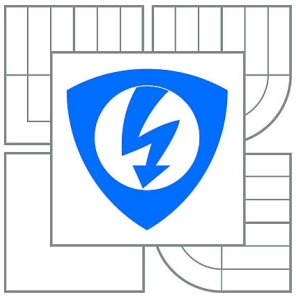
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

PODPŮRNÉ ALGORITMY PRO ŘÍZENÍ ELEKTRICKÝCH MOTORŮ

SUPPORTING ALGORITHMS FOR ELECTRICAL MOTOR CONTROL

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

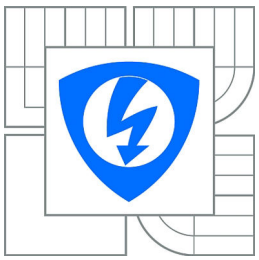
AUTOR PRÁCE — Bc. MARTIN ŘEZÁČ

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. PETR BLAHA, Ph.D.

SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student:	Bc. Martin Řezáč	***ID:***	134600
Ročník:	2	***Akademický rok:***	2014/2015

NÁZEV TÉMATU:

Podpůrné algoritmy pro řízení elektrických motorů

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s problematikou řízení elektrických motorů.
2. Proveďte analýzu kvality měření signálů ze snímače TLE5009. Sběr dat proveďte pomocí systému CompactRIO a LabView. Data analyzujte pomocí programu Matlab Simulink.
3. Seznamte se s procesorem TC275 CA od firmy Infineon a jeho periferiemi.
4. Implementujte ovladač pro periferii SADC, který umožní současné navzorkování několika (minimálně 3) vstupních signálů.
5. Implementujte ovladač pro generování centrované PWM pro třífázový a vícefázový střídač.
6. Implementujte ovladač pro připojení snímače úhlového natočení.
7. Ovladače otestujte na vývojovém kitu a optimalizujte jejich výpočetní nároky.

DOPORUČENÁ LITERATURA:

[1] Sul, S.K.: Control of Electric Machine Drive Systems. February 2011, Wiley-IEEE Press. ISBN: 978-0-470-59079-9.

Firemní literatura firmy Infineon.

Další dle doporučení vedoucího a konzultanta.

Termín zadání:	9.2.2015	***Termín odevzdání:***	18.5.2015

Vedoucí práce: doc. Ing. Petr Blaha, Ph.D.
Konzultanti diplomové práce:

doc. Ing. Václav Jirsík, CSc.
Předseda oborové rady

ABSTRAKT

Tato práce je zaměřena na vytvoření podpůrných ovladačů a algoritmů pro řízení elektrických motorů s využitím procesoru AURIX TriCore TC275 CA. První část práce je věnována popisu procesoru a jeho vybraným periferiím. Následně jsou realizovány ovladače pro jednotlivé periferie procesoru, kterými jsou A/D převodník, časovač pro tvorbu PWM signálu a druhý časovač pro zpracování signálů z inkrementálního snímače. Všechny ovladače jsou otestovány na vývojové kitu, který je osazen procesorem TC275 CA.
Druhá část práce se zabývá analýzou kvality měření z GMR snímače TLE5009. Pro snímání dat je vytvořen speciální přípravek, který je osazený GMR snímačem, inkrementálním snímačem v pozici etalonu a DC motorem. Data jsou získávána pomocí programu LabView. Následně je provedena analýza snímaných dat, jejich kompenzace a nakonec porovnání s naměřenými daty z inkrementálního snímače.

KLÍČOVÁ SLOVA

TriCore, mikrokontrolér TC275 CA, VADC, synchronizace, PWM, třífázový motor, devítifázový motor, GMR snímač natočení, dynamická kompenzace

ABSTRACT

This thesis is focused on a creating supporting drivers and algorithms for electrical motor control using CPU TriCore TC275 CA. The first part is devoted to processor description and selected peripherals, which are A/D converter, a timer for creating the PWM signals and the second timer for processing singnals from encoder. All drivers are tested on an aplication kit, which is equipped with TC275 CA processor.
The second part analyzes the measurement quality of GMR sensor TLE5009. Special testbench was prepared for sin cos data capturing and for their comparison with precise encoder position measurement. It was composed from DC motor having both sensor types on commons shaft. Data are acquired using LabView. Subsequently, it analyzes the sensor data, their compensation and subsequent comparison with measured data from the encoder.

KEYWORDS

TriCore, microcontroller TC275 CA, VADC, synchronization, PWM, 3 phase motor, 9 phase motor, GMR angle sensor, dynamic compenzation

ŘEZÁČ, Martin *Podpůrné algoritmy pro řízení elektrických motorů*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřící techniky, 2015. 70 s. Vedoucí práce byl doc. Ing. Petr Blaha, PhD.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Podpůrné algoritmy pro řízení elektrických motorů“ jsem vypracoval samostatně pod vedením vedoucího diplomové 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 diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové 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 autorského 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.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Ing. Petru Blahovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Tato diplomová práce se v první části zabývá tvorbou podpůrných ovladačů a algoritmů pro řízení elektrických motorů, druhá část je zaměřena na analýzu dat z GMR senzoru natočení.

V teoretické části práce je popsán mikrokontrolér Infineon TC275 CA a vývojový kit. Jedná se o tříjádrový mikrokontrolér, kdy 1 jádro je určeno pro méně výkonově náročné aplikace a zbylá 2 jádra jsou naopak výkonová. Procesor obsahuje velké množství periferií, z nichž si několik blíže popíšeme. Jedná se o periferie A/D převodníku, časovače GTM a GPT12, kdy první zmíněný časovač je určen pro generování PWM signálů a druhý pro zpracování dat z enkodéru. Dále je provedeno seznámení s vývojovým kitem, který je mikrokontrolérem TC275 CA osazen, včetně mapování vstupně/výstupních portů.

Praktické část se zabývá vytvořenými algoritmy pro jednotlivé periferie mikrokontroléru, které jsou vytvořeny v jazyce C, v programovacím prostředí TriCore Development Platform od firmy HighTec. Jednotlivé algoritmy vytváří kompaktní celek pro řízení elektrických motorů, kde jsou obsaženy všechny důležité části potřebné pro jejich správné řízení. Jsou vytvořeny algoritmy pro generování centrovaného PWM signálu, synchronizované převody vstupních signálů dle PWM pro převod proudů jednotlivých fází motoru a převody takzvaně na pozadí pro získání informace například o teplotách. Nakonec byly vytvořeny algoritmy pro zpracování dat z enkodéru pro zjištění polohy, rychlosti a směru otáčení motoru.

Druhá část práce je zaměřena na analýzu dat z GMR snímače. GMR jev je založen na změně elektrického odporu v závislosti na velikosti a směru působícího magnetického pole. Použitý snímač fungující na GMR principu má označení TLE5009, výstupem jsou dvojice průběhů ve tvaru sinu a kosinu. Pro získávání dat je snímač umístěn ve vytvořeném přípravku společně s DC motorem a enkodérem.

Získávání dat proběhlo pomocí programu LabView a přípravku CompactRio. Samotná práce s naměřenými daty probíhá v programu Matlab Simulink. Použitím statických a dynamických kompenzačních parametrů a PI regulátoru jsme provedli kompenzaci naměřených dat. Z takto upravených dat a dat z enkodéru jsme vytvořili rozdíl, který slouží pro zhodnocení kvality měření úhlového natočení pomocí GMR snímače..

1 ŘÍZENÍ ELEKTRICKÝCH MOTORŮ POMOCÍ MIKROKONTROLÉRU TC275 CA

Cílem práce bylo vytvořit několik ovladačů periferií mikrokontroléru TC275 CA za účelem jejich možného použití pro řízení elektrického motoru. Dále je potřeba využít inkrementální enkodér a pulzní šířkovou modulaci. Všechny tyto části si blíže popíšeme v následujících kapitolách.

1.1 Použité periferie

První periferií je časovač GTM, přesněji jeho TOM modul určený pro generování centrované PWM. Tato periferie byla navržena firmou Bosch a bývá součástí výkonných procesorů různých výrobců. Pro vytvoření vícefázové PWM je třeba provést zřetězení TOM modulů. Jeden modul si vybereme jako referenční a ten nám následně slouží k určování periody zbylých modulů, u kterých dle aktuálních požadavků měníme střídu.

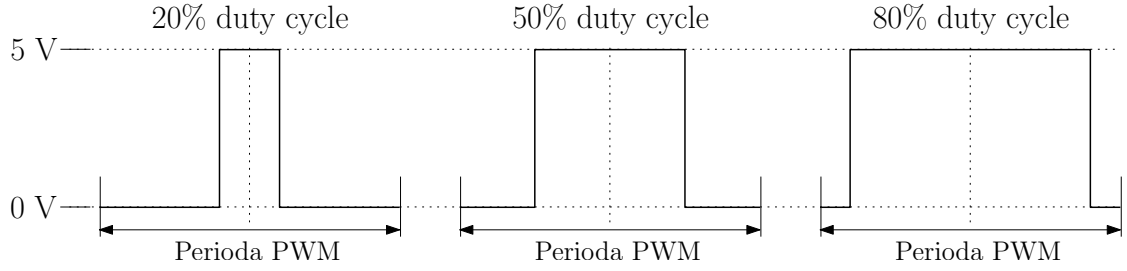
Druhou periferií je VADC, neboli A/D převodník s postupnou aproximací, u které je nutné vytvořit synchronní převod více kanálu v jeden časový okamžik, který je navíc synchronizován s průběhem PWM tak, aby byly vstupní signály převedeny v okamžiku, kdy nedochází ke spínání výkonových prvků. V jeden okamžik je však u TC275 CA možné převést maximálně 4 kanály. Z tohoto důvodu jsme rozdělili snímání signálů do dvou skupin tak, aby bylo dosaženo snímání maximálního počtu vstupních kanálů v jeden časový okamžik, či v co nejkratší době.

Poslední periferií je GPT12, jejíž 3 časovače využíváme pro zpracování signálů z inkrementálního enkodéru, neboli pro získání informace o poloze a otáčkách motoru. Měření otáček je rozděleno na dvě části (nízké a vysoké otáčky), kdy pro nízké otáčky je využit jeden časovač (T5) a pro otáčky vysoké jsou použity časovače dva (T2
a T3).

V průběhu vytváření ovladačů pro jednotlivé periferie jsme získali ovladače přímo od společnosti Infineon nazvané iLLD, neboli Infineon Low Level Driver. Ten sdružuje ovladače pro veškeré periferie kontroléru, z nichž lze poměrně jednoduše vytvořit funkční projekt. Neobsahuje však například ovladač pro vytvoření vícefázové PWM, což byl jeden z více důvodů, proč jsme se nakonec rozhodli pro dokončení námi již vytvářeného ovladače namísto použití iLLD.

1.2 Pulzně šířková modulace

Princip PWM [2] je, že změnou střídy signálu dochází ke změně hodnoty výstupního napětí. Proměnným parametrem je tedy střída (někdy udáváno jako duty cycle), která je udávána v procentech jako poměr mezi hodnotami logické 1 a 0. Existuje několik způsobů realizace PWM signálu a to centrovaná PWM nebo zarovnaná vlevo či vpravo. Na obrázku 1.1 vidíme PWM centrovanou na střed, kterou v projektu používáme, pro tři různé hodnoty střídy, a to 20 %, 50 % a 80 %. U centrované PWM je pulz v hodnotě log. 1 umístěn do středu periody signálu. Hodnota výstupního napětí (v našem případě 0 až 5 V) se může měnit dle použitých obvodů a potřeby vstupního napětí motoru.

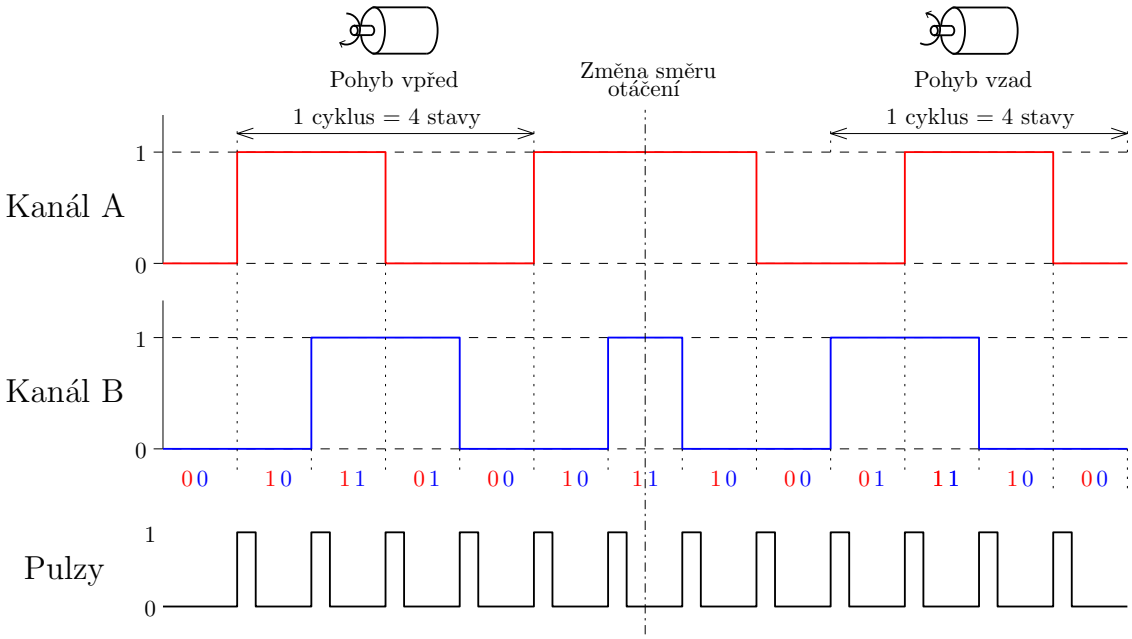


Obr. 1.1: Centrovaná PWM pro střídu 20 %, 50 % a 80 %

1.3 Měření rychlosti otáčení

Pro měření rychlosti otáčení motoru je použit inkrementální enkodér EL53A od firmy Eltra, mající rozlišení 1024 pulzů na otáčku a maximální otáčky jsou 6000 ot/min. Enkodér se k procesorům připojuje pomocí tří výstupních signálů nazvaných A, B a Z.

Na obrázku 1.2 jsou znázorněny časové průběhy signálů A a B při otáčení enkodérem jedním a následně druhým směrem. V první polovině průběhů je směr otáčení kladný (pohyb vpřed), v takovém případě je signál B zpožděný o 90° oproti signálu A, v druhém případě, to jest při záporném směru otáčení je zpoždění signálů opačné.



Obr. 1.2: Průběh výstupních signálů enkodéru

Směr otáčení je určován z posloupnosti stavů signálů A a B, které můžeme vidět na obrázku 1.2 pod průběhem kanálu B. Je vidět, že posloupnosti se liší v hodnotách stavů číslo 2 a 4, které jsou při změně směru otáčení zaměněny.

Poslední část obrázku tvoří pulzy, které jsou vytvářeny dekodérem při změně stavu signálů. U mikrokontroléru TC75 CA je dekodér implementován, tudíž jsou pulzy vytvářeny až po přivedení signálů k periferii GPT12. S těmito pulzy následně pracují časovače a čítače periferie.

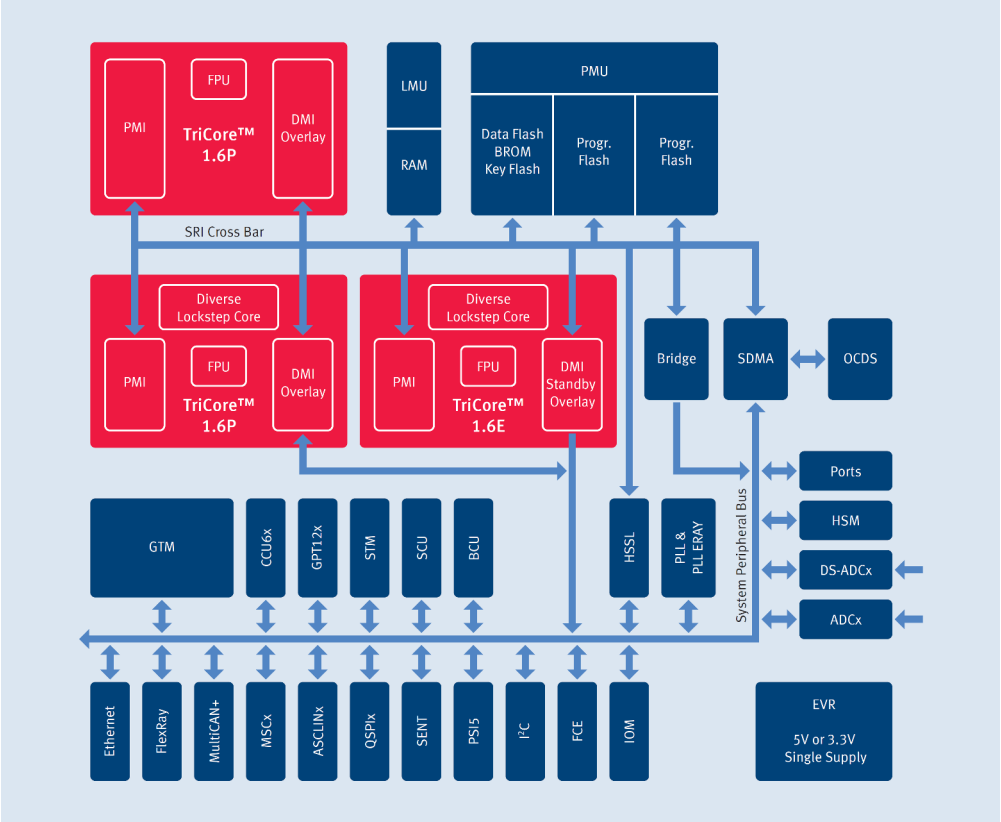
2 MIKROKONTROLÉR TC275 CA

TriCore TC275CA [4] je 32bitový DSP mikrokontrolér založený na architektuře Infineon TriCore. TC275 CA obsahuje 3 jádra v jediném pouzdře, z nichž dvě jsou označena $TC1.6P$ a jedná se o high performance cores, neboli výkonové jádra. Třetí nese označení $TC1.6E$ jako efficiency core, neboli jádro efektivní. Kontrolér pracuje s kombinovanou sadou CISC, RISC a DSP funkcí. Všechny jádra obsahují 32 bitovou FPU jednotku.

2.1 Struktura mikrokontroléru TC275 CA

Základní strukturu mikrokontroléru můžeme vidět na obrázku 2.1. Skládá se z velkého množství periferií, které jsou propojeny dvěma sběrnicemi nazvanými SPB (System Peripheral Bus) a SRI (Local Memory Bus). Pro naše použití jsou zajímavé následující periferie:

- GTM - Generic Timer Module
- GPIO - General Purpose I/O Ports and Peripheral I/O Lines
- VADC - Versatile Analog to Digital Converter

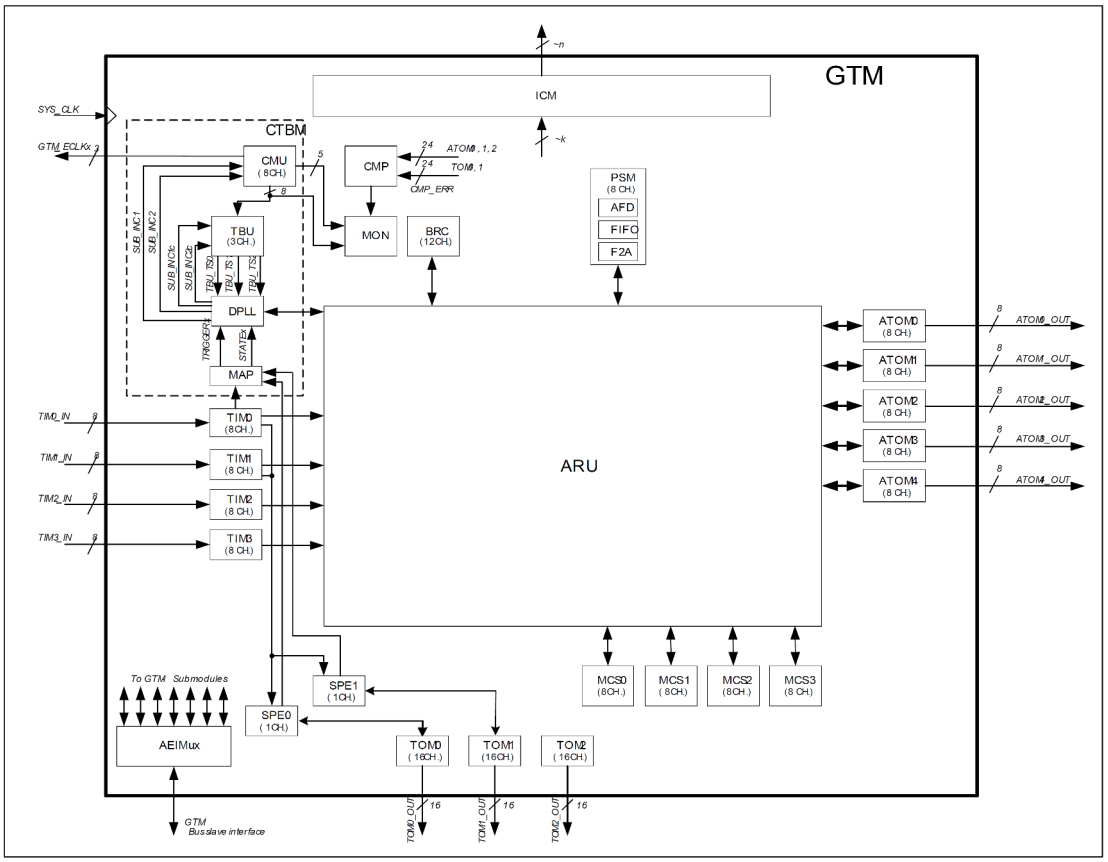


Obr. 2.1: Struktura mikrokontroléru TC275 CA, převzato z [10]

2.2 Periferie GTM

Generování, či zpracování časově závislých signálů je zajištěno periferií obecného časovače, ve zkratce GTM. Ta je složena z modulů a submodulů plnící jednotlivé funkce časovače. Těmito funkcemi rozumíme generování výstupních signálů (např. PWM), či zpracovávání signálů vstupních.

