součástí FPU a umožňují uložení 32bitových proměnných typu float. Instrukce `VSUB` slouží pro odčítání ve FP, instrukce `VMUL` pro násobení ve FP. Všechny instrukce sdílí stejnou syntax, kdy za názvem instrukce následuje podmínka .F32 nebo `.F64` značící zvolenou přesnost (single/double precision). Následuje název `Sn` registru pro uložení výsledku v SP a názvy operandů. Instrukce `VSUB` odečte hodnotu v druhém operandu od hodnoty v prvním operandu [16].

4.4.2 Radix-4

Pro implementaci algoritmu radix-4 s decimací ve frekvenční oblasti byly použity rovnice 2.19 – 2.26 odvozené v kapitole 2 této práce. Zvolen byl opět programovací přístup pomocí rekurzivního volání funkce, stejně jako při implementaci v jazyce C.

Podobně jako u algoritmu radix-2 DIF (4.4.1) byly využity instrukce `VADD`, `VSUB` a `VMUL` [16] pro práci s vektorovou FPU. V tomto případě je již lépe využíván potenciál procesoru, co se týče počtu SP (single precision floating point) registrů – využito 24 z 32 možných. Pro přehlednost kódu je vždy u bloku příkazů uveden v komentáři kód v jazyce C, který JSA reprezentuje.

```
//butterfly[0].r = (data[n].r + data[N2 + n].r + data[2*N2+n].r +
//data[3*N2+n].r);
VADD.F32 s21, s0, s1
VADD.F32 s20, s2, s3
VADD.F32 s14, s21, s20
```

5 VÝSLEDKY

Tato kapitola shrnuje výsledky měření analýzy signálů s využitím knihoven třetí strany, měření výkonu zpracování FFT v C a v JSA a porovnání obou technik. Měření rychlosti zpracování probíhalo s využitím 32bitového nezávislého čítače `TIM5` s frekvencí inkrementace 1MHz, tedy každou µs. Takt procesoru byl u všech měření 168MHz. V měření je zahrnuta pouze výpočetní část programu. Části kódu, které se starají o inicializaci periferií, MCU, nebo příprava polí se vzorky, nejsou v měření zahrnuty. Čítač TIM5 je spuštěn a jeho hodnota odečtena v momentě volání funkce pro zpracování signálů. Po jejím dokončení je znovu odečtena hodnota čítače a vyčíslen jejich rozdíl.

5.1 Porovnání kompilátorů při filtraci a korelaci

Srovnání rychlosti výpočtu filtrace a korelace v single precision s FPU a bez ní probíhalo s využitím knihovny arm_math.h a DSP knihoven od STM (pouze převzato od ARM) na kompilátorech GCC ARM Embedded 4.9 a ARM Keil V5.06. ARM Keil je lépe optimalizovaný a byl využit pouze pro porovnání rychlosti zpracování bez FPU, které i přes to probíhalo zhruba 2.5x pomaleji. Vliv FPU je tedy značný ve prospěch rychlosti zpracování operací v plovoucí řádové čárce, a to i přes 15-20% nevýhodu v podobě pomalejšího GCC kompilátoru. Pro použití v oboru zpracování signálů v single precision floating point (32bitů) v reálném čase je FPU takřka nutností, avšak při jeho občasném využití nebo nevyužívání může být vhodné ho vypnout a občasné výpočty v single (případně double) precision provést přes emulaci pomocí celočíselných operací, čímž se dá snížit spotřeba proudu.

Optimalizace kompilátoru, u GCC nejvyšší míra optimalizací pro rychlost (O3), mají smysl vždy. U použitých algoritmů měly optimalizace GCC vliv na výkon čtyř až pětinásobný. Tabulka 5.1 nabízí porovnání doby výpočtu filtrace signálu pomocí FIR a korelace s různými kombinacemi počtu vzorků, kompilátorů a jejich nastavení.

Tabulka 5.1 Porovnání rychlosti výpočtu algoritmů pro analýzu signálů.

