ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA ELEKTROMECHANIKY A VÝKONOVÉ ELEKTRONIKY

DIPLOMOVÁ PRÁCE

Implementace řízení asynchronního motoru za použití hardwarového akcelerátoru

Bc. Antonín Glac **2016/2017**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2016/2017**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Antonín GLAC**
Osobní číslo:	**E15N0053P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Průmyslová elektronika a elektromechanika**
Název tématu:	**Implementace řízení asynchronního motoru za použití hardwarového akcelerátoru**
Zadávající katedra:	**Katedra elektromechaniky a výkonové elektroniky**

Zásady pro vypracování:

Práce se bude zabývat implementací řízení asynchronního motoru za pomoci hardwarového akcelerátoru CLA (Control Law Accelerator) na mikrokontroléru TMS320F28377S.

1. Zpracujte současný stav a možnosti hardwarové akcelerace pro řízení pohonů obecně.
2. Implementujte řízení asynchronního motoru bez použití CLA.
3. Přeneste vybrané části implementovaného regulačního algoritmu z mikrokontroléru do CLA.
4. Zhodnoťte přínos použití hardwarového akcelerátoru pro implementovaný regulační algoritmus.

Abstrakt

Předkládaná diplomová práce se zabývá implementací řízení asynchronního motoru pomocí napěťového střídače, řízeného mikrokontrolérem TMS320F28377S. Procesor je součástí platformy MLC Interface, používané pro řízení různých pohonů v rámci KEV. Práce porovnává dobu zpracování regulačních výpočtu při použití procesorového jádra, případně kombinace procesoru a koprocesoru CLA. Je řešeno vhodné rozdělení regulačních výpočtů mezi 2 nezávislé výpočetní bloky. Pro kontrolu je kromě ručně psaného kódu vytvořen stejný algoritmus pomocí automaticky generovaného kódu z aplikace Matlab/Simulink.

Klíčová slova

CLA, koprocesor, vektorové řízení, FOC, mikrokontrolér, MLC Interface, akcelerátor, paralelizace, asynchronní motor

Abstract

The master theses presents the implementation of driving algorithm for induction motor using a voltage source inverter, driven by microcontroller TMS320F28377S. Microcontroller is a part of MLC Interface platform, which is used for driving applications at the university. The thesis compares the processing time of the computational calculation using a processor core, or a combination of processor and CLA coprocessor. The appropriate division of control computations between 2 independent computational blocks is solved. In addition to handwritten code, the same algorithm is created using Matlab / Simulink's automatically generated code.

Key words

CLA, coprocessor, FOC, microcontroller, MLC Interface, acceleration, parallel computing

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

.....

podpis

V Plzni dne 17.5.2017 Antonín Glac

Poděkování

Tímto bych rád poděkoval vedoucímu diplomové práce Ing. Tomáši Košanovi, Ph.D. za cenné profesionální rady, připomínky a metodické vedení práce.

Obsah

Úvod

Práce se zabývá implementací řízení asynchronního motoru pomocí mikrokontroléru TMS320F28377S, který pro výpočty využívá nejen hlavní procesor C28x, ale také koprocesor CLA (Control Law Accelerator). Hlavním cílem je provést porovnání přínosu tohoto typu mikroprocesoru oproti staršímu, široce rozšířenému typu mikroprocesorů TMS320F28335 v oblasti řízení a regulace elektrických pohonů. Porovnání hlavních parametrů těchto mikroprocesorů je uvedeno v Tab. 1.

Tab. 1. *Porovnání hlavních parametrů mikroprocesorů 28377S a 28335*

	TMS320F28377S	**TMS320F28335**
CPU	C28x + CLA	C28x
Celkový výpočetní výkon (MIPS)	400	150
Frekvence (MHz)	200	150
Flash (KB)	1024	512
RAM (KB)	164	68
Počet kanálů 12-bit A/D převodníku	24	16
Počet kanálů 16-bit A/D převodníku	12	0
Sigma-Delta Filtry	8	0
D/A převodníky	3	0
PWM (počet kanálů)	24	12
High Resolution PWM (počet kanálů)	16	6
VCU II	1	
FPU	Ano	Ano
TMU	1	

Náplní praktické části bylo vyzkoušení práce se samotným mikroprocesorem, konfigurace a spuštění koprocesoru CLA a následný výběr a implementace vhodného algoritmu řízení asynchronního motoru. Porovnání doby výpočtu samotného mikroprocesorového jádra a kombinace hlavního jádra a CLA bylo provedeno na algoritmu vektorového řízení (FOC).

Pro realizaci byl použit hardware z předmětu KEV/MRP – asynchronní motor 84V/250W, napěťový střídač s IGBT modulem a driverem Semikron a MLC Interface s upraveným mikroprocesorovým modulem 28377S. Postupně bylo implementováno skalární řízení s čidlem otáček a vektorové řízení (FOC). Dále byl kód algoritmu vektorového řízení pro ověření vlastností vygenerován z vytvořeného modelu v aplikaci Matlab/Simulink.

Seznam symbolů a zkratek

FPU ..... Jednotka pro výpočty v plovoucí řádové čárce (Floating Point Unit)

TMU ..... Jednotka pro trigonometrické výpočty (Trigonometric Math Unit)

VCU ..... Jednotka pro komplexní výpočty (Viterbi, Complex Math, and CRC Unit)

CLA ..... Koprocesor (Control Law Accelerator)

FPGA ..... Programovatelné hradlové pole (Field Programmable Gate Array)

CPLD ..... Komplexní programovatelný logický obvod

IRC ..... Inkrementální čidlo otáček (Incremental Rotary Coder)

FFT ..... Rychlá Fourierova transformace (Fast Fourier Transform)

NaN ..... „nečíslo“ (Not a Number)

FOC ..... Vektorové řízení (Field oriented control)

PWM ..... Pulzně šířková modulace (Pulse Width Modulation)

QEP ..... Kvadraturní enkodér (Quadrature Encoder Pulse)

DMA ..... Přímý přístup do paměti (Direct Memory Access)

ISR ..... Funkce obsluhy přerušení (Interrupt Service Routine)

MIPS ..... Milion instrukcí za sekundu – jednotka výkonu

DAC ..... D/A převodník

RAM ..... Paměť s náhodným přístupem (Random Access Memory)

SOC ..... Signál pro zahájení převodu A/D převodníku (Start of Conversion)

EOC ..... Signál dokončení převodu A/D převodníku (End of Conversion)

$(f_r)_{max}$ ..... Kritická rotorová frekvence

K_U ..... Napěťová konstanta skalárního řízení

K_{fr} ..... Napěťová konstanta kompenzace úbytku na odporu

f_{sN} ..... Jmenovitá statorová frekvence

$(U_{sN})_{ef}$ ..... Efektivní hodnota jmenovitého napětí motoru

TB_PRD_X_half..... Polovina maximální hodnoty čítače PWM

uint16..... datový typ – 16 bitové neznaménkové celé číslo

I_{sdw}..... Požadovaný proud v ose d

I_{sqw}..... Požadovaný proud v ose q

I_{sd} ..... Změřený proud přepočtený do osy d

I_{sq} ..... Změřený proud přepočtený do osy q

Ψ_{rw} ..... Požadovaný rotorový magnetický tok

U_{sdw} ..... Požadované napětí v ose d

U_{sqw} ..... Požadované napětí v ose q

dU_{sdw} ..... Výstup regulátoru proudu I_{sd}

dU_{sqw} ..... Výstup regulátoru proudu I_{sq}

U_{sd0} ..... Výstup dopředného modelu v ose d

U_{sq0} ..... Výstup dopředného modelu v ose q

α ..... úhel vektoru požadovaného napětí v souřadnicích [d, q]

β ..... úhel vektoru požadovaného napětí v souřadnicích [α, β]

ϑ ..... úhel natočení soustavy [d, q] oproti stojící soustavě

I_a ,I_b ,I_c ..... Fázové proudy motoru

Ψ_r ..... Rotorový magnetický tok

L_h ..... Hlavní indukčnost motoru

L_r ..... Indukčnost ze strany rotoru

R_r ..... Odpor rotoru

R_s ..... Odpor statoru

$\omega_{r\psi}$ ..... Úhlová rychlost otáčení rotorového magnetického toku

$\omega_{s\psi}$ ..... Úhlová rychlost otáčení statorového magnetického toku

ω_{sU} ..... Úhlová rychlost otáčení vektoru napětí

ω_m ..... Úhlová rychlost otáčení rotoru

ω_w ..... Požadovaná úhlová rychlost otáčení rotoru

p_p ..... Počet pólpárů motoru

$L_{\sigma r}$ ..... Rozptylová indukčnost rotoru

$L_{\sigma s}$ ..... Rozptylová indukčnost statoru

P_n ..... Jmenovitý výkon motoru

U_n ..... Jmenovité napětí motoru

I_n ..... Jmenovitý proud motoru

U_c_2 ..... Polovina napětí stejnosměrného meziobvodu

ε ..... Regulační odchylka

u ..... Výstupní veličina regulátoru

S ..... Suma integrační složky regulátoru

U_{max} ..... Maximální hodnota výstupní veličiny regulátoru

U_{min} ..... Minimální hodnota výstupní veličiny regulátoru

K_p ..... Proporcionální zesílení regulátoru

T_r ..... Integrační časová konstanta regulátoru

Δt ..... Perioda volání výpočtu regulátoru

ω ..... Úhlová rychlost otáčení rotoru

$\Delta X, dCNT$ ..... Přírůstek načtených pulzů čítače QEP

T ..... Perioda výpočtu otáček

X, CNT ..... Nastavený počet pulzů čítače QEP, za který se měří doba načtení

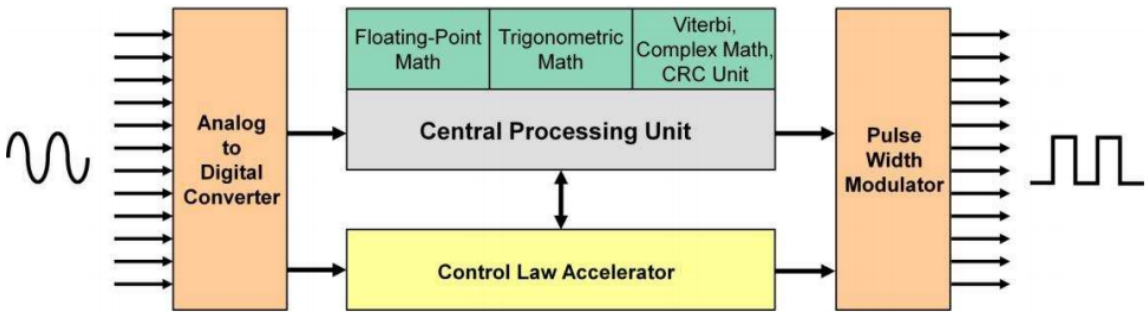
ΔT ..... Doba načtení daného počtu pulzů čítače QEP

1 Teoretická část - Hardwarová akcelerace

Hardwarová akcelerace je využití výpočetního hardwaru k efektivnějšímu provádění výpočtů, než by bylo možné se softwarovou implementací algoritmů na obecném procesoru [1]. Použití HW akcelerace je výhodné pro výpočty, které se neustále opakují, což platí pro většinu regulačních výpočtů. Dle typu akcelerátoru je někdy také možné souběžné zpracování více výpočtů najednou.

Výpočetní jednotka pro HW akceleraci může být přímo součástí procesorového jádra (např. FPU jednotka), integrována do čipu jako koprocesor (CLA), nebo připojena externě, např. ve formě hradlového pole (FPGA) nebo zákaznického obvodu ASIC. Jako příklad může sloužit blokové schéma HW akcelerátorů v rámci mikroprocesoru řady C2000 (Obr. 1).

Procesor bez hardwarové akcelerace má standardně implementovány jen výpočty v pevné řádové čárce. Výpočty ve formátu plovoucí řádové čárky jsou emulovány pomocí softwarové knihovny. Jejich zpracování trvá výrazně déle než při použití pevné řádové čárky a také vzrůstá obsazení programové paměti. Přesto se mikrokontroléry bez akcelerace výpočtů ve formátu plovoucí řádové čárky používají – např. v automotive průmyslu nebo spotřební elektronice, kde jsou řídicí jednotky s těmito mikrokontroléry vyráběny v milionových sériích a náklady na složitější implementaci jsou vykompenzovány nižší cenou. Podobně u jednoduchých aplikací (např. u domácích spotřebičů) není potřeba využívat formát plovoucí řádové čárky (a tím i FPU) z důvodu nízké náročnosti řídicích algoritmů.



Obr.1. *Blokové schéma HW akcelerátorů mikroprocesoru řady C2000 [2]*

1.1 Floating Point Unit (FPU)

Výpočty ve formátu plovoucí řádové čárky popisuje norma IEEE 754. Původní verze normy pochází z roku 1985, aktuální verze z roku 2008.

Norma IEEE 754:2008 [3] specifikuje:

- binární a dekadické formáty čísel a dalších symbolů (Inf, NaN)
- matematické operace $(+, -, \times, \div, \sqrt{\ })$
- pravidla pro konverzi mezi celočíselnými a desetinnými datovými typy
- pravidla pro konverzi mezi různými datovými typy plovoucí řádové čárky
- pravidla pro zaokrouhlování,
- zpracování výjimek (např. dělení nulou nebo přetečení).

Číslo ve formátu plovoucí řádové čárky se skládá ze třech informací – znaménka, mantisy a exponentu. Počet bitů jednotlivých částí závisí na použitém datovém typu.

Norma definuje 5 základních datových typů (Tab. 2). Číslo má tvar $(-\mathbf{1})^{\boldsymbol{s}} \cdot \boldsymbol{b}^{\boldsymbol{e}} \cdot \boldsymbol{m}$

- s – znaménko (0 nebo 1)
- b – základ soustavy (2 nebo 10)
- e – exponent ($e_{min} \leq e \leq e_{max}$)
- m – mantisa ($1 \leq m < b$)

Tab. 2. *Základní datové typy plovoucí řádové čárky*

	Binární formát (b=2)			**Dekadický formát** (b=10)	
parametr	**binary32**	**binary64**	**binary128**	**decimal64**	**decimal128**
p	24	53	113	16	34
e_{max}	127	1023	16383	384	6144

Výhodou formátů plovoucí desetinné čárky je jejich dynamický rozsah. 32 bitový formát plovoucí řádové čárky umožní počítat s čísly v rozsahu $\pm 2^{127}$ ($\pm 10^{38}$), celočíselný formát se stejným počtem bitů má rozsah pouze $\pm 2^{31}$ ($\pm 10^{10}$). Odpadá tak nutnost normalizovat počítané veličiny.

Nevýhodou může být neekvidistantní rozdělení jednotlivých čísel. Při neustálém inkrementování nebo dekrementování proměnné ve formátu plovoucí řádové čárky dojde ke stavu, kdy je sečtená hodnota menší než další možné zobrazitelné číslo a dojde k saturaci.

V mikrokontrolérech bývá obvykle implementována HW akcelerace pouze pro formát single precision (binary32, float).

1.2 Trigonometric Math Unit (TMU)

Použitý mikroprocesor obsahuje jednotku pro akceleraci trigonometrických výpočtů. Tato jednotka urychluje specifické operace, jako je například výpočet hodnoty funkce sinus, cosinus, arkus tangens nebo odmocniny. Ve výpočtech regulace pohonu jsou tyto funkce intenzivně využívány při transformacích.

TMU jednotka je rozšířením FPU jednotky mikroprocesoru. Přidává do instrukční sady efektivně vykonávané trigonometrické a aritmetické operace, které se využívají v systémech řízení v reálném čase.

Kompilátor pro řadu mikrokontrolérů 2837x má zabudovanou podporu pro automatické generování TMU instrukcí. Uživatel napíše svůj kód s využitím standardní knihovny *math.h* a pokud je to možné, kompilátor využije TMU instrukce.