GTM periferie byla navržena tak, aby odlehčila CPU od častého volání a vykonávání přerušení. Většina úloh této periferie je schopna po prvotním nastavení pomocí CPU běžet nezávisle a paralelně k právě vykonávanému programu. Některé z vykonávaných úloh přesto potřebují CPU využívat, GTM však byla navržena právě za účelem počet těchto situací snížit na minimum.



Obr. 2.2: Blokové schéma periferie GTM, převzato z [4]

Základní struktura periferie je zobrazena na obrázku 2.2, ze kterého je patrné, že periferie obsahuje jeden vstupní modul označený TIM a dva výstupní moduly TOM a ATOM. Další významnou částí periferie je jednotka ARU, ke které je připojena většina modulů a submodulů periferie. ARU slouží ke směrování dat mezi připojenými moduly, při využití Round-robin plánování, díky čemuž je zaručeno

deterministické chování jednotky. Směrovaná data mají délku 53 bitů a lze je logicky rozdělit do 3 částí. V bitech 0 - 23 ($Data\,0$) a 24 - 47 ($Data\,1$) jsou uloženy data operačních registrů ARU, které mohou reprezentovat například střídu a periodu měřeného vstupního signálu, nebo charakteristiky generovaného PWM signálu na výstupu. Poslední částí je ACB, neboli ARU control bit, který může mít různý význam dle použitých modulů a může jím být například řídící informace.

K jednotce ARU je připojen vstupní modul TIM a jeden z výstupních modul (ATOM), který slouží ke generování signálů. Druhý výstupní modul (TOM) naopak k jednotce ARU připojen není, čímž dochází ke ztrátě některých z vlastností ATOM modulu jako například použití programovatelných vstupních hodin pro čítače jednotlivých ATOM kanálů.

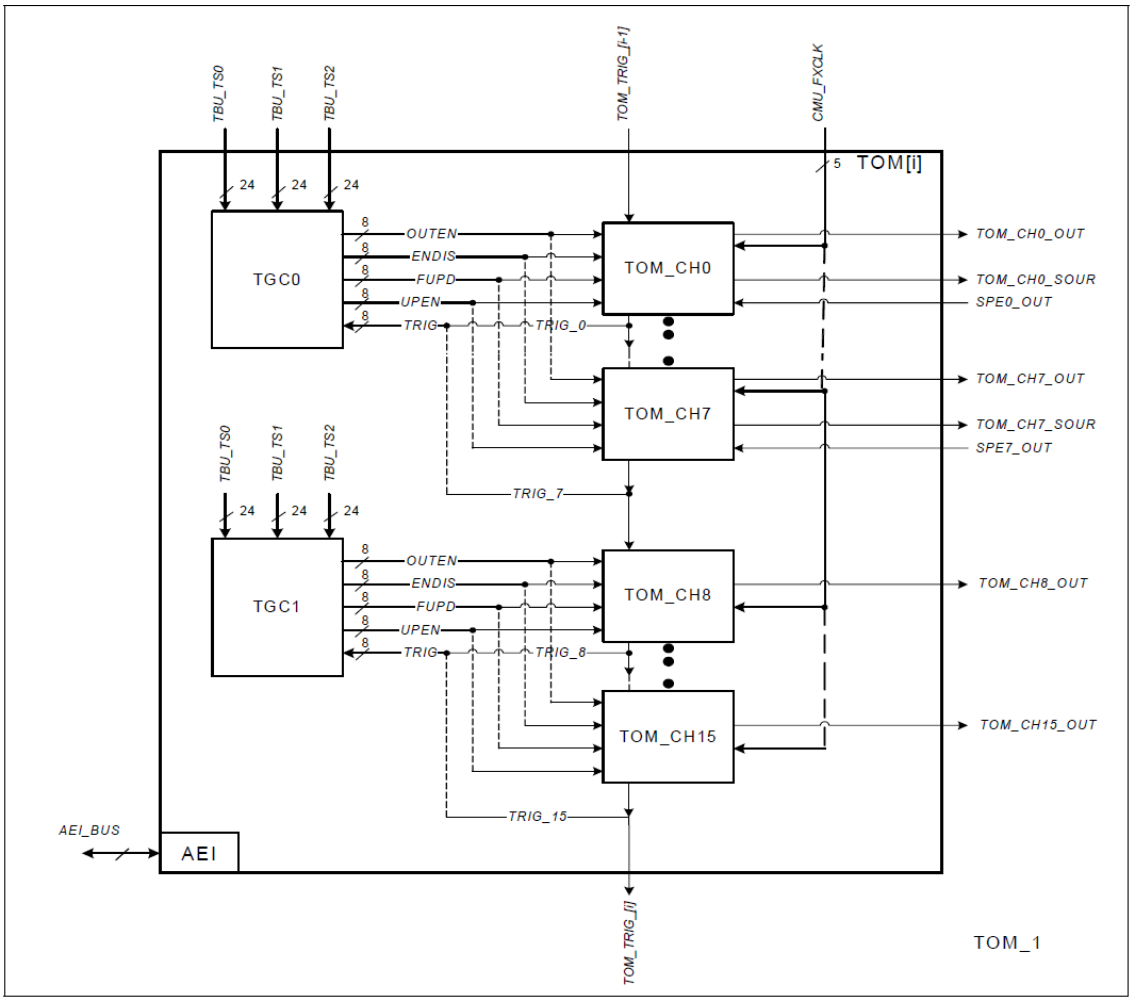
Jelikož pro generování výstupních signálů nevyužíváme ATOM modul, nebudeme se jím ani ARU jednotkou více zabývat. V následující kapitole si naopak blíže popíšeme používaný TOM modul.

2.2.1 Výstupní modul TOM

Periferie GTM neobsahuje pouze jeden výstupní TOM modul, nýbrž rovnou tři (s označením 0 - 2). Každý se skládá ze 2 řídicích jednotek a 16 výstupních kanálů. Řídící jednotky označované TGC0 a TGC1, slouží k nastavování parametrů jednotlivých kanálů (perioda a střída), povolení výstupu kanálu a povolení, či zakázání kanálů.

Na obrázku 2.3 vidíme blokové schéma TOM modulu číslo 1, schémata zbylých dvou modulů jsou naprosto totožná. Na obrázku je znázorněno rozdělení ovládání výstupních kanálů mezi řídící jednotky, kdy TGC0 ovládá kanály 0 - 7 a TGC1 kanály 8 - 15. Pro každou z jednotek je možné vybrat jednu ze tří časových základen ($TBU\_TS0 - TBU\_TS2$), díky kterým je možné dosáhnout synchronizace chování výstupních kanálů. Hodnotu základen je možné upravit během nastavení části TBU periferie GTM.

U TOM modulu je možné frekvenci výstupních kanálů zvolit jako jeden z 5 definovaných zdrojů hodin s různými frekvencemi označených $CMU\_FXCLK0 - CMU\_FXCLK4$. Frekvence jsou dány frekvencí periferie GTM, kterou dělíme hodnotou 1, 2^4, 2^8, 2^{12} nebo 2^{16}.

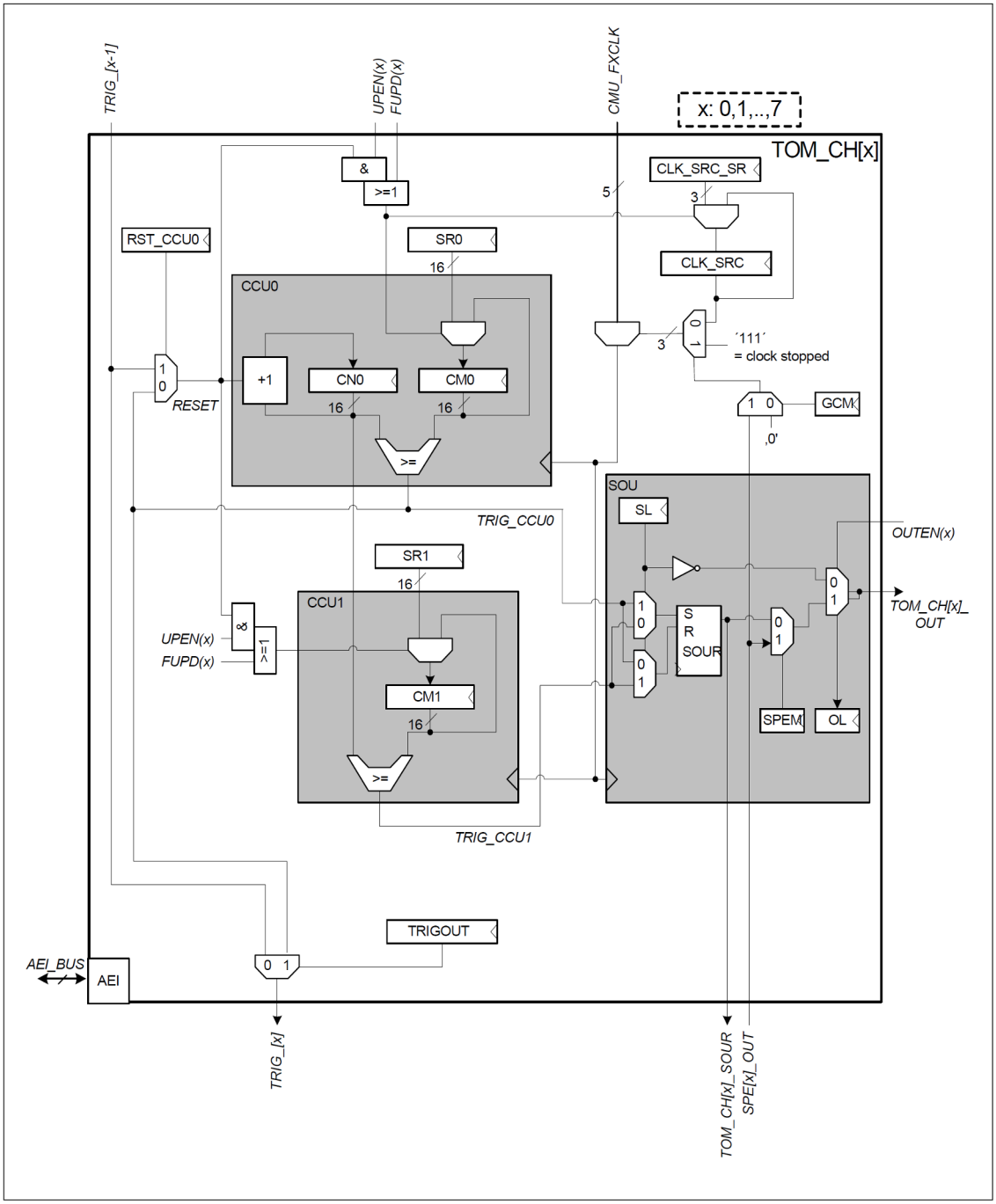


Obr. 2.3: Blokové schéma TOM modulu, převzato z [4]

Mezi architekturou jednotlivých kanálů každého modulu jsou drobné rozdíly, když kanály 0 až 7 jejichž architektura je zobrazena na obrázku 2.4, jsou totožné, dále jsou totožné kanály 8 až 14 a samostatně je veden kanál 15, pomocí kterého je možné generovat PCM neboli (Pulse count modulation). Odlišnosti v architektuře, však na generování PWM signálu nemají vliv, proto se jimi nebudeme blíže zabývat.

Každý kanál TOM modulů obsahuje dvě porovnávací jednotky čítače (CCU0 a CCU1) (viz obrázek 2.4). Čítač $CN0$ je obsažen v jednotce CCU0 a je časován jedním ze zdrojů hodin CMU_FXCLKx. Jednotky porovnávají hodnotu čítače $CN0$ s hodnotou v registrech $CM0$ a $CM1$, které reprezentují periodu, respektive střídu generovaného signálu.

Poslední částí je jednotka SCU, ve které se rozhoduje o úrovni výstupního signálu. V případě, že je bit SL nastaven na logickou hodnotu 1 je výstupní signál ve oblasti vymezené střídou v úrovni log. 1, jinak je 0 (výstup je zakázán). Pokud je bit $SL = 0$ je situace přesně opačná, to jest jeli výstup zakázán je v úrovni log. 1.



Obr. 2.4: Architektura kanálů 0 až 7 TOM modulu, převzato z [4]

2.2.2 Zřetězení TOM modulů

V případech, kdy chceme generovat paralelně více signálů se stejnou periodou, je jednou z možností využití zřetězení TOM modulů. Jeden z kanálů slouží jako referenční a zbylé generují výstupní signál (například PWM, či jiné časově závislé signály). Jako referenční musí být zvolen ten kanál, jehož číslo TOM modulu

a kanálu je nejnižší ze všech použitých. V jiném případě zřetězení modulů nebude fungovat.

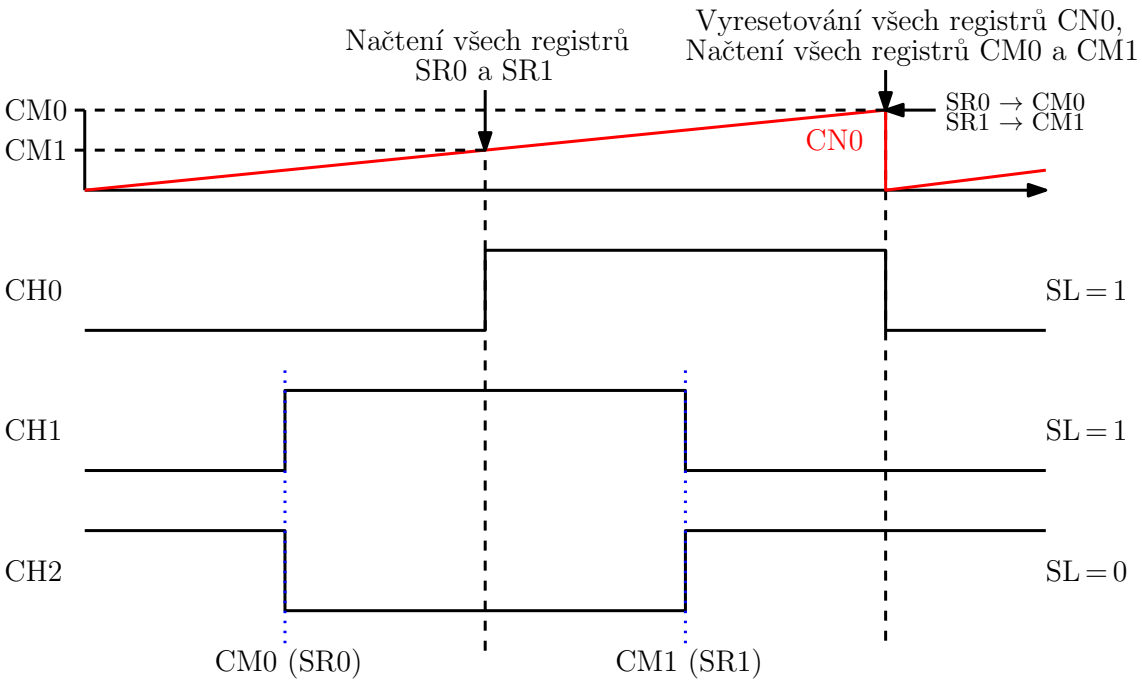
Obrázek 2.4 zobrazuje architekturu kanálů 0 až 7 TOM modulu, na které si popíšeme rozdíly mezi vytvářením kanálu referenčního a výstupního. Pro nastavování kanálů slouží registr $GTM\_TOMx\_CHy\_CTRL$, kde $x = 0$ - 2 určuje číslo TOM modulu a $y = 0$ - 15 číslo kanálu.

Při vytváření referenčního kanálu je nutné nastavit bit RST_CCU0 na hodnotu logické 0, čímž dojde k vynulování vnitřního čítače $CN0$ v případě, kdy jeho hodnota bude totožná s hodnotou v registru $CM0$, čímž určíme periodu signálu. Následně provedeme nastavení bitu $TRIGOUT$ na hodnotu log. 1. Tím vytvoříme možné připojení pro další kanály (tzv. spoušť), které se budou řídit dle tohoto signálu.

Kanál generující výstupní signál je třeba nastavit, co se týče bitů RST_CCU0 a $TRIGOUT$ uvedených u referenčního kanálu, přesně naopak. To jest RST_CCU0 na log. 1 a $TRIGOUT$ na hodnotu log. 0. Tímto nastavením je řečeno, že vnitřní čítač kanálu bude nulován v okamžiku příchodu logické hodnoty 1 „spouště“ předchozího (referenčního) kanálu. Tento signál bude díky bitu $TRIGOUT$ v log. 0 procházet k následujícímu kanálu modulu. Tímto nastavením je možné zřetězit všechny kanály všech 3 TOM modulů (to znamená možnost použít až 44 kanálů pro generování výstupních signálů).

2.2.3 Vytváření zřetězené PWM

Jak již bylo uvedeno v kapitole 2.2.1, každý kanál obsahuje dvě porovnávací jednotky, které porovnávají hodnotu čítače $CN0$ s hodnotami v registrech $CM0$ a $CM1$ (respektive v shadow registrech $SR0$ a $SR1$). Na obrázku 2.5 je zobrazen průběh čítače $CN0$ (červeně), jehož hodnota vzrůstá než dosáhne hodnoty uložené v registru $CM0$, čímž je dána perioda signálu $CH0$ (dle obrázku) i zbylých zřetězených signálů. V průběhu čítání $CN0$ je dosaženo nejprve hodnoty uložené v $CM1$, čímž je dána střída signálu. Tímto způsobem je vytvořen $CH0$, který v případě nastavení dle kapitoly 2.2.2 slouží jako referenční (viz obrázek 2.5). Při rovnosti $CN0$ a $CM1$ nultého kanálu dochází k načtení nových hodnot do shadow registrů $SR0$ a $SR1$, tyto hodnoty se následně při rovnosti čítače s $CM0$ dojde k přepsání hodnot do registrů $CM0$ a $CM1$.



Obr. 2.5: Princip vytváření zřetězené centrované PWM

Hodnoty registrů $CM0$ a $CM1$ u kanálů pro generování výstupních signálů ($CH1$ a $CH2$) mají jiný význam než u kanálu referenčního. Jak je patrné z obrázku hodnota registru $CM0$ určuje vzestupnou hranu a $CM1$ naopak hranu sestupnou (platí v případě kdy $SL = 1$), čímž je vymezena doba, kdy je výstupní signál v logické 1 ($CH1$). Pokud je $SL = 0$, takto vymezený signál je v logické 0 ($CH2$).

V případě, kdy střída kanálu 0 bude 50% a hodnoty v registrech $CM0$ a $CM1$ zbylých kanálů jsou symetrické podle vzestupné hrany $CH0$, bude se na výstupu jednat o centrovanou PWM.

2.3 Periferie VADC

Pro převod vstupního analogového signálu na digitální je u mikrokontroléru TC275 CA možno využít jeden ze dvou A/D převodníků, kterými jsou DSADC a VADC. V našem případě využíváme periferii VADC, kterou si blíže popíšeme.

Periferie VADC se skládá z osmi samostatných A/D převodníků, z nichž ke každému je připojeno osm vstupních kanálů, ze kterých je pomocí multiplexeru vybrán vždy jeden kanál určených pro převod. Zavedeme pojem *skupina*, pod kterým si představíme jeden A/D převodník se svými vstupními kanály. Každá *skupina* může pracovat nezávisle na ostatních, nebo je možné převodníky použít pro synchronizovaný převod více kanálů.

Pro každou ze *skupin* je možné použít jiné nastavení, přesněji jiný mód převodu, rozlišení či použít synchronizovaný převod více skupin najednou. Možností nastavení je mnoho, nejdůležitější vlastnosti každé skupiny si blíže popíšeme v následujících odstavcích.

2.3.1 Způsob výběru převáděných kanálů

Dříve než je spuštěn převod je nutné vhodným způsobem vybrat vstupní kanál či kanály, které chceme převádět. Existují 2 způsoby výběru kanálů, kterými jsou pevný výběr (princip fronty) a průběžné skenování, které lze dále rozdělit opět na 2 způsoby (globální nebo skupinové skenování).

- Princip fronty
  Princip je založen na výběru kanálů v rámci skupiny. Kanály jsou dle požadavků na spouštění převodu vkládány do FIFO paměti, neboli kanál který bude vložen první do paměti bude jako první převeden. Počet kanálů je maximálně 8, tudíž je možné do FIFO vložit celou skupinu. Sekvenci je možné spustit příkazem v programu, či externí událostí (například trigger od vzestupné, sestupné či obou hran PWM). Po dokončení převodu prvního kanálu dojde k jeho zařazení na konec fronty a automaticky dojde ke spuštění převodu druhého kanálu v pořadí. Opětovný převod od prvního kanálu je možné zahájit okamžitě, či s pevnou časovou základnou. Sekvenci je možné přerušit, a to v případě příchodu žádosti s vyšší prioritou. Právě vykonávaný převod je v tento moment přerušen, parametry převodu jsou uloženy a vykoná se převod s vyšší priority. Po jeho dokončení se nepokračuje dalším kanálem z fronty, ale dojde k opětovnému převodu přerušeného kanálu s dřívějšími parametry.
- Skenování kanálů
  Periferie VADC rozlišuje dva typy skenování, a to skupinové a globální. Oba typy pracují na stejném principu, který spočívá v převodu vybraných kanálů od toho s nejvyšším číslem kanálu směrem k číslu nejnižšímu. Spuštění sekvence je opět možné buďto příkazem v programu nebo od externí události.
  - Skenování skupiny
    Mezi skenované kanály je možné vybrat pouze ty, které jsou součástí vybrané skupiny.
  - Globální skenování (Background mód)
    V tomto případě je možné mezi skenované kanály vybrat libovolný vstupní kanál mikrokontroléru z libovolné skupiny.