Audio signál se spektrem 0-22khz			
Operace	**Počet vzorků [-]**	**Doba výpočtu [µs]**	**kompilátor**
FIR (O 3)	320	250	GCC
FIR (FPU)	320	1272	GCC
FIR (CPU O3)	320	5235	ARM Keil v5.06
FIR (FPU O3)	3200	2490	GCC
FIR (FPU)	3200	12705	GCC
korelace (FPU O3)	320	4676	GCC
korelace (FPU)	320	21141	GCC
korelace (CPU O3)	320	51460	ARM Keil v5.06
korelace (O3)	3200	459243	GCC

korelace (FPU)	3200	2083765	GCC
korelace (s optimalizacemi)	3200	5300020	ARM Keil v5.06

5.2 Porovnání optimalizací kompilátorů u FFT mixed-radix

Srovnání rychlosti výpočtu mixed-radix FFT v single precision floating point probíhalo s využitím knihovny arm_math.h a DSP knihoven od STM na kompilátorech GCC verze 5.4 a ARM Keil verze 5.06.

U kompilátoru GCC (tabulka 5.2) je vidět značný vliv optimalizací, nárůst rychlosti po zapnutí dostupných voleb v nastavení kompilátoru (obrázek 3.6 z kapitoly 3) je pro větší množství vzorků ($N>1024$) je více než pětinásobný. Pro menší počet vzorků ($N<256$) je nárůst rychlosti 3,5 – 4 násobný. Další výhodou je snížení velikosti kódu o 20%, což je zvlášť u zařízení s malou velikostí RAM výhodné. Nevýhodou optimalizací je několikanásobně delší doba kompilace, než v případě výchozího nastavení a méně přehledný strojový kód v debugeru (disassembly).

Tabulka 5.2 Porovnání vlivu optimalizací GCC na rychlost výpočtu FFT mixed-radix.

Mixed-radix, kompilátor GNU GCC ARM Bare-metal 5.4		
Počet vzorků [-]	**FPU O-0**	**FPU O-3**
	Doba výpočtu [µs]	
32	49	14
64	122	28
128	299	68
256	672	155
512	1477	301
1024	3414	706
2048	7401	1564
4096	15866	3115

Kompilátor ARM Keil je již v základu optimalizován velmi dobře. Další možnosti optimalizací nabízí snížení velikosti kódu o cca 20%, ale za cenu jeho pomalejšího běhu. Pokles výkonnosti je ovšem pouze 11% a doba zpracování je stále kratší, než při použití GCC. Tabulka 5.2 nabízí přehled o dobách výpočtů mixed-radix FFT v závislosti na počtu vstupních vzorků. Kompilátor ARM Keil nabízí možnost využít (spolu s HAL knihovnami) i výpočtu v single precision floating point pomocí CPU. Dosažená rychlost je 12 – 15krát nižší, než při použití FPU.

Tabulka 5.3 Porovnání vlivu FPU a optimalizací kompilátoru na rychlost výpočtu FFT mixed-radix.

Mixed-radix, kompilátor ARM Keil v5.06

Počet vzorků [-]	CPU O-0	CPU O-3	FPU O-0	FPU O-3
	Doba výpočtu [µs]			
32	143	134	12	13
64	313	291	23	24
128	781	628	59	63
256	1890	1882	131	147
512	5325	5170	263	289
1024	9202	9139	633	692
2048	>32kB	>32kB	1387	1550

5.3 Porovnání implementací FFT v jazyce C a JSA

Vlastní implementace FFT použitím algoritmu radix-2 s decimací v časové oblasti byla velice pomalá, je proto nutné předpočítat hodnoty twiddle konstant. Tím se dramaticky zrychlil běh programu (40-55x). Výkonnost takového řešení byla ale pořád zhruba 2x – 4x nižší, než algoritmus mixed-radix v matematických knihovnách od STM. Přepsáním klíčových částí (výpočet motýlků) algoritmu do jazyka symbolických adres je možné u většího počtu vzorků dosáhnout až o 33% kratších výpočetních časů. Tabulka 5.4 ukazuje srovnání doby běhu radix-2 v C s a bez počítání twiddle konstant a dobu běhu při použití JSA. Z hlediska využití paměti není mezi kódy v C a JSA výrazný rozdíl, většina okupované paměti je využita vzorky a twiddle konstantami. Co se týče počtu využitých registrů, kód přeložený z jazyka C si vystačí s třetinovým množstvím registrů, avšak za cenu nepřehlednosti kódu v disassembly. V případě nutnosti (nedostatku registrů) je možné kód v JSM upravit tak, aby registrů využíval méně (například ho rozdělit do více bloků) beze ztráty výkonu.