TMU jednotka výrazně urychluje výpočet trigonometrických operací (např. Parkova transformace – s TMU – 13 cyklů procesoru, bez TMU – 80-100 cyklů procesoru) Pokud je zrychlení výpočtu vztaženo k celému výpočtu regulace, TMU jednotka až 1,4x urychluje výpočty oproti samotné FPU jednotce procesoru [2].

1.3 Viterbi, Complex Math, and CRC Unit (VCU)

Dnešní složité systémy řízení vyžadují rychlou a bezchybnou komunikaci mezi jednotlivými částmi systému. V silně zarušeném prostředí (např. komunikace po napájecí síti) může být zajištění spolehlivosti problém – je potřeba využít algoritmy pro dekódování užitečné informace i při vysoké úrovni šumu. Tyto výpočty mohou vyžadovat další

procesor. Alternativou je využití VCU jednotky. Jde o akcelerátor pracující s pevnou řádovou čárkou, který několikanásobně urychluje výpočty konkrétních algoritmů.

Hlavní využití najdou tyto algoritmy

- Viterbi
- Komplexní FFT
- Komplexní filtry
- CRC

Pro použití VCU jednotky je vytvořena knihovna funkcí, které je možné použít v kódu v jazyce C.

1.4 Control Law Accelerator (CLA)

CLA je nezávislý, plně programovatelný 32 bitový hardwarový akcelerátor. Instrukční sada obsahuje matematické a logické operace s formátem single precision plovoucí řádové čárky. S celočíselnými datovými typy je možné provádět pouze sčítání, odečítání a porovnávání, případně konvertovat datové typy na formát single precision a zpět.

Tento akcelerátor je navržen pro vykonávání řídicích algoritmů v reálném čase. Pracuje paralelně s hlavním jádrem mikrokontroléru C28x, což zvyšuje teoretický výpočetní výkon až na dvojnásobek. CLA je vhodný pro obsluhu nízkoúrovňových řídicích smyček, jako je například obsluha A/D převodníku nebo QEP jednotky pro měření a výpočet otáček.

Program pro CLA sestává z inicializačního kódu a z určených rutin (tasků). Struktura kódu tasku je velmi podobná funkcím obsluhy přerušení (ISR). Každý task může být spuštěn buď softwarově (instrukce IACK# v procesoru), nebo událostí/přerušením od periferie, např. A/D převodník, PWM, QEP, SPI, Timer, atd..

Tasků může být nastaveno najednou celkem 8, zpracování probíhá sekvenčně. Priorita zpracování je dána číslem tasku (nižší číslo – vyšší priorita), zanořování tasků není možné. Adresa každého tasku je uložena v příslušném registru MVECTx. Při volání tasků není

nutné ukládat kontext a návratovou adresu, jako je tomu u obsluhy přerušení. Tato skutečnost eliminuje latenci a doba zpracování se stává deterministickou.

Výhodou CLA oproti jiným, specializovaným HW akcelerátorům, je jeho flexibilita. Pomocí programového kódu, který je podobný jako kód pro CPU, lze dosáhnout požadované funkce bez zásahu do hardwaru. Délka tasku je omezena pouze dostupnou pamětí. Navíc na rozdíl od externích HW akcelerátorů (např. FPGA – viz níže) je přímo součástí integrovaného obvodu a odpadá tak komplikované propojení adresovou a datovou sběrnicí.

Pro komunikaci a předávání dat s procesorem jsou určeny 2 bloky Message RAM a bloky LSxRAM. Bloky Message RAM jsou určeny k jednosměrné komunikaci. Do prvního bloku CLA1_to_CPU_RAM může zapisovat pouze CLA. Do druhého bloku CPU_to_CLA1_RAM může naopak zapisovat pouze CPU. Čtení dat z obou bloků není omezeno. Přístup k blokům LSxRAM je možné konfigurovat. Zápis i čtení je možný jak ze strany CPU, tak i CLA. Arbitrace je řešena pomocí algoritmu Round Robin.

1.5 Field-Programmable Gate Array (FPGA)

Dnes používaným řešením pro akceleraci složitých výpočtů je použití procesoru v kombinaci s FPGA. To může být použito jako koprocesor, případně lze použít složitější FPGA, které přímo obsahují procesorová jádra (tzv. SoC (System on Chip), které jsou vhodné pro operační systém spíše než pro systém řízení v reálném čase).

Uvnitř FPGA jsou signály zpracovávány sério-paralelně dle navrženého schématu. Programování hradlových polí se provádí v jazycích pro popis hardwaru VHDL nebo Verilog, případně je možné navrhnout design hardwarového akcelerátoru v programu Matlab/Simulink.

Princip a práce s FPGA je detailně popsána v [4], příklad aplikace FPGA jako akcelerátoru je uveden v [5].

2 Teoretická část - Řízení asynchronního motoru

2.1 Obecný úvod

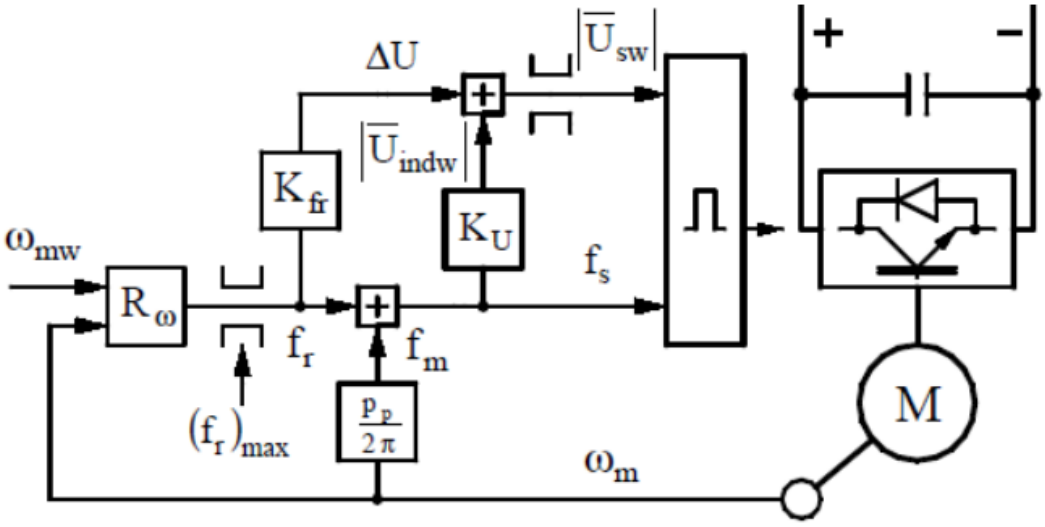
Cílem řízení je získat na výstupu motoru požadované veličiny (moment, otáčky, výkon) i při vnějším zatížení, které se v čase mění. Požadavky na řízení se odvíjí od aplikace, kde je motor použit. V některých aplikacích je požadavek na velkou dynamiku pohonu (servopohony), v jiných aplikacích (např. obráběcí stroje) je naopak požadavek na přesné dosažení požadovaných parametrů bez překmitů.

V dalších kapitolách jsou popsány způsoby řízení, které byly implementovány a prakticky otestovány na zvoleném asynchronním motoru.

2.2 Napěťově kmitočtové řízení (skalární řízení)

Skalární řízení je možné realizovat s čidlem i bez čidla otáček. Implementováno bylo řízení s čidlem otáček (Obr. 2), které má lepší dynamické vlastnosti. Implementace sloužila především pro odladění všech používaných periferií, nebyla určena k porovnávání výpočetní náročnosti. Výsledky nejsou dále prezentovány, hlavním cílem byla implementace vektorového řízení.

Skalární řízení vychází z vlastností asynchronního motoru, kde otáčky motoru jsou přímo úměrné statorové frekvenci napájecího napětí. Tudíž je nutné při snížené frekvenci úměrně snížit i napětí, aby nebyl překročen maximální statorový magnetický tok (tj. udržovat poměr U/f konstantní).



Obr. 2. *Skalární řízení asynchronního motoru s čidlem otáček – převzato z [6]*

Pro statorovou frekvenci nižší než jmenovitou požadujeme v ustáleném stavu konstantní absolutní hodnotu statorového toku. Při dosažení maximálního napětí poměr U/f klesá a dochází k odbuzování motoru.

Blok K_U odpovídá poměru amplitudy jmenovitého statorového napětí a jmenovité statorové frekvence. Omezovač zajistí, že hodnota napětí nepřekročí jmenovitou hodnotu. Blok K_{fr} kompenzuje vliv úbytku napětí na statorovém odporu. Uplatňuje se zejména při nízkých hodnotách statorové frekvence.

$$K_U = \frac{(U_{sN})_{ef} \cdot \sqrt{2}}{f_{sN}} \tag{1}$$

$$K_{fr} = \frac{(U_{sN})_{ef} \cdot \sqrt{2}}{f_{sN}} \cdot \frac{R_s}{R_r} \tag{2}$$

U pohonu s čidlem otáček jsou zadávány požadované otáčky. Výstupem regulátoru otáček je rotorová frekvence, která je až do omezení kritickou rotorovou frekvencí $(f_r)_{max}$ přímo úměrná momentu. Sečtením požadované rotorové frekvence a mechanických otáček, přepočtených přes počet pólpárů, získáme požadovanou frekvenci pro střídač. Požadované napětí střídače je vypočteno z rotorové frekvence tak, aby byl zachován konstantní magnetický tok. V ustáleném stavu a při pomalých přechodových dějích je $\omega_{sU} \cong \omega_{s\psi}$.

Výhodou skalárního řízení oproti vektorovému řízení je nízká náročnost na výpočetní výkon a jednodušší implementace. Ve variantě bez čidla otáček se jedná o nejlevnější možné řešení regulovaného pohonu asynchronního motoru.

Nevýhodou je horší dynamika a citlivost na přesnost měření otáček – malá chyba měřených otáček oproti skutečným způsobí velkou změnu vektoru napětí, a tím i velkou změnu momentu. U velkých motorů s tvrdou charakteristikou nelze tento typ řízení použít.

2.3 Vektorové řízení (FOC)

Vektorové řízení asynchronního motoru vzniklo jako snaha řídit střídavý motor podobně jako stejnosměrný motor – odděleně zadávat požadavek na moment a na magnetický tok motoru. Součástí vektorového řízení je vždy matematický model motoru,

který počítá velikost a úhel natočení magnetického toku a transformuje proudy a napětí do příslušných souřadných systémů.

2.3.1 Transformace Park a Clarke

Pro popis funkce vektorového řízení je nejprve třeba zavést souřadné soustavy pro jednotlivé veličiny.

Okamžité hodnoty třífázové veličiny lze transformovat na jeden „prostorový vektor“ v souřadnicích [α, β] (někdy značeny [x, y]).

$$\bar{x} = k \cdot (x_a + \bar{a} \cdot x_b + \bar{a}^2 \cdot x_c), \bar{a} = e^{j \cdot 120°} \tag{3}$$

$$x_0 = k_0 \cdot (x_a + x_b + x_c\ ) \tag{4}$$

$$Re\{\bar{x}\} = \ k \cdot [x_a - 0{,}5 \cdot (x_b + x_c)] \tag{5}$$

$$Im\{\bar{x}\} = \ k \cdot \frac{\sqrt{3}}{2} \cdot (x_b - x_c) \tag{6}$$

Konstanta *k* je vypočtena tak, aby platila podmínka dle rovnice (7) [6].

$$x_a = Re\{\bar{x}\} + x_0 \ \rightarrow \ k = \frac{2}{3} \tag{7}$$

$$x_\alpha = \frac{2}{3} \cdot x_a - \frac{1}{3} \cdot (x_b + x_c)] \tag{8}$$

$$x_\beta = \frac{1}{\sqrt{3}} \cdot (x_b - x_c) \tag{9}$$

V souřadnicích [α, β] uvažujeme soustavu I, která je svázána se statorem. Převod z třífázové soustavy na soustavu I se nazývá transformace dle Clarkové (Clark’s transform).

Další možností je soustava II, svázaná s polohou rotoru. Ta je vhodná především pro synchronní stroje, kde je rotorový tok přímo určen polohou rotoru.

Pro účely vektorového řízení je nejvíce vhodná soustava III [d, q], která se otáčí synchronně se statorovým (i rotorovým) magnetickým tokem. Úhel ϑ udává aktuální natočení mezi soustavou I a III.

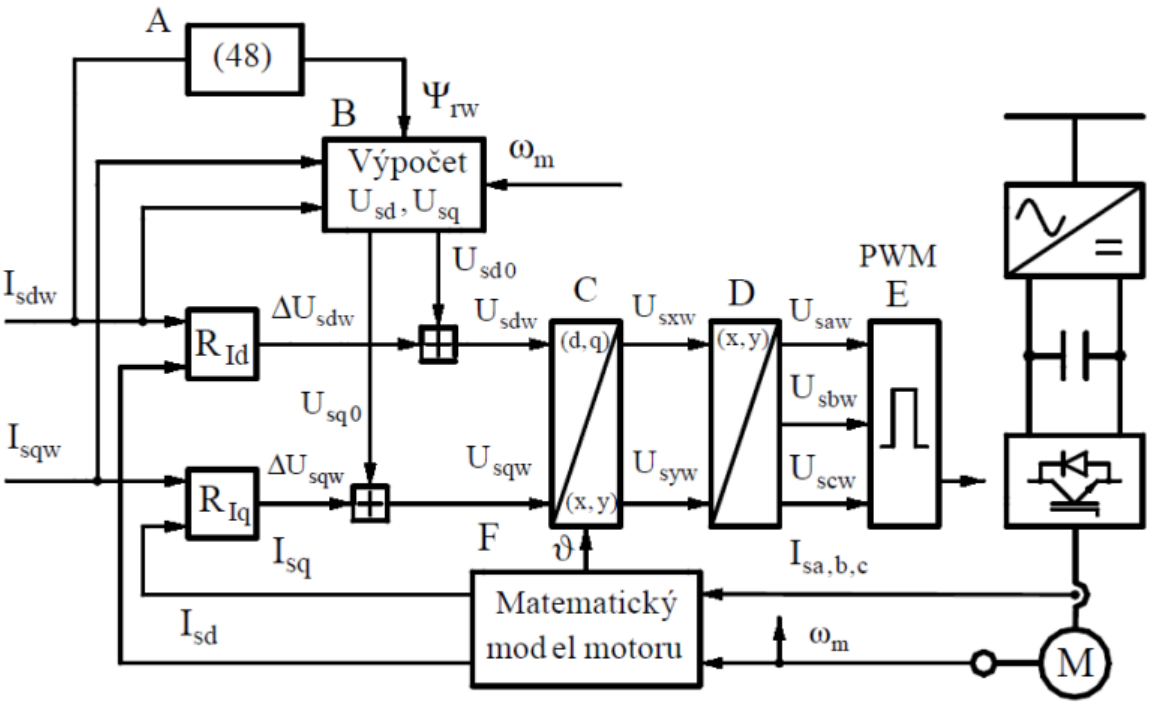
Pro převod veličin do soustavy III se využívá Parkova transformace.

$$x_d = x_\alpha \cdot cos(\vartheta) + x_\beta \cdot sin(\vartheta) \tag{10}$$

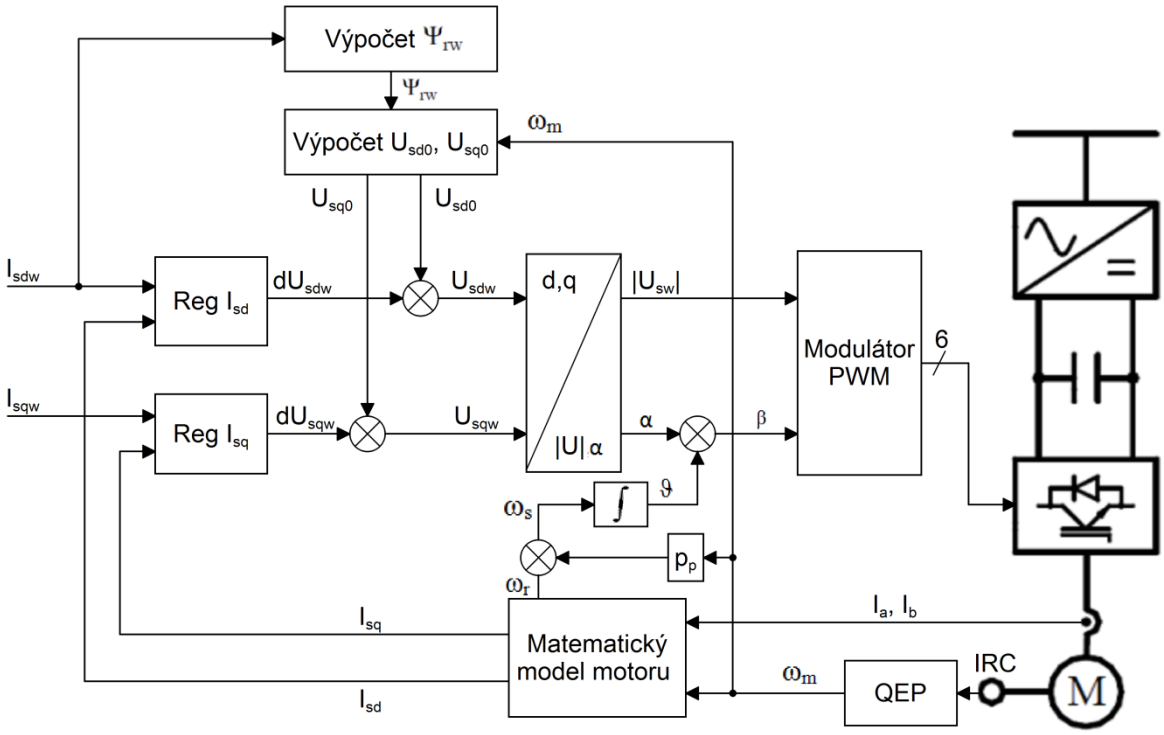
$$x_q = -x_\alpha \cdot sin(\vartheta) + x_\beta \cdot cos(\vartheta) \tag{11}$$

Vektorové řízení je potřeba orientovat na jeden konkrétní magnetický tok. Nejvhodnější je použít orientaci na rotorový tok, kdy v ustáleném stavu lze proud I_{sd} považovat za proud magnetizační a složka rotorového proudu I_{rd} je nulová.