  Stejně jako u fronty lze i skenování kanálů přerušit žádostí s vyšší prioritou. Opět dojde k přerušení převodu a uložení parametrů pro následný opakovaný převod.

Výběr používaného principu pro jednotlivé kanály se provádí nastavením jednoho z bitů $ASENy$ v registru $GxARBPR$. První parametr, to jest x=0-7 reprezentuje *skupinu* kanálů, druhý odpovídá vybranému principu:

$y = 0$ Princip fronty
$y = 1$ Skenování skupiny
$y = 2$ Globální skenování

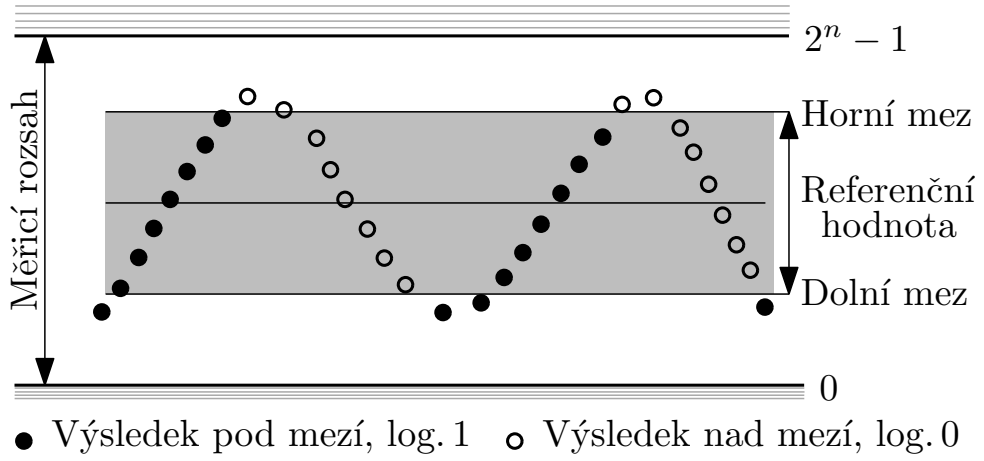
U všech způsobů je možné zvolit, zda po příchodu žádosti o převod s vyšší prioritou dojde k okamžitému převodu bez ohledu na dokončení předchozího převodu či vyčtení a uložení výsledku. Druhou možností je, že spuštění nově příchozí žádosti se pozdrží do té doby, než je vyčten nově uložený výsledek z výsledkového registru.

2.3.2 Módy převodu

A/D převodník je možné používat v jednom ze dvou předdefinovaných módů. Ty se od sebe výrazně liší, nejenom výsledkem, ale také celkovým principem převodu. Jednotlivé módy si popíšeme na následujících řádcích.

- Standardní mód
  Tento mód pracuje s volitelným rozlišením 8, 10 nebo 12 bitů, které je třeba zvolit na počátku buďto globálně pro všechny převodníky nebo různě pro každou skupinu. Jako výsledek je do předem vybraného výsledkového registru (globálního nebo skupinového) uloženo číslo ve tvaru 16 bitového čísla. Před spuštěním (u vybraných kanálů) či po dokončení (u všech kanálů) převodu je možné povolit či zakázat kalibraci převodníků.

  Výsledek standardního módu může být následně použit pro kontrolu limitů, nalezení extrémů. Další možností je filtrace výsledku jedním z definovaných filtrů, kterými jsou IIR a FIR.
- Mód rychlého převodu
  Rychlý převod znamená, že převodník pracuje s 10 bitovým rozlišením, a jako výsledek je nám vracen pouze jeden bit (FCR - Fast Compare Result), který říká zda převáděná hodnota je vyšší nebo nižší, nežli předdefinované meze. Tyto meze jsou jako 10 bitové číslo vloženy do odpovídajícího výsledkového registru, ve kterém je následně ukládán i výsledek. FCR nabývá hodnoty logická 1 od doby, kdy je hodnota vstupního signálu nižší než dolní mez, do doby kdy nabude hodnotu horní meze. Hodnotu log. 0 nabývá po dobu, od kdy byl vstupní signál vyšší než horní mez, do doby než-li narazí na mez dolní. Pro lepší pochopení je možno si prohlédnout obrázek 2.6, který zobrazuje výše uvedený princip.



Obr. 2.6: Zobrazení výsledku pomocí porovnávacích mezí, převzato z [4]

Mód převodu volíme pomocí bitu CMS ve skupinových registrech $GxICLASS0$, $GxICLASS1$, (x=0-7 je číslo *skupiny*), nebo registru $GLOBICLASS$ pro globální nastavení. Tento bit může pro dané módy převodu nabývat následujících hexadecimálních hodnot:

- 0_h 12-bitový standardní převod
- 1_h 10-bitový standardní převod
- 2_h 8-bitový standardní převod
- 5_h Mód rychlého převodu

2.3.3 Spuštění převodu

Převod vstupního signálu je zahájen po obdržením jedné z níže uvedených žádostí o spuštění převodu.

- Žádost o převod vyvolaná programem
  Pokud se ve vykonávaném programu objeví žádost o převod, je aktivován zdroj žádostí a dojde k okamžitému spuštění převodu vstupního kanálu vybraného A/D převodníku.
- Žádost o převod od externí události
  Při použití této žádosti je nutné nejprve provést synchronizaci zdroje žádostí a externích událostí. Je důležité vybrat vhodný trigger, jejichž přesný výpis je možné nalézt v manuálu mikrokontroléru TC275 CA. Mezi externí události patří například obdržení pulsu od časovače při vzestupné či sestupné hraně PWM signálu, či obdržený puls na některém ze vstupních pinů mikrokontroléru.

Při použití žádosti vyvolané programem, musí být na vhodné místo kódu umístěn příkaz pro spuštění převodu. Příkaz se liší dle použitého principu výběru kanálů převodníku. Například při použití *Background módu* spočívá v nastavení bitu $LDEV$ umístěného v registru $BRSM$ na hodnotu logické 1.

Při použití spuštění od externí události je třeba nejdříve povolit toto spuštění, což je provedeno nastavením bit $ENTR$ v registru $GxQMR0$ (x značí číslo skupiny). Následně v registru $GxQCTRL0$ vybereme trigger dle potřeby.

2.3.4 Synchronizovaný převod více kanálů

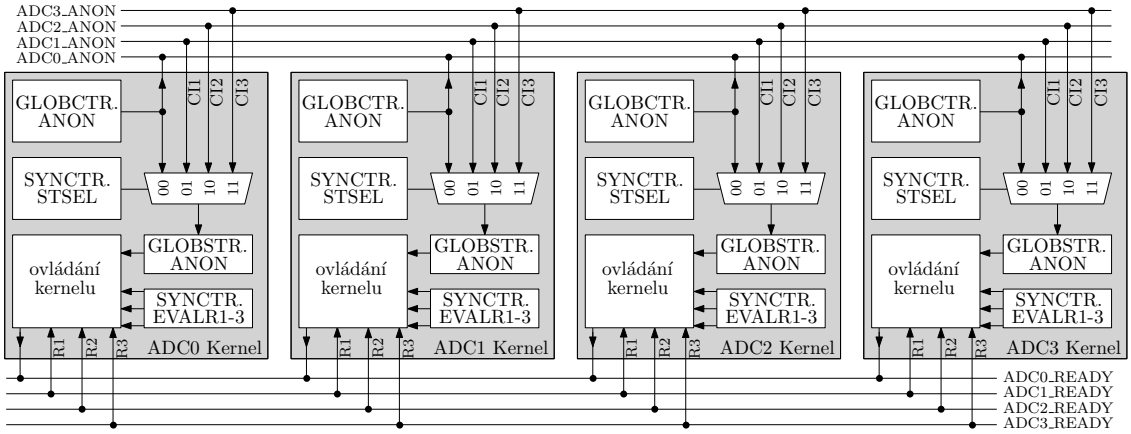
V některých případech je třeba provést synchronizovaný převod několika vstupních kanálů. Pro takový případ lze u mikrokontroléru TC275 CA použít jednu ze dvou možností, kterými jsou:

1. Ekvidistantní vzorkování
2. Paralelní převod

Paralelní převod

Princip paralelního převodu spočívá v současném sejmutí vzorků ze všech vybraných vstupních kanálů v jeden časový okamžik. U TC275CA je možno současně sejmout vstupní hodnoty maximálně 8 kanálů, přičemž každý kanál (musí mít stejné číslo) musí být z jiné skupiny, která je zde označována jako kernel. Jednotlivé kernely jsou rozděleny do dvou synchronizačních skupin označených A a B, skupina A obsahuje kernel 0 až 3 a skupina B kernel 4 až 7. Jeden kernel (libovolně vybraný) pracuje jako synchronizační master, všechny ostatní pracují jako slave.

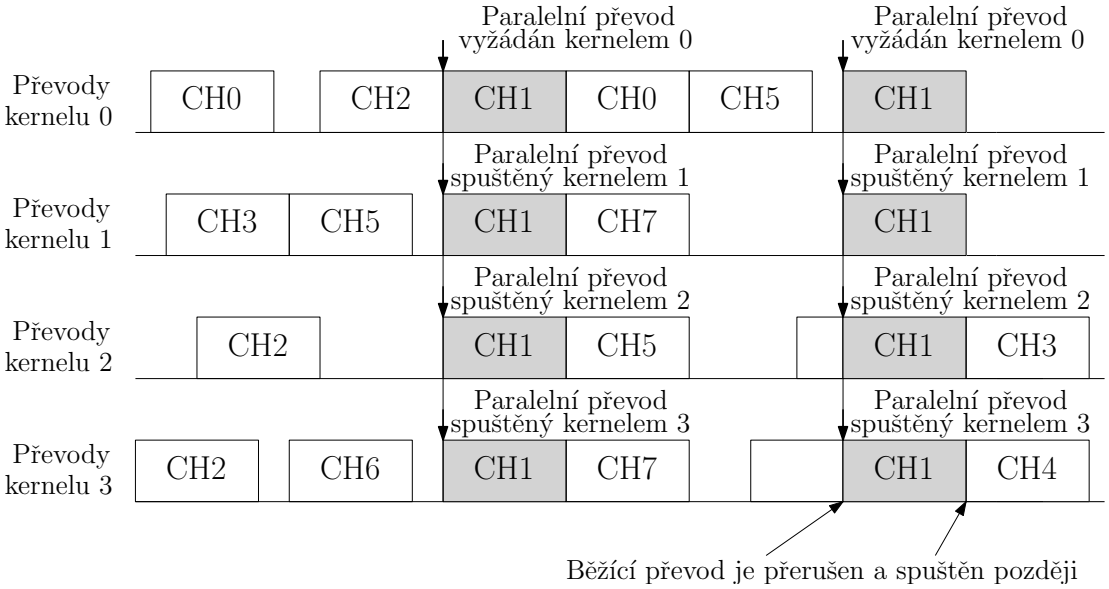
Výběr převáděného kanálu pro všechny kernely synchronizační skupiny je provedeno v nastavení master kernel, (bit $SYNC$ registru $GxCHCTRy$, kde x je číslo kernelu a y číslo kanálu vybraného pro převod). Rozlišení převodníku a výsledkový registr je již možno nastavit libovolně pro každý synchronizovaný kernel.



Obr. 2.7: Synchronizace pomocí signálů ANON a READY, převzato z [4]

Na obrázku 2.7 můžeme vidět synchronizaci jednotlivých kanálů pomocí signálů *ANON* a *READY*. Signál *ANON* slouží k rozhodnutí, zda se jedná o master, či slave. Pokud jde o slave, *ANON* říká, který ze zbylých kernelů je master, čili dle kterého bude synchronizován. Je-li například jako master veden kernel 0, mají všechny kernely včetně samotného mastera nastavenu hodnotu $ANON = 01_B$. Signál *READY* je využíván pro indikaci stavu připravenosti slave kernelů pro paralelní převod. Pokud master obdrží *READY* signál od všech slave je možné zahájit převod.

Jednotlivé kanály (kernely) pracují nezávisle, dokud není vyžádán paralelní převod. Po příchodu žádosti o převod master čeká na *READY* signál a následně zahájí převod. Pokud je v momentě žádosti prováděn jiný převod, je přerušen a je dokončen po paralelním převodu, viz obrázek 2.8.



Obr. 2.8: Princip paralelního převodu, převzato z [4]

2.3.5 Vyčtení výsledku převodu

Po dokončení převodu je nově uložený výsledek v registru indikován nastavením bitu VF do logické úrovně 1. Existují dva způsoby vyčtení výsledku po dokončení paralelního převodu:

- Událost od výsledku
- Událost od zdroje

První způsob spočívá ve vyvolání přerušení v momentu, kdy je nový výsledek uložen do vybraného registru. Je nutné mít povolena přerušení a mít správně vybrané číslo žádosti, které slouží k synchronizaci rutiny přerušení a jeho zdroje.
U paralelního převodu (a nejen zde) je možné vyvolávat přerušení po dokončení každého z vybraných kanálů, avšak v takový okamžik může dojít k zahlcení procesoru přerušeními.

Druhou možností je vyvolání přerušení po dokončení sekvence převodů, a to ať už byla tato sekvence vyvolána frontou či skenováním vybraných kanálů. Totéž platí pro paralelní převod, u kterého je možné vyvolat přerušení po dokončení převodu jednoho z kanálu, či po ukončení převodu všech zvolených kanálů.

U práce s přerušeními je nutné si dávat pozor na jejich priority, v žádném případě se nesmí objevit dvě či více rutin přerušení, která mají stejnou prioritu. Pokud by taková situace nastala není zaručeno, která z rutin se vykoná a zda se vykoná správně.

2.4 Periferie GPT12

GPT12 je složena z pěti 16 bitových časovačů ($T2$ - $T6$) uspořádaných do dvou bloků označených $GPT1$ a $GPT2$. Každý z časovačů a bloků může pracovat nezávisle na ostatních, nebo může být zřetězen spolu s jiným časovačem, avšak vždy jen v rámci jednoho bloku.

Blok $GPT1$ je složen ze tří časovačů: $T2$, $T3$ a $T4$. Každý z nich může pracovat v jednom ze čtyř definovaných módů, kterými jsou: časovač, čítač, hradlový časovač a jako rozhraní pro připojení inkrementálního snímače. Nastavení jednotlivých časovačů je prováděno pomocí registru $TxCON$, kde x je číslo vybraného časovače (2 až 4), ve kterém lze zvolit například pracovní mód či směr čítání.

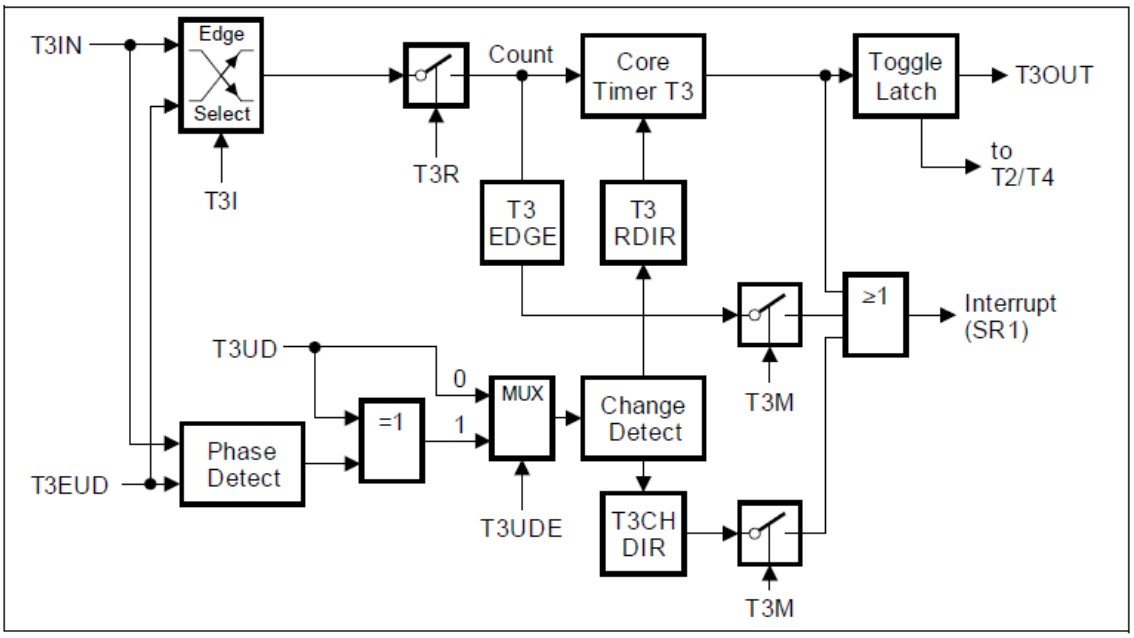
Zbylé časovače, to jest $T5$ a $T6$ jsou obsaženy v bloku $GPT2$. Opět si lze vybrat jeden z pracovních módů, v tomto případě však pouze ze tří možností, kterými jsou: časovač, čítač, hradlový časovač. Nastavení je opět prováděno pomocí registru $TxCON$ ($x = 5$ - 6). Blok $GPT2$ však obsahuje navíc $CAPREL$ registr, neboli zachyť a přehraj (Capture/Reload). Funkcí tohoto registru je zachycení obsahu časovače

$T5$ nebo přepsání hodnoty časovače $T6$. Speciálním režimem je možné využívat obě funkce současně.

U každého z bloků je možné nastavit frekvenci pomocí děliček systémových hodin. Děličky se jsou nastaveny bity $BPS1$ v registru $T3CON$ a $BPS2$ v $T6CON$. Jako zdroj hodin lze v případě použití vývojového kitu zvolit interní oscilátor s frekvencí 100 MHz nebo krystal externí s frekvencí 20 MHz. Maximální možná frekvence, se kterou je možné pracovat u jednotlivých bloků periferie (při použití oscilátoru s frekvencí 100 MHz) je různá, pro $GPT1$ je $f_{max} = 25$ MHz čemuž odpovídá rozlišení 40 μs, u $GPT2$ je pak možné pracovat až s $f_{max} = 50$ MHz (rozlišení 20 μs). V obou případech se jedná o 16 bitové časovače, jejichž maximální perioda pro výše uvedené frekvence je rovna 2,62 ms, respektive 1,31 ms.

2.4.1 Využití GPT12 pro zpracování dat z enkodéru

Enkodér je možné k mikrokontroléru TC275 CA připojit přímo, bez použití externího rozhraní. To znamená přímo na konektory vývojového kitu přivést výstupní signály A, B a Z enkodéru. Pro zpracování dat z enkodéru dle postupu uvedeného v kapitole 1.3, to jest při využití obou metod zpracování dat je třeba využít tří časovačů periferie GPT12, které pracují se signály enkodéru.

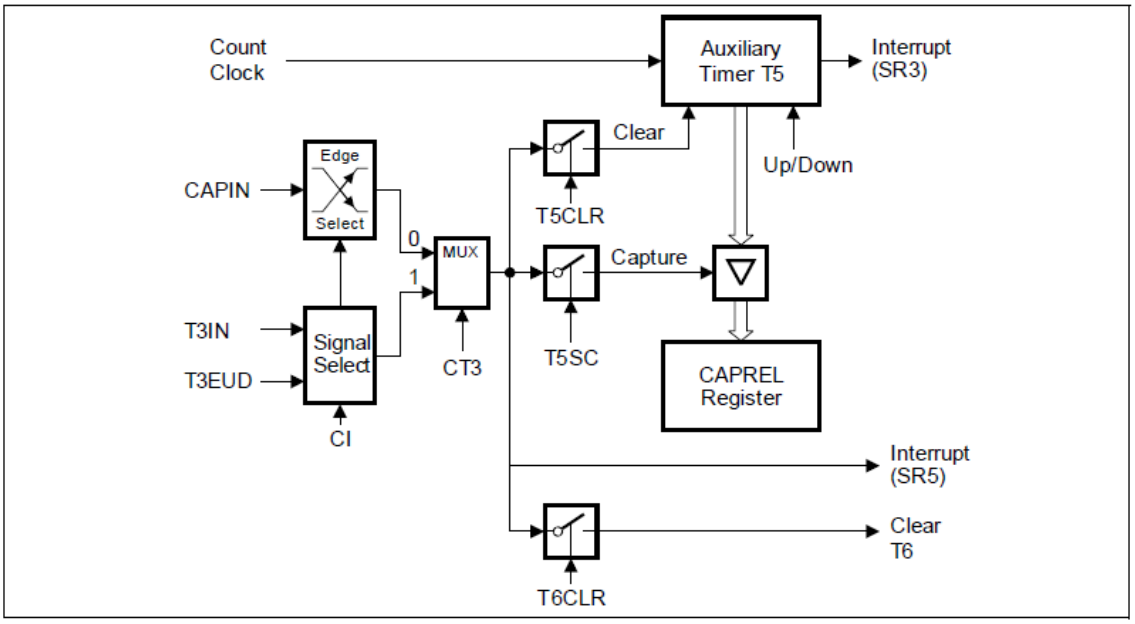


Obr. 2.9: Blokové schéma časovače T3, mód pro připojení inkrementálního enkodéru, převzato z [4]

Pro první metodu, to jest pro čítání pulzů za určitou dobu je třeba dvou časovačů, v našem případě $T2$ a $T3$. Časovač $T2$ slouží k vytvoření periody, po kterou

budou čítány pulzy enkodéru. Časovač $T3$ pracuje v módu rozhraní pro inkrementální snímač, jehož blokové schéma můžeme vidět na obrázku 2.9. Tento mód je velmi podobný módu čítače, jelikož dochází k čítání příchozích pulzů z enkodéru po dobu, kterou vymezuje perioda časovače $T2$.

Z blokového schématu je patrné, že pro připojení enkodéru je nutné použití dvou pinů, kterými jsou $T3IN$, $T3EUD$, na které jsou přivedeny signály enkodéru A, B. Tyto signály jsou určeny pro čítání pulzů a detekci směru otáčení. Pro správnou funkci je nutné použít i třetí výstup enkodéru, to jest signál Z. Ten je přiveden na vstup $T4IN$ a slouží k nulování čítače $T3$ při průchodu mechanickou nulovou pozicí. Pro výpočet rychlosti otáčení je vytvářen rozdíl mezi počtem pulzů za dvě po sobě jdoucí periody časovače $T2$.