Kompilátor ARM Keil byl ponechán v základním nastavení – míra optimalizací O0.

Tabulka 5.4 Porovnání rychlosti výpočtu FFT v C a v JSA pro radix-2.

Radix-2, kompilátor ARM Keil v5.06			
Počet vzorků [-]	**Doba výpočtu implementace v C [µs]**	**Doba výpočtu implementace v C s předkalkulovaným W [µs]**	**Doba výpočtu implementace v JSA (inline assembler) s předkalkulovaným W [µs]**
16	963	28	19
32	2718	68	49
64	7065	160	109
128	17437	368	251
256	41535	836	567
512	96439	1865	1265

1024	219672	4121	2792
2048	492977	9023	6108
4096	1093280	19600	13265

Radix-4 by měl být podle teorie [] efektivnější (pro velké počty vzorků), než radix-2 []. Jak ukazuje tabulka 5.5, doba běhu programu s použitím radix-4 DIF je zhruba o 30% kratší, než v případě radix-2. Přepsáním klíčové části algoritmu do JSA se výkon opět zvedne podobně jako v předchozím případě, tedy až o 33%. Na počet registrů je algoritmus radix-4 cca 4x náročnější než radix-2, avšak ani u něj nebylo využito potenciálu Cortex-M4 (využito maximálně 24 single precision registrů z 32 možných).

Tabulka 5.5 Porovnání rychlosti výpočtu FFT v C a v JSA pro radix-4.

Radix-4, kompilátor ARM Keil v5.06			
Počet vzorků [-]	**Doba výpočtu v C [µs]**	**Doba výpočtu v C s předkalkulovaným W [µs]**	**Doba výpočtu v JSA (inline assembler) s předkalkulovaným W [µs]**
16	558	19	14
64	4540	113	82
256	28494	600	431
1024	154385	2988	2134
4096	779604	14311	10179

6 ZÁVĚR

Cílem bakalářské práce bylo seznámit čtenáře s problematikou ARM procesorů, jejich architekturou a rozdělením podle instrukčních sad. Blíže představen byl mikroprocesor ARM Cortex-M4 osazen na vývojářské desce STM32F4Discovery od společnosti STMicroelectronics. Práce připomněla základní principy analýzy signálů, osvětlila rozdíly mezi FT, DFT a jejím zefektivněním. V práci jsou porovnány programové přístupy k analýze FFT v jazyce C a JSA. Zájemcům o programování vlastního zařízení v jazyce C a JSA nabídla návod na nastavení desky, vývojového studia a kompilátoru.

Výsledky porovnání rychlosti zpracování signálů s využitím matematických knihoven ukazují výrazný vliv optimalizací kompilátoru GCC a jednotky FPU na výkon zpracování signálů. Při optimalizování kódů byla použita nejvyšší úroveň optimalizací GCC (O3), což mělo za následek takřka pětinásobné zrychlení. To značí velmi dobře propracovaný kompilátor pro jedno-vláknové programy. Kompilátor ARM Keil je již v základu optimalizovaný dobře, kód s ním přeložen dosahoval v základním nastavení o 10-15% vyšších rychlostí.

Vypnutí FPU má za následek drastické snížení výkonu v plovoucí řádové čárce. Co se týče rozdílů mezi kompilátory, GCC je k dispozici zdarma bez omezení, zatímco kompilátor od Keilu je zdarma pouze při velikosti kódu do 32kB, což bylo částečně omezující už pro tuto práci. Z tohoto důvodu je pro osobní použití lepší použít ARM GCC. Jeho nevýhodou je delší doba strávená kompilací proti kompilátoru ARM Keil a krkolomné nastavování pro vybraný procesor.

Z porovnání programovacího přístupu v jazyce C a JSA plyne zjištění, že z technického hlediska má použití JSA stále smysl. I když jsou kompilátory a jejich optimalizace stále účinnější a propracovanější, obzvlášť pro jedno-vláknové aplikace, při provádění FFT byl přeložený kód z jazyka C o 33% pomalejší než kód psaný v JSA. Z ekonomického hlediska převažují nad JSA výhody psaní kódu v C. Krom nutnosti obeznámit se detailně s programovaným zařízením neumožňuje kód v JSA pohodlný debugging, a tím mohou uniknout pozornosti případné chyby v kódu.