Transformace veličin je možné provádět buď ve složkovém tvaru (Obr. 3), nebo v polárním tvaru (Obr. 4).



Obr. 3. *Vektorové řízení asynchronního motoru – složkový tvar - převzato z [6]*



Obr. 4. *Vektorové řízení asynchronního motoru – polární tvar*

Regulátory I_{sd} a I_{sq} společně s bloky výpočtů vyhodnocují z požadovaných proudů požadovaná napětí pro střídač. Požadavky na napětí ve složkách [d, q] jsou transformovány do polárního tvaru pomocí absolutní hodnoty a funkce atan2().

$$|U_{sw}| = \sqrt{{U_{sdw}}^2 + {U_{sqw}}^2} \tag{12}$$

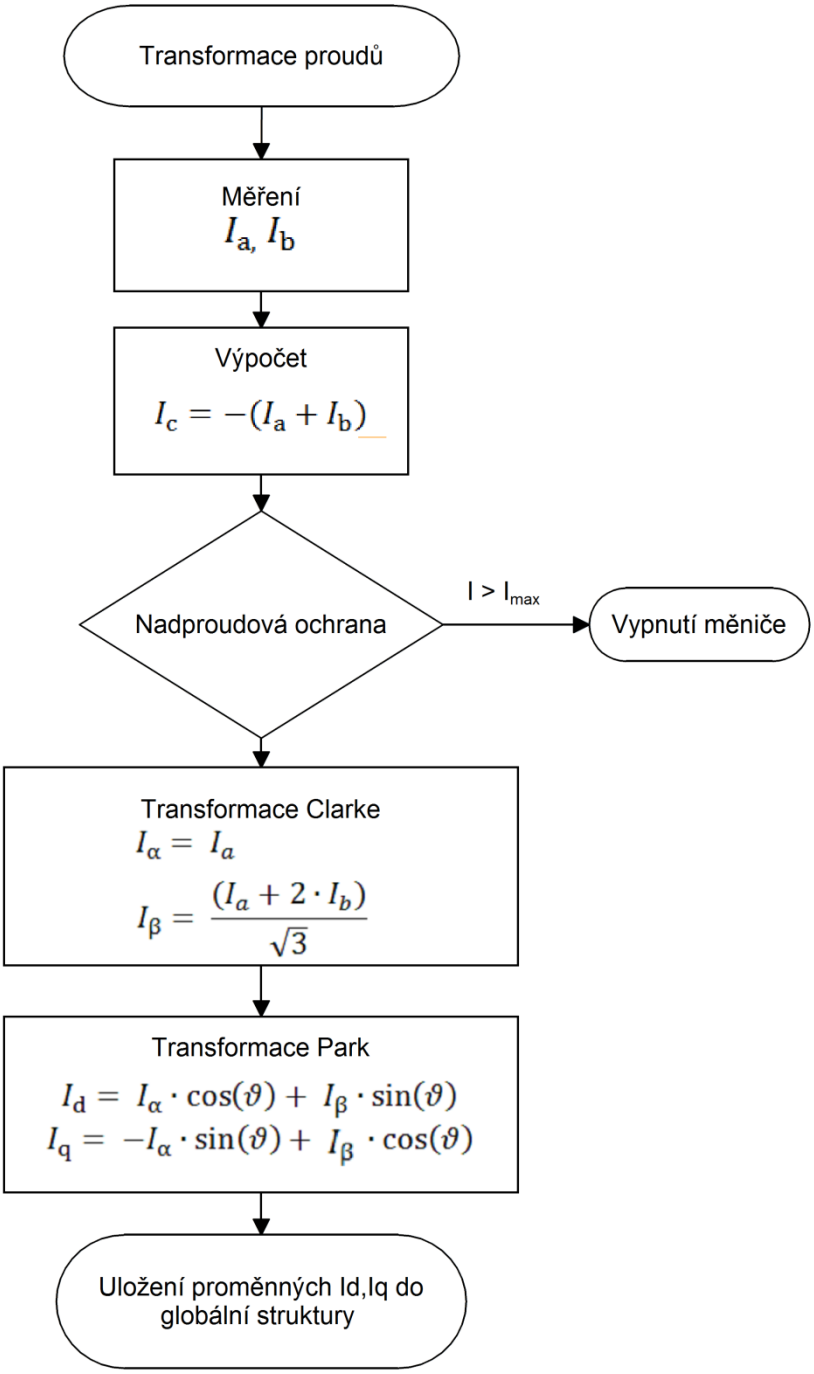
$$\alpha = atan\left(\frac{U_{sqw}}{U_{sdw}}\right) \tag{13}$$

Přičtením úhlu ϑ je provedena transformace do souřadného systému [α, β]. V této soustavě požadavek zpracuje modulátor PWM, který zajistí takové spínání prvků měniče, aby byl na výstupu měniče požadovaný vektor napětí.

2.3.2 Matematický model

Matematický model lze rozdělit do 2 hlavních částí.

První část je měření a transformace proudů (Obr. 5).



Obr. 5. *Implementace měření a transformací proudu*

Druhou částí je výpočet velikosti a úhlu natočení rotorového magnetického toku dle (14) – (18).

$$\frac{d\psi_r}{dt} = \left(\frac{L_h \cdot R_r}{L_r} \cdot I_{sd} - \frac{R_r}{L_r} \cdot \psi_r\right) \tag{14}$$

$$\psi_r = \int \left(\frac{L_h \cdot R_r}{L_r} \cdot I_{sd} - \frac{R_r}{L_r} \cdot \psi_r\right) dt \tag{15}$$

$$\omega_{r\psi} = I_{sq} \cdot \frac{L_h \cdot R_r}{L_r} \cdot \frac{1}{|\psi_r|} \tag{16}$$

$$\omega_{s\psi} = \omega_{r\psi} + p_p \cdot \omega_m \tag{17}$$

$$\vartheta = \vartheta_{(0)} + \int_0^t \omega_{s\psi}\, dt \tag{18}$$

2.3.3 Dopředný model

Vlivem rozptylových indukčností nejsou složky napětí U_{sd} a U_{sq} navzájem nezávislé. Proud I_{sd} způsobuje indukování napětí do osy q a naopak proud I_{sq} způsobuje indukování napětí do osy d. Úkolem dopředného modelu je tento vliv odstraňovat. Někdy se dopředný model označuje jako odvazbovací obvod (decoupling).

Dopředný model není bezpodmínečně nutný pro chod vektorového řízení, ale zlepšuje dynamiku systému – při rychlých změnách proudu není nutné čekat na reakci regulátoru.

$$\psi_{rw} = \int \left(\frac{L_h \cdot R_r}{L_r} \cdot I_{sdw} - \frac{R_r}{L_r} \cdot \psi_{rw}\right) dt \tag{19}$$

$$U_{sd0} = R_s \cdot I_{sd} - p_p \cdot \omega_m \cdot (L_{s\sigma} + L_{r\sigma}) \cdot I_{sq} \tag{20}$$

$$U_{sq0} = R_s \cdot I_{sq} + p_p \cdot \omega_m \cdot (L_{s\sigma} + L_{r\sigma}) \cdot I_{sd} + \omega_{s\psi} \cdot \psi_{rw} \tag{21}$$

Úhlovou rychlost otáčení statorového magnetického toku $\omega_{s\psi}$ (21) je možné nahradit přepočtenými mechanickými otáčkami $p_p \cdot \omega_m$. Vzniklá chyba je malá a je vykompenzována regulátorem.

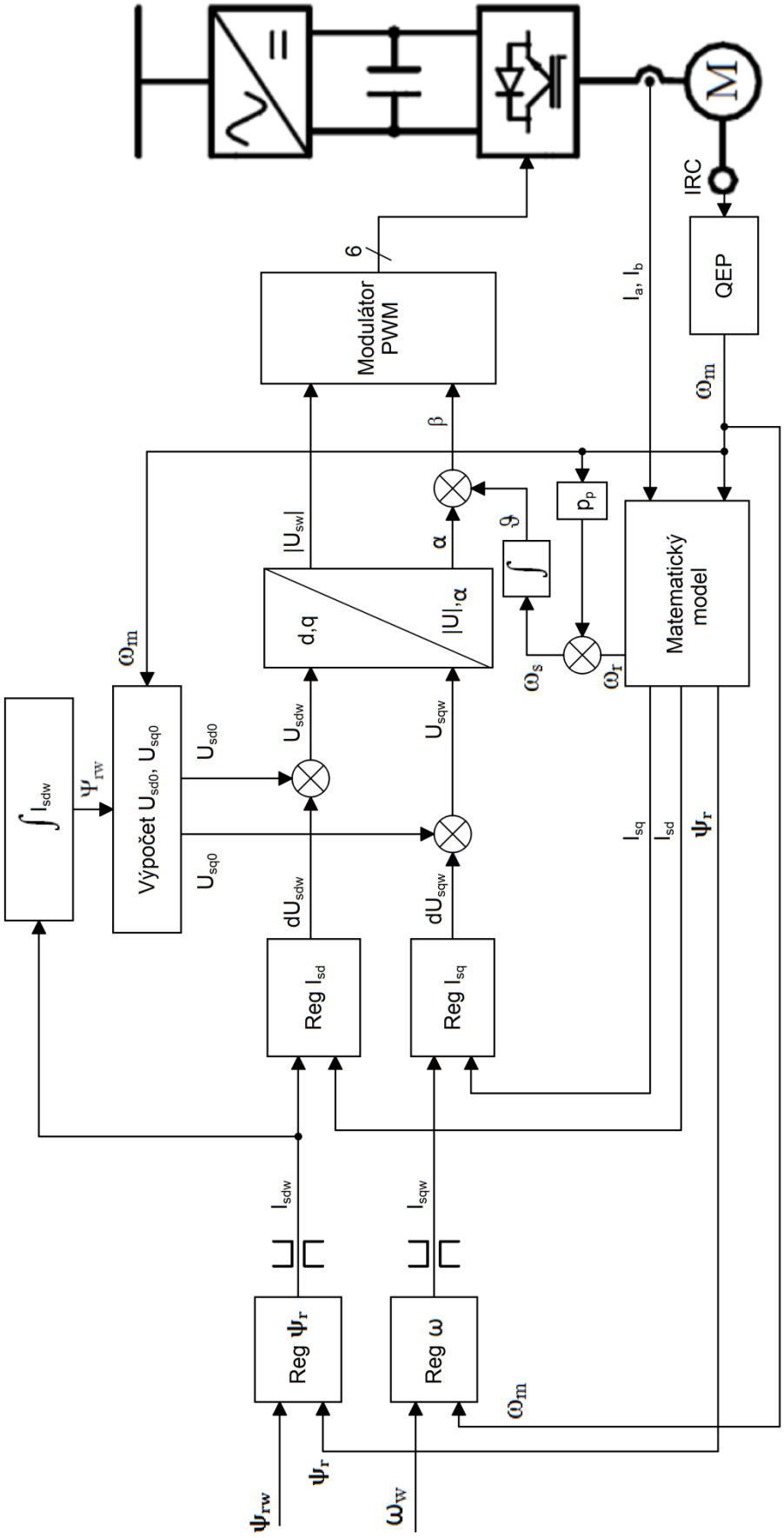
Požadované napětí vstupující do modulátoru je součtem výstupu regulátoru a dopředného modelu.

2.3.4 Nadřazené regulátory

Požadavek na otáčky je pomocí regulátoru otáček přepočten na požadovaný proud I_{sqw}. Požadavek je zadáván přímo uživatelem přes ovládací rozhraní, případně může být zadáván nadřazeným regulátorem polohy. Omezovačem je zajištěno, že nedojde k překročení maximálního celkového dovoleného proudu. Zpětná vazba je vedena od čidla otáček, případně z matematického modelu, pokud je použito bezsenzorové řízení.

Požadavek na magnetický tok je přepočten regulátorem toku na požadovaný proud I_{sdw}. Požadovaný magnetický tok je nastaven jako konstanta dle typu motoru a napájecího napětí zdroje. Zpětná vazba je vypočítávána matematickým modelem.

Kompletní schéma vektorového řízení včetně nadřazených regulátorů je zobrazeno na Obr. 6.