Obr. 2.10: Blokové schéma časovače T5, převzato z [4]

Obrázek 2.10 znázorňuje blokové schéma časovače $T5$ s možností zápisu hodnoty do $CAPREL$ registru, který je použit pro druhou metodu, to jest měření času mezi pulzy. Po zachycení pulzu dojde ke spuštění časovače $T5$ a k jeho zastavení dojde při detekci následujícího pulzu na jednom ze vstupních pinů $T3IN$ nebo $T3EUD$. V tento moment dochází k zachycení obsahu časovače $CAPREL$ registrem a jeho opětovnému spuštění od nulové hodnoty. Z hodnoty $CAPREL$ registru je počítána rychlost otáčení. V případě, kdy dojde k přetečení časovače, časový úsek oddělující dva po sobě jdoucí pulzy enkodéru je větší něž 1,31 ms, je rychlost otáčení považována za nulovou.

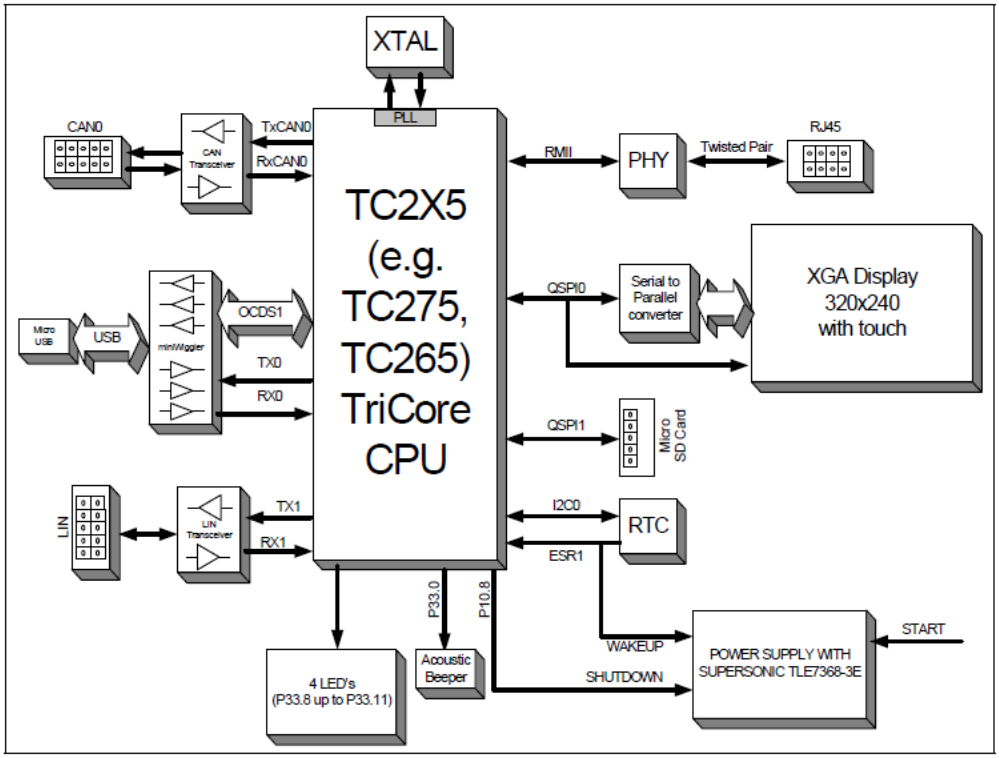
Princip měření doby mezi pulzy se používá pro nízké otáčky, kdežto čítání pulzů za časový úsek je využíváno pro otáčky vyšší. Mez přepínání mezi metodami je zá-

vislá na mnoha parametrech, jako například frekvence časovačů, rozlišení enkodéru či využití jednoho nebo obou výstupů (A a B) enkodéru. Proto je nejlepší mezní rychlost pro přepínání metod zjistit prakticky před prvním použitím a to proměřením závislosti chyby metod na otáčkách pro obě metody. Chyba metody měření času mezi pulzy bude od určitých otáček narůstat, kdežto u druhé metody bude chyba s rostoucími otáčkami klesat. Pokud si průběhy znázorníme graficky, mez přepínání bude v bodě průsečíku obou charakteristik.

3 VÝVOJOVÝ KIT S MIKROKONTROLÉREM TC275 CA

Pro usnadnění práce s mikrokontrolérem TC275 CA byl vyvinut vývojový kit [5], který je prvním krokem pro vývoj aplikací a jejich odzkoušení na reálných přípravcích.

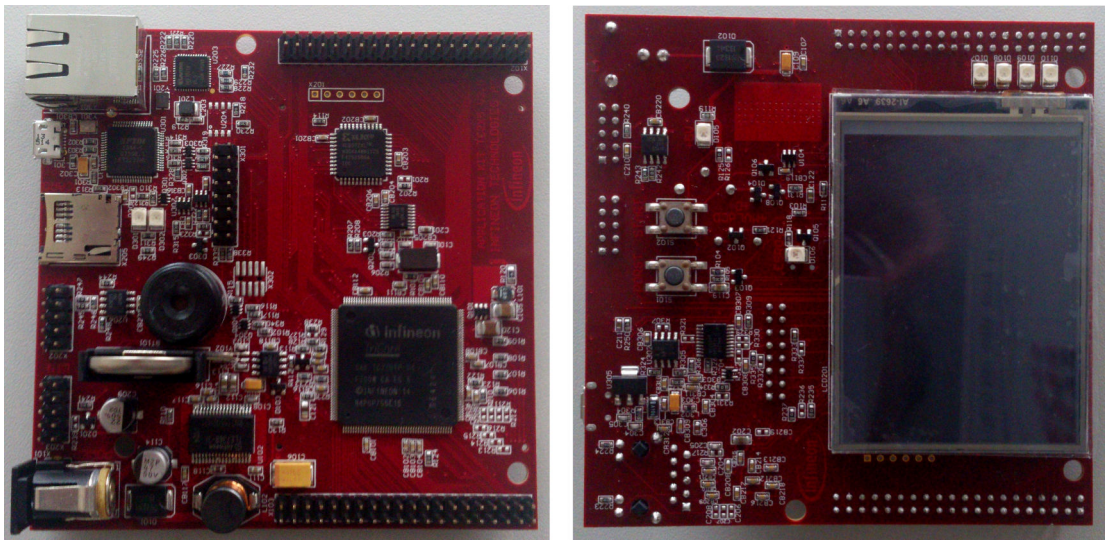
Na obrázku 3.1 je znázorněno blokové schéma, na kterém jsou vidět všechny důležité části vývojového kitu. Těmi jsou samotný mikrokontrolér, krystal externích hodin s frekvencí 20 MHz, množství rozhraní pro připojení vývojového kitu k přípravkům. Velkou výhodou kitu je snadné propojení s PC pomocí USB kabelu s micro USB koncovkou, která je na kitu vyvedena. Následná komunikace probíhá pomocí programu Memtool, který si blíže popíšeme v kapitole 4.2.



Obr. 3.1: Blokové schéma vývojového kitu, převzato z [5]

Pro napájení je potřeba napětí v rozsahu 5,5 V až 40 V. Spotřeba vývojového kitu není specifikována, avšak pro napájení by mělo být dostačující napětí 6 V a proud 600 mA.

Vývojový kit má tvar čtverce o hraně 10 cm, a jehož nejvýraznějším prvkem je zajisté dotykový LCD displej s rozlišením 320 x 240 pixel. Ten můžeme vidět v pravé polovině obrázku 3.2 společně se čtveřicí programovatelných LED diod a dvojicí tlačítek, z nichž jedno je pro zapnutí kitu a druhé pro jeho manuální reset.



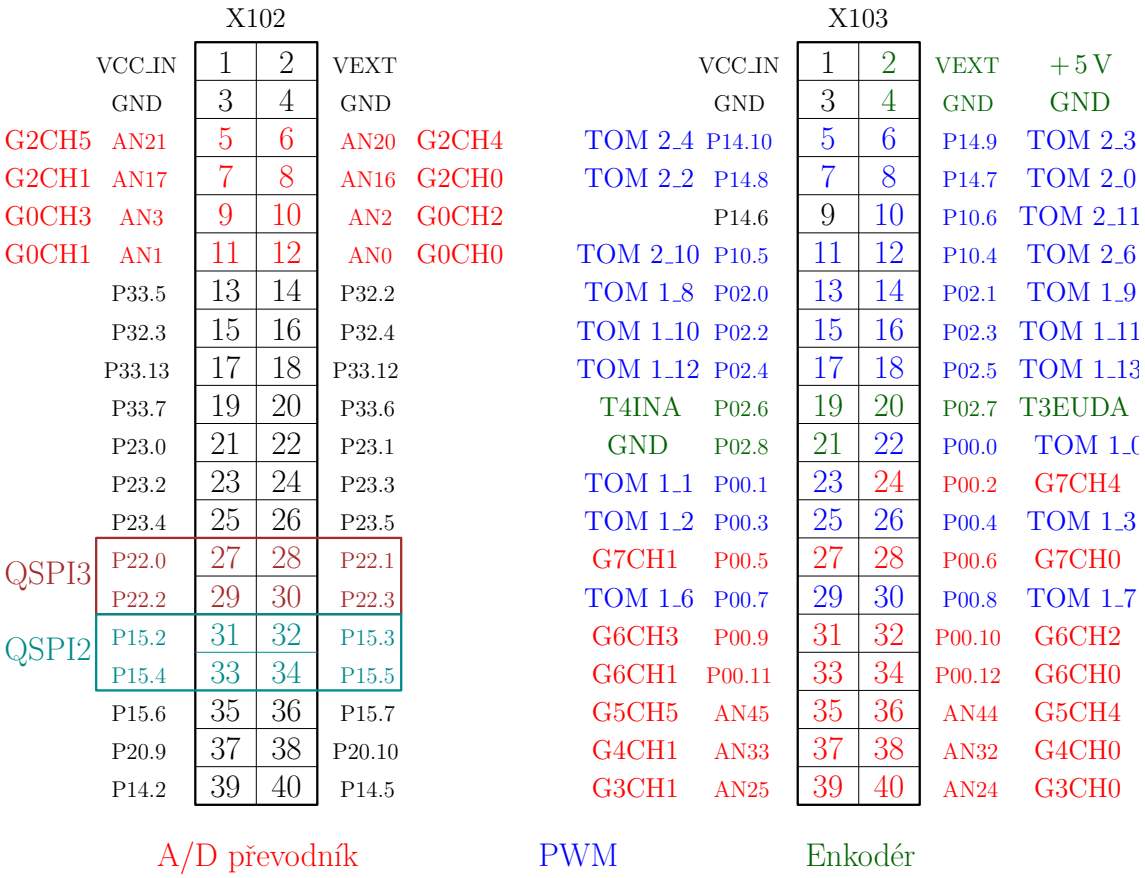
Obr. 3.2: Vývojový kit s mikrokontrolérem TC275 CA

Na levé polovině obrázku 3.2 je vidět samotný mikrokontrolér spolu s napájecím konektorem, slotem pro micro SD a konektory RJ45 a micro USB. Pro připojení vstupních nebo výstupních zařízení je kit vybaven dvojicí konektorů ($X102$ a $X103$), z nichž každý má 40 pinů. Je třeba si dávat pozor na polohu pinů na kitu vůči obrázku 3.3, kdy na desce je jeden z konektorů otočen. Na obrázku 3.3 vidíme uspořádání pinů u obou konektorů, které obsahují čtyři skupiny pinů. Jedná se o piny napájecí (VCC_IN a $VEXT$), zemnící (GND), analogové vstupy (ANx) a programovatelné vstupně/výstupní porty označené Px, kde x značí číslo pinu.

	X102				X103	
VCC_IN	1 2	VEXT		VCC_IN	1 2	VEXT
GND	3 4	GND		GND	3 4	GND
AN21	5 6	AN20		P14.10	5 6	P14.9
AN17	7 8	AN16		P14,8	7 8	P14.7
AN3	9 10	AN2		P14,6	9 10	P10.6
AN1	11 12	AN0		P10.5	11 12	P10.4
P33.5	13 14	P32.2		P02.0	13 14	P02.1
P32.3	15 16	P32.4		P02.2	15 16	P02.3
P33.13	17 18	P33.12		P02.4	17 18	P02.5
P33.7	19 20	P33.6		P02.6	19 20	P02.7
P23.0	21 22	P23.1		P02.8	21 22	P00.0
P23.2	23 24	P23.3		P00.1	23 24	P00.2
P23.4	25 26	P23.5		P00.3	25 26	P00.4
P22.0	27 28	P22.1		P00.5	27 28	P00.6
P22.2	29 30	P22.3		P00.7	29 30	P00.8
P15.2	31 32	P15.3		P00.9	31 32	P00.10
P15.4	33 34	P15.5		P00.11	33 34	P00.12
P15.6	35 36	P15.7		AN45	35 36	AN44
P20,9	37 38	P20.10		AN33	37 38	AN32
P14.2	39 40	P14.5		AN25	39 40	AN24

Obr. 3.3: Vstupně/výstupní porty vývojového kitu, převzato z [5]

Použití jednotlivých portů je znázorněno na obrázku 3.4. Barevně rozlišené jsou porty, které využívají různé periferie, červeně jsou označeny piny které používá A/D převodníku, modře piny TOM modulů a zeleně piny periferie GPT12. Orámované úseky pinů je možné použít pro QSPI.



Obr. 3.4: Využití vstupně/výstupních portů vývojového kitu

Ani u jedné z použitých periferií není možné využít všechny její kanály, jelikož na vývojovém kitu je počet vstupně/výstupních portů omezen. Některé porty je možné naopak využít pro několik námi vybraných periferií. Z tohoto důvodu je nutné se rozhodnout, které porty budou využity pro kterou z periferií. V našem případě jsme rozhodli, že porty $P00.2$, $P00.5$ a $P00.6$ budou využity pro periferii VADC, porty $P00.1$, $P00.3$, $P00.4$, $P00.7$ a $P00.8$ slouží jako výstupy periferie TOM a nakonec porty $P02.6$, $P02.7$, $P02.8$ slouží jako vstupy pro periferii GPT12.

4 POUŽITÉ PROSTŘEDÍ

Pro komunikaci s vývojovým kitem a následný vývoj aplikací by za běžných okolností byly potřeba tři programy. V našem případě jsou však všechny tři programy od různých výrobců implementovány do jediného prostředí s názvem TriCore Development Platform.

- Vývojové prostředí Eclipse IDE
- UDE debugger
- Infineon Memtool

Jedná se o platformu určenou pro vytváření projektů u nejnovějších vícejádrových architektur procesorů TriCore. Spravuje veškeré nastavení vytvářeného projektu a celé jeho sestavení.

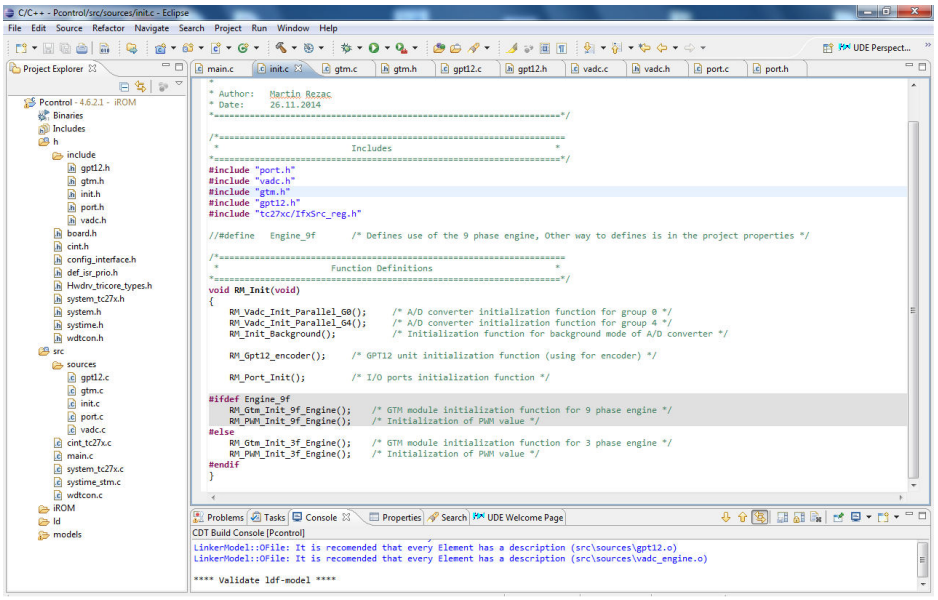
4.1 Vývojové prostředí Eclipse IDE

Jedná se o prostředí vytvářené firmou HighTec [6], které je založené na otevřené vývojové platformě Eclipse. Pracujeme s verzí prostředí s označením 4.6.2.1.

Při založení projektu je nutné vybrat přesný typ používaného mikrokontroléru, včetně jeho verze (v našem případě verze C) a jazyk ve kterém budeme projekt vytvářet (C nebo $C++$). Dle vybraného mikrokontroléru a programovacího jazyku je vygenerována hardwarová konfigurace, uspořádání paměti a jsou vygenerovány hlavičkové souboru odpovídající vytvořené konfiguraci. Tím je dokončeno vytváření projektu a je možné začít vytvářet vlastní kód, ve vytvořeném projektu je jednoduchá funkce pro demonstraci (blikání LED diodami).

Na obrázku 4.1 vidíme okno prostředí Eclipse IDE, které se skládá ze tří částí, kterými jsou:

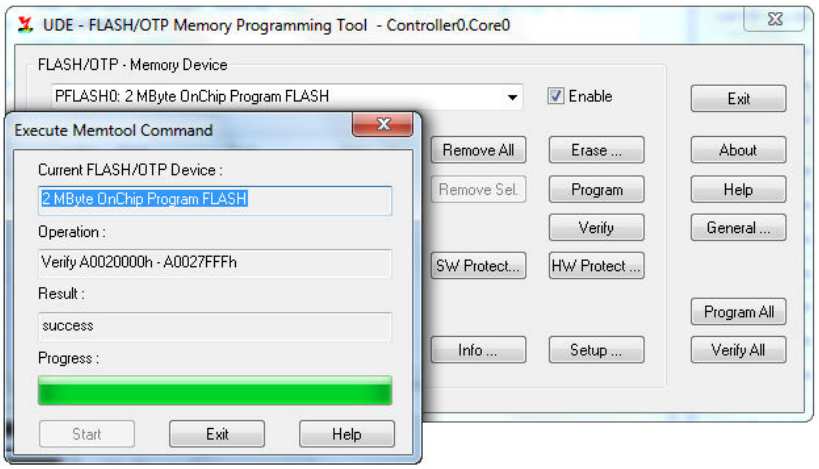
- Okno průzkumníka (Project Explorer)
  Zobrazuje strukturu vytvářeného projektu a umožňuje vstoupit do nastavení projektu, ve kterém je možné vytvářet makra, či přidávat umístění knihoven
- Pracovní okno
  Jedná se o hlavní okno prostředí, ve kterém je vytvářen kód projektu.
- Konzolové okno (Console, Problems)
  Slouží k zobrazování stavu projektu při jeho překladu, je možné se přepínat mezi chybovým hlášením a oknem zobrazující stav překladu.



Obr. 4.1: Základní okno programu Eclipse IDE

4.2 Infineon Memtool

Pro propojení vývojového kitu a debuggeru, respektive PC je zapotřebí program Memtool [7] od firmy Infineon. Ten slouží pro nahrání vytvořeného projektu, přesněji souboru s příponou *hex*, který vzniká po úspěšné kompilaci. Na obrázku 4.2 je znázorněno okno programu Memtool během nahrávání programu do mikrokontroléru.

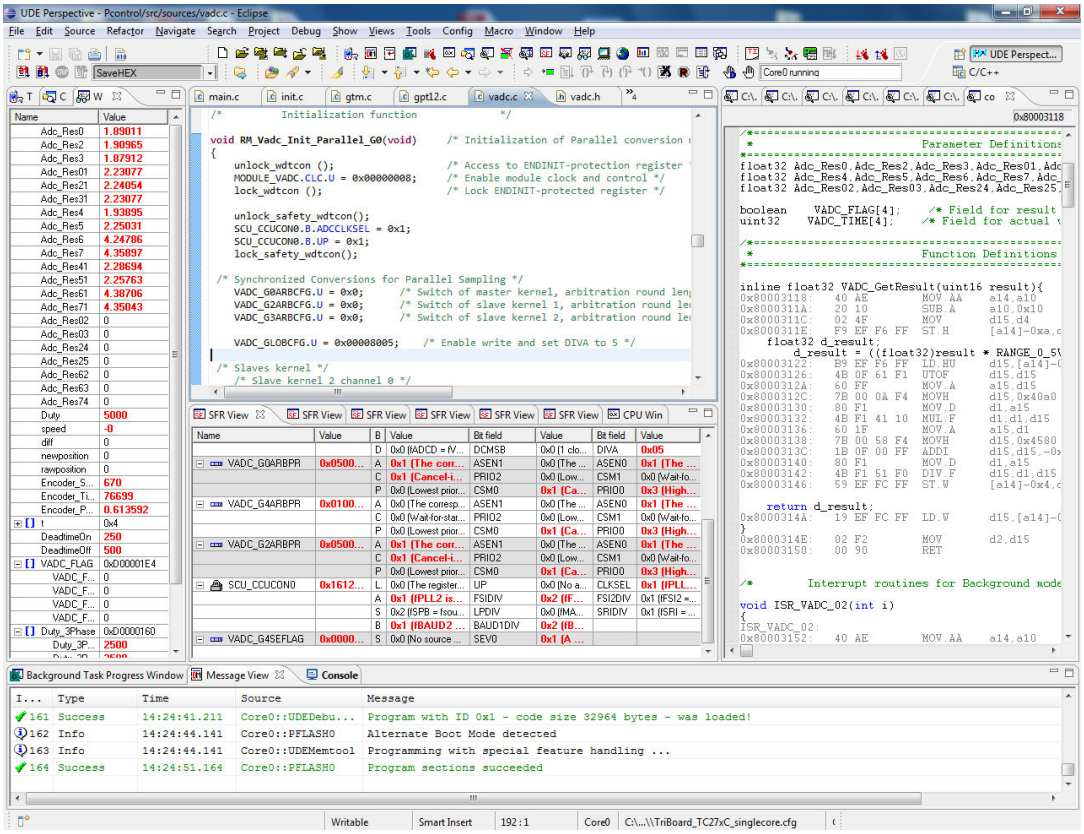


Obr. 4.2: Okno aplikace Memtool během nahrávání projektu do mikrokontroléru

Pro připojení je také možné použít hardware s názvem DAP miniWiggler. Nástroj konvertuje USB rozhraní na 10 pinový JTAG konektor, čímž je dosaženo vyšší rychlosti při nahrávání programu do mikrokontroléru.