LITERATURA

[1] OPPENHEIM, Alan V, Ronald W SCHAFER a Alan V OPPENHEIM (ed.). Discrete-time signal processing. Englewood Cliffs: Prentice-Hall, 1989, xv, 879 s. Prentice Hall signal processing series. ISBN 0-13-216771-9.

[2] JAN, Jiří. Číslicová filtrace, analýza a restaurace signálu. Vyd. 2. Brno: VUTIUM, 2002, 427 s. ISBN 80-214-1558-4.

[3] ARM HOLDINGS. Cortex-M4 Devices: Generic User Guide [online]. 2010 [cit. 11. 5. 2014]. Dostupné z: http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf.

[4] CHEN, C., NOVICK, G., SHIMANO, K. RISC Architecture. *Stanford University's Computer Science Department* [online]. 2000 [cit. 2015-12-11]. Dostupné z: http://cs.stanford.edu/people/eroberts/courses/soco/projects/2000-01/risc/.

[5] BEREZINSKI, J. RISC / CISC. *Northern Ilinois University: Department of Computer Science* [online]. 2006, 2015-03-05 [cit. 2015-12-11]. Dostupné z: http://faculty.cs.niu.edu/~berezin/463/lec/05/risc01.html.

[6] ARM. *Cortex-M Series* [online]. 2015 [cit. 2015-12-11]. Dostupné z: http://www.arm.com/products/processors/cortex-m/.

[7] STMICROELECTRONICS. STM32F407 Reference Manual [online]. Rev. 11. 2015, 1731 p. [cit. 2015-12-12]. Dostupné z: http://www.st.com/web/en/resource/technical/document/reference_manual/DM00031020.pdf

[8] NULL, Linda a Julia LOBUR. The essentials of computer organization and architecture [online]. 2nd ed. Sudbury, Mass.: Jones and Bartlett Publishers, 2006, xxxii, 799 p. [cit. 2015-12-11]. ISBN 978-0763737696.

[9] STMICROELECTRONICS. STM32F4 Discovery User Manual [online]. Rev. 4. 2015 [cit. 2015-12-11]. Dostupné z: http://www.st.com/st-web-ui/static/active/cn/resource/technical/document/user_manual/DM00039084.pdf.

[10] STMICROELECTRONICS. Description of STM32F4xx HAL drivers [online]. Rev. 3. 2015 [cit. 2015-12-11]. Dostupné z: http://www.st.com/st-web-ui/static/active/jp/resource/technical/document/user_manual/DM00105879.pdf.

[11] WEISSTEIN, Eric. "Landau Symbols." [online]. [cit. 2015-12-11]. Dostupné z: Wolfram Web Resource. http://mathworld.wolfram.com/LandauSymbols.html.

[12] STMICROELECTRONICS. STM32F4xx Technical Training [online]. 2013 [cit. 2015-12-16]. Dostupné z: http://web.eece.maine.edu/~hummels/classes/ece486/docs/ STM32F4-Technical-Training.pdf.

[13] ARM HOLDINGS. ARM Q4/FY2016 Roadshow Slides [online]. 2017 [cit. 2017-05-20]. Dostupné z: http://www.arm.com/-/media/arm-com/company/Investors/Quarterly%20Results%20-%20PDFs/ARM_SB_Q4_2016_Roadshow_Slides_FINAL.pdf?la=en.

[14] VONIČKA, M. Využití procesorů ARM pro zpracování signálů. Brno: Vysoké učení technické v Brně, Fakulta elektroniky a komunikačních technologií. Ústav radioelektroniky, 2015. 15 s., 1 s. příloh. Semestrální projekt. Vedoucí práce: ing. Roman Mego.

[15] ARMKeil. Complex FFT Functions Slides [online]. 2017 [cit. 2017-02-08]. Dostupné z: https://www.keil.com/pack/doc/CMSIS/DSP/html/group__ComplexFFT.html#ga68cdacd2267a2967955e40e6b7ec1229

[16] ARMKeil. Inline assembler and register access in C and C++ code [online]. 2017 [cit. 2017-05-21]. Dostupné z: http://www.keil.com/support/man/docs/armcc/armcc_chr1359124249383.htm

[17] STMICROELECTRONICS. STM32F4 and STM32L4 Series Cortex-M4 programming manual[online]. 2017 [cit. 2016-05-21]. Dostupné z: http://www.st.com/content/ccc/resource/technical/document/programming_manual/6c/3a/cb/e7/e4/ea/44/9b/DM00046982.pdf/files/DM00046982.pdf/jcr:content/translations/en.DM00046982.pdf