Obr. 6. *Vektorové řízení asynchronního motoru – kompletní schéma*

3 Praktická část - Implementace

3.1 Použitý hardware

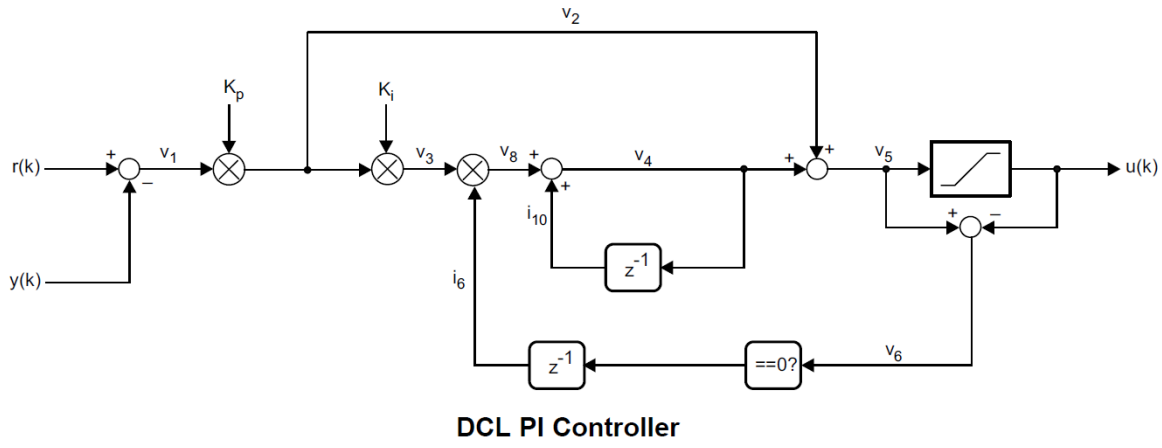
- MLC interface + MCU modul TMS320F28377S
- Nepřímý měnič kmitočtu s napěťovým střídačem (MRP)
- Asynchronní motor Siemens RRA2704-073 (parametry viz Tab. 3)
- Zdroj 60V DC s proudovým omezením

Tab. 3. *Parametry asynchronního motoru*

R_s	1,86 Ω
R_r	1,53 Ω
L_h	0,033 H
$L_{\sigma r}$	0,0043 H
$L_{\sigma s}$	0,0053 H
p_p	2
P_n	250 W
U_n	48 V (D) / 83 V (Y)
I_n	5,16 A

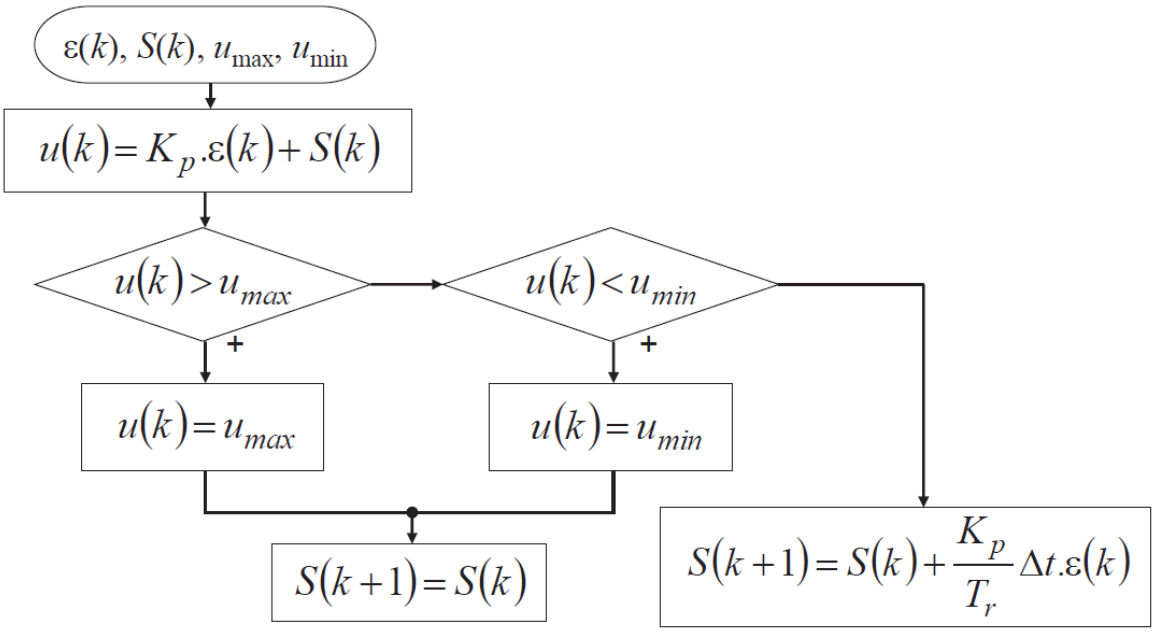
3.2 PI regulátory

Při použití CLA byl implementován PI regulátor z knihovny DCL (Obr. 7) [7] (C2000 Digital Controller Library), která je dostupná v programu ControlSuite. Veškeré proměnné jsou datového typu float. Kód PI regulátoru je vytvořen v assembleru.



Obr. 7. *DCL PI [7]*

Pro implementaci regulátorů v CPU byla vytvořena vlastní funkce pro PI regulátor dle vývojového diagramu (Obr. 8) [8]. Regulační odchylka ε je rozdílem požadované a skutečné hodnoty regulované veličiny. Statická proměnná S vyjadřuje integrační složku PI regulátoru. Nezbytnou součástí regulátoru je saturace výstupní veličiny a následné zastavení integrace odchylky. Pro správnou funkci PI regulátoru je třeba vhodně nastavit proporční zesílení K_p a integrační časovou konstantu T_r.

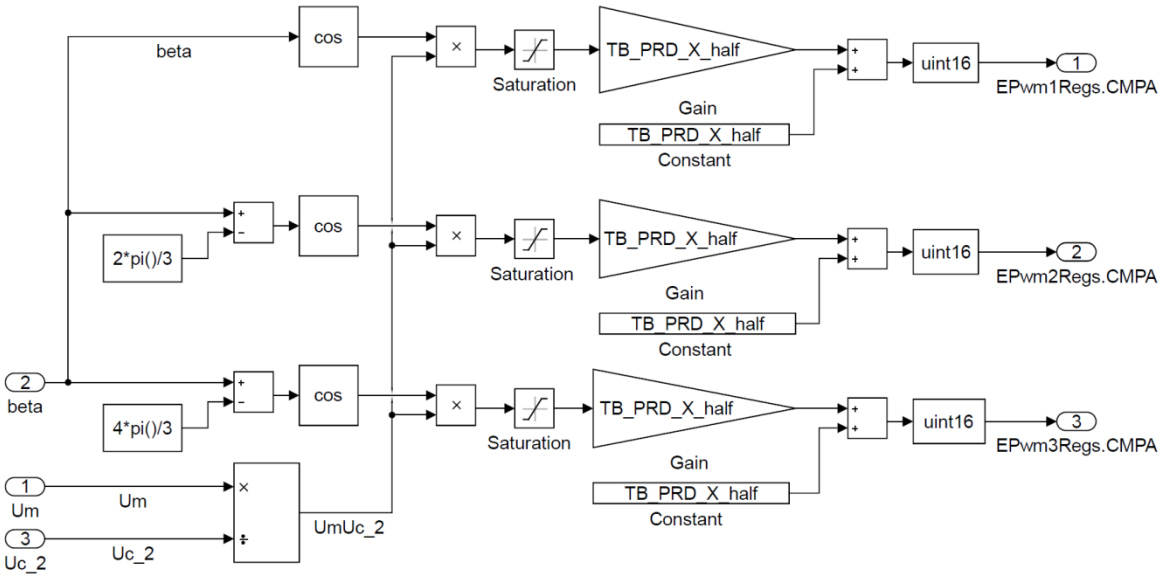


Obr. 8. *Složkový tvar PI (PS) regulátoru [8]*

3.3 Modulátor

Modulátor PWM přepočítává požadovaný vektor napětí v souřadnicích [α, β] na požadované poměrné sepnutí jednotlivých prvků třífázového napěťového střídače. Kód modulátoru byl ručně napsán na základě [8] (kapitola 6). Při následném testování pomocí generovaného kódu bylo schéma modulátoru (Obr. 9) převedeno do aplikace Simulink.

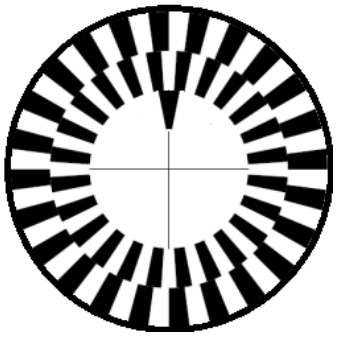
Vstupem pro modulátor PWM je amplituda napětí, úhel natočení vektoru napětí a měřené napětí stejnosměrného meziobvodu. Saturace omezuje poměrné požadované napětí v rozsahu (-1,1). Konstanta *TB_PRD_X_half* je dána spínací frekvencí – odpovídá polovině maximální hodnoty čítače PWM periferie.



Obr. 9. *Modulátor PWM*

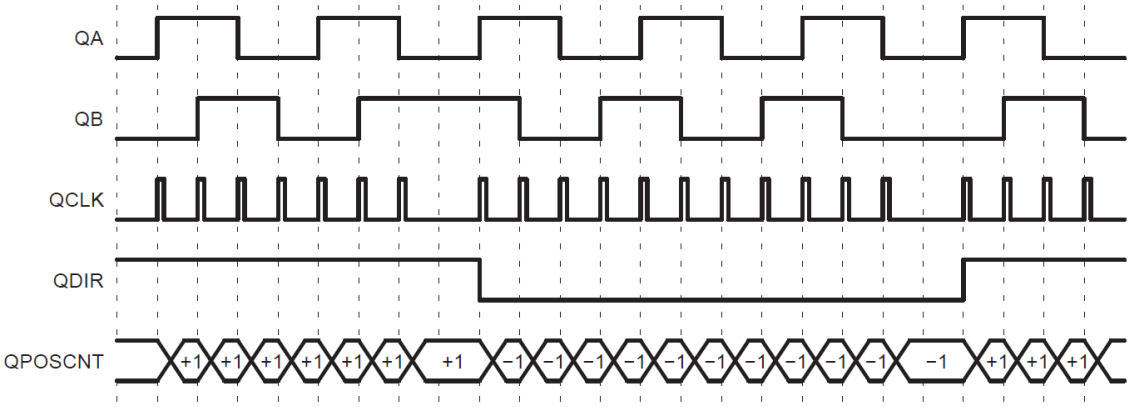
3.4 Měření otáček

IRC čidlo (Obr. 10) sestává z otočného kotouče s 2 řadami okének, navzájem posunutých o 90°. Jednou za otáčku vyšle čidlo index pulz, který signalizuje průchod konkrétním úhlem natočení rotoru.

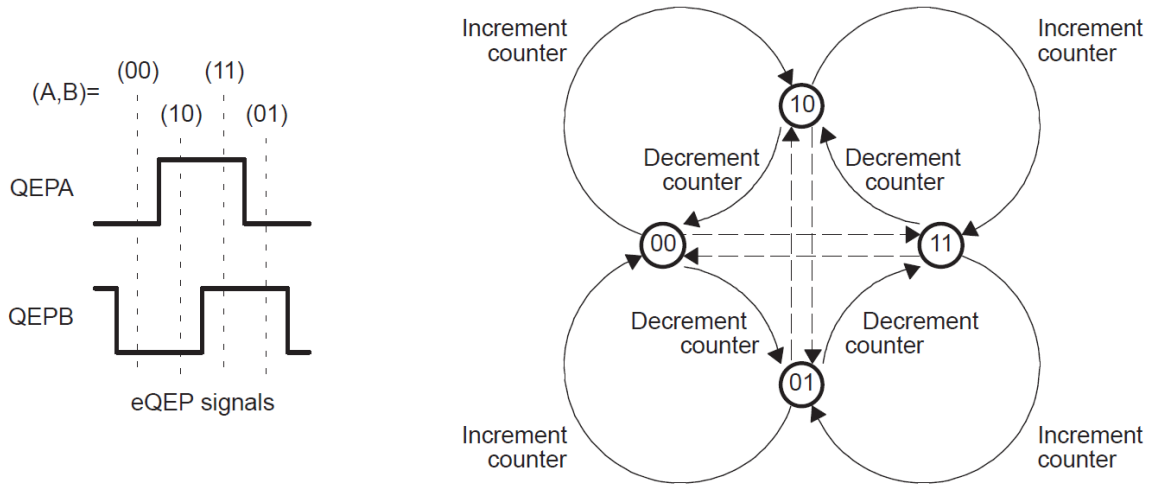


Obr. 10. IRC čidlo

Kvadraturní enkodér QEP ze vstupních signálů (Obr. 11) určuje směr a rychlost otáčení. Určení směru otáčení probíhá dle jednoduchého stavového automatu (Obr. 12).



Obr. 11. *Signály periferie QEP (QA,QB – vstupní signály z IRC, QCLK – výstupní signál QEP, QDIR – směr otáčení, QPOSCNT – hodnota čítače pulzů) [9]*



Obr. 12. *Určení směru otáčení [9]*

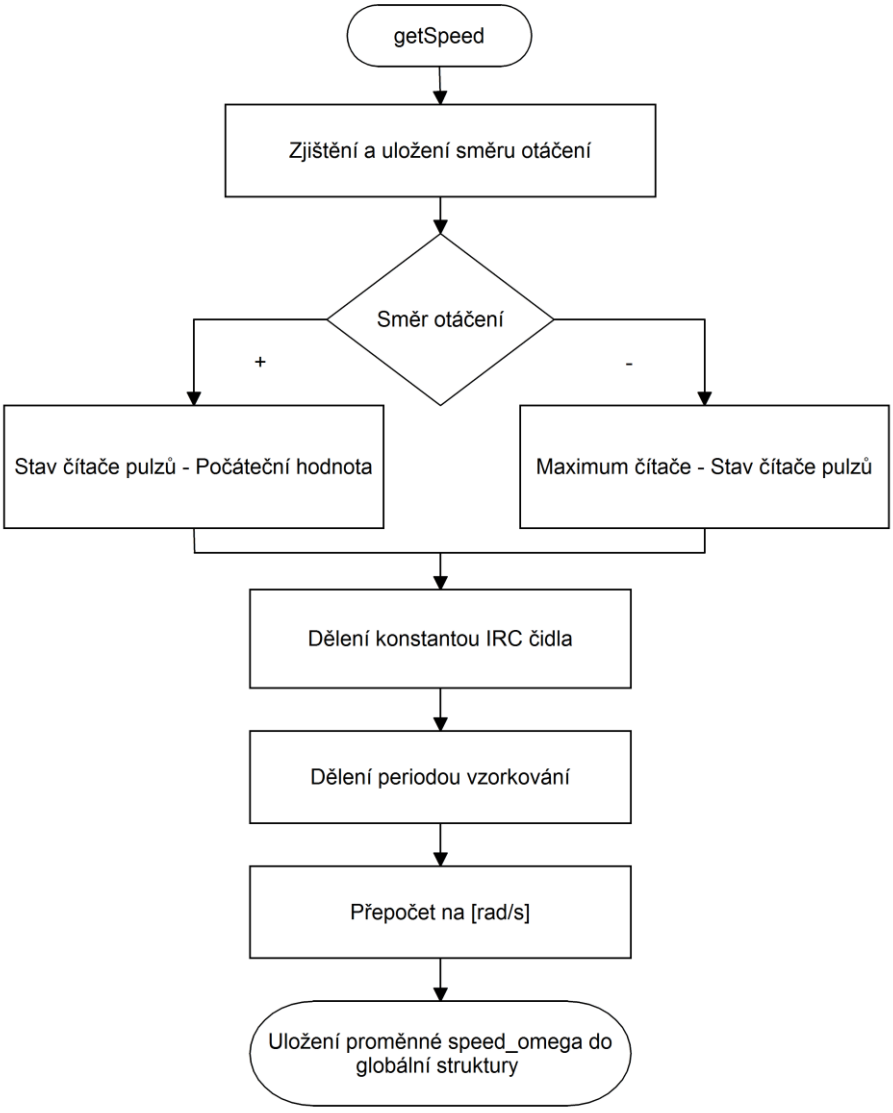
Signál z IRC čidla je nejprve převeden v CPLD z dvoubitového diferenciálního signálu na jednobitový single-ended signál a následně zpracován periferií QEP v mikroprocesoru. Pro měření otáček je použito čidlo IRC s rozlišením 1024 pulzů na otáčku.