4.3 UDE debugger

UDE debugger [8] je nejmodernější ladící nástroj pro vývoj aplikací na procesorech TriCore, který vytváří firma PLS. Pomocí tohoto nástroje můžeme provádět ladění a testování běžícího programu. Další výhodou debuggeru je možnost grafického zobrazení, pomocí kterého můžeme zobrazit několik průběhů (výstupních veličin) do jednoho grafu. Dále je možné si zobrazit obsah registrů a sledovat změny hodnot jednotlivých bitů v těchto registrech při ladění programu.



Obr. 4.3: Print screen ladícího programu UDE debugger

Na obrázku 4.3 můžeme vidět obsah obrazovky prostředí UDE debuggeru. V horní střední části je vidět oblast pro úpravu kódu, kterou je však po jakékoliv změně nutné uložit a opět zkompilovat. V dolní střední části jsou pak zobrazeny obsahy pracovních registrů ($SFR\ View$), kdy černé položky od předchozího spuštění programu nezměnili svou hodnotu. Naopak červené položky registrů svou hodnotu změnili. Měnit obsah je také možno pravým kliknutím na vybraný bit v registru a následným výběrem požadované hodnoty. Pravá část zobrazuje právě vykonávaný kód programu, do kterého je možné vkládat zarážky (breakpointy). Je možné se přepínat v zobrazení vykonávaného kódu mezi jazykem $C/C++$ nebo příkazy v Assembleru.

5 VYTVÁŘENÝ PROJEKT

Projekt je vytvořen v prostředí HighTec v programovacím jazyce *C*. Hlavní funkcí projektu je funkce *main* v souboru *main.c*, ze které jsou volány inicializační funkce mikrokontroléru (funkce *SYSTEM_Init*) a jeho periferií.

Po vytvoření projektu ve verzi 4.6.2.1 prostředí HighTec nebylo možné pracovat s přerušeními, a to z důvodu nedefinování makra *USE_IRQ*. Po definici však problémy neustaly, a proto jsme byli nuceni využít projektu z novější verze 4.6.5.0, ze které jsme převzali funkci pro inicializaci tabulky přerušení *_init_vectab* a umístili ji do oblasti definované makrem *USE_IRQ*. Po této úpravě již bylo možné přerušení při práci využívat.

5.1 Inicializace periferií

Inicializace periferií mikrokontroléru je rozdělena na samostatné funkce dle periferií či jejich částí. Volání těchto funkcí je umístěno ve funkci s názvem *RM_Init*, která je obsažena v souboru *init.c* a deklarována je v *init.h*. Zdrojový kód této funkce můžeme vidět na následujících řádcích.

```
void RM_Init(void)
 {
      RM_Port_Init();
            /* I/O ports initialization function */
      RM_Vadc_Init_Parallel_G0();
            /* A/D converter initialization function for group 0 */
      RM_Vadc_Init_Parallel_G4();
            /* A/D converter initialization function for group 4 */
      RM_Init_Background();
            /* Initialization function for background mode of converter */
      RM_Gpt12_encoder();
            /* GPT12 unit initialization function (using for encoder) */
#ifdef Engine_9f
      RM_Gtm_Init_9f_Engine();
            /* GTM module initialization function for 9 phase engine */
      RM_PWM_Init_9f_Engine();
            /* Initialization of PWM value */
#else
      RM_Gtm_Init_3f_Engine();
            /* GTM module initialization function for 3 phase engine */
```

```
    RM_PWM_Init_3f_Engine();
        /* Initialization of PWM value */
#endif
 }
```

Každá inicializační funkce musí obsahovat povolení periferie a jejího časování, které se provádí nastavením registru CLC vybrané periferie na hodnotu $0x0$. Zápis do registru je chráněný, proto je nutné jej nejprve povolit příkazem $unlock\_wdtcon()$ a po zapsání hodnoty opět přístup zakázat příkazem $lock\_wdtcon()$. Tato posloupnost je velmi důležitá, jelikož bez jejího vykonání nebude možné periferii jakkoliv inicializovat.

5.2 Inicializace vstupně/výstupních portů

Velmi důležitou částí ovladače je správná inicializace vstupně/výstupních portů mikrokontroléru. Ve funkci RM_Port_Init je u všech používaných portů rozhodnuto, zda se bude jednat o port vstupní, výstupní, či port připravený pro testování, tzv. togglování. Soubor *port.c* obsahuje definici výše uvedené funkce, ze které si ukážeme jen pro demonstraci nastavení vždy pouze jednoho portu na vstupní či výstupní.

```
void RM_Port_Init(void)
 {
    MODULE_P00.IOCR0.B.PC2 = 0x0;
        /* Pin P00.2 to VADC7 channel 4 */
    MODULE_P02.IOCR0.B.PC0 = 0x11;
        /* Pin P00.0 to GTM output TOUT10 1X001B */
    MODULE_P14.IOCR4.B.PC5 = 0x10;
        /* Pin P14.5 to output for toggling */
 }
```

5.3 Ovladač periferie GTM

Ovladač periferie GTM slouží pro generování centrované PWM a je rozdělen na 2 části, kterými jsou:

- Ovladač pro třífázový střídač
- Ovladač pro devítifázový střídač

Mezi částmi, které se budou vykonávat je možné se přepínat za pomoci makra.
V případě použití devítifázového střídače je nutné ve vlastnostech projektu nadefinovat makro $Engine\_9f$. Pokud žádné makro není definováno je použit ovladač

pro třífázový střídač. Definice tohoto makra je nutné provést v souboru *init.c*.

Další makro je použito pro volbu deadtime. Pokud není nadefinováno makro *Manual_Deadtime* v souboru *gtm.c*, není třeba se zabývat velikostí deadtime mezi sepnutí horního a dolního tranzistoru střídače. Pokud je toto makro nadefinováno je použita část kódu s možností manuálního nastavení velikosti deadtime dle požadavků.

5.3.1 Ovladač centrované PWM pro třífázový střídač

Ovladač pro vytváření na střed centrovaného PWM signálu je složen ze 4 funkcí a 1 přerušovací rutiny.

void RM_Gtm_Init_3f_Engine(void)
Funkce pro inicializaci periferie obsahující její povolení, nastavení všech děliček hodin a povolení výstupních modulů a jejich kanálů.

void RM_PWM_Init_3f_Engine(void)
Funkce obstarávající inicializaci střídy PWM signálu, která je pro správnou funkci periferie nezbytná (inicializuje se na zanedbatelně malou hodnotu).

void RM_Calculate_3f_Duty(uint32 Duty)
Výpočet střídy PWM signálu dle aktuálních požadavků na řízení motoru. Vstupní hodnota *Duty*, která může nabývat hodnot v rozmezí $0 - 1388_H$ je dle požadavků přepočtena a uložena do pole o 3 prvcích pro 3 fáze motoru.

void ISR_GTM_3F_Engine(int i)
Přerušovací rutina volající jednu z funkcí pro nastavení hodnotu střídy. Současně je v ní pomocí makra *Manual_Deadtime* rozhodnuto, zda se bude používat manuální nebo automaticky generovaná hodnota deadtime.

void Update_Duty_3f_Engine_without_deadtime(uint32 *Duty_3Phase)
nebo
void Update_Duty_3f_Engine_with_deadtime(uint32 *Duty_3Phase)

Funkce provádí nastavení střídy PWM signálu na hodnotu vypočtenou ve funkci *void RM_Calculate_3f_Duty(uint32 Duty)*, jejíž výstupní proměnná je použita na vstupu. Dle hodnot pole se přenastaví střída výstupních kanálů TOM modulů, sloužící jako zdroje signálu pro fáze motoru. U funkce s použitím manuálního deadtime jsou k hodnotám střídy komplementárních signálů přidány hodnoty *DeadtimeOn* a *DeadtimeOff*.

Zdrojový kód ovladače pro generování třífázové PWM obsahuje soubor *gtm.c*. Deklarace funkcí a definice maker jsou obsaženy v hlavičkovém souboru *gtm.h*. Oba soubory je možné nalézt na přiloženém CD u diplomové práce.

Při použití ovladače je nejdříve spuštěna inicializační funkce periferie, při které jsou nastaveny jmenovatel a čitatel děličky systémových hodin (SYS_CLK). Dále je vybrán zdroj hodin pro TGC a nakonec zvolena frekvence pro jednotlivé kanály TOM modulů, čímž je dána frekvence čítače $CN0$.

Maximální hodnota čítače $CN0$ referenčního kanálu je zvolena jako $0x2710_H$, čemuž odpovídá perioda výstupního signálu 100 μs neboli frekvence 9,99 kHz (viz. obrázek 5.1, změřenou pomoci kurzorů).

Střída jednotlivých kanálů je počítána ve zvláštní funkci s názvem $RM\_Calculate\_3f\_Duty$, jejíž vstupním parametrem je požadovaná hodnota střídy ($Duty$). Tato proměnná může nabývat hodnot do jedné poloviny periody referenčního kanálu, což v našem případě odpovídá rozsahu $0 - 1388_H$. Ve funkci jsou do pole o 3 prvcích ($Duty\_3Phase[x]$, kde x=0 až 2), uloženy vypočtené hodnoty střídy jednotlivých kanálů.

V okamžiku, kdy čítač referenčního kanálu dosáhne maximální přednastavené hodnoty ($0x2710_H$), dojde k vynulování čítače a spuštění přerušovací rutiny. Ta obsahuje volání jedné z funkcí $Update\_Duty\_3f\_Engine\_without\_deadtime$ nebo $Update\_Duty\_3f\_Engine\_with\_deadtime$ pro změnu střídy výstupních kanálů. Vstupním parametrem je výše zmíněné pole se třemi prvky, ze kterých vypočteme hodnoty registrů $SR0$ a $SR1$ dle rovnic 5.1 a 5.2 pro automatický deadtime nebo 5.3 a 5.4 pro deadtime manuální. Význam jednotlivých registrů je popsán kapitole 2.2.3. Použitá funkce je vybrána na základě definice makra $Manual\_Deadtime$.

$$SR0 = (0.5 \cdot Period\_CH0) - Duty\_3Phase[x] \quad (5.1)$$

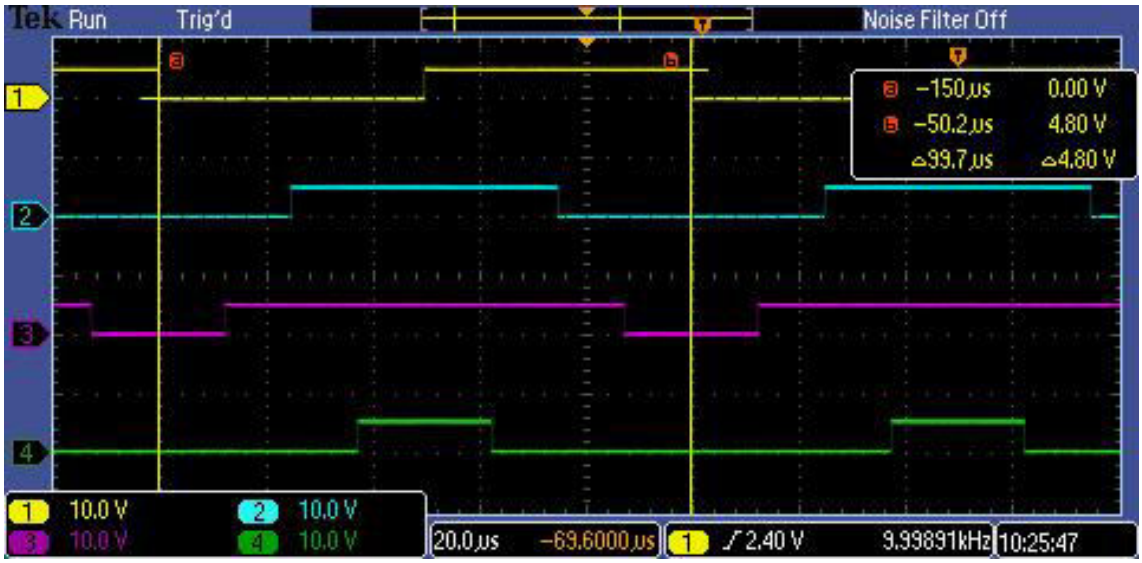
$$SR1 = (0.5 \cdot Period\_CH0) + Duty\_3Phase[x] \quad (5.2)$$

$$SR0 = (0.5 \cdot Period\_CH0) - Duty\_3Phase[x] - DeadtimeOn \quad (5.3)$$

$$SR1 = (0.5 \cdot Period\_CH0) + Duty\_3Phase[x] + DeadtimeOff[5pt] \quad (5.4)$$

kde:

$Period\_CH0$	...	Perioda referenčního kanálu $0x2710_H$
$Duty\_3Phase[x]$	...	Vypočtená hodnota střídy
x	...	Číslo fáze PWM signálu, u třífázového signálu hodnoty 0 až 2



Obr. 5.1: Centrovaná PWM pro třífázový střídač

Na obrázku 5.1 jsou zobrazeny pouze signály pro horní tranzistory střídače, to jest polovina ze signálů PWM pro třífázový střídač a signál referenční (žlutá barva). Signály pro horní tranzistory mají různé hodnoty střídy, signály pro dolní tranzistory mají střídu odpovídající hornímu tranzistoru stejné fáze, ale mají opačnou hodnotu výstupního napětí. Dále je z obrázku patrné, že amplituda všech signálů je 5 V a střída referenčního signálu je 50 % (podmínka pro vytváření centrované PWM).

Tabulka 5.1 znázorňuje využití kanálů TOM modulu u ovladače pro třífázový střídač. Při použití ovladače je nutné toto uspořádání bezpodmínečně dodržet.

Tab. 5.1: Přiřazení V/V portů signálům PWM pro třífázový střídač

Port	Použitý kanál	Využití výstupu
P00.0	TOM1_0	Referenční kanál
P00.1	TOM1_1	Fáze 1, Horní tranzistor
P00.3	TOM1_2	Fáze 1, Dolní tranzistor
P00.4	TOM1_3	Fáze 2, Horní tranzistor
P00.7	TOM1_6	Fáze 2, Dolní tranzistor
P00.8	TOM1_7	Fáze 3, Horní tranzistor
P02.0	TOM1_8	Fáze 3, Dolní tranzistor

5.3.2 Ovladač centrované PWM pro devítifázový střídač

Struktura ovladače, jeho rozčlenění a použité funkce jsou obdobné jako u ovladače pro třífázový střídač uvedený v předchozí kapitole. Malou odlišností jsou názvy funkcí, ve kterých je uvedena hodnota 9 místo 3. Větší odlišností už je inicializace a obsluha 18 výstupních kanálů, které jsou uloženy ve 3 polích o 3 prvcích namísto jednoho pole. Pole využívají funkce pro výpočet střídy a změnu hodnot registrů $SR0$ a $SR1$ výstupních kanálů. Deklaraci takové funkce můžeme vidět na následujícím řádku.

```
void Update_Duty_3f_Engine_with_deadtime(uint32 *Duty_3Phase_1,
    uint32 *Duty_3Phase_2, uint32 *Duty_3Phase_3)
```

Uspořádání kanálů ovladače je znázorněno v tabulce 5.2. Opět jako u třífázového střídače je nutné poctivě dodržet uspořádání dle tabulky. Horní a dolní tranzistory jsou popsány následujícími zkratkami:

- Top - Horní tranzistor
- Bottom - Dolní tranzistor

Tab. 5.2: Přiřazení V/V portů signálům PWM pro devítifázový střídač

Port	Použitý kanál	Využití výstupu	Port	Použitý kanál	Využití výstupu
P00.0	TOM1_0	Referenční kanál	P02.4	TOM1_12	Fáze 5, Bottom
P00.1	TOM1_1	Fáze 1, Top	P02.5	TOM1_13	Fáze 6, Top
P00.3	TOM1_2	Fáze 1, Bottom	P14.7	TOM2_0	Fáze 6, Bottom
P00.4	TOM1_3	Fáze 2, Top	P14.8	TOM2_2	Fáze 7, Top
P00.7	TOM1_6	Fáze 2, Bottom	P14.9	TOM2_3	Fáze 7, Bottom
P00.8	TOM1_7	Fáze 3, Top	P14.10	TOM2_4	Fáze 8, Top
P02.0	TOM1_8	Fáze 3, Bottom	P10.4	TOM2_6	Fáze 8, Bottom
P02.1	TOM1_9	Fáze 4, Top	P10.5	TOM2_10	Fáze 9, Top
P02.2	TOM1_10	Fáze 4, Bottom	P10.6	TOM2_4	Fáze 9, Bottom
P02.3	TOM1_11	Fáze 5, Top			

Seznam funkcí ovladače pro devítifázový střídač z důvodu malých odlišností nebudeme znovu uvádět. Funkce jsou opět obsaženy v souborech *gtm.c* a *gtm.h*. Oba soubory jsou přiloženy na CD u diplomové práce.

5.3.3 Ovladač centrované PWM pro střídače s jiným počtem fází

V případě, kdy je nutné použít motor a tudíž i střídač s jiným počtem fází, než je 3 nebo 9 je možné vhodně upravit ovladač pro devítifázový střídač. Počet fází použitého střídače však nesmí přesáhnout 9, což je maximum při použití vývojového kitu s mikrokontrolérem TC275 CA.

Potřebného počtu fází je možné dosáhnout postupným odebírání výstupních kanálů a to vždy ze všech funkcí ovladače pro devítifázový střídač. Kanály je však možné odebírat pouze od nejvyššího, to jest od TOM kanálů pro 9. fázi střídače (vždy odebrat dvojici kanálů). Pokud bychom postupně odebrali 6 fází, dostaneme ovladač naprosto totožný s tím, který je uveden v kapitole 5.3.1.

5.4 Ovladač periferie VADC

Snímání a převod vstupních signálů provádíme dvěma způsoby, kterými jsou synchronizovaný paralelní převod a převod na pozadí. Paralelní převod je při měření veličin motoru použit pro snímání proudů jednotlivými fázemi, kdežto převod na pozadí pro získání dat ze snímačů teploty či pro převádění výstupních signálů GMR snímače natočení.

Inicializace periferie je rozdělena na 3 části. Jedna část slouží k inicializaci převodu na pozadí a zbylé dvě inicializují background mód neboli převod na pozadí.

void RM_Vadc_Init_Parallel_G0(void)

Funkce pro inicializaci první části paralelního převodu, který obsahuje kanály 0 a 1 ze skupin 0 až 3. Dále tato funkce obsahuje nastavení děliček $DIVD$, $DIVA$ a především povolení hodin periferie bite $ADCCLKSEL$, který je v registru $CCUCON0$, který má chráněný přístup. Ten je možný povolit voláním funkce $unlock\_safety\_wdtcon()$ a následně zakázat funkcí $lock\_safety\_wdtcon()$.

void RM_Vadc_Init_Parallel_G4(void)

Druhá část inicializace paralelního převodu pro 0. a 1. kanál skupin 4 až 7.

void RM_Init_Background(void)

Inicializace vybraných kanálů pro převod na pozadí.

Všechny 3 funkce jsou obsaženy v souborech *vadc.c* a *vadc.h*, které jsou přiložené na CD u diplomové práce.

Periferie VADC je časována zdrojem hodin o frekvenci 100 MHz, která je po vstupu dělena dvěma děličkami, jejichž dělicí hodnoty jsou zapsány do proměnných s označením $DIVD$ a $DIVA$. První z děliček (s dělícím poměrem dle proměnné

$DIVD$) určuje frekvenci f_{VADC}, se kterou je časována arbitráž o vykonávané metodě převodu. Druhá dělička ($DIVA$) určuje frekvenci f_{ADCI} s jakou budou pracovat samotné A/D převodníky. V našem případě využíváme hodnoty $DIVD = 0$, a $DIVA = 5$, což odpovídá frekvencím $f_{VADC} = 100\,\text{MHz}$ a $f_{ADCI} = 16{,}67\,\text{MHz}$ nebo také periodě vykonávání $t_{VADC} = 10\,\text{ns}$ respektive $t_{ADCI} = 60\,\text{ns}$.

Jelikož využíváme u všech převodníků 12bitové rozlišení, máme povolenou post kalibraci a k převodu nepřidáváme žádné hodinové cykly, je doba převodu jednoho převodníku t_{12bit} stanovena dle rovnice 5.5.

$$t_{12bit} = (2 + STC + N + PC) \cdot t_{ADCI} + 2 \cdot t_{VADC} \tag{5.5}$$

kde: STC ... Přidaný čas, v našem případ 0, dle hodnoty bitu $STCS$
N ... Rozlišení, v našem případě 12 bitů
PC ... Post kalibrace, jeli zapnuta (náš případ) počítá se s hodnotou 2

Dle nastavených hodnot a použitého vzorce má výsledná doba jednoho převodu hodnotu $t_{12bit} = 980\,\text{ns}$.