[18] A Premier Farnell Company. Discovery Kit for STM32 F4 Series with STM32F407VG MCU [online]. 2017 [cit. 2012-07-23]. Dostupné z: https://www.element14.com/community/docs/DOC-48699/l/discovery-kit-for-stm32-f4-series-with-stm32f407vg-mcu

[19] ARMKeil. Assembler User Guide [online]. 2012 [cit. 2016-07-23]. Dostupné z: http://www.keil.com/ support/man/docs/armasm/armasm_pge1423738743329.htm

[20] ARMKeil. CMSIS DSP Software Library [online]. 2015 [cit. 2017-04-20]. Dostupné z: http://www.keil.com/pack/doc/CMSIS/DSP/html/index.html

[21] STMICROELECTRONICS. STM32Cube Embedded Software [online]. 2017 [cit. 2017-04-20]. Dostupné z: http://www.st.com/en/embedded-software/stm32cube-embedded-software.html?querycriteria=productId=LN1897

[22] ARMKeil. Getting Started with MDK [online]. 2017 [cit. 2017-05-20]. Dostupné z: https://armkeil.blob.core.windows.net/product/mdk5-getting-started.pdf

[23] STMICROELECTRONICS. STSW-LINK009 [online]. 2017 [cit. 2017-04-02]. Dostupné z: http://www.st.com/content/st_com/en/products/embedded-software/development-tool-software/stsw-link009.html

[24] GNU GCC. GCC online documentation [online]. 2017 [cit. 2017-05-10]. Dostupné z: https://gcc.gnu.org/onlinedocs/

[25] OMAR, N. a SARMA, T.C.. Design of 64point Fast Fourier Transform by Using Radix-4 Implementation [online]. 2014 [cit. 2017-04-05]. Dostupné z: https://www.arcjournals.org/pdfs/ijirec/v1-i7/4.pdf

[26] JONES, Douglas L. Radix-4 FFT Algorithms [online]. 2006 [cit. 2017-05-12]. Dostupné z: https://pdfs.semanticscholar.org/fec9/b9727e74d8f487997705c85c50a8e9876583.pdf

[27] ARMKeil. Finite Impulse Response (FIR) Filters [online]. 2017 [cit. 2017-05-15]. Dostupné z: https://www.keil.com/pack/doc/CMSIS/DSP/html/group__FIR.html#gae8fb334ea67eb6ecbd31824ddc14cd6a

[28] ARMKeil, Correlation [online]. 2017 [cit. 2017-05-15]. Dostupné z: https://www.keil.com/pack/doc/CMSIS/DSP/html/group__Corr.html#ga22021e4222773f01e9960358a531cfb8

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

DFT	Discrete Fourier transform – Fourierova transformace pro diskrétní signály
FFT	Fast Fourier transform – rychá Fourierova transformace (DFT)
DSP	Digital signal processing – číslicové zpracování signálů.
ARM	Acorn (později Advanced) RISC Machines – procesory s RISC architekturou ovládající svět mobilních a embedded zařizení.
RISC	Reduced instruction set computing – způsob návrhu procesorů s důrazem na zjednodušenou instrukční sadu.
DIF	Decimation in frequency – typ FFT založen na rozkladu výstupní posloupnosti.
DIT	Decimation in time – typ FFT založen na metodě rozkladu vstupních dat.
C	Imperativní programovací jazyk původně vyvinut mezi lety 1969 a 1973.
STM	STMicroelectronics – firma vyrábějící mikrokontroléry založené především na procesorech od firmy ARM Holdings.
IDE	Integrated development environment – vývojové prostředí pro programování a často i kompilaci a debugování.
GCC	GNU Compiler Collection – kompilátor licencovaný pod GNU GPL.
HAL	Hardware Abstraction Layer – knihovny STM pro práci s jejich mikrokontroléry vyznačující se vysokou mírou abstrakce.
FPU	Floating-point unit – logická jednotka pro operace v plovoucí řádové čárce.
DAC	Digital-to-analog converter – DA převodník.
SWD	Serial Wire Debug – rozhraní pro programování a debugování mikrokontrolérů s možností přímého přístupu k paměti.