Pro výpočet otáček pomocí periferie QEP je možné využít 2 způsoby [9].

$$\omega(k) \approx \frac{x(k) - x(k-1)}{T} = \frac{\Delta X}{T} \tag{22}$$

$$\omega(k) \approx \frac{X}{t(k) - t(k-1)} = \frac{X}{\Delta T} \tag{23}$$

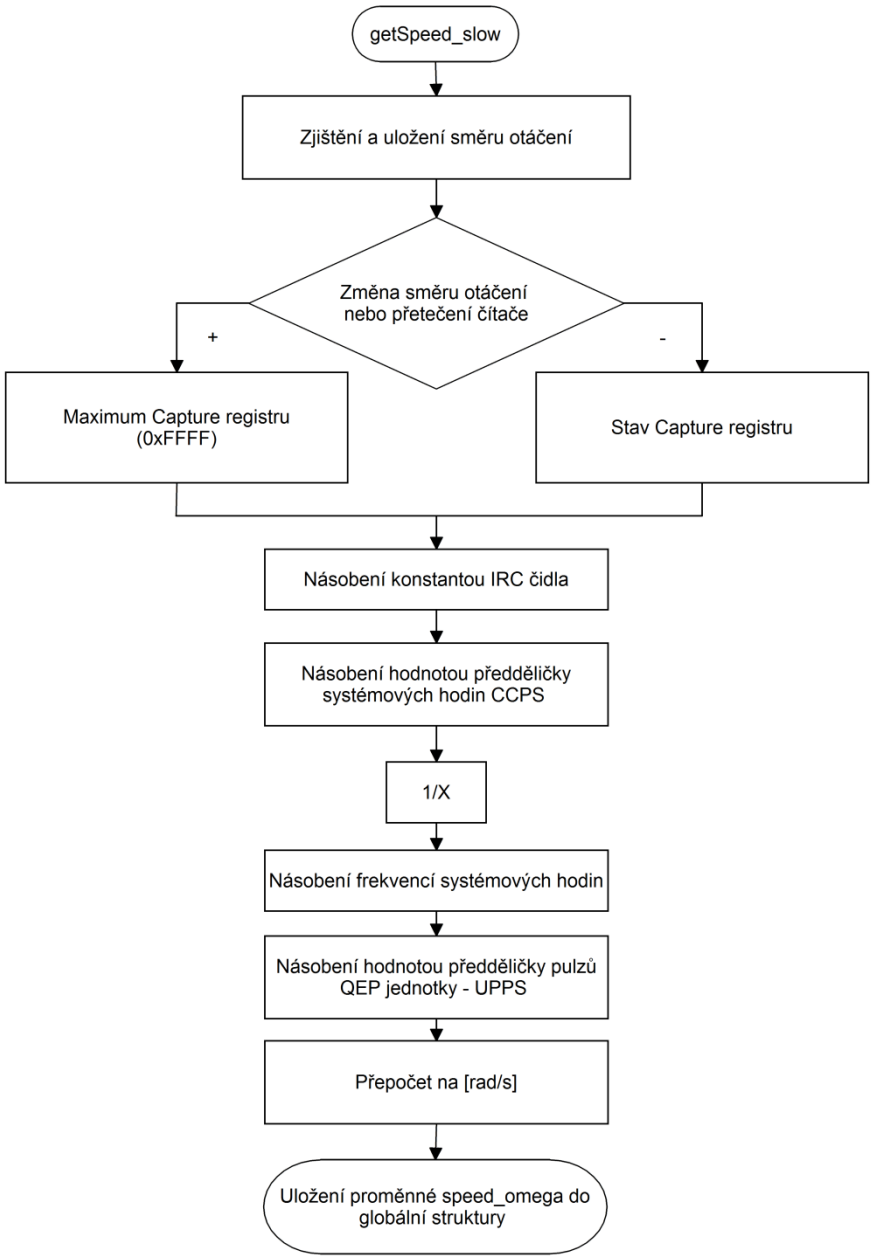
První metoda (22), (Obr. 13) vzorkuje s pevně danou periodou počet načítaných pulzů. Podíl počtu pulzů a periody odpovídá rychlosti otáčení. Čítač je pak vynulován. Minimální rozlišení je takové, kdy je za dobu vzorkování načten pouze 1 pulz. Obecně je tato metoda vhodná pro vyšší otáčky. Absolutní přesnost metody klesá s rostoucí vzorkovací frekvencí.



Obr. 13. *Implementace měření rychlosti 1. metodou*

Druhá metoda (23), (Obr. 14) měří čas v počtu pulzů předdělených systémových hodin, za který se načítá určitý počet hran signálu QCLK. Čtení stavu čítače a výpočet rychlosti probíhá s pevnou periodou, která ale nemá vliv na přesnost. Metoda je vhodná pro nižší otáčky, při vyšších otáčkách roste relativní chyba měření času.

Tento způsob nemůže poskytovat korektní výsledky, pokud během měření došlo ke změně směru otáčení, nebo došlo k přetečení 16bitového čítače. Tyto stavy je třeba ošetřit podmínkou. Periferie eQEP přímo tyto stavy signalizuje pomocí stavového registru.



Obr. 14. *Implementace měření rychlosti 2. Metodou*

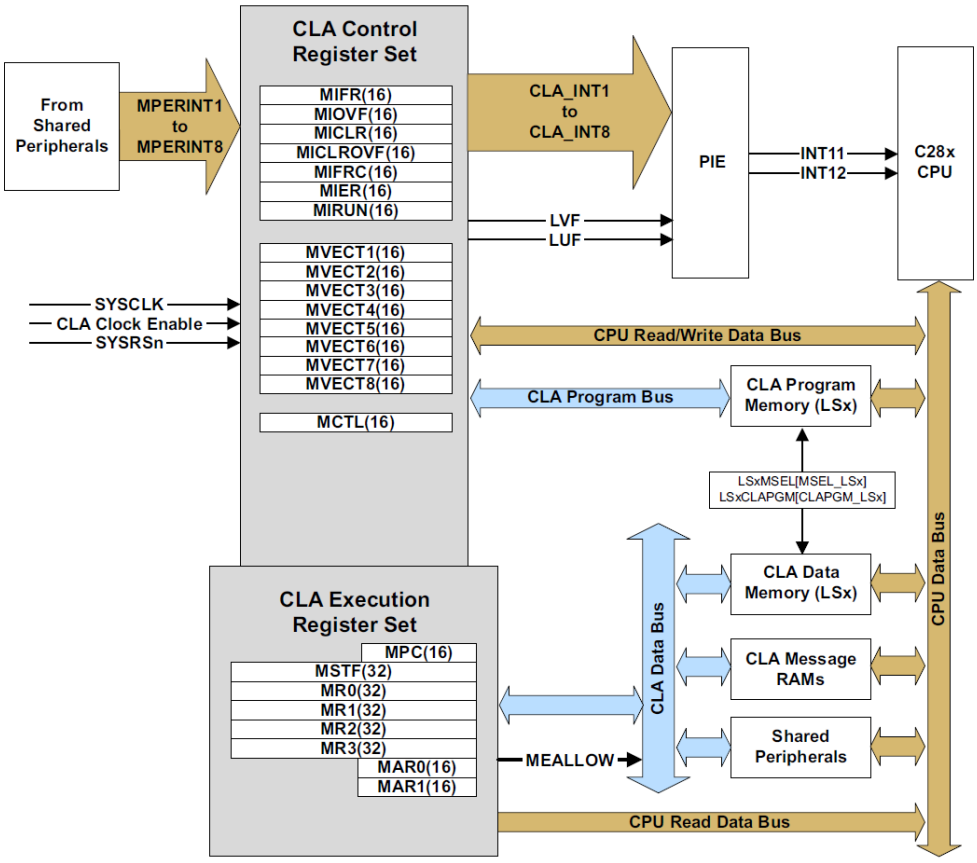
Otáčky motoru byly zpracovávány v přerušení volaném každých 10 ms, což se ukázalo jako nedostatečné z hlediska dynamiky pohonu. Perioda přerušení byla proto zkrácena na 1 ms a výsledné otáčky jsou zpracovány jednoduchým filtrem typu klouzavý průměr.

3.5 Programování CLA

Inicializace CLA začíná povolením systémových hodin pro CLA a nastavením paměti (Obr. 15). Nejprve je potřeba inicializovat bloky Message RAM (každý blok 128 x 16 bitů) a zkontrolovat úspěšné provedení inicializace v příslušném registru. Bloky paměti LS0 – LS5 (každý blok 2K x 16 bitů) je možné přiřadit CLA, nebo ponechat jako paměť CPU. Každý z bloků je možné nastavit jako programovou nebo datovou paměť.

Další krok je nastavení adres funkcí jednotlivých tasků do registrů MVECTx. Adresa má 16 bitů a určuje začátek kódu tasku v dolních 64kB paměti. Ukazatel je třeba přetypovat na 16 bitový datový typ.

Pomocí registru MCTL je možné povolit nebo zakázat softwarové spouštění tasků pomocí instrukce #IACK.



Obr. 15. *Blokové schéma CLA [9]*

Dále jsou v tabulce řadiče přerušení (PieVectTable) nastaveny funkce obsluhy přerušení po dokončení tasku (*Výp.1*).

Kód programu (*Výp.2*), který má být zpracován CLA, je podobný funkci obsluhy přerušení. Implementace se provádí v jazyce C s určitými omezeními. Jednotlivé funkce musí být uloženy v souboru s příponou .cla . Pro matematické funkce je určena speciální knihovna *CLAmath.h*, standardní knihovny *math.h* nebo *stdio.h* nejsou podporovány.

```
542 __interrupt void cla1Isr1 ()
543 {
544     // Acknowledge the end-of-task interrupt for task 1
545     PieCtrlRegs.PIEACK.all = M_INT11;
546 //  asm(" ESTOP0");
547 }
```

Výp.1 *Příklad funkce obsluhy přerušení po dokončení tasku*

```
147 __interrupt void Cla1Task4()
148 {
149     //regulatory Isd, Isq
150     dUsdw_CLA_fpga = DCL_runPIc(&Reg_Isd_DCL_fpga , Isdw_CLA_fpga , Isd_CLA_fpga);
151     dUsqw_CLA_fpga = DCL_runPIc(&Reg_Isq_DCL_fpga , Isqw_CLA_fpga , Isq_CLA_fpga);
152 }
153
```

Výp.2 *Struktura tasku*

Veškeré proměnné, se kterými CLA pracuje, je nutné umístit do příslušných paměťových sekcí (*Výp.3*).

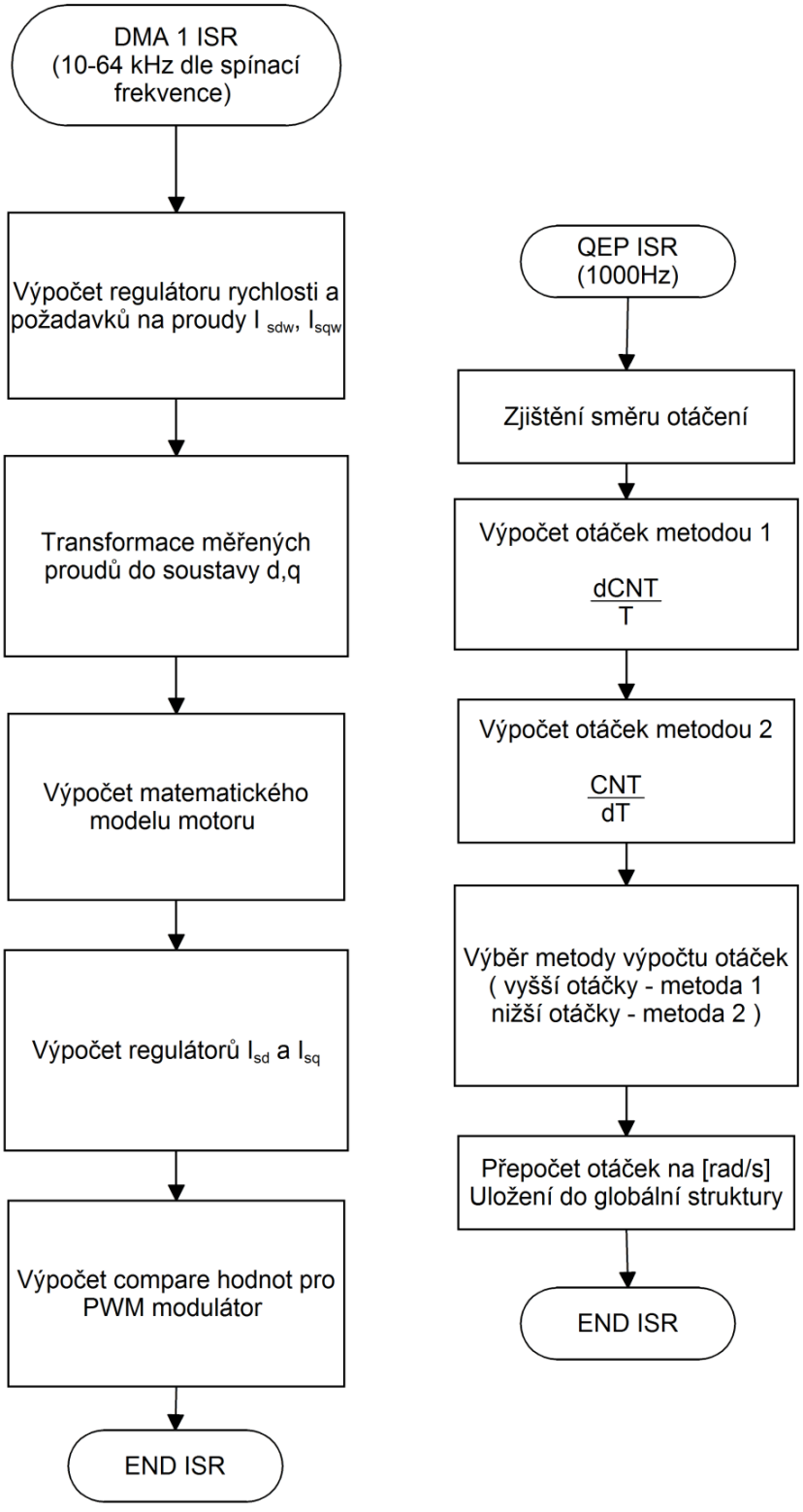
```
59 //Task 5 (C) Variables
60
61 #pragma DATA_SECTION(speed_omega_pre,"CpuToCla1MsgRAM")
62 float speed_omega_pre = 0 ;
63 #pragma DATA_SECTION(speed_omega_filtered,"Cla1ToCpuMsgRAM")
64 float speed_omega_filtered = 0 ;
65
66 #pragma DATA_SECTION(filtr_N,"CLADataLS0")
67 int filtr_N = 0;
68 #pragma DATA_SECTION(filtr_sum,"CLADataLS0")
69 float filtr_sum = 0;
```

Výp.3 *Přiřazení proměnných do paměťových sekcí*

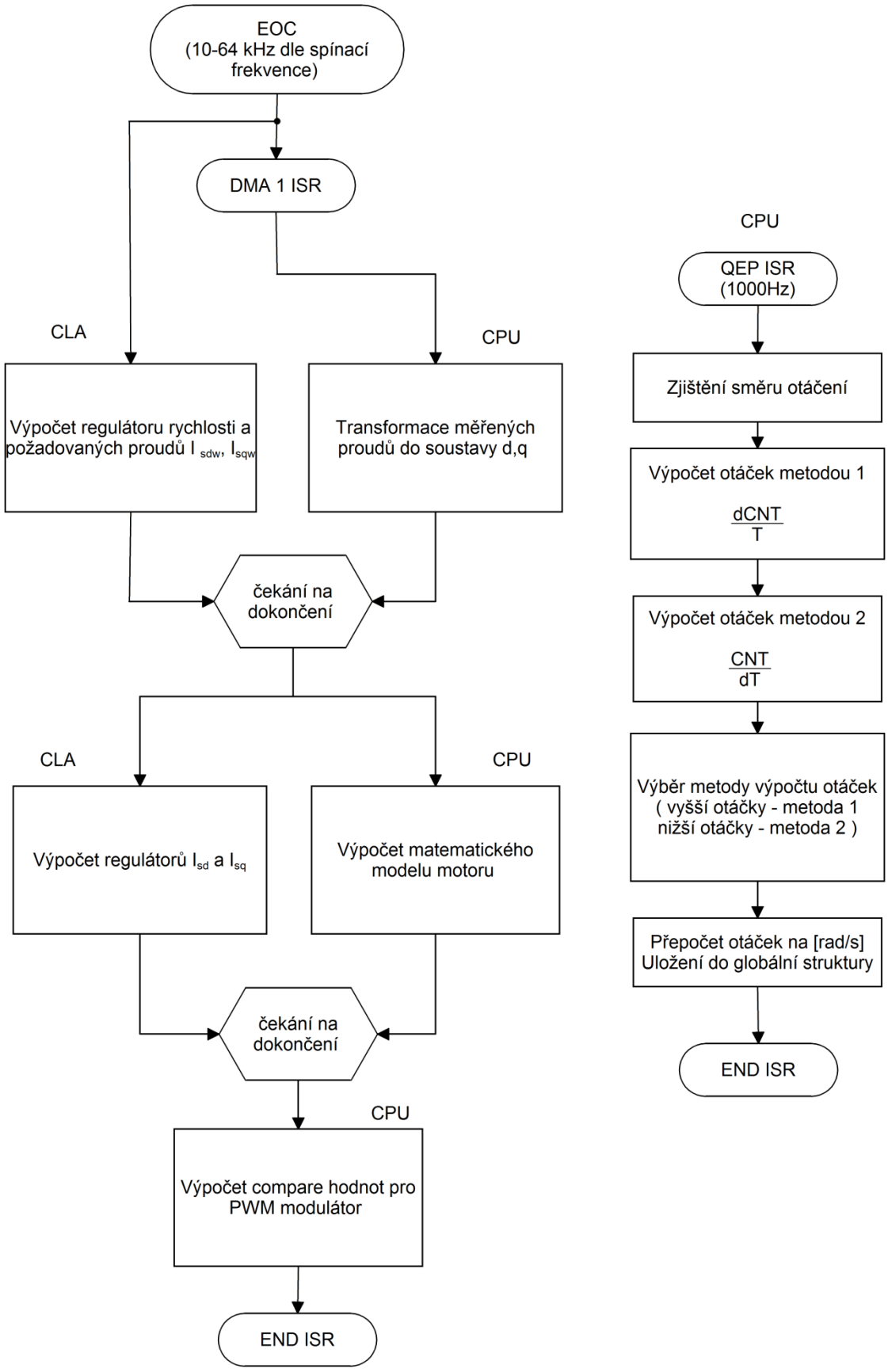
3.6 Průběh zpracování kódu

Postup zpracování programu mikroprocesorem je popsán vývojovými diagramy – jak ve verzi CPU (Obr. 16), tak ve verzi CPU+CLA (Obr. 17). Rozdělení výpočtů mezi CPU a CLA je graficky znázorněno na Obr. 18.