Tab. 5.3: Přiřazení portů vývojového kitu kanálům A/D převodníku

	CH0	CH1	CH2	CH3	CH4	CH5	CH6	CH7
G0	AN0	AN1	AN2	AN3	X	X	X	X
G1	X	X	X	X	X	X	X	X
G2	AN16	AN17	X	X	AN20	AN21	X	X
G3	AN24	AN25	X	X	X	X	X	X
G4	AN32	AN33	X	X	X	X	X	X
G5	X	X	X	X	AN44	AN45	X	X
G6	P00.12	P00.11	P00.10	P00.9	TOM	TOM	X	X
G7	P00.6	P00.5	TOM	TOM	P00.2	TOM	X	X

Všechny použitelné porty, které je možné použít jako vstupy pro jednotlivé skupiny a kanály A/D převodníku jsou zobrazeny v tabulce 5.3, včetně jejich rozdělení mezi obě metody převodu. Červeně jsou zobrazeny čísla vstupních kanálů určených pro paralelní převod, modře pak ty pro převod na pozadí. Text TOM ve vybraných kolonkách tabulky značí, že tyto porty by se dali použít jako vstupní kanály převodníku, ale z důvodu omezeného počtu portů byl udělán kompromis a tyto porty slouží jako výstupní pro TOM moduly periferie GTM. X značí, že porty těchto kanálů nejsou na vývojovém kitu vyvedeny ani na jeden z konektorů.

Při každém vyčítání výsledku je použita funkce $VADC\_GetResult$, která přepočítává výsledek z 12 bitového čísla (rozsah 0 až 4095), na výsledek v rozsahu 0 až 5 V.

Definice této funkce umístěnou v souboru *vadc.c* je možné vidět na následujících řádcích. Jako vstup je použita hodnota z výsledkového registru a výstup je ukládán do připravená proměnné v daném rozsahu.

```
inline float32 VADC_GetResult(uint16 result)
 {
     float32 d_result
        d_result = ((float32)result ·  RANGE_0_5V) / CONVERTER_12B;
          /* Get result with selected converter mode (8/10/12 bits) in selected
             voltage range (0-3 V or 0-5V) */
     return d_result;
 }
```

Ve funkci jsou použita 2 makra: $RANGE\_0\_5V$ a $CONVERTER\_12B$, která definují rozsah výstupních hodnot a rozlišení převodníku. Tato makra je možné zaměnit za jiné z předdefinovaných v souboru *vadc.h*. Jako rozsah výstupních hodnot lze použít 0 až 3 V ($RANGE\_0\_3V$) a rozlišení převodníku lze použít ještě 10 či 8 bitové ($CONVERTER\_10B$ nebo $CONVERTER\_8B$).

5.4.1 Synchronizovaný převod dle sestupné hrany PWM

Ze dvou možností synchronizovaného převodu, které jsou uvedeny v kapitole 2.3.4 využíváme paralelní převod více kanálů. Ten je používán pro převod proudů jednotlivých fázi motoru a je spouštěn dle sestupné hrany signálu PWM vybraného kanálu. U mikrokontroléru TC275 CA je výběr kanálů TOM modulu, od kterých je možné spouštět převod omezený pouze na 4 kanály. Jedná se o kanály 6, 7, 13 a 14 TOM modulu číslo 1. V našem případě využíváme kanál 7 ($TOM1\_7$). Synchronizovaný převod proudů dle PWM jednotlivých fází motoru je pro správné řízení motorů nezbytný.

Pro paralelní převod využíváme kanály číslo 0 a 1, tudíž je možné převést až 16 kanálů. Jak je však patrné z tabulky 5.3, nejsou tyto kanály (zapsané červeně) u všech skupin dostupné. Ze skupiny č. 2 nepoužíváme žádný z kanálů, u skupiny 5 využíváme funkce *Alias*, která spočívá v možnosti nahradit 0. a 1. kanál jiným kanálem stejné skupiny. V našem případě jako náhradu používáme kanál číslo 4 (místo kanálu č. 0) a č. 5 (namísto 1. kanálu).

Dále je nutné zvolit, který kanál ze které skupiny bude zvolen jako master a který jako slave. Jelikož využíváme 2 kanály u 7 skupin je nutné vybrat 4 mastery (najednou je možné převádět maximálně 4 kanály). Jako master jsou zvoleny skupiny 0 a 4, zbylé skupiny jsou slave. U master se vybírá rozlišení a typ spuštění převodu, v tomto případě od externí události. Spouštění dle PWM se však týká pouze 0. kanálu u master skupin 0 a 4, převod kanálu číslo 1 je spuštěn okamžitě po dokončení

převodu 0. kanálu. To znamená, že po detekování sestupné hrany u PWM přichází trigger na vstupy masterů, čímž dojde k postupnému převodu nultých a následně prvních kanálů.

U každého z použitých kanálů je možné zvolit výsledkový registr, do kterého bude výsledek ukládán (používáme pouze výsledkové registry 0 a1 pro každou skupinu). Přečtení výsledku je možné dvěma způsoby popsanými v kapitole 2.3.5, u ovladače používáme vyvolání přerušení po dokončení paralelního převodu. Vyvolané přerušovací rutiny obsahují vyčtení výsledků jejich uložení do připravených proměnných. Jsou vytvořeny 2 rutiny ISR_VADC_00 a ISR_VADC_40, když první slouží pro uložení výsledků z kanálů 0 a 1 skupin 0 až 3 a druhá pro vyčtení výsledků skupin 4 až 7.

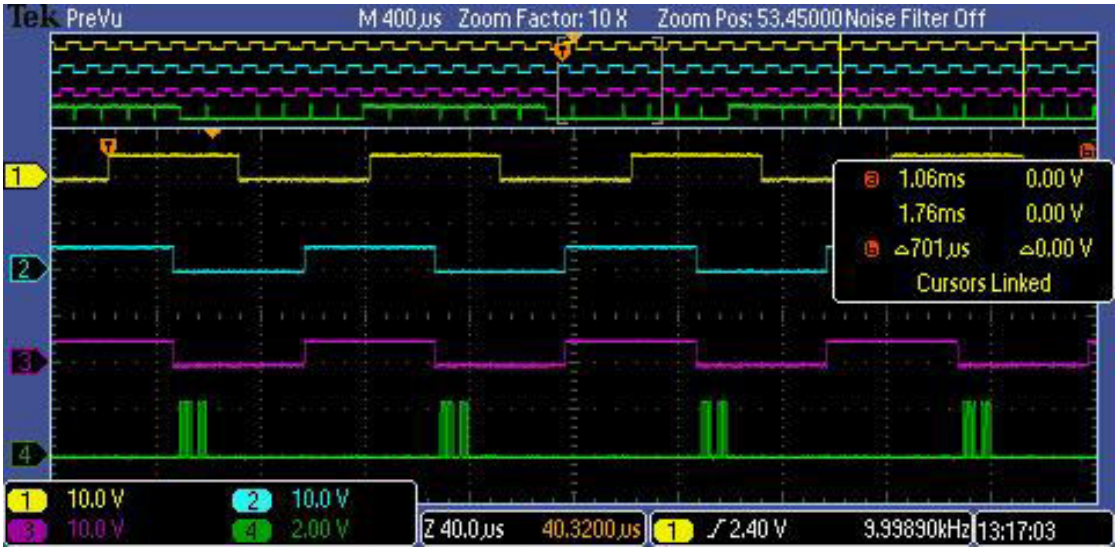
Po provedení synchronizace se signálem PWM a jeho spuštění dojde k vyvolání přerušení a vyčítání výsledků a jejich uložení do připravených proměnných. Dle manuálu mikrokontroléru [4] je dostačující v rutině nastavit flag odpovídající události, která vyvolala přerušení zpět na hodnotu logické 0 (pomocí příkazu $VADC\_GxSEFCLR.B.SEV0 = 1$, kde x značí číslo skupiny 0 nebo 4) a mělo by dojít k jedinému vyčtení výsledku. Další přerušení by se mělo vyvolat až za další periodu signálu PWM. Po zobrazení časů vyčítání výsledkových registrů pomocí tzv. togglingu[1] jsme však zjistili, že to neplatí a k vyvolávání přerušení dochází neustále, viz obrázek 5.2. [1]

Obr. 5.2: Časový průběh vyčítání výsledků bez kontroly VF bitu

Z tohoto důvodu jsme museli navíc přistoupit ke kontrole bitů všech výsledkových registrů detekujících existenci nového výsledku ve výsledkovém registru s označením

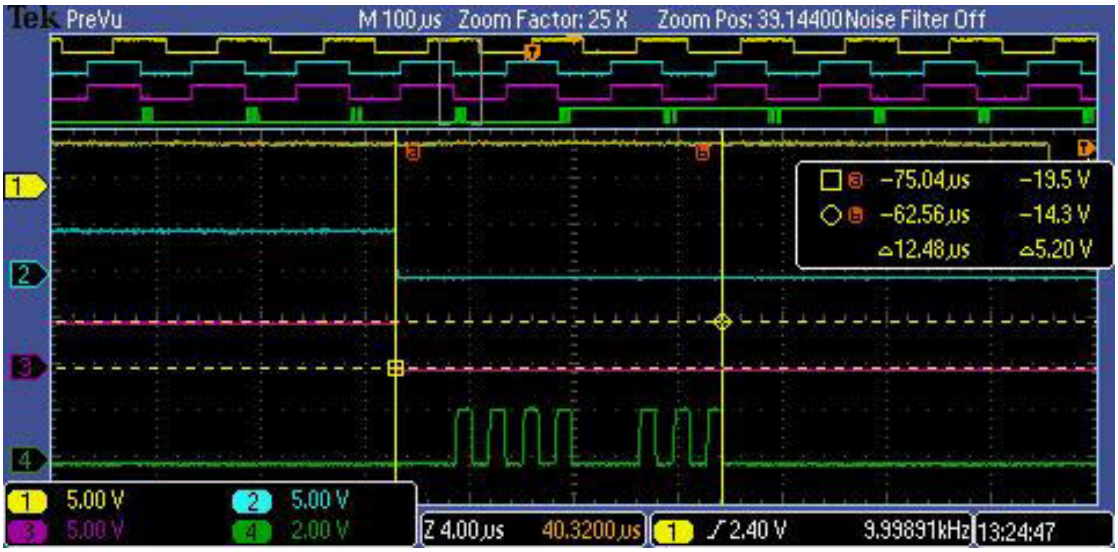
[1]Změna úrovně na pinu 14.5 mikrokontroléru po načtení příkazu $P14\_OMR.U = 0x00200020$, který je umístěn před vyčtení výsledku

VF. Pokud je bit v logické 1, je dostupný nový výsledek, kontrolou tohoto bitu dojde zároveň k vyčtení a nastavení na logickou 0. Pokud použijeme oba způsoby, to jest kontrolu VF bitů i nastavení bitu $SEV0$ začne volání přerušovací rutiny fungovat správně, viz obrázek 5.3. Je patrné, že k vyčtení výsledků dochází jednou za periodu signálu PWM, tak jak by správně mělo.



Obr. 5.3: Časový průběh vyčítání výsledků paralelních převodů s kontrolou VF bitu

Detailní časový průběh jednoho z převodů si můžeme blíže prohlédnout na obrázku 5.4. Je patrné, že převod spolu s vyčtením všech výsledků trvá $12,48\mu$s. Převod obou kanálů paralelního převodu trvá $2,24\mu$s, jako první je vyčítáno 8 výsledků ze skupin 4 až 7 (doba mezi spuštěním dvou převodů je 640 ns). Před vyčtení výsledků ze zbylých 3 skupin (6 výsledků) je časová prodleva způsobená ukládáním parametrů přerušení a následným spouštění druhé přerušovací rutiny.



Obr. 5.4: Časový průběh jednoho paralelního převodu

Deklarace přerušovacích rutin obsažených v souboru *vadc.h* je následující:

void ISR_VADC_00(int i)

void ISR_VADC_40(int i)

5.4.2 Převod vstupních signálů na pozadí, background mód

Pro převod signálů, které není nutné synchronizovat s PWM, využíváme tzv. background mód. Převod je vykonáván v okamžicích, ve kterých není třeba převádět naplánované úlohy, či úlohy s vyšší prioritou. Na pozadí jsou převáděny vybrané kanály ze čtyř skupin, viz tabulka 5.3 modře vyznačené porty. Inicializace módu spočívá ve výběru kanálů z každé skupiny určených pro převod a přiřazení odpovídajících výsledkových registrů.

Vyčítání výsledku je možné pomocí přerušení, které je vyvoláno po detekci nového výsledku pomocí bitu VF. V tomto případě docházelo v projektu k přerušování paralelního převodu právě vyčítáním výsledků z background módu. Časový průběh vyčítání výsledků sledovaný pomocí togglování na osciloskopu byl v takovém případě velmi podobný tomu na obrázku 5.2.

Z tohoto důvodu jsme se rozhodli vytvořit funkci, pomocí které budou výsledky vyčítány pouze na základě jejího zavolání. Deklarace funkce je následující:

void RM_Read_Background(void)

Funkce obsahuje vyčtení výsledků ze všech registrů, které byly nadefinovány jako výsledkové v inicializační funkci *RM_Init_Background*. Bit VF není potřeba kontrolovat, jelikož převod je vykonáván neustále kromě doby, kdy je převáděn paralelní převod.

5.5 Ovladač periferie GPT12

Ovladač se skládá ze 3 funkcí, kterými jsou:

void RM_Gpt12_encoder(void)

Jedná se o inicializační funkci, obsahující nastavení děliček hodin bloků $GPT1$, $GPT2$ a výpočet konstant potřebných pro výpočet rychlosti

void ISR_GPT_ENCODER(int i)

Přerušovací rutina spouštěná při přetečení časovače $T2$, to jest každé $2,62\,ms$

void Update_Speed(sint32 newposition,sint32 rawposition)

Funkce pro výpočet aktuální rychlosti otáčení enkodéru

Všechny 3 funkce jsou definovány v souboru $gpt12.c$ a deklarovány v $gpt12.h$, oba soubory jsou umístěny na CD u diplomové práce.

Ovladač pro zpracování dat z enkodéru využívá principy popsané v kapitole 2.4.1. Data jsou zpracovávána dvěma způsoby, mezi kterými je přepínáno na mezi, kterou jsme zjistili experimentálně a je zadána v otáčkách za minutu jako makro $Speed\_Limit\_RPM$. Následně je mez z otáček za minutu přepočítána na pulzy za periodu časovače $T2$ jako proměnná s názvem $Speed\_Limit\_ticks$.

U každého z principů je pro výpočet rychlosti potřeba výpočetní konstanta, které jsou nazvané $Pulse\_Count\_Constant$ pro metodu čítání pulzů a $Time\_Diff\_Constant$ pro měření doby mezi pulzy. Hodnoty těchto konstant jsou závislé na parametrech enkodéru (počet pulzů na otáčku), způsobu připojení enkodéru k mikrokontroléru (použití jednoho vstupu nebo dvou) a frekvenci jednotlivých bloků periferie GPT12. Jsou určeny dle rovnic 5.6 a 5.7.

$$Time\_Diff\_Constant = \frac{2 \cdot PI}{Encoder\_Resolution \cdot 4 \cdot Period\_GPT2} \tag{5.6}$$

$$Pulse\_Count\_Constant = \frac{2 \cdot PI \cdot 400}{Encoder\_Resolution \cdot 4} \tag{5.7}$$

kde:

$Encoder\_Resolution$	...	Rozlišení enkodéru (1024 pulzů na otáčku)
$Period\_GPT2$	...	Perioda časovače T5 (1,31 ms)

Rozlišení enkodéru je v obou případech násobeno hodnotou 4, jelikož pro připojení využíváme oba dva signály enkodéru (A a B), u kterých pracujeme se vzestupnou i sestupnou hranou vstupního signálu.

Při měření doby mezi dvěma pulzy získáme hodnotu časovače $T5$, která je ukládána do registru $CAPREL$. Pokud tuto hodnotu násobíme konstantou $2 \cdot \pi \cdot Time\_Diff\_Constant$ získáme rychlost otáčení v rad/s. Pro získání rychlosti

v ot/min je třeba provést výpočet dle rovnice 5.8.

$$Speed = \frac{RADS\_TO\_RPM \cdot Time\_Diff\_Constant}{GPT120\_CAPREL.B.CAPREL} \tag{5.8}$$

kde:

$RADS\_TO\_RPM$	...	Konstanta pro přepočet rychlosti otáčení z rad/s na ot/min,která má hodnotu 9.5493
$GPT120\_CAPREL.B.CAPREL$	...	Obsah $CAPREL$ registru

Druhá metoda je založena na vytváření rozdílu počtu pulzů v závislosti na směru otáčení dvou po sobě následujících period časovače $T2$. Tato diference (v kódu označena $diff$) dle vzorce 5.10 může nabývat hodnot 0 až 1024. V případě, že během čítání dojde k detekci mechanické nuly enkodéru (signál Z přivedený na vstup $T4IN$) a tudíž k vynulování čítače $T5$ je třeba výpočet diference pozměnit a to navíc v závislosti na směru otáčení. Při čítání směrem nahoru je výpočet prováděn dle rovnice 5.10, v opačném případě dle rovnice 5.11. Z takto vypočtené diference následně určíme rychlost otáčení v ot/min podle rovnice 5.12.

$$diff = newposition - rawposition \tag{5.9}$$

$$diff = newposition + (1024 - rawposition) \tag{5.10}$$

$$diff = rawposition + (1024 - newposition) \tag{5.11}$$

$$Speed = RADS\_TO\_RPM \cdot diff \cdot Pulse\_Count\_Constant \tag{5.12}$$

kde:

$newposition$	...	Nová pozice enkodéru
$rawposition$	...	Předchozí pozice enkodéru
$diff$	...	Diference mezi předchozí a současnou pozicí enkodéru (v pulzech)

U obou metod je směr a tedy znaménko výsledné rychlosti otáčení zjišťován pomocí bitu $T3RDIR$, z registru $GPT120\_T3CON$, který je určován dle posloupnosti hodnot na vstupech $T3IN$ a $T3EUD$.

6 ANALÝZA KVALITY MĚŘENÍ SIGNÁLŮ ZE SNÍMAČE TLE5009

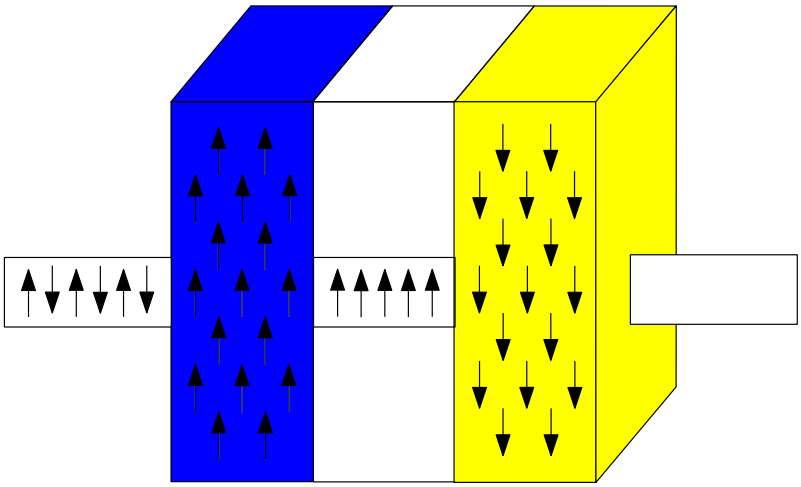
6.1 Seznámení se se snímačem TLE5009

Jedná se o snímač úhlu natočení, využívající GMR principu, kdy jednotlivé GMR elementy jsou umístěny do měřícího můstku, čímž je dosaženo zvýšení citlivosti měření. Výstupem snímače jsou čtyři napěťové signály ve tvaru sinusového a kosinového harmonického signálu.

6.1.1 GMR jev

GMR [11] jev je založen na změně elektrického odporu v závislosti na velikosti působícího magnetického pole. Popis samotného principu je však složitější, jelikož pracuje na úrovni nanovrstev a jednotlivých atomů. Pro vznik GMR jevu je třeba struktura složená z feromagnetických slitin, tenkých jen několik nanometrů, které jsou odděleny ještě tenčí nemagnetickou, avšak elektricky vodivou vrstvou (nejčastěji je používána měď). Elektrický odpor tohoto materiálu je při vrstvě jen několika atomů výrazně závislý na spinu (orientaci) elektronů. V případě, že vystavíme celou strukturu působení vnějšího magnetického pole, spiny ve feromagnetických vrstvách se vyrovnají a tím se odpor struktury změní.

Lépe je tento jev možné vysvětlit na úrovni atomů. Zatímco elektrický odpor kovů je závislý na tom, zda je cesta pro elektrony volná, u struktury GMR závisí na orientaci spinů elektronů (nahoru nebo dolů) vůči magnetickému momentu magnetické vrstvy. U elektricky vodivých nemagnetických materiálů je stejný počet elektronů se spinem nahoru i dolů, avšak u feromagnetických materiálů se mohou elektrony volně pohybovat jen tehdy, jeli jejich spin souhlasný se směrem magnetického momentu v materiálu. V opačném případě jsou elektrony brzděny a v materiálu se mohou pohybovat jen velmi obtížně.

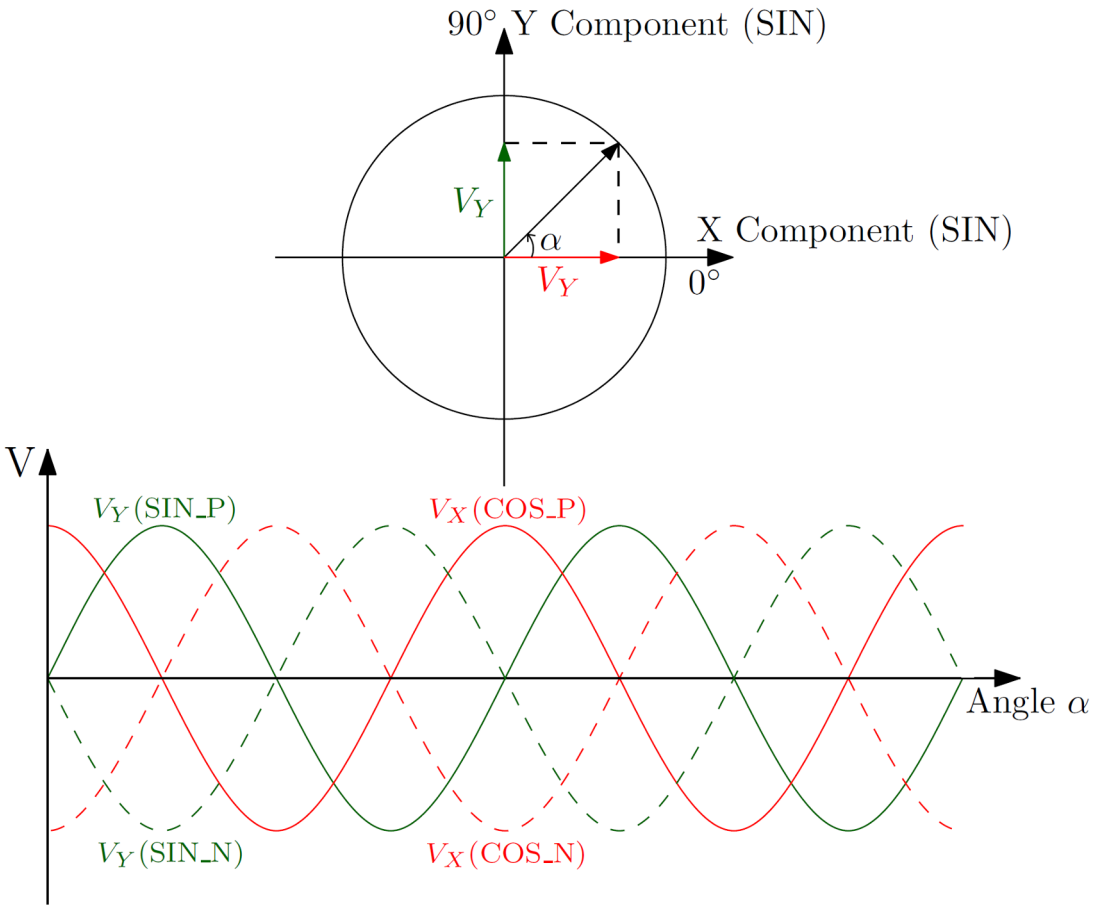


Obr. 6.1: Princip GMR jevu, převzato z [11]

Pokud výše uvedený popis použijeme na GMR strukturu, dostaneme výsledek uvedený na obrázku 6.1. Ke struktuře proudí stejné množství elektronů (počet spinů nahoru a dolů je si roven), přes první feromagnetickou desku projdou jen elektrony se spinem stejným jako má tato deska. Působí-li na druhou feromagnetickou desku opačné magnetické pole, nežli na desku první, pak elektrony touto deskou neprojdou. Při průchodu elektronů první deskou (modrá) je elektrický odpor nízký, avšak při průchodu druhou (žlutá) je již elektrický odpor materiálu vysoký. V případě dle obrázku 6.1 by se na výstupu struktury neměly objevit téměř žádné elektrony, neboli odpor je vysoký.

6.1.2 Snímač TLE5009

Jedná se o snímač úhlového natočení fungující na principu GMR jevu. TLE5009 [9] obsahuje čtyři GMR elementy uspořádané do plného Wheatstonova můstku, jehož výstupy jsou dvě dvojice signálů ve tvaru sinusového a kosinusového průběhu (vždy pozitivní a negativní) viz obrázek 6.2, které jsou taktéž i výstupy samotného snímače. Použitím plného můstku je dosaženo zdvojnásobení amplitudy výstupního signálu a potlačení vlivu měnící se teploty na měření. Amplituda výstupních signálů je 3,3 V, stejně tak lze pro napájení snímače použít napětí 3,3 V nebo 5 V.



Obr. 6.2: Ideální výstup ze snímače TLE5009, převzato z [9]

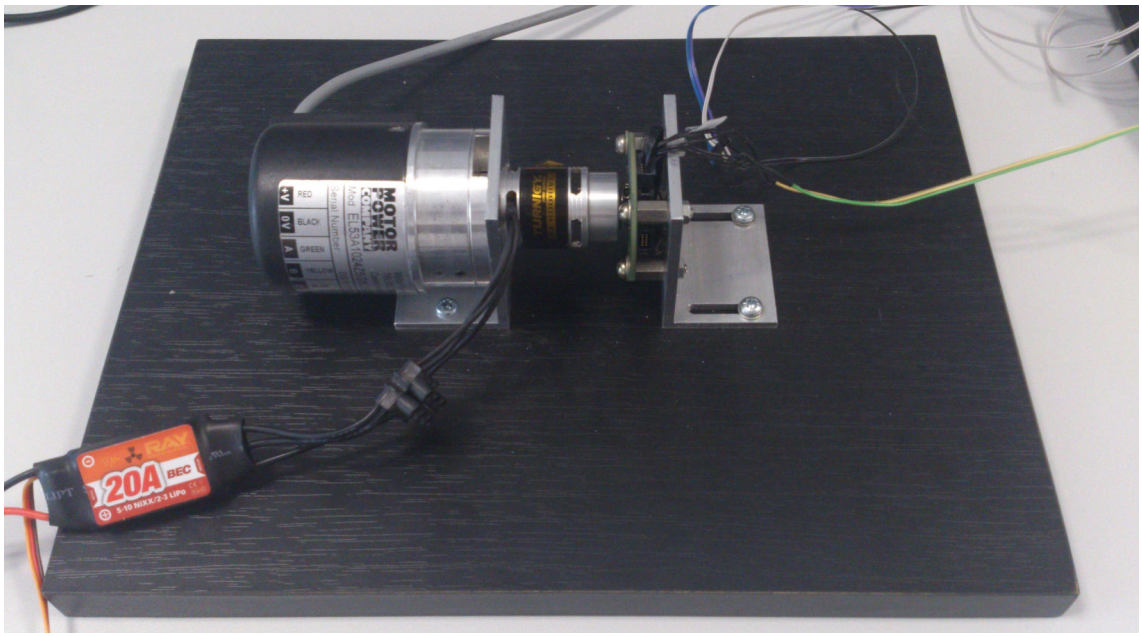
Skutečné výstupy, se však od těch na obrázku 6.2 liší, jelikož nemají ideální sinusový či kosinusový průběh, stejně tak amplituda jednotlivých signálů může být různá. Z tohoto důvodu musíme sáhnout ke kompenzaci parametrů snímače.

Pro zjištění posunutí mezi výstupními signály (sinus a kosinus) ve stupních využíváme trigonometrickou funkci Arkus tangens, dle vztahu 6.1.

$$\alpha = \arctan(\frac{V_Y}{V_X}) \tag{6.1}$$

6.2 Přípravek pro sběr dat

Pro sběr dat byl vytvořen speciální přípravek (viz obrázek 6.3), který je složený z DC motoru na jehož jedné straně je umístěn enkodéru, který slouží pro porovnávání dat a na druhé je umístěn magnet pro snímače TLE5009. Snímač je od otáčejícího se magnetu umístěn ve vzdálenosti přibližně 2 mm, což je pro vyvolání a možnost měření GMR jevu vzdálenost naprosto dostačující, na druhou stranu je zajištěn dostatečný odstup mezi pevně upevněnou a pohybující se částí přípravku.



Obr. 6.3: Přípravek pro sběr dat ze senzoru TLE5009

Pro řízení DC motor je generováno obdélníkové napětí pomocí generátoru signálů, které je připojeno na regulátor, která byl zakoupen společně s motorem. Generovaný signál má amplitudu $5\,V$ a periodu $20\,ms$, přičemž měníme jeho střídu čímž dochází ke změně otáček motoru.

7 SBĚR DAT

7.1 Sběr dat ze senzoru

Pro sběr dat z přípravku využíváme program vytvoření v prostředí LabView s využitím systému CompactRio a dvou jeho modulů. Pro snímání dat z enkodéru je využíván modul NI-9505 (obsahuje patici pro připojení enkodéru a jeho napájení). Pro sběr dat ze snímače TLE5009 využíváme modul NI-9215, což je analogový vstupní modul pro rozsah napětí $\pm 10\,\mathrm{V}$. Jsou používány pouze dva vstupní kanály, jelikož pracujeme pouze s pozitivními signály sinus a kosinus a tím pádem je délka převodu $6\,\mu\mathrm{s}$.

Program pro sběr dat a vytváření PWM signálu vytvořený v prostředí LabView se skládá ze dvou navzájem propojených částí:

- FPGA
- Real-time

7.1.1 FPGA

Nejdůležitější částí je synchronizace signálů sinus a kosinus z GMR snímače a signálu z enkodéru. Takto synchronizovaná data jsou následně za pomoci FIFO fronty posílána do Real-time části programu. Poslední částí kódu je výpočet parametrů $M45$ a $M135$, což jsou vypočtené hodnoty úhlu φ natočení motoru o 45°, respektive 135°. Pro výpočet hodnot $M45$ a $M135$ je nutné nejprve zjistit minimální a maximální hodnotu sinu a kosinu. Výpočet probíhá dle vzorců uvedených v rovnicích 7.1 a 7.2.

$$M_{45} = \sqrt{X_{45}^2 + Y_{45}^2} \tag{7.1}$$

$$M_{135} = \sqrt{X_{135}^2 + Y_{135}^2} \tag{7.2}$$

kde: X_{45} a X_{135} ... Kosinusové hodnoty při 45° a 135°
Y_{45} a Y_{135} ... Sinusové hodnoty při 45° a 135°

Výše uvedené parametry jsou počítány pouze tehdy, je-li splněna 1. podmínka pro $M45$ nebo 2. podmínka pro $M135$.

1. $(Y - X) \cdot (Y_{(-1)} - Y_{(-1)}) < 0$
2. $(Y + X) \cdot (Y_{(-1)} + Y_{(-1)}) < 0$

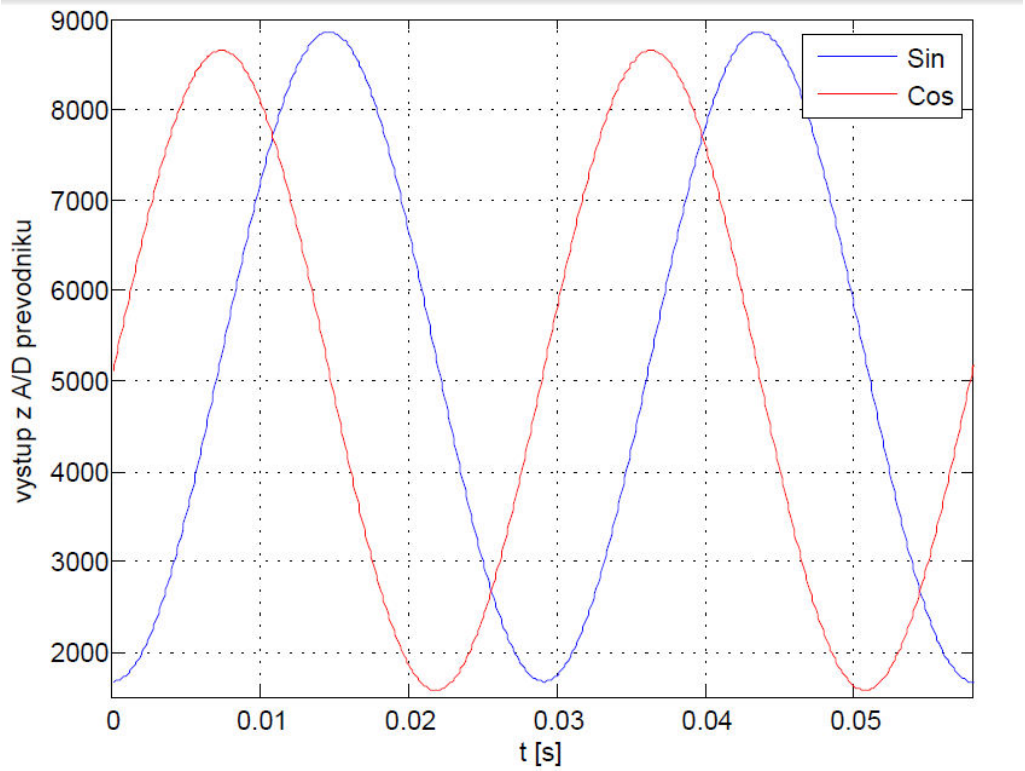
kde: X a Y ... Aktuální kosinusová a sinusová hodnota
$X_{(-1)}$ a $Y_{(-1)}$ ... Předchozí kosinusová a sinusová hodnota

7.1.2 Real-time

Real-time část kódu slouží pouze k uložení dat z FIFO fronty ve formátu textového dokumentu, který je vhodný pro další zpracování v programu Matlab. Data jsou ukládána do sloupců, které jsou odděleny tabulátory, v následujícím pořadí:

- Časový vektor
- Data z GMR snímače (kosinus)
- Časový vektor
- Data z GMR snímače (sinus)
- Časový vektor
- Data z enkodéru

Samotné uložení probíhá manuálně pomocí prvku *chart*, neboli graf, který slouží pro vykreslování dat.



Obr. 7.1: Výstupní data ze snímače TLE5009 (po uložení pomocí prvku chart)

Z obrázku 7.1 je patrné, že průběhy nemají ideální tvar, jak již bylo zmíněno v kapitole 6.1.2. Z toho důvodu je nutné nejprve data vhodně upravit a dále s nimi pracovat.

7.2 Postup získávání dat

Pro správnou kompenzaci je nutné dodržet následující postup získávání dat z přípravku:

1. Ručním otáčením motoru (přibližně 2 až 3 otáčky jedním směrem a následně směrem opačným), čímž dostaneme data pro statickou kompenzaci snímače. Takto získaná data nazýváme Statická data.
2. Nastavením určitých otáček motoru pomocí zdroje napětí, či jejich změnou během získávání dat, čímž získáme data pro dynamickou kompenzaci parametrů snímače. Takto naměřeným datům říkáme data Dynamická.

8 ZPRACOVÁNÍ DAT ZE SNÍMAČE TLE5009

Pro analýzu dat byl vytvořen program v prostředí Matlab Simulink. Je rozdělen do několika částí, které musí bát vykonány v následujícím pořadí:

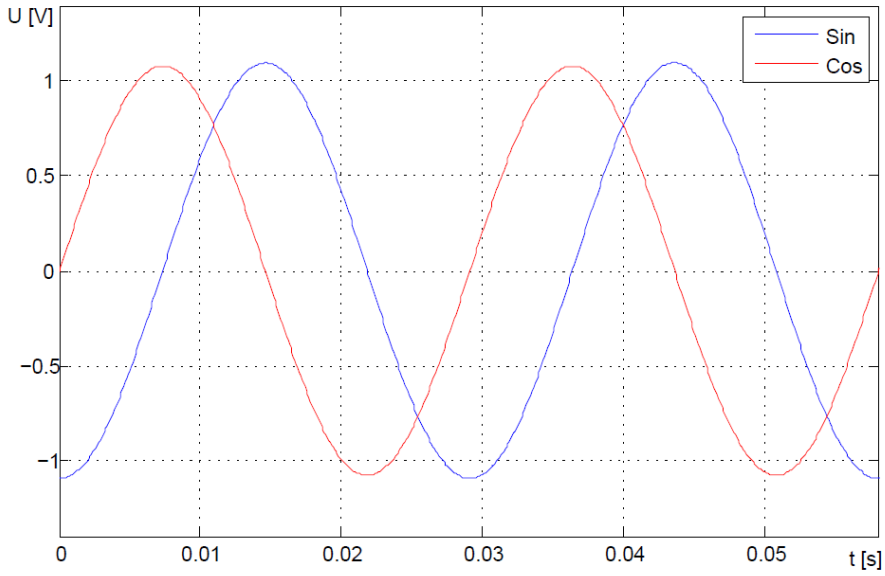
1. Úprava dat do vhodného formátu
2. Statická kompenzace parametrů snímače
3. Dynamická kompenzace parametrů snímače
4. Porovnání dat po kompenzaci s daty z enkodéru

V následujících kapitolách si jednotlivé části programu popíšeme podrobněji.

8.1 Úprava dat do vhodného formátu

Data ze snímače TLE5009 získané pomocí programu LabView a systému CompactRio je třeba před dalším použitím vhodně upravit. Část kódu, obsažená v příloze A obsahuje úpravu statických dat ze snímače TLE5009, úprava dynamických dat je naprosto totožná.

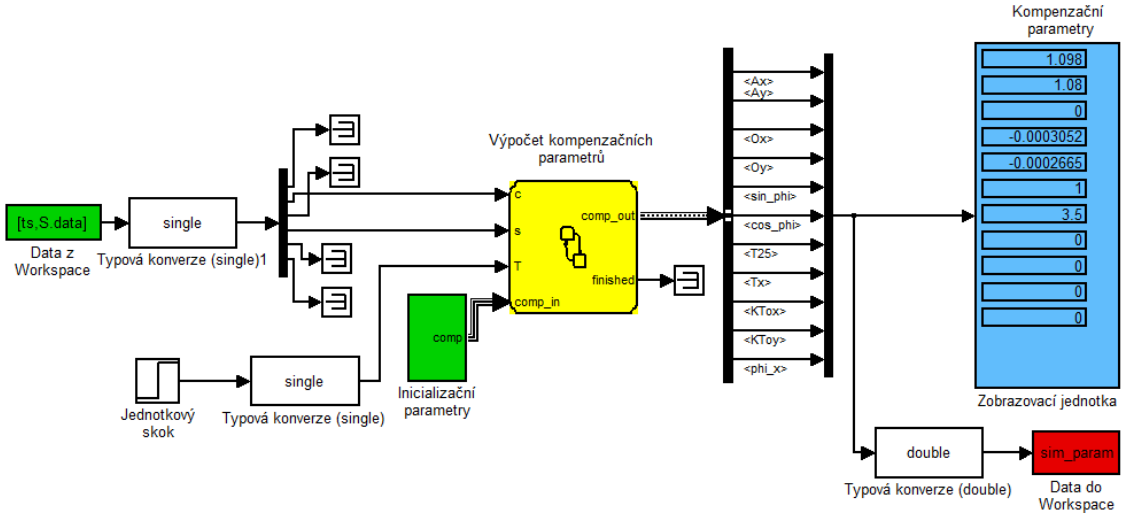
Na obrázku 8.1 jsou zobrazena výstupní data snímače již po úpravě. Je patrné, že stále neodpovídají normalizovaným průběhům sinu a kosinu (amplituda je větší, než hodnota 1 a průběhy nemají nulovou střední hodnotu). Z těchto důvodů je nutné provést kompenzaci, díky čemuž dosáhneme normalizovaných průběhů.



Obr. 8.1: Výstupní data ze snímače TLE5009 po úpravě v programu Matlab

8.2 Zjištění statických kompenzačních parametrů

Zjištění statických kompenzačních parametrů provádíme pomocí schématu, které je zobrazeno na obrázku 8.2.



Obr. 8.2: Blokové schéma pro zjištění statických kompenzačních parametrů

Data, které jsou načítána z *Workspace*, jsou upravená dle postupu uvedeného v kapitole 8.1. Blok *Výpočet kompenzačních parametr*ů obsahuje výpočty dle rovnic 7.1 a 7.2, dále pak výpočty amplitud (ukázka výpočtu pro sinus uveden v rovnici 8.1, pro kosinus je výpočet totožný), offsetu jednotlivých signálů (rovnice 8.2 pro sinus), úhlu posunutí signálů φ (rovnice 8.3) a nakonec výpočty sinu a kosinu tohoto úhlu.

$$A_Y = \frac{Y_{max} - Y_{min}}{2} \tag{8.1}$$

$$O_Y = \frac{Y_{max} + Y_{min}}{2} \tag{8.2}$$

$$\varphi = 2 \cdot \arctan\left(\frac{M_{135} - M_{45}}{M_{135} + M_{45}}\right) \tag{8.3}$$

$$sin\_phi = \sin(\varphi) \tag{8.4}$$

$$cos\_phi = \cos(\varphi) \tag{8.5}$$

8.3 Dynamická kompenzace parametrů snímače

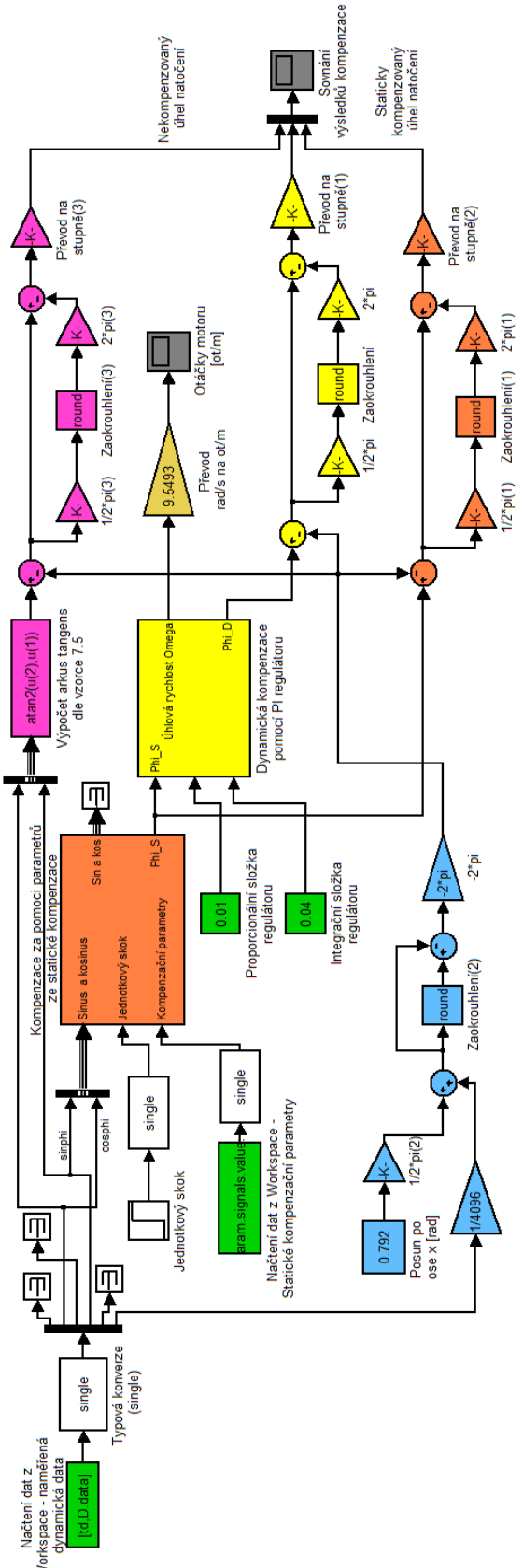
Na obrázku 8.3 můžeme vidět schéma dynamické kompenzace. Jako vstupní data používáme statické kompenzační parametry získané v předchozí kapitole (8.2) a dynamická data, ze kterých však používáme pouze tři sloupcové vektory, odpovídajícím datům GMR snímače (sinusový a kosinusový signál) a enkodéru. Data

z GMR snímače slouží společně se statickými parametry jako vstupní data pro blok *Kompenzace za pomoci parametrů ze statické kompenzace*. Výstupem mu je staticky kompenzovaný úhel φ_S, který společně se složkami PI regulátoru slouží jako vstup do bloku *Dynamická kompenzace pomocí PI regulátoru*. Po provedení tohoto bloku již dostáváme dynamicky kompenzovaný úhel φ_D, který následně porovnáváme s výstupem enkodéru.