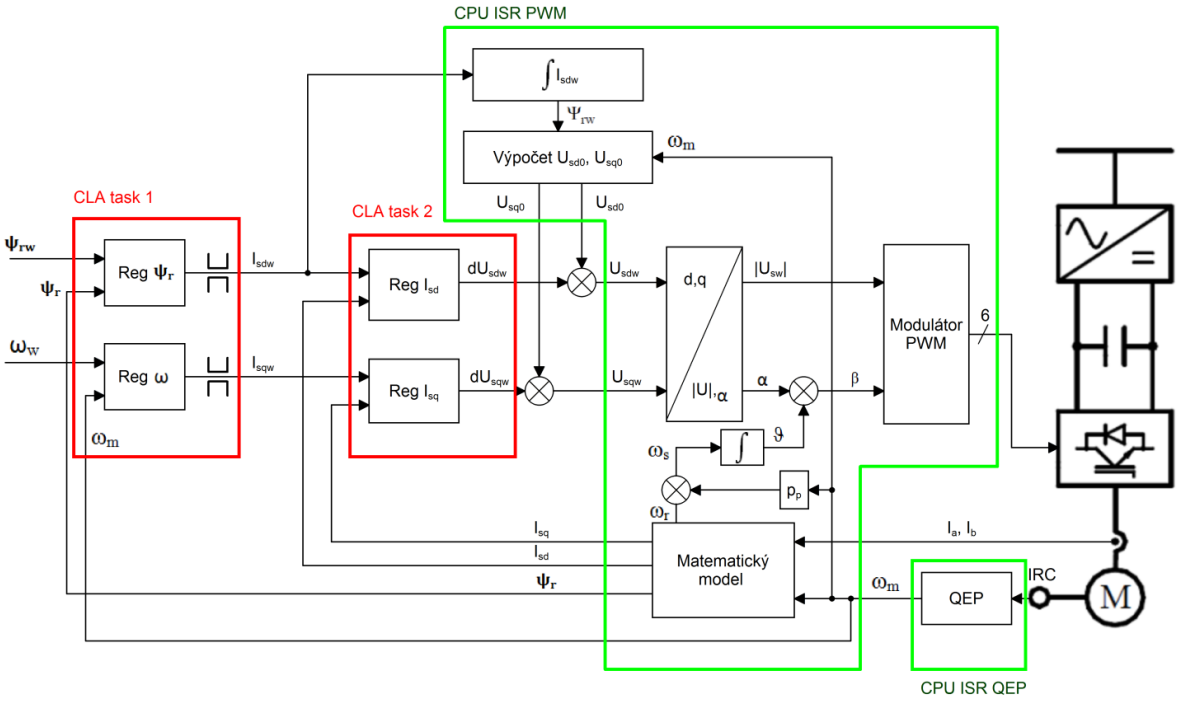
Každý cyklus výpočtu začíná vysláním signálu SOC pro spuštění A/D převodníku. Signál SOC je generován periferií ePWM. Taktovací frekvence periferie ePWM je ve výchozím stavu poloviční oproti taktovací frekvenci mikroprocesorového jádra. V případě použití externích A/D převodníků (např. u MLC Interface) je nutné povolit průchod signálu k externímu převodníku pomocí registru *SyncSocRegs*. Po dokončení převodu je vyslán signál EOC, kterým je spuštěn DMA přenos naměřených dat. Po ukončení DMA přenosu je vyvoláno přerušení, které spouští výpočty algoritmů regulace pohonu.



Obr. 16. *Vývojový diagram pro CPU*



Obr. 17. *Vývojový diagram pro CPU+CLA*

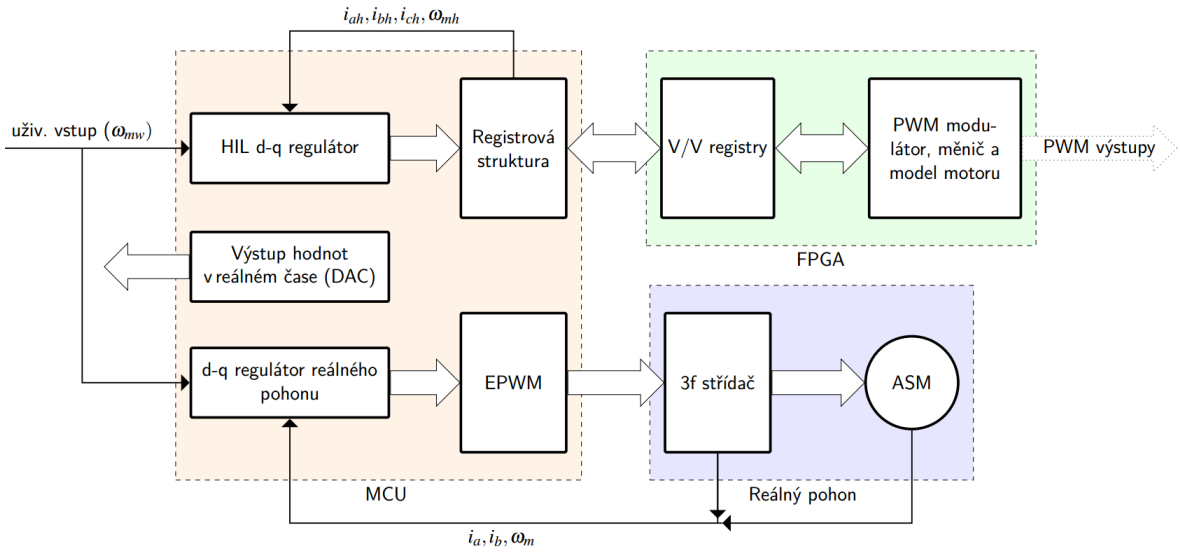


Obr. 18. *Rozdělení výpočtů mezi CPU a CLA*

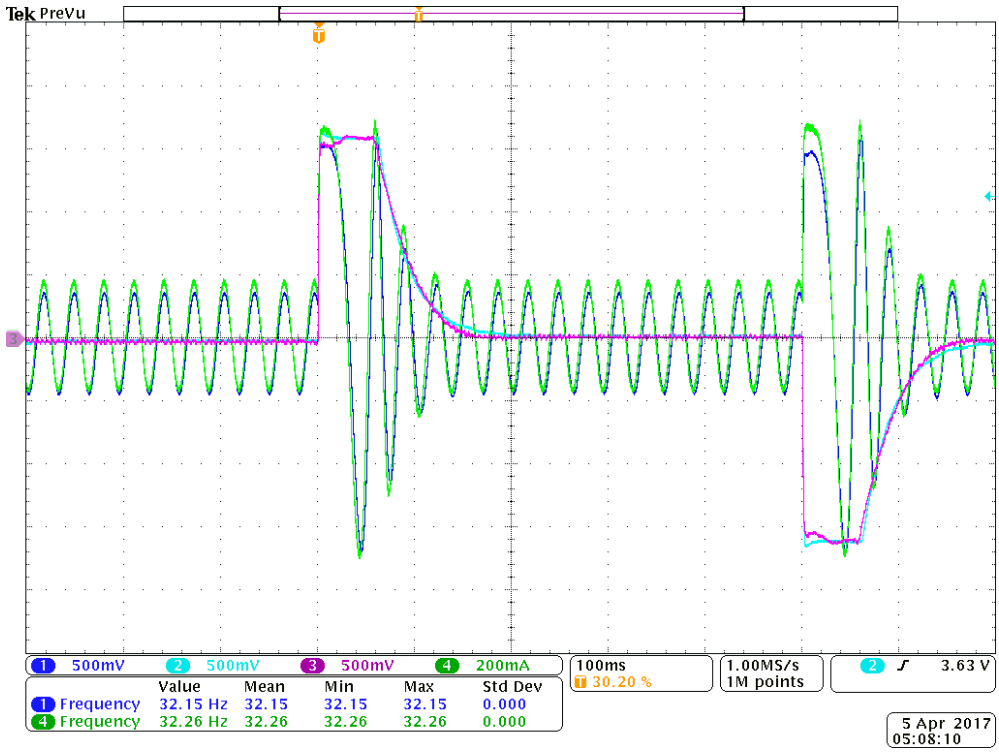
3.7 Model měniče a motoru v FPGA

Implementované vektorové řízení bylo kromě řízení asynchronního motoru použito i pro řízení modelu měniče a motoru v FPGA [10], které je součástí MLC Interface. Cílem této implementace bylo porovnání vlastností a výstupů modelu pohonu v FPGA ve srovnání s fyzickým asynchronním motorem a měničem. Pro tyto účely byl částečně upraven kód programu – signál SOC byl vysílán z FPGA a nebyl generován PWM periferií. Dále byla doplněna funkce modulátoru, která předává hodnotu poměrných sepnutí ve formátu Q0.15 pevné řádové čárky do FPGA.

Model i fyzický asynchronní motor je možné provozovat současně (Obr. 19). V takovém případě je vhodně využita akcelerace pomocí CLA, protože celý výpočet regulačních zásahů se musí postupně vypočítat dvakrát během jedné periody volání přerušení. Průběhy jednotlivých veličin, změřené na motoru, odpovídají vypočteným veličinám z modelu (Obr. 20).



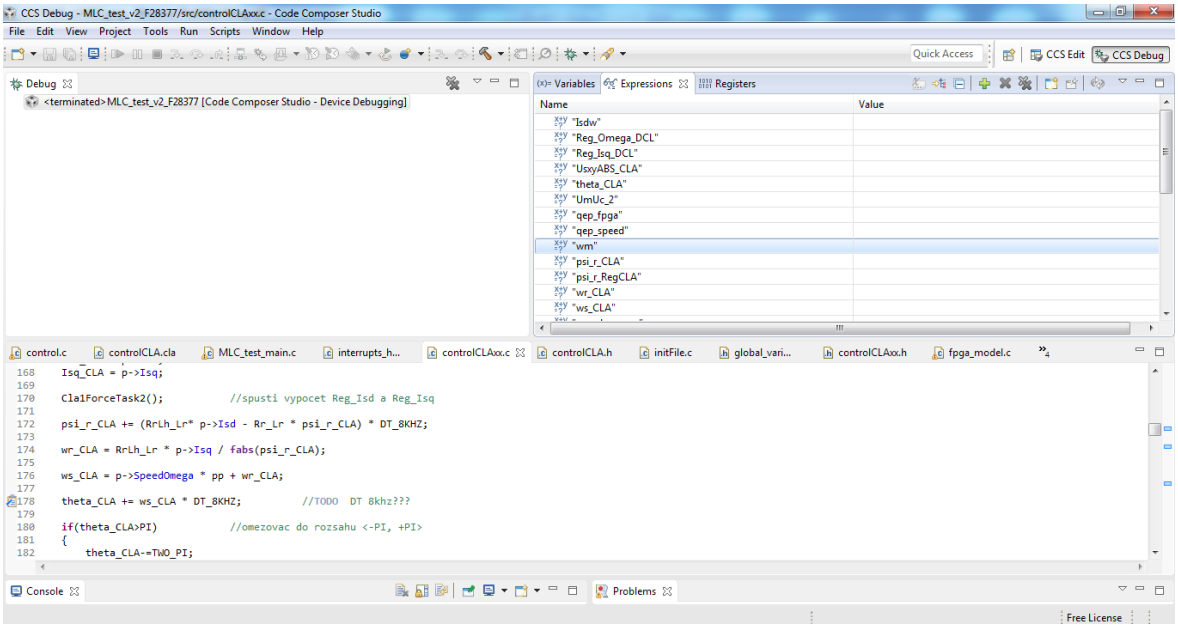
Obr. 19. *Blokové schéma řízení s modelem motoru i fyzickým motorem*



Obr. 20. *Reverzace motoru ±100 rad/s, CH1 (modrá):proud I_a vypočtený modelem (2 A/d) CH2 (světle modrá): proud I_{sq} vypočtený modelem (2 A/d), CH3 (fialová): proud I_{sq} motoru (2 A/d), CH4 (zelená): měřený proud motoru I_a (2 A/d)*

3.8 Ovládání a uživatelské rozhraní

Požadovaná rychlost otáčení je zadávána do proměnné w_m [rad/s] pomocí panelu *Expressions* (Obr. 21) v programu Code Composer Studio.



Obr. 21. *Zadávání požadovaných otáček*

Spínací frekvenci měniče je možné měnit pomocí konstanty *SPINACI_FREKVENCE* v souboru *global_variables.h*. Ve stejném souboru jsou uloženy i parametry motoru a maximální hodnoty fázových proudů pro nastavení nadproudové ochrany.

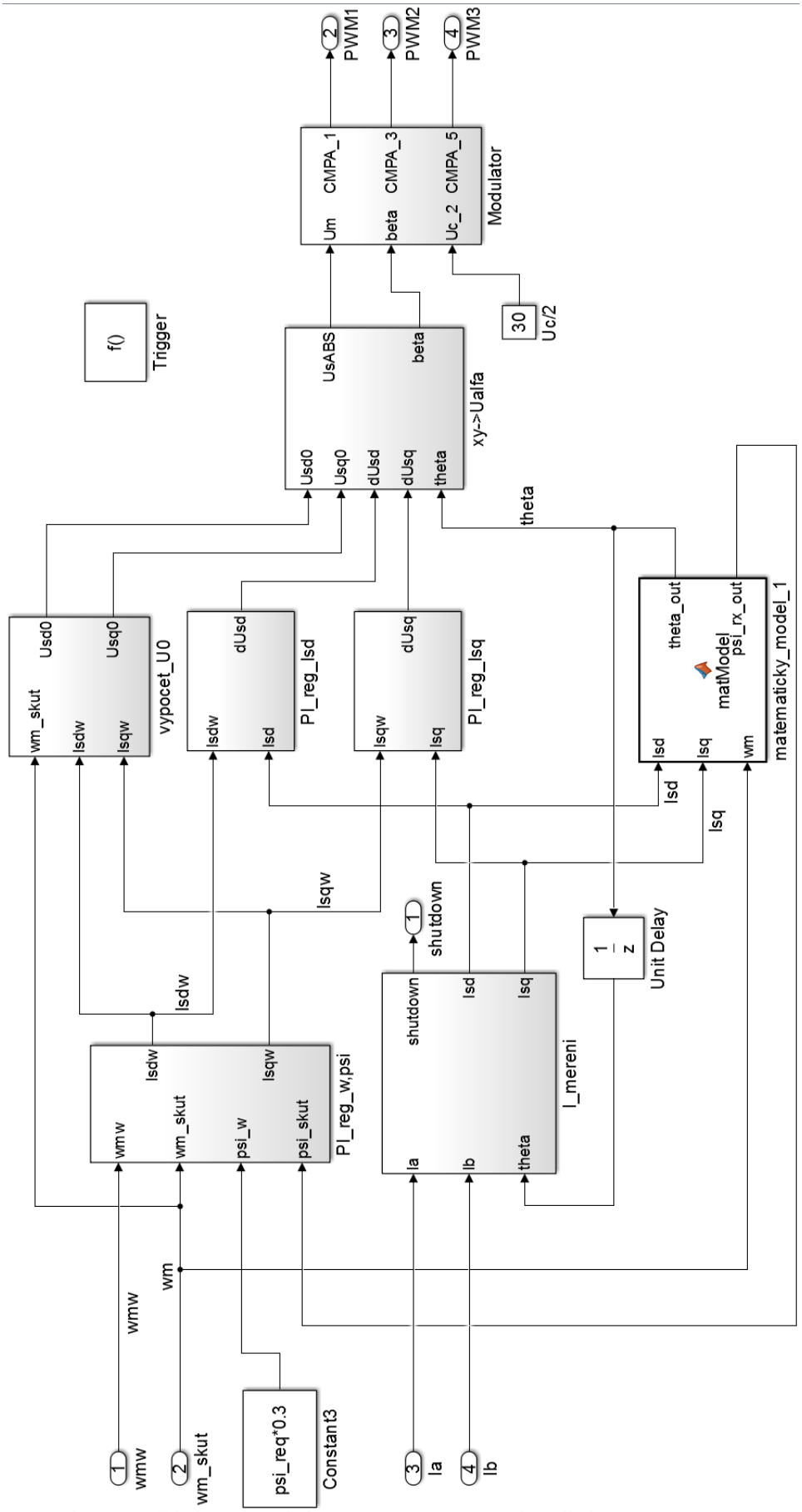
Při nulové požadované rychlosti je po dobrzdění nastaven požadavek na nulový proud I_{sq} a nenulový proud I_{sd}, aby byl motor stále nabuzený a nedocházelo ke kmitání kolem nulových otáček.

3.9 Generování kódu

Pro ověření správné funkčnosti a pro porovnání efektivity byl kromě ručně psaného kódu implementován i automaticky generovaný kód. Pomocí aplikace Embedded Coder, který je doplňkem programu Matlab/Simulink je možné generovat kód v jazyce C z modelů vytvořených v Simulinku. Je možné vygenerovat buď celý projekt s vlastní *main()* funkcí, nebo generovat jen jednotlivé funkce, které se následně vloží do existujícího, ručně psaného projektu.

Vzhledem k omezené podpoře současných verzí Matlabu pro mikroprocesor TMS320F28377S byla vygenerována pouze funkce pro zpracování vektorového řízení (Obr. 22), která byla vložena do stávajícího projektu. K dispozici nebyla dostupná podpora pro CLA, jsou využívány pouze CPU instrukce.

Vstupy do generované funkce jsou požadované otáčky (zadáno uživatelem), skutečné otáčky a okamžité hodnoty 2 fázových proudů (měřeno). Výstupem jsou 3 vypočtené hodnoty pro Compare registry jednotlivých PWM a signál *shutdown* pro vybavení nadproudové ochrany. Algoritmy matematického modelu, transformací, PI regulátorů i modulátoru jsou stejné jako v případě ručně psaného kódu.

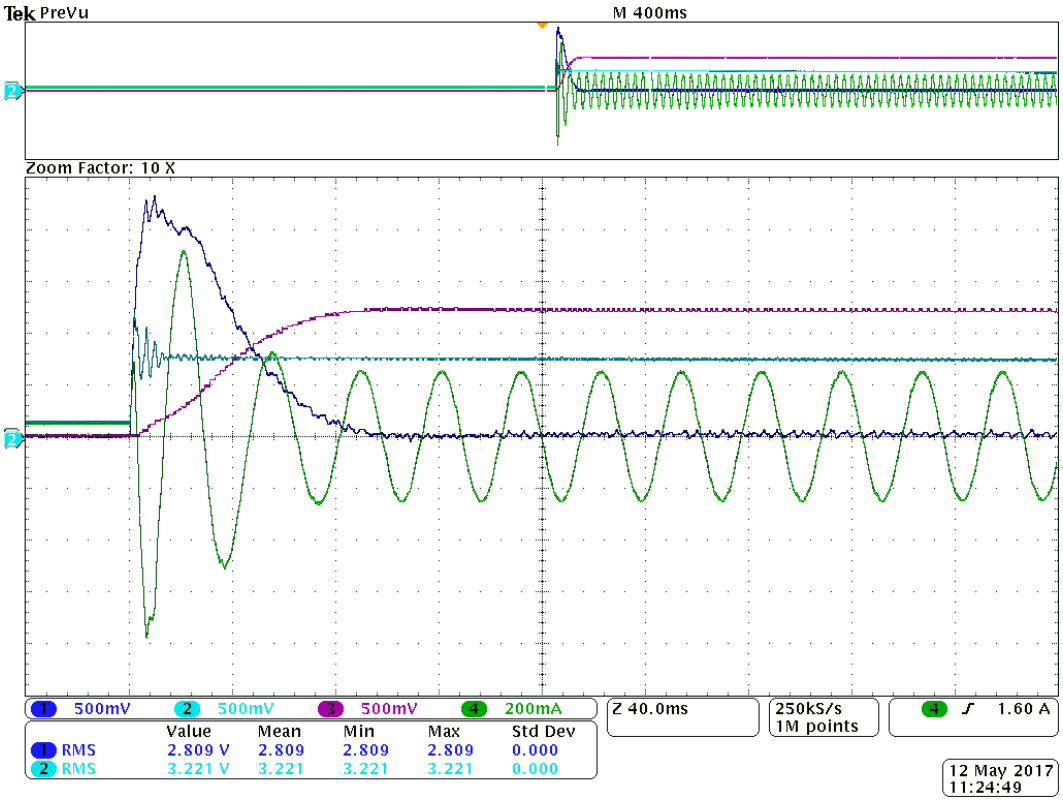


Obr. 22. *Blokové schéma v programu Simulink*

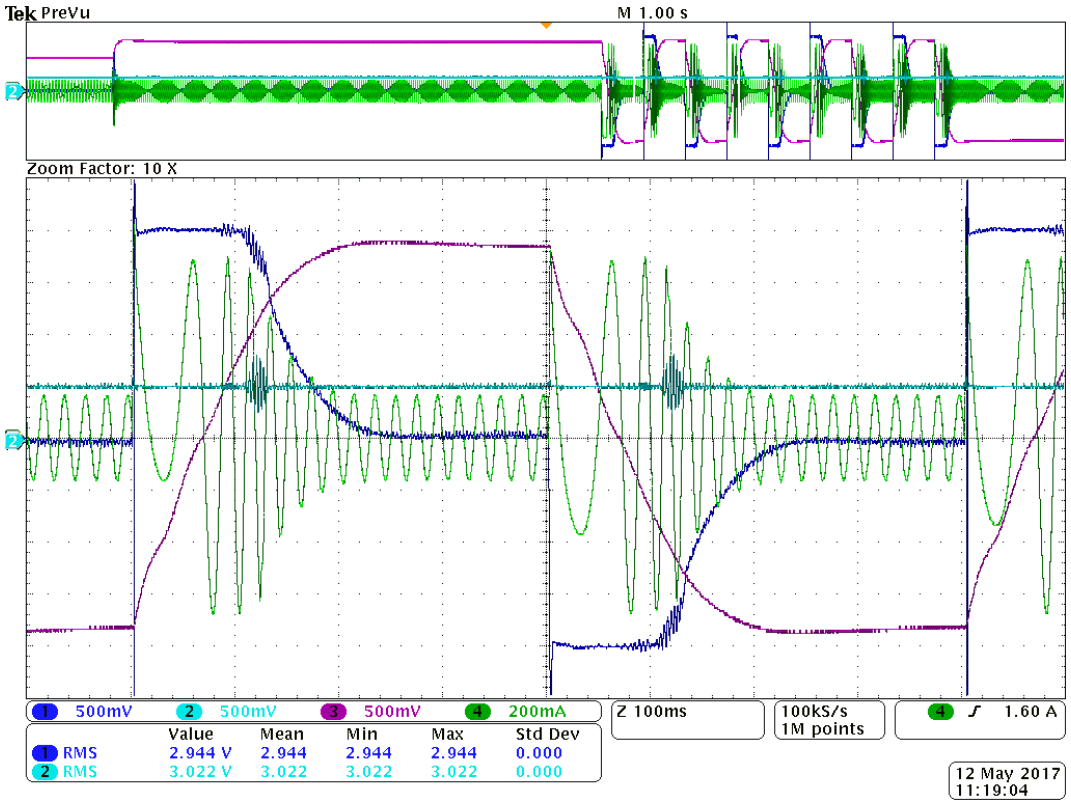
4 Měření

Motor byl testován bez zátěže při spínací frekvenci 32 kHz.

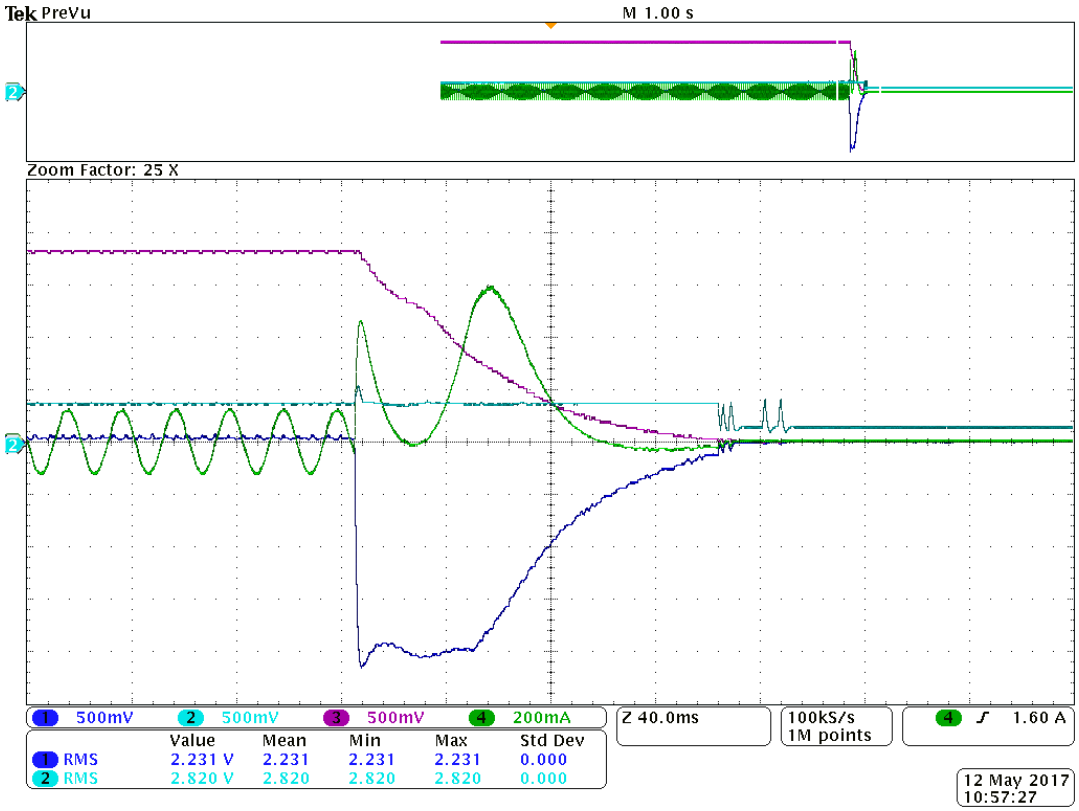
Měřené a přepočtené proudy I_{sd}, I_{sq} a otáčky motoru byly pomocí D/A převodníku zobrazeny na osciloskopu. Fázový proud byl měřen na fázi *U* pomocí proudové sondy. Zaznamenán je průběh rozběhu (Obr. 23), reverzace (Obr. 24) a brzdění (Obr. 25).



Obr. 23. *Rozběh motoru na 100 rad/s, CH1 (modrá):proud I_{sq} (1,6A/d) CH2 (světle modrá): proud I_{sd} (1,6 A/d), CH3 (fialová): mechanické otáčky motoru (41 rad/s /d), CH4 (zelená): fázový proud 1. fáze (2 A/d)*



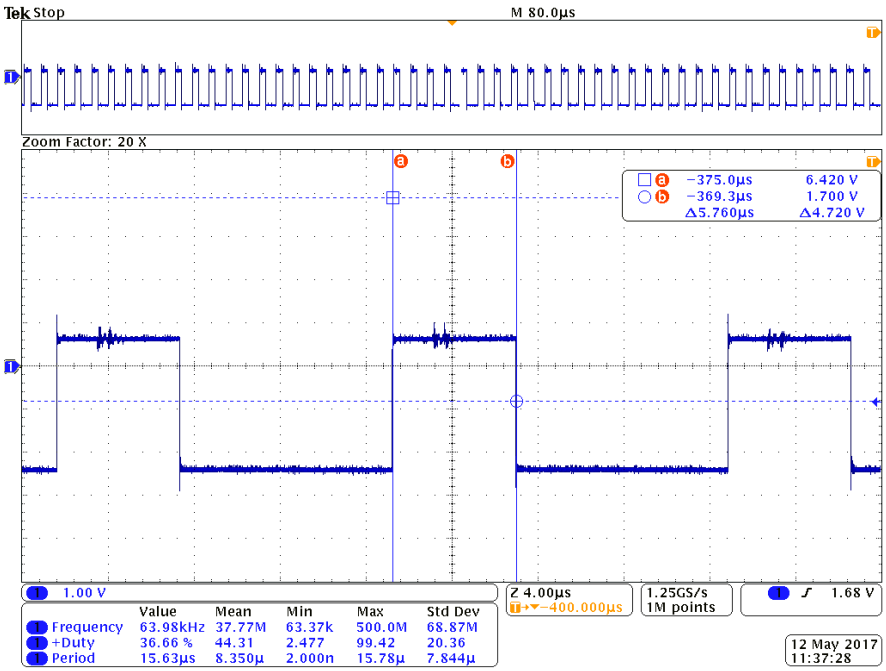
Obr. 24. *Reverzace motoru ± 150 rad/s, CH1 (modrá):proud I_{sq} (1,6A/d) CH2 (světle modrá): proud I_{sd} (1,6 A/d), CH3 (fialová): mechanické otáčky motoru (41 rad/s /d), CH4 (zelená): fázový proud 1. fáze (2 A/d)*



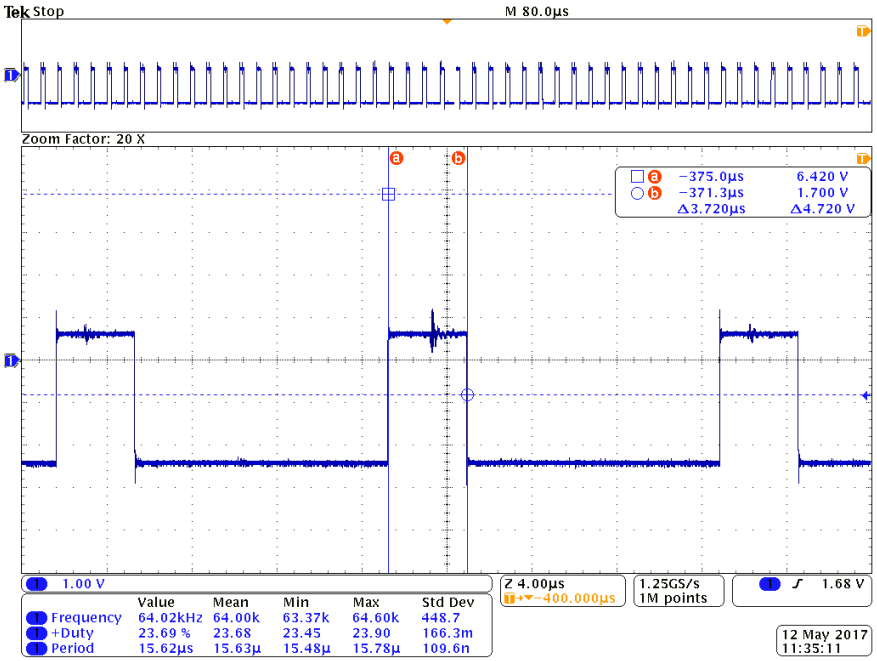
Obr. 25. *Brzdění motoru ze 150 rad/s, CH1 (modrá):proud I_{sq} (1,6 A/d) CH2 (světle modrá): proud I_{sd} (1,6 A/d), CH3 (fialová): mechanické otáčky motoru (41 rad/s /d), CH4 (zelená): fázový proud fáze U (2 A/d)*

Doba výpočtu programu byla změřena pomocí osciloskopu – na začátku výpočtu byl pomocí makra `MLC_WRITE(WRITE_DBGLEDS, 0x03);` nastaven GPIO výstup CPLD do logické 1 a po skončení výpočtu nastaven zpět do logické 0. Dle průběhu změřeného signálu lze určit dobu výpočtu a vytížení procesoru. Dále byla k měření použita funkce Clock v Code Composer Studio. Tato funkce měří počet taktů procesoru, které proběhly mezi 2 breakpointy. Při znalosti taktovací frekvence procesoru je opět možné určit dobu výpočtu.