Data z enkodéru je třeba před porovnáním s kompenzovaným úhlem φ upravit do vhodného tvaru. Surová data jsou z důvodu 16-ti bitového datového typu integer v rozsahu ± 32768 a reprezentují natočení motoru v radiánech. Abychom dosáhli tvaru totožného s výstupem dynamické kompenzace je třeba zmenšil amplitudu, či posunout celý průběh po ose x. Úprava dat z enkodéru je na obrázku 8.3 vyznačena světle modrou barvou.

Pro grafické porovnání nekompenzovaných dat s těmi již kompenzovanými je třeba, aby byly upraveny do stejného tvaru (v obrázku 8.3 zobrazeno růžovou barvou). Toho dosáhneme, pokud vypočteme ze surových dynamických dat (sinus a kosinus) úhel natočení dle vzorce 8.6. Následně z takto vypočteného úhlu φ vytvoříme diferenci s úhlem získaným z enkodéru. Další úprava takto vzniklého rozdílu je popsána v kapitole 8.4. Takto upravená data je již možné porovnávat s daty kompenzovanými.

Na výstupu bloku *Dynamická kompenzace pomocí PI regulátoru* jsou úhlová rychlost ω v jednotkách rad/s a dynamicky kompenzovaný úhel φ_D. Pokud chceme proměnnou ω převést na otáčky/minutu musíme ji násobit hodnotou $9,549$, což je konstanta pro převod těchto jednotek.

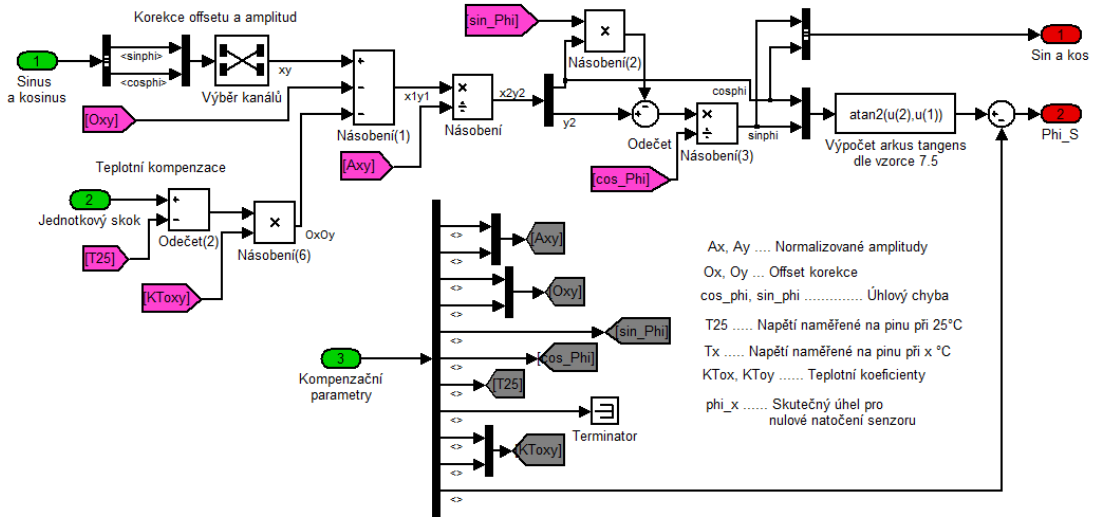


Obr. 8.3: Blokové schéma dynamické kompenzace

8.3.1 Blok kompenzace za pomoci parametrů ze statické kompenzace

V tomto bloku pracujeme s parametry, které jsme získali během statické kompenzace popsané v kapitole 8.2. Z těchto získaných parametrů, však využíváme pouze některé, a to $Offset$, $Amplitudu$, sin_phi a cos_phi. Tyto vybrané parametry používáme dle schéma na obrázku 8.4 k úpravě dynamických dat. Po provedení všech úprav vypočteme arctan 2, dle rovnice 8.6, čímž dostaneme úhel posunutí φ, se kterým dále pracujeme.

$$\varphi = arctan2\left(\frac{sin\_phi}{cos\_phi}\right) \tag{8.6}$$

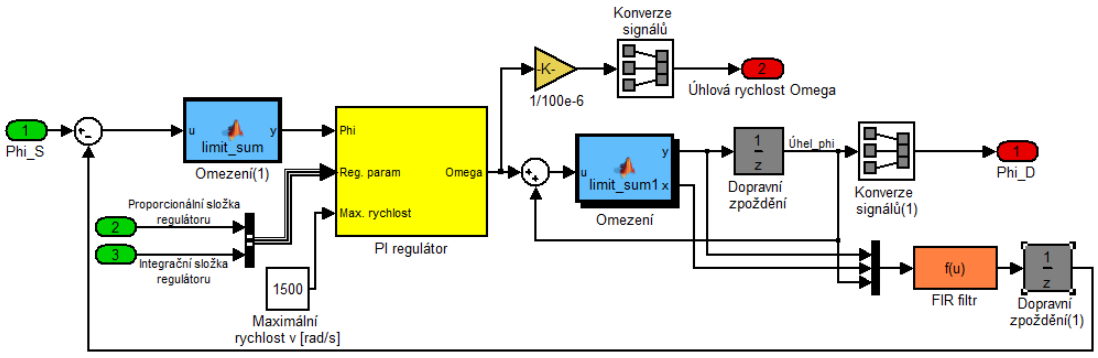


Obr. 8.4: Vnitřní schéma bloku Kompenzace za pomoci parametrů ze statické kompenzace

8.3.2 Blok dynamické kompenzace

Ve schématu na obrázku 8.5 vidíme vnitřní schéma bloku *Dynamická kompenzace pomocí PI regulátoru*, jehož nejdůležitější částí je PI regulátor. Vstupem mu jest úhel posunutí φ_S, omezený na rozsah $\pm\pi$, proporcionální a integrační složka regulátoru. Na výstupu z regulátoru poté získáváme úhlovou rychlost ω, která nám po vynásobení hodnotou $1/1 \cdot 10^{-6}$ (perioda vzorkování) reprezentuje úhlovou rychlost otáčení motoru v jednotkách rad/s.

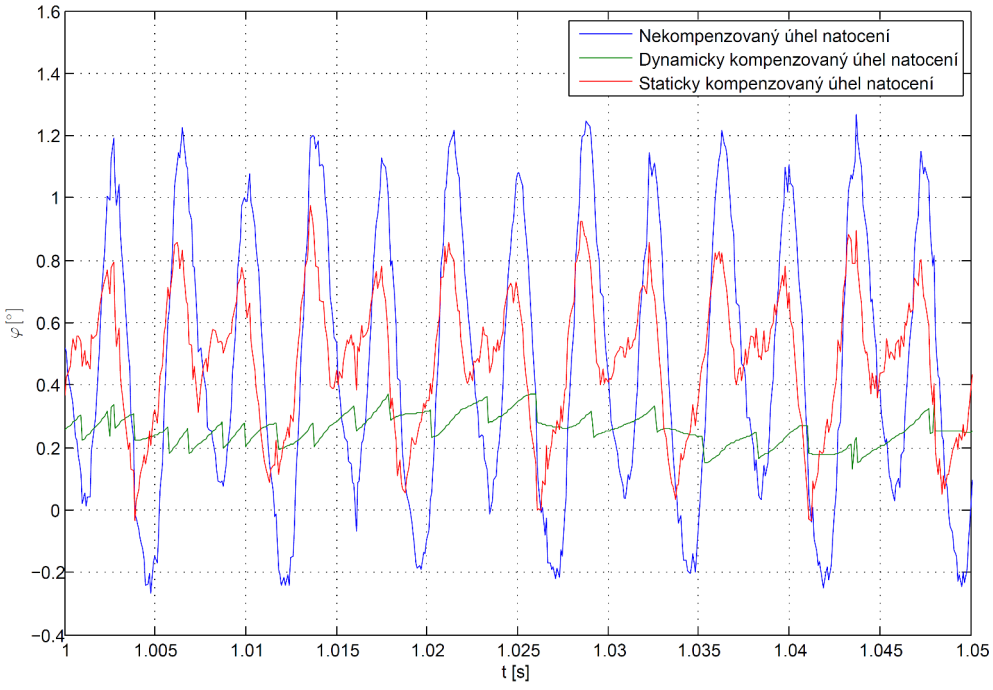
Pokud výstup z PI regulátoru opět omezíme na $\pm\pi$, zpozdíme o jeden krok a přivedeme na vstup FIR filtru 2. řádu a následně přes další zpožďovací člen přivedeme zpět na vstup regulátoru jako zápornou zpětnou vazbu dostaneme na výstupu prvního členu *Dopravní zpoždění* již dynamicky kompenzovaný úhel natočení φ_D.



Obr. 8.5: Vnitřní schéma bloku Dynamická kompenzace pomocí PI regulátoru

8.4 Úprava dat pro porovnání

Jelikož dynamicky kompenzovaný úhel φ_D (z kapitoly 8.3.2) získáváme v radiánech, proto je možné okamžitě vytvořit diferenci s upravenými daty z enkodéru (viz zobrazovací prvek *Zobrazení datyz dynamické kompenzace a enkodéru*). Takto vytvořený rozdíl obsahuje pulzy způsobené zpožděním jednoho ze signálů, a které je třeba odstranit. Pulzy se projeví jen při malém množství ze skokových změn signálu z hodnoty $-\pi$ na $+\pi$, kdy je signál z enkodéru posunut o jeden impulz oproti signálu z GMR snímače. Odstranění těchto pulzů je na obrázku 8.3 umístěno za vytvoření diference úhlů a před blok *Převod na stupně*(1).



Obr. 8.6: Diference úhlů natočení

Na obrázku 8.6 je zobrazena část grafu, která obsahuje diference úhlů natočení (úhel získaný z enkodéru a nekompenzovaný/kompenzovaný úhel z GMR snímače). Je vidět, že již statickou kompenzací dochází ke zmenšení diference oproti té, která je počítána z nekompenzovaného úhlu posunutí signálů (sinus a kosinus).

Požitím dynamické kompenzace dojde k dalšímu, tentokrát však daleko výraznějšímu poklesu diference. Pokud by jsme tyto diference měli vyjádřit numericky, tak by to bylo přibližně $1,5\,^\circ$ při počítání úhlu bez jakékoliv kompenzace, dále pak přibližně $1\,^\circ$ po vypočtu ze staticky kompenzovaných dat. Diference vypočtena z dynamicky kompenzovaného úhlu φ_D se pohybuje okolo hodnoty $0,4\,^\circ$. Z těchto čísel je patrné, že jsme dosáhli zlepšení v přesnosti zjištění úhlu natočení pomocí GMR snímače přibližně o 70%, oproti nekompenzovaným datům.

9 ZÁVĚR

Hlavním cílem první části diplomové práce bylo seznámit se s problematikou řízení elektrických motorů pomocí mikrokontroléru Infineon TriCore TC275 CA. Pro řízení motorů je nejvýhodnější použít pulzní šířkovou modulaci neboli PWM, která spočívá ve změně střídy signálu, čímž dosáhneme změny vstupního napětí motoru a tím i změnu otáček motoru.

AURIX TriCore TC275CA je tříjádrový mikrokontrolér, primárně vyvinutý pro použití v automobilovém průmyslu. Obsahuje velké množství periferií, my však využíváme pouze část z nich a to periferie GTM, VADC a GPT12, které jsme si v jednotlivých kapitolách podrobněji popsali. Mikrokontrolér je v našem případě umístěn na vývojovém kitu, který je propojen s PC pomocí rozhraní USB. Pro programování mikrokontroléru využíváme prostředí Tricore Development Platform, které je složeno ze 3 částí. První část je určena pro vytváření kódu, druhou částí je debugger, neboli ladící nástroj a poslední částí je Memtool, který slouží pro propojení PC a vývojového kitu a nahrání vytvořeného kódu do mikrokontroléru.

Postupně byly vytvořeny ovladače pro jednotlivé periferie, jako první byl vytvořen ovladač pro třífázový a devítifázový PWM signál. Dále ovladač pro A/D převodník s rozlišením 12 bitů, který se skládá ze dvou částí a to paralelního převodu synchronizovaného od sestupné hrany PWM signálu výstupního kanálu $TOM1\_7$ a převodu na pozadí tzv. background mód. U paralelního převodu je převáděno 7 vstupních kanálů v jeden časový okamžik. Procesor umožňuje převést až 8 kanálů, avšak s využitím portů vývojového kitu to není možné. Po dokončení tohoto převodu je okamžitě převáděno dalších 7 kanálů. Převod na pozadí je vykonáván v okamžicích mezi paralelním převodem. Nakonec byl vytvořen ovladač pro zpracování dat z enkodéru, pro jehož připojení je periferie GPT12 uzpůsobena. Všechny takto vytvořené ovladače jsou prakticky odzkoušené na vývojovém kitu s mikrokontrolérem TC275 CA. Dále již proběhla úspěšná zkouška ovladače A/D převodníku na devítifázovém motoru, který je určen do elektromobilu.

V druhé část práce byla provedena analýzu dat z GMR snímače natočení a kompenzaci jeho parametrů s cílem dosažení co nejpřesnějších výsledků. Před provedením analýzy bylo nutné vytvoření přípravku, na kterém je umístěn GMR snímač, DC motor a enkodér. Snímání dat je provedeno pomocí systému CompactRio a programu Lab-View, ve kterém si data ze snímače natočení a enkodéru upravíme pro další použití.

Kód pro kompenzaci parametrů GMR snímače byl vytvořen v programu Matlab, přesněji jeho toolboxu Simulink. Samotná kompenzace je rozdělena na dvě části. V té první si zjistíme statické kompenzační parametry. Druhá část je samotná kompenzace, provedená nejprve s pomocí statických parametrů a následně byla prove-

dena kompenzace dynamická obsahující PI regulátor ve struktuře fázového závěsu (angle tracking observer). Z takto upravených dat byl vypočten úhel natočení a ten následně porovnán (vytvořena diference) s úhlem natočení získaným z dat enkodéru. Tyto diference byla vytvořena pro nekompenzovaný, kompenzovaný pomocí statických parametrů a dynamicky kompenzovaný úhel. Porovnání můžeme vidět na obrázku 8.6, ze kterého je patrné výrazné zmenšení diference úhlů. Hodnota diference vypočtené z dynamicky kompenzovaného úhlu GMR snímače má hodnotu pohybující se okolo $0,4\,^{\circ}$. Diference z nekompenzovaného úhlu měla hodnotu přibližně $1,4\,^{\circ}$, což znamená zlepšení o 70%.

LITERATURA

[1] Sul, S.K.: Control of Electric Machine Drive Systems. *February 2011, Wiley-IEEE Press. ISBN:978-0-470-59079-9.*

[2] BARR, Michael. *Introduction to Pulse Width Modulation. 2001.* [online]. [cit. 2015-01-07]. ISSN 1803-6392. Dostupné z URL: <http://www.embedded.com/electronics-bloetgs/beginner-s-corner/4023833/Introduction-to-Pulse-Width-Modulation>.

[3] ĎAĎO, Stanislav a Marcel KREIDL. Senzory a měřicí obvody. Vyd. 2. *Praha: Vydavatelství ČVUT, 1999, 315 s. ISBN 80-01-02057-6.*

[4] INFINEON TECHNOLOGIES. *AURIX TC27x C-Step: 32-Bit Single-Chip Microcontroller. Mnichov, Německo, 2014.* [cit. 2015-01-05].

[5] INFINEON TECHNOLOGIES. *Application Kit TC2X5: Hardware: APPLICATION KIT TC2X5. Mnichov, Německo, 2013.* [cit. 2015-01-07].

[6] *TriCore Development Platform* [online]. [cit. 2015-05-08]. Dostupné z URL: <https://www.hightec-rt.com/en/downloads/product-information/18-tricore-development-platform-1/file.html>.

[7] *FLASH programming with UDE MemTool FLASH programmer* [online]. [cit. 2015-05-08]. Dostupné z URL: <http://www.pls-mc.com/flash-programming-with-ude-memtool-flash\-programmer/ude_memtool_flash_programming-a-815.html>.

[8] *Universal Debug Engine UDE and Debugger for AURIX, TriCore* [online]. [cit. 2015-05-08]. Dostupné z URL: <http://www.pls-mc.com/universal-debug-engine-ude-and-debugger-\for-aurix-tricore-power-architecture-cortex-arm-xe166xc2000-\xscale-sh-2a_c166st10/universal_debug_engine-a-802.html>.

[9] INFINEON TECHNOLOGIES. *Angle Sensor TLE5009. 1.1. 2014* [online]. [cit. 2015-01-05]. Dostupné z URL: <http://www.infineon.com/dgdl/Infineon-TLE5009_FDS-DS-v01_01-en.pdf?fileId=db3a304330f686060131421d8ddd56b0>.

[10] INFINEON TECHNOLOGIES. *Highly Integrated and Performance Optimized: 32-bit Microcontrollers for Automotive and Industrial Applications. 2014* [online]. [cit. 2015-01-07]. Dostupné z URL: <http://www.infineon.com/dgdl/TriCore_Family_BR-2014.pdf?fileId=db3a30431f848401011fc664882a7648>.

[11] *GMR senzory mag. pole - 1. díl - princip a struktura* [online]. 12.7.2011 [cit. 2015-01-05]. ISSN 1803-6392. Dostupné z URL: <http://automatizace.hw.cz/gmr-senzory-mag-pole-1-dil-princip-a-struktura>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

GMR	Velký magnetorezistenční jev – Giant MagnetoResistive Effect
DSP	Digitální signálový procesor – Digital Signal Processor
PWM	Pulzně šířková modulace – Pulse-width Modulation
GTM	Modul obecného časovače – Generic Timer Module
ARU	Směrovací jednotka – Advanced Routing Unit
TOM	Výstupní modul časovače – Timer Output Module
ATOM	Výstupní modul časovače připojený k jednotce ARU – ARU-connected Timer Output Module
TIM	Vstupní modul časovače – Timer Input Module
TBU	Jednotka časové základny – Timer Base Unit
TGC	TOM Globální jednotka kanálů – TOM Global Channel Unit
CCU	Porovnávací jednotka čítače – Counter Compare Unit
SCU	Systémová řídící jednotka – System Control Unit
VADC	Univerzální anologově-digitální převodník – Versatile Analog to Digital Converter
DSADC	Delta-sigma anologově-digitální převodník – Delta-Sigma to Digital Converter
GPT12	Jednotka obecného časovače – General Purpose Time Unit
QSPI	Sériové periferní rozhraní – Queued Synchronous Peripheral Interface
IDE	Integrované vývojové prostředí – Integrated Development Environment
UDE	Univerzální ladící nástroj – Universal Debug Engine

SEZNAM PŘÍLOH

A ÚPRAVA DAT Z GMR SNÍMAČE

```
% %  Nacteni MAT souboru %%
load('CalibrationPositionSensorParams.mat');
% Nacteni kompenzacnich parametru
load('FOC_bus.mat');
% Nacteni FOC bus
%%       Nacteni a uprava statickych dat ze snimace TLE5009      %%
Ds=fileread('data_GMR_raw_470_init.txt');
% Nacteni textoveho dokumentu s daty do promenne Ds
Ds=strrep(Ds,',','.');
% Prepsani carek teckami
fid=fopen('datas.txt','w');
% Vytvoreni noveho textoveho souboru
fwrite(fid,Ds,'char');
% Zapsani dat z promenne Ds do textoveho dokumentu
fclose(fid);
% Uzavreni textoveho dokumentu
S=importdata('datas.txt','\t',1);
% Import tabulatorem oddelenych dat
SS=(size(S.data)/1000);
 % Zjisteni rozmeru promenne S
sc=(max(S.data(:,2))+min(S.data(:,2)))/2;
% Nalezeni minima a maxima druheho sloupce, vytvoreni offsetu
ss=(max(S.data(:,4))+min(S.data(:,4)))/2;
% Nalezeni minima a maxima ctvrteho sloupce, vytvoreni offsetu
S.data(:,2)=((S.data(:,2)-sc)*10)/32768;
% Uprava druheho sloupce promenne S
%(odecteni offsetu a vydeleni hodnotoou 32768)
S.data(:,4)=((S.data(:,4)-ss)*10)/32768;
% Uprava ctvrteho sloupce promenne S
%(odecteni offsetu a vydeleni hodnotoou 32768)
ts=(0:100e-6:((SS(1,1))/10-100e-6))';
% Vytvoreni casoveho vektoru
```