Jádro mikroprocesoru i CLA pracují s taktovací frekvencí 200 MHz. Doba výpočtu zvoleného algoritmu vektorového řízení trvá 5,76 µs (1150 taktů) při výpočtu pouze pomocí CPU (Obr. 26), případně 3,72 µs (745 taktů) při využití CPU i CLA (Obr. 27). Při spínací frekvenci polovodičových součástek 32 kHz, což odpovídá volání regulačního algoritmu s dvojnásobnou frekvencí 64 kHz (perioda 15,6 µs) je využití samotného mikroprocesoru 36,6%, případně 23,7% při použití CPU i CLA.

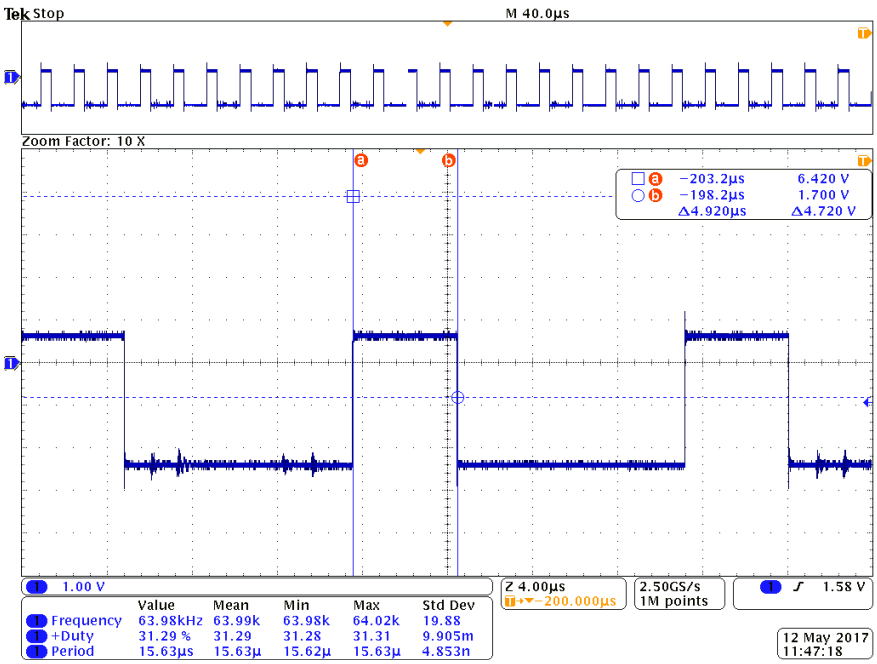


Obr. 26. *Využití CPU bez použití CLA*



Obr. 27. *Využití CPU v kombinaci s CLA*

Pro porovnání byla změřena i doba zpracování automaticky generovaného kódu (Obr. 28) – 4,92 µs (985 taktů). Generování kódu s využitím CLA nebylo dostupné, výpočet byl prováděn pouze na jádru mikroprocesoru, které bylo využito na 31,2%. Z důvodu lepší optimalizace automaticky generovaného kódu je výpočetní funkce algoritmů regulace v tomto případě efektivnější oproti ručně psanému kódu, který měl v rámci projektu vypnuté optimalizace.



Obr. 28. *Využití CPU – generovaný kód*

5 Závěr

Nový typ mikroprocesoru TMS320F28377S přináší oproti staršímu typu TMS320F28335 zvýšení výpočetního výkonu díky vyšší taktovací frekvenci a přítomnosti koprocesoru CLA. Vyšší výpočetní výkon lze využít ke zvýšení spínací frekvence nebo k využití složitějších algoritmů řízení, jako je například řízení bez čidla otáček nebo v dnešní době velmi moderní prediktivní řízení.

Pro zjištění přínosu koprocesoru CLA bylo implementováno vektorové řízení asynchronního motoru. Toto řízení má dobré dynamické vlastnosti. Byla porovnána doba výpočtu algoritmu bez využití CLA a s využitím CLA. Použití CLA přineslo urychlení o 35% (3,72 μs vs. 5,76 μs). Teoreticky je možné při zvoleném algoritmu vektorového řízení využít spínací frekvenci polovodičových prvků až 66 kHz při zhruba 50% využití mikroprocesoru za použití CLA.

Použití novější řady TMS320F2837x se jeví jako perspektivní pro široké použití v rámci platformy MLC Interface – jak v testované verzi s jedním jádrem a CLA (TMS320F28377S), tak ve variantě s dvojicí CPU + CLA (TMS320F28377D). Přechod na novější řadu prozatím brzdí omezená podpora automatického generování kódu v programu Matlab/Simulink, která je ale postupně rozšiřována v každé nové verzi.

Seznam literatury a informačních zdrojů

[1] Wikipedia: the free encyclopedia. *Hardware acceleration* [online]. Poslední změna 14.4.2017. [Cit. 20.4.2017]. Dostupné z: https://en.wikipedia.org/wiki/Hardware_acceleration

[2] Texas Instruments. *Accelerators: Enhancing the Capabilities of the C2000™ MCU Family*. [online]. Poslední změna 5.11.2016. [Cit. 14.3.2017]. Dostupné z: http://www.ti.com/lit/an/spry288a/spry288a.pdf

[3] IEEE Standard for Floating-Point Arithmetic, in *IEEE Std 754-2008* , vol., no., pp.1-70, Aug. 29 2008, doi: 10.1109/IEEESTD.2008.4610935 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4610935&isnumber=4610934

[4] PINKER, Jiří, POUPA, Martin. *Číslicové systémy a jazyk VHDL*, 1. vyd. Plzeň: BEN – technická literatura, 2006. 344 s. ISBN 80-7300-198-5.

[5] DUFOUR C., CENSE S. a BELANGER J., "*An FPGA HIL Reconfigurable Testing Platform for Vehicular Traction Systems*," 2014 IEEE Vehicle Power and Propulsion Conference (VPPC), Coimbra, 2014, pp. 1-4. doi:10.1109/VPPC.2014.7007012 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7007012&isnumber=7006878

[6] ZEMAN, Karel, PEROUTKA, Zdeněk a JANDA, Martin. *Automatická regulace pohonů s asynchronními motory*, 1. vyd. Plzeň: Západočeská univerzita, 2007. 204 s. ISBN 978-80-7043-350-8.

[7] Texas Instruments. *C2000™ Digital Controller Library - User's Guide*. [online]. Poslední změna 12.7.2015. [Cit. 20.3.2017]. Dostupné z: http://www.ti.com/lit/ug/sprui31/sprui31.pdf

[8] PEROUTKA, Zdeněk. *Mikroprocesorové řízení pohonů – KEV/MRP – Výběr z přednášek.* Plzeň: Západočeská univerzita, RICE. 2016.

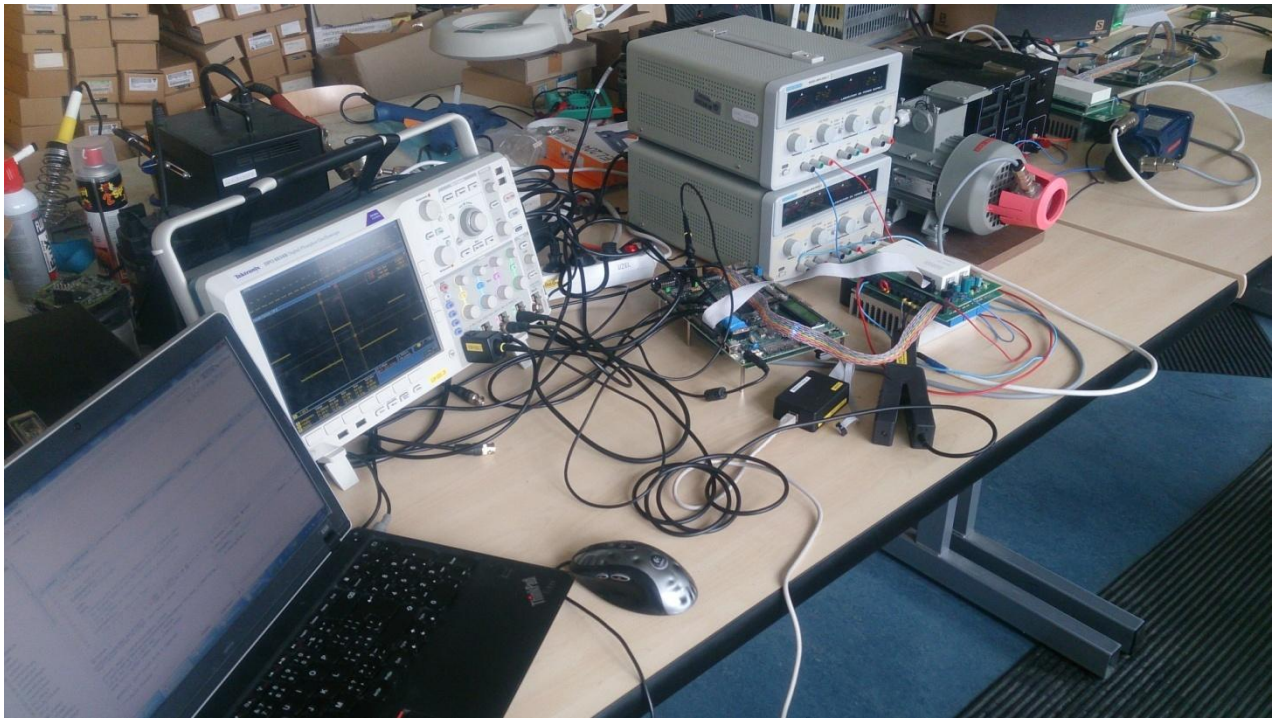
[9] Texas Instruments. *TMS320F2837xS Delfino Microcontrollers - Technical Reference Manual*. [online]. Poslední změna 19.4.2016. [Cit. 10.2.2017]. Dostupné z: http://www.ti.com/lit/ug/spruhx5d/spruhx5d.pdf

[10] KOŠAN, T., TALLA, J., GLAC, A. *Design and Verification of FPGA-based Real-time HIL Simulator of Induction Motor Drive*, Plzeň: Západočeská univerzita, RICE. 2017.

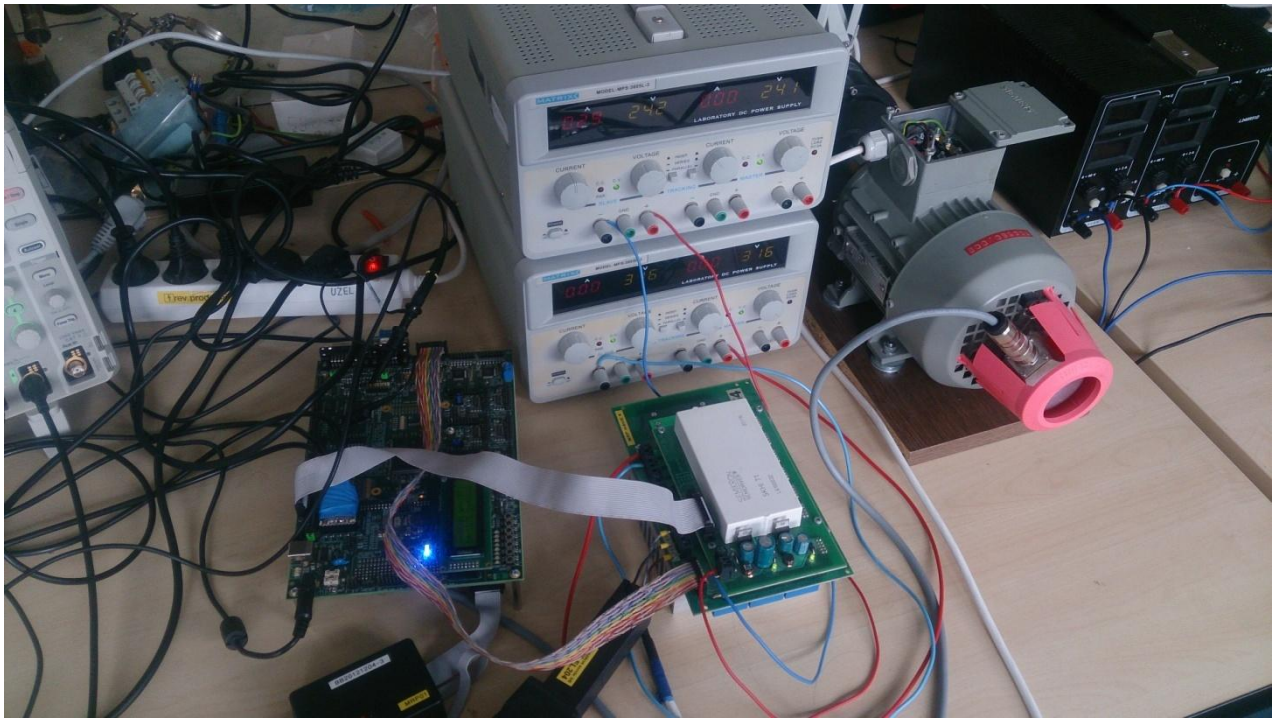
[11] KOŠAN, T.; JÁRA, M.; JANÍK, D., PEROUTKA, Z. *Complete development platform for multilevel converters and complex control algorithms*. In Mechatronics - Mechatronika (ME), 2014, 16th International Conference on. 2014, s. 152–157. Dostupný z WWW: http://dx.doi.org/10.1109/MECHATRONIKA.2014.7018251.

Přílohy

Příloha A – Fotografie zkušebního pracoviště



Obr. 1 *Zkušební pracoviště – celkový přehled*



Obr. 2 *Zkušební pracoviště – výkonová část*